

KASH

Reference Manual

KANT-Group
Technische Universität Berlin
Fachbereich 3 Mathematik
Straße des 17. Juni 136
10623 Berlin, Germany

February 22, 2004

Copyright © Prof. Dr. Michael E. Pohst, 1987–2003

TU Berlin

Fachbereich 3 Mathematik

Straße des 17. Juni 136

10623 Berlin, Germany

KASH can be copied and distributed freely for any non-commercial purpose.

If you copy KASH for somebody else, you may ask this person to refund your expenses. This should cover cost of media, copying and shipping. You are not allowed to ask for more than this. In any case you must give a copy of this copyright notice along with the program.

If you obtain KASH please send us a short notice to that effect, e.g., an e-mail message to the address *kant@math.tu-berlin.de* containing your full name and address. This allows us to keep track of the number of KASH users.

If you publish a mathematical result that was partly obtained using KASH, please cite

M. Daberkow, C. Fieker, J. Klüners, M. Pohst, K. Roegner
and K. Wildanger, *KANT V4*, in J. Symbolic Comp. **24** (1997), 267-283.

Also we would appreciate it if you could inform us about such a paper.

You are permitted to modify and redistribute KASH, but you are not allowed to restrict further redistribution. That is to say proprietary modifications will not be allowed. We want all versions of KASH to remain free. If you modify any part of KASH and redistribute it, you must supply a 'README' document. This should specify what modifications you made in which files. We do not want to take credit or be blamed for your modifications.

Of course we are interested in all of your modifications. In particular we would like to see bug-fixes, improvements and new functions. So again we would appreciate it if you would inform us about all modifications you make.

KASH is distributed by us without any warranty, to the extent permitted by applicable state law. We distribute KASH *as is* without warranty of any kind, either expressed or implied, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose.

The entire risk as to the quality and performance of the program is with you. Should KASH prove defective, you assume the cost of all necessary servicing, repair or correction. In no case unless required by applicable law will we, and/or any other party who may modify and redistribute KASH as permitted above, be liable to you for damages, including lost profits, lost monies or other special, incidental or consequential damages arising out of the use or inability to use KASH.

Acknowledgements

This program could not have been written without the support of Prof. J. Cannon, W. Bosma, S. Collins and A. Steele at the University of Sydney. Additionally, we would like to thank Prof. Dr. J. Neubüser at the RWTH Aachen, Germany for his permission to use and modify a large portion of the GAP source code. Special thanks also go to M. Schönert, a main contributor to the creation of GAP, for his kind support and aid. Finally, we would like to thank A. Weber at Cornell University, who developed the database for algebraic number fields and M. Klebel who did the (Ray) class fields for imaginary quadratic number fields.

- | | | |
|---------|-------------|---|
| KANT V4 | Copyright © | Prof. Dr. M. Pohst, 1987-2003
TU Berlin, Fachbereich 3 Mathematik
Straße des 17. Juni 136, 10623 Berlin, Germany
EMail: kant@math.tu-berlin.de |
| KASH | Copyright © | Prof. Dr. M. Pohst, 1994-2003
TU Berlin, Fachbereich 3 Mathematik
Straße des 17. Juni 136, 10623 Berlin, Germany
EMail: kant@math.tu-berlin.de |
| MAGMA | Copyright © | Prof. J. Cannon, 1995 -2003
Sydney, Australia
Email: john@maths.usyd.edu.au |
| GAP | Copyright © | Lehrstuhl D für Mathematik, 1992
RWTH Aachen
Templergraben 64, 52062 Aachen, Germany |

Short Contents

1	KASH functions	1
2	The Programming Language	893
3	Lists	913
4	Sets	937
5	Records	943
	Bibliography	959
	Index	963

Contents

1 KASH functions	1
Package: <i>AbelianGroup</i>	3
Package: <i>Alff</i>	59
Package: <i>Arc</i>	251
Package: <i>Drinf</i>	285
Package: <i>Ecc</i>	291
Package: <i>Elt</i>	303
Package: <i>FF</i>	355
Package: <i>Find</i>	385
Package: <i>Galois</i>	392
Package: <i>Ideal</i>	423
Package: <i>Im</i>	475
Package: <i>Infty</i>	483
Package: <i>Integer</i>	488
Package: <i>Is</i>	507
Package: <i>Lat</i>	537
Package: <i>List</i>	562
Package: <i>Mat</i>	566
Package: <i>Min</i>	600
Package: <i>Module</i>	604
Package: <i>Order</i>	636
Package: <i>Poly</i>	734
Package: <i>Pvm</i>	766
Package: <i>Qf</i>	797
Package: <i>Qp</i>	806
Package: <i>Random</i>	818

Package: <i>Ray</i>	826
Package: <i>Simplex</i>	858
Package: <i>Subfield</i>	866
Package: <i>Thue</i>	870
2 The Programming Language	893
2.1 Lexical Structure	894
2.2 Symbols	895
2.3 Whitespaces	895
2.4 Keywords	896
2.5 Identifiers	896
2.6 Expressions	897
2.7 Variables	897
2.8 Function Calls	898
2.9 Comparisons	899
2.10 Operations	900
2.11 Statements	901
2.12 Assignments	902
2.13 Procedure Calls	902
2.14 If	903
2.15 While	904
2.16 Repeat	904
2.17 For	905
2.18 Functions	906
2.19 Return	909
2.20 The Syntax in BNF	909
3 Lists	913
3.1 IsList	914
3.2 List	914
3.3 List Elements	915
3.4 Length	917
3.5 List Assignment	917
3.6 Add	919
3.7 Append	919

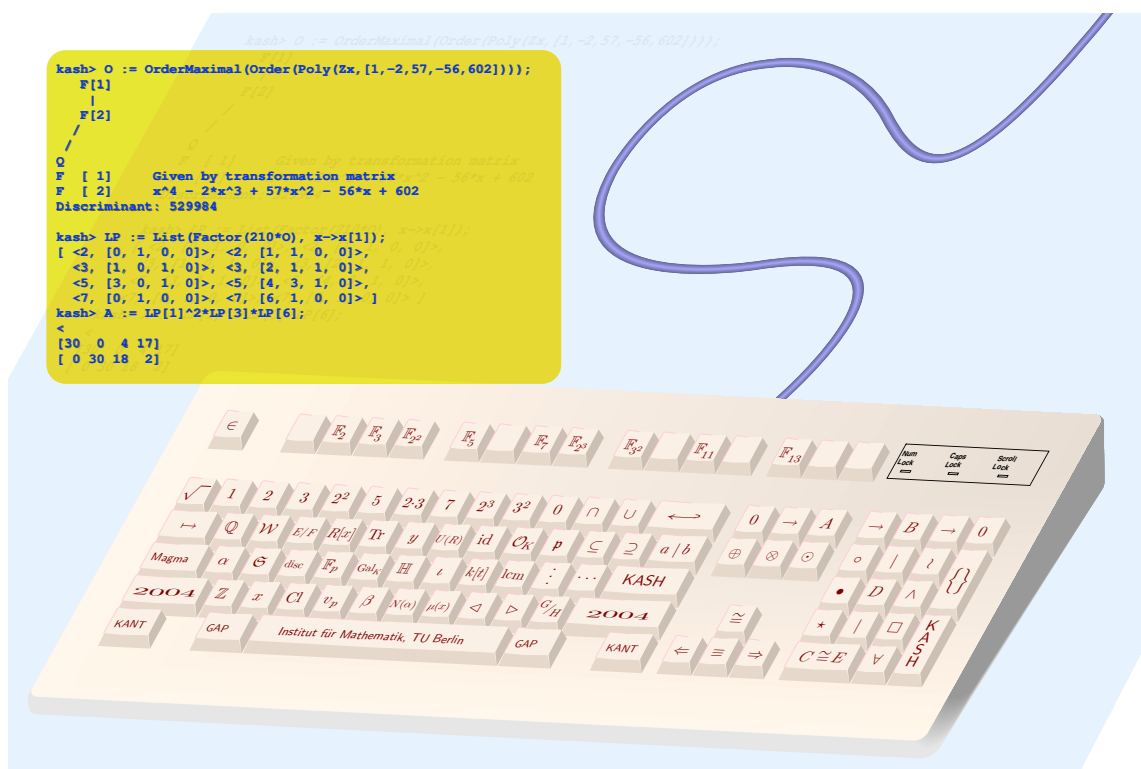
3.8 Identical Lists	920
3.9 IsIdentical	922
3.10 Enlarging Lists	922
3.11 Comparisons of Lists	923
3.12 Operations for Lists	924
3.13 In	925
3.14 Position	925
3.15 PositionSorted	926
3.16 PositionProperty	927
3.17 Concatenation	927
3.18 Flat	928
3.19 Reversed	928
3.20 Sublist	928
3.21 Cartesian	929
3.22 Number	929
3.23 Collected	930
3.24 Filtered	930
3.25 ForAll	931
3.26 ForAny	931
3.27 First	932
3.28 Sort	932
3.29 SortParallel	933
3.30 Sortex	933
3.31 Permuted	934
3.32 Product	934
3.33 Sum	934
3.34 Maximum	935
3.35 Minimum	935
3.36 Iterated	936
3.37 RandomList	936

4	Sets	937
4.1	IsSet	938
4.2	Set	938
4.3	SetIsEqual	939
4.4	SetAdd	939
4.5	SetRemove	939
4.6	SetUnite	940
4.7	SetIntersect	940
4.8	SetSubtract	941
4.9	Set Functions for Sets	941
4.10	More about Sets	942
5	Records	943
5.1	Accessing Record Elements	944
5.2	Record Assignment	944
5.3	Identical Records	945
5.4	Comparisons of Records	948
5.5	Operations for Records	950
5.6	In for Records	951
5.7	Printing of Records	952
5.8	IsRec	953
5.9	IsBound	953
5.10	Unbind	954
5.11	Copy	955
5.12	ShallowCopy	956
5.13	RecFields	957
	Bibliography	959
	Index	963

Chapter 1

KASH functions

This chapter contains an alphabetically sorted line-up of relevant functions available in KASH.



NAME	AbelianDualGroup
PURPOSE	Returns the dual group of a given finite abelian group.
SYNTAX	$d := \text{AbelianDualGroup}(g);$ groups g, d
DESCRIPTION	Returns the dual group d of a group g , i.e. the group consisting of the isomorphisms from the group g to \mathbb{C} . The isomorphisms are presented by rational numbers q corresponding to $e^{(i*2*\pi*q)}$.

EXAMPLE Compute the dual group and apply one homomorphism:

```
kash> g := AbelianGroupCreate([[0,1,2],[5,6,0],[0,4,5]]);
Group with relations:
[0 1 2]
[5 6 0]
[0 4 5]
kash> d := AbelianDualGroup(g);
Group with relations:
[15]
kash> eltd := AbelianGroupEltCreate(d, [1]);
[1]
kash> hom := AbelianGroupDiscreteExp(eltd);
HomMatrix =
[ -6   6]
[ 10 -10]
[ -5   5]
from Group with relations:
[0 1 2]
[5 6 0]
[0 4 5]
to Group with relations:
[3 0]
[0 5]

kash> eltg := AbelianGroupEltCreate(g, [1,0,0]);
[1 0 0]
kash> AbelianGroupDiscreteExp(hom*eltg);
1/5
```

It is allowed to apply an element of the group to an element of the corresponding dual group:

```
kash> eltd*eltg;  
1/5
```


NAME	AbelianDualHom
PURPOSE	Creates the dual homomorphism of a given homomorphism between two finite Abelian groups.
SYNTAX	<pre>dualhom := AbelianDualHom(hom);</pre> <p>homomorphism hom homomorphism from g_1 to g_2</p>
DESCRIPTION	Creates the dual homomorphism from the group g_2^* to group g_1^* of a homomorphism $\text{hom}: g_1 \rightarrow g_2$. The dual homomorphism is defined by: $\text{hom}^*: g_2^* \rightarrow g_1^*$ and $\text{hom}^*: \chi \rightarrow \chi * \text{hom}$.
SEE ALSO	AbelianGroupHomCreate , AbelianHomGroup ,
EXAMPLE	

```
kash> g1 := AbelianGroupCreate([[0,1,2],[5,6,0],[0,4,5]]);;
kash> g2 := AbelianGroupCreate([[0,2],[3,0]]);;
kash> mat := Mat(Z, [[-24,24],[20,-20],[-10,10]]);;
kash> hom := AbelianGroupHomCreate(g1, g2, mat, true);
HomMatrix =
[-24  24]
[ 20 -20]
[-10  10]
from Group with relations:
[0 1 2]
[5 6 0]
[0 4 5]
to Group with relations:
[0 2]
[3 0]

kash> dualhom:=AbelianDualHom(hom);
HomMatrix =
[5]
from Group with relations:
[6]
to Group with relations:
[15]
```

NAME	AbelianFieldToRCF
PURPOSE	Computes the data necessary to get an abelian field as a Ray Class Field.
SYNTAX	<pre>rc := AbelianFieldToRCF(o [, I]);</pre> <p>list rc [ideal, inf, relations]</p> <p>order o of an abelian field with known automorphisms</p> <p>ideal I of the coef. ring of o. Must be a multiple of the conductor.</p>

DESCRIPTION

EXAMPLE Suppose we know that the field generated by a root of $x^4 - 4x^2 + 1$ is abelian, and we'd like to work with this field:

```
kash> o := OrderMaximal(x^4-4*x^2+1);
Generating polynomial: x^4 - 4*x^2 + 1
Discriminant: 2304

kash> OrderAutomorphisms(o);
[ [0, 1, 0, 0], [0, -4, 0, 1], [0, 4, 0, -1], [0, -1, 0, 0] ]
kash> rc1 := AbelianFieldToRCF(o, Disc(o));
Quotient of RayClassGroupToAbelianGroup(<2304>, [ ])
Group with relations:
[ 2 0]
[ 0 2]
[ 2 0]
[ 0 192]
kash> f := RayConductor(rc1);
[ <24>, [ ] ]
kash> rc2 := AbelianFieldToRCF(o, f[1]);
Quotient of RayClassGroupToAbelianGroup(<24>, [ ])
Group with relations:
[2 0]
[0 2]
[2 0]
[0 2]
kash> rcf := RayClassField(rc2);
[ x^2 - 2, x^2 - 3 ]
```

NAME	AbelianFixPointGroup
PURPOSE	Computes the fix point group of endomorphisms.
SYNTAX	<pre>f := AbelianFixPointGroup(hom); f := AbelianFixPointGroup(L);</pre> <pre>group f homomorphism hom list L list of homomorphisms</pre>
DESCRIPTION	Computes the group consisting of elements that are (pointwise) fix under each given endomorphism.

EXAMPLE Compute the fix point group and apply an endomorphism:

```
kash> g := AbelianGroupCreate([[0,0],[0,60]]);
kash> mat := Mat(Z, [[1,1],[0,1]]);
kash> hom := AbelianGroupHomCreate(g, g, mat, true);
HomMatrix =
[1 1]
[0 1]
from Group with relations:
[ 0  0]
[ 0 60]
to Group with relations:
[ 0  0]
[ 0 60]

kash> f := AbelianFixPointGroup(hom);
Group with relations:
[60  0]
[ 0  0]
kash> elt := AbelianGroupEltCreate(f, [1,2]);
[1 2]
kash> elt := AbelianGroupEltMove(elt, g);
[120  1]
kash> hom*elt;
[120  1]
```

NAME	AbelianGroupBasis
PURPOSE	Returns a basis of a finite abelian group.
SYNTAX	$L := \text{AbelianGroupBasis}(g \text{ [, exp] });$ <div style="display: flex; justify-content: space-between;"> <div> group list boolean </div> <div> g L exp </div> <div> list of basis elements of g </div> </div>
DESCRIPTION	Returns a list of basis elements of the group g . If the second (optional) parameter <i>exp</i> is true then the discrete exp is applied.
SEE ALSO	AbelianGroupMinNumberGenerators ,

EXAMPLE Compute the basis of a ray class group:

```
kash> 0 := OrderMaximal(Z, 2, 10);
Generating polynomial: x^2 - 10
Discriminant: 40

kash> g := RayClassGroupToAbelianGroup(3*0, [1, 2]);
RayClassGroupToAbelianGroup(<3>, [ 1, 2 ])
Group with relations:
[4 0]
[0 2]
kash> l := AbelianGroupBasis(g);
[ [0 1], [1 0] ]
kash> l := AbelianGroupBasis(g, true);
[ <2, [0, 1]>, <[1, 3]> ]
```

NAME	AbelianGroupCanonicalQuotient
PURPOSE	Returns the canonical quotient group of a homomorphism.
SYNTAX	<pre>q := AbelianGroupCanonicalQuotient(hom);</pre> <div style="margin-left: 100px;"> $\begin{array}{ccc} \text{group} & & q \\ \text{homomorphism} & \text{hom} & \end{array}$ </div>
DESCRIPTION	Let hom be a homomorphism from a group g to a group h . Then <code>AbelianGroupCanonicalQuotient</code> returns the quotient group q of g and the kernel of hom . All homomorphisms and isomorphisms are installed. This is useful if you frequently consider elements of g , h and the kernel and the image of hom .
SEE ALSO	AbelianGroupHomImage , AbelianGroupHomKernel , AbelianQuotientGroup ,
EXAMPLE	Compute the canonical quotient of a homomorphism:

```

kash> g := AbelianGroupCreate([[1,2,3],[2,4,0],[2,0,0]]);
kash> h := AbelianGroupCreate([[1,0,0],[0,4,0],[1,1,1]]);
kash> mat := Mat(Z, [[0,0,0],[6,0,-6],[4,0,4]]);
kash> hom := AbelianGroupHomCreate(g, h, mat, true);
HomMatrix =
[ 0  0  0]
[ 6  0 -6]
[ 4  0  4]
from Group with relations:
[1 2 3]
[2 4 0]
[2 0 0]
to Group with relations:
[1 0 0]
[0 4 0]
[1 1 1]

kash> q := AbelianGroupCanonicalQuotient(hom);
Group with relations:
[1 0 0]
[0 2 0]
[0 0 1]
[1 2 3]

```

[2 4 0]

[2 0 0]

kash> AbelianGroupSmithCreate(q);

Group with relations:

[2]

NAME	AbelianGroupCyclicFactors
PURPOSE	Returns cyclic factors and their orders from the Smith normal form.
SYNTAX	$L := \text{AbelianGroupCyclicFactors}(g);$ $\text{group } g$ $\text{list } L$ list of cyclic factors of g
DESCRIPTION	Returns a list of lists of cyclic factors of the group and the orders of the cyclic factors obtained from the Smith normal form of the relations for g .
SEE ALSO	AbelianGroupSmithCreate ,
EXAMPLE	Compute the cyclic factors of a ray class group:

```
kash> O := OrderMaximal(Z, 2, 10);
```

```
Generating polynomial: x^2 - 10
```

```
Discriminant: 40
```

```
kash> g := RayClassGroupToAbelianGroup(3*O, [1, 2]);
```

```
RayClassGroupToAbelianGroup(<3>, [ 1, 2 ])
```

```
Group with relations:
```

```
[4 0]
```

```
[0 2]
```

```
kash> L := AbelianGroupCyclicFactors(g);
```

```
[ [ <2, [0, 1]>, 2 ], [ <[1, 3]>, 4 ] ]
```


NAME	AbelianGroupDirectProduct
PURPOSE	Returns the direct product of two finite abelian groups.
SYNTAX	<pre>d := AbelianGroupDirectProduct(g1, g2); d := AbelianGroupDirectProduct(L);</pre> <p> groups g1, g2, d list L list of groups </p>
DESCRIPTION	<p>In the first case the direct product d of the group $g1$ and the group $g2$ is computed. In the second case the direct product d of all groups in the list is computed.</p> <p>Instead of using <code>AbelianGroupDirectProduct</code> you may use the <code>*</code> operator (see example).</p>

EXAMPLE Compute the direct product using `AbelianGroupDirectProduct`:

```
kash> g1 := AbelianGroupCreate([[5]]);
Group with relations:
[5]
kash> g2 := AbelianGroupCreate([[7,7],[6,8]]);
Group with relations:
[7 7]
[6 8]
kash> d := AbelianGroupDirectProduct([g1, g2]);
Group with relations:
[5 0 0]
[0 7 7]
[0 6 8]
```

Compute the direct product using `*`:

```
kash> d := g1*g2;
Group with relations:
[5 0 0]
[0 7 7]
[0 6 8]
```

Compute the power $\bigoplus_{i=1}^4 g_1$:

```
kash> p := g1^4;  
Group with relations:  
[5 0 0 0]  
[0 5 0 0]  
[0 0 5 0]  
[0 0 0 5]
```

NAME	AbelianGroupDiscreteExp
PURPOSE	Returns the object corresponding to a representation in a group.
SYNTAX	<pre>b := AbelianGroupDiscreteExp(a);</pre> <p> group g group element a representation of <i>b</i> in the abstract group <i>g</i> object or boolean b </p>
DESCRIPTION	<p>Returns the concrete object <i>b</i> corresponding to the abstract representation of <i>a</i> in <i>g</i>. For example, if the abstract group <i>g</i> represents a concrete class group of a number field, then <code>AbelianGroupDiscreteExp(a)</code> calculates an Ideal <i>b</i>, that is representing a class of ideals in the class group. If <i>g</i> is the dual group of a given group, $a \in g$, then $b := \text{AbelianGroupDiscreteExp}(a)$ is the homomorphism represented by <i>a</i>. In this form it can directly be used to evaluate element of the original group, to which <i>g</i> is dual.</p> <p>If no element is found, <i>false</i> is returned.</p>
SEE ALSO	AbelianGroupDiscreteLog ,

EXAMPLE

```
kash> O := OrderMaximal(x^2-2*x-5);
Generating polynomial: x^2 - 2*x - 5
Discriminant: 24

kash> g := RayResidueRingToAbelianGroup(27*O, [2]);
Group with relations:
[2 0 0 0 0]
[0 3 0 0 0]
[0 0 9 0 0]
[0 0 0 9 0]
[0 0 0 0 2]
kash> a := AbelianGroupEltCreate(g, [1, 2, 3, 0, 0]);
[1 2 3 0 0]
kash> b := AbelianGroupDiscreteExp(a);
[-337479530000, -232975502000]
kash> a := AbelianGroupDiscreteLog(g, b);
[1 2 3 0 0]
```

NAME	AbelianGroupDiscreteLog						
PURPOSE	Returns the representation of an object in a group.						
SYNTAX	<pre>a := AbelianGroupDiscreteLog(g, b);</pre> <table> <tr> <td>group</td><td>g</td></tr> <tr> <td>group element</td><td>a representation of b in the abstract group g</td></tr> <tr> <td>object</td><td>b</td></tr> </table>	group	g	group element	a representation of b in the abstract group g	object	b
group	g						
group element	a representation of b in the abstract group g						
object	b						
DESCRIPTION	Returns the abstract representation a (exponent vector corresponding to the generators of g) of an element b in the group g . For example if g represents a class group of a number field, then a given ideal is represented by <code>AbelianGroupDiscreteLog(g,a)</code> as an abstract element of this group.						
SEE ALSO	AbelianGroupDiscreteExp ,						
EXAMPLE							

```
kash> 0 := OrderMaximal(x^2-10);
Generating polynomial: x^2 - 10
Discriminant: 40
```

```
kash> g := RayResidueRingToAbelianGroup(27*0, [2]);
Group with relations:
[2 0 0 0 0]
[0 9 0 0 0]
[0 0 2 0 0]
[0 0 0 9 0]
[0 0 0 0 2]
kash> a := AbelianGroupEltCreate(g, [1, 2, 3, 0, 0]);
[1 2 3 0 0]
kash> b := AbelianGroupDiscreteExp(a);
[22271000, -10959000]
kash> a := AbelianGroupDiscreteLog(g, b);
[1 2 1 0 0]
```

```
kash> g := RayClassGroupToAbelianGroup(1*0);  
RayClassGroupToAbelianGroup(<1>, [  ])  
Group with relations:  
[2]  
kash> I := Ideal(2,Elt(0,[0,1]));  
<2, [0, 1]>  
kash> b := AbelianGroupDiscreteLog(g,I);  
[1]
```

NAME	AbelianGroupEltCreate						
PURPOSE	Creates an element of a group.						
SYNTAX	<pre>elt := AbelianGroupEltCreate(g, vector);</pre> <table> <tr> <td>group element</td><td>elt</td></tr> <tr> <td>group</td><td>g</td></tr> <tr> <td>list or matrix with one row of integers</td><td>vector</td></tr> </table>	group element	elt	group	g	list or matrix with one row of integers	vector
group element	elt						
group	g						
list or matrix with one row of integers	vector						
DESCRIPTION	<p>Assume that g is generated by g_1, \dots, g_r. Then vector contains $v_1, \dots, v_r \in \mathbb{Z}$ and the element elt $\in g$ is generated as $\prod_i g_i^{v_i}$. Only the exponent vector is stored.</p>						
SEE ALSO	AbelianGroupEltReduce ,						

EXAMPLE Create an element in an abstract group:

```
kash> g := AbelianGroupCreate([[0,1,2],[5,6,0],[0,4,5]]);
Group with relations:
[0 1 2]
[5 6 0]
[0 4 5]
kash> elt := AbelianGroupEltCreate(g, [3,2,1]);
[3 2 1]
```

NAME	AbelianGroupEltMove
PURPOSE	Moves an element from a subgroup into a supergroup.
SYNTAX	<pre>elt2 := AbelianGroupEltMove(elt1, g);</pre> <p> group g group element elt1 group element elt2 element of the group <i>g</i> </p>
DESCRIPTION	Returns the representation <code>elt2</code> of <code>elt1</code> in the group <i>g</i> .

EXAMPLE Move an element from a subgroup *s* of *g* into *g*:

```
kash> g := AbelianGroupCreate([[0,1,2],[5,6,0],[0,4,5]]);
Group with relations:
[0 1 2]
[5 6 0]
[0 4 5]
kash> s := AbelianSubGroup(g, [3,3,3]);
Group with relations:
[5]
kash> elt1 := AbelianGroupEltCreate(s, [2]);
[2]
kash> elt2 := AbelianGroupEltMove(elt1, g);
[6 6 6]
```

NAME	AbelianGroupEltOrder
PURPOSE	Returns the order of an element.
SYNTAX	<pre>a := AbelianGroupEltOrder(elt);</pre> <div> <div>group element</div> <div>integer</div> <div>elt</div> <div>a</div> </div>
DESCRIPTION	Returns the order of the element <code>elt</code> .
SEE ALSO	AbelianGroupOrder , AbelianGroupEltCreate ,
EXAMPLE	

```
kash> g := AbelianGroupCreate([[0,1,2],[5,6,0],[0,4,5]]);
Group with relations:
[0 1 2]
[5 6 0]
[0 4 5]
kash> elt := AbelianGroupEltCreate(g, [1,0,0]);
[1 0 0]
kash> a := AbelianGroupEltOrder(elt);
5
```


NAME AbelianGroupEltRandom

PURPOSE Generates a random element of an abelian group.

SYNTAX `e := AbelianGroupEltRandom(G);`

AbelianGroupElt e

AbelianGroup G

DESCRIPTION

EXAMPLE We'll produce some representatives for random rays modulo 9 in $\mathbb{Q}(\sqrt{10})$:

```
kash> o := OrderMaximal(Z, 2, 10);
Generating polynomial: x^2 - 10
Discriminant: 40

kash> G := RayClassGroupToAbelianGroup(9*o);
RayClassGroupToAbelianGroup(<9>, [ ])
Group with relations:
[2 0]
[0 3]
kash> for i in [1..10] do
> a := AbelianGroupEltRandom(G);
> Print(a, "\t", AbelianGroupDiscreteExp(a), "\n");
> od;
[0 0] <1>
[1 2] <[-1096, -612], [-3060, -548]>
[0 1] <[-2, -3]>
[1 0] <[-4, -6], [-30, -2]>
[1 1] <[188, 24], [120, 94]>
[1 0] <[-4, -6], [-30, -2]>
[0 0] <1>
[0 0] <1>
[0 1] <[-2, -3]>
[1 1] <[188, 24], [120, 94]>
```

NAME	AbelianGroupEltReduce
PURPOSE	Returns a reduced representation of a group element.
SYNTAX	<pre>elt2 := AbelianGroupEltReduce(elt1 [, positive]);</pre> <p> group element elt1 group element elt2 reduced representation boolean positive </p>
DESCRIPTION	Returns the reduced representation elt2 of a group element (exponent vector) elt1 . If the second (optional) parameter is true then all entries of elt2 are non negative. Only in this case it is sure that a good reduction is performed.
EXAMPLE	

```

kash> g := AbelianGroupCreate(Mat(Z, [[2,1], [1,3]]));
Group with relations:
[2 1]
[1 3]
kash> elt1 := AbelianGroupEltCreate(g, [-100, -217]);
[-100 -217]
kash> elt2 := AbelianGroupEltReduce(elt1);
[-100 -217]
kash> AbelianGroupSmithCreate(g);
Group with relations:
[5]
kash> elt2 := AbelianGroupEltReduce(elt1);
[-1 0]
kash> elt2 := AbelianGroupEltReduce(elt1, true);
[4 0]

```

NAME	AbelianGroupEnumInit
PURPOSE	Generates a environment for the enumeration of all elements of a finite abelian group G .
SYNTAX	<pre>s := AbelianGroupEnumInit(G [, "gen"]);</pre> <pre>record s AbelianGroup G arbitray gen if present, only a set of generators will be enumerated.</pre>
DESCRIPTION	
SEE ALSO	AbelianGroupEnumNext ,
EXAMPLE	See <code>AbelianGroupEnumNext</code> for an example.

NAME	AbelianGroupEnumNext		
PURPOSE	Computes “the next” element of an abelian group.		
SYNTAX	flag := AbelianGroupEnumNext(s);		
	boolean	flag	true iff there is a valid element in s
	record	s	generated by AbelianGroupEnumInit
	AbelianGroupElt	s.elc	

DESCRIPTION

SEE ALSO [AbelianGroupEnumInit](#),

EXAMPLE We’ll produce an ideal in every ray modulo 9 in $\mathbb{Q}(\sqrt{10})$:

```
kash> o := OrderMaximal(Z, 2, 10);
Generating polynomial: x^2 - 10
Discriminant: 40

kash> G := RayClassGroupToAbelianGroup(9*o);
RayClassGroupToAbelianGroup(<9>, [ ])
Group with relations:
[2 0]
[0 3]
kash> s := AbelianGroupEnumInit(G);
Record of type AbelianGroupEnum

kash> while AbelianGroupEnumNext(s) do
> Print(s.elc, "\t", AbelianGroupDiscreteExp(s.elc), "\n");
> od;
[0 0] <1>
[1 1] <[188, 24], [120, 94]>
[0 2] <[94, 12]>
[1 0] <[-4, -6], [-30, -2]>
[0 1] <[-2, -3]>
[1 2] <[-1096, -612], [-3060, -548]>
```

NAME	AbelianGroupEqual
PURPOSE	Tests if two groups are equal.
SYNTAX	<pre>test := AbelianGroupEqual(g1, g2); groups g1, g2 boolean test</pre>
DESCRIPTION	Returns true iff the groups g1 and g2 are contained in each other. Otherwise false is returned.
EXAMPLE	

```
kash> g1 := AbelianGroupCreate([[0,1,2],[5,6,0],[0,4,5]]);
Group with relations:
[0 1 2]
[5 6 0]
[0 4 5]
kash> g2 := AbelianSubGroup(g1, [1,1,1]);
Group with relations:
[15]
kash> test := AbelianGroupEqual(g1, g2);
true
```

Note:

```
kash> g1 = g2;
false
```

NAME	AbelianGroupExponent
PURPOSE	Returns the exponent of a group.
SYNTAX	$a := \text{AbelianGroupExponent}(g);$ group g integer a
DESCRIPTION	Returns the exponent a of the group g .
SEE ALSO	AbelianGroupOrder , AbelianGroupEltOrder ,
EXAMPLE	

```
kash> g := AbelianGroupCreate(Mat(Z, [[2,0], [0,4]]));
Group with relations:
[2 0]
[0 4]
kash> a := AbelianGroupExponent(g);
4
```

NAME	AbelianGroupGenerators
PURPOSE	Returns generators of a group.
SYNTAX	<pre>L := AbelianGroupGenerators(g [, exp]);</pre> <p> group g list L list of generators of g boolean exp </p>
DESCRIPTION	Returns a list of generators of the group g . If the second (optional) parameter exp is <code>true</code> then the discrete exp is applied.
SEE ALSO	AbelianGroupNumberGenerators , AbelianGroupBasis ,
EXAMPLE	

```
kash> O := OrderMaximal(Z, 2, 10);
```

```
Generating polynomial: x^2 - 10
```

```
Discriminant: 40
```

```
kash> g := RayClassGroupToAbelianGroup(3*O, [1, 2]);
```

```
RayClassGroupToAbelianGroup(<3>, [ 1, 2 ])
```

```
Group with relations:
```

```
[4 0]
```

```
[0 2]
```

```
kash> l := AbelianGroupGenerators(g);
```

```
[ [1 0], [0 1] ]
```

```
kash> l := AbelianGroupGenerators(g, true);
```

```
[ <2, [0, 1]>, <[1, 3]> ]
```

NAME	AbelianGroupHomCreate								
PURPOSE	Creates the homomorphism corresponding to a matrix.								
SYNTAX	<pre>hom := AbelianGroupHomCreate(g1, g2, mat [, check]); hom := AbelianGroupHomCreate(g1, g2, mat, [matinv]);</pre> <table> <tr> <td>groups</td><td>g1, g2</td></tr> <tr> <td>homomorphism</td><td>hom homomorphism from $g1$ to $g2$</td></tr> <tr> <td>boolean</td><td>check</td></tr> <tr> <td>matrix</td><td>matinv</td></tr> </table>	groups	g1, g2	homomorphism	hom homomorphism from $g1$ to $g2$	boolean	check	matrix	matinv
groups	g1, g2								
homomorphism	hom homomorphism from $g1$ to $g2$								
boolean	check								
matrix	matinv								
DESCRIPTION	Creates the homomorphism from group $g1$ to group $g2$ corresponding to the matrix mat . The columns of mat are the images of the generators of $g1$. If the fourth (optional) parameter $check$ is true then it is checked whether the object hom is a homomorphism, i.e. mat has the correct number of rows and columns. If the fourth parameter is a matrix then it has to be the inverse of mat .								
SEE ALSO	AbelianHomGroup ,								
EXAMPLE									

```
kash> g1 := AbelianGroupCreate([[0,1,2],[5,6,0],[0,4,5]]);;
kash> g2 := AbelianGroupCreate([[0,2],[3,0]]);;
kash> mat := Mat(Z, [[-24,24],[20,-20],[-10,10]]);;
kash> hom := AbelianGroupHomCreate(g1, g2, mat);
HomMatrix =
[-24  24]
[ 20 -20]
[-10  10]
from Group with relations:
[0 1 2]
[5 6 0]
[0 4 5]
to Group with relations:
[0 2]
[3 0]
```

If you feel insecure, you can use the fourth parameter *check* to test, whether the matrix corresponds to a homomorphism:


```
kash> mat := Mat(Z, [[3,0],[3,-2],[-1,2]]);;  
kash> hom := AbelianGroupHomCreate(g1, g2, mat, true);  
false
```

NAME	AbelianGroupHomImage
PURPOSE	Returns the image of a group homomorphism.
SYNTAX	<pre>g := AbelianGroupHomImage(hom [, generators]);</pre> <p> group g homomorphism hom boolean generators </p>
DESCRIPTION	Returns the image g of the homomorphism hom . g is a group. If the second (optional) parameter <code>generators</code> is <code>true</code> then the generators of the image g are generated, too.
SEE ALSO	AbelianGroupKernel , AbelianQuotientGroup , AbelianGroupCanonicalQuotient ,

EXAMPLE

```

kash> g1 := AbelianGroupCreate([[1,2,3],[2,4,0],[2,0,0]]);
kash> g2 := AbelianGroupCreate([[1,0,0],[0,4,0],[1,1,1]]);
kash> mat := Mat(Z, [[0,0,0],[6,0,-6],[4,0,4]]);
kash> hom := AbelianGroupHomCreate(g1, g2, mat, true);
HomMatrix =
[ 0  0  0]
[ 6  0 -6]
[ 4  0  4]
from Group with relations:
[1 2 3]
[2 4 0]
[2 0 0]
to Group with relations:
[1 0 0]
[0 4 0]
[1 1 1]

kash> g := AbelianGroupHomImage(hom);
Group with relations:
[1 0 0]
[0 2 0]
[0 0 1]
kash> AbelianGroupSmithCreate(g);
Group with relations:
[2]

```

NAME	AbelianGroupHomKernel						
PURPOSE	Returns the kernel of a group homomorphism.						
SYNTAX	<pre>g := AbelianGroupHomKernel(hom [,generators]);</pre> <table> <tr> <td>group</td><td>g</td></tr> <tr> <td>homomorphism</td><td>hom</td></tr> <tr> <td>boolean</td><td>generators</td></tr> </table>	group	g	homomorphism	hom	boolean	generators
group	g						
homomorphism	hom						
boolean	generators						
DESCRIPTION	Returns the kernel g of the homomorphism <code>hom</code> . g is a group. If the second (optional) parameter <code>generators</code> is <code>true</code> then the generators of the kernel g are generated, too.						
SEE ALSO	AbelianGroupHomImage , AbelianQuotientGroup , AbelianGroupCanonicalQuotient ,						
EXAMPLE	<pre>kash> g1 := AbelianGroupCreate([[1,2,3],[2,4,0],[2,0,0]]);; kash> g2 := AbelianGroupCreate([[1,0,0],[0,4,0],[1,1,1]]);; kash> mat := Mat(Z, [[0,0,0],[6,0,-6],[4,0,4]]);; kash> hom := AbelianGroupHomCreate(g1, g2, mat, true); HomMatrix = [0 0 0] [6 0 -6] [4 0 4] from Group with relations: [1 2 3] [2 4 0] [2 0 0] to Group with relations: [1 0 0] [0 4 0] [1 1 1] kash> g := AbelianGroupHomKernel(hom); Group with relations: [1 1 3] [0 2 0] [0 0 6] kash> AbelianGroupSmithCreate(g);</pre>						

Group with relations:

[2 0]

[0 6]

NAME	AbelianGroupIndex
PURPOSE	Returns the index of a subgroup in a group.
SYNTAX	<pre>a := AbelianGroupIndex(g, s);</pre> <p> group g group s subgroup of g integer a </p>
DESCRIPTION	Returns the index a of s in g . s has to be a subgroup of g .
SEE ALSO	AbelianSubGroup , AbelianGroupOrder ,
EXAMPLE	

```
kash> g := AbelianGroupCreate([[0]]);
Group with relations:
[0]
kash> bas := AbelianGroupBasis(g);
[ [1] ]
kash> s := AbelianSubGroup(g, 2*bas[1]);
Group with relations:
[0]
kash> a := AbelianGroupIndex(g, s);
2
```

NAME	AbelianGroupIntersect
PURPOSE	Returns the intersection of two groups.
SYNTAX	$g := \text{AbelianGroupIntersect}(g1, g2);$ groups $g, g1, g2$
DESCRIPTION	Returns the intersection g of the groups $g1, g2$. g is a group.
SEE ALSO	AbelianGroupUnite ,
EXAMPLE	

```

kash> g0 := AbelianGroupCreate([[0,1,2],[5,6,0],[0,4,5]]);
Group with relations:
[0 1 2]
[5 6 0]
[0 4 5]
kash> g1 := AbelianSubGroup(g0, [1,0,0]);
Group with relations:
[5]
kash> g2 := AbelianSubGroup(g0, [[0,1,0],[0,0,5]]);
Group with relations:
[1 1]
[0 3]
kash> g := AbelianGroupIntersect(g1, g2);
Group with relations:
[1]

```

NAME	AbelianGroupIsAut
PURPOSE	Tests whether a function is a group automorphism.
SYNTAX	<pre>L := AbelianGroupIsAut(hom [, check]);</pre> <p> homomorphism hom boolean check list L </p>
DESCRIPTION	Tests whether the function <code>hom</code> is an automorphism. <code>AbelianGroupIsAut</code> returns the list <code>[true, hom^{-1}]</code> iff <code>hom</code> is an automorphism, and the list <code>[false]</code> otherwise. If the second (optional) parameter <code>check</code> is <code>false</code> then there is no check, whether <code>hom</code> is an endomorphism.

EXAMPLE

```
kash> g := AbelianGroupCreate(Mat(Z, [[2,2], [1,3]]));;
kash> mat := Mat(Z, [[0,2], [0,2]]);;
kash> hom := AbelianGroupHomCreate(g, g, mat, true);;
kash> AbelianGroupIsAut(hom);
[ false ]
```

Further information:

```
kash> AbelianGroupPrintLevel := 1;;
kash> AbelianGroupIsAut(hom);
homomorphism is not surjective
[ false ]
kash> AbelianGroupPrintLevel := 0;;
```

Compute the inverse homomorphism:

```
kash> mat := Mat(Z, [[-1,2], [3,2]]);;
kash> hom := AbelianGroupHomCreate(g, g, mat, true);;
kash> hominv := hom^-1;
HomMatrix =
[ 0 1/5]
```

[0 1/5]

from Group with relations:

[2 2]

[1 3]

to Group with relations:

[2 2]

[1 3]

NAME	AbelianGroupIsSub
PURPOSE	Tests whether s is a subgroup of g .
SYNTAX	<pre>test := AbelianGroupIsSub(s, g);</pre> <p>groups g, s boolean $test$</p>
DESCRIPTION	Returns true iff the group s is a subgroup of g and false otherwise.
SEE ALSO	AbelianSubGroup ,
EXAMPLE	

```
kash> g := AbelianGroupCreate([[0,1,2],[5,6,0],[0,4,5]]);
Group with relations:
[0 1 2]
[5 6 0]
[0 4 5]
kash> s1 := AbelianSubGroup(g, [1,0,0]);
Group with relations:
[5]
kash> s2 := AbelianSubGroup(g, [[0,1,0],[0,0,5]]);
Group with relations:
[1 1]
[0 3]
kash> AbelianGroupIsSub(s1, g);
true
kash> AbelianGroupIsSub(s1, s2);
false
```

NAME	AbelianGroupMinNumberGenerators
PURPOSE	Returns the minimal number of generators of a group.
SYNTAX	<pre>a := AbelianGroupMinNumberGenerators(g);</pre> <p> group g integer a </p>
DESCRIPTION	Returns the minimal number a of generators of the group g .
SEE ALSO	AbelianGroupBasis ,
EXAMPLE	

```
kash> g := AbelianGroupCreate(Mat(Z, [[2,1], [1,3]]));
Group with relations:
[2 1]
[1 3]
kash> a := AbelianGroupMinNumberGenerators(g);
1
```


tensor product $t = g1 \otimes g2$:

```
kash> t := 1[3];  
Group with relations:  
[2]
```

create the multilinear mapping:

```
kash> f := AbelianGroupMultiHomCreate(d, h, t, [[0,1]]);  
MultiHom [ 1, 1 ] from Group with relations:  
[2 0]  
[0 4] to Group with relations:  
[8 0]  
[0 2]  
  
kash> elt := AbelianGroupEltCreate(d, [1,1]);  
[1 1]  
kash> AbelianGroupEltReduce(f*elt, true);  
[0 1]  
kash> f := AbelianGroupMultiHomCreate(d, h, t, [[1,0]]);  
false  
kash> AbelianGroupPrintLevel := 1;  
1  
kash> f := AbelianGroupMultiHomCreate(d, h, t, [[1,0]]);  
false  
kash> AbelianGroupPrintLevel := 0;  
0
```

NAME	AbelianGroupName
PURPOSE	Gives a name to a group.
SYNTAX	<p>AbelianGroupName(<i>g</i>, <i>s</i>);</p> <p>group <i>g</i></p> <p>string <i>s</i> name of <i>g</i></p>
DESCRIPTION	Gives the name of a string <i>s</i> to a group <i>g</i> . The name of <i>g</i> is shown everytime when <i>g</i> is printed.
SEE ALSO	AbelianGroupPrintLevel ,
EXAMPLE	

```

kash> g1 := AbelianGroupCreate(Mat(Z, [[2,1], [1,3]]));
Group with relations:
[2 1]
[1 3]
kash> g2 := AbelianGroupCreate(Mat(Z, [[2,1], [1,3]]));
Group with relations:
[2 1]
[1 3]
kash> AbelianGroupName(g1, "first group");
kash> AbelianGroupName(g2, "second group");
kash> g1;
first group
Group with relations:
[2 1]
[1 3]
kash> g2;
second group
Group with relations:
[2 1]
[1 3]

```

NAME	AbelianGroupNumberGenerators
PURPOSE	Returns the number of generators.
SYNTAX	<pre>a := AbelianGroupNumberGenerators(g);</pre> <p>group g integer a</p>
DESCRIPTION	Returns the number of generators of a group g in the current representation.
SEE ALSO	AbelianGroupGenerators ,
EXAMPLE	

```
kash> g := AbelianGroupCreate(Mat(Z, [[2,1], [1,3]]));
Group with relations:
[2 1]
[1 3]
kash> a := AbelianGroupNumberGenerators(g);
2
```

NAME	AbelianGroupOrder
PURPOSE	Returns the order of a group.
SYNTAX	<pre>a := AbelianGroupOrder(g); group g integer a</pre>
DESCRIPTION	Returns the order a of the group g .
SEE ALSO	AbelianGroupExponent ,
EXAMPLE	
<pre>kash> g := AbelianGroupCreate(Mat(Z, [[2,1], [1,3]])); Group with relations: [2 1] [1 3] kash> a := AbelianGroupOrder(g); 5</pre>	

NAME	AbelianGroupPrintLevel
PURPOSE	Sets printlevel.
SYNTAX	AbelianGroupPrintLevel := a; integer a
DESCRIPTION	Sets printlevel for printing groups: <ul style="list-style-type: none"> • AbelianGroupPrintLevel := 0; : few information (standard) • AbelianGroupPrintLevel := 1; : more information • AbelianGroupPrintLevel := 2; : all information

EXAMPLE Few information (standard):

```
kash> g := AbelianGroupCreate(Mat(Z, [[4,2], [8,2]]));
Group with relations:
[4 2]
[8 2]
kash> AbelianGroupSmithCreate(g);
kash> h := AbelianGroupHomCreate(g, g, Mat(Z, [[0,1],[1,0]]), true);
false
```

More information:

```
kash> AbelianGroupPrintLevel := 1;;
kash> g;
Group with relations:
[4 2]
[8 2]
exponent   : 4
order      : 8
has SNF    : Group with relations:
[2 0]
[0 4]
kash> h := AbelianGroupHomCreate(g, g, Mat(Z, [[0,1],[1,0]]), true);
false
```


NAME	AbelianGroupSmithCreate
PURPOSE	Returns the smith normal form of a group.
SYNTAX	<pre>s := AbelianGroupSmithCreate(g [, generators]);</pre> <p> group g, s boolean generators </p>
DESCRIPTION	Returns the smith normal form s of g : the relation matrix of s is in smith normal form. The groups s and g are isomorphic. If the second (optional) parameter generators is true then the new generators are computed, too. This is useful if you often use the discrete exp.

EXAMPLE

```
kash> g := AbelianGroupCreate([[15,5,-5,10],[0,0,0,50],[6,6,6,6]]);
Group with relations:
[15  5 -5 10]
[ 0  0  0 50]
[ 6  6  6  6]
kash> s := AbelianGroupSmithCreate(g);
Group with relations:
[ 10   0   0]
[  0 300   0]
[  0   0   0]
```

NAME	AbelianGroupTensorProduct
PURPOSE	Returns the tensor product of groups.
SYNTAX	$t := \text{AbelianGroupTensorProduct}(L);$ $\text{group } t$ $\text{list } L \text{ list of groups}$
DESCRIPTION	Returns the tensor product t of a list L of groups.
EXAMPLE	

```

kash> g1 := AbelianGroupCreate(MatDiag(Z, [1,2,3]));
Group with relations:
[1 0 0]
[0 2 0]
[0 0 3]
kash> g2 := AbelianGroupCreate(MatDiag(Z, [4,5]));
Group with relations:
[4 0]
[0 5]
kash> g3 := AbelianGroupCreate(MatDiag(Z, [4, 5, 0]));
Group with relations:
[4 0 0]
[0 5 0]
[0 0 0]

```

Compute the tensor product $t = g1 \otimes g2 \otimes g3$:

```

kash> t := AbelianGroupTensorProduct([g1, g2, g3]);
Group with relations:
[2 0]
[0 2]

```

NAME	AbelianGroupUnite
PURPOSE	Returns the smallest group containing two given groups.
SYNTAX	$g := \text{AbelianGroupUnite}(g1, g2);$ groups $g, g1, g2$
DESCRIPTION	Returns the group g generated by the elements of the groups $g1, g2$.
SEE ALSO	AbelianGroupIntersect ,
EXAMPLE	

```

kash> g0 := AbelianGroupCreate([[0,1,2],[5,6,0],[0,4,5]]);
Group with relations:
[0 1 2]
[5 6 0]
[0 4 5]
kash> g1 := AbelianSubGroup(g0, [1,0,0]);
Group with relations:
[5]
kash> g2 := AbelianSubGroup(g0, [[0,1,0],[0,0,5]]);
Group with relations:
[1 1]
[0 3]
kash> g := AbelianGroupUnite(g1, g2);
Group with relations:
[15]

```

NAME	AbelianHomGroup
PURPOSE	Returns the group consisting of homomorphisms between two groups.
SYNTAX	$h := \text{AbelianHomGroup}(g_1, g_2);$ $\text{groups } g_1, g_2, h$
DESCRIPTION	Returns the group h consisting of homomorphisms from g_1 to g_2 .
SEE ALSO	AbelianGroupHomCreate ,
EXAMPLE	

```
kash> g1 := AbelianGroupCreate([[1,2,3],[2,4,0],[2,0,0]]);;
kash> g2 := AbelianGroupCreate([[1,0,0],[0,4,0],[1,1,1]]);;
kash> h := AbelianHomGroup(g1, g2);
Group with relations:
[2 0]
[0 4]
```

elt is an element of h :

```
kash> elt := AbelianGroupEltCreate(h, [0,1]);
[0 1]
```

hom is a homomorphism from g_1 to g_2 :

```
kash> hom := AbelianGroupDiscreteExp(elt);
HomMatrix =
[ 0  0  0]
[ 0  3  0]
[ 0 -2  0]
from Group with relations:
[1 2 3]
[2 4 0]
[2 0 0]
to Group with relations:
```

[1 0 0]

[0 4 0]

[1 1 1]

kash> elt1 := AbelianGroupEltCreate(g1, [1,1,0]);

[1 1 0]

kash> elt2 := hom*elt1;

[0 3 0]

NAME	AbelianMultiHomGroup
PURPOSE	Computes the group of multilinear mappings.
SYNTAX	$L := \text{AbelianMultiHomGroup}([g_1, \dots, g_n], h);$ <p> groups g_1, \dots, g_n, h list L </p>
DESCRIPTION	<p>Computes the group of multilinear mappings from $d = g_1 \times \dots \times g_n$ to h. <code>AbelianMultiHomGroup</code> returns a list with three entries:</p> <ol style="list-style-type: none"> 1. group of multilinear mappings from $g_1 \times \dots \times g_n$ to h. 2. direct product $d = g_1 \times \dots \times g_n$ 3. tensor product $t = g_1 \otimes \dots \otimes g_n$. <p>You need d and t only if you want to create multilinear mappings from $g_1 \times \dots \times g_n$ to h corresponding to a given matrix using the function <code>AbelianGroupMultiHomCreate</code>.</p>
SEE ALSO	<code>AbelianGroupMultiHomCreate</code> ,
EXAMPLE	

```

kash> z := AbelianGroupCreate(Mat(Z, [[0]]));;
kash> z2 := AbelianGroupCreate(Mat(Z, [[2]]));;
kash> z4 := AbelianGroupCreate(Mat(Z, [[4]]));;
kash> z8 := AbelianGroupCreate(Mat(Z, [[8]]));;
kash> l := AbelianMultiHomGroup([z, z4, z8], z2*z8);
[ Group with relations:
  [2 0]
  [0 4], Group with relations:
  [0 0 0]
  [0 4 0]
  [0 0 8], Group with relations:
  [4] ]

```

mg is the group of multilinear mappings from $d = z \times z4 \times z8$ to $h = z2 \times z8$:

```
kash> mg := l[1];
Group with relations:
[2 0]
[0 4]
```

d is the direct product $z \times z_4 \times z_8$:

```
kash> d := l[2];
Group with relations:
[0 0 0]
[0 4 0]
[0 0 8]
```

t is the tensor product $z \otimes z_4 \otimes z_8$:

```
kash> t := l[3];
Group with relations:
[4]
```

elt is element of group of multilinear mappings from $d = z \times z_4 \times z_8$ to $h = z_2 \times z_8$:

```
kash> elt := AbelianGroupEltCreate(mg, [1,0]);
[1 0]
```

f is a multilinear mapping from $d = z \times z_4 \times z_8$ to $h = z_2 \times z_8$:

```
kash> f := AbelianGroupDiscreteExp(elt);
MultiHom [ 1, 1, 1 ] from Group with relations:
[0 0 0]
[0 4 0]
[0 0 8] to Group with relations:
[2 0]
[0 8]
```

```
kash> a := AbelianGroupEltCreate(d, [1,1,1]);
[1 1 1]
kash> f * a;
[1 0]
```

NAME	AbelianQuotientGroup
PURPOSE	Returns the quotient of a group and a subgroup.
SYNTAX	$q := \text{AbelianQuotientGroup}(g, s);$ <div style="display: flex; justify-content: space-between;"> <div> groups g, q group s </div> <div>subgroup of g</div> </div>
DESCRIPTION	Returns the quotient group q of the group g and its subgroup s .
SEE ALSO	AbelianGroupCanonicalQuotient ,
EXAMPLE	

```

kash> g := AbelianGroupCreate([[0,1,2],[5,6,0],[0,4,5]]);
Group with relations:
[0 1 2]
[5 6 0]
[0 4 5]
kash> s := AbelianSubGroup(g, [3,3,3]);
Group with relations:
[5]
kash> q := AbelianQuotientGroup(g, s);
Group with relations:
[3 3 3]
[0 1 2]
[5 6 0]
[0 4 5]
kash> AbelianGroupSmithCreate(q);
Group with relations:
[3]

```


NAME	AbelianRayClassGroupAutoCreate		
PURPOSE	Given a field automorphism, this function generates the corresponding automorphism of abelian groups.		
SYNTAX	aut := AbelianRayClassGroupAutoCreate(G, sigma);		
	AbelianGroupHom	aut	
	AbelianGroup	G	must be RayClassGroupToAbelianGroup
	KASH function	sigma	acting on ideals

DESCRIPTION

EXAMPLE We demonstrate this in $\mathbb{Q}(\sqrt{10})$:

```
kash> o := OrderMaximal(Z, 2, 10);
Generating polynomial: x^2 - 10
Discriminant: 40

kash> G := RayClassGroupToAbelianGroup(27*o);
RayClassGroupToAbelianGroup(<27>, [ ])
Group with relations:
[18]
kash> OrderAutomorphisms(o);
[ [0, 1], [0, -1] ]
kash> aut := AbelianRayClassGroupAutoCreate(G, x->IdealAutomorphism(x, 2));
HomMatrix =
[1]
from RayClassGroupToAbelianGroup(<27>, [ ])
Group with relations:
[18]
to RayClassGroupToAbelianGroup(<27>, [ ])
Group with relations:
[18]
```

This should be equivalent to (but is faster for larger groups):

```
kash> l := AbelianGroupGenerators(G);
[ [1] ]
kash> Apply(l, AbelianGroupDiscreteExp);
```

```
kash> Apply(1, x->IdealAutomorphism(x, 2));
kash> Apply(1, x->AbelianGroupDiscreteLog(G, x));
kash> Apply(1, x->x.expvec);
kash> IsMat(1);
true
kash> aut2 := AbelianGroupHomCreate(G, G, 1);
HomMatrix =
[1]
from RayClassGroupToAbelianGroup(<27>, [ ])
Group with relations:
[18]
to RayClassGroupToAbelianGroup(<27>, [ ])
Group with relations:
[18]
```

NAME	AbelianRayGroupImbed
PURPOSE	Imbeds a ClassGroup into a RayClassGroup.
SYNTAX	<pre>dualhom := AbelianRayGroupImbed(G,g); abstract rayclassgroup G abstract classgroup g</pre>
DESCRIPTION	<p>If $I^{(\mathfrak{f})}/P_{\mathfrak{f}}$ is a ray class group with conductor \mathfrak{f} and $I^{\mathfrak{f}'}/U_{\mathfrak{f}'}$ is a classgroup with conductor \mathfrak{f}', then we may imbed $U_{\mathfrak{f}'}$ canonically into $I^{(\mathfrak{f})}/P_{\mathfrak{f}}$: $U_{\mathfrak{f}'}^{(\mathfrak{f})}/P_{\mathfrak{f}}$ is a subgroup of $I^{(\mathfrak{f})}/P_{\mathfrak{f}}$.</p>
SEE ALSO	RayClassGroupToAbelianGroup ,
EXAMPLE	<pre>kash> O:=OrderMaximal(x^3-x^2-9*x+8);; kash> G :=RayClassGroupToAbelianGroup(25*O);; kash> g:=RayClassGroupToAbelianGroup(1*O);; kash> u:=AbelianRayGroupImbed(G,g); Group with relations: [2 0] [0 5]</pre>

NAME	AbelianSubGroup
PURPOSE	Creates a subgroup.
SYNTAX	<pre>s := AbelianSubGroup([g,] L [, generators]); s := AbelianSubGroup(g, mat [, generators]);</pre> <p> groups g, s list L list of elements of g or list of exponent vectors matrix mat boolean generators </p>
DESCRIPTION	Creates a subgroup s of the group g . The subgroup s is generated by the elements of the list L . If you use <code>s := AbelianSubGroup(g, mat);</code> then the subgroup s is generated by the elements represented by the rows of <code>mat</code> . If the last (optional) parameter <code>generators</code> is <code>true</code> then the new generators are computed, too.
EXAMPLE	

```
kash> g := AbelianGroupCreate([[0,1,2],[5,6,0],[0,4,5]]);;
```

Create subgroup with list of exponent vectors:

```
kash> s := AbelianSubGroup(g, [[0,1,0],[0,0,5]]);
Group with relations:
[1 1]
[0 3]
```

Create subgroup with list of group elements:

```
kash> elt1 := AbelianGroupEltCreate(g, [0,1,0]);;
kash> elt2 := AbelianGroupEltCreate(g, [0,0,5]);;
kash> s := AbelianSubGroup([elt1, elt2]);
Group with relations:
[1 1]
[0 3]
```

Create subgroup with matrix:

```
kash> mat := Mat(Z, [[0,1,0], [0,0,5]]);;  
kash> s := AbelianSubGroup(g, mat);  
Group with relations:  
[1 1]  
[0 3]
```

Abs

NAME Abs

PURPOSE Returns the absolute value of a number.

SYNTAX `a := Abs(x);`

 complex a

 complex x

EXAMPLE

```
kash> Abs(-3);
```

```
3
```

```
kash> Abs(Comp(1,1));
```

```
1.414213562373095048801688724209698078569671875377
```

NAME	Alff
PURPOSE	Creates an algebraic function field $F/k(T)$.
SYNTAX	$F := \text{Alff}(f);$ <div style="margin-left: 100px;"> algebraic function field F polynomial f </div>
DESCRIPTION	<p>Let $f \in k[T, y]$ be an irreducible polynomial which is separable in y, where k is a finite field or a number field (represented by \mathbb{Q} or by number field orders). This function defines the algebraic function field</p> $F := k(T, \rho) \quad \text{where } f(T, \rho) = 0.$ <p>In KASH, F is often also considered as a finite extension of function fields $F/k(T)$ such as in minimal polynomial computations or in determining the places of F over places of $k(T)$. For further information, see the KASH manual.</p>
SEE ALSO	AlffInit , AlffOrders , AlffGenus , AlffPlaceSplit ,

EXAMPLE The definition of an algebraic function field:

```
kash> AlffInit(FF(5,2));
"Defining global variables: k, w, kT, kTf, kTy, T, y, AlffGlobals"
kash> F := Alff(y^3+T^4+1);
Algebraic function field defined by
$.1^3 + $.2^4 + 1
over
Univariate rational function field over GF(5^2)
Variables: T
```

NAME	AlffCanonicalDivisor
PURPOSE	Computes a canonical divisor of an algebraic function field F/k .
SYNTAX	$W := \text{AlffCanonicalDivisor}(F);$ <div style="display: flex; justify-content: space-between;"> alff divisor W </div> <div style="display: flex; justify-content: space-between;"> algebraic function field F </div>
DESCRIPTION	Let F/k be an algebraic function field and ω be a differential of F/k . The divisor $W = (\omega)$ of the differential ω is called a canonical divisor of F/k . For W holds $\deg W = 2g - 2$ and $\dim W = g$, where g is the genus of F/k .
SEE ALSO	AlffDifferentDeg , AlffDifferent , AlffDiffDivisor , AlffDivisorLDim , AlffGenus , AlffInit , AlffOrders ,
EXAMPLE	

```

kash> AlffInit(FF(5,1));;
kash> AlffOrders(y^4 + T^7 + 1);
"Defining global variables: F, o, oi, one"
kash> AlffCanonicalDivisor(F);
Alff divisor
[ [ Alff place < [ 1/T, 0, 0, 0 ], [ 0, 1, 0, 0 ] >, -5 ],
[ Alff place < [ T + 1, 0, 0, 0 ], [ 0, 1, 0, 0 ] >, 3 ],
[ Alff place < [ T^6 + 4*T^5 + T^4 + 4*T^3 + T^2 + 4*T + 1, 0, 0, 0 ], [ 0, 1,\
0, 0 ] >, 3 ] ]

```


NAME	AlffClassGroup
PURPOSE	Computes the structure of the divisor class group of a global function field.
SYNTAX	<pre>Cl := AlffClassGroup(F [, b] [, A] [, "fast"]);</pre> <p> list Cl of integer and list (class group structure) alff F global function field integer b bound for degree of places to be used alff divisor A for reduction </p>
DESCRIPTION	<p>Let F/k be a global function field. The divisor class group is the factor group of all divisors factored by the principle divisors. This function returns a list Cl: Cl[1] is the class number and Cl[2] is a list of integers $1 < c_1 \mid \dots \mid c_m$ such that the class group of F/k is isomorphic to $\mathbb{Z}/c_1\mathbb{Z} \times \dots \times \mathbb{Z}/c_m\mathbb{Z}$. The function requires that the constant field of definition is the exact constant field.</p> <p>The following additional parameters can be given:</p> <ul style="list-style-type: none"> • b is a bound for the places to be considered in the factor base. For large q or g it can be reasonable to take smaller values for b than returned by <code>AlffClassGroupGenBoundStrong(F)</code>. • A is a reduction divisor. It must factor over the factor base (which is tested automatically) and should have small, positive degree. If the degree $[F : k(T)]$ is large than it can be reasonable to take e.g. <code>A1 := AlffDivisorDegOne(F)</code>. • If b is chosen very small and if the class group is a product of many cyclic factors then the option "fast" gives a speedup. The result of the computation however is not proven be correct anymore. <p>The algorithm is described in [Heß99].</p>
SEE ALSO	AlffClassGroupGenBound , AlffClassNumberApprox , AlffDivisorDegOne , AlffLPoly , Alff , AlffInit , AlffOrders ,
EXAMPLE	<pre>kash> AlffInit(FF(7));; kash> AlffOrders(y^2 + T^3 + T + 1);; kash> AlffClassGroup(F); [11, [11]]</pre>

NAME	AlffClassGroupGenBound
PURPOSE	Returns a bound for the degree of prime divisors which generate the divisor class group.
SYNTAX	<pre> b := AlffClassGroupGenBound(q, g); b := AlffClassGroupGenBound(F); integer b generation bound integers q, g exact constant field size and genus of F alff F global function field </pre>
DESCRIPTION	<p>Let F/k be a global function field of genus g over the exact constant field of q elements, let A_1 be a divisor of degree one and let S be the set of prime divisors of degree $\leq b$. This function computes $b \leq \lceil 2 \log_q(4g - 2) \rceil$ without using further information on F such that the classes of the elements of $S \cup \text{supp}(A_1)$ generate the divisor class group of F/k. See [Heß99] for details.</p>
SEE ALSO	<p>AlffClassGroupGenBoundStrong, AlffClassGroupProdBound, AlffClassGroup, AlffPlacesNum, AlffInit, AlffOrders, AlffGenus,</p>
EXAMPLE	<pre> kash> AlffInit(FF(7, 2));; kash> AlffOrders(y^4 + T^7 + T);; kash> b := AlffClassGroupGenBound(F); 2 kash> Sum([1..b], i->AlffPlacesNum(F, i)); 848 </pre>

NAME	<code>AlffClassGroupGenBoundStrong</code>
PURPOSE	Return a bound for the degree of prime divisors which generate the divisor class group.
SYNTAX	<pre> b := AlffClassGroupGenBoundStrong(F); integer b generation bound alff F global function field </pre>
DESCRIPTION	Let F/k be a global function field of genus g over the exact constant field of q elements, let A_1 be a divisor of degree one and let S be the set of prime divisors of degree $\leq b$. This function computes $b \leq \lceil 2 \log_q(4g - 2) \rceil$ using information on the number of places of F of bounded degree such that the classes of the elements of $S \cup \text{supp}(A_1)$ generate the divisor class group of F/k . See [Heß99] for details.
SEE ALSO	AlffClassGroupGenBound , AlffClassGroupProdBound , AlffClassGroup , AlffInit , AlffOrders ,

EXAMPLE

```

kash> AlffInit(FF(7, 2));
kash> AlffOrders(y^4 + T^7 + T);
kash> b := AlffClassGroupGenBoundStrong(F);
1
kash> Sum([1..b], i->AlffPlacesNum(F, i));
176

```

NAME	AlffClassGroupGens
PURPOSE	Computes generators of the divisor class group of a global function field.
SYNTAX	<p><code>L := AlffClassGroupGens(F, [b,] [A,] ["fast"]);</code></p> <p> <code>list</code> <code>L</code> of generating divisors <code>alff</code> <code>F</code> global function field <code>integer</code> <code>b</code> bound for degree of places to be used <code>alff divisor</code> <code>A</code> for reduction </p>
DESCRIPTION	<p>Let F/k be a global function field. The divisor class group is the factor group of all divisors factored by the principle divisors. This function returns a list $[A_1, D_1, \dots, D_m]$ of divisors of F/k such that $\deg(A_1) = 1$ and $\deg(D_i) = 0$. The classes $[D_i]$ correspond to generators of the cyclic factors of the class group of F/k as returned by <code>AlffClassGroup(F)</code>. The calling sequence is identical to that of <code>AlffClassGroup()</code>.</p>
SEE ALSO	<p>AlffClassGroup, AlffClassGroupGenBound, AlffClassGroupGenBoundStrongAlffDivisorDegOne, AlffInit, AlffOrders,</p>

EXAMPLE

```

kash> AlffInit(FF(7));;
kash> AlffOrders(y^2 + T^3 + T + 1);;
kash> AlffClassGroupGens(F);
[ Alff divisor
  [ [ Alff place < [ T + 1, 0 ], [ 6, 1 ] >, 1 ] ]
  , Alff divisor
  [ [ Alff place < [ T + 6, 0 ], [ 5, 1 ] >, -1 ],
  [ Alff place < [ 1/T, 0 ], [ 0, 1 ] >, 1 ] ]
]
```

NAME	AlffClassGroupPRank
PURPOSE	Computes the p -rank of the class group of a global function field.
SYNTAX	<pre>s := AlffClassGroupPRank(F);</pre> <p>integer s</p> <p>global function field F</p>
DESCRIPTION	<p>Let F/k be a global function field of characteristic p. Consider the subgroup $Cl^0(F/k)[p]$ of p-torsion elements of the group of divisor classes of degree zero. This function returns its dimension as an \mathbb{F}_p-vector space. Possible values range from 0 to g, where g is the genus of F/k.</p>
SEE ALSO	AlffHasseWittInvariant ,
EXAMPLE	<pre>kash> AlffInit(FF(2, 1)); "Defining global variables: k, w, kT, kTf, kTy, T, y, AlffGlobals" kash> F := Alff(T*y^3 + y + T^3); Algebraic function field defined by \$.1^3*\$.2 + \$.1 + \$.2^3 over Univariate rational function field over GF(2) Variables: T kash> AlffClassGroupPRank(F); 1</pre>

NAME	AlffClassNumberApprox
PURPOSE	Compute an approximation of the class number of a global function field.
SYNTAX	<pre>hbar := AlffClassNumberApprox(F, b);</pre> <p> real hbar approximated class number alff F global function field integer b bound </p>
DESCRIPTION	<p>Let F/k be a global function field of genus g over the exact constant field of q elements. If h is the class number of F/k we have (see [He399])</p> $\left \log(h / q^g) - \sum_{r=1}^b \frac{q^{-r}}{r} (N_r - (q^r + 1)) \right \leq \frac{2g}{q^{1/2} - 1} \cdot \frac{q^{-b/2}}{b + 1},$ <p>where N_r is the number of places of degree one in the constant field extension of degree r of F/k. By this formula an approximation of h is computed using the values of N_r for $1 \leq r \leq b$. The bound b can be determined from a given multiplicative error bound $a \in \mathbb{R}^{>1}$ via <code>AlffClassNumberApproxBound()</code>.</p>
SEE ALSO	AlffClassNumberApproxBound , AlffClassGroup , AlffLPoly , AlffInit , AlffOrders ,
EXAMPLE	<pre>kash> AlffInit(FF(7, 2));; kash> AlffOrders(y^4 + T^7 + T);; kash> b := AlffClassNumberApprox(F, 1); 21306983003206122.33001105081895164915977901789304 kash> b := AlffClassNumberApprox(F, 2); 17731831072348895.78932172128458952555036536781544</pre>

NAME	AlffClassNumberApproxBound
PURPOSE	Return a bound for the prime divisors to be considered for the approximation of the class number of a global function field.
SYNTAX	<pre> b := AlffClassNumberApproxBound(q, g, a); b := AlffClassNumberApproxBound(F, a); integer b approximation bound integers q, g exact constant field size and genus alff F global function field real a $\in \mathbb{R}^{>1}$ </pre>
DESCRIPTION	<p>Let F/k be a global function field of genus g over the exact constant field of q elements. This function computes $b \in \mathbb{Z}$ such that</p> $\frac{2g}{q^{1/2} - 1} \cdot \frac{q^{-b/2}}{b + 1} < \log(a)/2$ <p>holds. See function <code>AlffClassNumberApprox()</code>.</p>
SEE ALSO	AlffClassNumberApprox , AlffClassGroup , AlffLPoly , AlffInit , AlffOrders ,
EXAMPLE	

```

kash> AlffInit(FF(7, 2));;
kash> AlffOrders(y^4 + T^7 + T);;
kash> b := AlffClassNumberApproxBound(F, 2);
1
kash> b := AlffClassNumberApproxBound(F, 1.2);
2

```

NAME	AlffConstField
PURPOSE	Returns the constant field k of definition of an algebraic function field $F/k(T)$.
SYNTAX	<pre>k := AlffConstField(F);</pre> <div> <div>field</div> <div>algebraic function field</div> </div> <div> <div>k</div> <div>F</div> </div>
SEE ALSO	AlffDimExactConstField ,

EXAMPLE

```
kash> AlffInit(FF(5,2));
"Defining global variables: k, w, kT, kTf, kTy, T, y, AlffGlobals"
kash> AlffOrders(y^3+T^4+1);
"Defining global variables: F, o, oi, one"
kash> AlffConstField(F);
Finite field of size 5^2
```


NAME	AlffConstFieldSize
PURPOSE	Returns the size of the constant field of definition of a global function field.
SYNTAX	<pre>q := AlffConstFieldSize(F);</pre> <p>integer q size of constant field algebraic function field F</p>
SEE ALSO	AlffConstField , AlffIsGlobal , Characteristic ,
EXAMPLE	

```
kash> AlffInit(FF(2,3));
"Defining global variables: k, w, kT, kTf, kTy, T, y, AlffGlobals"
kash> F := Alff(y^3+T^3*y+T);
Algebraic function field defined by
$.1^3 + $.1*$.2^3 + $.2
over
Univariate rational function field over GF(2^3)
Variables: T

kash> AlffConstFieldSize(F);
8
```

NAME	AlffDeg
PURPOSE	Returns the degree of the extension $F/k(T)$.

SYNTAX	<code>n := AlffDeg(F);</code>				
	<table> <tr> <td>integer</td> <td>n</td> </tr> <tr> <td>algebraic function field</td> <td>F</td> </tr> </table>	integer	n	algebraic function field	F
integer	n				
algebraic function field	F				

SEE ALSO	AlffOrderPoly ,
----------	---------------------------------

EXAMPLE

```
kash> AlffInit(FF(5,2));
"Defining global variables: k, w, kT, kTf, kTy, T, y, AlffGlobals"
kash> AlffOrders(y^3+T^4+1);
"Defining global variables: F, o, oi, one"
kash> F;
Algebraic function field defined by
$.1^3 + $.2^4 + 1
over
Univariate rational function field over GF(5^2)
Variables: T

kash> AlffDeg(F);
3
```

NAME	AlffDiff
PURPOSE	Returns the exact differential dh of an alff element h .
SYNTAX	<pre>dh := AlffDiff(h);</pre> <div style="margin-left: 100px;"> <pre>alff differential dh alff element h</pre> </div>
SEE ALSO	AlffDiffDivisor , Alff ,

EXAMPLE

```
kash> AlffInit(FF(5,1));;
kash> AlffOrders(y^2 + T^3 + 1);
"Defining global variables: F, o, oi, one"
kash> dT := AlffDiff(AlffEltGenT(F));
Alff Differential
[ 1, 0 ] d[ T, 0 ]
kash> dy := AlffDiff(AlffEltGenY(F));
Alff Differential
[ 0, 4*T^2 ] / (T^3 + 1) d[ T, 0 ]
kash> dy / dT;
[ 0, 4*T^2 ] / (T^3 + 1)
kash> dT / dy;
[ 0, 1 ] / T^2
kash> dT + dy;
Alff Differential
[ T^3 + 1, 4*T^2 ] / (T^3 + 1) d[ T, 0 ]
```

NAME	AlffDiffAlff
PURPOSE	Returns the algebraic function field of an alff differential.
SYNTAX	<pre>F := AlffDiffAlff(hdx);</pre> <p>algebraic function field F</p> <p>alff differential hdx</p>

SEE ALSO [AlffDiff](#),

EXAMPLE

```
kash> AlffInit(FF(5,1));;
kash> AlffOrders(y^2 + T^3 + 1);
"Defining global variables: F, o, oi, one"
kash> dT := AlffDiff(AlffEltGenT(F));
Alff Differential
[ 1, 0 ] d[ T, 0 ]
kash> AlffDiffAlff(dT);
Algebraic function field defined by
$.1^2 + $.2^3 + 1
over
Univariate rational function field over GF(5)
Variables: T
```

NAME	AlffDiffCartier
PURPOSE	Applies the Cartier operator.
SYNTAX	<pre>hdx := AlffDiffCartier(gdx, r);</pre> <p> differentials hdx, gdx integer r number of iterated applications </p>
DESCRIPTION	<p>Let F/k be a global function field, $x \in F$ be a separating variable and $g dx \in \Omega(F/k)$ be a differential. The Cartier operator is defined by</p> $C(\omega) := (-d^{p-1}g/dx^{p-1})^{1/p} dx.$ <p>This function computes the the rth iterated application of C to $g dx$.</p>
SEE ALSO	AlffDiff ,
EXAMPLE	

```
kash> AlffInit(FF(5,1));;
kash> AlffOrders(y^4 + T^7 + 1);
"Defining global variables: F, o, oi, one"
kash> a := AlffElt(o, [0, 1/T, 0, T+1]);
[ 0, 1, 0, T^2 + T ] / T
kash> da := AlffDiff(a);
Alff Differential
[ 0, 2*T^7 + 4, 0, 4*T^8 + T^2 ] / (T^9 + T^2) d[ T, 0, 0, 0 ]
kash> AlffDiffCartier(da, 1);
Alff Differential
[ 0, 0, 0, 0 ] d[ T, 0, 0, 0 ]
kash> gdx := (1/a)*da;
Alff Differential
[ 4*T^17 + 4*T^16 + T^11 + 4*T^9 + 2*T^7 + T^4 + T^3 + 4, 0, 3*T^9 + 2*T^8 + 2\
*T^2 + T, 0 ] / (T^19 + 2*T^18 + T^17 + 2*T^12 + 4*T^11 + 2*T^10 + T^8 + T^5 +\
2*T^4 + T^3 + T) d[ T, 0, 0, 0 ]
kash> AlffDiffCartier(gdx, 1);
Alff Differential
[ 4*T^17 + 4*T^16 + T^11 + 4*T^9 + 2*T^7 + T^4 + T^3 + 4, 0, 3*T^9 + 2*T^8 + 2\
*T^2 + T, 0 ] / (T^19 + 2*T^18 + T^17 + 2*T^12 + 4*T^11 + 2*T^10 + T^8 + T^5 +\
2*T^4 + T^3 + T) d[ T, 0, 0, 0 ]
```

NAME	AlffDiffCartierMatrix
PURPOSE	Computes a representation matrix of the Cartier operator.
SYNTAX	<p><code>M := AlffDiffCartierMatrix(F, r);</code></p> <p> matrix M global function field F integer r C is r times applied. </p>
DESCRIPTION	<p>Let F/k be a global function field, $\omega_1, \dots, \omega_g \in \Omega(F/k)$ be a basis for the holomorphic differentials and $r \in \mathbb{Z}^{\geq 1}$. Let $M = (\lambda_{i,j})_{i,j} \in k^{g \times g}$ be the matrix such that</p> $C^r(\omega_i) = \sum_{m=1}^g \lambda_{i,m} \omega_m$ <p>for all $1 \leq i \leq g$. This function returns M.</p>
SEE ALSO	AlffDiff ,
EXAMPLE	

```

kash> AlffInit(FF(5,1));;
kash> AlffOrders(y^4 + T^7 + 1);
"Defining global variables: F, o, oi, one"
kash> AlffDiffCartierMatrix(F, 1);
[0 0 2 0 0 0 0 0 0]
[0 0 0 0 0 0 0 0 0]
[0 2 0 0 0 0 0 0 0]
[0 0 0 0 4 0 0 0 0]
[4 0 0 0 0 0 0 0 0]
[0 0 0 0 0 0 0 1 0]
[0 0 0 0 0 0 0 0 0]
[0 0 0 0 0 0 2 0 0]
[0 0 0 0 0 0 0 0 0]

```

NAME	AlffDiffDivisor
PURPOSE	Computes the divisor of an alff differential $h dT$.
SYNTAX	<pre>D := AlffDiffDivisor(hdT);</pre> <pre> alff divisor D alff differential hdT </pre>
SEE ALSO	AlffDivisor , AlffDiffSpace ,
EXAMPLE	

```

kash> AlffInit(F5,1));;
kash> AlffOrders(y^4 + T^7 + 1);
"Defining global variables: F, o, oi, one"
kash> dT := AlffDiff(AlffEltGenT(F));
Alff Differential
[ 1, 0, 0, 0 ] d[ T, 0, 0, 0 ]
kash> AlffDiffDivisor(dT);
Alff divisor
[ [ Alff place < [ 1/T, 0, 0, 0 ], [ 0, 1, 0, 0 ] >, -5 ],
[ Alff place < [ T + 1, 0, 0, 0 ], [ 0, 1, 0, 0 ] >, 3 ],
[ Alff place < [ T^6 + 4*T^5 + T^4 + 4*T^3 + T^2 + 4*T + 1, 0, 0, 0 ], [ 0, 1,\
0, 0 ] >, 3 ] ]

```

NAME	AlffDiffFirstKind
PURPOSE	Computes a basis for the differentials of the first kind (holomorphic differentials) of an algebraic function field F/k .
SYNTAX	<p><code>B := AlffDiffFirstKind(F);</code></p> <p>list B of alff differentials algebraic function field F</p>

SEE ALSO [AlffDiffSpace](#), [AlffDiffDivisor](#),

EXAMPLE

```
kash> AlffInit(F(5,1));;
kash> AlffOrders(y^4 + T^7 + 1);
"Defining global variables: F, o, oi, one"
kash> AlffDiffFirstKind(F);
[ Alff Differential
  [ 0, 1, 0, 0 ] / (T^7 + 1) d[ T, 0, 0, 0 ], Alff Differential
  [ 0, T, 0, 0 ] / (T^7 + 1) d[ T, 0, 0, 0 ], Alff Differential
  [ 0, T^2, 0, 0 ] / (T^7 + 1) d[ T, 0, 0, 0 ], Alff Differential
  [ 0, T^3, 0, 0 ] / (T^7 + 1) d[ T, 0, 0, 0 ], Alff Differential
  [ 0, T^4, 0, 0 ] / (T^7 + 1) d[ T, 0, 0, 0 ], Alff Differential
  [ 0, 0, 1, 0 ] / (T^7 + 1) d[ T, 0, 0, 0 ], Alff Differential
  [ 0, 0, T, 0 ] / (T^7 + 1) d[ T, 0, 0, 0 ], Alff Differential
  [ 0, 0, T^2, 0 ] / (T^7 + 1) d[ T, 0, 0, 0 ], Alff Differential
  [ 0, 0, 0, 1 ] / (T^7 + 1) d[ T, 0, 0, 0 ] ]
```


NAME	AlffDiffResiduum
PURPOSE	Computes the residuum of a differential.
SYNTAX	<pre>r := AlffDiffResiduum(P, hdT);</pre> <p> field element r the residuum of hdT at P alff place P alff differential hdT </p>
DESCRIPTION	Let F/k be an algebraic function field and $\omega = hdT$ be a differential of F/k . For a place P , which is currently required to be of degree one, this function returns the residuum of ω at P .
SEE ALSO	AlffDiffDivisor , AlffDiffValuation , AlffEltResiduum ,
EXAMPLE	

```
kash> AlffInit(FF(5,1));;
kash> AlffOrders(y^4 + T^7 + 1);
"Defining global variables: F, o, oi, one"
kash> P := AlffPlaceSplit(F, T+3)[1];
Alff place < [ T + 3, 0, 0, 0 ], [ 1, 1, 0, 0 ] >
kash> a := AlffEltGenY(F) + 1;
[ 1, 1, 0, 0 ]
kash> AlffEltValuation(P, a);
1
kash> hdT := (1/a^2)*AlffDiff(a^2);
Alff Differential
[ T^13 + T^6, T^6, 4*T^6, T^6 ] / (T^14 + 3*T^7 + 2) d[ T, 0, 0, 0 ]
kash> AlffDiffResiduum(P, hdT);
2
```

NAME AlffDiffSpace

PURPOSE Computes a basis for the space of Weil differentials

$$\Omega(D) := \{ \omega \in \Omega(F/k) \mid (\omega) \geq D \}$$

for a divisor D of an algebraic function field F/k .

SYNTAX `B := AlffDiffSpace(D);`

list B of alff differentials
alff divisor D

SEE ALSO [AlffDiffFirstKind](#), [AlffDiffDivisor](#),

EXAMPLE

```
kash> AlffInit(F5,1);;
kash> AlffOrders(y^4 + T^7 + 1);
"Defining global variables: F, o, oi, one"
kash> D := AlffDivisor(F);
Alff divisor
[ ]

kash> AlffDiffSpace(D);
[ Alff Differential
  [ 0, 1, 0, 0 ] / (T^7 + 1) d[ T, 0, 0, 0 ], Alff Differential
  [ 0, T, 0, 0 ] / (T^7 + 1) d[ T, 0, 0, 0 ], Alff Differential
  [ 0, T^2, 0, 0 ] / (T^7 + 1) d[ T, 0, 0, 0 ], Alff Differential
  [ 0, T^3, 0, 0 ] / (T^7 + 1) d[ T, 0, 0, 0 ], Alff Differential
  [ 0, T^4, 0, 0 ] / (T^7 + 1) d[ T, 0, 0, 0 ], Alff Differential
  [ 0, 0, 1, 0 ] / (T^7 + 1) d[ T, 0, 0, 0 ], Alff Differential
  [ 0, 0, T, 0 ] / (T^7 + 1) d[ T, 0, 0, 0 ], Alff Differential
  [ 0, 0, T^2, 0 ] / (T^7 + 1) d[ T, 0, 0, 0 ], Alff Differential
  [ 0, 0, 0, 1 ] / (T^7 + 1) d[ T, 0, 0, 0 ] ]
```

NAME	AlffDiffValuation
PURPOSE	Returns the order of a non zero alff differential at a place.
SYNTAX	<pre>v := AlffDiffValuation(p, fdx);</pre> <div> integer v alff place p alff differential fdx </div>
SEE ALSO	AlffDiffDivisor , AlffDiff ,

EXAMPLE

```
kash> AlffInit(FF(5,1));;
kash> AlffOrders(y^2 + T^3 + 1);
"Defining global variables: F, o, oi, one"
kash> dT := AlffDiff(AlffEltGenT(F));
Alff Differential
[ 1, 0 ] d[ T, 0 ]
kash> infty := AlffPlaceSplit(F, 1/T)[1];
Alff place < [ 1/T, 0 ], [ 0, 1 ] >
kash> AlffDiffValuation(infty, dT);
-3
```

NAME	AlffDifferent
PURPOSE	Computes the different of an algebraic function field extension $F/k(T)$.
SYNTAX	$D := \text{AlffDifferent}(F);$ <div style="margin-left: 100px;"> $\text{alff divisor} \quad D$ $\text{algebraic function field} \quad F$ </div>
DESCRIPTION	<p>Let F/k be an algebraic function field. Let P be a place of the rational function field $k(T)$ and P' a place of F over P. This function computes the divisor of F</p> $\text{Diff}(F/k(T)) = \sum_P \sum_{P'/P} d(P'/P)P'$ <p>where $d(P'/P)$ is the different exponent of P' over P.</p>
SEE ALSO	AlffDifferentDeg , AlffCanonicalDivisor , AlffDiffDivisor ,
EXAMPLE	

```

kash> AlffInit(F(5,1));;
kash> AlffOrders(y^4 + T^7 + 1);
"Defining global variables: F, o, oi, one"
kash> AlffDifferent(F);
Alff divisor
[ [ Alff place < [ T + 1, 0, 0, 0 ], [ 0, 1, 0, 0 ] >, 3 ],
[ Alff place < [ T^6 + 4*T^5 + T^4 + 4*T^3 + T^2 + 4*T + 1, 0, 0, 0 ], [ 0, 1,\
0, 0 ] >, 3 ],
[ Alff place < [ 1/T, 0, 0, 0 ], [ 0, 1, 0, 0 ] >, 3 ] ]

```

NAME	AlffDifferentDeg
PURPOSE	Computes the degree of the different of an algebraic function field extension $F/k(T)$.
SYNTAX	<pre>d := AlffDifferentDeg(F); integer d algebraic function field F</pre>
SEE ALSO	AlffDifferent , AlffCanonicalDivisor , AlffDiffDivisor ,
EXAMPLE	

```
kash> AlfflInit(FF(5,1));
kash> AlffOrders(y^4 + T^7 + 1);
"Defining global variables: F, o, oi, one"
kash> AlffDifferentDeg(F);
24
```

NAME	AlffDifferentiation												
PURPOSE	Computes higher differentiations of function field elements.												
SYNTAX	<pre>b := AlffDifferentiation(p, m, a);</pre> <table><tr><td>alff element</td><td>b</td><td>m-times differentiated element</td></tr><tr><td>integer</td><td>m</td><td></td></tr><tr><td>alff place or element</td><td>p</td><td></td></tr><tr><td>alff element</td><td>a</td><td>element to differentiate</td></tr></table>	alff element	b	m -times differentiated element	integer	m		alff place or element	p		alff element	a	element to differentiate
alff element	b	m -times differentiated element											
integer	m												
alff place or element	p												
alff element	a	element to differentiate											
DESCRIPTION	<p>Let F/k be an algebraic function field. For a separating element p and an integer m this function returns the m-th differentiation of $a \in F$. If p is a place, a local parameter of the place as returned by <code>AlffPlacePrimeElt()</code> is taken. Remark: The implementation is currently not efficient if m is larger than the positive characteristic.</p>												
SEE ALSO	AlffDiff , AlffWronskian , AlffRamDivisor ,												
EXAMPLE													

```

kash> AlffInit(FF(5,1));;
kash> AlffOrders(y^2 + T^3 + 1);
"Defining global variables: F, o, oi, one"
kash> T := AlffEltGenT(F);
[ T, 0 ]
kash> AlffDifferentiation(T, 1, T);
[ 1, 0 ]
kash> AlffDifferentiation(T, 2, T);
[ 0, 0 ]
kash> AlffDifferentiation(T, 1, T^2);
[ 2*T, 0 ]
kash> AlffDifferentiation(T, 2, T^2);
[ 1, 0 ]
kash> AlffDifferentiation(T, 5, T^5);
[ 1, 0 ]
kash> AlffDifferentiation(T, 5, T^7);
[ T^2, 0 ]

```

NAME	AlffDimExactConstField
PURPOSE	Returns the k -dimension of the exact constant field \tilde{k} of an algebraic function field $F/k(T)$.
SYNTAX	<pre> 1 := AlffDimExactConstField(F); integer 1 algebraic function field F </pre>
DESCRIPTION	The function returns $[\tilde{k} : k]$, where \tilde{k} is the algebraic closure of k in F .
SEE ALSO	AlffConstField ,
EXAMPLE	

```

kash> AlffInit(FF(5,1));
"Defining global variables: k, w, kT, kTf, kTy, T, y, AlffGlobals"
kash> AlffOrders(y^6 + (2*T^4 + 1)*y^3 + T^8 + T^4 + 2);
"Defining global variables: F, o, oi, one"
kash> AlffDimExactConstField(F);
2

```

NAME	AlffDivisor																							
PURPOSE	Creation of divisors.																							
SYNTAX	<pre>D := AlffDivisor(F); D := AlffDivisor(a); D := AlffDivisor(P); D := AlffDivisor(I, J); D := AlffDivisor(u, v);</pre> <table><tr><td>alff divisor</td><td>D</td><td></td></tr><tr><td>algebraic function field</td><td>F</td><td></td></tr><tr><td>alff order element</td><td>a</td><td></td></tr><tr><td>alff place</td><td>P</td><td></td></tr><tr><td>alff order ideal</td><td>I</td><td>of the finite maximal order</td></tr><tr><td>alff order ideal</td><td>J</td><td>of the infinite maximal order</td></tr><tr><td>alff order elements</td><td>u,v</td><td></td></tr></table>			alff divisor	D		algebraic function field	F		alff order element	a		alff place	P		alff order ideal	I	of the finite maximal order	alff order ideal	J	of the infinite maximal order	alff order elements	u,v	
alff divisor	D																							
algebraic function field	F																							
alff order element	a																							
alff place	P																							
alff order ideal	I	of the finite maximal order																						
alff order ideal	J	of the infinite maximal order																						
alff order elements	u,v																							
DESCRIPTION	<p>This function allows the creation of algebraic function field divisors. There are several possibilities for the arguments:</p> <ul style="list-style-type: none">• Given the algebraic function field F the zero divisor is created,• given an alff order element a its principal divisor is created,• given an alff place \mathfrak{P} its prime divisor is created,• given a pair of alff order ideals I, J or alff order elements u, v in the finite or infinite maximal order, the divisor corresponding to their ideal factorizations is created.																							
SEE ALSO	AlffPlaceSplit , AlffIdealFactor , AlffEltMove , Alff ,																							

EXAMPLE

```

kash> AlffInit(FF(5,1));
"Defining global variables: k, w, kT, kTf, kTy, T, y, AlffGlobals"
kash> AlffOrders(y^2+T^3+1);
"Defining global variables: F, o, oi, one"
kash> AlffDivisor(F);
Alff divisor
[ ]

kash> a := AlffElt(o, T);

```



```

[ T, 0 ]
kash> AlffDivisor(a);
Alff divisor
[ [ Alff place < [ T, 0 ], [ 2, 1 ] >, 1 ],
  [ Alff place < [ T, 0 ], [ 3, 1 ] >, 1 ],
  [ Alff place < [ 1/T, 0 ], [ 0, 1 ] >, -2 ] ]

kash> l := AlffPlaceSplit(F, T);
[ Alff place < [ T, 0 ], [ 2, 1 ] >, Alff place < [ T, 0 ], [ 3, 1 ] > ]
kash> D := AlffDivisor(l[1]);
Alff divisor
[ [ Alff place < [ T, 0 ], [ 2, 1 ] >, 1 ] ]

kash> l := AlffPlacesDegOne(F);
[ Alff place < [ 1/T, 0 ], [ 0, 1 ] >, Alff place < [ T, 0 ], [ 2, 1 ] >,
  Alff place < [ T, 0 ], [ 3, 1 ] >, Alff place < [ T + 3, 0 ], [ 1, 1 ] >,
  Alff place < [ T + 3, 0 ], [ 4, 1 ] >,
  Alff place < [ T + 1, 0 ], [ 0, 1 ] > ]
kash> 3*Sum(l) + D;
Alff divisor
[ [ Alff place < [ 1/T, 0 ], [ 0, 1 ] >, 3 ],
  [ Alff place < [ T, 0 ], [ 2, 1 ] >, 4 ],
  [ Alff place < [ T, 0 ], [ 3, 1 ] >, 3 ],
  [ Alff place < [ T + 3, 0 ], [ 1, 1 ] >, 3 ],
  [ Alff place < [ T + 3, 0 ], [ 4, 1 ] >, 3 ],
  [ Alff place < [ T + 1, 0 ], [ 0, 1 ] >, 3 ] ]

kash> 2*D > D;
true
kash> 2*D > D + l[1];
false

```

NAME AlffDivisorAlff

PURPOSE Given a divisor this function returns the algebraic function field it belongs to.

SYNTAX $F := \text{AlffDivisorAlff}(D);$

algebraic function field F

alff divisor D

SEE ALSO [AlffOrderAlff](#), [AlffPlaceAlff](#),

EXAMPLE

```
kash> AlffInit(FF(2,1));
"Defining global variables: k, w, kT, kTf, kTy, T, y, AlffGlobals"
kash> F := Alff(y^3+T^3*y+T);
Algebraic function field defined by
$.1^3 + $.1*$.2^3 + $.2
over
Univariate rational function field over GF(2)
Variables: T

kash> P := AlffPlaceSplit(F, T+1)[1];
Alff place < [ T + 1, 0, 0 ] >
kash> D := AlffDivisor(P);
Alff divisor
[ [ Alff place < [ T + 1, 0, 0 ] >, 1 ] ]

kash> AlffDivisorAlff(D);
Algebraic function field defined by
$.1^3 + $.1*$.2^3 + $.2
over
Univariate rational function field over GF(2)
Variables: T
```

NAME	AlffDivisorClassRep
PURPOSE	Find the class representation of a divisor class in given generators.
SYNTAX	$L := \text{AlffDivisorClassRep}(D);$ <p>list L exponents alff divisor D</p>
DESCRIPTION	<p>Let F/k be a global function field and A_1, D_1, \dots, D_m be a generating system as returned by <code>AlffClassGroupGens(F)</code>. Let d_i be the (finite) orders of $[D_i]$. This function returns a list consisting of $a, r_1, \dots, r_m \in \mathbb{Z}$ with $0 \leq r_i < d_i$ such that</p> $[D] = a[A_1] + r_1[D_1] + \dots + r_m[D_m]$ <p>holds.</p>
SEE ALSO	AlffClassGroupGens , AlffClassGroup , AlffInit , AlffOrders ,
EXAMPLE	

```

kash> AlffInit(FF(7));;
kash> AlffOrders(y^2 + T^3 + T + 1);;
kash> L := AlffClassGroupGens(F);
[ Alff divisor
  [ [ Alff place < [ T + 1, 0 ], [ 6, 1 ] >, 1 ] ]
  , Alff divisor
  [ [ Alff place < [ T + 6, 0 ], [ 5, 1 ] >, -1 ],
  [ Alff place < [ 1/T, 0 ], [ 0, 1 ] >, 1 ] ]
]
kash> D := -17135*L[1] + 15*L[2];
Alff divisor
[ [ Alff place < [ T + 1, 0 ], [ 6, 1 ] >, -17135 ],
[ Alff place < [ T + 6, 0 ], [ 5, 1 ] >, -15 ],
[ Alff place < [ 1/T, 0 ], [ 0, 1 ] >, 15 ] ]

kash> AlffDivisorClassRep(D);
[ -17135, 4 ]

```

NAME	AlffDivisorDeg
PURPOSE	Degree of a divisor.
SYNTAX	<pre>d := AlffDivisorDeg(D); integer d alff divisor D</pre>
DESCRIPTION	This function returns the degree of an alff divisor over the constant field of definition (not the exact constant field) of the algebraic function field.
SEE ALSO	AlffPlaceDeg , AlffDivisorLDim , AlffGenus ,
EXAMPLE	

```
kash> AlffInit(FF(3,1));
"Defining global variables: k, w, kT, kTf, kTy, T, y, AlffGlobals"
kash> AlffOrders(y^3+T^3*y+T);
"Defining global variables: F, o, oi, one"
kash> l := AlffPlaceSplit(F, 1/T);
[ Alff place < [ 1/T, 0, 0 ], [ 2/T, 1/T, (2*T + 2)/T ] >,
  Alff place < [ 1/T, 0, 0 ], [ 2, (2*T + 2)/T, 2 ] > ]
kash> D := Sum(l);
Alff divisor
[ [ Alff place < [ 1/T, 0, 0 ], [ 2/T, 1/T, (2*T + 2)/T ] >, 1 ],
  [ Alff place < [ 1/T, 0, 0 ], [ 2, (2*T + 2)/T, 2 ] >, 1 ] ]

kash> AlffDivisorDeg(D);
2
kash> a := AlffElt(o, [0, 1, 2]);
[ 0, 1, 2 ]
kash> AlffDivisorDeg(AlffDivisor(a));
0
```

NAME	AlffDivisorDegOne
PURPOSE	Computes a divisor of degree one for a global function field.
SYNTAX	<pre>D := AlffDivisorDegOne(F);</pre> <p>alff divisor D of degree one global function field F</p>
DESCRIPTION	Given a global function field this function determines a divisor of degree one.
SEE ALSO	AlffPlaces , AlffPlacesDegOne ,
EXAMPLE	

```
kash> AlffInit(FF(5,1));;
kash> AlffOrders(y^2+T^3+1);;
kash> AlffDivisorDegOne(F);
Alff divisor
[ [ Alff place < [ T + 1, 0 ], [ 0, 1 ] >, 1 ] ]
```

NAME	AlffDivisorDen
PURPOSE	Returns the denominator of an alff divisor.
SYNTAX	<pre>D2 := AlffDivisorDen(D);</pre> <pre> alff divisor D2 alff divisor D </pre>
DESCRIPTION	This function returns the denominator of an alff divisor.
SEE ALSO	AlffDivisorNum ,
EXAMPLE	

```

kash> AlffInit(FF(3,1));
"Defining global variables: k, w, kT, kTf, kTy, T, y, AlffGlobals"
kash> F := Alff(y^3+T^3*y+T);
Algebraic function field defined by
$.1^3 + $.1*$.2^3 + $.2
over
Univariate rational function field over GF(3)
Variables: T

kash> l := AlffPlaceSplit(F, 1/T);
[ Alff place < [ 1/T, 0, 0 ], [ 2/T, 1/T, (2*T + 2)/T ] >,
  Alff place < [ 1/T, 0, 0 ], [ 2, (2*T + 2)/T, 2 ] > ]
kash> D := l[1] - l[2];
Alff divisor
[ [ Alff place < [ 1/T, 0, 0 ], [ 2/T, 1/T, (2*T + 2)/T ] >, 1 ],
  [ Alff place < [ 1/T, 0, 0 ], [ 2, (2*T + 2)/T, 2 ] >, -1 ] ]

kash> AlffDivisorDen(D);
Alff divisor
[ [ Alff place < [ 1/T, 0, 0 ], [ 2, (2*T + 2)/T, 2 ] >, 1 ] ]

```

NAME	AlffDivisorIdeals
PURPOSE	Divisor corresponding ideals of maximal orders.
SYNTAX	$L := \text{AlffDivisorIdeals}(D);$ $\text{list} \quad L$ of finite and infinite ideals $\text{alff divisor } D$
DESCRIPTION	Given an alff divisor D this function returns the ideals of the finite and infinite maximal order whose ideal factorizations define $-D$ (note the minus sign).
SEE ALSO	AlffDivisor ,
EXAMPLE	

```

kash> AlffInit(FF(2,4));
"Defining global variables: k, w, kT, kTf, kTy, T, y, AlffGlobals"
kash> F := Alff(y^3+T^3*y+T);
Algebraic function field defined by
$.1^3 + $.1*$.2^3 + $.2
over
Univariate rational function field over GF(2^4)
Variables: T

kash> P := AlffPlaceSplit(F,T+1)[1];
Alff place < [ T + 1, 0, 0 ] >
kash> D := AlffDivisor(P);
Alff divisor
[ [ Alff place < [ T + 1, 0, 0 ] >, 1 ] ]

kash> I := AlffDivisorIdeals(D);
[ <
  [ 1 0 0]
  [ 0 1 0]
  [ 0 0 1]
  / T + 1
  >, < [ 1, 0, 0 ] > ]

```

NAME	AlffDivisorLBasis
PURPOSE	Computes a basis of a Riemann-Roch space
SYNTAX	<p><code>B := AlffDivisorLBasis(D);</code></p> <p><code>list</code> <code>B</code> of basis elements</p> <p><code>alff divisor</code> <code>D</code></p>
DESCRIPTION	Given a divisor D of an algebraic function field F this function returns a basis for the k -vector space $\mathcal{L}(D)$, where k denotes the constant field of definition of F . The basis elements are represented in the finite maximal order.
SEE ALSO	AlffDivisorLBasisShort , AlffDivisorLDim , AlffDivisorDeg , AlffDivisorLIndex , AlffGenus , Alff ,
EXAMPLE	

```

kash> AlffInit(FF(5,1));;
kash> AlffOrders(y^2+T^3+1);;
kash> infty := AlffPlaceSplit(F, 1/T)[1];
Alff place < [ 1/T, 0 ], [ 0, 1 ] >
kash> l := AlffPlaceSplit(F, T+3);
[ Alff place < [ T + 3, 0 ], [ 1, 1 ] >,
  Alff place < [ T + 3, 0 ], [ 4, 1 ] > ]
kash> D := 2*infty + 3*l[1] - l[2];
Alff divisor
[ [ Alff place < [ 1/T, 0 ], [ 0, 1 ] >, 2 ],
  [ Alff place < [ T + 3, 0 ], [ 1, 1 ] >, 3 ],
  [ Alff place < [ T + 3, 0 ], [ 4, 1 ] >, -1 ] ]

kash> AlffDivisorLBasis(D);
[ [ 4*T^3 + 4*T^2 + 2*T + 1, T ] / (T^3 + 4*T^2 + 2*T + 2),
  [ 4*T^4 + 4*T^3 + 2*T^2 + T, T^2 ] / (T^3 + 4*T^2 + 2*T + 2),
  [ 4*T^3 + 2*T^2 + 4, 1 ] / (T^3 + 4*T^2 + 2*T + 2),
  [ 4*T^4 + 2*T^3 + 4*T, T ] / (T^3 + 4*T^2 + 2*T + 2) ]

```


NAME	AlffDivisorLBasis
PURPOSE	Computes a basis of a Riemann-Roch space
SYNTAX	<p><code>B := AlffDivisorLBasis(D);</code></p> <p>list B of basis elements</p> <p>alff divisor D</p>
DESCRIPTION	Given a divisor D of an algebraic function field F this function returns a basis for the k -vector space $\mathcal{L}(D)$, where k denotes the constant field of definition of F . The basis elements are represented in the finite maximal order.
SEE ALSO	AlffDivisorLBasisShort , AlffDivisorLDim , AlffDivisorDeg , AlffDivisorLIndex , AlffGenus , Alff ,

EXAMPLE

```

kash> AlffInit(FF(5,1));;
kash> AlffOrders(y^2+T^3+1);;
kash> infty := AlffPlaceSplit(F, 1/T)[1];
Alff place < [ 1/T, 0 ], [ 0, 1 ] >
kash> l := AlffPlaceSplit(F, T+3);
[ Alff place < [ T + 3, 0 ], [ 1, 1 ] >,
  Alff place < [ T + 3, 0 ], [ 4, 1 ] > ]
kash> D := 2*infty + 3*l[1] - l[2];
Alff divisor
[ [ Alff place < [ 1/T, 0 ], [ 0, 1 ] >, 2 ],
  [ Alff place < [ T + 3, 0 ], [ 1, 1 ] >, 3 ],
  [ Alff place < [ T + 3, 0 ], [ 4, 1 ] >, -1 ] ]

kash> AlffDivisorLBasis(D);
[ [ 4*T^3 + 4*T^2 + 2*T + 1, T ] / (T^3 + 4*T^2 + 2*T + 2),
  [ 4*T^4 + 4*T^3 + 2*T^2 + T, T^2 ] / (T^3 + 4*T^2 + 2*T + 2),
  [ 4*T^3 + 2*T^2 + 4, 1 ] / (T^3 + 4*T^2 + 2*T + 2),
  [ 4*T^4 + 2*T^3 + 4*T, T ] / (T^3 + 4*T^2 + 2*T + 2) ]

```

NAME	AlffDivisorLBasisShort
PURPOSE	Computes a basis of a Riemann-Roch space.
SYNTAX	<p><code>B := AlffDivisorLBasisShort(D);</code></p> <p><code>list</code> <code>B</code> of pairs of basis elements b_i and degree bounds d_i</p> <p><code>alff divisor</code> <code>D</code></p>
DESCRIPTION	<p>Let $F = k(T, y)$ be an algebraic function field defined by $f(T, y) = 0$ over k. Given a divisor D of F this function returns a basis of the k-vector space</p> $\mathcal{L}(D) = \{a \in F^\times \mid (a) \geq -D\} \cup \{0\}$ <p>in the short form</p> $B = [[b_1, d_1], \dots, [b_k, d_k]]$ <p>with $b_i \in F^\times$ and $d_i \in \mathbb{Z}^{\geq 0}$ for all $1 \leq i \leq k$ and with $k \leq n$, where n denotes the degree in y of the defining equation f of F, such that</p> $\mathcal{L}(D) = \left\{ \sum_{i=1}^k \lambda_i b_i \mid \lambda_i \in k[T] \text{ with } \deg \lambda_i \leq d_i \text{ for } 1 \leq i \leq k \right\}.$ <p>The basis elements b_i are represented in the finite maximal order.</p> <p>With option "raw" a complete list of $n := \deg_y f$ tuples $[b_i, d_i]$ with $d_i \in \mathbb{Z}$ and the property as above is returned. The algorithm is described in [Heß99].</p>
SEE ALSO	AlffDivisorLBasis , AlffDivisorLDim , AlffDivisorLIndex , AlffDivisorDeg , AlffGenus , Alff ,

EXAMPLE

```

kash> AlffInit(FF(5,1));;
kash> AlffOrders(y^2+T^3+1);;
kash> infty := AlffPlaceSplit(F, 1/T)[1];
Alff place < [ 1/T, 0 ], [ 0, 1 ] >
kash> l := AlffPlaceSplit(F, T+3);
[ Alff place < [ T + 3, 0 ], [ 1, 1 ] >,
  Alff place < [ T + 3, 0 ], [ 4, 1 ] > ]
kash> D := 2*infty + 3*l[1] - l[2];
Alff divisor
[ [ Alff place < [ 1/T, 0 ], [ 0, 1 ] >, 2 ],
  [ Alff place < [ T + 3, 0 ], [ 1, 1 ] >, 3 ],
  [ Alff place < [ T + 3, 0 ], [ 4, 1 ] >, -1 ] ]

```

```
kash> AlffDivisorLBasisShort(D);  
[ [ [ 4*T^3 + 4*T^2 + 2*T + 1, T ] / (T^3 + 4*T^2 + 2*T + 2), 1 ],  
  [ [ 4*T^3 + 2*T^2 + 4, 1 ] / (T^3 + 4*T^2 + 2*T + 2), 1 ] ]
```

NAME	AlffDivisorLDim
PURPOSE	Dimension of a Riemann-Roch space.
SYNTAX	<pre>l := AlffDivisorLDim(D); integer l alff divisor D</pre>
DESCRIPTION	Let $F = k(T, y)$ be an algebraic function field defined by $f(T, y) = 0$ over k . Given a divisor D of F this function returns the dimension of $\mathcal{L}(D)$ as a k -vector space. Note that the field k is the constant field of definition (not the exact constant field) of F .
SEE ALSO	AlffDivisorLBasis , AlffDivisorLIndex , AlffDivisorDeg , AlffGenus , Alff ,
EXAMPLE	

```
kash> AlffInit(FF(5,1));;
kash> AlffOrders(y^2+T^3+1);;
kash> infty := AlffPlaceSplit(F, 1/T)[1];
Alff place < [ 1/T, 0 ], [ 0, 1 ] >
kash> l := AlffPlaceSplit(F, T+3);
[ Alff place < [ T + 3, 0 ], [ 1, 1 ] >,
  Alff place < [ T + 3, 0 ], [ 4, 1 ] > ]
kash> D := 2*infty + 3*l[1] - l[2];
Alff divisor
[ [ Alff place < [ 1/T, 0 ], [ 0, 1 ] >, 2 ],
  [ Alff place < [ T + 3, 0 ], [ 1, 1 ] >, 3 ],
  [ Alff place < [ T + 3, 0 ], [ 4, 1 ] >, -1 ] ]

kash> AlffDivisorLDim(D);
4
```

NAME	AlffDivisorLIndex
PURPOSE	Index of speciality of alff divisors.
SYNTAX	<pre>i := AlffDivisorLIndex(D); integer i divisor D</pre>
DESCRIPTION	<p>Let $F = k(T, y)$ be an algebraic function field defined by $f(T, y) = 0$ over k. Given a divisor D of F this function computes the index of speciality $i_k(D) = \dim_k \mathcal{L}(D) - \deg_k D + m * (g - 1)$, where g denotes the genus of F and m the dimension of the exact constant field over the constant field of definition k, as dimension over k.</p>
SEE ALSO	AlffDivisorLBasis , AlffDivisorLDim , AlffDivisorDeg , AlffGenus , Alff ,

EXAMPLE

```
kash> AlffInit(FF(5,1));;
kash> AlffOrders(y^2+T^3+1);;
kash> infty := AlffPlaceSplit(F, 1/T)[1];
Alff place < [ 1/T, 0 ], [ 0, 1 ] >
kash> l := AlffPlaceSplit(F, T+3);
[ Alff place < [ T + 3, 0 ], [ 1, 1 ] >,
  Alff place < [ T + 3, 0 ], [ 4, 1 ] > ]
kash> D := 2*infty + 3*l[1] - l[2];
Alff divisor
[ [ Alff place < [ 1/T, 0 ], [ 0, 1 ] >, 2 ],
  [ Alff place < [ T + 3, 0 ], [ 1, 1 ] >, 3 ],
  [ Alff place < [ T + 3, 0 ], [ 4, 1 ] >, -1 ] ]

kash> AlffDivisorLIndex(D);
0
```

NAME	AlffDivisorLargeLBasisShort
PURPOSE	Computes a basis of a large divisor.
SYNTAX	<pre>B := AlffDivisorLargeLBasis(D, "raw");</pre> <p>list B the short basis</p> <p>alff divisor D</p>
DESCRIPTION	<p>Given a divisor D of an algebraic function field F which exponents are very large this function returns a basis of $\mathcal{L}(D)$ similar to <code>AlffDivisorLBasisShort()</code> with option "raw". If the option "raw" is given to <code>AlffDivisorLargeLBasisShort()</code> the basis elements are represented as power product as in <code>AlffDivisorReduction()</code>.</p>
SEE ALSO	<p>AlffDivisorLargeLDim, AlffDivisorLBasisShort, AlffDivisorReduction, AlffDivisorLDim, AlffDivisorDeg, AlffDivisorLIndex, AlffGenus, Alff,</p>

EXAMPLE

```
kash> AlffInit(FF(7));
"Defining global variables: k, w, kT, kTf, kTy, T, y, AlffGlobals"
kash> AlffOrders(y^3 + T*y + T^7 + T + 1);
"Defining global variables: F, o, oi, one"
kash> p1 := AlffPlaceSplit(F, T)[1];
Alff place < [ T, 0, 0 ], [ 1, 1, 0 ] >
kash> p2 := AlffPlaceSplit(F, T)[2];
Alff place < [ T, 0, 0 ], [ 2, 1, 0 ] >
kash> p3 := AlffPlaceSplit(F, 1/T)[1];
Alff place < [ 1/T, 0, 0 ], [ 2/T, 2, (2*T + 3)/T ] >
kash> D := -p1 - p2 + 2*p3;
Alff divisor
[ [ Alff place < [ T, 0, 0 ], [ 1, 1, 0 ] >, -1 ],
[ Alff place < [ T, 0, 0 ], [ 2, 1, 0 ] >, -1 ],
[ Alff place < [ 1/T, 0, 0 ], [ 2/T, 2, (2*T + 3)/T ] >, 2 ] ]

kash> AlffDivisorDeg(D);
0
kash> AlffDivisorLargeLBasisShort(D, "raw");
[ [ [ [ [ [ 1, 0, 0 ] ], 1 ], [ [ [ T, 0, 0 ] ], 1 ] ], -1 ],
[ [ [ [ [ 0, 1, 0 ] ], 1 ], [ [ [ T, 0, 0 ] ], 1 ] ], -3 ],
[ [ [ [ [ 2, 3, 1 ] / T ], 1 ], [ [ [ T, 0, 0 ] ], 1 ] ], -4 ] ]
```

NAME	AlffDivisorLargeLDim
PURPOSE	Computes the dimension of a large divisor.
SYNTAX	<pre>d := AlffDivisorLargeLDim(D);</pre> <p>integer d the dimension alff divisor D</p>
DESCRIPTION	Given a divisor D of an algebraic function field F which exponents are very large this function returns the dimension of $\mathcal{L}(D)$ much more faster than <code>AlffDivisorLDim()</code> .
SEE ALSO	AlffDivisorLargeLBasisShort , AlffDivisorLBasisShort , AlffDivisorReduction , AlffDivisorLDim , AlffDivisorDeg , AlffDivisorLIndex , AlffGenus , Alff ,
EXAMPLE	<pre>kash> AlffInit(FF(7)); "Defining global variables: k, w, kT, kTf, kTy, T, y, AlffGlobals" kash> AlffOrders(y^3 + T*y + T^7 + T + 1); "Defining global variables: F, o, oi, one" kash> p1 := AlffPlaceSplit(F, T)[1]; Alff place < [T, 0, 0], [1, 1, 0] > kash> p2 := AlffPlaceSplit(F, T)[2]; Alff place < [T, 0, 0], [2, 1, 0] > kash> p3 := AlffPlaceSplit(F, 1/T)[1]; Alff place < [1/T, 0, 0], [2/T, 2, (2*T + 3)/T] > kash> D := -11107286187918025*p1 -271773078445978235700*p2 + > 271784185732166153725*p3; Alff divisor [[Alff place < [T, 0, 0], [1, 1, 0] >, -11107286187918025], [Alff place < [T, 0, 0], [2, 1, 0] >, -271773078445978235700], [Alff place < [1/T, 0, 0], [2/T, 2, (2*T + 3)/T] >, 271784185732166153725\]] kash> AlffDivisorDeg(D); 0 kash> AlffDivisorLargeLDim(D); 1</pre>

NAME	AlffDivisorNorm
PURPOSE	Computes the norm of a divisor.
SYNTAX	$N := \text{AlffDivisorNorm}(D);$ <div style="margin-left: 100px;"> $\begin{array}{ll} \text{qf element} & N \\ \text{alff divisor} & D \end{array}$ </div>
DESCRIPTION	Given a divisor D of an algebraic function field F this function returns the norm of D down to the rational subfield $k(x)$ as a rational function.
SEE ALSO	AlffEltGenT , AlffIdealNorm ,
EXAMPLE	

```

kash> AlffInit(FF(5,1));;
kash> AlffOrders(y^2+T^3+1);;
kash> infty := AlffPlaceSplit(F, 1/T)[1];
Alff place < [ 1/T, 0 ], [ 0, 1 ] >
kash> l := AlffPlaceSplit(F, T+3);
[ Alff place < [ T + 3, 0 ], [ 1, 1 ] >,
  Alff place < [ T + 3, 0 ], [ 4, 1 ] > ]
kash> D := 2*infty + 3*l[1] - l[2];
Alff divisor
[ [ Alff place < [ 1/T, 0 ], [ 0, 1 ] >, 2 ],
  [ Alff place < [ T + 3, 0 ], [ 1, 1 ] >, 3 ],
  [ Alff place < [ T + 3, 0 ], [ 4, 1 ] >, -1 ] ]

kash> AlffDivisorNorm(D);
(T^2 + T + 4)/T^2

```


NAME	AlffDivisorNum
PURPOSE	Returns the numerator of an alff divisor.
SYNTAX	<pre>D1 := AlffDivisorNum(D); alff divisor D1 alff divisor D</pre>
DESCRIPTION	This function returns the numerator of an alff divisor.
SEE ALSO	AlffDivisorDen ,
EXAMPLE	

```
kash> AlffInit(FF(3,1));
"Defining global variables: k, w, kT, kTf, kTy, T, y, AlffGlobals"
kash> F := Alff(y^3+T^3*y+T);
Algebraic function field defined by
$.1^3 + $.1*$.2^3 + $.2
over
Univariate rational function field over GF(3)
Variables: T

kash> l := AlffPlaceSplit(F, 1/T);
[ Alff place < [ 1/T, 0, 0 ], [ 2/T, 1/T, (2*T + 2)/T ] >,
  Alff place < [ 1/T, 0, 0 ], [ 2, (2*T + 2)/T, 2 ] > ]
kash> D := l[1] - l[2];
Alff divisor
[ [ Alff place < [ 1/T, 0, 0 ], [ 2/T, 1/T, (2*T + 2)/T ] >, 1 ],
  [ Alff place < [ 1/T, 0, 0 ], [ 2, (2*T + 2)/T, 2 ] >, -1 ] ]

kash> AlffDivisorNum(D);
Alff divisor
[ [ Alff place < [ 1/T, 0, 0 ], [ 2/T, 1/T, (2*T + 2)/T ] >, 1 ] ]
```

NAME	AlffDivisorPlaces
PURPOSE	Places and exponents of an alff divisor.
SYNTAX	<p><code>L := AlffDivisorPlaces(D);</code></p> <p>list L of lists of alff places and integers alff divisor D</p>
DESCRIPTION	This function returns a list containing pairs of places and exponents as occuring in a given alff divisor.
SEE ALSO	AlffDivisor ,
EXAMPLE	

```

kash> AlffInit(FF(2,3));
"Defining global variables: k, w, kT, kTf, kTy, T, y, AlffGlobals"
kash> AlffOrders(y^2+y+T^3+T);
"Defining global variables: F, o, oi, one"
kash> D := AlffDivisor(AlffElt(o, [0, 1]));
Alff divisor
[ [ Alff place < [ T, 0 ], [ 0, 1 ] >, 1 ],
  [ Alff place < [ T + 1, 0 ], [ 0, 1 ] >, 2 ],
  [ Alff place < [ 1/T, 0 ], [ 0, 1 ] >, -3 ] ]

kash> AlffDivisorPlaces(D);
[ [ Alff place < [ T, 0 ], [ 0, 1 ] >, 1 ],
  [ Alff place < [ T + 1, 0 ], [ 0, 1 ] >, 2 ],
  [ Alff place < [ 1/T, 0 ], [ 0, 1 ] >, -3 ] ]

```

NAME	AlffDivisorReduction
PURPOSE	Computes a reduced divisor.
SYNTAX	<pre> L := AlffDivisorReduction(D); L := AlffDivisorReduction(D, A); list L as above alff divisor D to reduce alff divisor D </pre>
DESCRIPTION	<p>Let A, D be divisors of an algebraic function field F of genus g and $\deg(A) \geq 1$. Let $r \in \mathbb{Z}$ be maximal such that $D = \tilde{D} + rA - (a)$ with a positive divisor \tilde{D} of F and $a \in F^\times$. Then $\deg(\tilde{D}) < g + \deg(A)$ and \tilde{D} is maximally reduced along A. This function returns on input of D and A a list containing \tilde{D}, r, A and a list $((a_{i,j})_j, e_i)_i$, such that $a = \prod_{i,j} a_{i,j}^{e_i}$. If A is omitted, $A := (T)_\infty$ is taken. \tilde{D} is returned in ideal representation. The algorithm is described in [Heß99].</p>
SEE ALSO	AlffDivisorLargeLDim ,
EXAMPLE	<pre> kash> AlffInit(F7); "Defining global variables: k, w, kT, kTf, kTy, T, y, AlffGlobals" kash> AlffOrders(y^3 + T*y + T^7 + T^2 + 1); "Defining global variables: F, o, oi, one" kash> p1 := AlffPlaceSplit(F, T)[1]; Alff place < [T, 0, 0], [1, 1, 0] > kash> p2 := AlffPlaceSplit(F, T+1)[1]; Alff place < [T + 1, 0, 0], [5, 1, 0] > kash> D := 51*p1 + 37*p2; Alff divisor [[Alff place < [T, 0, 0], [1, 1, 0] >, 51], [Alff place < [T + 1, 0, 0], [5, 1, 0] >, 37]] kash> A := 1*AlffPlaceSplit(F, T)[2]; Alff divisor [[Alff place < [T, 0, 0], [2, 1, 0] >, 1]] kash> L := AlffDivisorReduction(D, A); [< </pre>

```

[ T^6 + 4*T^5 + 6*T^4 + 5*T^3 + T^2 + 3*T + 4    0    T^5 + 6*T^4 + 5*T^3 + T\
^2 + 5*T]
[0    T^6 + 4*T^5 + 6*T^4 + 5*T^3 + T^2 + 3*T + 4    6*T^5 + 5*T^3 + 3*T^2 +\
6*T + 2]
[0    0    1]
/ T^6 + 4*T^5 + 6*T^4 + 5*T^3 + T^2 + 3*T + 4
>, <
[1 0 0]
[0 1 0]
[0 0 1]
/ (1)
> ]
, 82, Alff divisor
[ [ Alff place < [ T, 0, 0 ], [ 2, 1, 0 ] >, 1 ] ]
,
[ [ [ [ 2*T^11 + 4*T^10 + 3*T^9 + 6*T^8 + 3*T^7 + 2*T^6 + T^5 + 4*T^4 + 4*T\
3 + 6*T + 3, 6*T^9 + 4*T^8 + 4*T^7 + 3*T^4 + 6*T^2 + 4, 2*T^7 + T^6 + 4*T^5 + \
5*T^2 + 3 ] / (T^11 + 6*T^10 + 5*T^8 + 2*T^7 + 3*T^6 + 2*T^5 + T^4 + 4*T^2 + T\
) ], 1 ],
[ [ [ 4*T^11 + 4*T^9 + 6*T^8 + 4*T^7 + 3*T^5 + 5*T^3 + 3*T + 4, T^8 + 5*\
T^7 + 4*T^6 + 4*T^5 + 2*T^2 + 6, 6*T^7 + 4*T^6 + 4*T^5 + 4*T^4 + T^3 + 2*T^2 +\
4*T + 2 ] / (T^12 + 3*T^11 + T^9 + 3*T^7 + 4*T^6 + 3*T^5 + 4*T^4 + 3*T^3 + T\
2) ], 2 ],
[ [ [ 3*T^7 + 5*T^6 + T^5 + 2*T^4 + 3*T^3 + T^2 + 4*T + 6, T^3 + 4*T^2 +\
2*T + 1, 5*T^2 + 3*T + 6 ] / (T^7 + 3*T^6 + 5*T^5 + 3*T^4 + 6*T^3 + 4*T^2 + 6\
*T + 1) ], 4 ],
[ [ [ 3*T^10 + 4*T^9 + 3*T^8 + 3*T^7 + T^6 + 2*T^5 + T^4 + 3*T^3 + 1, 5*\
T^7 + 3*T^6 + 2*T^5 + 2*T^4 + 4*T^3 + T^2 + 3*T + 6, 6*T^5 + 2*T^4 + 4*T^2 + 2\
*T + 1 ] / (T^10 + 4*T^9 + 6*T^8 + 4*T^7 + T^6) ], 8 ] ] ]
kash> D - ( L[1] + L[2]*A -
> AlffDivisor(Product(L[4], x->Product(x[1])^x[2])) );
<
[1 0 0]
[0 1 0]
[0 0 1]
/ 1
>, <
[1 0 0]
[0 1 0]
[0 0 1]
/ (1)
> ]

```


NAME AlffDivisorSupp

PURPOSE Returns the support of a divisor as a list of places.

SYNTAX L := AlffDivisorSupp(D);

list L of places
alff divisor D

SEE ALSO [AlffDivisorPlaces](#),

EXAMPLE

```
kash> AlffInit(FF(5,1));;
kash> AlffOrders(y^2+T^3+1);;
kash> infty := AlffPlaceSplit(F, 1/T)[1];
Alff place < [ 1/T, 0 ], [ 0, 1 ] >
kash> l := AlffPlaceSplit(F, T+3);
[ Alff place < [ T + 3, 0 ], [ 1, 1 ] >,
  Alff place < [ T + 3, 0 ], [ 4, 1 ] > ]
kash> D := 2*infty + 3*l[1] - l[2];
Alff divisor
[ [ Alff place < [ 1/T, 0 ], [ 0, 1 ] >, 2 ],
  [ Alff place < [ T + 3, 0 ], [ 1, 1 ] >, 3 ],
  [ Alff place < [ T + 3, 0 ], [ 4, 1 ] >, -1 ] ]

kash> AlffDivisorSupp(D);
[ Alff place < [ 1/T, 0 ], [ 0, 1 ] >, Alff place < [ T + 3, 0 ], [ 1, 1 ] >,
  Alff place < [ T + 3, 0 ], [ 4, 1 ] > ]
```

NAME	AlffDivisorsSmoothNum
PURPOSE	Compute the number of (n, m) -smooth divisors.
SYNTAX	<p><code>N := AlffDivisorsSmoothNum(n, m, P);</code></p> <p>integer N number of (n, m)-smooth divisors</p> <p>integers n, m</p> <p>list P of integers</p>
DESCRIPTION	<p>Given two integers $n, m \geq 0$ and a list P such that $P[i]$ is the number of places of degree $1 \leq i \leq \min\{n, m\}$ of a given global function field, return the number of divisors D which are (n, m)-smooth, i.e. $D \geq 0$ and the support of D consists of places of degree $\leq m$. A recursion from [Heß99] is used.</p>
SEE ALSO	AlffInit , AlffOrders , AlffPlacesNum ,
EXAMPLE	

```

kash> AlffInit(FF(5, 2));;
kash> AlffOrders(y^3+T^3+2);;
kash> P := List([1..3], i->AlffPlacesNum(F, i));
[ 36, 270, 5280 ]
kash> AlffDivisorsSmoothNum(7, 3, P);
2885395249
kash> Zx1 := PolyAlg(Z, "x1");;
kash> Zx1x2 := PolyAlg(Zx1, "x2");;
kash> Zx1x2x3 := PolyAlg(Zx1x2, "x3");;
kash> x1 := Poly(Zx1, [1,0]);;
kash> x2 := Poly(Zx1x2, [1,0]);;
kash> x3 := Poly(Zx1x2x3, [1,0]);;
kash> N := AlffDivisorsSmoothNum(7, 3, [ x1, x2, x3 ]);
(1/2*x1 + 1/2)*x3^2 + (1/2*x2^2 + (1/2*x1^2 + 3/2*x1 + 3/2)*x2 + (1/24*x1^4 + \
5/12*x1^3 + 35/24*x1^2 + 31/12*x1 + 3/2))*x3 + (1/6*x1 + 1/6)*x2^3 + (1/12*x1^ \
3 + 1/2*x1^2 + 17/12*x1 + 1)*x2^2 + (1/120*x1^5 + 1/8*x1^4 + 19/24*x1^3 + 19/8 \
*x1^2 + 53/15*x1 + 11/6)*x2 + 1/5040*x1^7 + 1/180*x1^6 + 23/360*x1^5 + 7/18*x1 \
^4 + 967/720*x1^3 + 469/180*x1^2 + 363/140*x1 + 1
kash> Eval(Eval(Eval(N, P[3]), P[2]), P[1]);
2885395249

```

NAME	AlffDivisorsSmoothRatio
PURPOSE	Return a ratio of numbers of smooth divisors times a number of places.
SYNTAX	<pre>r := AlffDivisorsSmoothRatio(n, m, P);</pre> <p> real r smoothness ratio integers n, m list P of integers </p>
DESCRIPTION	<p>Given two integers $n, m \geq 0$ and a list P such that $P[i]$ is the number of places of degree $1 \leq i \leq \max\{n, m\}$ of a given global function field, return the ratio of the number of (n, m)-smooth divisors and the number of (n, n)-smooth divisors times the number of places of degree $\leq m$.</p>
SEE ALSO	AlffDivisorsSmoothNum , AlffInit , AlffOrders , AlffPlacesNum ,
EXAMPLE	

```

kash> AlffInit(FF(5, 2));;
kash> AlffOrders(y^3+T^3+2);;
kash> P := List([1..3], i->AlffPlacesNum(F, i));
[ 36, 270, 5280 ]
kash> P := Concatenation(P, [ (1/4)*5^8, (1/5)*5^10, (1/6)*5^12,
> (1/7)*5^14 ] );
[ 36, 270, 5280, 390625/4, 1953125, 244140625/6, 6103515625/7 ]
kash> AlffDivisorsSmoothRatio(7, 3, P);
18482.46533437107631419684229195180185174000056032

```


NAME	AlffEllipticFunField
PURPOSE	Creates an elliptic algebraic function field F/k .
SYNTAX	$F := \text{AlffEllipticFunField}(k, L);$ <div> algebraic function field F field k list L </div>
DESCRIPTION	<p>Let k be a field. The function creates the algebraic elliptic function field F. The field is defined by the polynomial $y^2 + a_1Ty + a_3y - (T^3 + a_2T^2 + a_4T + a_6)$ with $a_i \in k$. In characteristic 2,3 the user should be careful</p>
EXAMPLE	<pre> kash> F := AlffEllipticFunField(FF(2,16), [0,1,1,1,1]); Algebraic function field defined by \$.1^2 + \$.1 + \$.2^3 + \$.2^2 + \$.2 + 1 over Univariate rational function field over GF(2^16) Variables: T </pre>

NAME	AlffElt										
PURPOSE	Creates an element of an algebraic function field $F/k(T)$.										
SYNTAX	<pre> a := AlffElt(o, s); a := AlffElt(o, b); a := AlffElt(o, L); </pre> <table> <tr> <td>algebraic function field element</td><td>a</td></tr> <tr> <td>algebraic function field order</td><td>o</td></tr> <tr> <td>finite field element, rational or order element</td><td>s a constant</td></tr> <tr> <td>polynomial or quotient field element</td><td>b a rational function</td></tr> <tr> <td>list</td><td>L of above s or b of length n</td></tr> </table>	algebraic function field element	a	algebraic function field order	o	finite field element, rational or order element	s a constant	polynomial or quotient field element	b a rational function	list	L of above s or b of length n
algebraic function field element	a										
algebraic function field order	o										
finite field element, rational or order element	s a constant										
polynomial or quotient field element	b a rational function										
list	L of above s or b of length n										
DESCRIPTION	Let o be an alff order in F of rank n with basis b_1, \dots, b_n . Invoked with s or b these arguments will be embedded into $k(T) \subseteq F$. Invoked with a list $L := [c_1, \dots, c_n]$ of length n the element $\sum_{i=1}^n c_i b_i$ will be returned. The c_i have to be of the same type as s or b .										
SEE ALSO	Alff , AlffOrderEqFinite , AlffOrderEqInfty , AlffOrderMaxFinite , AlffOrderMaxInfty ,										

EXAMPLE Define an element of the finite maximal order:

```

kash> AlffInit(FF(5,2));
"Defining global variables: k, w, kT, kTf, kTy, T, y, AlffGlobals"
kash> AlffOrders(y^3+T^4+1);
"Defining global variables: F, o, oi, one"
kash> a := AlffElt(o, [0, 1, 0]);
[ 0, 1, 0 ]
kash> a^3+T^4+1;
[ 0, 0, 0 ]

```

NAME	AlffEltAlff
PURPOSE	Returns the algebraic function field of an alff element.
SYNTAX	<pre>F := AlffEltAlff(a);</pre> <div style="margin-left: 100px;"> <pre>alff F alff element a</pre> </div>
SEE ALSO	AlffEltOrder ,
EXAMPLE	

```
kash> AlffInit(FF(2,1));
"Defining global variables: k, w, kT, kTf, kTy, T, y, AlffGlobals"
kash> AlffOrders(y^3+T^3*y+T);
"Defining global variables: F, o, oi, one"
kash> a := AlffElt(o, [0,1,0] / (T+1));
[ 0, 1, 0 ] / (T + 1)
kash> AlffEltAlff(a);
Algebraic function field defined by
$.1^3 + $.1*$.2^3 + $.2
over
Univariate rational function field over GF(2)
Variables: T
```

NAME	AlffEltApprox												
PURPOSE	Computes an approximating element of an algebraic function field F .												
SYNTAX	<pre>alpha := AlffEltApprox(S, Lambda, A [,a]);</pre> <table><tr><td>element</td><td>alpha</td><td>of the algebraic function field F</td></tr><tr><td>list</td><td>S</td><td>of places of F</td></tr><tr><td>alff divisor</td><td>A</td><td></td></tr><tr><td>list</td><td>a</td><td>of elements of F</td></tr></table>	element	alpha	of the algebraic function field F	list	S	of places of F	alff divisor	A		list	a	of elements of F
element	alpha	of the algebraic function field F											
list	S	of places of F											
alff divisor	A												
list	a	of elements of F											
DESCRIPTION	<p>Let F be an algebraic function field, $S = \{P_1, \dots, P_r\}$ a set of r places of F (r a positive integer), $\Lambda = \{n_1, \dots, n_r\}$ a list of r integers and $a = \{a_1, \dots, a_r\}$ a list of r elements of F (if a is not given as a parameter, then a_i is assumed be a zero for all i). Let A be a divisor of F whose support $Supp(A)$ is disjoint to S and whose degree is positive. Then this function computes an element α of F with</p> <p>(1) $v_{P_i}(\alpha - a[i]) = n_i$ ($i = 1, \dots, r$) and</p> <p>(2) $v_P(\alpha) \geq 0$ for all $P \in F$, $P \notin (S \cup Supp(A))$.</p>												

EXAMPLE

```

kash> k := FF(3,2);
Finite field of size 3^2
kash> AlffInit(k);
"Defining global variables: k, w, kT, kTf, kTy, T, y, AlffGlobals"
kash> f := y^3-(T+1)*y^2+2*y*T-T^5;
y^3 + (2*T + 2)*y^2 + 2*T*y + 2*T^5
kash> F:=Alff(f);
Algebraic function field defined by
$.1^3 + 2*$.1^2*$.2 + 2*$.1^2 + 2*$.1*$.2 + 2*$.2^5
over
Univariate rational function field over GF(3^2)
Variables: T

kash> AlffOrders(f);
"Defining global variables: F, o, oi, one"
kash> S := [AlffPlaces(F,1)[1],AlffPlaces(F,3)[3],AlffPlaces(F,2)[5]];
[ Alff place < [ 1/T, 0, 0 ], [ 0, 1, 0 ] >, Alff place < [ T + w^5, 0, 0 ] >,
  Alff place < [ T^2 + w*T + 2, 0, 0 ], [ w*T + w^3, 1, 0 ] > ]
kash> Lambda := [-2,3,0];
[ -2, 3, 0 ]

```

```

kash> D := Sum([AlffPlaces(F,3)[2],AlffPlaces(F,3)[4]]);
Alff divisor
[ [ Alff place < [ T + w^3, 0, 0 ] >, 1 ],
  [ Alff place < [ T + w^7, 0, 0 ] >, 1 ] ]

kash> Alpha := AlffEltApprox(S, Lambda, D);
[ w^6*T^12 + 2*T^11 + w^5*T^10 + T^9 + w*T^7 + T^6 + w^2*T^4 + w*T^3 + w*T^2 + \
  w^7*T + w^3, T^11 + w^3*T^9 + w^2*T^8 + T^7 + T^6 + w*T^4 + w^3*T^3 + w^6*T^2 \
  + w^7*T + w^5, 2*T^10 + T^9 + w^3*T^8 + T^7 + T^6 + w*T^5 + w^5*T^4 + w^2*T^3 \
  + w^3*T^2 + w^5*T + w^2 ] / (T^12 + w^2*T^6 + 2)

kash> a := [RandomElt(o,3,3),RandomElt(o,3,3),RandomElt(o,3,3)];
[ [ T^3 + w^6*T + w^7, T^3 + w^3*T + w^3, T^3 + w^6*T^2 + 1 ],
  [ T^3 + w^6*T + w, T^3 + w^6*T^2 + 2*T + w^7, T^3 + 2*T^2 + w^2*T ],
  [ T^3 + T^2 + w^3*T + w^6, T^3 + w^3*T^2 + 2*T + w^7, T^3 + 2*T^2 + w^7*T + \
  2 ] ]

kash> Beta := AlffEltApprox(S, Lambda, D, a);
[ T^23 + w^5*T^20 + T^19 + w^2*T^18 + w^6*T^17 + 2*T^16 + w^6*T^15 + w^7*T^14 \
  + w^2*T^12 + w^6*T^10 + w^6*T^9 + 2*T^8 + w^3*T^7 + w^7*T^6 + T^5 + w^2*T^4 + \
  w^5*T^3 + w*T^2 + w^2*T + 2, T^23 + 2*T^21 + w^3*T^20 + w^3*T^19 + w^3*T^18 + \
  w^2*T^17 + w^6*T^16 + w*T^14 + 2*T^12 + w*T^10 + w*T^9 + w^6*T^8 + w*T^7 + 2*T \
  ^6 + T^4 + w^6*T^3 + w*T^2 + w^5*T + w^5, T^23 + w^6*T^22 + w^2*T^21 + 2*T^20 \
  + w*T^18 + w^5*T^17 + w*T^16 + T^15 + w*T^14 + w^3*T^13 + w^7*T^12 + w*T^11 + \
  w^6*T^10 + w^3*T^9 + w^2*T^8 + w^3*T^7 + w^7*T^6 + w*T^4 + w^5*T^3 + 2*T^2 + w \
  ^7*T + w^6 ] / (T^20 + w^2*T^18 + w^2*T^2 + 2)

```

NAME	AlffEltBstar
PURPOSE	Computes $B^*(a)$ for a global function field element a .
SYNTAX	<pre>q := AlffEltBstar(a);</pre> <div style="display: flex; justify-content: space-between; margin-top: 10px;"> <div style="text-align: left;"> <p>rational</p> <p>global function field element</p> </div> <div style="text-align: right;"> <p>q</p> <p>a</p> </div> </div>
DESCRIPTION	<p>Let v_i denote the s distinct normalized valuations at the s places of F over the infinite place of $k(T)$, with ramification indices e_i. Let $e = \text{lcm}\{e_1, \dots, e_s\}$ and define</p> $B^* : F \longrightarrow \{a/e \mid a \in \mathbb{Z}\} \cup \{-\infty\} : \alpha \longmapsto -\min_{i=1}^s v_i(\alpha)/e_i.$ <p>This function computes B^* for elements of a global function field F. See [Sch96] for further information.</p>
EXAMPLE	<pre>kash> AlffInit(FF(5,2)); "Defining global variables: k, w, kT, kTf, kTy, T, y, AlffGlobals" kash> AlffOrders(y^3+T^4+1); "Defining global variables: F, o, oi, one" kash> a := AlffElt(o, [0, 1, 1]); [0, 1, 1] kash> AlffEltBstar(a); 8/3</pre>

NAME	AlffEltCharPoly
PURPOSE	Returns the characteristic polynomial of an algebraic function field element.
SYNTAX	<pre>p := AlffEltMinPoly(a);</pre> <div> polynomial p alff element a </div>
DESCRIPTION	Given an alff element a in F this function computes the characteristic polynomial of a over $k(T)$.
SEE ALSO	AlffEltCharPoly ,
EXAMPLE	

```
kash> AlffInit(FF(2,1));
"Defining global variables: k, w, kT, kTf, kTy, T, y, AlffGlobals"
kash> AlffOrders(y^3+T^3*y+T);
"Defining global variables: F, o, oi, one"
kash> a := AlffElt(o, [0,1,0]);
[ 0, 1, 0 ]
kash> p := AlffEltCharPoly(a);
y^3 + T^3*y + T
```

NAME	AlffEltDen
PURPOSE	Returns the denominator of an algebraic function field element.
SYNTAX	$d := \text{AlffEltDen}(a);$ <div style="display: flex; justify-content: space-between; margin-top: 10px;"> <div> quotient field element or polynomial algebraic function field element </div> <div> d a </div> </div>
DESCRIPTION	<p>Let a be defined with respect to the R-order o in $F/k(T)$ of rank n with basis b_1, \dots, b_n, R being $k[T]$ or the valuation ring of the degree valuation of $k(T)$. For $a = \sum_{i=1}^n c_i b_i / d$, $c_i, d \in R$ coprime, $(1 \leq i \leq n)$, the function returns d.</p>
SEE ALSO	AlffEltNum , AlffEltToList , Num , Den ,
EXAMPLE	

```

kash> AlffInit(FF(5,2));
"Defining global variables: k, w, kT, kTf, kTy, T, y, AlffGlobals"
kash> AlffOrders(y^3+T^4+1);
"Defining global variables: F, o, oi, one"
kash> a := AlffElt(o, [0, 1, 0]);
[ 0, 1, 0 ]
kash> 1/a;
[ 0, 0, 4 ] / (T^4 + 1)
kash> AlffEltDen(1/a);
T^4 + 1

```


NAME	AlffEltEval
PURPOSE	Evaluates an algebraic function at a place.
SYNTAX	<pre>b := AlffEltEval(P, a);</pre> <p>field element b value of a at P alff place P alff element a</p>
DESCRIPTION	Let F/k be an algebraic function field and P be a place of F/k . If \mathfrak{o}_P is the ring of algebraic functions defined at P then $k(P) := \mathfrak{o}_P/P$ is the residue class field of P . The image $a(P)$ of an algebraic function $a \in F$ in $k(P)$ or false for $a \notin \mathfrak{o}_P$ is returned.
SEE ALSO	AlffResidueField , AlffEltLift , AlffEltValuation ,
EXAMPLE	

```
kash> AlffInit(Q);
"Defining global variables: k, w, kT, kTf, kTy, T, y, AlffGlobals"
kash> AlffOrders(y^2 - T^4 + T + 1);
"Defining global variables: F, o, oi, one"
kash> P := AlffPlaceSplit(F, T+2)[1];
Alff place < [ T + 2, 0 ] >
kash> K := AlffResidueField(P);
Generating polynomial: x^2 - 17

kash> a := AlffEltGenT(F);
[ T, 0 ]
kash> b := AlffEltEval(P, a);
-2
```

NAME	AlffEltGenT				
PURPOSE	Returns the indeterminate T as maximal order element.				
SYNTAX	<p><code>T := AlffEltGenT(F);</code></p> <table> <tr> <td>algebraic function field</td><td>F</td></tr> <tr> <td>alff element</td><td>T</td></tr> </table>	algebraic function field	F	alff element	T
algebraic function field	F				
alff element	T				
DESCRIPTION	Returns the image of the indeterminate T with respect to the generating equation of the algebraic function field in it's finite maximal order.				
SEE ALSO	AlffEltGenY , Alff , AlffOrderMaxFinite ,				
EXAMPLE					

```

kash> AlffInit(FF(7,2));
"Defining global variables: k, w, kT, kTf, kTy, T, y, AlffGlobals"
kash> F := Alff((T^2+1)*y^3+y+T^4+1);
Algebraic function field defined by
$.1^3*$.2^2 + $.1^3 + $.1 + $.2^4 + 1
over
Univariate rational function field over GF(7^2)
Variables: T

kash> T := AlffEltGenT(F);
[ T, 0, 0 ]
kash> Y := AlffEltGenY(F);
[ 0, 1, 0 ] / (T^2 + 1)
kash> (T^2+1)*Y^3+Y+T^4+1;
[ 0, 0, 0 ]

```

NAME	AlffEltGenY
PURPOSE	Returns the indeterminate y as finite maximal order element.
SYNTAX	$Y := \text{AlffEltGenY}(F);$ <div style="display: flex; justify-content: space-between; margin-top: 10px;"> <div>algebraic function field</div> <div>F</div> </div> <div style="display: flex; justify-content: space-between; margin-top: 10px;"> <div>alff element</div> <div>Y</div> </div>
DESCRIPTION	Returns the image of the indeterminate y with respect to the generating equation of the algebraic function field in it's finite maximal order.
SEE ALSO	AlffEltGenT , Alff , AlffOrderMaxFinite ,
EXAMPLE	

```

kash> AlffInit(FF(7,2));
"Defining global variables: k, w, kT, kTf, kTy, T, y, AlffGlobals"
kash> F := Alff((T^2+1)*y^3+y+T^4+1);
Algebraic function field defined by
$.1^3*$.2^2 + $.1^3 + $.1 + $.2^4 + 1
over
Univariate rational function field over GF(7^2)
Variables: T

kash> T := AlffEltGenT(F);
[ T, 0, 0 ]
kash> Y := AlffEltGenY(F);
[ 0, 1, 0 ] / (T^2 + 1)
kash> (T^2+1)*Y^3+Y+T^4+1;
[ 0, 0, 0 ]

```

NAME	AlffEltInftyVals
PURPOSE	Computes the inequivalent valuations of a global function field element at infinity.
SYNTAX	$L := \text{AlffEltInftyVals}(a);$ <div style="margin-left: 100px;"> $\text{list} \qquad \qquad \qquad L$ $\text{global function field element} \quad a$ </div>
DESCRIPTION	Denote by v_i the s inequivalent normalized valuations extending the degree valuation of $k(T)$ to F . The function returns $L := [v_1(a), \dots, v_s(a)]$ in the tame ramified case.
SEE ALSO	AlffPlaceSplit , AlffSignature , Alff ,
EXAMPLE	

```

kash> AlffInit(F(5,2));
"Defining global variables: k, w, kT, kTf, kTy, T, y, AlffGlobals"
kash> AlffOrders(y^3+T^4+1);
"Defining global variables: F, o, oi, one"
kash> a := AlffElt(o, [0, 1, 0]);
[ 0, 1, 0 ]
kash> AlffEltInftyVals(a);
[ -4 ]

```

NAME	AlffEltIsInIdeal
PURPOSE	Checks whether a function field element is in an ideal.
SYNTAX	<pre>b := AlffEltIsInIdeal(a, I);</pre> <div> <div>boolean</div> <div>alff order element</div> <div>alff order ideal</div> </div> <div> <div>b</div> <div>a</div> <div>I</div> </div>

EXAMPLE Some computations:

```
kash> AlffInit(F(5,2));
"Defining global variables: k, w, kT, kTf, kTy, T, y, AlffGlobals"
kash> AlffOrders(y^3+T^4+1);
"Defining global variables: F, o, oi, one"
kash> I := T*o;
< [ T, 0, 0 ] >
kash> AlffEltIsInIdeal(T, I);
true
kash> AlffEltIsInIdeal(1/T, I);
false
kash> a := AlffElt(o, [0, 1, 0]);
[ 0, 1, 0 ]
kash> I := a*o;
< [ 0, 1, 0 ] >
kash> AlffEltIsInIdeal(T+w, I);
false
```

NAME	AlffEltMin
PURPOSE	Returns the denominator d of an alff element a from a form of the representation matrix of $da \in \mathfrak{o}$.
SYNTAX	<pre>m := AlffEltMin(a);</pre> <p>coefficient ring element m alff element a</p>
DESCRIPTION	If d is the denominator of an alff element a of the order \mathfrak{o} this function returns the upper left corner element of the upper column Hermite normal form of the representation matrix of $da \in \mathfrak{o}$ on the basis of \mathfrak{o} .
SEE ALSO	AlffEltRepMat ,
EXAMPLE	

```
kash> AlffInit(FF(2,1));
"Defining global variables: k, w, kT, kTf, kTy, T, y, AlffGlobals"
kash> AlffOrders(y^3+T^3*y+T);
"Defining global variables: F, o, oi, one"
kash> a := AlffElt(o, [0,1,0] / T);
[ 0, 1, 0 ] / T
kash> AlffEltMin(a);
T
```

NAME	AlffEltMinPoly
PURPOSE	Returns the minimal polynomial of an algebraic function field element.
SYNTAX	<pre>p := AlffEltMinPoly(a);</pre> <p>polynomial p alff element a</p>
DESCRIPTION	Given an alff element a of F this function computes the minimal polynomial of a over $k(T)$.
SEE ALSO	AlffEltCharPoly ,
EXAMPLE	

```
kash> AlffInit(FF(2,1));
"Defining global variables: k, w, kT, kTf, kTy, T, y, AlffGlobals"
kash> AlffOrders(y^3+T^3*y+T);
"Defining global variables: F, o, oi, one"
kash> a := AlffElt(o, [0,1,0]);
[ 0, 1, 0 ]
kash> p := AlffEltMinPoly(a);
y^3 + T^3*y + T
```

NAME	AlffEltMove
PURPOSE	Moves an algebraic function field element between orders.
SYNTAX	<pre>a := AlffEltMove(b, o);</pre> <p>algebraic function field elements a,b algebraic function field order o</p>
DESCRIPTION	For given $a \in F$ this function returns $b \in F$ with $a = b$ mathematically but b is represented in the basis of o . Thus <code>AlffEltMove()</code> is a change of basis.
EXAMPLE	Some transfers:

```
kash> AlffInit(FF(5,2));
"Defining global variables: k, w, kT, kTf, kTy, T, y, AlffGlobals"
kash> AlffOrders(y^3+T^4+1);
"Defining global variables: F, o, oi, one"
kash> a := AlffElt(o, [0,1,0]);
[ 0, 1, 0 ]
kash> b := AlffEltMove(a, oi);
[ 0, 1, 0 ] / 1/T^2
kash> AlffEltMove(b, o);
[ 0, 1, 0 ]
```


NAME	AlffEltNewtonLift														
PURPOSE	Lifts an algebraic element with the Newton lifting method.														
SYNTAX	<pre>beta:=AlffEltNewtonLift(o, g, alpha, p, k, den);</pre> <table> <tr> <td>algebraic function field element</td><td>beta</td></tr> <tr> <td>algebraic function field order</td><td>o</td></tr> <tr> <td>polynomial</td><td>g</td></tr> <tr> <td>algebraic function field element</td><td>alpha</td></tr> <tr> <td>polynomial</td><td>p</td></tr> <tr> <td>integer</td><td>k</td></tr> <tr> <td>polynomial</td><td>den</td></tr> </table>	algebraic function field element	beta	algebraic function field order	o	polynomial	g	algebraic function field element	alpha	polynomial	p	integer	k	polynomial	den
algebraic function field element	beta														
algebraic function field order	o														
polynomial	g														
algebraic function field element	alpha														
polynomial	p														
integer	k														
polynomial	den														
DESCRIPTION	Given a function field element α with $\mathfrak{g}\alpha \equiv 0 \pmod{\mathfrak{p}}$, this function calculates an algebraic element β with $\mathfrak{g}(\beta) \equiv 0 \pmod{\mathfrak{p}^{2^k}\mathfrak{o}}$.														

NAME	AlffEltNorm
PURPOSE	Computes the norm of an element of an algebraic function field.
SYNTAX	<pre>a := AlffEltNorm(b);</pre> <p> quotient field element or polynomial a algebraic function field element b </p>
DESCRIPTION	Given $a \in F$ this function returns the norm of a over $k(T)$.
SEE ALSO	AlffTrace , AlffEltCharPoly , Norm , Trace ,
EXAMPLE	

```

kash> AlffInit(FF(5,2));
"Defining global variables: k, w, kT, kTf, kTy, T, y, AlffGlobals"
kash> AlffOrders(y^3+T^4+1);
"Defining global variables: F, o, oi, one"
kash> b := AlffElt(o, [0, 1, 0]);
[ 0, 1, 0 ]
kash> AlffEltNorm(b);
4*T^4 + 4

```

NAME	AlffEltNum
PURPOSE	Returns the numerator of an algebraic function field element.
SYNTAX	<pre>b := AlffEltNum(a);</pre> <p>algebraic function field element a algebraic function field element b</p>
DESCRIPTION	<p>Let a be defined with respect to the R-order o in $F/k(T)$ of rank n with basis b_1, \dots, b_n, R being $k[T]$ or the valuation ring of the degree valuation of $k(T)$. For $a = \sum_{i=1}^n c_i b_i / d$, $c_i, d \in R$ coprime, $(1 \leq i \leq n)$, the function returns $\sum_{i=1}^n c_i b_i / d$.</p>
SEE ALSO	AlffEltDen , AlffEltToList , Num , Den ,
EXAMPLE	

```
kash> AlffInit(FF(5,2));
"Defining global variables: k, w, kT, kTf, kTy, T, y, AlffGlobals"
kash> AlffOrders(y^3+T^4+1);
"Defining global variables: F, o, oi, one"
kash> a := AlffElt(o, [0, 1, 0]);
[ 0, 1, 0 ]
kash> 1/a;
[ 0, 0, 4 ] / (T^4 + 1)
kash> AlffEltNum(1/a);
[ 0, 0, 4 ]
```

NAME	AlffEltOrder
PURPOSE	Returns the order with respect to which the element has been defined.
SYNTAX	<pre>o := AlffEltOrder(a);</pre> <p>algebraic function field order o</p> <p>algebraic function field element a</p>
SEE ALSO	AlffElt , AlffEltMove , AlffOrderAlff ,

EXAMPLE

```
kash> AlffInit(FF(5,2));
"Defining global variables: k, w, kT, kTf, kTy, T, y, AlffGlobals"
kash> AlffOrders(y^3+T^4+1);
"Defining global variables: F, o, oi, one"
kash> a := AlffElt(o, [0, 1, 0]);
[ 0, 1, 0 ]
kash> AlffEltOrder(a);
Finite maximal order of
Algebraic function field defined by
$.1^3 + $.2^4 + 1
over
Univariate rational function field over GF(5^2)
Variables: T
```

NAME	AlffEltPthRoot
PURPOSE	Computes the p^r th root of an element of a global function field.
SYNTAX	<pre>b := AlffEltPthRoot(a, r);</pre> <p> alff element b global function field F positive integer r </p>
DESCRIPTION	Let F/k be a global function field of characteristic p . This function computes the unique p^r th root b of an alff element a in F , or returns false if it does not exist.
SEE ALSO	AlffDivisor , AlffDivisorLDim ,

EXAMPLE

```

kash> AlffInit(F(5,1));
kash> AlffOrders(y^4 + T^7 + 1);
"Defining global variables: F, o, oi, one"
kash> c := AlffEltGenY(F);
[ 0, 1, 0, 0 ]
kash> a := c^25;
[ 0, T^42 + T^35 + T^7 + 1, 0, 0 ]
kash> AlffEltPthRoot(a, 1);
[ 0, 4*T^7 + 4, 0, 0 ]
kash> AlffEltPthRoot(a, 2);
[ 0, 1, 0, 0 ]
kash> AlffEltPthRoot(a, 3);
false

```

NAME	AlffEltRepMat
PURPOSE	Returns a representation matrix of an algebraic function field element.
SYNTAX	$M := \text{AlffEltRepMat}(a);$ <div style="margin-left: 100px;"> matrix M alff element a </div>
DESCRIPTION	Given an alff element a represented in its order \mathfrak{o} this function computes the representation matrix of a on the basis of \mathfrak{o} .
SEE ALSO	AlffEltCharPoly ,
EXAMPLE	

```

kash> AlffInit(F(2,1));
"Defining global variables: k, w, kT, kTf, kTy, T, y, AlffGlobals"
kash> AlffOrders(y^3+T^3*y+T);
"Defining global variables: F, o, oi, one"
kash> a := AlffElt(o, [0,1,0] / (T+1));
[ 0, 1, 0 ] / (T + 1)
kash> AlffEltRepMat(a);
[      0      0  T/(T + 1)]
[ 1/(T + 1)      0 T^3/(T + 1)]
[      0  1/(T + 1)      0]

```

NAME	AlffEltResiduum
PURPOSE	Computes the residuum of an algebraic function.
SYNTAX	<pre>r := AlffEltResiduum(P, t, a);</pre> <p> field element r the residuum of a at P for t alff place P alff element t a local paramter at P alff element a an algebraic function </p>
DESCRIPTION	Let F/k be an algebraic function field and P be a place of F/k . It is currently required that $\deg(P) = 1$ holds. For a local parameter $t \in F$ at P this function computes the residuum of an algebraic function $a \in F$.
SEE ALSO	AlffPlacePrimeElt , AlffResidueField , AlffEltEval , AlffEltLift , AlffEltValuation , AlffDiffResiduum ,
EXAMPLE	

```
kash> AlffInit(Q);
"Defining global variables: k, w, kT, kTf, kTy, T, y, AlffGlobals"
kash> AlffOrders(y^2 - T^4 + T + 1);
"Defining global variables: F, o, oi, one"
kash> P := AlffPlaceSplit(F, T+1)[1];
Alff place < [ T + 1, 0 ], [ -1, 1 ] >
kash> t := AlffPlacePrimeElt(P);
[ T + 1, 0 ]
kash> AlffEltResiduum(P, t, 1/t^2 + 3 + t);
0
kash> AlffEltResiduum(P, t, 1/t^2 + 2/t + 3 + t);
2
```

NAME	AlffEltRoot
PURPOSE	Computes an n -th root of a function field element a .
SYNTAX	<pre>b := AlffEltRoot(a,n);</pre> <p> alff element b alff element a integer n </p>
SEE ALSO	AlffEltPthRoot ,
EXAMPLE	

```
kash> AlffInit(FF(2,1));
"Defining global variables: k, w, kT, kTf, kTy, T, y, AlffGlobals"
kash> AlffOrders(y^3+T^3*y+T);
"Defining global variables: F, o, oi, one"
kash> a := AlffElt(o, [0,1,0] / T);
[ 0, 1, 0 ] / T
kash> AlffEltRoot(a,2);
false
```


NAME	AlffEltToList
PURPOSE	Returns the coefficients and the denominator of an algebraic function field element.
SYNTAX	$L := \text{AlffEltToList}(a);$ <div style="display: flex; justify-content: space-between; margin-top: 10px;"> <div>list</div> <div>L</div> </div> <div style="display: flex; justify-content: space-between; margin-top: 10px;"> <div>algebraic function field element</div> <div>a</div> </div>
DESCRIPTION	<p>Let a be defined with respect to the R-order o in $F/k(T)$ of rank n with basis b_1, \dots, b_n, R being $k[T]$ or the valuation ring of the degree valuation of $k(T)$. For $a = \sum_{i=1}^n c_i b_i / d$, $c_i, d \in R$ coprime, $(1 \leq i \leq n)$, the function returns $L := [[c_1, \dots, c_n], d]$.</p>
SEE ALSO	AlffEltNum , AlffEltDen , AlffElt , Num , Den ,

EXAMPLE Decompose an element into a list:

```
kash> AlffInit(FF(5,2));
"Defining global variables: k, w, kT, kTf, kTy, T, y, AlffGlobals"
kash> AlffOrders(y^3+T^4+1);
"Defining global variables: F, o, oi, one"
kash> a := AlffElt(o, [0, 1, 0]) / T^7;
[ 0, 1, 0 ] / T^7
kash> AlffEltToList(a);
[ [ 0, 1, 0 ], T^7 ]
```

NAME	AlffEltToResField
PURPOSE	Returns a representative of the class of an algebraic function in the residue class field of a place.
SYNTAX	$b := \text{AlffEltToResField}(a, P);$ <p> field element b value of a at P alff element a alff place P </p>
DESCRIPTION	Let F/k be an algebraic function field and P be a place of F/k . If \mathfrak{o}_P is the ring of algebraic functions defined at P then $k(P) := \mathfrak{o}_P/P$ is the residue class field of P . The image $a(P)$ of an algebraic function $a \in F$ in $k(P)$ or false for $a \notin \mathfrak{o}_P$ is returned.
SEE ALSO	AlffResFieldEltLift , AlffEltToResField , AlffEltValuation ,

EXAMPLE

```

kash> AlffInit(FF(5,3));
"Defining global variables: k, w, kT, kTf, kTy, T, y, AlffGlobals"
kash> F := Alff(y^3+T^3*y+T);
Algebraic function field defined by
$.1^3 + $.1*$.2^3 + $.2
over
Univariate rational function field over GF(5^3)
Variables: T

kash> P := AlffPlaceSplit(F, T+1)[1];
Alff place < [ T + 1, 0, 0 ], [ 3, 1, 0 ] >
kash> AlffPlaceResField(P);
Finite field of size 5^3
kash> of := AlffOrderMaxFinite(F);
Finite maximal order of
Algebraic function field defined by
$.1^3 + $.1*$.2^3 + $.2
over
Univariate rational function field over GF(5^3)
Variables: T

kash> a := AlffEltToResField(AlffElt(of,T),P);

```

```

4
kash> AlffResFieldEltLift(a,P);
[ 4, 0, 0 ]
kash> p := AlffPlaceSplit(F, 1/T)[1];
Alff place < [ 1/T, 0, 0 ], [ w^48/T, (w^102*T + w^5)/T, (w^18*T + w^82)/T ] >
kash> AlffPlaceResField(p);
Finite field of size 5^3
kash> oi := AlffOrderMaxInfty(F);
Infinite maximal order of
Algebraic function field defined by
$.1^3 + $.1*$.2^3 + $.2
over
Univariate rational function field over GF(5^3)
Variables: T
given by transformation matrix
[1/T  0  0]
[ 0 1/T  0]
[ 0  0  1]
with denominator 1/T
kash> b := AlffEltToResField(AlffElt(oi,1/T),p);
0
kash> AlffResFieldEltLift(b,p);
[ 0, 0, 0 ]

kash> AlffInit(Q);
"Defining global variables: k, w, kT, kTf, kTy, T, y, AlffGlobals"
kash> F := Alff(y^3+T^3*y+T);
Algebraic function field defined by
$.1^3 + $.1*$.2^3 + $.2
over
Univariate rational function field over Rational Field
Variables: T

kash> P := AlffPlaceSplit(F, T+1)[1];
Alff place < [ T + 1, 0, 0 ] >
kash> AlffPlaceResField(P);
Generating polynomial: x^3 - x - 1

kash> of := AlffOrderMaxFinite(F);
Finite maximal order of
Algebraic function field defined by
$.1^3 + $.1*$.2^3 + $.2
over

```

Univariate rational function field over Rational Field

Variables: T

```
kash> a := AlffEltToResField(AlffElt(of,2),P);
```

2

```
kash> AlffResFieldEltLift(a,P);
```

[2, 0, 0]

```
kash> p := AlffPlaceSplit(F, 1/T)[1];
```

Alff place < [1/T, 0, 0], [-13/T, (11/3*T - 35)/T, (-3/2*T + 32/3)/T] >

```
kash> AlffPlaceResField(p);
```

Rational Field

```
kash> oi := AlffOrderMaxInfty(F);
```

Infinite maximal order of

Algebraic function field defined by

$\$.1^3 + \$.1*\$.2^3 + \$.2$

over

Univariate rational function field over Rational Field

Variables: T

given by transformation matrix

[1/T 0 0]

[0 1/T 0]

[0 0 1]

with denominator 1/T

```
kash> b := AlffEltToResField(AlffElt(oi,1/T),p);
```

0

```
kash> AlffResFieldEltLift(b,p);
```

[0, 0, 0]

NAME	AlffEltTrace
PURPOSE	Computes the trace of an element of an algebraic function field.
SYNTAX	<pre>a := AlffEltTrace(b);</pre> <p> quotient field element or polynomial a algebraic function field element b </p>
DESCRIPTION	Given $a \in F$ this function returns the trace of a over $k(T)$.
SEE ALSO	AlffEltNorm ,
EXAMPLE	

```
kash> AlffInit(FF(5,2));
"Defining global variables: k, w, kT, kTf, kTy, T, y, AlffGlobals"
kash> AlffOrders(y^3+T^4+1);
"Defining global variables: F, o, oi, one"
kash> b := AlffElt(o, [0, 1, 0]);
[ 0, 1, 0 ]
kash> AlffEltTrace(b);
0
```

NAME	AlffEltValuation
PURPOSE	The valuation of an algebraic function field element at a place.
SYNTAX	<pre>v := AlffEltValuation(P, a);</pre> <div> integer v alff place P alff order element a </div>
DESCRIPTION	This function returns the valuation $\nu_{\mathfrak{P}}(a)$ of an algebraic function field order element a at a given place \mathfrak{P} .
SEE ALSO	AlffPlaceSplit , AlffDivisor ,
EXAMPLE	Compute a valuation:

```
kash> AlffInit(FF(2,1));
"Defining global variables: k, w, kT, kTf, kTy, T, y, AlffGlobals"
kash> AlffOrders(y^3+T^3*y+T);
"Defining global variables: F, o, oi, one"
kash> a := AlffElt(o, [0, 1, 0]);
[ 0, 1, 0 ]
kash> P := AlffPlaceSplit(F, T)[1];
Alff place < [ T, 0, 0 ], [ 0, 1, 0 ] >
kash> AlffEltValuation(P, a);
1
kash> AlffEltValuation(P, a^(-2));
-2
```

NAME	AlffGapNumbers
PURPOSE	Returns the gap numbers of a divisor.
SYNTAX	$L := \text{AlffGapNumbers}(D [, P]);$ $L := \text{AlffGapNumbers}(F [, P]);$ <p> <code>list</code> <code>L</code> containing the gap numbers <code>alff divisor</code> <code>D</code> <code>alff</code> <code>F</code> equivalent to taking $D = 0$ </p>
DESCRIPTION	<p>Let F/k be an algebraic function field, D a divisor and P a place of degree one. An integer $m \geq 1$ is a D-gap number of P if $\dim(D + (m-1)P) = \dim(D + mP)$ holds. The D-gap numbers m satisfy $1 \leq m \leq 2g - 1 - \deg(D)$ and their cardinality equals the index of speciality $i(D)$. If P of degree one is given as argument this function returns the sequence of its D-gap numbers. The sequences of D-gap numbers are independent of constant field extensions for perfect k and are the same for all but a finite number of places P of degree one (consider e.g. k algebraically closed). If P is omitted in the function call, this uniform sequence is returned. The places P which have different sequences of D-gap numbers are called D-Weierstraß places. The constant field k is required to be exact.</p>
SEE ALSO	AlffWeierstrassPlaces , AlffWronskian , AlffWronskianOrders , AlffDiff , AlffDifferentiation ,
EXAMPLE	<pre> kash> AlffInit(Q); kash> AlffOrders(y^2 - (T-1)*(T-2)*(T-3)*(T-4)*(T-5)*(T-6)*(T-7)); "Defining global variables: F, o, oi, one" kash> P := AlffPlaceSplit(F, T-1)[1]; Alff place < [T - 1, 0], [0, 1] > kash> AlffGapNumbers(F); [1, 2, 3] kash> AlffGapNumbers(F, P); [1, 3, 5] </pre>

NAME	AlffGenus
PURPOSE	missing shortdoc
SYNTAX	<pre>g := AlffGenus(F);</pre> <p>algebraic function field F integer g the genus</p>
DESCRIPTION	Computes the genus of the algebraic function field.
SEE ALSO	AlffDimExactConstField , Alff , AlffDivisorLBasis ,
EXAMPLE	

```
kash> AlffInit(FF(5,2));
"Defining global variables: k, w, kT, kTf, kTy, T, y, AlffGlobals"
kash> AlffOrders(y^3+T^4+1);
"Defining global variables: F, o, oi, one"
kash> AlffGenus(F);
3
```


NAME	AlffHasseWittInvariant
PURPOSE	Computes the Hasse-Witt invariant of a global function field.
SYNTAX	<pre>s := AlffHasseWittInvariant(F);</pre> <p>integer s</p> <p>global function field F</p>
DESCRIPTION	<p>Let F/k be a global function field of characteristic p. Let $F\bar{k}/\bar{k}$ be the constant field extension by the algebraic closure \bar{k} of k within an algebraic closure \bar{F} of F. Consider the subgroup $Cl^0(F\bar{k}/\bar{k})[p]$ of p-torsion elements of the group of divisor classes of degree zero. This function returns its dimension as an \mathbb{F}_p-vector space. Possible values range from 0 to g, where g is the genus of F/k.</p>
SEE ALSO	AlffClassGroupPRank ,
EXAMPLE	

```
kash> AlffInit(FF(2, 1));
"Defining global variables: k, w, kT, kTf, kTy, T, y, AlffGlobals"
kash> F := Alff(T*y^3 + y + T^3);
Algebraic function field defined by
$.1^3*$.2 + $.1 + $.2^3
over
Univariate rational function field over GF(2)
Variables: T

kash> AlffHasseWittInvariant(F);
3
```

NAME	AlffHermitianFunField
PURPOSE	Creates a Hermitian global function field $F/\mathbb{F}_{q^2}(T)$.
SYNTAX	<pre>F := AlffHermitianFunField(p, d);</pre> <pre>global function field F integer p, d</pre>
DESCRIPTION	Creates a Hermitian global function field defined by the polynomial $y^q + y = T^{q+1} \in \mathbb{F}_{q^2}[T, y]$ with $q = p^d$.
SEE ALSO	AlffInit ,
EXAMPLE	The definition of a Hermitian global function field:

```
kash> F := AlffHermitianFunField(2, 3);
Algebraic function field defined by
$.1^8 + $.1 + $.2^9
over
Univariate rational function field over GF(2^6)
Variables: T
```

NAME	AlffHermitianFunField
PURPOSE	Creates a Hermitian global function field $F/\mathbb{F}_{q^2}(T)$.
SYNTAX	<pre>F := AlffHermitianFunField(p, d);</pre> <p>global function field F integer p, d</p>
DESCRIPTION	Creates a Hermitian global function field defined by the polynomial $y^q + y = T^{q+1} \in \mathbb{F}_{q^2}[T, y]$ with $q = p^d$.
SEE ALSO	AlffInit ,

EXAMPLE The definition of a Hermitian global function field:

```
kash> F := AlffHermitianFunField(2, 3);
Algebraic function field defined by
$.1^8 + $.1 + $.2^9
over
Univariate rational function field over GF(2^6)
Variables: T
```

NAME	AlffIdeal2EltAssure
PURPOSE	Compute a two element representation of an ideal.
SYNTAX	AlffIdeal2EltAssure(I); alff order ideal I
DESCRIPTION	This function tries to represent a given ideal as the sum of two principal ideals (currently only for global function fields).
SEE ALSO	AlffIdealBasisUpperHNF ,
EXAMPLE	

```

kash> AlffInit(FF(5,1));
"Defining global variables: k, w, kT, kTf, kTy, T, y, AlffGlobals"
kash> AlffOrders(y^3+(2*T+4)*y^2+(3*T^2+2*T+2)*y+T^7+T^6+2*T^5+3*T^4);
"Defining global variables: F, o, oi, one"
kash> a := AlffElt(o, [ 3, 0, 1 ]);
[ 3, 0, 1 ]
kash> I := (T+2)*o + a*o;
<
[ 1 0 0 ]
[ 0 T + 2 0 ]
[ 2 0 T + 2 ]
/ 1
>
kash> AlffIdeal2EltAssure(I);
kash> I;
< [ T + 2, 0, 0 ], [ 4*T + 1, 2*T + 4, 3*T + 2 ] >
kash> AlffIdeal2EltAssure(I);
kash> I;
< [ T + 2, 0, 0 ], [ 4*T + 1, 2*T + 4, 3*T + 2 ] >

```

NAME	AlffIdealAlff
PURPOSE	Returns the algebraic function field of an alff order ideal.
SYNTAX	<pre>F := AlffIdealAlff(I);</pre> <div style="margin-left: 100px;"> <pre>alff F</pre> <pre>alff order ideal I</pre> </div>
SEE ALSO	AlffIdealOrder ,
EXAMPLE	

```
kash> AlffInit(FF(2,1));
"Defining global variables: k, w, kT, kTf, kTy, T, y, AlffGlobals"
kash> AlffOrders(y^3+T^3*y+T);
"Defining global variables: F, o, oi, one"
kash> I := AlffElt(o, [0,1,0] / (T+1)) * o;
< [ 0, 1, 0 ] / (T + 1) >
kash> AlffIdealAlff(I);
Algebraic function field defined by
$.1^3 + $.1*$.2^3 + $.2
over
Univariate rational function field over GF(2)
Variables: T
```

NAME	AlffIdealBasis
PURPOSE	Returns a list containing the basis elements of an alff order ideal.
SYNTAX	<pre>B := AlffIdealBasis(I);</pre> <p>list B of basis elements alff order ideal I</p>

SEE ALSO [AlffOrderBasis](#), [AlffIdealBasisUpperHNF](#),

EXAMPLE

```
kash> AlffInit(FF(2,1));
"Defining global variables: k, w, kT, kTf, kTy, T, y, AlffGlobals"
kash> AlffOrders(y^3+T^3*y+T);
"Defining global variables: F, o, oi, one"
kash> I := AlffElt(o, [0,1,0] / (T+1)) * o;
< [ 0, 1, 0 ] / (T + 1) >
kash> AlffIdealBasis(I);
[ [ T, 0, 0 ] / (T + 1), [ 0, 1, 0 ] / (T + 1), [ 0, 0, 1 ] / (T + 1) ]
```

NAME	AlffIdealBasisUpperHNF
PURPOSE	Return an ideal basis in upper Hermite normal form.
SYNTAX	<pre>L := AlffIdealBasisUpperHNF(I);</pre> <div> <pre>list L</pre> <pre>alff order ideal I</pre> </div>
DESCRIPTION	This function returns a list of two entries. The first entry is a matrix in upper Hermite normal form representing an ideal basis with respect to the order basis. The second entry is the denominator of the ideal.
SEE ALSO	AlffIdealBasis , AlffIdeal2EltAssure ,
EXAMPLE	Some computations:

```
kash> AlffInit(FF(5,2));
"Defining global variables: k, w, kT, kTf, kTy, T, y, AlffGlobals"
kash> AlffOrders(y^3+T^4+1);
"Defining global variables: F, o, oi, one"
kash> I := T*o;
< [ T, 0, 0 ] >
kash> AlffIdealBasisUpperHNF(I);
[ [T 0 0]
  [0 T 0]
  [0 0 T], 1 ]
kash> a := AlffElt(o, [0, 1, 0]);
[ 0, 1, 0 ]
kash> I := a*o;
< [ 0, 1, 0 ] >
kash> AlffIdealBasisUpperHNF(I);
[ [ T^4 + 1      0      0]
  [      0      1      0]
  [      0      0      1], 1 ]
```

NAME	<code>AlffIdealClassGroupUnitsInfty</code>
PURPOSE	Compute the unit group and the ideal class group of the finite maximal order $\text{Cl}(k[T], F)$.
SYNTAX	$L := \text{AlffIdealClassGroupUnitsInfty}(F);$ <p> <code>list</code> <code>L</code> units and ideal class group info <code>alff</code> <code>F</code> global function field </p>
DESCRIPTION	Let F/k be a global function field and \mathfrak{o} be the finite maximal order $\text{Cl}(k[T], F)$. This function returns a list containing the regulator and the structure of the ideal class group of \mathfrak{o} in the form group order, list of cyclic factors.
SEE ALSO	AlffClassGroup , AlffSUnits , AlffInit , AlffOrders ,
EXAMPLE	

```

kash> AlffInit(FF(7));
kash> AlffOrders(y^2 + T^3 + T + 1);
kash> AlffPlaceSplitType(F, 1/T);
[ [ 2, 1 ] ]
kash> AlffIdealClassGroupUnitsInfty(F);
[ 1, [ 11, [ 11 ] ] ]
kash> AlffClassGroup(F);
[ 11, [ 11 ] ]

```


NAME	AlffIdealFactor
PURPOSE	Return the factorization of an ideal.
SYNTAX	$L := \text{AlffIdealFactor}(I);$ <div style="margin-left: 100px;"> $\begin{array}{ll} \text{list} & L \\ \text{alff order ideal} & I \text{ in maximal order} \end{array}$ </div>
DESCRIPTION	This function computes the factorization of an ideal I in its order which must be maximal. The return value is a list of pairs consisting of prime ideals \mathfrak{P}_i and exponents e_i such that $I = \prod \mathfrak{P}_i^{e_i}$.
SEE ALSO	AlffIdealValuation , AlffIdealIsPrime ,
EXAMPLE	Factor some ideals:

```

kash> AlffInit(F5,2));
"Defining global variables: k, w, kT, kTf, kTy, T, y, AlffGlobals"
kash> AlffOrders(y^3+T^4+1);
"Defining global variables: F, o, oi, one"
kash> AlffIdealFactor((T+w^3)*o);
[ [ < [ T + w^3, 0, 0 ], [ 0, 1, 0 ] >, 3 ] ]
kash> a := AlffElt(o, [0, 1, 0]);
[ 0, 1, 0 ]
kash> I := (T + w^3)*o + a*o;
<
[ T + w^3      0      0]
[      0      1      0]
[      0      0      1]
/ 1
>
kash> AlffIdealFactor(I);
[ [ < [ T + w^3, 0, 0 ], [ 0, 1, 0 ] >, 1 ] ]

```

NAME	AlffIdealGenerators
PURPOSE	Returns a list of generators of an ideal.
SYNTAX	$L := \text{AlffIdealGenerators}(I);$ <div style="margin-left: 40px;"> ideal I list L of two algebraic numbers </div>

SEE ALSO [AlffIdealBasis](#),

EXAMPLE

```
kash> k := FF(3);
Finite field of size 3
kash> AlffInit(k);
"Defining global variables: k, w, kT, kTf, kTy, T, y, AlffGlobals"
kash> f := y^3-y^2*(T+1)+2*y*T-T^5;
y^3 + (2*T + 2)*y^2 + 2*T*y + 2*T^5
kash> F:=Alff(f);
Algebraic function field defined by
$.1^3 + 2*$.1^2*$.2 + 2*$.1^2 + 2*$.1*$.2 + 2*$.2^5
over
Univariate rational function field over GF(3)
Variables: T

kash> AlffOrders(f);
"Defining global variables: F, o, oi, one"
kash> AlffIdealGenerators((1+T^2)*o);
[ [ T^2 + 1, 0, 0 ] ]
```

NAME	AlffIdealIsPrime
PURPOSE	missing shortdoc
SYNTAX	<pre>b := AlffIdealIsPrime(I);</pre> <p>boolean b</p> <p>alff order ideal i in maximal order</p>
DESCRIPTION	Perform a primality test for an ideal which is defined in a maximal order.
SEE ALSO	AlffIdealFactor , AlffIdealIsPrimeKnown ,
EXAMPLE	

```
kash> AlffInit(FF(5,2));
"Defining global variables: k, w, kT, kTf, kTy, T, y, AlffGlobals"
kash> AlffOrders(y^3+T^3+1);
"Defining global variables: F, o, oi, one"
kash> I := T*o;
< [ T, 0, 0 ] >
kash> AlffIdealIsPrime(I);
false
kash> l := AlffIdealFactor(I);
[ [ < [ T, 0, 0 ], [ 1, 1, 0 ] >, 1 ], [ < [ T, 0, 0 ], [ w^8, 1, 0 ] >, 1 ],
  [ < [ T, 0, 0 ], [ w^16, 1, 0 ] >, 1 ] ]
kash> AlffIdealIsPrime(l[1][1]);
true
```

NAME	AlffIdealIsPrimeKnown
PURPOSE	missing shortdoc
SYNTAX	<pre>b := AlffIdealIsPrimeKnown(I);</pre> <p>boolean b</p> <p>alff order ideal I in maximal order</p>
DESCRIPTION	Return whether an ideal defined in a maximal order is already known to be prime.
SEE ALSO	AlffIdealIsPrime , AlffIdealFactor ,
EXAMPLE	

```
kash> AlffInit(Q);
"Defining global variables: k, w, kT, kTf, kTy, T, y, AlffGlobals"
kash> AlffOrders(y^3+T^4+1);
"Defining global variables: F, o, oi, one"
kash> I := (T + 2)*o;
< [ T + 2, 0, 0 ] >
kash> AlffIdealIsPrimeKnown(I);
false
kash> l := AlffIdealFactor(I);
[ [ < [ T + 2, 0, 0 ] >, 1 ] ]
kash> AlffIdealIsPrimeKnown(I);
false
```

NAME	AlffIdealNorm
PURPOSE	missing shortdoc
SYNTAX	<pre>a := AlffIdealNorm(I);</pre> <p> rational function or polynomial a alff order ideal I </p>
DESCRIPTION	Compute the norm of an ideal over the rational function field $k(T)$.
SEE ALSO	AlffIdealBasisUpperHNF ,
EXAMPLE	

```
kash> AlffInit(FF(7,1));
"Defining global variables: k, w, kT, kTf, kTy, T, y, AlffGlobals"
kash> AlffOrders(y^3+T^2+2);
"Defining global variables: F, o, oi, one"
kash> I := T*o;
< [ T, 0, 0 ] >
kash> AlffIdealNorm(I);
T^3
```

NAME	AlffIdealOrder
PURPOSE	missing shortdoc
SYNTAX	<pre>o := AlffIdealOrder(a); alff order o alff order ideal a</pre>
DESCRIPTION	Return the order in which the ideal is defined.
SEE ALSO	AlffOrderAlff ,
EXAMPLE	

```
kash> AlffInit(FF(5,2));
"Defining global variables: k, w, kT, kTf, kTy, T, y, AlffGlobals"
kash> AlffOrders(y^3+T^4+1);
"Defining global variables: F, o, oi, one"
kash> a := AlffElt(o, [0, 1, 0]);
[ 0, 1, 0 ]
kash> I := (T + w^3)*o + a*o;
<
[ T + w^3      0      0]
[      0      1      0]
[      0      0      1]
/ 1
>
kash> AlffIdealOrder(I);
Finite maximal order of
Algebraic function field defined by
$.1^3 + $.2^4 + 1
over
Univariate rational function field over GF(5^2)
Variables: T
```

NAME	AlffIdealPlace
PURPOSE	Given a prime ideal this function returns the corresponding place.
SYNTAX	<pre>P := AlffIdealPlace(I); prime ideal I alff place P</pre>
SEE ALSO	AlffPlaceIdeal , AlffPlaces ,
EXAMPLE	

```
kash> k := FF(3);
Finite field of size 3
kash> AlffInit(k);
"Defining global variables: k, w, kT, kTf, kTy, T, y, AlffGlobals"
kash> f := y^3-y^2*(T+1)+2*y*T-T^5;
y^3 + (2*T + 2)*y^2 + 2*T*y + 2*T^5
kash> F:=Alff(f);
Algebraic function field defined by
$.1^3 + 2*$.1^2*$.2 + 2*$.1^2 + 2*$.1*$.2 + 2*$.2^5
over
Univariate rational function field over GF(3)
Variables: T

kash> AlffOrders(f);
"Defining global variables: F, o, oi, one"
kash> u := AlffElt(o,T);
[ T, 0, 0 ]
kash> ud:=AlffIdealFactor(u*o);
[ [ < [ T, 0, 0 ], [ T, 1, 2*T + 2 ] >, 1 ],
  [ < [ T, 0, 0 ], [ 2*T + 1, T + 2, T ] >, 1 ],
  [ < [ T, 0, 0 ], [ T + 1, 2*T, 2 ] >, 1 ] ]
kash> AlffIdealPlace(ud[1][1]);
Alff place < [ T, 0, 0 ], [ 0, 0, 1 ] >
```

NAME	AlffIdealValuation
PURPOSE	Return the valuation of an ideal at a prime ideal.
SYNTAX	<pre>v := AlffIdealValuation(P, I);</pre> <p>integer v alff order ideals P, I in maximal order o</p>
DESCRIPTION	This function returns the valuation of an ideal at a prime ideal. They have to be defined in the same order which must be maximal (i. e. a Dedekind ring).
SEE ALSO	AlffIdealFactor , AlffIdealIsPrime ,
EXAMPLE	Compute a valuation:

```
kash> AlffInit(FF(5,2));
"Defining global variables: k, w, kT, kTf, kTy, T, y, AlffGlobals"
kash> AlffOrders(y^3+T^4+1);
"Defining global variables: F, o, oi, one"
kash> a := AlffElt(o, [0, 1, 0]);
[ 0, 1, 0 ]
kash> I := (T + w^3)*o + a*o;
<
[ T + w^3      0      0]
[      0      1      0]
[      0      0      1]
/ 1
>
kash> P := AlffIdealFactor(I)[1][1];
< [ T + w^3, 0, 0 ], [ 0, 1, 0 ] >
kash> AlffIdealValuation(P, I^(-2));
-2
```


NAME	AlffIharaBound
PURPOSE	missing shortdoc
SYNTAX	<pre> b := AlffIharaBound(F); b := AlffIharaBound(q, g); global function field F integer q, g integer b </pre>
DESCRIPTION	Computes the Ihara bound for a global function field over the exact constant field \mathbb{F}_q for the number of places of degree one from the function field or for given q and g , where q is the size of the finite field and g denotes the genus.
SEE ALSO	AlffSerreBound , AlffPlacesDegOneNumBound ,
EXAMPLE	

```

kash> AlffInit(FF(2,1));
"Defining global variables: k, w, kT, kTf, kTy, T, y, AlffGlobals"
kash> AlffOrders(y^3 + T^3 * y + T);
"Defining global variables: F, o, oi, one"
kash> AlffGenus(F);
3
kash> AlffIharaBound(F);
8
kash> AlffIharaBound(2, 1);
5

```

NAME	AlffInit
PURPOSE	Initializes some useful variables for an algebraic function field session.
SYNTAX	<pre>AlffInit(k); AlffInit(k, T); AlffInit(k, T, y);</pre> <p>field k strings T, y the variable names</p>
DESCRIPTION	<p>This function is used for convenience to initialize some useful variables for an algebraic function field session and is not recommended for use in programming. It initializes:</p> <ul style="list-style-type: none"> • k, the constant field, • w, a generator of the multiplicative group of k if finite, • kT, the polynomial ring $k[T]$, • kTf, the field $k(T)$, • kTy, the polynomial ring $k[T][y]$, • T, the variable T, • y, the variable y, • AlffGlobals, a record containing all the above values. <p>The optional two parameter strings define the names of variables T and y.</p>
SEE ALSO	AlffOrders , Alff ,

EXAMPLE

```
kash> AlffInit(FF(5, 2));
"Defining global variables: k, w, kT, kTf, kTy, T, y, AlffGlobals"
kash> k;
Finite field of size 5^2
kash> kTy;
Univariate Polynomial Ring in y over Univariate Polynomial Ring in T over GF(5\
^2)

kash> y^4+T^3+2;
y^4 + T^3 + 2
```

NAME	<code>AlffInit</code>
PURPOSE	Initializes some useful variables for an algebraic function field session.
SYNTAX	<pre>AlffInit(k); AlffInit(k, T); AlffInit(k, T, y);</pre> <p> <code>field</code> <code>k</code> <code>strings</code> <code>T, y</code> the variable names </p>
DESCRIPTION	<p>This function is used for convenience to initialize some useful variables for an algebraic function field session and is not recommended for use in programming. It initializes:</p> <ul style="list-style-type: none"> • <code>k</code>, the constant field, • <code>w</code>, a generator of k over its base field if finite, • <code>kT</code>, the polynomial ring $k[T]$, • <code>kTf</code>, the field $k(T)$, • <code>kTy</code>, the polynomial ring $k[T][y]$, • <code>T</code>, the variable T, • <code>y</code>, the variable y, • <code>AlffGlobals</code>, a record containing all the above values. <p>The optional two parameter strings define the names of variables T and y.</p>
SEE ALSO	AlffOrders , Alff ,

EXAMPLE

```
kash> AlffInit(FF(5, 2));
"Defining global variables: k, w, kT, kTf, kTy, T, y, AlffGlobals"
kash> k;
Finite field of size 5^2
kash> kTy;
Univariate Polynomial Ring in y over Univariate Polynomial Ring in T over GF(5\
^2)

kash> y^4+T^3+2;
y^4 + T^3 + 2
```

NAME AlffIsAbs

PURPOSE Returns whether a function field is an absolute extension.

SYNTAX `b := AlffIsAbs(F);`

 boolean b

 function field F

DESCRIPTION

EXAMPLE

```
kash> k := FF(3);
Finite field of size 3
kash> AlffInit(k);
"Defining global variables: k, w, kT, kTf, kTy, T, y, AlffGlobals"
kash> f := y^3-y^2*(T+1)+2*y*T-T^5;
y^3 + (2*T + 2)*y^2 + 2*T*y + 2*T^5
kash> F:=Alff(f);
Algebraic function field defined by
$.1^3 + 2*$.1^2*$.2 + 2*$.1^2 + 2*$.1*$.2 + 2*$.2^5
over
Univariate rational function field over GF(3)
Variables: T

kash> AlffIsAbs(F);
true
kash> AlffOrders(f);
"Defining global variables: F, o, oi, one"
kash> a1:=AlffElt(o,[1,3,T]);
[ 1, 0, T ]
kash> a2:=AlffElt(o,[T,T^2,0]);
[ T, T^2, 0 ]
kash> a3:=AlffElt(o,[1,3,0]);
[ 1, 0, 0 ]
kash> L:=[a3,a2,a1];
[ [ 1, 0, 0 ], [ T, T^2, 0 ], [ 1, 0, T ] ]
kash> A:=PolyAlg(o);
Univariate Polynomial Ring in x over Finite maximal order of
Algebraic function field defined by
$.1^3 + 2*$.1^2*$.2 + 2*$.1^2 + 2*$.1*$.2 + 2*$.2^5
```

```

over
Univariate rational function field over GF(3)
Variables: T
given by transformation matrix
[T 0 0]
[0 T 2]
[0 0 1]
with denominator T

kash> g:=Poly(A,L);
x^2 + [ T, T^2, 0 ]*x + [ 1, 0, T ]
kash> G:=Alff(g);
Algebraic function field defined by
$.1^2 + $.1*$.2*$.3^2 + $.1*$.3 + $.2^2 + 2*$.2 + 1
over
Algebraic function field defined by
$.1^3 + 2*$.1^2*$.2 + 2*$.1^2 + 2*$.1*$.2 + 2*$.2^5
over
Univariate rational function field over GF(3)
Variables: T

kash> AlffIsAbs(G);
false

```

NAME	AlffIsGlobal
PURPOSE	Returns whether an algebraic function field F is global.
SYNTAX	<pre>b := AlffIsGlobal(F);</pre> <p>boolean b whether global or not algebraic function field F</p>
SEE ALSO	AlffIsGlobalAssert , AlffConstField , Characteristic ,

EXAMPLE

```
kash> AlffInit(Q);
"Defining global variables: k, w, kT, kTf, kTy, T, y, AlffGlobals"
kash> F := Alff(y^3+T^3*y+T);
Algebraic function field defined by
$.1^3 + $.1*$.2^3 + $.2
over
Univariate rational function field over Rational Field
Variables: T

kash> AlffIsGlobal(F);
false
```

NAME	AlffIsGlobalAssert
PURPOSE	Signals an error if an algebraic function field F is not global.
SYNTAX	AlffIsGlobalAssert(F); algebraic function field F
SEE ALSO	AlffIsGlobal , AlffConstField , Characteristic ,
EXAMPLE	

```

kash> AlffInit(FF(2,1));
"Defining global variables: k, w, kT, kTf, kTy, T, y, AlffGlobals"
kash> F := Alff(y^3+T^3*y+T);
Algebraic function field defined by
$.1^3 + $.1*$.2^3 + $.2
over
Univariate rational function field over GF(2)
Variables: T

kash> AlffIsGlobalAssert(F);

```

NAME	AlffLPoly
PURPOSE	Computes the L -polynomial of a global function field.
SYNTAX	$L := \text{AlffLPoly}(F);$ <p style="margin-left: 100px;"> polynomial L $\text{global function field}$ F </p>
DESCRIPTION	This function computes the L -polynomial of a global function field F/k . The algorithm used is exponential in the genus of F/k . The constant field of definition of F/k has to be exact.
SEE ALSO	AlffPlacesDegOneNum , AlffPlacesDegOne ,
EXAMPLE	

```

kash> AlffInit(FF(5,1));;
kash> AlffOrders(y^4 + T^4 + 1);
"Defining global variables: F, o, oi, one"
kash> AlffLPoly(F);
125*x^6 - 150*x^5 + 135*x^4 - 68*x^3 + 27*x^2 - 6*x + 1

```


NAME	AlffLPolyLift
PURPOSE	Lifts the L -polynomial of a global function field to constant field extensions.
SYNTAX	<pre>Lr := AlffLPolyLift(L, r);</pre> <p style="margin-left: 40px;">polynomial Lr, L integer r</p>
DESCRIPTION	This function computes the L -polynomial of a constant field extension of a global function field F/k of degree r from the L -polynomial L of F/k .
SEE ALSO	AlffLPoly , AlffPlacesDegOne ,
EXAMPLE	

```
kash> AlffInit(FF(5,1));;
kash> AlffOrders(y^4 + T^4 + 1);
"Defining global variables: F, o, oi, one"
kash> L := AlffLPoly(F);
125*x^6 - 150*x^5 + 135*x^4 - 68*x^3 + 27*x^2 - 6*x + 1
kash> AlffLPolyLift(L, 2);
15625*x^6 + 11250*x^5 + 4575*x^4 + 1116*x^3 + 183*x^2 + 18*x + 1
kash> AlffInit(FF(5,2));;
kash> AlffOrders(y^4 + T^4 + 1);
"Defining global variables: F, o, oi, one"
kash> AlffLPoly(F);
15625*x^6 + 11250*x^5 + 4575*x^4 + 1116*x^3 + 183*x^2 + 18*x + 1
```

NAME	AlffLPolyRed
PURPOSE	Computes the L -polynomial of a global function field.
SYNTAX	<pre> L := AlffLPolyRed(F); L := AlffLPolyRed(F, "nocheck"); polynomial L F global function field F </pre>
DESCRIPTION	<p>This function computes the L-polynomial of a global function field of characteristic p modulo p. The constant field of definition of F/k has to be exact and F/k must contain a non special system of places. If the parameter "nocheck" is given, the existence of the non special system is not checked (hence the resulting L-polynomial may be false).</p>
SEE ALSO	AlffLPolyRed ,
EXAMPLE	

```

kash> AlffInit(FF(5,1));;
kash> AlffOrders(y^4 + T^4 + T + 1);
"Defining global variables: F, o, oi, one"
kash> AlffLPolyRed(F);
3*x^3 + 4*x^2 + 4*x + 1
kash> AlffLPoly(F) mod 5;
-2*x^3 - x^2 - x + 1

```

NAME	AlffLinearSeriesEnumElt
PURPOSE	Return the current divisor in the series enumeration.
SYNTAX	<p><code>E := AlffLinearSeriesEnumElt(env);</code></p> <p>alff divisor E current divisor record env of enumeration data</p>
DESCRIPTION	Let F/k be a global function field and D a divisor. Let v_1, \dots, v_l be k -linearly independent elements of F . This function returns the (according to an internal counter) current divisor $D + (\sum_{i=1}^l \lambda_i v_i)$ in the linear series enumeration.
SEE ALSO	AlffLinearSeriesEnumNext , AlffLinearSeriesEnumEnv ,
EXAMPLE	

```

kash> AlffInit(FF(5,1));;
kash> AlffOrders(y^2+T^3+1);;
kash> D := 1*AlffPlaceRandom(F, 3);
Alff divisor
[ [ Alff place < [ T^3 + 4*T^2 + T + 2, 0 ], [ 3*T + 1, 1 ] >, 1 ] ]

kash> B := AlffDivisorLBasis(D);
[ [ 1, 0 ], [ 2*T + 4, 1 ] / (T^3 + 4*T^2 + T + 2),
  [ 2*T^2 + 4*T, T ] / (T^3 + 4*T^2 + T + 2) ]
kash> env := AlffLinearSeriesEnumEnv(D, B);
rec(
  ffelts := [ 0, 2, 4, 3, 1 ],
  D := Alff divisor
    [ [ Alff place < [ T^3 + 4*T^2 + T + 2, 0 ], [ 3*T + 1, 1 ] >, 1 ] ]
  ,
  basis := [ [ 1, 0 ], [ 2*T + 4, 1 ] / (T^3 + 4*T^2 + T + 2),
    [ 2*T^2 + 4*T, T ] / (T^3 + 4*T^2 + T + 2) ],
  dim := 3 )
kash> AlffLinearSeriesEnumNext(env);
true
kash> AlffLinearSeriesEnumElt(env);
<
[1 0]
[0 1]
/ T

```

```
>, <
[1/T  0]
[  0  1]
/ (1/T)
> ]
```

```
kash> AlffLinearSeriesEnumNext(env);
true
kash> AlffLinearSeriesEnumElt(env);
<
[T^3 + 2      4]
[      0      1]
/ T^3 + 2
>, <
[1 0]
[0 1]
/ (1)
> ]
```

NAME	<code>AlffLinearSeriesEnumEnv</code>
PURPOSE	Prepare for enumeration of a linear series over a finite field.
SYNTAX	<pre>env := AlffLinearSeriesEnumEnv(D, B);</pre> <pre> record env of enumeration data alff divisor D list B of v_1, \dots, v_l</pre>
DESCRIPTION	Let F/k be a global function field and D a divisor. Let v_1, \dots, v_l be k -linearly independent elements of F . This function returns a record which is used by <code>AlffLinearSeriesEnumNext()</code> and <code>AlffLinearSeriesEnumElt()</code> to enumerate successively the divisors $D + (\sum_{i=1}^l \lambda_i v_i)$ (each once).
SEE ALSO	AlffLinearSeriesEnumNext , AlffLinearSeriesEnumElt ,
EXAMPLE	

```

kash> AlffInit(FF(5,1));;
kash> AlffOrders(y^2+T^3+1);;
kash> D := 1*AlffPlaceRandom(F, 3);
Alff divisor
[ [ Alff place < [ T^3 + 4*T^2 + T + 2, 0 ], [ 3*T + 1, 1 ] >, 1 ] ]

kash> B := AlffDivisorLBasis(D);
[ [ 1, 0 ], [ 2*T + 4, 1 ] / (T^3 + 4*T^2 + T + 2),
  [ 2*T^2 + 4*T, T ] / (T^3 + 4*T^2 + T + 2) ]
kash> env := AlffLinearSeriesEnumEnv(D, B);
rec(
  ffelts := [ 0, 2, 4, 3, 1 ],
  D := Alff divisor
    [ [ Alff place < [ T^3 + 4*T^2 + T + 2, 0 ], [ 3*T + 1, 1 ] >, 1 ] ]
  ,
  basis := [ [ 1, 0 ], [ 2*T + 4, 1 ] / (T^3 + 4*T^2 + T + 2),
    [ 2*T^2 + 4*T, T ] / (T^3 + 4*T^2 + T + 2) ],
  dim := 3 )

```

NAME	AlffLinearSeriesEnumNext
PURPOSE	Whether there is a next divisor in the linear series enumeration.
SYNTAX	<pre>b := AlffLinearSeriesEnumNext(env);</pre> <p> bool b whether there is a next divisor record env of enumeration data </p>
DESCRIPTION	<p>Let F/k be a global function field and D a divisor. Let v_1, \dots, v_l be k-linearly independent elements of F. This function returns true or false with respect to whether there is a not yet considered divisor in a linear series enumeration of the form $D + (\sum_{i=1}^l \lambda_i v_i)$ and increases an internal counter.</p>
SEE ALSO	AlffLinearSeriesEnumEnv , AlffLinearSeriesEnumElt ,
EXAMPLE	

```

kash> AlffInit(FF(5,1));;
kash> AlffOrders(y^2+T^3+1);;
kash> D := 1*AlffPlaceRandom(F, 3);
Alff divisor
[ [ Alff place < [ T^3 + 4*T^2 + T + 2, 0 ], [ 3*T + 1, 1 ] >, 1 ] ]

kash> B := AlffDivisorLBasis(D);
[ [ 1, 0 ], [ 2*T + 4, 1 ] / (T^3 + 4*T^2 + T + 2),
  [ 2*T^2 + 4*T, T ] / (T^3 + 4*T^2 + T + 2) ]
kash> env := AlffLinearSeriesEnumEnv(D, B);
rec(
  ffelts := [ 0, 2, 4, 3, 1 ],
  D := Alff divisor
    [ [ Alff place < [ T^3 + 4*T^2 + T + 2, 0 ], [ 3*T + 1, 1 ] >, 1 ] ]
  ,
  basis := [ [ 1, 0 ], [ 2*T + 4, 1 ] / (T^3 + 4*T^2 + T + 2),
    [ 2*T^2 + 4*T, T ] / (T^3 + 4*T^2 + T + 2) ],
  dim := 3 )
kash> AlffLinearSeriesEnumNext(env);
true
kash> AlffLinearSeriesEnumNext(env);
true

```

NAME	AlffOesterleBound
PURPOSE	Computes the Oesterle bound of a global function field.
SYNTAX	<pre>b := AlffOesterleBound(q,g); integer b integer q, g</pre>
DESCRIPTION	Let g be the genus of a function field F over the exact constant field \mathbb{F}_q . This function computes the Oesterle bound for the number of places of degree one of F . This function is the optimization of the explicit formulae. To use if the genus is large.
SEE ALSO	AlffSerreBound , AlffIharaBound , AlffPlacesDegOneNum ,
EXAMPLE	<pre>kash> AlffInit(FF(7));; kash> F:=Alff(y^9 + y + 6*T^16); Algebraic function field defined by \$.1^9 + \$.1 + 6*\$.2^16 over Univariate rational function field over GF(7) Variables: T kash> g:=AlffGenus(F); 60 kash> AlffOesterleBound(7,g); 176</pre>

NAME AlffOrderAlff

PURPOSE Returns the algebraic function field of an algebraic function field order.

SYNTAX F := AlffOrderAlff(o);

 algebraic function field F
 algebraic function field order o

SEE ALSO AlffOrderPoly, Alff,

EXAMPLE

```
kash> AlffInit(FF(5,2));
"Defining global variables: k, w, kT, kTf, kTy, T, y, AlffGlobals"
kash> AlffOrders(y^3+T^4+1);
"Defining global variables: F, o, oi, one"
kash> AlffOrderAlff(o);
Algebraic function field defined by
$.1^3 + $.2^4 + 1
over
Univariate rational function field over GF(5^2)
Variables: T
```


NAME	AlffOrderBasis
PURPOSE	Returns a basis as elements of the given alff order with a denominator.
SYNTAX	<pre>L := AlffOrderBasis(o);</pre> <div> list of algebraic elements L alff order o </div>
DESCRIPTION	The basis is returned as elements of the equation order.
SEE ALSO	AlffOrder ,
EXAMPLE	

```
kash> AlffInit(FF(2,1));
"Defining global variables: k, w, kT, kTf, kTy, T, y, AlffGlobals"
kash> AlffOrders(y^3+T^3*y+T);
"Defining global variables: F, o, oi, one"
kash> Bf := AlffOrderBasis(o);
[ [ 1, 0, 0 ], [ 0, 1, 0 ], [ 0, 0, 1 ] ]
kash> Bi := AlffOrderBasis(oi);
[ [ 1, 0, 0 ], [ 0, 1, 0 ], [ 0, 0, 1 ] ]
kash> Bf = Bi;
false
```

NAME	AlffOrderBasis
PURPOSE	Returns a list containing the basis elements of an alff order.
SYNTAX	<pre>B := AlffOrderBasis(o);</pre> <p>list B of basis elements alff order o</p>

SEE ALSO [AlffIdealBasis](#),

EXAMPLE

```
kash> AlffInit(FF(2,1));
"Defining global variables: k, w, kT, kTf, kTy, T, y, AlffGlobals"
kash> AlffOrders(y^3+T^3*y+T);
"Defining global variables: F, o, oi, one"
kash> Bf := AlffOrderBasis(o);
[ [ 1, 0, 0 ], [ 0, 1, 0 ], [ 0, 0, 1 ] ]
kash> Bi := AlffOrderBasis(oi);
[ [ 1, 0, 0 ], [ 0, 1, 0 ], [ 0, 0, 1 ] ]
kash> Bf = Bi;
false
```

NAME	AlffOrderBasisValues
PURPOSE	Returns some values depending on the B^* -values of the basis of an order of a global function field.
SYNTAX	$L := \text{AlffOrderBasisValues}(o);$ <div style="display: flex; justify-content: space-between; margin-top: 10px;"> list L </div> <div style="margin-top: 10px;"> global function field order o </div>
DESCRIPTION	<p>Denote by n the $\mathbb{F}_q[T]$-rank of the order o and by $b_1, \dots, b_n \in o$ its basis. Then the function returns</p> $L := [[eB^*(b_1), \dots, eB^*(b_n)], \max_{i=1}^n B^*(b_i), \sum_{i=1}^n B^*(b_i), e].$ <p>The infinite place of $k(T)$ has to be tamely ramified in F. See <code>AlffEltBstar()</code> for the definition of B^*, e and [Sch96] for further definitions and algorithms.</p>
SEE ALSO	Alff , AlffOrderL0 , AlffOrderReduce ,
EXAMPLE	<pre> kash> AlffInit(FF(5,2)); "Defining global variables: k, w, kT, kTf, kTy, T, y, AlffGlobals" kash> AlffOrders(y^3+T^4+1); "Defining global variables: F, o, oi, one" kash> AlffOrderBasisValues(o); [[0, 4, 8], 8, 12, 3] </pre>

NAME	AlffOrderDedekindTest
PURPOSE	Performs the Dedekind-Test on an equation order in an algebraic function field.
SYNTAX	<pre> b := AlffOrderDedekindTest(o); b := AlffOrderDedekindTest(o, g); boolean b algebraic function field order o polynomial g </pre>
DESCRIPTION	Let o be an equation order in an algebraic function field $F/k(T)$. Invoked with one argument, the function returns whether o is $1/T$ -maximal or not. Invoked with a prime polynomial $g \in k[T]$, the function returns whether o is g -maximal or not.

EXAMPLE

```

kash> AlffInit(FF(5,2));
"Defining global variables: k, w, kT, kTf, kTy, T, y, AlffGlobals"
kash> F := Alff(y^3+T^4+1);
Algebraic function field defined by
$.1^3 + $.2^4 + 1
over
Univariate rational function field over GF(5^2)
Variables: T

kash> AlffOrderDedekindTest(AlffOrderEqFinite(F), T+1);
true
kash> AlffOrderDedekindTest(AlffOrderEqInfty(F));
false

```

NAME	AlffOrderDeg
PURPOSE	Returns the degree of an alff order over its coefficient ring.
SYNTAX	<pre>d := AlffOrderDeg(o);</pre> <p>integer d degree alff order o</p>
SEE ALSO	AlffDeg ,
EXAMPLE	

```
kash> AlffInit(FF(2,1));
"Defining global variables: k, w, kT, kTf, kTy, T, y, AlffGlobals"
kash> AlffOrders(y^3+T^3*y+T);
"Defining global variables: F, o, oi, one"
kash> AlffOrderDeg(o);
```

3

NAME	AlffOrderDisc
PURPOSE	Returns the discriminant of an algebraic function field order up to a unit.
SYNTAX	$d := \text{AlffOrderDisc}(o);$ <p> polynomial or quotient field element d algebraic function field order o </p>
DESCRIPTION	The discriminant d is an element of the order's coefficient ring, i.e. d is in $k[T]$ or in the valuation ring of the degree valuation of $k(T)$. It is determined up to a unit.
SEE ALSO	Alff , AlffOrderMaxFinite , AlffOrderMaxInfty ,
EXAMPLE	

```

kash> AlffInit(FF(5,2));
"Defining global variables: k, w, kT, kTf, kTy, T, y, AlffGlobals"
kash> AlffOrders(y^3+T^4+1);
"Defining global variables: F, o, oi, one"
kash> AlffOrderDisc(o);
3*T^8 + T^4 + 3
kash> AlffOrderDisc(oi);
(3*T^8 + T^4 + 3)/T^10

```

NAME	AlffOrderEqFinite
PURPOSE	Computes an equation order of the given algebraic function field.
SYNTAX	<pre>o := AlffOrderEqFinite(F);</pre> <p>algebraic function field order o algebraic function field F</p>
DESCRIPTION	Computes the $k[T]$ -equation order of an algebraic function field $F/k(T)$, that is $k[T, \rho]$ where $f(T, \rho) = 0$ and $F/k(T)$ is defined by f .
SEE ALSO	AlffOrderEqInfty , Alff , AlffElt ,
EXAMPLE	

```
kash> AlffInit(FF(5,2));
"Defining global variables: k, w, kT, kTf, kTy, T, y, AlffGlobals"
kash> F := Alff(y^3+T^4+1);
Algebraic function field defined by
$.1^3 + $.2^4 + 1
over
Univariate rational function field over GF(5^2)
Variables: T

kash> AlffOrderEqFinite(F);
Finite equation order of
Algebraic function field defined by
$.1^3 + $.2^4 + 1
over
Univariate rational function field over GF(5^2)
Variables: T
```

NAME	AlffOrderEqInfty
PURPOSE	Computes an equation order of the given algebraic function field.
SYNTAX	<pre>o := AlffOrderEqInfty(F);</pre> <p>algebraic function field order o algebraic function field F</p>
DESCRIPTION	<p>Let R be the valuation ring of the degree valuation of $k(T)$, that is</p> $R = \{g/h \mid g, h \in k[T], h \neq 0, \deg(g) \leq \deg(h)\},$ <p>and let $F/k(T)$ be an algebraic function field. This function computes an R-equation order $R[\rho]$ where $\rho \in F$ generates F over $k(T)$ and is integral over R.</p>
SEE ALSO	AlffOrderEqFinite , Alff , AlffElt ,
EXAMPLE	

```
kash> AlffInit(FF(5,2));
"Defining global variables: k, w, kT, kTf, kTy, T, y, AlffGlobals"
kash> F := Alff(y^3+T^4+1);
Algebraic function field defined by
$.1^3 + $.2^4 + 1
over
Univariate rational function field over GF(5^2)
Variables: T

kash> AlffOrderEqInfty(F);
Infinite equation order of
Algebraic function field defined by
$.1^3 + $.2^4 + 1
over
Univariate rational function field over GF(5^2)
Variables: T
```


NAME	AlffOrderIndex
PURPOSE	Returns the index of an order.
SYNTAX	<pre>d := AlffOrderIndex(o);</pre> <p> element d of the coefficient ring algebraic function field order o </p>
DESCRIPTION	This function returns the index of the equation order in the given order (as a module over its coefficient ring).
SEE ALSO	AlffOrderMaxFinite , AlffOrderMaxInfty ,
EXAMPLE	

```
kash> AlffInit(FF(7,1));
"Defining global variables: k, w, kT, kTf, kTy, T, y, AlffGlobals"
kash> AlffOrders(y^3+T^3*y+T);
"Defining global variables: F, o, oi, one"
kash> o := AlffOrderMaxFinite(F);
Finite maximal order of
Algebraic function field defined by
$.1^3 + $.1*$.2^3 + $.2
over
Univariate rational function field over GF(7)
Variables: T
given by transformation matrix
[T^3 + T^2 + 5*T + 6      0      4*T^2 + 6*T + 3]
[      0 T^3 + T^2 + 5*T + 6      3*T^2 + 5*T + 3]
[      0      0      1]
with denominator T^3 + T^2 + 5*T + 6
kash> p := AlffOrderIndex(o);
1
```

NAME AlffOrderIsFinite

PURPOSE Returns whether a function field order is finite.

SYNTAX b := AlffOrderIsFinite(o);

 boolean b

 alff order P

DESCRIPTION

EXAMPLE

```
kash> k := FF(3);
Finite field of size 3
kash> AlffInit(k);
"Defining global variables: k, w, kT, kTf, kTy, T, y, AlffGlobals"
kash> f := y^3-(T+1)*y^2+2*y*T-T^5;
y^3 + (2*T + 2)*y^2 + 2*T*y + 2*T^5
kash> F:=Alff(f);
Algebraic function field defined by
$.1^3 + 2*$.1^2*$.2 + 2*$.1^2 + 2*$.1*$.2 + 2*$.2^5
over
Univariate rational function field over GF(3)
Variables: T

kash> AlffOrders(f);
"Defining global variables: F, o, oi, one"
kash> oe := AlffOrderEqFinite(F);
Finite equation order of
Algebraic function field defined by
$.1^3 + 2*$.1^2*$.2 + 2*$.1^2 + 2*$.1*$.2 + 2*$.2^5
over
Univariate rational function field over GF(3)
Variables: T

kash> oie:=AlffOrderEqInfty(F);
Infinite maximal order of
Algebraic function field defined by
$.1^3 + 2*$.1^2*$.2 + 2*$.1^2 + 2*$.1*$.2 + 2*$.2^5
over
Univariate rational function field over GF(3)
```

Variables: T

```
kash> AlffOrderIsFinite(o);  
true  
kash> AlffOrderIsFinite(oi);  
false  
kash> AlffOrderIsFinite(oe);  
true  
kash> AlffOrderIsFinite(oie);  
false
```

NAME	AlffOrderL0
PURPOSE	Returns a list of 0-reduced basis elements of an order of a global function field with their B^* -values.
SYNTAX	$L := \text{AlffOrderL0}(o);$ <div style="display: flex; justify-content: space-between; margin-top: 10px;"> list L </div> <div style="display: flex; justify-content: space-between; margin-top: 10px;"> global function field order o </div>
DESCRIPTION	<p>For the $\mathbb{F}_q[T]$-order o, the function returns a 0-reduced $\mathbb{F}_q[T]$-basis $b_1, \dots, b_n \in o$ with their B^*-values, i.e.</p> $L := [b_1, B^*(b_1), \dots, b_n, B^*(b_n)].$ <p>Here n denotes the $\mathbb{F}_q[T]$-rank of o. The infinite place of $k(T)$ has to be tamely ramified in F. See <code>AlffEltBstar()</code> for the definition of B^* and [Sch96] for the definition of 0-reduced and for algorithms.</p>
SEE ALSO	AlffOrderBasisValues , AlffOrderReduce ,
EXAMPLE	

```

kash> AlffInit(FF(5,2));
"Defining global variables: k, w, kT, kTf, kTy, T, y, AlffGlobals"
kash> AlffOrders(y^3+T^4+1);
"Defining global variables: F, o, oi, one"
kash> AlffOrderL0(o);
[ [ 1, 0, 0 ], 0, [ 0, 1, 0 ], 4/3, [ 0, 0, 1 ], 8/3 ]

```

NAME	AlffOrderMaxFinite
PURPOSE	Computes a maximal order of an algebraic function field.
SYNTAX	<pre>o := AlffOrderMaxFinite(F);</pre> <p>algebraic function field order o algebraic function field F</p>
DESCRIPTION	Computes the $k[T]$ -maximal order of an algebraic function field $F/k(T)$. It is the integral closure of $k[T]$ in F .
SEE ALSO	AlffOrderMaxInfty , AlffOrderMaximal , Alff , AlffElt ,
EXAMPLE	

```
kash> AlffInit(FF(5,2));
"Defining global variables: k, w, kT, kTf, kTy, T, y, AlffGlobals"
kash> F := Alff(y^3+T^4+1);
Algebraic function field defined by
$.1^3 + $.2^4 + 1
over
Univariate rational function field over GF(5^2)
Variables: T

kash> AlffOrderMaxFinite(F);
Finite maximal order of
Algebraic function field defined by
$.1^3 + $.2^4 + 1
over
Univariate rational function field over GF(5^2)
Variables: T
```

NAME	AlffOrderMaxInfty
PURPOSE	Computes a maximal order of an algebraic function field.
SYNTAX	<pre>o := AlffOrderMaxInfty(F);</pre> <p>algebraic function field order o algebraic function field F</p>
DESCRIPTION	<p>Let R be the valuation ring of the degree valuation of $k(T)$, that is</p> $R = \{g/h \mid g, h \in k[T], h \neq 0, \deg(g) \leq \deg(h)\},$ <p>and let $F/k(T)$ be an algebraic function field. This function computes the R-maximal order of F. It is the integral closure of R in F.</p>
SEE ALSO	AlffOrderMaxFinite , AlffOrderMaximal , Alff , AlffElt ,
EXAMPLE	

```
kash> AlffInit(FF(5,2));
"Defining global variables: k, w, kT, kTf, kTy, T, y, AlffGlobals"
kash> F := Alff(y^3+T^4+1);
Algebraic function field defined by
$.1^3 + $.2^4 + 1
over
Univariate rational function field over GF(5^2)
Variables: T

kash> AlffOrderMaxInfty(F);
Infinite maximal order of
Algebraic function field defined by
$.1^3 + $.2^4 + 1
over
Univariate rational function field over GF(5^2)
Variables: T
given by transformation matrix
[1/T  0  0]
[ 0 1/T  0]
[ 0  0  1]
with denominator 1/T
```

NAME	AlffOrderMaximal
PURPOSE	missing shortdoc
SYNTAX	$O := \text{AlffOrderMaximal}(o);$ algebraic function field order O algebraic function field order o
DESCRIPTION	Returns the maximal overorder of a given order o of an algebraic function field $F/k(T)$. It is the integral closure of o in F .
SEE ALSO	AlffOrderMaxFinite , AlffOrderMaxInfty ,
EXAMPLE	

```

kash> AlffInit(FF(5,2));
"Defining global variables: k, w, kT, kTf, kTy, T, y, AlffGlobals"
kash> F := Alff(y^3+T^4+1);
Algebraic function field defined by
$.1^3 + $.2^4 + 1
over
Univariate rational function field over GF(5^2)
Variables: T

kash> o := AlffOrderEqInfty(F);
Infinite equation order of
Algebraic function field defined by
$.1^3 + $.2^4 + 1
over
Univariate rational function field over GF(5^2)
Variables: T

kash> AlffOrderMaximal(o);
Infinite maximal order of
Algebraic function field defined by
$.1^3 + $.2^4 + 1
over
Univariate rational function field over GF(5^2)
Variables: T
given by transformation matrix
[1/T  0  0]

```

$$\begin{bmatrix} 0 & 1/T & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

with denominator $1/T$

NAME	AlffOrderPoly
PURPOSE	Returns the defining polynomial of the associated equation order of an algebraic function field.
SYNTAX	<pre>f := AlffOrderPoly(o);</pre> <p>polynomial f algebraic function field order o</p>
DESCRIPTION	The polynomial f is an element of $k[T, y]$ or $R[y]$. See <code>AlffOrderEqInfty()</code> for the definition of R .
SEE ALSO	AlffOrderDeg , Alff ,
EXAMPLE	

```
kash> AlffInit(F5,2));
"Defining global variables: k, w, kT, kTf, kTy, T, y, AlffGlobals"
kash> AlffOrders(y^3+T^4+1);
"Defining global variables: F, o, oi, one"
kash> AlffOrderPoly(o);
y^3 + T^4 + 1
kash> AlffOrderPoly(oi);
x^3 + (T^4 + 1)/T^6
```

NAME	AlffOrderReduce
PURPOSE	Given an order of a global function field, the function returns an order with a 0-reduced basis.
SYNTAX	<pre>o2 := AlffOrderReduce(o1); global function field order o2 global function field order o1</pre>
DESCRIPTION	The infinite place of $k(T)$ has to be tamely ramified in F . See [Sch96] for definitions and algorithms.
SEE ALSO	AlffOrderL0 , AlffOrderBasisValues ,
EXAMPLE	

```
kash> AlffInit(FF(5,2));
"Defining global variables: k, w, kT, kTf, kTy, T, y, AlffGlobals"
kash> AlffOrders(y^3+T^13*y^2+T^4+T+1);
"Defining global variables: F, o, oi, one"
kash> o;
Finite maximal order of
Algebraic function field defined by
$.1^3 + $.1^2*$.2^13 + $.2^4 + $.2 + 1
over
Univariate rational function field over GF(5^2)
Variables: T
given by transformation matrix
[  T + 3      0      3]
[      0    T + 3      4]
[      0      0      1]
with denominator T + 3
kash> AlffOrderReduce(o);
Finite maximal order of
Algebraic function field defined by
$.1^3 + $.1^2*$.2^13 + $.2^4 + $.2 + 1
over
Univariate rational function field over GF(5^2)
Variables: T
given by transformation matrix
[      T + 3      3*T + 4      4*T + 3]
[      0      T^14 + 3*T^13      3*T^14 + T^13 + 4]
```

[0 T + 3 3*T + 1]
with denominator T + 3

NAME	AlffOrderTransformationMatrix
PURPOSE	Returns a list containing information about the transformation from the alff-suborder of the given alfforder to the given alfforder.
SYNTAX	<pre>L := AlffOrderTransformationMatrix(0);</pre> <pre>list L</pre> <pre>alfforder 0</pre>
DESCRIPTION	<p>The list L contains the denominator d and the $n \times n$-matrix T with</p> $(\alpha 1, \dots, \alpha n) = \frac{1}{d}(a 1, \dots, a n)T,$ <p>where $\alpha 1, \dots, \alpha n$ is a basis of the given alfforder \mathcal{O} and $a 1, \dots, a n$ is a basis of the alffsuborder of the given order. If the order \mathcal{O} is the equation order, simply $d = 1$ and $T = \text{id}$ are returned.</p>

NAME	AlffOrders
PURPOSE	Initializes variables for some structures of an algebraic function field.
SYNTAX	<p>AlffOrders(f);</p> <p>polynomial in $k[T][y]$ f</p>
DESCRIPTION	<p>This function is used for convenience to initialize variables for some structures of an algebraic function field and is not recommended for use in programming. It initializes:</p> <ul style="list-style-type: none"> • F, the algebraic function field generated by $f(T, y) = 0$, • o, the integral closure of $k[T]$ in F, • oi, the integral closure of the degree valuation ring, • one, the $1 \in F$. <p>The polynomial f is checked to be irreducible and separable.</p>
SEE ALSO	AlffInit , Alff ,

EXAMPLE

```

kash> AlffInit(FF(5,2));
"Defining global variables: k, w, kT, kTf, kTy, T, y, AlffGlobals"
kash> AlffOrders(y^3+T^3+2);
"Defining global variables: F, o, oi, one"
kash> F;
Algebraic function field defined by
$.1^3 + $.2^3 + 2
over
Univariate rational function field over GF(5^2)
Variables: T

kash> o;
Finite maximal order of
Algebraic function field defined by
$.1^3 + $.2^3 + 2
over
Univariate rational function field over GF(5^2)
Variables: T

```

```
kash> AlffGlobals.oi;  
Infinite maximal order of  
Algebraic function field defined by  
$.1^3 + $.2^3 + 2  
over  
Univariate rational function field over GF(5^2)  
Variables: T  
  
kash> one;  
[ 1, 0, 0 ]
```

NAME	AlffOrders
PURPOSE	Initializes variables for some structures of an algebraic function field.
SYNTAX	<p><code>AlffOrders(f);</code></p> <p>polynomial in $k[T][y]$ <code>f</code></p>
DESCRIPTION	<p>This function is used for convenience to initialize variables for some structures of an algebraic function field and is not recommended for use in programming. It initializes:</p> <ul style="list-style-type: none"> • <code>F</code>, the algebraic function field generated by $f(T, y) = 0$, • <code>o</code>, the integral closure of $k[T]$ in F, • <code>oi</code>, the integral closure of the degree valuation ring, • <code>one</code>, the $1 \in F$. <p>The polynomial f is checked to be irreducible and separable.</p>
SEE ALSO	AlffInit , Alff ,

EXAMPLE

```

kash> AlffInit(FF(5,2));
"Defining global variables: k, w, kT, kTf, kTy, T, y, AlffGlobals"
kash> AlffOrders(y^3+T^3+2);
"Defining global variables: F, o, oi, one"
kash> F;
Algebraic function field defined by
$.1^3 + $.2^3 + 2
over
Univariate rational function field over GF(5^2)
Variables: T

kash> o;
Finite maximal order of
Algebraic function field defined by
$.1^3 + $.2^3 + 2
over
Univariate rational function field over GF(5^2)
Variables: T

```

```
kash> AlffGlobals.oi;  
Infinite maximal order of  
Algebraic function field defined by  
$.1^3 + $.2^3 + 2  
over  
Univariate rational function field over GF(5^2)  
Variables: T  
  
kash> one;  
[ 1, 0, 0 ]
```


NAME	AlffPlaceAlff
PURPOSE	Given a place this function returns the algebraic function field it is belonging to.
SYNTAX	<pre>F := AlffPlaceAlff(P);</pre> <div> algebraic function field F alff place P </div>
SEE ALSO	AlffOrderAlff ,

EXAMPLE

```
kash> AlffInit(FF(2,3));
"Defining global variables: k, w, kT, kTf, kTy, T, y, AlffGlobals"
kash> F := Alff(y^3+T^3*y+T);
Algebraic function field defined by
$.1^3 + $.1*$.2^3 + $.2
over
Univariate rational function field over GF(2^3)
Variables: T

kash> P := AlffPlaceSplit(F, T);
[ Alff place < [ T, 0, 0 ], [ 0, 1, 0 ] > ]
kash> AlffPlaceAlff(P[1]);
Algebraic function field defined by
$.1^3 + $.1*$.2^3 + $.2
over
Univariate rational function field over GF(2^3)
Variables: T
```

NAME	AlffPlaceBeta
PURPOSE	Return an inverse prime element for a place.
SYNTAX	<pre>b := AlffPlaceBeta(P);</pre> <pre> alff element b alff place P </pre>
DESCRIPTION	<p>Let a finite (infinite) place \mathfrak{P} of the algebraic function field F/k be given and let p be the minimum of \mathfrak{P}. This function returns an element $\beta \in F$ such that β/p is integral at all finite (infinite) places different from \mathfrak{P} and satisfies $v_{\mathfrak{P}}(\beta/p) = -1$.</p>
SEE ALSO	AlffEltValuation , AlffPlacePrimeElt , AlffPlaceMin ,
EXAMPLE	

```

kash> AlffInit(FF(2,1));
"Defining global variables: k, w, kT, kTf, kTy, T, y, AlffGlobals"
kash> E := Alff(y^2+y+T^3+T);
Algebraic function field defined by
$.1^2 + $.1 + $.2^3 + $.2
over
FunctionField(GF(2))

kash> P := AlffPlacesDegOne(E)[2];
Alff place < [ T, 0 ], [ 0, 1 ] >
kash> p := AlffPlaceMin(P);
T
kash> b := AlffPlaceBeta(P);
[ 1, 1 ]
kash> AlffDivisor(b/p);
Alff divisor
[ [ Alff place < [ T, 0 ], [ 0, 1 ] >, -1 ],
[ Alff place < [ T + 1, 0 ], [ 1, 1 ] >, 2 ],
[ Alff place < [ 1/T, 0 ], [ 0, 1 ] >, -1 ] ]

```

NAME	AlffPlaceDeg
PURPOSE	Computes the degree of a place.
SYNTAX	<pre>d := AlffPlaceDeg(P); integer d alff place P</pre>
DESCRIPTION	This function returns the degree d of a place \mathfrak{P} over the constant field of definition (not the exact constant field) of the algebraic function field.
SEE ALSO	AlffPlaceSplit , AlffPlaceRam , AlffPlaceResDeg ,
EXAMPLE	

```
kash> AlffInit(FF(3,4));
"Defining global variables: k, w, kT, kTf, kTy, T, y, AlffGlobals"
kash> F := Alff(y^7+y+T^9+T+1);
Algebraic function field defined by
$.1^7 + $.1 + $.2^9 + $.2 + 1
over
Univariate rational function field over GF(3^4)
Variables: T

kash> l := AlffPlaceSplit(F, T);
[ Alff place < [ T, 0, 0, 0, 0, 0, 0, 0 ], [ 2, 1, 0, 0, 0, 0, 0, 0 ] >,
  Alff place < [ T, 0, 0, 0, 0, 0, 0, 0 ], [ w^50, w^50, w^10, 1, 0, 0, 0, 0 ] >,
  Alff place < [ T, 0, 0, 0, 0, 0, 0, 0 ], [ w^70, w^70, w^30, 1, 0, 0, 0, 0 ] > ]
kash> d := AlffPlaceDeg(l[2]);
3
```

NAME	AlffPlaceIdeal
PURPOSE	missing shortdoc
SYNTAX	<pre>I := AlffPlaceIdeal(P); alff order ideal I alff place P</pre>
DESCRIPTION	Given a place \mathfrak{P} this function returns a prime ideal corresponding to \mathfrak{P} .
SEE ALSO	AlffPlaceOrder , AlffPlaceSplit , AlffIdealFactor ,
EXAMPLE	

```
kash> AlffInit(FF(2,1));
"Defining global variables: k, w, kT, kTf, kTy, T, y, AlffGlobals"
kash> E := Alff(y^2+y+T^3+T);
Algebraic function field defined by
$.1^2 + $.1 + $.2^3 + $.2
over
FunctionField(GF(2))

kash> P := AlffPlacesDegOne(E)[2];
Alff place < [ T, 0 ], [ 0, 1 ] >
kash> I := AlffPlaceIdeal(P);
< [ T, 0 ], [ 0, 1 ] >
```

NAME	AlffPlaceIsFinite
PURPOSE	Returns whether a place is finite.
SYNTAX	<pre>b := AlffPlaceIsFinite(P);</pre> <p> boolean b alff place P </p>
DESCRIPTION	<p>Let $F = k(T, y)$ be an algebraic function field defined by $f(T, y) = 0$ over k and $\mathfrak{P} \mathfrak{p}$ be a place of F over a place \mathfrak{p} of the rational function field $k(T)$. The function returns true if and only if \mathfrak{p} is not the place at infinity (degree valuation).</p>
SEE ALSO	AlffPlaceSplit ,
EXAMPLE	<pre>kash> H := AlffHermitianFunField(2,1); Algebraic function field defined by \$.1^2 + \$.1 + \$.2^3 over Univariate rational function field over GF(2^2) Variables: T kash> P := AlffPlacesDegOne(H)[1]; Alff place < [1/T, 0], [0, 1] > kash> AlffPlaceIsFinite(P); false</pre>

NAME	AlffPlaceMin
PURPOSE	Returns the “minimum” of a place.
SYNTAX	$\mathfrak{p} := \text{AlffPlaceMin}(P);$ <div style="display: flex; justify-content: space-between; margin-top: 10px;"> <div style="text-align: left;"> prime polynomial or $1/T$ alff place </div> <div style="text-align: right;"> \mathfrak{p} P </div> </div>
DESCRIPTION	Let $F = k(T, y)$ be an algebraic function field defined by $f(T, y) = 0$ over k and $\mathfrak{P} \mathfrak{p}$ be a place of F over a place \mathfrak{p} of the rational function field $k(T)$. This function returns a monic prime polynomial in $k[T]$ or $1/T$, corresponding to \mathfrak{p} .
SEE ALSO	AlffPlaceSplit ,
EXAMPLE	

```

kash> AlffInit(FF(5,2));
"Defining global variables: k, w, kT, kTf, kTy, T, y, AlffGlobals"
kash> AlffOrders(y^3+T^3+1);
"Defining global variables: F, o, oi, one"
kash> P := AlffPlaceSplit(F, T+3)[1];
Alff place < [ T + 3, 0, 0 ], [ w^4, 1, 0 ] >
kash> AlffPlaceMin(P);
T + 3

```

NAME	AlffPlaceOrder
PURPOSE	missing shortdoc
SYNTAX	<pre>o := AlffPlaceOrder(P); alff order o alff place P</pre>
DESCRIPTION	This function returns the finite or infinite maximal order according to whether the given place is finite or infinite.
SEE ALSO	AlffPlaceIdeal , AlffPlaceIsFinite , Alff ,
EXAMPLE	

```
kash> AlffInit(FF(2,1));
"Defining global variables: k, w, kT, kTf, kTy, T, y, AlffGlobals"
kash> E := Alff(y^2+y+T^3+T);
Algebraic function field defined by
$.1^2 + $.1 + $.2^3 + $.2
over
FunctionField(GF(2))

kash> P := AlffPlacesDegOne(E)[1];
Alff place < [ 1/T, 0 ], [ 0, 1 ] >
kash> o := AlffPlaceOrder(P);
Infinite maximal order of
Algebraic function field defined by
$.1^2 + $.1 + $.2^3 + $.2
over
FunctionField(GF(2))
```

NAME	AlffPlacePrimeElt
PURPOSE	missing shortdoc
SYNTAX	<pre>u := AlffPlacePrimeElt(P); alff element u alff place P</pre>
DESCRIPTION	Given a place \mathfrak{P} this function returns a prime element (local uniformizer) for \mathfrak{P} .
SEE ALSO	AlffEltValuation , AlffPlaceBeta ,
EXAMPLE	

```
kash> AlffInit(FF(2,1));
"Defining global variables: k, w, kT, kTf, kTy, T, y, AlffGlobals"
kash> E := Alff(y^2+y+T^3+T);
Algebraic function field defined by
$.1^2 + $.1 + $.2^3 + $.2
over
FunctionField(GF(2))

kash> P := AlffPlacesDegOne(E)[2];
Alff place < [ T, 0 ], [ 0, 1 ] >
kash> AlffPlacePrimeElt(P);
[ T, 0 ]
```


NAME	AlffPlaceRam
PURPOSE	Ramification of a place.
SYNTAX	$r := \text{AlffPlaceRam}(P);$ integer r alff place P
DESCRIPTION	Let $F = k(T, y)$ be an algebraic function field defined by $f(T, y) = 0$ over k and $\mathfrak{P} \mathfrak{p}$ be a place of F over a place \mathfrak{p} of the rational function field $k(T)$. This function returns the ramification index r of \mathfrak{P} over \mathfrak{p} .
SEE ALSO	AlffPlaceSplit , AlffPlaceDeg , AlffPlaceResDeg ,
EXAMPLE	

```

kash> AlffInit(FF(2,3));
"Defining global variables: k, w, kT, kTf, kTy, T, y, AlffGlobals"
kash> F := Alff(y^3+T^3*y+T);
Algebraic function field defined by
$.1^3 + $.1*$.2^3 + $.2
over
Univariate rational function field over GF(2^3)
Variables: T

kash> P := AlffPlaceSplit(F, T+1)[1];
Alff place < [ T + 1, 0, 0 ], [ w, 1, 0 ] >
kash> r := AlffPlaceRam(P);
1

```

NAME	AlffPlaceRandom
PURPOSE	Returns a random place of degree d .
SYNTAX	<p><code>p := AlffPlaceRandom(F,d);</code></p> <p> <code>alff place</code> <code>p</code> of degree d <code>global function field</code> <code>F</code> <code>integer</code> <code>d</code> </p>
DESCRIPTION	This function returns a random place of degree d of a global function field F . It may return false though there exist such a place (try calling again).
SEE ALSO	AlffPlaces , AlffPlacesDegOne ,
EXAMPLE	

```

kash> AlffInit(FF(2, 3));
"Defining global variables: k, w, kT, kTf, kTy, T, y, AlffGlobals"
kash> AlffOrders(y^3+T^3*y+T);
"Defining global variables: F, o, oi, one"
kash> AlffPlaceRandom(F, 2);
Alff place < [ T^2 + w^4*T + w, 0, 0 ], [ w^2*T + w^6, 1, 0 ] >

```

NAME	AlffPlaceResDeg
PURPOSE	Residue class field extension degree of a place.
SYNTAX	<pre>f := AlffPlaceResDeg(P); integer f alff place P</pre>
DESCRIPTION	<p>Let $F = k(T, y)$ be an algebraic function field defined by $f(T, y) = 0$ over k and $\mathfrak{P} \mathfrak{p}$ be a place of F over a place \mathfrak{p} of the rational function field $k(T)$. This function returns the degree f of the extension of the residue class fields of \mathfrak{P} and \mathfrak{p} (k is the constant field of definition, not the exact constant field).</p>
SEE ALSO	AlffPlaceSplit , AlffPlaceRam , AlffPlaceDeg ,
EXAMPLE	

```
kash> AlffInit(FF(2,3));
"Defining global variables: k, w, kT, kTf, kTy, T, y, AlffGlobals"
kash> F := Alff(y^3+T^3*y+T);
Algebraic function field defined by
$.1^3 + $.1*$.2^3 + $.2
over
Univariate rational function field over GF(2^3)
Variables: T

kash> P := AlffPlaceSplit(F, T+1)[1];
Alff place < [ T + 1, 0, 0 ], [ w, 1, 0 ] >
kash> f := AlffPlaceResDeg(P);
1
```

NAME	AlffPlaceResField
PURPOSE	Return the residue field of a place.
SYNTAX	$K := \text{AlffPlaceResField}(P);$ <div style="display: flex; justify-content: space-between;"> <div>field</div> <div>K</div> <div>residue field of P</div> </div> <div style="display: flex; justify-content: space-between;"> <div>alff place</div> <div>P</div> </div>
DESCRIPTION	Let F/k be an algebraic function field and P be a place of F/k . If \mathfrak{o}_P is the ring of algebraic functions defined at P then $k(P) := \mathfrak{o}_P/P$ is the residue class field of P . It is returned by this function.
SEE ALSO	AlffResFieldEltLift , AlffELtToResField , AlffEltValuation ,
EXAMPLE	

```

kash> AlffInit(FF(2,3));
"Defining global variables: k, w, kT, kTf, kTy, T, y, AlffGlobals"
kash> F := Alff(y^3+T^3*y+T);
Algebraic function field defined by
$.1^3 + $.1*$.2^3 + $.2
over
Univariate rational function field over GF(2^3)
Variables: T

kash> P := AlffPlaceSplit(F, T+1)[1];
Alff place < [ T + 1, 0, 0 ], [ w, 1, 0 ] >
kash> f := AlffPlaceResField(P);
Finite field of size 2^3

```

NAME	AlffPlaceSplit
PURPOSE	Decompose a rational function field place.
SYNTAX	$L := \text{AlffPlaceSplit}(F, p);$ <p>list L of places \mathfrak{P} above \mathfrak{p} algebraic function field F prime polynomial or $1/T$ p</p>
DESCRIPTION	Let $F = k(T, y)$ be an algebraic function field defined by $f(T, y) = 0$ over k . This function computes the places \mathfrak{P} of F lying above a place \mathfrak{p} of $k(T)$ given by a prime polynomial of $k[T]$ or $1/T$, the later specifying the place at infinity (degree valuation). A list containing the places is returned.
SEE ALSO	AlffPlaceRam , AlffPlaceDeg , AlffPlaceResDeg , AlffEltValuation ,
EXAMPLE	

```

kash> AlffInit(FF(2,3));
"Defining global variables: k, w, kT, kTf, kTy, T, y, AlffGlobals"
kash> F := Alff(y^3+T^3*y+T);
Algebraic function field defined by
$.1^3 + $.1*$.2^3 + $.2
over
Univariate rational function field over GF(2^3)
Variables: T

kash> L := AlffPlaceSplit(F, 1/T);
[ Alff place < [ 1/T, 0, 0 ], [ w^4/T, w^3/T, (w^4*T + w^4)/T ] >,
  Alff place < [ 1/T, 0, 0 ], [ (w^4*T + 1)/T, (w*T + w^2)/T, (w^4*T + w^6)/T \
] > ]
kash> L := AlffPlaceSplit(F, T+1);
[ Alff place < [ T + 1, 0, 0 ], [ w, 1, 0 ] >,
  Alff place < [ T + 1, 0, 0 ], [ w^2, 1, 0 ] >,
  Alff place < [ T + 1, 0, 0 ], [ w^4, 1, 0 ] > ]

```

NAME	AlffPlaceSplitType						
PURPOSE	Returns a list with ramification indices and relative degrees of all pairwise distinct places lying over a rational function field place.						
SYNTAX	<pre>L := AlffPlaceSplitType(F, p);</pre> <table><tr><td>list</td><td>L</td></tr><tr><td>algebraic function field</td><td>F</td></tr><tr><td>polynomial or $1/T$</td><td>p</td></tr></table>	list	L	algebraic function field	F	polynomial or $1/T$	p
list	L						
algebraic function field	F						
polynomial or $1/T$	p						
SEE ALSO	AlffPlaceSplit , AlffPlaceRam , AlffPlaceResDeg , AlffSignature , AlffIdealFactor ,						
EXAMPLE							

NAME	AlffPlaceSplitType
PURPOSE	Returns a list with ramification indices and relative degrees of all pairwise distinct places lying over a rational function field place.
SYNTAX	<pre>L := AlffPlaceSplitType(F, p);</pre> <div> <div>list</div> <div>algebraic function field</div> <div>polynomial or $1/T$</div> </div> <div> <div>L</div> <div>F</div> <div>p</div> </div>
SEE ALSO	AlffPlaceSplit , AlffPlaceRam , AlffPlaceResDeg , AlffSignature , AlffIdealFactor ,
EXAMPLE	

NAME	AlffPlaces
PURPOSE	Computes places of a global function field.
SYNTAX	$L := \text{AlffPlaces}(F, d);$ <div style="display: flex; justify-content: space-between;"> <div> list global function field integer </div> <div> L of alff places F d </div> </div>
DESCRIPTION	This function returns a list of all places of a degree d of a global function field F . The constant field of definition should be the exact constant field of F .
SEE ALSO	AlffPlacesNum , AlffPlacesDegOne , AlffPlacesDegOneNum ,
EXAMPLE	

```

kash> AlffInit(FF(2, 3));
"Defining global variables: k, w, kT, kTf, kTy, T, y, AlffGlobals"
kash> AlffOrders(y^3+T^3*y+T);
"Defining global variables: F, o, oi, one"
kash> AlffPlaces(F, 2);
[ Alff place < [ T^2 + w^5*T + w^3, 0, 0 ], [ w^6*T + w^4, 1, 0 ] >,
  Alff place < [ T^2 + w^3*T + w^6, 0, 0 ], [ w^5*T + w, 1, 0 ] >,
  Alff place < [ T^2 + w^6*T + w^5, 0, 0 ], [ w^3*T + w^2, 1, 0 ] >,
  Alff place < [ T^2 + w^4*T + w, 0, 0 ], [ w^2*T + w^6, 1, 0 ] >,
  Alff place < [ T^2 + T + 1, 0, 0 ], [ T + 1, 1, 0 ] >,
  Alff place < [ T^2 + w*T + w^2, 0, 0 ], [ w^4*T + w^5, 1, 0 ] >,
  Alff place < [ T^2 + w^2*T + w^4, 0, 0 ], [ w*T + w^3, 1, 0 ] > ]

```


NAME	AlffPlacesDegOne
PURPOSE	Compute all places of degree one of a global function field.
SYNTAX	<pre>L := AlffPlacesDegOne(F);</pre> <p>global function field F list L of alff places</p>
DESCRIPTION	This function returns a list of all places of degree one of a global function field F . The constant field of definition must be the exact constant field of F .
SEE ALSO	AlffPlacesDegMNum ,
EXAMPLE	

```
kash> AlffInit(FF(2, 3));
"Defining global variables: k, w, kT, kTf, kTy, T, y, AlffGlobals"
kash> AlffOrders(y^3+T^3*y+T);
"Defining global variables: F, o, oi, one"
kash> AlffPlacesDegOne(F);
[ Alff place < [ 1/T, 0, 0 ], [ w^3/T, w, (w^2*T + w^6)/T ] >,
  Alff place < [ 1/T, 0, 0 ], [ (T + w)/T, (w*T + w^4)/T, (T + w^6)/T ] >,
  Alff place < [ T, 0, 0 ], [ 0, 1, 0 ] >,
  Alff place < [ T + w, 0, 0 ], [ 1, 1, 0 ] >,
  Alff place < [ T + w, 0, 0 ], [ w^2, 1, 0 ] >,
  Alff place < [ T + w, 0, 0 ], [ w^6, 1, 0 ] >,
  Alff place < [ T + w^2, 0, 0 ], [ 1, 1, 0 ] >,
  Alff place < [ T + w^2, 0, 0 ], [ w^4, 1, 0 ] >,
  Alff place < [ T + w^2, 0, 0 ], [ w^5, 1, 0 ] >,
  Alff place < [ T + w^3, 0, 0 ], [ w^2, 1, 0 ] >,
  Alff place < [ T + w^3, 0, 0 ], [ w^3, 1, 0 ] >,
  Alff place < [ T + w^3, 0, 0 ], [ w^5, 1, 0 ] >,
  Alff place < [ T + w^4, 0, 0 ], [ 1, 1, 0 ] >,
  Alff place < [ T + w^4, 0, 0 ], [ w, 1, 0 ] >,
  Alff place < [ T + w^4, 0, 0 ], [ w^3, 1, 0 ] >,
  Alff place < [ T + w^5, 0, 0 ], [ w, 1, 0 ] >,
  Alff place < [ T + w^5, 0, 0 ], [ w^5, 1, 0 ] >,
  Alff place < [ T + w^5, 0, 0 ], [ w^6, 1, 0 ] >,
  Alff place < [ T + w^6, 0, 0 ], [ w^3, 1, 0 ] >,
  Alff place < [ T + w^6, 0, 0 ], [ w^4, 1, 0 ] >,
  Alff place < [ T + w^6, 0, 0 ], [ w^6, 1, 0 ] >,

```

```
Alff place < [ T + 1, 0, 0 ], [ w, 1, 0 ] >,
Alff place < [ T + 1, 0, 0 ], [ w^2, 1, 0 ] >,
Alff place < [ T + 1, 0, 0 ], [ w^4, 1, 0 ] > ]
```

NAME	AlffPlacesDegOne
PURPOSE	Computes all places of degree one of a global function field.
SYNTAX	<pre>L := AlffPlacesDegOne(F);</pre> <p>global function field F list L of alff places of degree 1</p>
DESCRIPTION	This function returns a list of all places of degree one of a global function field F . The constant field of definition should be the exact constant field of F since the degree is taken over the former.
SEE ALSO	AlffPlacesDegOneNum , AlffPlaces , AlffPlacesNum , AlffPlaceRandom ,
EXAMPLE	

```
kash> AlffInit(FF(2, 3));
"Defining global variables: k, w, kT, kTf, kTy, T, y, AlffGlobals"
kash> AlffOrders(y^3+T^3*y+T);
"Defining global variables: F, o, oi, one"
kash> AlffPlacesDegOne(F);
[ Alff place < [ 1/T, 0, 0 ], [ w^3/T, w, (w^2*T + w^6)/T ] >,
  Alff place < [ 1/T, 0, 0 ], [ (T + w)/T, (w*T + w^4)/T, (T + w^6)/T ] >,
  Alff place < [ T, 0, 0 ], [ 0, 1, 0 ] >,
  Alff place < [ T + w, 0, 0 ], [ 1, 1, 0 ] >,
  Alff place < [ T + w, 0, 0 ], [ w^2, 1, 0 ] >,
  Alff place < [ T + w, 0, 0 ], [ w^6, 1, 0 ] >,
  Alff place < [ T + w^2, 0, 0 ], [ 1, 1, 0 ] >,
  Alff place < [ T + w^2, 0, 0 ], [ w^4, 1, 0 ] >,
  Alff place < [ T + w^2, 0, 0 ], [ w^5, 1, 0 ] >,
  Alff place < [ T + w^3, 0, 0 ], [ w^2, 1, 0 ] >,
  Alff place < [ T + w^3, 0, 0 ], [ w^3, 1, 0 ] >,
  Alff place < [ T + w^3, 0, 0 ], [ w^5, 1, 0 ] >,
  Alff place < [ T + w^4, 0, 0 ], [ 1, 1, 0 ] >,
  Alff place < [ T + w^4, 0, 0 ], [ w, 1, 0 ] >,
  Alff place < [ T + w^4, 0, 0 ], [ w^3, 1, 0 ] >,
  Alff place < [ T + w^5, 0, 0 ], [ w, 1, 0 ] >,
  Alff place < [ T + w^5, 0, 0 ], [ w^5, 1, 0 ] >,
  Alff place < [ T + w^5, 0, 0 ], [ w^6, 1, 0 ] >,
  Alff place < [ T + w^6, 0, 0 ], [ w^3, 1, 0 ] >,
  Alff place < [ T + w^6, 0, 0 ], [ w^4, 1, 0 ] >
```

```
Alff place < [ T + w^6, 0, 0 ], [ w^6, 1, 0 ] >,
Alff place < [ T + 1, 0, 0 ], [ w, 1, 0 ] >,
Alff place < [ T + 1, 0, 0 ], [ w^2, 1, 0 ] >,
Alff place < [ T + 1, 0, 0 ], [ w^4, 1, 0 ] > ]
```

NAME	AlffPlacesDegOneNonSingFiniteNum
PURPOSE	Computes the number of finite non singular places of degree one of a global function field's constant field extension of degree m.
SYNTAX	<pre>b := AlffPlacesDegOneNonSingFiniteNum(F, m); global function field F integers b,m</pre>
SEE ALSO	AlffPlacesDegOneNumBound ,
EXAMPLE	

```
kash> AlffInit(FF(7,2));
"Defining global variables: k, w, kT, kTf, kTy, T, y, AlffGlobals"
kash> AlffOrders(y^7+y+T^4+1);
"Defining global variables: F, o, oi, one"
kash> AlffPlacesDegOneNonSingFiniteNum(F, 1);
175
```

NAME	AlffPlacesDegOneNum
PURPOSE	Computes the number of places of degree one of a global function field's constant field extension.
SYNTAX	$N := \text{AlffPlacesDegOneNum}(F, m);$ <p> global function field F integer N, m </p>
DESCRIPTION	Let $N_1(F')$ be the number of places of degree one of the constant field extension F' of degree m of a global function field F . This function calculates $N_1(F')$ (using a simple algorithm).
SEE ALSO	AlffPlacesDegOneNumBound , AlffPlacesDegOne ,
EXAMPLE	

```

kash> AlffInit(FF(2, 3));
"Defining global variables: k, w, kT, kTf, kTy, T, y, AlffGlobals"
kash> AlffOrders(y^3+T^3*y+T);
"Defining global variables: F, o, oi, one"
kash> AlffPlacesDegOneNum(F,1);

```

NAME	AlffPlacesDegOneNum
PURPOSE	Computes the number of places of degree one of a global function field's constant field extension.
SYNTAX	<pre>N := AlffPlacesDegOneNum(F,m);</pre> <p>global function field F integer N, m</p>
DESCRIPTION	Let $N_1(F')$ be the number of places of degree one of the constant field extension F' of degree m of a global function field F . This function calculates $N_1(F')$ (using a simple algorithm).
SEE ALSO	AlffPlacesDegOneNumBound , AlffPlacesDegOne , AlffPlaces , AlffPlacesNum ,
EXAMPLE	

```
kash> AlffInit(FF(2, 3));
"Defining global variables: k, w, kT, kTf, kTy, T, y, AlffGlobals"
kash> AlffOrders(y^3+T^3*y+T);
"Defining global variables: F, o, oi, one"
kash> AlffPlacesDegOneNum(F,1);
```

24

NAME	AlffPlacesDegOneNumBound
PURPOSE	Computes a bound for the number of places of degree one of a global function field.
SYNTAX	<pre> b := AlffPlacesDegOneNumBound(F); b := AlffPlacesDegOneNumBound(q, g); global function field F integer q, g integer b </pre>
DESCRIPTION	Let $N_q(g)$ be the maximal number of places of degree one of a global function field F with genus g over \mathbb{F}_q . This function calculates the minimum of the Serre bound and the Ihara bound as an upper bound for $N_q(g)$ from the given special function field or for given q and g .
SEE ALSO	AlffSerreBound , AlffIharaBound ,
EXAMPLE	

```

kash> AlffInit(FF(7,2));
"Defining global variables: k, w, kT, kTf, kTy, T, y, AlffGlobals"
kash> AlffOrders(y^7+y+T^4+1);
"Defining global variables: F, o, oi, one"
kash> AlffGenus(F);
9
kash> AlffPlacesDegOneNumBound(F);
176
kash> AlffPlacesDegOneNumBound(5, 6);
25

```


NAME	AlffPlacesNonSpecial
PURPOSE	Computes a non special system of g distinct places for a global function field F/k of genus g .
SYNTAX	$S := \text{AlffPlacesNonSpecial}(F);$ $S := \text{AlffPlacesNonSpecial}(F, d);$ <div style="display: flex; justify-content: space-between;"> <div> <p>list</p> <p>global function field</p> <p>integer</p> </div> <div> <p>S of alff places</p> <p>F</p> <p>$d \geq 1$</p> </div> </div>
DESCRIPTION	Let g be the genus of the global function field F/k and c be the dimension of the exact constant field over the constant field of definition. This function returns a list of distinct places $\mathfrak{p}_1, \dots, \mathfrak{p}_m$ of F/k such that $\sum_{i=1}^m \deg(\mathfrak{p}_i) = cg$ and $\max_{i=1}^m \deg(\mathfrak{p}_i)$ is minimal among any such places. If a second parameter $d \in \mathbb{Z}^{\geq 1}$ is given, the maximum is forced to be $\leq d$. If there do not exist such places in either case, false is returned.
SEE ALSO	AlffPlaces , AlffDivisorLDim ,

EXAMPLE

```

kash> AlffInit(FF(5));
"Defining global variables: k, w, kT, kTf, kTy, T, y, AlffGlobals"
kash> AlffOrders(y^2 + T^17 + T + 1);
"Defining global variables: F, o, oi, one"
kash> AlffPlacesNonSpecial(F);
[ Alff place < [ 1/T, 0 ], [ 0, 1 ] >, Alff place < [ T, 0 ], [ 2, 1 ] >,
  Alff place < [ T + 3, 0 ], [ 0, 1 ] >, Alff place < [ T + 1, 0 ], [ 1, 1 ] >
  , Alff place < [ T^2 + 3*T + 4, 0 ], [ T + 4, 1 ] >,
  Alff place < [ T^2 + 4*T + 1, 0 ], [ T + 2, 1 ] > ]
kash> AlffPlacesNonSpecial(F, 1);
false

```

NAME AlffPlacesNum

PURPOSE Computes the number of places of a given degree of a global function field.

SYNTAX N := AlffPlacesNum(F, m);

 global function field F
 integers N, m

SEE ALSO [AlffPlacesDegMNum](#), [AlffPlacesDegOne](#),

EXAMPLE

```
kash> AlffInit(FF(2, 3));
"Defining global variables: k, w, kT, kTf, kTy, T, y, AlffGlobals"
kash> AlffOrders(y^3+T^3*y+T);
"Defining global variables: F, o, oi, one"
kash> AlffPlacesNum(F, 2);
```

7

NAME	AlffPlacesNum
PURPOSE	Computes the number of places of a given degree of a global function field.
SYNTAX	<pre>N := AlffPlacesNum(F, m); global function field F integers N, m</pre>
SEE ALSO	AlffPlacesDegOneNum , AlffPlaces , AlffPlacesDegOne ,
EXAMPLE	

```
kash> AlffInit(FF(2, 3));
"Defining global variables: k, w, kT, kTf, kTy, T, y, AlffGlobals"
kash> AlffOrders(y^3+T^3*y+T);
"Defining global variables: F, o, oi, one"
kash> AlffPlacesNum(F, 2);
```

7

NAME	AlffPoly
PURPOSE	missing shortdoc
SYNTAX	$f := \text{AlffPoly}(F);$ <div style="display: flex; justify-content: space-between;"> polynomial f </div> <div style="display: flex; justify-content: space-between;"> algebraic function field F </div>
DESCRIPTION	Returns defining polynomial of an algebraic function field.
SEE ALSO	Alff ,
EXAMPLE	

```

kash> AlffInit(FF(7,2));
"Defining global variables: k, w, kT, kTf, kTy, T, y, AlffGlobals"
kash> F := Alff((T^2+1)*y^3+y+T^4+1);
Algebraic function field defined by
$.1^3*$.2^2 + $.1^3 + $.1 + $.2^4 + 1
over
Univariate rational function field over GF(7^2)
Variables: T

kash> AlffPoly(F);
(T^2 + 1)*y^3 + y + T^4 + 1

```

NAME	AlffPolyIrrIsSep
PURPOSE	missing shortdoc
SYNTAX	<pre>b := AlffPolyIrrIsSep(f);</pre> <p>polynomial in T and y over field f</p> <p>boolean b</p>
DESCRIPTION	Returns whether an irreducible polynomial $f(T, y)$ over a field is separable in y .
SEE ALSO	AlffPolyIsIrreducible , AlffPolyIsIrrSep , Alff ,
EXAMPLE	

```
kash> AlffInit(FF(5,2));
"Defining global variables: k, w, kT, kTf, kTy, T, y, AlffGlobals"
kash> AlffPolyIrrIsSep(y+T);
true
kash> AlffPolyIrrIsSep(y^5+T);
false
```


NAME	AlffPolyIsIrreducible
PURPOSE	missing shortdoc
SYNTAX	<pre>b := AlffPolyIsIrreducible(f);</pre> <p>polynomial in T and y over field f</p> <p>boolean b</p>
DESCRIPTION	Returns whether a polynomial $f(T, y)$ over a field is irreducible or not.
SEE ALSO	AlffPolyIsIrrSep , Alff ,
EXAMPLE	

```
kash> AlffInit(FF(5,2));
"Defining global variables: k, w, kT, kTf, kTy, T, y, AlffGlobals"
kash> AlffPolyIsIrreducible(y+T);
true
kash> AlffPolyIsIrreducible(y^5+T);
true
```

NAME	AlffPuisseuxCoeff
PURPOSE	Returns the finite constant field of the Puiseux series field into which all completions of a global function field with respect to the infinite valuations can be embedded.
SYNTAX	$k := \text{AlffPuisseuxCoeff}(F);$ <div style="display: flex; justify-content: space-between;"> finite field k </div> <div style="display: flex; justify-content: space-between;"> global function field F </div>
DESCRIPTION	Let $f(T, y)$ be a generating irreducible equation for a global function field F , of degree n and separable in y . Let d be a minimal positive integer such that all n distinct roots of $f(T, y)$ in y can be represented in $\mathbb{F}_{q^d}((T^{-1/e}))$, where e is the least common multiple of the ramification indices of the infinite places and $p \nmid e$. The function returns \mathbb{F}_{q^d} .
SEE ALSO	AlffRoots ,
EXAMPLE	

```

kash> AlffInit(FF(5,2));
"Defining global variables: k, w, kT, kTf, kTy, T, y, AlffGlobals"
kash> AlffOrders(y^3+T^4+1);
"Defining global variables: F, o, oi, one"
kash> AlffRoots(F);
kash> AlffPuisseuxCoeff(F);
Finite field of size 5^2

```


NAME	AlffRamDivisor
PURPOSE	Returns the ramification divisor of a divisor.
SYNTAX	<p> <code>A := AlffRamDivisor(D);</code> <code>A := AlffRamDivisor(F);</code> </p> <p> <code>alff divisor A</code> the ramification divisor of D <code>alff divisor D</code> <code>alff F</code> equivalent to taking $D = 0$ </p>
DESCRIPTION	<p>Let F/k be an algebraic function field, x a separating variable and D a divisor. The ramification divisor of D is defined to be</p> $i(D)(W - D) + (W_x(D)) + \nu(dx),$ <p>where W is a canonical divisor of F/k, $W_x(D)$ is the determinant of the Wronski matrix of D with respect to x and ν is the sum of the Wronski orders of D with respect to x. It is positive and consists of the D-Weierstraß places. The constant field k is required to be exact.</p>
SEE ALSO	AlffWronskian , AlffWronskianOrders , AlffWeierstrassPlaces , AlffGapNumbers , AlffDiff , AlffDifferentiation ,
EXAMPLE	

```

kash> AlffInit(Q);
kash> AlffOrders(y^2 - (T-1)*(T-2)*(T-3)*(T-4)*(T-5)*(T-6)*(T-7));
"Defining global variables: F, o, oi, one"
kash> AlffRamDivisor(F);
Alff divisor
[ [ Alff place < [ 1/T, 0 ], [ 0, 1 ] >, 3 ],
[ Alff place < [ T - 7, 0 ], [ 0, 1 ] >, 3 ],
[ Alff place < [ T - 6, 0 ], [ 0, 1 ] >, 3 ],
[ Alff place < [ T - 5, 0 ], [ 0, 1 ] >, 3 ],
[ Alff place < [ T - 4, 0 ], [ 0, 1 ] >, 3 ],
[ Alff place < [ T - 3, 0 ], [ 0, 1 ] >, 3 ],
[ Alff place < [ T - 2, 0 ], [ 0, 1 ] >, 3 ],
[ Alff place < [ T - 1, 0 ], [ 0, 1 ] >, 3 ] ]

```

NAME	AlffRegulator
PURPOSE	Computes the regulator of a global function field's finite maximal order's unit group.
SYNTAX	<pre>R := AlffRegulator(o);</pre> <p>integer R finite maximal order o</p>
DESCRIPTION	For $1 \leq i \leq s-1$ let ν_i denote distinct normalized valuations of F at the infinite places with degrees f_i over the degree valuation of $k(T)$. Let $M \in \mathbb{Z}^{(s-1) \times (s-1)}$ be the image of a basis of the finite maximal order's unit group in the "logarithm space" obtained by applying the ν_i for $1 \leq i \leq s-1$ on each basis element. The function returns $\det(M) \prod_{i=1}^r f_i$.
SEE ALSO	AlffSignature , AlffUnitRank , AlffUnitsFund ,
EXAMPLE	

```
kash> AlffInit(FF(5,1));
"Defining global variables: k, w, kT, kTf, kTy, T, y, AlffGlobals"
kash> AlffOrders(y^4+(2*T+3)*y^3+y^2+(3*T+2)*y+1);
"Defining global variables: F, o, oi, one"
kash> AlffUnitsFund(o);
[ [ 1, 1, 0, 0 ], [ 1, 4*T, 4*T + 3, 2 ], [ 3*T + 1, 1, 2*T + 3, 1 ] ]
kash> AlffRegulator(o);
1
```

NAME	AlffResFieldEltLift
PURPOSE	Lifts a residue field element.
SYNTAX	$a := \text{AlffResFieldEltLift}(b, P);$ alff element a in \mathfrak{o}_P field element b value of a at P to be lifted alff place P
DESCRIPTION	Let F/k be an algebraic function field and P be a place of F/k . If \mathfrak{o}_P is the ring of algebraic functions defined at P then $k(P) := \mathfrak{o}_P/P$ is the residue class field of P . For $b \in k(P)$ this function returns $a \in \mathfrak{o}_P$ such that $a(P) = b$. Constants are lifted to constants.
SEE ALSO	AlffResFieldEltLift , AlffELtToResField , AlffEltValuation ,

EXAMPLE

```

kash> AlffInit(FF(5,3));
"Defining global variables: k, w, kT, kTf, kTy, T, y, AlffGlobals"
kash> F := Alff(y^3+T^3*y+T);
Algebraic function field defined by
$.1^3 + $.1*$.2^3 + $.2
over
Univariate rational function field over GF(5^3)
Variables: T

kash> P := AlffPlaceSplit(F, T+1)[1];
Alff place < [ T + 1, 0, 0 ], [ 3, 1, 0 ] >
kash> AlffPlaceResField(P);
Finite field of size 5^3
kash> of := AlffOrderMaxFinite(F);
Finite maximal order of
Algebraic function field defined by
$.1^3 + $.1*$.2^3 + $.2
over
Univariate rational function field over GF(5^3)
Variables: T

kash> a := AlffEltToResField(AlffElt(of,T),P);
4

```

```

kash> AlffResFieldEltLift(a,P);
[ 4, 0, 0 ]
kash> p := AlffPlaceSplit(F, 1/T)[1];
Alff place < [ 1/T, 0, 0 ], [ w^48/T, (w^102*T + w^5)/T, (w^18*T + w^82)/T ] >
kash> AlffPlaceResField(p);
Finite field of size 5^3
kash> oi := AlffOrderMaxInfty(F);
Infinite maximal order of
Algebraic function field defined by
$.1^3 + $.1*$.2^3 + $.2
over
Univariate rational function field over GF(5^3)
Variables: T
given by transformation matrix
[1/T  0  0]
[ 0 1/T  0]
[ 0  0  1]
with denominator 1/T
kash> b := AlffEltToResField(AlffElt(oi,1/T),p);
0
kash> AlffResFieldEltLift(b,p);
[ 0, 0, 0 ]

kash> AlffInit(Q);
"Defining global variables: k, w, kT, kTf, kTy, T, y, AlffGlobals"
kash> F := Alff(y^3+T^3*y+T);
Algebraic function field defined by
$.1^3 + $.1*$.2^3 + $.2
over
Univariate rational function field over Rational Field
Variables: T

kash> P := AlffPlaceSplit(F, T+1)[1];
Alff place < [ T + 1, 0, 0 ] >
kash> AlffPlaceResField(P);
Generating polynomial: x^3 - x - 1

kash> of := AlffOrderMaxFinite(F);
Finite maximal order of
Algebraic function field defined by
$.1^3 + $.1*$.2^3 + $.2
over
Univariate rational function field over Rational Field

```

Variables: T

```
kash> a := AlffEltToResField(AlffElt(of,2),P);
```

```
2
```

```
kash> AlffResFieldEltLift(a,P);
```

```
[ 2, 0, 0 ]
```

```
kash> p := AlffPlaceSplit(F, 1/T)[1];
```

```
Alff place < [ 1/T, 0, 0 ], [ -13/T, (11/3*T - 35)/T, (-3/2*T + 32/3)/T ] >
```

```
kash> AlffPlaceResField(p);
```

Rational Field

```
kash> oi := AlffOrderMaxInfty(F);
```

Infinite maximal order of

Algebraic function field defined by

$\$.1^3 + \$.1*\$.2^3 + \$.2$

over

Univariate rational function field over Rational Field

Variables: T

given by transformation matrix

```
[1/T  0  0]
```

```
[ 0 1/T  0]
```

```
[ 0  0  1]
```

with denominator 1/T

```
kash> b := AlffEltToResField(AlffElt(oi,1/T),p);
```

```
0
```

```
kash> AlffResFieldEltLift(b,p);
```

```
[ 0, 0, 0 ]
```

NAME	AlffRootParams
PURPOSE	Sets/Displays the iteration depth of root expansions and the precision for series operations.
SYNTAX	$L := \text{AlffRootParams}(F);$ $L := \text{AlffRootParams}(F, n, m);$ <div style="display: flex; justify-content: space-between;"> <div>list</div> <div>L</div> </div> <div style="display: flex; justify-content: space-between;"> <div>global function field</div> <div>F</div> </div> <div style="display: flex; justify-content: space-between;"> <div>integer</div> <div>n</div> </div> <div style="display: flex; justify-content: space-between;"> <div>integer</div> <div>m</div> </div>
DESCRIPTION	<p>If the infinite place of $k(T)$ is tamely ramified in F, all roots of the defining polynomial f of a global function field $F/\mathbb{F}_q(T)$ can be expanded into Puiseux series in $1/T$. The root expansions are obtained via the Newton–Puiseux method which computes the roots up to a given iteration depth n. The roots are represented as a truncated Puiseux series, and all series operations in F are performed with precision at least m. If $n < 25$ ($m < 50$) it will be set to $n = 25$ ($m = 50$). These parameters for the root computation can only be changed before the roots are computed (<code>AlffRoots()</code>). For details see [Sch96].</p>
SEE ALSO	AlffRoots ,

EXAMPLE Change iteration depth and precision:

```
kash> AlffInit(FF(5,2));
"Defining global variables: k, w, kT, kTf, kTy, T, y, AlffGlobals"
kash> AlffOrders(y^3+T^4+T+1);
"Defining global variables: F, o, oi, one"
kash> AlffRootParams(F);
[ 25, 50 ]
kash> AlffRootParams(F, 50, 100);
[ 50, 100 ]
```

NAME	AlffRoots
PURPOSE	Computes the Puiseux expansions in $T^{-1/e}$ of the roots of the defining polynomial of a global function field $F/\mathbb{F}_q(T)$.
SYNTAX	$L := \text{AlffRoots}(F);$ <div style="display: flex; justify-content: space-between; margin-top: 10px;"> <div>list</div> <div>L</div> </div> <div style="display: flex; justify-content: space-between; margin-top: 10px;"> <div>global function field</div> <div>F</div> </div>
DESCRIPTION	<p>If the infinite place ∞ of $k(T)$ is tamely ramified in F, then all roots of the defining polynomial f of $F/\mathbb{F}_q(T)$ can be expanded in $T^{-1/e}$. The function returns a list</p> $L := [\rho_{1,1}, \dots, \rho_{1,e_1 f_1}, \rho_{2,1}, \dots, \rho_{s,e_s f_s}],$ <p>where $\rho_{i,j} \in \mathbb{F}_{q^d} \langle T^{-1/e_i} \rangle \subset \mathbb{F}_{q^d} \langle T^{-1/e} \rangle$, ($1 \leq j \leq e_i f_i$), are the root expansions at the s infinite places above ∞ (the e_i and f_i are the ramification indices and residue degrees of these places, e is the lcm of the e_i).</p>
SEE ALSO	AlffRootParams , InftyVal ,
EXAMPLE	Compute all roots of $y^3 + T^4 + 1$ over \mathbb{F}_{25} :

```

kash> AlffInit(FF(5,2));
"Defining global variables: k, w, kT, kTf, kTy, T, y, AlffGlobals"
kash> AlffOrders(y^3+T^4+1);
"Defining global variables: F, o, oi, one"
kash> AlffRoots(F);
[ 4*(T^(-1/3))^-4 + 3*(T^(-1/3))^8 + 4*(T^(-1/3))^20 + 2*(T^(-1/3))^56 + 4*(T^(-1/3))^68 + 2*(T^(-1/3))^80 + 2*(T^(-1/3))^116 + 4*(T^(-1/3))^128 + 2*(T^(-1/3))^140 + 4*(T^(-1/3))^176 + 3*(T^(-1/3))^188 + 4*(T^(-1/3))^200 + 4*(T^(-1/3))^296 + 3*(T^(-1/3))^308 + 4*(T^(-1/3))^320 + 2*(T^(-1/3))^356 + 4*(T^(-1/3))^368 + 2*(T^(-1/3))^380 + 2*(T^(-1/3))^416 + 4*(T^(-1/3))^428 + 2*(T^(-1/3))^440 + 4*(T^(-1/3))^476 + 3*(T^(-1/3))^488 + 4*(T^(-1/3))^500 + 2*(T^(-1/3))^1496 + 0((T^(-1/3))^1498),
  w^20*(T^(-1/3))^-4 + w^2*(T^(-1/3))^8 + w^20*(T^(-1/3))^20 + w^14*(T^(-1/3))^56 + w^20*(T^(-1/3))^68 + w^14*(T^(-1/3))^80 + w^14*(T^(-1/3))^116 + w^20*(T^(-1/3))^128 + w^14*(T^(-1/3))^140 + w^20*(T^(-1/3))^176 + w^2*(T^(-1/3))^188 + w^20*(T^(-1/3))^200 + w^20*(T^(-1/3))^296 + w^2*(T^(-1/3))^308 + w^20*(T^(-1/3))^320 + w^14*(T^(-1/3))^356 + w^20*(T^(-1/3))^368 + w^14*(T^(-1/3))^380 + w^14*(T^(-1/3))^416 + w^20*(T^(-1/3))^428 + w^14*(T^(-1/3))^440 + w^20*(T^(-1/3))^476 + w^2*(T^(-1/3))^488 + w^20*(T^(-1/3))^500 + w^14*(T^(-1/3))^1496 + 0((T^(-1/3))^1498),

```

```

w^4*(T^(-1/3))^(-4 + w^10*(T^(-1/3))^8 + w^4*(T^(-1/3))^20 + w^22*(T^(-1/3))^56 + w^4*(T^(-1/3))^68 + w^22*(T^(-1/3))^80 + w^22*(T^(-1/3))^116 + w^4*(T^(-1/3))^128 + w^22*(T^(-1/3))^140 + w^4*(T^(-1/3))^176 + w^10*(T^(-1/3))^188 + w^4*(T^(-1/3))^200 + w^4*(T^(-1/3))^296 + w^10*(T^(-1/3))^308 + w^4*(T^(-1/3))^320 + w^22*(T^(-1/3))^356 + w^4*(T^(-1/3))^368 + w^22*(T^(-1/3))^380 + w^22*(T^(-1/3))^416 + w^4*(T^(-1/3))^428 + w^22*(T^(-1/3))^440 + w^4*(T^(-1/3))^476 + w^10*(T^(-1/3))^488 + w^4*(T^(-1/3))^500 + w^22*(T^(-1/3))^1496 + O((T^(-1/3))^1498) ]

```


NAME	AlffSUnits
PURPOSE	Find the S -regulator and a basis for the free part of the S -units.
SYNTAX	<pre>L := AlffSUnits(S [, "raw"]);</pre> <p>list L the S-regulator and a basis of S-units</p> <p>list S a list of places</p>
DESCRIPTION	Let F/k be a global function field and S be a set of places of F/k . This function returns a list L such that $L[1]$ is the S -regulator and $L[2]$ is a basis for the free part of the S -unit subgroup of F^\times , consisting of principal divisors if the option "raw" is given.
SEE ALSO	AlffClassGroupGens , AlffClassGroup , AlffInit , AlffOrdersAlffDivisorReduction , AlffDivisorLargeLDim , AlffPlacesDegOne ,
EXAMPLE	

```
kash> AlffInit(FF(7));;
kash> AlffOrders(y^2 + T^3 + T + 1);;
kash> S := AlffPlacesDegOne(F);
[ Alff place < [ 1/T, 0 ], [ 0, 1 ] >, Alff place < [ T + 3, 0 ], [ 1, 1 ] >,
  Alff place < [ T + 3, 0 ], [ 6, 1 ] >, Alff place < [ T + 2, 0 ], [ 3, 1 ] >,
  Alff place < [ T + 2, 0 ], [ 4, 1 ] >,
  Alff place < [ T + 6, 0 ], [ 2, 1 ] >, Alff place < [ T + 6, 0 ], [ 5, 1 ] >,
  Alff place < [ T + 4, 0 ], [ 2, 1 ] >,
  Alff place < [ T + 4, 0 ], [ 5, 1 ] >, Alff place < [ T + 1, 0 ], [ 1, 1 ] >,
  Alff place < [ T + 1, 0 ], [ 6, 1 ] > ]
kash> AlffSUnits(S);
[ 11,
  [ [ 3*T^5 + 4*T^4 + 2*T^3 + 5*T^2 + 6*T + 4, 2*T^4 + T^3 + 3*T^2 + 3*T + 2 ] \
/ (T^11 + 5*T^10 + 5*T^9 + 3*T^8 + 4*T^7 + 3*T^4 + T^3 + T^2 + 2*T + 5),
    [ 1, 0 ] / (T + 3),
    [ 4*T^4 + 4*T^3 + 5*T^2 + 1, 4*T + 2 ] / (T^8 + 2*T^7 + 3*T + 6),
    [ 2*T^2 + 6*T + 2, 6*T + 2 ] / (T^5 + T^3 + 6*T^2 + 3*T + 1),
    [ 5*T^3 + 2*T^2 + 4*T + 2, 6*T + 5 ] / (T^6 + 5*T^4 + 5*T^3 + 2*T^2 + 6*\
T + 2),
    [ 5*T^3 + 2*T^2 + 3, 6*T^2 + 4*T + 2 ] / (T^7 + 3*T^6 + 5*T^5 + 6*T^4 + \
3*T^3 + 5*T^2 + 6*T + 6), [ 6*T^2 + T + 4, 1 ] / (T^4 + 6*T^3 + 2*T + 3),
    [ 2*T^4 + 3*T^3 + 2*T + 1, 4*T^3 + 3*T^2 + T + 1 ] / (T^9 + 5*T^7 + 3*T^6
```

```
2 + 1), [ 6, 1 ] / (T^3 + T + 2),
[ 2*T^5 + 3*T^4 + T^3 + 4*T^2 + 6*T + 6, 4*T^3 + 5*T^2 + T + 4 ] / (T^10\
+ T^8 + 2*T^7 + 3*T^3 + 3*T + 6) ] ]
```

```
kash> AlffSUnits(S, "raw");
```

```
[ 11, [ Alff divisor
[ [ Alff place < [ 1/T, 0 ], [ 0, 1 ] >, -11 ],
[ Alff place < [ T + 3, 0 ], [ 1, 1 ] >, 11 ] ]
, Alff divisor
[ [ Alff place < [ 1/T, 0 ], [ 0, 1 ] >, -2 ],
[ Alff place < [ T + 3, 0 ], [ 1, 1 ] >, 1 ],
[ Alff place < [ T + 3, 0 ], [ 6, 1 ] >, 1 ] ]
, Alff divisor
[ [ Alff place < [ 1/T, 0 ], [ 0, 1 ] >, -8 ],
[ Alff place < [ T + 3, 0 ], [ 1, 1 ] >, 7 ],
[ Alff place < [ T + 2, 0 ], [ 3, 1 ] >, 1 ] ]
, Alff divisor
[ [ Alff place < [ 1/T, 0 ], [ 0, 1 ] >, -5 ],
[ Alff place < [ T + 3, 0 ], [ 1, 1 ] >, 4 ],
[ Alff place < [ T + 2, 0 ], [ 4, 1 ] >, 1 ] ]
, Alff divisor
[ [ Alff place < [ 1/T, 0 ], [ 0, 1 ] >, -6 ],
[ Alff place < [ T + 3, 0 ], [ 1, 1 ] >, 5 ],
[ Alff place < [ T + 6, 0 ], [ 2, 1 ] >, 1 ] ]
, Alff divisor
[ [ Alff place < [ 1/T, 0 ], [ 0, 1 ] >, -7 ],
[ Alff place < [ T + 3, 0 ], [ 1, 1 ] >, 6 ],
[ Alff place < [ T + 6, 0 ], [ 5, 1 ] >, 1 ] ]
, Alff divisor
[ [ Alff place < [ 1/T, 0 ], [ 0, 1 ] >, -4 ],
[ Alff place < [ T + 3, 0 ], [ 1, 1 ] >, 3 ],
[ Alff place < [ T + 4, 0 ], [ 2, 1 ] >, 1 ] ]
, Alff divisor
[ [ Alff place < [ 1/T, 0 ], [ 0, 1 ] >, -9 ],
[ Alff place < [ T + 3, 0 ], [ 1, 1 ] >, 8 ],
[ Alff place < [ T + 4, 0 ], [ 5, 1 ] >, 1 ] ]
, Alff divisor
[ [ Alff place < [ 1/T, 0 ], [ 0, 1 ] >, -3 ],
[ Alff place < [ T + 3, 0 ], [ 1, 1 ] >, 2 ],
[ Alff place < [ T + 1, 0 ], [ 1, 1 ] >, 1 ] ]
, Alff divisor
[ [ Alff place < [ 1/T, 0 ], [ 0, 1 ] >, -10 ],
[ Alff place < [ T + 3, 0 ], [ 1, 1 ] >, 9 ],
[ Alff place < [ T + 1, 0 ], [ 6, 1 ] >, 1 ] ]
```

]]

NAME	AlffSerreBound
PURPOSE	Computes the Serre bound of a global function field.
SYNTAX	<pre> b := AlffSerreBound(F); b := AlffSerreBound(q, g); global function field F integer q, g integer b </pre>
DESCRIPTION	Let g be the genus of a function field F over the exact constant field \mathbb{F}_q . This function computes the Serre bound (improvement of the Hasse-Weil bound) for the number of places of degree one of F using the function field or for given q and g .
SEE ALSO	AlffIharaBound , AlffPlacesDegOneNumBound ,
EXAMPLE	

```

kash> AlffInit(FF(2,3));
"Defining global variables: k, w, kT, kTf, kTy, T, y, AlffGlobals"
kash> AlffOrders(y^3+T^3*y+T);
"Defining global variables: F, o, oi, one"
kash> AlffGenus(F);
3
kash> AlffSerreBound(F);
24
kash> AlffSerreBound(5, 7);
34

```

NAME	AlffSignature
PURPOSE	Returns the signature of a global function field extension $F/\mathbb{F}_q(T)$.
SYNTAX	$L := \text{AlffSignature}(F);$ <div style="margin-left: 100px;"> $\text{list} \qquad \qquad \qquad L$ $\text{global function field} \quad F$ </div>
DESCRIPTION	<p>The function returns the list $L := [[e_1, f_1], \dots, [e_s, f_s]]$ consisting of the ramification and relative degrees of the inequivalent valuations of F extending the degree valuation of $\mathbb{F}_q(T)$ ordered according to</p> $(e_i < e_j) \text{ or } (e_i = e_j \text{ and } f_i \leq f_j) \text{ for all } i < j.$

SEE ALSO [AlffPlaceSplit](#), [AlffUnitRank](#),

EXAMPLE

```
kash> AlffInit(FF(5,2));
"Defining global variables: k, w, kT, kTf, kTy, T, y, AlffGlobals"
kash> AlffOrders(y^3+T^4+1);
"Defining global variables: F, o, oi, one"
kash> AlffSignature(F);
[ [ 3, 1 ] ]
```


NAME	AlffUnitRank
PURPOSE	Returns the unit rank of a global function field extension $F/\mathbb{F}_q(T)$.
SYNTAX	<pre>r := AlffUnitRank(F);</pre> <p>integer r</p> <p>global function field F</p>
DESCRIPTION	The number $s - 1$, where s denotes the number of inequivalent valuations extending the degree valuation of $\mathbb{F}_q(T)$, is returned.
SEE ALSO	AlffSignature ,
EXAMPLE	

```
kash> AlffInit(FF(5,2));
"Defining global variables: k, w, kT, kTf, kTy, T, y, AlffGlobals"
kash> AlffOrders(y^3+T^4+1);
"Defining global variables: F, o, oi, one"
kash> AlffUnitRank(F);
0
```

NAME	AlffUnitsFund
PURPOSE	Computes $s - 1$ fundamental units of a global function fields's finite maximal order.
SYNTAX	$L := \text{AlffUnitsFund}(o);$ <div style="margin-left: 100px;"> $list \quad L$ $finite\ maximal\ order \quad o \quad \text{of a global function field}$ </div>
DESCRIPTION	<p>This function computes $s - 1$ fundamental units of the given finite maximal order of a global function field F (the integral closure of $k[T]$ in F), where s is the number of places of F at infinity. A list containing the units is returned. The infinite place of $k(T)$ must have tame ramification in F. See [Sch96] for further definitions and algorithms.</p>
SEE ALSO	AlffRegulator ,
EXAMPLE	

```

kash> AlffInit(FF(5,1));
"Defining global variables: k, w, kT, kTf, kTy, T, y, AlffGlobals"
kash> AlffOrders(y^4+(2*T+3)*y^3+y^2+(3*T+2)*y+1);
"Defining global variables: F, o, oi, one"
kash> AlffUnitsFund(o);
[ [ 1, 1, 0, 0 ], [ 1, 4*T, 4*T + 3, 2 ], [ 3*T + 1, 1, 2*T + 3, 1 ] ]

```


NAME	AlffVarT
PURPOSE	missing shortdoc
SYNTAX	$T := \text{AlffVarT}(F);$ algebraic function field F polynomial T
DESCRIPTION	Returns the indeterminate T of the generating equation of the algebraic function field.
SEE ALSO	AlffEltGenT , Alff ,
EXAMPLE	

```

kash> AlffInit(FF(7,2));
"Defining global variables: k, w, kT, kTf, kTy, T, y, AlffGlobals"
kash> F := Alff((T^2+1)*y^3+y+T^4+1);
Algebraic function field defined by
$.1^3*$.2^2 + $.1^3 + $.1 + $.2^4 + 1
over
Univariate rational function field over GF(7^2)
Variables: T

kash> AlffVarT(F);
T

```

NAME	AlffVarY
PURPOSE	missing shortdoc
SYNTAX	<pre>y := AlffVarY(F);</pre> <div> algebraic function field F polynomial y </div>
DESCRIPTION	Returns the indeterminate y of the generating equation of the algebraic function field.
SEE ALSO	AlffEltGenY , Alff ,
EXAMPLE	

```
kash> AlffInit(FF(7,2));
"Defining global variables: k, w, kT, kTf, kTy, T, y, AlffGlobals"
kash> F := Alff((T^2+1)*y^3+y+T^4+1);
Algebraic function field defined by
$.1^3*$.2^2 + $.1^3 + $.1 + $.2^4 + 1
over
Univariate rational function field over GF(7^2)
Variables: T

kash> AlffVarY(F);
y
```

NAME	AlffWeierstrassPlaces
PURPOSE	Returns the Weierstraß places of a divisor.
SYNTAX	<pre>L := AlffWeierstrassPlaces(D);</pre> <pre>L := AlffWeierstrassPlaces(F);</pre> <pre>list L containing the Weierstraß places</pre> <pre>alff divisor D</pre> <pre>alff F equivalent to taking $D = 0$</pre>
DESCRIPTION	<p>Let F/k be an algebraic function field, D a divisor and P a place of degree one. An integer $m \geq 1$ is a D-gap number of P if $\dim(D + (m-1)P) = \dim(D + mP)$ holds. The D-gap numbers m of P satisfy $1 \leq m \leq 2g - 1 - \deg(D)$ and their cardinality equals the index of speciality $i(D)$. The sequences of D-gap numbers are independent of constant field extensions for perfect k and are the same for all but a finite number of places P of degree one (consider e.g. k algebraically closed). The places P of degree one which have different sequences of D-gap numbers are called D-Weierstraß places.</p> <p>This function returns a list of all places of F/k (having not necessarily degree one) which are lying below D-Weierstraß places of $F\bar{k}/\bar{k}$ (k perfect). The constant field k is required to be exact. Remark: If the characteristic of F is positive this function is currently very slow for large genus because of <code>AlffDifferentiation()</code>.</p>
SEE ALSO	AlffGapNumbers , AlffRamDivisor , AlffWronskian , AlffWronskianOrders , AlffDiff , AlffDifferentiation ,
EXAMPLE	

```
kash> AlffInit(Q);
kash> AlffOrders(y^2 - (T-1)*(T-2)*(T-3)*(T-4)*(T-5)*(T-6)*(T-7));
"Defining global variables: F, o, oi, one"
kash> AlffWeierstrassPlaces(F);
[ Alff place < [ 1/T, 0 ], [ 0, 1 ] >, Alff place < [ T - 7, 0 ], [ 0, 1 ] >,
  Alff place < [ T - 6, 0 ], [ 0, 1 ] >, Alff place < [ T - 5, 0 ], [ 0, 1 ] >
  , Alff place < [ T - 4, 0 ], [ 0, 1 ] >,
  Alff place < [ T - 3, 0 ], [ 0, 1 ] >, Alff place < [ T - 2, 0 ], [ 0, 1 ] >
  , Alff place < [ T - 1, 0 ], [ 0, 1 ] > ]
```

NAME	AlffWronskian
PURPOSE	Returns the Wronski matrix of a Riemann-Roch space.
SYNTAX	<p>W := AlffWronskian(D);</p> <p>W := AlffWronskian(F);</p> <p>list W list of rows</p> <p>alff divisor D $\mathcal{L}(D)$ is computed</p> <p>alff F equivalent to taking $D = 0$</p>
DESCRIPTION	<p>Let F/k be an algebraic function field with separating element x and let v_1, \dots, v_l be a basis of $\mathcal{L}(D)$. For the differentiation D_x with respect to x consider the successively smallest $\nu_1 \leq \dots \leq \nu_l \in \mathbb{Z}^{\geq 0}$ such that the rows $D_x^{(\nu_i)}(v_1), \dots, D_x^{(\nu_i)}(v_l)$, $1 \leq i \leq l$ are F-linearly independent. These rows form the Wronski matrix of D with respect to x and are returned as a list of lists W. If D has dimension zero, false is returned. The constant field k is required to be exact.</p>
SEE ALSO	AlffWronskianOrders , AlffDiff , AlffDifferentiation , AlffRamDivisor ,

EXAMPLE

```

kash> AlffInit(Q);
kash> AlffOrders(y^2 - (T-1)*(T-2)*(T-3)*(T-4)*(T-5)*(T-6)*(T-7));
"Defining global variables: F, o, oi, one"
kash> AlffWronskian(AlffCanonicalDivisor(F));
[ [ [ 0, 1 ] / (T^7 - 28*T^6 + 322*T^5 - 1960*T^4 + 6769*T^3 - 13132*T^2 + 130\
68*T - 5040),
    [ 0, T ] / (T^7 - 28*T^6 + 322*T^5 - 1960*T^4 + 6769*T^3 - 13132*T^2 + 1\
3068*T - 5040),
    [ 0, T^2 ] / (T^7 - 28*T^6 + 322*T^5 - 1960*T^4 + 6769*T^3 - 13132*T^2 + \
13068*T - 5040) ],
  [ [ 0, -7/2*T^6 + 84*T^5 - 805*T^4 + 3920*T^3 - 20307/2*T^2 + 13132*T - 6534\
] / (T^14 - 56*T^13 + 1428*T^12 - 21952*T^11 + 226982*T^10 - 1667568*T^9 + 89\
62364*T^8 - 35733376*T^7 + 105994833*T^6 - 232253336*T^5 + 369120808*T^4 - 411\
449472*T^3 + 303143184*T^2 - 131725440*T + 25401600),
    [ 0, -5/2*T^7 + 56*T^6 - 483*T^5 + 1960*T^4 - 6769/2*T^3 + 6534*T - 5040\
] / (T^14 - 56*T^13 + 1428*T^12 - 21952*T^11 + 226982*T^10 - 1667568*T^9 + 89\
62364*T^8 - 35733376*T^7 + 105994833*T^6 - 232253336*T^5 + 369120808*T^4 - 411\
449472*T^3 + 303143184*T^2 - 131725440*T + 25401600),
    [ 0, -3/2*T^8 + 28*T^7 - 161*T^6 + 6769/2*T^4 - 13132*T^3 + 19602*T^2 - \

```

$10080*T] / (T^{14} - 56*T^{13} + 1428*T^{12} - 21952*T^{11} + 226982*T^{10} - 1667568*T^9 + 8962364*T^8 - 35733376*T^7 + 105994833*T^6 - 232253336*T^5 + 369120808*T^4 - 411449472*T^3 + 303143184*T^2 - 131725440*T + 25401600)]$,

$[[0, 63/8*T^{12} - 378*T^{11} + 16331/2*T^{10} - 104860*T^9 + 3562405/4*T^8 - 52\backslash 64168*T^7 + 44354149/2*T^6 - 67008956*T^5 + 1151518655/8*T^4 - 214111870*T^3 + \backslash 209157193*T^2 - 120435336*T + 30947094] / (T^{21} - 84*T^{20} + 3318*T^{19} - 8192\backslash 8*T^{18} + 1417983*T^{17} - 18282684*T^{16} + 182182984*T^{15} - 1436603616*T^{14} + 910\backslash 4502015*T^{13} - 46835050444*T^{12} + 196681551150*T^{11} - 675791033064*T^{10} + 1898\backslash 151600817*T^9 - 4340764298724*T^8 + 8021533034676*T^7 - 11838983956912*T^6 + 1\backslash 3715719388784*T^5 - 12159503504064*T^4 + 7936942375872*T^3 - 3582803508480*T^2\backslash + 995844326400*T - 128024064000)]$,

$[0, 35/8*T^{13} - 196*T^{12} + 7763/2*T^{11} - 44492*T^{10} + 1292585/4*T^9 - 1\backslash 512140*T^8 + 8504693/2*T^7 - 4475548*T^6 - 120419341/8*T^5 + 76204800*T^4 - 15\backslash 9963615*T^3 + 188151768*T^2 - 120624498*T + 32931360] / (T^{21} - 84*T^{20} + 331\backslash 8*T^{19} - 81928*T^{18} + 1417983*T^{17} - 18282684*T^{16} + 182182984*T^{15} - 14366036\backslash 16*T^{14} + 9104502015*T^{13} - 46835050444*T^{12} + 196681551150*T^{11} - 67579103306\backslash 4*T^{10} + 1898151600817*T^9 - 4340764298724*T^8 + 8021533034676*T^7 - 118389839\backslash 56912*T^6 + 13715719388784*T^5 - 12159503504064*T^4 + 7936942375872*T^3 - 3582\backslash 803508480*T^2 + 995844326400*T - 128024064000)]$,

$[0, 15/8*T^{14} - 70*T^{13} + 2051/2*T^{12} - 6076*T^{11} - 69307/4*T^{10} + 5723\backslash 20*T^9 - 9420035/2*T^8 + 22324484*T^7 - 544398673/8*T^6 + 134268134*T^5 - 1599\backslash 63615*T^4 + 85289400*T^3 + 30947094*T^2 - 65862720*T + 25401600] / (T^{21} - 84\backslash *T^{20} + 3318*T^{19} - 81928*T^{18} + 1417983*T^{17} - 18282684*T^{16} + 182182984*T^{15}\backslash - 1436603616*T^{14} + 9104502015*T^{13} - 46835050444*T^{12} + 196681551150*T^{11} - \backslash 675791033064*T^{10} + 1898151600817*T^9 - 4340764298724*T^8 + 8021533034676*T^7 \backslash - 11838983956912*T^6 + 13715719388784*T^5 - 12159503504064*T^4 + 7936942375872\backslash *T^3 - 3582803508480*T^2 + 995844326400*T - 128024064000)]]$

NAME	AlffWronskianOrders
PURPOSE	Returns the Wronski orders of a Riemann-Roch space.
SYNTAX	<p> $W := \text{AlffWronskianOrders}(D);$ $W := \text{AlffWronskianOrders}(F);$ </p> <p> <code>list</code> <code>W</code> list of rows <code>alff divisor</code> <code>D</code> $\mathcal{L}(D)$ is computed <code>alff</code> <code>F</code> equivalent to taking $D = 0$ </p>
DESCRIPTION	<p>Let F/k be an algebraic function field with separating element x and let v_1, \dots, v_l be a basis of $\mathcal{L}(D)$. For the differentiation D_x with respect to x consider the successively smallest $\nu_1 \leq \dots \leq \nu_l \in \mathbb{Z}^{\geq 0}$ such that the rows $D_x^{(\nu_i)}(v_1), \dots, D_x^{(\nu_i)}(v_l)$, $1 \leq i \leq l$ are F-linearly independent. The numbers ν_1, \dots, ν_l are the Wronski orders of D with respect to x and are returned. If D has dimension zero, the empty list is returned. The constant field k is required to be exact.</p>
SEE ALSO	AlffWronskian , AlffDiff , AlffDifferentiation , AlffRamDivisor ,
EXAMPLE	
<pre> kash> AlffInit(Q);; kash> AlffOrders(y^2 - (T-1)*(T-2)*(T-3)*(T-4)*(T-5)*(T-6)*(T-7)); "Defining global variables: F, o, oi, one" kash> AlffWronskianOrders(AlffCanonicalDivisor(F)); [0, 1, 2] </pre>	

NAME	ArcCos
PURPOSE	Returns the inverse cosine of a number.
SYNTAX	<pre>y := ArcCos(x);</pre> <pre>complex y</pre> <pre>complex x</pre>
DESCRIPTION	Given an x the function returns the inverse cosine (arccos) of x . The computation is done in the current precision of the real field.
SEE ALSO	Cos , Tan ,
EXAMPLE	

```
kash> ArcCos(.5);  
1.047197551196597746154214461093167628065723133126
```

NAME	ArcSin
PURPOSE	Returns the inverse sine of a number.
SYNTAX	<pre>y := ArcSin(x); complex y complex x</pre>
DESCRIPTION	Given an <code>x</code> the function returns the inverse sine (<code>arcsin</code>) of <code>x</code> . The computation is done in the current precision of the real field.
SEE ALSO	<code>Sin</code> , <code>Cos</code> , <code>Tan</code> ,
EXAMPLE	

```
kash> ArcSin(.5);  
0.5235987755982988730771072305465838140328615665624224
```


NAME	ArcTan
PURPOSE	Returns the inverse tangens of a number.
SYNTAX	<pre>y := ArcTan(x); complex y complex x</pre>
DESCRIPTION	Given an <code>x</code> the function returns the inverse tangens (arctan) of <code>x</code> . The computation is done in the current precision of the real field.
SEE ALSO	Sin , Cos , Tan ,
EXAMPLE	

```
kash> ArcTan(.5);  
0.4636476090008061162142562314612144020285370542861202
```

Arg

NAME Arg

PURPOSE Returns the argument of a complex number.

SYNTAX `phi := Arg(z);`

`real phi`
 `complex z`

DESCRIPTION Returns $\phi \in (-\pi, \pi]$ for $z = |z|e^{i\phi} \in \mathbb{C}$.

EXAMPLE Compute the argument of i

```
kash> z := Comp(0, 1);  
1*i  
kash> Arg(z);  
1.570796326794896619231321691639751442098584699688
```

 Compute the argument of -1

```
kash> z := Comp(-1, 0);  
-1  
kash> Arg(z);  
3.141592653589793238462643383279502884197169399375
```

NAME	BagRead
PURPOSE	Reads a bag from an open file.
SYNTAX	<pre>BagRead(file); BagRead("filename");</pre> <pre>File file</pre>
DESCRIPTION	
SEE ALSO	BagWrite ,
EXAMPLE	See BagWrite for an example.

BagWrite

NAME BagWrite

PURPOSE Writes a bag to an open file.

SYNTAX BagWrite (file, arg [,arg]); BagWrite("filename", arg [,arg]);

File file
any expression arg

DESCRIPTION

SEE ALSO **BagRead**,

EXAMPLE Open new file for writing.

```
kash> file:=Open("/tmp/dump","w");  
Filename: /tmp/dump / Mode: w / Open (fid): 5  
kash> f:=Poly(Zx,[1,0,-5]);;
```

Write data to file.

```
kash> BagWrite(file,f);
```

Close file.

```
kash> Close(file);  
true
```

Open existing file for appending.

```
kash> file:=Open("/tmp/dump","a");  
Filename: /tmp/dump / Mode: a / Open (fid): 5
```

Append data to file.

```

kash> o:=Order(f);;
kash> BagWrite(file,o);
kash> e1:=Elt(o,[1,0]/1);;
kash> e2:=Elt(o,1/1);;
kash> e3:=Elt(o,[-5,0]/1);;
kash> e4:=Elt(o,[1,1]/1);;
kash> BagWrite(file,e1,e2,e3,e4);
kash> M:=OrderMaximal(Order (o,[e1,e2,e3,e4]));;
kash> BagWrite(file,M);

```

Close file.

```

kash> Close(file);
true

```

Open existing file for reading.

```

kash> file:=Open("/tmp/dump","r");
Filename: /tmp/dump / Mode: r / Open (fid): 5

```

Now, reread the data from this file.

```

kash> BagRead(file);
[ x^2 - 5 ]
kash> BagRead(file);
[ Generating polynomial: x^2 - 5
  ]
kash> BagRead(file);
[ 1, 1, -5, [1, 1] ]
kash> BagRead(file);
[   F[1]
    |
    F[2]
    /
    /
    Q

```

BagWrite

```
F [ 1]    Given by transformation matrix
F [ 2]    x^2 - 5
Discriminant: 5
]
```

```
kash> BagRead(file);
false
```

Close file.

```
kash> Close(file);
true
```

NAME Bell

PURPOSE Sounds the terminal bell.

SYNTAX Bell();

DESCRIPTION

EXAMPLE

`Bell();`

Bernoulli

NAME Bernoulli

PURPOSE Returns a list of Bernoulli numbers.

SYNTAX `bern := Bernoulli(n);`

 integer `n`
 list `bern`

DESCRIPTION Given an integer n , the list of the rational Bernoulli numbers $0, 1, 2, 4, 6, \dots, 2n - 2$ is returned. For small n `BernoulliMagma` is much faster, but not exact and for $n \geq 130$ not computable.

SEE ALSO [BernoulliMagma](#),

EXAMPLE

```
kash> bern := Bernoulli(6);  
[ 1/1, -1/2, 1/6, -1/30, 1/42, -1/30 ]
```


NAME	BernoulliMagma
PURPOSE	Returns a list of Bernoulli numbers.
SYNTAX	<pre>bern := BernoulliMagma(n); integer n list bern</pre>
DESCRIPTION	Given an integer n , the list of Bernoulli numbers $2, \dots, 2n$ is returned. For $n \geq 130$ this function does not work, see Bernoulli instead.
SEE ALSO	Bernoulli ,
EXAMPLE	

```
kash> bern := BernoulliMagma(6);  
[ 0.1666666666666666651508421637117862701416015625,  
  -0.03333333333333333712289459072053432464599609375,  
  0.023809523809523807358345948159694671630859375,  
  -0.03333333333333333712289459072053432464599609375,  
  0.0757575757575757576205432997085154056549072265625,  
  -0.2531135531135530982282944023609161376953125 ]
```

C

NAME	C
PURPOSE	Predefined constant: Complex field \mathbb{C} .
SYNTAX	<code>C;</code> <code>ring C</code>
DESCRIPTION	This is a predefined constant for the complex field \mathbb{C} and is referenced by the variable <code>C</code> . The precision can be set or displayed with <code>Prec</code> . Default precision is 50.
SEE ALSO	<code>Z</code> , <code>Q</code> , <code>R</code> , <code>Prec</code> ,
EXAMPLE	

```
kash> C;
```

```
Complex Field of precision 52
```

NAME	Ceil
PURPOSE	Returns the minimal integer greater than or equal to a number.
SYNTAX	$y := \text{Ceil}(x);$ integer y real x
DESCRIPTION	<p>Given an x in \mathbb{Z}, \mathbb{Q} or \mathbb{R} the function returns</p> $\min\{k \in \mathbb{Z} : k \geq x\}.$ <p>The computation is done in the current precision of the real (complex) field.</p>
SEE ALSO	Trunc , Round , Floor ,
EXAMPLE	<pre>kash> Ceil(3); 3 kash> Ceil(-3); -3 kash> Ceil(3.5); 4 kash> Ceil(-3.5); -3</pre>

NAME CharPoly

PURPOSE Computes the characteristic polynomial of an algebraic element, a matrix or an alff element

SYNTAX `p := CharPoly (a [,PA]);`
`p := CharPoly (a [,0]);`
`p := CharPoly(M);`

polynomial	p
polynomial algebra	PA
suborder	0
algebraic element alff order element	a
matrix	M

DESCRIPTION For an algebraic element a in an Order O and a polynomial algebra PA over another order o or this order, this function computes the characteristic polynomial of a over o i.e. the returned polynomial is contained in PA .
 Given a matrix this function returns its characteristic polynomial
 For an alff element a in an order o in the alff F this function computes the characteristic polynomial of a over o

SEE ALSO [EltCharPolyAlffEltCharPolyMatCharPoly](#),

EXAMPLE

```
kash> o := Order(Z,2,2);
Generating polynomial: x^2 - 2

kash> a := Elt(o,[0,1]);
[0, 1]
kash> p := CharPoly(a);
x^2 - 2
kash> M := Mat(o,[[1,a],[1,-a]]);
[1 [0, 1]]
[1 [0, -1]]
kash> p := CharPoly(M);
x^2 + [-1, 1]*x + [0, -2]
```

NAME	CharToInt
PURPOSE	Returns the ASCII code of a character.
SYNTAX	<pre>i := CharToInt(c);</pre> <pre>character c integer i</pre>
SEE ALSO	IntToChar ,
EXAMPLE	

```
kash> CharToInt('a');
97
```

Characteristic

NAME	Characteristic
------	----------------

PURPOSE	Return the characteristic of a ring.
---------	--------------------------------------

SYNTAX	<code>m := Characteristic(R);</code>
--------	--------------------------------------

	<code>integer m</code>
--	------------------------

	<code>ring R</code>
--	---------------------

EXAMPLE

```
kash> Characteristic(Z);
```

```
0
```

```
kash> Characteristic(C);
```

```
0
```

```
kash> Characteristic(FF(7, 2));
```

```
7
```

NAME CheckArgus

PURPOSE Handy tool to check the input to a kash function.

SYNTAX `r = CheckArgus (hdCall,string);`

<code>r</code>	<code>int</code>	number of type matching format string
<code>hdCall</code>	<code>TypHandle</code>	all arguments
string of format strings	<code>t_string</code>	

DESCRIPTION

Handle-Type	Format	Description
T_BOOL	%B	
T_INT	%d	
T_KantELT	%E	
T_KantFF	%FF	
T_KantFFE	%FFE	
T_KantFIELD	%FLD	
T_KantFFO	%FFO	
T_KantFFOE	%FFOE	
T_KantFFOI	%FFOI	
T_KantFFS	%FFS	
T_KantFFSE	%FFSE	
T_KantFFPL	%FFPL	
T_KantFFDI	%FFDI	
T_KantIDEAL	%I	
T_LIST, T_VECTOR	%L	
T_KantLATELT	%LE	
T_KantLAT	%LT	
T_KantMAT	%M	
T_KantMODULE	%MO	
T_KantORDER	%O	
T_KantPOLY	%P	
T_KantPOLYALG	%PA	
T_PERM16, T_PERM32	%PERM	
T_RAT	%Q	
T_KantQF	%QF	
T_KantQFE	%QFE	
T_KantREAL	%R	
T_KantRING	%r	
T_STRING	%s	
T_KantSIMP	%SI	
T_KantTHUE	%TH	
T_KantVR	%VR	
T_KantVRE	%VRE	
T_KantINT	%Z	

NAME	ChineseRemainder												
PURPOSE	Chinese remainder for integers, ideals, and polynomials.												
SYNTAX	<pre>elt := ChineseRemainder(M,L); elt := ChineseRemiander(LM); elt := ChineseRemainder(m1, m2, a1, a2);</pre> <table><tr><td>integer, polynommmial, or order element</td><td>elt</td></tr><tr><td>list of elements (integers, polynomials, or order elements)</td><td>L</td></tr><tr><td>list of modules (integers, polynomials, or ideals)</td><td>M</td></tr><tr><td>list of pairs of elements and modules</td><td>LM</td></tr><tr><td>modules (integers, polynomials, or ideals)</td><td>m1,m2</td></tr><tr><td>elements (integers, polynomials, or order elements)</td><td>a1,a2</td></tr></table>	integer, polynommmial, or order element	elt	list of elements (integers, polynomials, or order elements)	L	list of modules (integers, polynomials, or ideals)	M	list of pairs of elements and modules	LM	modules (integers, polynomials, or ideals)	m1,m2	elements (integers, polynomials, or order elements)	a1,a2
integer, polynommmial, or order element	elt												
list of elements (integers, polynomials, or order elements)	L												
list of modules (integers, polynomials, or ideals)	M												
list of pairs of elements and modules	LM												
modules (integers, polynomials, or ideals)	m1,m2												
elements (integers, polynomials, or order elements)	a1,a2												
SEE ALSO	IdealChineseRemainder ,												

Close

NAME `Close`

PURPOSE Closes a file.

SYNTAX `closed := Close(f);`

File `f`
boolean `closed`

DESCRIPTION

SEE ALSO `BagRead`, `BagWrite`, `ECHOon`, `ECHOoff`, `FLDin`, `FLDout`, `LOFILES`,
`Open`,

EXAMPLE See `Open` for an example.

NAME Coeff

PURPOSE

SYNTAX a := Coeff(poly, i [, "x"]);

 ring element a
 polynomial poly
 positive integer i

DESCRIPTION

SEE ALSO Coeff,

EXAMPLE

```
kash> a := Coeff(2*x^3+4*x^2-7*x+18,2);
```

```
4
```

ColorString

NAME	ColorString
PURPOSE	Defines a string for use in <code>Print</code> to get colorful output.
SYNTAX	<pre>s := ColorString(l);</pre> <pre>string s list of strings l</pre>
DESCRIPTION	<p>This function generates a <i>VT100/ANSI</i> compatible escape sequence to change the output color. Regardless of the names used here, the actual outcome is influenced by the terminal itself: the terminal may have a visible bell, different colors. The terminal can ignore certain combinations, refuse to blink, ...</p> <p>The <code>l</code> argument contains strings of the following form:</p> <ul style="list-style-type: none">• <code>"f_color"</code> with <i>color</i> one of <code>black</code>, <code>red</code>, <code>green</code>, <code>brown</code>, <code>blue</code>, <code>magenta</code>, <code>cyan</code>, <code>white</code>. This defines the <i>foreground</i> color.• <code>"b_color"</code> for the background color.• <code>"bright"</code>, <code>"blink"</code>, <code>"underscore"</code>, <code>"reverse"</code>, <code>"blank"</code> to set additional attributes.• <code>"bell"</code> to include a terminal-bell. <p>For example to produce red letters on a blue background, one should use <code>ColorString("f_red", "b_blue")</code>. Depending on the <i>termcap</i> entries and (eventually) the <i>.Xresources</i>.</p> <p>To get yellow, one should use <code>"brown"</code> together with <code>"bright"</code>.</p>
EXAMPLE	<pre>Print(ColorString("f_red", "b_black", "bright"), "Hallo", ColorString("normal"), "\n");</pre>

NAME	Colors
PURPOSE	Activate or deactivate color mode and customize colors.
SYNTAX	<pre>Colors(flag); Colors(s1, ..., sn); Colors(L);</pre> <p> boolean flag switch color mode on or off string s1, ..., sn resource strings list L list of lists of resource strings </p>
DESCRIPTION	<p>This function can be used to switch on or off a color mode in KASH compatible with ANSI/ISO 6429 standards. The resource strings consist first of one of the strings "normal", "prompt", "command", "result" or "error" to indicate for which type of output a color should be defined. The other strings can be "black", "red", "green", "yellow", "blue", "magenta", "cyan" or "white" and additionally "bold" or "light". The color mode works only for terminals which can display colored letters like "color_xterm" or "rxvt". See the instructions for your terminal.</p> <p>The default colors can be customized via the <code>_ColorTable</code> array. <code>_ColorTable</code> is a list with exactly 5 entries defining (in this order) the "normal", "prompt", "command", "result", "error" colors. Each entry is a list as described in <code>ColorString</code>.</p>

EXAMPLE

```
kash> Colors(true);
kash> Colors("error", "red", "bold");
kash> Colors([ [ "error", "red" ], [ "result", "green", "bold" ] ]);
kash> Colors(false);
```

Comp

NAME Comp

PURPOSE Creates a complex number.

SYNTAX `z := Comp(a, b);`

 complex z

 real a

 real b

DESCRIPTION Creates the complex number $z = a + ib$.

EXAMPLE Create the complex number $1 + 2i$:

```
kash> z := Comp(1, 2);
```

```
1 + 2*i
```

NAME	ComplexGamma
PURPOSE	Returns .
SYNTAX	$c := \text{ComplexGamma}(z);$ complex z complex c

DESCRIPTION Given .

EXAMPLE

```
kash> c := ComplexGamma(Comp(2.4,3.1));  
-0.1751454757439141626699741825902816647895320549295366 + 0.052332890056926068\  
5054381992239370308373178721307248*i
```

Conj

NAME Conj

PURPOSE Returns the conjugate complex number.

SYNTAX `w := Conj(z);`

 complex w

 complex z

EXAMPLE Compute the conjugate complex number of $1 + 2i$:

```
kash> z := Comp(1, 2);
```

```
1 + 2*i
```

```
kash> w := Conj(z);
```

```
1 - 2*i
```

NAME	Cos
PURPOSE	Returns the cosine of a number.
SYNTAX	<pre>y := Cos(x);</pre> <p>complex y complex x</p>
DESCRIPTION	Given an x (in radians) the function returns the cosine of x. The computation is done in the current precision of the real (complex) field.
SEE ALSO	Sin , Tan ,
EXAMPLE	

```
kash> Cos(.5);
0.877582561890372716116281582603829651991645197109744
kash> i := Comp(0, 1);
1*i
kash> Cos(i);
1.543080634815243778477905620757061682601529112366
```

Date

NAME **Date**

PURPOSE Returns the current date and time in a record

SYNTAX `t := Date();`

`record t`

EXAMPLE An example:

```
kash> t := Date();  
22.2.2004 15:55:17
```

NAME	DbQuery
PURPOSE	Performs a query in the database "kate"
SYNTAX	<pre>DbQuery(query [, num]) string query integer num</pre>
DESCRIPTION	<p>DbQuery(query) performs a SQL-Query in the KANT-database "kate". SQL means Structured-Query-Language, a language for executing a query in the database.</p> <p>You can also access the database through a web interface at</p> <p>http://www.math.tu-berlin.de/cgi-bin/kant/database.cgi</p> <p>The following table lists all possible names of table-names in the database:</p> <ol style="list-style-type: none"> 1. degree 2. class_number 3. galois_grno 4. sig_real 5. sig_im 6. discriminant 7. disc_dec_len 8. disc_ascii 9. regulator 10. reg_ascii disc_known 11. reg_known 12. generator_roots_unity 13. poly_coeff 14. number_roots_unity 15. number_cyclic_factors_clgr 16. list_of_orders_cyclic_factors_clgr

If you use this table-names in your query, it is not case sensitive and the same holds for using underscore, minus and space, this means for example their is no difference

between sig_real, sig-real or sig real.

Most of the table-names are self-explanatory:

degree	degree of a number field
class_number	class number of a number field
galois_grno	every galois-group has a galois-number, which is to be the same as in the Atlas for groups. If you know this number, you can use this in your query
sig_real	real signature of a number field
sig_im	imaginary signature of a number field
discriminant	discriminant of a number field, but if it is too big, so see disc_ascii
disc_dec_len	decimal length of discriminant
disc_ascii	discriminant as ascii, here you have the exact value for the discriminant, but as text(ascii) and you have to convert, if possible, into longinteger
regulator	regulator of a number field
reg_ascii	regulator as ascii, the exact value for the regulator as text
reg_known	is an integer, which means regulator is known, if reg_known = 1, regulator unknown otherwise
generator_roots_unity	generator of the roots of unity
poly_coeff	list of coefficients of the defining
polynomial	
number_roots_unity	number roots of unity
number_cyclic_factors_clgr	number of cyclic factors of the class group
list_of_orders_cyclic_factor_clgr	list of orders of the cyclic factors of classgroup

EXAMPLE For example, if you want fields with degree=3, class number=2 discriminant > 0 and discriminant < 10000 ordered by discriminant:

```
kash> DbQuery("deg=3 and class number=2 and disc> 0 and disc<10000 order by disc");
```

and for a descending order

```
kash> DbQuery("deg=3 and class number=2 and disc>0 and disc<10000 order by -disc");
```

So you will get a list up to 1000 orders in the database with this properties. If you want the next 1000 orders, you have to change your estimate for the discriminant: if the last order of the list L has discriminant=9876, then your query should be

```
kash> DbQuery("deg=3 and class number=2 and disc>=9876 order by disc");
```

If you want to know how many fields are in the database with this porperties,
you have to type

```
kash> DbQuery("count deg=3 and class number=2 and disc>0 and disc<10000 order by disc");
```

You will get a statistic foro the database, if you type

```
kash> DbQuery("Statistic");
```

DedekindEta

NAME `DedekindEta`

PURPOSE Calculates the value of Dedekind's η -function.

SYNTAX `u := DedekindEta(z);`

`complex u`

`complex z` complex number with $\text{Im}(z) > 0$

DESCRIPTION Let z be a complex number with $\text{Im}(z) > 0$. The value

$$\eta(z) = q^{1/24} \prod_{n=1}^{\infty} (1 - q^n)$$

of the Dedekind's η -function (with $q = e^{2\pi i \cdot z}$) is then a well-defined complex number. It is determined by writing the infinite product as a (quickly converging) infinite sum and calculating this sum up to a certain precision, that depends on the precision of the defining complex field. For further information about the algorithm see [\[Sch90b\]](#).

EXAMPLE Compute $\eta(1 + \sqrt{-5})$:

```
kash> DedekindEta(1+Sqrt(-5));  
0.5379066478127104520437266840371093596748318066880552 + 0.1441316518847481027\  
896657792883659909845154545709301*i
```

NAME	Den
PURPOSE	Returns the denominator of an object.
SYNTAX	<pre> d := Den(a); d := Den(a, "rep"); d := Den(q); I := Den(I); integer d quotient field element or polynomial d rational q algebraic element a algebraic function field order element a quotient field element or polynomial q ideal I </pre>
DESCRIPTION	<p>For a rational number or a rational function the function returns the denominator of the rational argument.</p> <p>The denominator of the algebraic number a is the smallest natural number d such that $d \cdot a$ is an algebraic integer. If you use <code>Den(a, "rep")</code> the denominator in the representation of a is returned.</p> <p>For an algebraic function field order element see <code>AlffEltDen</code>.</p> <p>The denominator of a fractional ideal \mathfrak{a} is the smallest positive integer d that $d \cdot \mathfrak{a}$ is an integral ideal.</p>

EXAMPLE An absolute example for an algebraic element:

```

kash> o := Order (x^3 - 23*x^2 + 146*x - 244);;
kash> a := Elt (o, [1/2,3/5,45/5657567]);
[28287835, 33945402, 450] / 56575670
kash> Den ( a );
56575670
kash> Den ( a, "rep" );
56575670

```

NAME Disc

PURPOSE Computes the discriminant of a polynomial, an order, an algebraic element, an alff order or a lattice.

SYNTAX `d := Disc(f);`
 `d := Disc(o);`
 `d := Disc(l);`

ring element	d
polynomial	f
order alff order	o
lattice	l

DESCRIPTION

EXAMPLE We will compute some indices:

```
kash> o := OrderMaximal(Order(x^6 - 9*x^4 - 4*x^3 + 27*x^2 - 36*x - 23));;
kash> 0 := OrderShort(o);;
kash> Disc(OrderPoly(Zx, o)) / Disc(o);
121352256
kash> Disc(OrderPoly(Zx, 0)) / Disc(0);
25
```


NAME	DrinfMPhi
PURPOSE	Definition of a Drinfeld Modul
SYNTAX	<p><code>D:= DrinfMPhi(f,L);</code></p> <p> <code>finite field polynomial</code> <code>D</code> <code>finite field polynomial</code> <code>f</code> <code>list</code> <code>L</code> </p>
DESCRIPTION	<p>Consider the ring $A := \mathbb{F}_q[T]$ as \mathbb{F}_q vector space and the ring $A\{\tau\}$, defined by the Frobenius map. This function is an implementation of the homomorphism $\phi : A \rightarrow A\{\tau\}$ defined by</p> $\phi(T) = T + a_1\tau + a_2\tau^2 + \dots + a_r\tau^r$ <p>with $L := [a_1, \dots, a_r]$, $a_i \in \mathbb{F}_q$. r is the rank of the modul. The output is given by</p> $z \mapsto Tz + a_1z^q + a_2z^{q^2} + \dots + a_rz^{q^r}$ <p>This is a preliminary version. To compute $\phi(T^2)$ the user must calculate using the homomorphism $\phi(T^2) = \phi(T)\phi(T)$</p>
EXAMPLE	<pre> kash> k:=FF(3); Finite field of size 3 kash> kT:=PolyAlg(k,"T"); Univariate Polynomial Ring in T over GF(3) kash> kTz:=PolyAlg(kT,"z"); Univariate Polynomial Ring in z over Univariate Polynomial Ring in T over GF(3\) kash> L:=[1,1]; [1, 1] kash> DrinfMPhi(T,L); Drinfeld module of rank 2 z^9 + z^3 + T*z </pre>

NAME	DrinfMProduct
PURPOSE	Computes a non-commutative product of polynomials
SYNTAX	<pre>az := DrinMProduct(a,z);</pre> <p> finite field polynomial az finite field polynomial a finite field polynomial z </p>
DESCRIPTION	Let $a, z \in A = \mathbb{F}_q[T]$ be polynomials and τ the Frobenius map. This function computes the product of a and z given by $\tau a = a^q \tau$ in the non-commutative subring of all \mathbb{F}_q -linear maps of A .
SEE ALSO	DrinTau ,
EXAMPLE	<pre>kash> k:=FF(3); Finite field of size 3 kash> kT:=PolyAlg(k,"T"); Univariate Polynomial Ring in T over GF(3) kash> kTz:=PolyAlg(kT,"z"); Univariate Polynomial Ring in z over Univariate Polynomial Ring in T over GF(3\) kash> DrinfMProduct(T,z); T*z^3</pre>

NAME	DrinfMTau
PURPOSE	Computes the Frobenius map for polynomials.
SYNTAX	<pre>f := DrinfMTau(g);</pre> <p> finite field polynomial f finite field polynomial g </p>
DESCRIPTION	For given $g \in A = \mathbb{F}_q[T]$ this function computes the Frobenius map $g \mapsto g^q$.
EXAMPLE	<pre>kash> k:=FF(3); Finite field of size 3 kash> kT:=PolyAlg(k,"T"); Univariate Polynomial Ring in T over GF(3) kash> DrinfMTau(T); T^3</pre>

ECHOff

NAME ECHOff

PURPOSE Switches back to normal behaviour after ECHOn.

SYNTAX ECHOff();

DESCRIPTION

SEE ALSO ECHOn,

EXAMPLE See ECHOn for an example.

NAME	ECHOon
PURPOSE	Switches the output of stdout to a file.
SYNTAX	<p>ECHOon (name file [, arg]);</p> <p>string name filename to write to</p> <p>file file open for writing</p> <p>expression arg should produce an output to stdout</p>
DESCRIPTION	<p>Changes the output stream to output in a file. Note that it is dangerous if an error occurs, because you will not be able to see most of the KASHmessages. It is advisable to use it only in the form:</p> <p style="text-align: center;">ECHOon(file, arg); ECHOoff();</p> <p>because now KASHwill change back to normal behaviour even if an error occurs. ECHOoff switches the indirection off again. Note, that a call like</p> <p style="text-align: center;">ECHOon(file);</p> <p>forces you to type all commands, including</p> <p style="text-align: center;">ECHOoff();</p> <p>without any echo on the screen. All output is redirected to file.</p>
SEE ALSO	ECHOoff ,

EXAMPLE Debug information of an order maximal call.

```

kash> PRINTLEVEL ("ROUND2", 2);
2
kash> f := Open ("/tmp/dump","w");
Filename: /tmp/dump / Mode: w / Open (fid): 5
kash> ECHOon (f, OrderMaximal (Order (x^5 + 5*x^3 + 16*x^2 - 6*x + 38)));
  F[1]
  |
  F[2]
  /
  /
Q
F [ 1]      Given by transformation matrix
F [ 2]      x^5 + 5*x^3 + 16*x^2 - 6*x + 38

```

ECHOon

Discriminant: 3828384

```
kash> Exec ("cat /tmp/dump");  
kash> Close (f);  
true
```

NAME	EccDecrypt
PURPOSE	Decrypt a message that was encrypted using the ElGamal public key cryptosystem and a subgroup of the group of an elliptic curve.
SYNTAX	$P := \text{EccDecrypt}(K, E, a, M);$ <div style="margin-left: 100px;"> $\text{finite field} \quad K$ $\text{list} \quad E$ $\text{integer} \quad a$ $\text{list} \quad M$ $\text{list} \quad P$ </div>
DESCRIPTION	<p>Let E be an elliptic curve over a finite field K, let B be a basepoint of a subgroup of the group of E. Bob chooses an integer a, as his secret key. The message $M = [P_1, P_2]$ consisting of two points on E is decrypted to a point P by setting $P = P_1 - a \cdot P_2$. The elliptic curve E is either given by a list of two or five elements of K or integers. If the equation of E is $y^2 = x^3 + a_4x + a_6$ the curve is represented by $[a_4, a_6]$, if the equation of E is $y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6$ then the representation is $[a_1, a_2, a_3, a_4, a_6]$. Points on the curve are given by a pair of elements of K or integers; the point at infinity is represented by the empty list $[]$.</p>
SEE ALSO	EccPointsAdd , EccIntPointMult , EccPointIsOnCurve , EccEncrypt , FF ,

EXAMPLE

```

kash> field := FF(751);
Finite field of size 751
kash> ec := [0,0,1,-1,0];
[ 0, 0, 1, -1, 0 ]
kash> base_point := [0,0];
[ 0, 0 ]
kash> secret_key := 58;
58
kash> public_key := EccIntPointMult(field,ec,secret_key,base_point);
[ 201, 380 ]
kash> plaintext := [[562,576],[581,395],[484,214],[501,220],[1,0]];
[ [ 562, 576 ], [ 581, 395 ], [ 484, 214 ], [ 501, 220 ], [ 1, 0 ] ]
kash> k := [254,180,99,472,275];
[ 254, 180, 99, 472, 275 ]
kash> ciphertext := List([1..5], \

```

```
> x-> EccEncrypt(field,ec,base_point,public_key,k[x],plaintext[x]));  
[ [ [ 268, 146 ], [ 378, 547 ] ], [ [ 680, 469 ], [ 409, 94 ] ],  
  [ [ 710, 395 ], [ 195, 432 ] ], [ [ 747, 222 ], [ 101, 371 ] ],  
  [ [ 13, 246 ], [ 386, 303 ] ] ]  
kash> List(ciphertext, x-> EccDecrypt(field,ec,secret_key,x));  
[ [ 562, 576 ], [ 581, 395 ], [ 484, 214 ], [ 501, 220 ], [ 1, 0 ] ]
```


NAME	EccEncrypt
PURPOSE	Encrypt a message (point) using the ElGamal public key cryptosystem and a subgroup of the group of an elliptic curve.
SYNTAX	$M := \text{EccEncrypt}(K, E, B, aB, k, P);$ <div style="margin-left: 40px;"> <code>finite field</code> <code>K</code> <code>list</code> <code>E</code> <code>list</code> <code>B</code> <code>list</code> <code>aB</code> <code>integer</code> <code>k</code> <code>list</code> <code>P</code> <code>list</code> <code>M</code> </div>
DESCRIPTION	<p>Let E be an elliptic curve over a finite field K, let B be a basepoint of a subgroup of the group of E. Bob chooses an integer a, as his secret key. He publishes the public key $aB := a \cdot B$. Before sending a message to Bob Alice picks a random integer k, then she sends him the point pair $[k \cdot B, P + k \cdot aB]$. The elliptic curve E is either given by a list of two or five elements of K or integers. If the equation of E is $y^2 = x^3 + a_4x + a_6$ the curve is represented by $[a_4, a_6]$, if the equation of E is $y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6$ then the representation is $[a_1, a_2, a_3, a_4, a_6]$. Points on the curve are given by a pair of elements of K or integers; the point at infinity is represented by the empty list $[]$.</p>
SEE ALSO	EccPointsAdd , EccIntPointMult , EccPointIsOnCurve , EccDecrypt , FF ,
EXAMPLE	

```

kash> K := FF(11);
Finite field of size 11
kash> E := [0,0,0,1,6];
[ 0, 0, 0, 1, 6 ]
kash> B := [2,7];
[ 2, 7 ]
kash> a := 8;
8
kash> aB := EccIntPointMult(K,E,a,B);
[ 3, 5 ]
kash> k := 7;
7

```

```
kash> M := EccEncrypt(K,E,B,aB,k,[3,6]);  
[ [ 7, 2 ], [ 10, 9 ] ]  
kash> EccDecrypt(K,E,a,M);  
[ 3, 6 ]  
kash> k := 5;  
5  
kash> M := EccEncrypt(K,E,B,aB,k,[3,6]);  
[ [ 3, 6 ], [ 7, 9 ] ]  
kash> EccDecrypt(K,E,a,M);  
[ 3, 6 ]
```

NAME	EccInit												
PURPOSE	Initializes a representation of an elliptic curve for the Ecc-package.												
SYNTAX	<pre>ec := EccInit(p,a,b); ec := EccInit(p,li); ec := EccInit(fli);</pre> <table> <tr> <td>elliptic curve</td><td>ec</td></tr> <tr> <td>integer</td><td>a</td></tr> <tr> <td>integer</td><td>b</td></tr> <tr> <td>integer</td><td>p</td></tr> <tr> <td>list of integers</td><td>li</td></tr> <tr> <td>list of finite field elements</td><td>fli</td></tr> </table>	elliptic curve	ec	integer	a	integer	b	integer	p	list of integers	li	list of finite field elements	fli
elliptic curve	ec												
integer	a												
integer	b												
integer	p												
list of integers	li												
list of finite field elements	fli												

DESCRIPTION

EXAMPLE

```
kash> p:=65112*2^144-1;
1452046121366725933991673688168680114377396846591
kash> IsPrime(p);
true
kash> EccInit(p,-3,49);
[ 1452046121366725933991673688168680114377396846588, 49 ]
```

NAME	EccIntPointMult
PURPOSE	Scalar multiplication of a point on an elliptic curve.
SYNTAX	<pre>nP := EccIntPointMult(K,E,n,P);</pre> <div> <div>finite field</div> <div>list</div> <div>integer</div> <div>list</div> <div>list</div> </div> <div> <div>K</div> <div>E</div> <div>n</div> <div>P</div> <div>nP</div> </div>
DESCRIPTION	<p>Using the group law on an elliptic curve E over a finite field K a point $P = [x, y]$ on E is multiplied by the scalar n. E is either given by a list of two or five elements of K or integers. If the equation of E is $y^2 = x^3 + a_4x + a_6$ the curve is represented by $[a_4, a_6]$, if the equation of E is $y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6$ then the representation is $[a_1, a_2, a_3, a_4, a_6]$. Points on the curve are given by a pair of elements of K or integers; the point at infinity is represented by the empty list $[]$.</p>
SEE ALSO	EccPointsAdd , EccPointIsOnCurve , EccEncryptEccDecrypt , FF ,

EXAMPLE

```
kash> K := FF(11);
Finite field of size 11
kash> E := [0,0,0,1,6];
[ 0, 0, 0, 1, 6 ]
kash> EccIntPointMult(K,E,-1,[2,7]);
[ 2, 4 ]
kash> EccIntPointMult(K,E,5,[2,7]);
[ 3, 6 ]
kash> EccIntPointMult(K,E,12,[2,7]);
[ 2, 4 ]
kash> EccIntPointMult(K,E,13,[2,7]);
[ ]
kash> EccIntPointMult(K,E,14,[2,7]);
[ 2, 7 ]
```

NAME	EccKangaroo
PURPOSE	Computes the number of places of an elliptic curve, given by the Weierstrass form over a prime finite field \mathcal{F}_p .
SYNTAX	<pre>NP:=EccNumberOfPoints(p F,a,b); NP:=EccNumberOfPoints(ec);</pre> <pre>int or finite field element a int or finite field element b int p finite field F list ec</pre>
DESCRIPTION	

NAME	EccPointIsOnCurve								
PURPOSE	Checks whether a point is on a given elliptic curve.								
SYNTAX	<pre>b := EccPointIsOnCurve(K,ec,point);</pre> <table> <tr> <td>finite field</td><td>K</td></tr> <tr> <td>list</td><td>ec</td></tr> <tr> <td>list</td><td>point</td></tr> <tr> <td>boolean</td><td>b</td></tr> </table>	finite field	K	list	ec	list	point	boolean	b
finite field	K								
list	ec								
list	point								
boolean	b								
DESCRIPTION	<p>Checks whether a point $P = [x, y]$ is on the elliptic curve E over a finite field K. E is either given by a list of two or five elements of K or integers. If the equation of E is $y^2 = x^3 + a_4x + a_6$ the curve is represented by $[a_4, a_6]$, if the equation of E is $y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6$ then the representation is $[a_1, a_2, a_3, a_4, a_6]$. Points on the curve are given by a pair of elements of K or integers; the point at infinity is represented by the empty list $[]$.</p>								
SEE ALSO	EccPointsAdd , EccIntPointMult , EccEncrypt , EccDecrypt , FF ,								
EXAMPLE									

```
kash> K := FF(11);
Finite field of size 11
kash> ec := [0,0,0,1,6];
[ 0, 0, 0, 1, 6 ]
kash> EccPointIsOnCurve(K,ec,[2,7]);
true
kash> EccPointIsOnCurve(K,ec,[1,1]);
false
kash> EccPointIsOnCurve(K,ec,[]);
true
```

NAME	EccPointsAdd
PURPOSE	Returns the sum of two points on an elliptic curve over a finite field.
SYNTAX	<pre>sum := EccPointsAdd(K,ec,P1,P2);</pre> <div> <div>finite field</div> <div>list</div> <div>list</div> <div>list</div> <div>list</div> <div>list</div> </div> <div> <div>K</div> <div>E</div> <div>P1</div> <div>P2</div> <div>sum</div> </div>
DESCRIPTION	<p>Using the group law on an elliptic curve E over a finite field K two points $P_1 = [x_1, y_1]$ and $P_2 = [x_2, y_2]$ are added. E is either given by a list of two or five elements of K or integers. If the equation of E is $y^2 = x^3 + a_4x + a_6$ the curve is represented by $[a_4, a_6]$, if the equation of E is $y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6$ then the representation is $[a_1, a_2, a_3, a_4, a_6]$. Points on the curve are given by a pair of elements of K or integers; the point at infinity is represented by the empty list $[]$.</p>
SEE ALSO	EccIntPointMult , EccPointIsOnCurve , EccEncrypt , EccDecrypt , FF ,
EXAMPLE	

```
kash> K := FF(11);
Finite field of size 11
kash> E := [0,0,0,1,6];
[ 0, 0, 0, 1, 6 ]
kash> EccPointsAdd(K,E,[2,7],[5,9]);
[ 2, 4 ]
kash> EccPointsAdd(K,E,[2,7],[]);
[ 2, 7 ]
```

NAME EccRandomPoint

PURPOSE Returns a random point on a given elliptic curve.

SYNTAX `P := EccRandomPoint(F,E);`

point on the elliptic curve E	P
finite field	F
elliptic curve	E

DESCRIPTION

EXAMPLE

```
kash> p:=65112*2^144-1;
1452046121366725933991673688168680114377396846591
kash> IsPrime(p);
true
kash> E := EccInit(p,-3,49);
[ 1452046121366725933991673688168680114377396846588, 49 ]
kash> EccRandomPoint(FF(p),E);
[ 279321685036220943951873246284411164161011401467,
  278535878987595364269277267695870885018052411927 ]
```


NAME Ei

PURPOSE This function computes an approximation of the integral from $-\infty$ to x of $\exp(t)/t$.

SYNTAX `a := Ei(b);`

`real a`

`real b`

DESCRIPTION

$$\int_{-\infty}^x \frac{\exp(t)}{t} dt$$

EXAMPLE

```
kash> Ei(1.0);
```

```
1.895117816355936755466520934331634269017060581732
```

EisensteinSeries

NAME EisensteinSeries

PURPOSE Returns the value of the non-holomorphic Eisensteinseries.

SYNTAX `c := EisensteinSeries(tau,s,a1,a2,q,N);`

complex	tau
real	s
integer	a1
integer	a2
integer	q
positive integer	N

DESCRIPTION Given $E(\tau, s, a_1, a_2, q) := \sum_{\substack{n \equiv a_1 \pmod q, m \equiv a_2 \pmod q}} |n\tau + m|^s$ for $s \in \mathbb{C}$ and $\operatorname{Re}(s) > 1$. This function has a meromorphic continuation to \mathbb{C} . We use the Fourier-series- expansion to calculate this function. [\[Maa49\]](#).

EXAMPLE

```
kash> c := EisensteinSeries(Comp(2.4,3.1),0.4,2,3,6,5);  
-0.0464001568398209806672836216520773803072859195067551 + 1.016212292896410126\  
673369727459856258569024111299e-52*i
```

NAME	Elt
PURPOSE	Creates an algebraic number.
SYNTAX	<pre> a := Elt(0,L); a := Elt(0,L/d); a := Elt(0,h/d); a := Elt(0,h); algebraic element a order 0 list L of coefficients integer h integer d </pre>
DESCRIPTION	<p>Let \mathfrak{o} be an order with coefficient order \mathfrak{c}. Denote by $Q(\mathfrak{o})$ and $Q(\mathfrak{c})$ the quotient field of \mathfrak{o}, respectively \mathfrak{c}. Let $\omega_1, \dots, \omega_n$ be the $Q(\mathfrak{c})$ basis of $Q(\mathfrak{o})$ which belongs to the order \mathfrak{o}. Each algebraic number a in \mathfrak{o} is of a form</p> $a = a_1\omega_1 + \dots + a_n\omega_n$ <p>with coefficients $a_1, \dots, a_n \in Q(\mathfrak{c})$.</p> <p>To create a in KASH the coefficients of a must be passed to the <code>Elt</code> function as a list <code>L</code> of length n. The entries of <code>L</code> correspond to the coefficients a_i of a. Each entry <code>l</code> of <code>L</code> is either an algebraic number in \mathfrak{c} or it has to be a valid argument for the function call <code>Elt(c,l)</code>. In the special case that \mathfrak{c} equals \mathbb{Z} the <code>Elt</code> function requires that the entries of <code>L</code> are rational integers.</p> <p><code>Elt(o,L)</code> creates the element $a = a_1\omega_1 + \dots + a_n\omega_n$.</p> <p><code>Elt(0,L/d)</code> creates the element $a = \frac{a_1}{d}\omega_1 + \dots + \frac{a_n}{d}\omega_n$.</p> <p><code>Elt(0,h)</code> creates the algebraic number which corresponds to the rational integer h.</p> <p><code>Elt(0,h/d)</code> creates the algebraic number which corresponds to the rational $\frac{h}{d}$.</p>

EXAMPLE Create some algebraic elements:

```

kash> o1 := Order (Poly(Zx,[1,0,73,-280,-2399]));;
kash> u := Elt (o1,[1,2,3,4]/2);
[1, 2, 3, 4] / 2

```

```

kash> v := Elt (o1,2);
2
kash> u*v;
[1, 2, 3, 4]

kash> o2 := Order (o1,2,2);;
kash> o3 := Order (o2,2,3);
      F[1]
      /
      /
      E2[1]
      /
      /
      E1[1]
      /
      /
Q
F [ 1]      x^2 - 3
E 2[ 1]      x^2 - 2
E 1[ 1]      x^4 + 73*x^2 - 280*x - 2399

kash> a := Elt (o3,[1,2]);
[1, 2]
kash> b := Elt (o3,[[[1,1,1,1],1/2],2]);
[[[2, 2, 2, 2], 1], 4] / 2
kash> c := Elt (o3,[[[1,1,1,1],1/2],[1/3,[4,5,6,8]/4]]);
[[[12, 12, 12, 12], 6], [4, [12, 15, 18, 24]]] / 12
kash> d := Elt (o3,1/2);
1 / 2

```

NAME	EltAbs
PURPOSE	missing shortdoc
SYNTAX	<pre>v := EltAbs (a);</pre> <p> real matrix v algebraic element a </p>
DESCRIPTION	<p>Let a be an algebraic number whose underlying order is over \mathbb{Z} having r_1 real and r_2 complex places. The <code>EltAbs</code> function returns the real matrix $(a^{(1)} , \dots, a^{(r_1+r_2)})$.</p>
SEE ALSO	OrderSig , EltCon ,
EXAMPLE	<pre>kash> o := Order (Poly(Zx,[1,0,73,-280,-2399])); Generating polynomial: x^4 + 73*x^2 - 280*x - 2399 kash> a := Elt(o,[0,1,0,0]); [0, 1, 0, 0] kash> EltAbs (a); [6.104004643496482052536282272258995385473953052355 3.867936665996692356127108\ 603527719150033334692743 10.080173776775582528154588754110436634147502950471]</pre>

NAME	EltAbsLogHeight
PURPOSE	Computes the absolute logarithmic height of an algebraic number.
SYNTAX	<pre>h := EltAbsLogHeight(a);</pre> <p> real h algebraic element a </p>
DESCRIPTION	<p>Let α be an algebraic number. Denote by</p> $m_\alpha(t) = a_0 t^n + \cdots + a_n = a_0 \prod_{j=1}^n (t - \alpha^{(j)})$ <p>its minimal polynomial over \mathbb{Z}. The absolute logarithmic height of α is defined by</p> $h(\alpha) = \frac{1}{n} \log \left(a_0 \prod_{j=1}^n \max(1, \alpha^{(j)}) \right).$
EXAMPLE	<pre>kash> o := Order(Z,4,7); Generating polynomial: x^4 - 7 kash> a := Elt(o,[1,1,1,1]); [1, 1, 1, 1] kash> EltAbsLogHeight(a); 1.343819601921041250609358018785526704542243019136 kash> EltAbsLogHeight(a/100); 4.085309800568132385973058818275095989145577876487</pre>

NAME	EltApproximation
PURPOSE	Returns an element with certain valuations at prime ideals.
SYNTAX	$E := \text{EltApproximation}(P, L);$ <div style="margin-left: 100px;"> list P of distinct prime ideals over the same order list L of small integers algebraic element E </div>
DESCRIPTION	This is a version of the approximation theorem for algebraic numbers over number rings.
SEE ALSO	IdealChineseRemainder , RayCantoneseRemainder ,

EXAMPLE Weak approximation theorem in $\mathbb{Q}(\rho)$ with $\rho^3 - 12\rho + 10 = 0$.

```
kash> k := OrderMaximal (Order (x^3 - 12*x + 10));
Generating polynomial: x^3 - 12*x + 10
Discriminant: 4212

kash> P := Filtered (Flat (Factor (56*k)), IsIdeal);
[ <2, [0, 1, 0]>, <7, [1, 1, 0]>, <7, [3, 6, 1]> ]
kash> L := [3,6,7];
[ 3, 6, 7 ]
kash> mu := EltApproximation (P,L);
[10117814, -4470662, -470596]
kash> for I in P do Print (I," --> ", Valuation (I, mu),"\n"); od;
<2, [0, 1, 0]> --> 3
<7, [1, 1, 0]> --> 6
<7, [3, 6, 1]> --> 7
```

NAME	EltAutomorphism
PURPOSE	Applies an automorphism to an algebraic number.
SYNTAX	<pre> L := EltAutomorphism (a); c := EltAutomorphism (a,i); list L algebraic element c algebraic element a integer i </pre>
DESCRIPTION	<p>Let \mathcal{F} be a normal number field with \mathbb{Q}-automorphisms $\sigma_1, \dots, \sigma_n$. Let a be an algebraic number in \mathcal{F}.</p> <p>EltAutomorphism(a) returns the list $L = \{\sigma_1(a), \dots, \sigma_n(a)\}$. Each entry of L is an algebraic number whose underlying order is the same order as a is defined over.</p> <p>EltAutomorphism(a,i) returns the algebraic number $\sigma_i(a)$. Note that σ_1 is always the identity mapping of \mathcal{F}.</p> <p>Before calling EltAutomorphism, the automorphisms $\sigma_1, \dots, \sigma_n$ of \mathcal{F} must be computed by the OrderAutomorphisms routine.</p>
SEE ALSO	OrderAutomorphisms ,

EXAMPLE Consider the normal field $\mathbb{Q}(\sqrt[6]{-108})$:

```

kash> o := Order(Z,6,-108);
Generating polynomial: x^6 + 108

kash> OrderAutomorphisms(o);;
kash> a := Elt(o,[1,2,3,4,5,6]);
[1, 2, 3, 4, 5, 6]
kash> A := EltAutomorphism(a);
[ [1, 2, 3, 4, 5, 6], [12, -552, 630, 48, -28, -39] / 12,
  [12, 528, -666, 48, -32, -33] / 12, [1, -2, 3, -4, 5, -6],
  [12, 552, 630, -48, -28, 39] / 12, [12, -528, -666, -48, -32, 33] / 12 ]
kash> A[1]+A[2]+A[3]+A[4]+A[5]+A[6];
6
kash> EltTrace(a);
6
kash> EltAutomorphism(a,1);
[1, 2, 3, 4, 5, 6]

```


NAME	EltCharPoly
PURPOSE	Characteristic polynomial of an algebraic element over a subfield.
SYNTAX	<pre>p := EltCharPoly (a [, PA]); p := EltCharPoly (a [, 0]);</pre> <p> polynomial p polynomial algebra PA suborder 0 algebraic element a </p>
DESCRIPTION	<p>Given an algebraic element a in an Order O and a polynomial algebra PA over another order o or this order, this function computes the characteristic polynomial of a over o i.e. the returned polynomial is contained in PA.</p>

EXAMPLE Characteristic polynomials of elements in $\mathbb{Q}(\sqrt{10}, \sqrt{5}, \sqrt{3})$.

```
kash> E1 := Order (Z,2,10);
Generating polynomial: x^2 - 10
```

```
kash> E2 := Order (E1, 2, 5);
```

```

      F[1]
      /
      /
    E1[1]
    /
    /
  Q
F  [ 1]      x^2 - 5
E 1[ 1]      x^2 - 10
```

```
kash> F := Order (E2, 2, 3);
```

```

      F[1]
      /
      /
    E2[1]
    /
    /
  E1[1]
  /
  /
Q
```

```

F [ 1]      x^2 - 3
E 2[ 1]      x^2 - 5
E 1[ 1]      x^2 - 10

```

```
kash> E1x := PolyAlg (E1);
```

```
Univariate Polynomial Ring in x over Generating polynomial: x^2 - 10
```

```
kash> Fx := PolyAlg (F);
```

```
Univariate Polynomial Ring in x over          F[1]
```

```

      /
     /
    E2[1]
   /
  /
 E1[1]
/
/
Q

```

```

Q
F [ 1]      x^2 - 3
E 2[ 1]      x^2 - 5
E 1[ 1]      x^2 - 10

```

```
kash> a := Elt (F, [[[1,2],[3,4]],[[3/5, 5],[7,9]]]);
```

```
[[[5, 10], [15, 20]], [[3, 25], [35, 45]]] / 5
```

```
kash> g := EltCharPoly (a, E1x);
```

```

x^4 + [-4, -8]*x^3 + [-717904, -100800] / 25*x^2 + [-3443792, -2723584] / 25*x\
+ [99481555504, 24875390400] / 625

```

```
kash> f := PolyMove (g, Fx);
```

```

x^4 + [[[-4, -8], 0], 0]*x^3 + [[[-717904, -100800], 0], 0] / 25*x^2 + [[[-344\
3792, -2723584], 0], 0] / 25*x + [[[99481555504, 24875390400], 0], 0] / 625

```

```
kash> Eval (f, a);
```

```
0
```


NAME	EltDen
PURPOSE	Returns the denominator of an algebraic element.
SYNTAX	<pre>d := EltDen(a); d := EltDen(a, "rep");</pre> <p>integer d</p> <p>algebraic element a</p>
DESCRIPTION	The denominator of the algebraic number a is the smallest natural number d such that $d \cdot a$ is an algebraic integer. If you use <code>EltDen(a, "rep")</code> the denominator in the representation of a is returned.

EXAMPLE An absolute example.

```
kash> o := Order (x^3 - 23*x^2 + 146*x - 244);;
kash> a := Elt (o, [1/2,3/5,45/5657567]);
[28287835, 33945402, 450] / 56575670
kash> EltDen (a);
56575670
kash> EltDen (a, "rep");
56575670
```

NAME	EltDivisors
PURPOSE	Computes all divisors of an algebraic integer.
SYNTAX	<pre>L := EltDivisors (a);</pre> <div> <pre>list L</pre> <pre>algebraic integer a</pre> </div>
DESCRIPTION	The <code>EltDivisors</code> function computes a list containing all divisors of an algebraic integer a up to multiplication by a unit. The underlying order of a must be known to be maximal.
EXAMPLE	

```
kash> o := OrderMaximal(Z,2,-5);
Generating polynomial: x^2 + 5
Discriminant: -20
```

```
kash> a := Elt(o,6);
6
kash> EltDivisors(a);
[ 1, 2, [1, 1], [1, -1], 3, 6 ]
```

NAME	EltExcepUnitOrbit
PURPOSE	Computes the orbit of an exceptional unit.
SYNTAX	$L := \text{EltExcepUnitOrbit}(\text{epsilon});$ <div style="margin-left: 100px;"> $\text{list} \qquad \qquad \qquad L$ $\text{algebraic element} \quad \text{epsilon}$ </div>
DESCRIPTION	<p>Computes the orbit of an exceptional unit ε. The orbit is given by</p> $\left\{ \varepsilon, \quad \frac{1}{\varepsilon}, \quad 1 - \varepsilon, \quad \frac{1}{1 - \varepsilon}, \quad \frac{\varepsilon - 1}{\varepsilon}, \quad \frac{\varepsilon}{\varepsilon - 1} \right\}.$ <p>If ε is not an exceptional unit, an error message is returned.</p>
SEE ALSO	OrderUnitsExcep ,

EXAMPLE Compute the orbit of $1 + \zeta_7 \in \mathbb{Z}[\zeta_7]$:

```
kash> o := Order(Poly(Zx,[1,1,1,1,1,1,1]));
Generating polynomial: x^6 + x^5 + x^4 + x^3 + x^2 + x + 1

kash> EltExcepUnitOrbit(Elt(o,[1,1,0,0,0,0]));
[ [0, -1, 0, 0, 0, 0], [1, 1, 0, 0, 0, 0], [0, -1, -1, -1, -1, -1],
  [0, -1, 0, -1, 0, -1], [1, 1, 0, 1, 0, 1], [1, 1, 1, 1, 1, 1] ]
```

NAME	EltFactor
PURPOSE	Computes the prime ideal factorization of an algebraic integer.
SYNTAX	<pre>L := EltFactor(a);</pre> <div> <div>list</div> <div>algebraic element</div> <div>L</div> <div>a</div> </div>
DESCRIPTION	The EltFactor function returns the prime ideal decomposition of the principal ideal generated by the algebraic element a . The element must be defined over a maximal order or its norm must be coprime to the discriminant of the equation order.
SEE ALSO	Factor ,
EXAMPLE	

```
kash> o := Order(Z,2,-110);
Generating polynomial: x^2 + 110
```

```
kash> O := OrderMaximal(o);
Generating polynomial: x^2 + 110
Discriminant: -440
```

```
kash> alpha := Elt(O, [0,1]);
[0, 1]
kash> EltFactor(alpha);
[ [ <11, [0, 1]>, 1 ], [ <5, [0, 1]>, 1 ], [ <2, [0, 1]>, 1 ] ]
kash> IdealFactor(alpha*O);
[ [ <2, [0, 1]>, 1 ], [ <5, [0, 1]>, 1 ], [ <11, [0, 1]>, 1 ] ]
```


NAME	EltIdealReduce
PURPOSE	Returns a canonical representative modulo an ideal.
SYNTAX	<pre> b := EltIdealReduce(a,I); same as b := EltIdealReduce(a,I,HNF); b := EltIdealReduce(a,I,LLL); b := EltIdealReduce(a,I,HNF_POS); algebraic element b algebraic element a Ideal I interpreted as strings HNF, LLL, HNF_POS </pre>
DESCRIPTION	<p>This function returns a representation of $a \bmod I$ using either the usual ideal base, a LLL-reduced basis or yet another basis.</p> <p>LLL-reducing results in smaller elements in terms of the norm of the element but takes longer — firstly to compute another basis of the ideal (which is then stored for this ideal) and secondly to compute the representative is more difficult.</p> <p>The HNF_POS method reduces in the same way as HNF, but the coefficients of the returned element are positive.</p> <p>A fractional ideal is treated as follows: the element α is reduced to β with the ideal $\frac{\mathfrak{a}}{n}$ if the element $n\alpha$ is reduced to $n\beta$ with \mathfrak{a}.</p> <p>A fractional elements is treated as follows: the element $\frac{\alpha}{n}$ is reduced to $\frac{\beta}{n}$ with the ideal \mathfrak{a} if the element α is reduced to β with $n\mathfrak{a}$.</p> <p>There is a different generalization to fractional elements where the denominator (as a principal ideal) is inverted in the factor ring.</p> <p>This function can only handle \mathbb{Z}-orders.</p>

EXAMPLE

```

kash> o := OrderMaximal(Order(Z,6,2));
Generating polynomial: x^6 - 2
Discriminant: 1492992

kash> I := 2*o;
<2>
kash> elt := Elt(o,[214124,2144,3245,3252354545,436436436436,564643544365]);
[214124, 2144, 3245, 3252354545, 436436436436, 564643544365]
kash> b := EltIdealReduce(elt,I);
[0, 0, 1, 1, 0, 1]

```

NAME	EltIndex						
PURPOSE	Computes the index of an equation suborder of a given order						
SYNTAX	<pre>index := EltIndex (alpha [,Z]);</pre> <table> <tr> <td>integer</td><td>index</td></tr> <tr> <td>algebraic integer</td><td>alpha</td></tr> <tr> <td>ring of integers</td><td>Z</td></tr> </table>	integer	index	algebraic integer	alpha	ring of integers	Z
integer	index						
algebraic integer	alpha						
ring of integers	Z						
DESCRIPTION	<p>Let \mathfrak{o} be an arbitrary order over \mathbb{Z} and let α be an algebraic integer from \mathfrak{o}. The <code>EltIndex</code> function returns the module index $(\mathfrak{o} : \mathbb{Z}[\alpha])$. If the index is infinite, 0 is returned. In the relative case, \mathbb{Z} has to be the argument that determines the ring of the index.</p>						
SEE ALSO	OrderIndexFormEquation ,						
EXAMPLE	Compute the index $(\mathbb{Z}[\sqrt[4]{5}] : \mathbb{Z}[\sqrt[4]{5} + \sqrt{5}])$:						

```
kash> o := Order(Z,4,5);
Generating polynomial: x^4 - 5
```

```
kash> a := Elt(o,[0,1,1,0]);
[0, 1, 1, 0]
```

```
kash> EltIndex(a);
```

```
21
```

NAME	EltIsInIdeal
PURPOSE	missing shortdoc
SYNTAX	<pre> b := EltIsInIdeal (alpha,a); boolean b algebraic element alpha ideal a INDES test on; element of an ideal check for; element of an ideal algebraic number; element of an ideal test element of an ideal test </pre>
DESCRIPTION	The <code>EltIsInIdeal</code> function returns <code>true</code> if the algebraic number α lies in the ideal <code>a</code> . Otherwise <code>false</code> is returned.
EXAMPLE	<pre> kash> o := Order (Poly(Zx,[1,0,73,-280,-2399]));; kash> O := OrderMaximal (o);; kash> a := 2*O + Elt (O,[0, 1, -1, -2])*O; <2, [0, 1, -1, -2]> kash> EltIsInIdeal (2,a); true kash> EltIsInIdeal (1,a^-1); true kash> EltIsInIdeal (Elt(O,1/3),a^-1); false </pre>

NAME	EltIsInt
PURPOSE	missing shortdoc
SYNTAX	<pre>b := EltIsInt(a);</pre> <p> boolean b algebraic element a </p>
DESCRIPTION	The EltIsInt function returns true , if the algebraic number a is a rational integer. Otherwise false is returned.

EXAMPLE

```
kash> o := Order(Z,2,2);
Generating polynomial: x^2 - 2
```

```
kash> a := Elt(o,[0,1]);
[0, 1]
kash> EltIsInt (a);
false
kash> EltIsInt (a^2);
true
kash> EltIsInt (a^2/4);
false
```

NAME EltIsIntegral

PURPOSE Tests whether the element is an algebraic integer resp. an element of the order. The second parameter “order” checks, if the element is integral in the mathematical sence, e.g. if it has a denominator.

SYNTAX **b** := EltIsIntegral(**a**);
 b := EltIsIntegral(**a**, "order");

 boolean **b**
 algebraic element **a**

EXAMPLE

```
kash> o := Order(Z,4,3);
Generating polynomial: x^4 - 3
```

```
kash> a := Elt(o,[1,2,3,4]);
[1, 2, 3, 4]
kash> EltIsIntegral(a);
true
kash> EltIsIntegral(a/13);
false
```

```
kash> o:= Order(Z,2,5);
Generating polynomial: x^2 - 5
```

```
kash> a := Elt(o, [1,1]/2);
[1, 1] / 2
kash> EltIsIntegral(a);
true
kash> EltIsIntegral(a, "order");
false
```

NAME	EltIsPrimitive				
PURPOSE	Checks whether an algebraic element is primitive.				
SYNTAX	<pre>b := EltIsPrimitive (a);</pre> <table><tr><td>boolean</td><td>b</td></tr><tr><td>algebraic element</td><td>a</td></tr></table>	boolean	b	algebraic element	a
boolean	b				
algebraic element	a				
SEE ALSO	EltMinPoly ,				
EXAMPLE					

```
kash> O := Order(Z,4,2);  
Generating polynomial: x^4 - 2
```

```
kash> alpha := Elt(O, [0,0,1,0]);  
[0, 0, 1, 0]  
kash> EltIsPrimitive(alpha);  
false  
kash> beta := Elt(O, [0,0,0,1]);  
[0, 0, 0, 1]  
kash> EltIsPrimitive(beta);  
true
```

NAME	EltListAbsDisc
PURPOSE	Discriminant of a module.
SYNTAX	<pre>d := EltListAbsDisc (L);</pre> <p>list L of algebraic elements integer d</p>
DESCRIPTION	<p>Computes the (absolute) discriminant of the module $M = \mathfrak{o} \cdot L[1] + \cdots + \mathfrak{o} \cdot L[n]$, where \mathfrak{o} is the defining order of the elements of L and n is the number of elements in L. The discriminant is computed as the determinant of $(\text{Tr}_{\mathfrak{o}/\mathbb{Q}}(L[i]L[j]))$.</p>

EXAMPLE Discriminant of $\mathbb{Q}(\sqrt{10}, \sqrt{5})$.

```
kash> F := OrderMaximal (Order (Z,2,10));;
kash> E := Order (F,2,5);
      F[1]
      /
      /
      E1[1]
      /
      /
      Q
      F [ 1]      x^2 - 5
      E 1[ 1]     x^2 - 10

kash> EltListAbsDisc (OrderKextGenAbs (E));
0
kash> OrderDisc (OrderMaximal (OrderAbs (E)));
1600
```

NAME	EltListToMat
PURPOSE	Converts a list of algebraic numbers into a matrix of their coefficients.
SYNTAX	$M := \text{EltListToMat}(L);$ <p> matrix M matrix over the coefficient order of the elements of L list L of algebraic numbers </p>
DESCRIPTION	This procedure cares for the denominators of the elements.
SEE ALSO	EltMatToList ,

EXAMPLE

```

kash> o := OrderMaximal(Order(Z,2,3));;
kash> O := Order(o,2,7);
      F[1]
      /
      /
      E1[1]
      /
      /
      Q
F  [ 1]      x^2 - 7
E 1[ 1]      x^2 - 3

kash> alpha := Elt(0,[1,Elt(o,[0,1])]);
[1, [0, 1]]
kash> beta := Elt(0,[Elt(o,[0,2]),Elt(o,[1,1])]);
[[0, 2], [1, 1]]
kash> EltListToMat([alpha,beta]);
[1 [0, 2]]
[[0, 1] [1, 1]]

```


NAME	EltLogs
PURPOSE	Returns a matrix v with the logarithms of the absolute values of the conjugates the algebraic number a . Matrix v has length $r 1 + r 2$.
SYNTAX	<pre>v := EltLogs(a);</pre> <pre>real matrix v algebraic element a</pre>
DESCRIPTION	<p>Let $\tau 1, \dots, \tau s$ be all archimedean valuations of the the number field containing a. Then v is defined as</p> $v = [\tau 1(a), \dots, \tau s(a)].$
SEE ALSO	OrderSig , EltCon ,

EXAMPLE Matrix of logarithms of archimedean valuations of an algebraic element.

```
kash> 0 := Order (Poly(Zx, [1,0,73,-280,-2399]));
Generating polynomial: x^4 + 73*x^2 - 280*x - 2399

kash> rho := Elt(0, [0,1,0,0]);
[0, 1, 0, 0]
kash> EltLogs (rho);
[1.80894505471195860607076588169642554084529697289 1.3527212036195319561389981\
23553287717215552888911 2.310570502254102005606434026695873075937323698075]
```

NAME	EltMatToList
PURPOSE	Converts a matrix of algebraic numbers over a maximal order o into a dynamic array of elements of the relative order O .
SYNTAX	$L := \text{EltMatToList}(O, M);$ <p> <code>list</code> L of algebraic numbers over O <code>matrix</code> M of algebraic numbers <code>order</code> O relative order over the coefficient order of the elements of the matrix </p>
DESCRIPTION	Denominators are collected to a single denominator (lcm) and the coefficients are multiplied accordingly.
SEE ALSO	EltListToMat ,

EXAMPLE

```

kash> o := OrderMaximal(Order(Z,2,3));;
kash> O := Order(o,2,7);
      F[1]
      /
      /
      E1[1]
      /
      /
      Q
      F [ 1]      x^2 - 7
      E 1[ 1]      x^2 - 3

kash> M := Mat(o, [ [1, Elt(o,[0,2])], [Elt(o,[0,1]), Elt(o,[1,1])] ]);
      [1 [0, 2]]
      [[0, 1] [1, 1]]
kash> EltMatToList(O, M);
      [ [1, [0, 1]], [[0, 2], [1, 1]] ]

```

NAME	EltMinPoly
PURPOSE	Minimal polynomial of an algebraic element over a subfield.
SYNTAX	<pre>p := EltMinPoly (a [, PA]); p := EltMinPoly (a [, 0]);</pre> <p> polynomial p polynomial algebra PA suborder 0 algebraic element a </p>
DESCRIPTION	Given an algebraic element a in an Order O and a polynomial algebra PA over another order o or this order, this function computes the minimal polynomial of a over o i.e. the returned polynomial is contained in PA .

EXAMPLE Minimal polynomials of elements in $\mathbb{Q}(\sqrt{10}, \sqrt{5}, \sqrt{3})$.

```
kash> E1 := Order (Z,2,10);
Generating polynomial: x^2 - 10
```

```
kash> E2 := Order (E1, 2, 5);
```

```

      F[1]
      /
      /
    E1[1]
    /
    /
  Q
F [ 1]    x^2 - 5
E 1[ 1]    x^2 - 10
```

```
kash> F := Order (E2, 2, 3);
```

```

      F[1]
      /
      /
    E2[1]
    /
    /
  E1[1]
  /
  /
Q
```

```

F [ 1]      x^2 - 3
E 2[ 1]      x^2 - 5
E 1[ 1]      x^2 - 10

```

```
kash> E1x := PolyAlg (E1);
```

```
Univariate Polynomial Ring in x over Generating polynomial: x^2 - 10
```

```
kash> Fx := PolyAlg (F);
```

```
Univariate Polynomial Ring in x over          F[1]
```

```

      /
     /
    E2[1]
     /
    /
   E1[1]
  /
 /
Q
F [ 1]      x^2 - 3
E 2[ 1]      x^2 - 5
E 1[ 1]      x^2 - 10

```

```
kash> a := Elt (F, [[[1,2],[3,4]],[[3/5, 5],[7,9]]]);
```

```
[[[5, 10], [15, 20]], [[3, 25], [35, 45]]] / 5
```

```
kash> g := EltMinPoly (a, E1x);
```

```

x^4 + [-4, -8]*x^3 + [-717904, -100800] / 25*x^2 + [-3443792, -2723584] / 25*x\
+ [99481555504, 24875390400] / 625

```

```
kash> f := PolyMove (g, Fx);
```

```

x^4 + [[[-4, -8], 0], 0]*x^3 + [[[-717904, -100800], 0], 0] / 25*x^2 + [[[-344\
3792, -2723584], 0], 0] / 25*x + [[[99481555504, 24875390400], 0], 0] / 625

```

```
kash> Eval (f, a);
```

```
0
```

NAME	EltMinkowski
PURPOSE	Returns the image of an algebraic element under the Minkowski map.
SYNTAX	$v := \text{EltMinkowski}(a);$ <div style="margin-left: 100px;"> $\text{real matrix} \quad v$ $\text{algebraic element} \quad a$ </div>
DESCRIPTION	<p>Let $F = \mathbb{Q}(O)$ be the field of fractions of the order O, which contains the algebraic element a and let $\sigma 1, \dots, \sigma n$ be the Galois isomorphisms of F, where $\sigma 1, \dots, \sigma r 1$ are precisely the real isomorphisms and $\overline{\sigma j+r 2} = \sigma j$ for $r 1 < j \leq r 1 + r 2$ are the complex ones. Then the image of a under the Minkowski map $\psi : O \mapsto \mathbb{R}^{r 1+2r 2}$ is defined as</p>

$$[\sigma|1(a), \dots, \sigma|r|1(a), \sqrt{2}\Re(\sigma|r|1 + r|2(a)), \dots, \sqrt{2}\Re(\sigma|r|1 + r|2(a)), \\ \sqrt{2}\Im(\sigma|r|1 + r|2(a)), \dots, \sqrt{2}\Im(\sigma|r|1 + r|2(a))].$$

EXAMPLE Matrix of logarithms of archimedean valuations of an algebraic element.

```
kash> O := Order (Z,2,-1);
Generating polynomial: x^2 + 1

kash> rho := Elt (O,[0,1]);
[0, 1]
kash> EltMinkowski (rho);
[ 0, 1.414213562373095048801688724209698078569671875377]
kash> O := Order (Poly(Zx,[1,0,73,-280,-2399]));
Generating polynomial: x^4 + 73*x^2 - 280*x - 2399

kash> rho := Elt (O,[1,1,0,1]);
[1, 1, 0, 1]
kash> EltMinkowski (rho);
[ 234.532336541434390915922266933690607679916764155403, -60.735882043911047217\
34054416286226981382540207225, 473.9054718000681082230949125242621731996360759\
3719, -1354.559212605075329197656456425992015910014499693796]
```

NAME	EltMove
PURPOSE	Returns an algebraic element in a different representation.
SYNTAX	<pre>b := EltMove(a,S);</pre> <p> algebraic element or lattice element b algebraic element or lattice element a order or lattice S </p>
DESCRIPTION	The element a is contained in a certain order or lattice and is therefore given in a specific representation. This function tries to give a representation of a in the structure S.
SEE ALSO	PolyMove , IdealMove ,
EXAMPLE	Moving elements in relative extensions.

```

kash> o := Order (Poly(Zx,[1,0,73,-280,-2399]));;
kash> O := OrderMaximal (o);;
kash> oo := Order (O,2,3);;
kash> a := Elt (O,[0,0,0,1]);;
kash> OrderBasis (O);
[ 1, [0, 1, 0, 0], [1, 1, 1, 0] / 2, [414, 857, 795, 1] / 1646 ]
kash> EltMove (a,o);
[414, 857, 795, 1] / 1646
kash> EltMove (a,oo);
[[0, 0, 0, 1], 0]

```

NAME	EltNewtonLift												
PURPOSE	Lifts an algebraic element with the Newton lifting method.												
SYNTAX	<pre>alpha := EltNewtonLift(o, a, f, p, k);</pre> <table> <tr> <td>algebraic element</td><td>alpha</td></tr> <tr> <td>order</td><td>o</td></tr> <tr> <td>algebraic element</td><td>a</td></tr> <tr> <td>polynomial</td><td>f</td></tr> <tr> <td>integer</td><td>p</td></tr> <tr> <td>integer</td><td>k</td></tr> </table>	algebraic element	alpha	order	o	algebraic element	a	polynomial	f	integer	p	integer	k
algebraic element	alpha												
order	o												
algebraic element	a												
polynomial	f												
integer	p												
integer	k												
DESCRIPTION	Given an algebraic element a with $f(a) \equiv 0 \pmod{p}$, this function calculates an algebraic element α with $f(\alpha) \equiv 0 \pmod{p^{2^k}}$.												
EXAMPLE	<pre>kash> o:=Order(Poly(Zx,[1,0,-4,0,1])); Generating polynomial: x^4 - 4*x^2 + 1 kash> f:=Poly(Zx,[1,40,596,3920,9601]); x^4 + 40*x^3 + 596*x^2 + 3920*x + 9601 kash> a:=Elt(o,[0,4,0,0]); [0, 4, 0, 0] kash> b:=EltNewtonLift(o,a,f,5,2); [-10, -1, 0, 0] kash> Eval(f, b); 0</pre>												

NAME EltNorm

PURPOSE Returns the norm of an algebraic element in a given order.

SYNTAX n := EltNorm(a [,o]);

 rational number or algebraic element n

 algebraic element a

 order o

SEE ALSO [EltTrace](#),

EXAMPLE Computation of a norm of an algebraic element.

```
kash> o := Order (Poly(Zx,[1,0,73,-280,-2399]));;
```

```
kash> O := OrderMaximal (o);;
```

```
kash> oo := Order (O,2,3);
```

```
      F[1]
```

```
      /
```

```
      /
```

```
      E1[1]
```

```
      |
```

```
      E1[2]
```

```
      /
```

```
      /
```

```
Q
```

```
F [ 1]       x^2 - 3
```

```
E 1[ 1]       Given by transformation matrix
```

```
E 1[ 2]       x^4 + 73*x^2 - 280*x - 2399
```

```
kash> a := Elt (oo,[[0,0,0,1],4]/2);;
```

```
kash> EltNorm (a);
```

```
[705, 55, -765, 758] / 4
```


NAME	EltNumberReduce
PURPOSE	Returns a canonical representative modulo an integer or rational.
SYNTAX	<pre> b := EltNumberReduce(a,m); integer or rational number m algebraic element a algebraic element b </pre>
DESCRIPTION	Returns a canonical representation of an algebraic element modulo an integer or a rational.
EXAMPLE	<pre> kash> o := OrderMaximal(Order(Z,6,2)); Generating polynomial: x^6 - 2 Discriminant: 1492992 kash> elt := Elt(o,[214124,2144,3245,3252354545,436436436436,564643544365]); [214124, 2144, 3245, 3252354545, 436436436436, 564643544365] kash> b := EltNumberReduce(elt,2); [0, 0, 1, 1, 0, 1] </pre>

NAME	EltOrder
PURPOSE	Returns the order of an algebraic element.
SYNTAX	<pre>o := EltOrder(alpha);</pre> <div style="display: flex; justify-content: space-around; margin-top: 10px;"> <div style="text-align: center;"> <code>order</code> algebraic element or integer </div> <div style="text-align: center;"> <code>o</code> alpha </div> </div>
DESCRIPTION	The EltOrder function returns the order in which the algebraic element α is defined.
EXAMPLE	

```
kash> O:=Order(Z,2,-3);
Generating polynomial: x^2 + 3
```

```
kash> alpha:=Elt(0,[1,2]);
[1, 2]
kash> EltOrder(alpha);
Generating polynomial: x^2 + 3
```

NAME	EltPowerMod						
PURPOSE	Computes $a^{\text{pow}} \bmod M$ for an algebraic element.						
SYNTAX	<pre>e := EltPowerMod(a, pow, M);</pre> <table> <tr> <td>algebraic element</td><td>e, a</td></tr> <tr> <td>integer</td><td>pow</td></tr> <tr> <td>integer</td><td>M</td></tr> </table>	algebraic element	e, a	integer	pow	integer	M
algebraic element	e, a						
integer	pow						
integer	M						

EXAMPLE

```
kash> o := Order(Z,6,2);
Generating polynomial: x^6 - 2

kash> a := Elt(o,[1,2,3,4,5,6]);
[1, 2, 3, 4, 5, 6]
kash> EltPowerMod(a, 10, 5);
[1, 1, 1, 0, 2, 1]
```

NAME	EltPowerProduct								
PURPOSE	Returns an algebraic element β with the power product of the algebraic elements in Alpha and the corresponding exponents in Expons. The matrices Alpha and Expons must have the same length and only one row.								
SYNTAX	<pre>beta := EltPowerProduct(o,Alpha,Expons);</pre> <table> <tr> <td>algebraic element</td><td>beta</td></tr> <tr> <td>order</td><td>o</td></tr> <tr> <td>matrix of algebraic elements</td><td>Alpha</td></tr> <tr> <td>matrix of integers</td><td>Expons</td></tr> </table>	algebraic element	beta	order	o	matrix of algebraic elements	Alpha	matrix of integers	Expons
algebraic element	beta								
order	o								
matrix of algebraic elements	Alpha								
matrix of integers	Expons								
DESCRIPTION	Returns an algebraic element β with the power product of the algebraic elements in Alpha and the corresponding exponents in Expons. The matrices Alpha and Expons must have the same length and only one row.								
EXAMPLE	Power product of algebraic elements.								

NAME	EltRayResidueRingRep								
PURPOSE	Returns a representative of the class of an algebraic element in the multiplicative group of the ray residue ring.								
SYNTAX	<pre>r := EltRayResidueRingRep(elt,m0,minf);</pre> <table> <tr> <td>matrix</td><td>r</td></tr> <tr> <td>algebraic element</td><td>elt</td></tr> <tr> <td>ideal</td><td>m0</td></tr> <tr> <td>list</td><td>minf of integers/infinite primes</td></tr> </table>	matrix	r	algebraic element	elt	ideal	m0	list	minf of integers/infinite primes
matrix	r								
algebraic element	elt								
ideal	m0								
list	minf of integers/infinite primes								
DESCRIPTION	<p>Calculates a representative of an algebraic element <code>elt</code> which is represented by a matrix $[a_0, \dots, a_k]$. If η_0, \dots, η_k is a basis of the residue ring as given by <code>RayResidueRingCyclicFactors</code>, $\text{elt} \equiv \eta_0^{a_0} \cdots \eta_k^{a_k} \pmod{(\mathbf{m}_0 \mathbf{m}_\infty)}$ holds.</p> <p>If the residue ring is not known yet, it will be computed. Infinite primes are represented in the same way as for <code>RayResidueRing</code>.</p> <p>This function uses a mixture of the discrete logarithm algorithms developed in [Pau96, PP98] and [CDO96, CDO97].</p>								
SEE ALSO	<code>EltCon</code> , <code>RayResidueRing</code> , <code>RayResidueRingRepToElt</code> , <code>RayResidueRingCyclicFactors</code> ,								

EXAMPLE

```
kash> O := OrderMaximal(Order(x^2-2*x-5));
Generating polynomial: x^2 - 2*x - 5
Discriminant: 24

kash> m0 := 27*O;;
kash> minf := [2];;
kash> L := RayResidueRingCyclicFactors(m0,minf);
[ [ [-1, -27], 2 ], [ [1344, 388], 3 ], [ [5, 2], 9 ], [ [1612, 444], 9 ],
  [ [1, 27], 2 ] ]
kash> elt := L[1][1];
[-1, -27]
kash> EltRayResidueRingRep(elt,m0,minf);
[1 0 0 0 0]
```

NAME	EltReconstruct
PURPOSE	Lifts an element from a modulo m approximation to an element with rational coefficients
SYNTAX	<pre>alpha := EltReconstruct (gamma, m);</pre> <pre> false or algebraic element alpha algebraic element gamma integer m </pre>
DESCRIPTION	<p>Given an element $\gamma = c_1\omega_1 + \dots + c_n\omega_n$ in an order over \mathbb{Z} with coefficients less than the positive integer m, the function EltReconstruct computes an element α in the same order which is equal to the element $q_1\omega_1 + \dots + q_n\omega_n$, where the q_i are the reconstructed rationals from c_i and m ($q_i = \frac{a_i}{b_i}$ such that $a_i \equiv b_i c_i \pmod{m}$ and $0 \leq a_i , b_i < \sqrt{\frac{m}{2}}, b_i \neq 0$, if such a pair exists). Otherwise false is returned.</p>
SEE ALSO	RationalReconstruct ,
EXAMPLE	

```
kash> O := Order(Z,2,3);
Generating polynomial: x^2 - 3
```

```
kash> gamma := Elt(0,[17,4]);
[17, 4]
kash> EltReconstruct(gamma,49);
[2, 12] / 3
```

NAME	EltRepMat
PURPOSE	A representation matrix of an algebraic element over o together with a suitable denominator.
SYNTAX	$L := \text{EltRepMat}(a[,o]);$ <p> list L algebraic element a order o must be a direct suborder of EltOrder (a) </p>
DESCRIPTION	Given an algebraic element a in an order O and another order o , such that o is a direct suborder of O , this function computes a list consisting of an integer den and a matrix M , such that $1/den \cdot M$ is a representation matrix of a with coefficients in o .

EXAMPLE The representation matrix of certain algebraic elements.

```
kash> o := Order (Poly(Zx,[1,0,73,-280,-2399]));
Generating polynomial: x^4 + 73*x^2 - 280*x - 2399
```

```
kash> O1 := Order (o,2,2);
      F[1]
      /
      /
      E1[1]
      /
      /
      Q
      F [ 1]      x^2 - 2
      E 1[ 1]      x^4 + 73*x^2 - 280*x - 2399
```

```
kash> O2 := Order (O1, 2, 3);
      F[1]
      /
      /
      E2[1]
      /
      /
      E1[1]
      /
      /
      Q
```

```

F [ 1]      x^2 - 3
E 2[ 1]      x^2 - 2
E 1[ 1]      x^4 + 73*x^2 - 280*x - 2399

```

```

kash> alpha := Elt (O2,[[[1,2,3,4],1],[2,[7,8,9,0]]]/ 34);
[[[1, 2, 3, 4], 1], [2, [7, 8, 9, 0]]] / 34
kash> EltRepMat (alpha,o);
[ 34, [[1, 2, 3, 4] 2 6 [42, 48, 54, 0]]
      [1 [1, 2, 3, 4] [21, 24, 27, 0] 6]
      [2 [14, 16, 18, 0] [1, 2, 3, 4] 2]
      [[7, 8, 9, 0] 2 1 [1, 2, 3, 4]] ]

```


NAME	EltRoot
PURPOSE	Computes a root of an algebraic element.
SYNTAX	<pre> beta := EltRoot(alpha,m); beta := EltRoot(alpha,m [, "enum" "mode"]); beta := EltRoot(alpha,m , "enum", mode); false or algebraic element beta algebraic element alpha small integer m small integer or string mode string enum </pre>
DESCRIPTION	<p>Let α be an algebraic element from a maximal order \mathcal{O}.</p> <p><code>EltRoot(alpha,m)</code> (or <code>EltRoot(alpha,m,"mode")</code>) The <code>EltRoot</code> function checks whether an algebraic number $\beta \in \mathcal{O}$ exists such that $\beta^m = \alpha$. The <code>EltRoot</code> uses a generalization of a method described in P. L. Montgomery, <i>Square roots of products of algebraic numbers</i>, Proceedings of Symposia in Applied Mathematics, Vol. 48 (1994), 567 – 571.</p> <p><code>EltRoot(alpha,m,"enum")</code> (or <code>EltRoot(alpha,m,"enum",0)</code>) The <code>EltRoot</code> function does the same as above, but a method is used which bases on the enumeration of a weighted positive definite quadratic form.</p> <p><code>EltRoot(alpha,m,"enum",1)</code> The <code>EltRoot</code> function checks whether an algebraic number $\beta \in \mathcal{O}$ and a torsion unit $\zeta \in \mathcal{O}$ exist such that $\beta^m = \zeta \alpha$.</p> <p><code>EltRoot(alpha,m,"enum",2)</code> The <code>EltRoot</code> function checks whether an algebraic number $\beta \in \mathcal{O}$ and a unit $\varepsilon \in \mathcal{O}$ exist such that $\beta^m = \varepsilon \alpha$.</p> <p>EXAMPLE Computation of roots of some algebraic elements:</p>

```

kash> o := Order (Poly(Zx,[1,0,73,-280,-2399]));;
kash> o := OrderMaximal (Order (Poly(Zx,[1,0,73,-280,-2399])));;
kash> a := Elt (o,[48427150173, 2658413621, -99500340384, 103801743979]);
[48427150173, 2658413621, -99500340384, 103801743979]
kash> EltRoot (a, 2);
[156369, 8584, -321281, 335170]
kash> o := OrderMaximal (Order (Z,7,2));;
kash> a := Elt (o, 2);

```

2

```
kash> EltRoot (a,2);
```

```
false
```

```
kash> EltRoot (a,7);
```

```
[0, 1, 0, 0, 0, 0, 0]
```

NAME	EltSimplify
PURPOSE	Returns a in simplified representation.
SYNTAX	<pre>e := EltSimplify(a);</pre> <p>integer or algebaric element e integer or algebraic element a</p>
DESCRIPTION	<p>This function returns an improved representation of the algebraic element a. It might be the case that a has a simpler basis representation than given. EltSimplify will find the simplest representation possible. This function is important for programs written in KASH, since the output of an algebraic number is always simplified, although the number itself can have a bad representation.</p>
EXAMPLE	There is no example demonstrating the effect, since output is always simplified.

NAME	EltToFFE
PURPOSE	Returns the class of an algebraic element viewed as an element of a finite field.
SYNTAX	<pre>f := EltToFFE(a, p);</pre> <p> integer or algebraic element a prime ideal p finite field element or integer f interpreted as finite field element </p>
DESCRIPTION	<p>This is the canonical homomorphism</p> $\mathcal{O} \rightarrow \mathcal{O}/\mathfrak{p}.$ <p>The prime ideal must not lie over a number which divides the discriminant of the generating polynomial of \mathcal{O}.</p>
SEE ALSO	IdealResidueField , IdealResidueFieldIsomorphism , FFToElt ,
EXAMPLE	

```
kash> O := OrderMaximal(Order(Poly(Zx,[1,4,1,-4,-3,7])));
Generating polynomial: x^5 + 4*x^4 + x^3 - 4*x^2 - 3*x + 7
Discriminant: 28442269
```

```
kash> p := Factor(7*O)[1][1];
<7, [0, 1, 0, 0, 0]>
kash> b := Elt(O,[3, 1,5,1,8]);
[3, 1, 5, 1, 8]
kash> EltToFFE(b, p);
```

```
3
```

NAME	EltToList
PURPOSE	Returns the coefficient list of the algebraic element a .
SYNTAX	$L := \text{EltToList}(a);$ <div style="display: flex; justify-content: space-between; margin-top: 10px;"> <div style="text-align: left;"> <p>algebraic element</p> <p>list</p> </div> <div style="text-align: right;"> <p>a</p> <p>L of integers or algebraic elements</p> </div> </div>

EXAMPLE

```

kash> o := Order(Z,6,2);
Generating polynomial: x^6 - 2

kash> a := Elt(o,[1,2,3,4,5,6]);
[1, 2, 3, 4, 5, 6]
kash> EltToList(a);
[ 1, 2, 3, 4, 5, 6 ]
kash> a := a/2;
[1, 2, 3, 4, 5, 6] / 2
kash> EltToList(a);
[ 1/2, 1, 3/2, 2, 5/2, 3 ]

```

NAME	EltTrace
PURPOSE	Returns the trace of an algebraic element. The trace is contained in the coefficient ring of the defining order or in the given order <i>o</i> .
SYNTAX	<pre>t := EltTrace (a [,o]);</pre> <p> rational number or algebraic element <i>t</i> algebraic element <i>a</i> order <i>o</i> </p>

SEE ALSO [EltNorm](#),

EXAMPLE Computation of several traces.

```
kash> o := Order (Poly(Zx,[1,0,73,-280,-2399]));;
```

```
kash> O := OrderMaximal (o);;
```

```
kash> oo := Order (O,2,3);
```

```
F[1]
```

```
/
```

```
/
```

```
E1[1]
```

```
|
```

```
E1[2]
```

```
/
```

```
/
```

```
Q
```

```
F [ 1]        x^2 - 3
```

```
E 1[ 1]        Given by transformation matrix
```

```
E 1[ 2]        x^4 + 73*x^2 - 280*x - 2399
```

```
kash> a := Elt (O,[0,0,0,1]);;
```

```
kash> b := EltMove (a,oo);
```

```
[[0, 0, 0, 1], 0]
```

```
kash> EltTrace (b);
```

```
[0, 0, 0, 2]
```

```
kash> EltTrace (EltTrace (b));
```

```
-138
```

NAME	EltUnitDecompose
PURPOSE	Returns the decomposition of an unit with respect to the computed system of fundamental units of a maximal order.
SYNTAX	<pre> F := EltUnitDecompose(u); L := EltUnitDecompose(u,"expons"); list F list L algebraic element u </pre>
DESCRIPTION	<p>Let \mathfrak{o} be an order over \mathbb{Z}. Let $\varepsilon_1, \dots, \varepsilon_r$ be the system of fundamental units of \mathfrak{o} computed by the <code>OrderUnitsFund</code> function and let $\zeta \in \mathfrak{o}$ be the generator of the cyclic torsion subgroup which is accessible by the <code>OrderTorsionUnit</code> function. Each unit $u \in \mathfrak{o}$ is of a form</p> $u = \zeta^{m_0} \varepsilon^{m_1} \dots \varepsilon^{m_r}$ <p>with rational integers m_0, m_1, \dots, m_r.</p> <pre> F := EltUnitDecompose(u) returns the above decomposition of u. L := EltUnitDecompose(u,"expons") returns the exponents m_0, m_1, \dots, m_r of the above decomposition of u. </pre>

EXAMPLE Decompose certain units in $\mathbb{Z}[\sqrt[4]{2}]$:

```

kash> o := Order(Z,4,2);
Generating polynomial: x^4 - 2

kash> L := OrderUnitsFund(o);
[ [1, 0, 1, 0], [1, -1, 0, 0] ]
kash> u := L[1]; v:= L[2];
[1, 0, 1, 0]
[1, -1, 0, 0]
kash> EltUnitDecompose(u*v);
[ [ [1, 0, 1, 0], 1 ], [ [1, -1, 0, 0], 1 ] ]
kash> EltUnitDecompose(u*v,"expons");
[ 0, 1, 1 ]
kash> EltUnitDecompose(-u*v);
[ [ -1, 1 ], [ [1, 0, 1, 0], 1 ], [ [1, -1, 0, 0], 1 ] ]
kash> EltUnitDecompose(-u*v,"expons");

```

```
[ 1, 1, 1 ]  
kash> EltUnitDecompose(u^3*v^2);  
[ [ [1, 0, 1, 0], 3 ], [ [1, -1, 0, 0], 2 ] ]  
kash> EltUnitDecompose(u^3*v^2,"expons");  
[ 0, 3, 2 ]
```


EulerGamma

NAME EulerGamma

PURPOSE Returns the value of the Euler constant γ .

SYNTAX `y := EulerGamma();`

`real y`

DESCRIPTION

EXAMPLE

```
kash> EulerGamma();  
0.5772156649015328606065120900824024310421593359399236
```

NAME	EutacticCoef								
PURPOSE	Returns a list of eutactic coefficients of a Humbert form in two variables								
SYNTAX	<p><code>E := EutacticCoef(HF,M,n);</code></p> <table> <tr> <td>list</td><td>L</td></tr> <tr> <td>Humbert form with $S_i \in \overline{Q}^{2 \times 2}$</td><td>HF=[S1,S2]</td></tr> <tr> <td>List of minimal vectors of HF</td><td>M</td></tr> <tr> <td>integers,</td><td>n</td></tr> </table>	list	L	Humbert form with $S_i \in \overline{Q}^{2 \times 2}$	HF=[S1,S2]	List of minimal vectors of HF	M	integers,	n
list	L								
Humbert form with $S_i \in \overline{Q}^{2 \times 2}$	HF=[S1,S2]								
List of minimal vectors of HF	M								
integers,	n								
DESCRIPTION	<p>Given a Humbert form in two variables with algebraic entries and its minimal vectors, it will return a list of positive (in the n-th conjugate field) algebraic elements if the given Humbert form is eutactic. This means that the given humbert form is in the linear convex hull of matrix tuples, which will be received by its minimal vectors. All computations would be made in the n-th conjugate field from which the entries of the humbert form are.</p> <p>If the form is not eutactic a list of zeros is returned.</p>								

SEE ALSO [MinVec](#),

EXAMPLE

```
kash> o := OrderMaximal(x^2-3);
Generating polynomial: x^2 - 3
Discriminant: 12
```

```
kash> u := Elt(o,[2, 1]);
[2, 1]
kash> us := Elt(o,[2,-1]);
[2, -1]
kash> ur := Re(EltCon(Elt(o, [2, 1]))[1][1]);
3.73205080756887729352744634150587236694280525381
kash> urs := Re(EltCon(Elt(o, [2, 1]))[1][2]);
0.26794919243112270647255365849412763305719474619
```

```
kash> S1 := Mat(R, [ [ 1, ur/2 ], [ur/2, ur ] ]);
[1 1.866025403784438646763723170752936183471402626905]
[1.866025403784438646763723170752936183471402626905 3.732050807568877293527446\
34150587236694280525381]
kash> S2 := Mat(R, [ [ 1, urs/2 ], [urs/2, urs] ]);
[1 0.133974596215561353236276829247063816528597373095]
```

```
[0.133974596215561353236276829247063816528597373095 0.267949192431122706472553\
65849412763305719474619]
kash> SA1 := Mat(o, [ [ 1, u/2 ], [u/2, u ] ]);
[1 [2, 1] / 2]
[[2, 1] / 2 [2, 1]]
kash> SA2 := Mat(o, [ [ 1, us/2 ], [ us/2,us ] ]);
[1 [2, -1] / 2]
[[2, -1] / 2 [2, -1]]
kash> HF := [S1,S2];
[ [1 1.866025403784438646763723170752936183471402626905]
  [1.866025403784438646763723170752936183471402626905 3.73205080756887729352\
744634150587236694280525381],
  [1 0.133974596215561353236276829247063816528597373095]
  [0.133974596215561353236276829247063816528597373095 0.26794919243112270647\
255365849412763305719474619] ]
kash> HFA := [SA1,SA2];
[ [1 [2, 1] / 2]
  [[2, 1] / 2 [2, 1]], [1 [2, -1] / 2]
  [[2, -1] / 2 [2, -1]] ]
kash> L := MinVec(HF,o,0)[1];
[ [ [0, 1], [1, -1] ], [ [0, 1], -1 ], [ [1, 1], [0, -1] ], [ [1, 1], -1 ],
  [ [2, 1], [0, -1] ], [ [2, 1], -2 ], [ 0, 1 ], [ 1, [1, -1] ],
  [ 1, [-2, 1] ], [ 1, -1 ], [ 1, 0 ], [ 2, -1 ] ]
kash> EutacticCoef(HFA,L,1);
[ 20 / 243, 55 / 486, 8 / 243, 8 / 243, 44 / 243, 56 / 243, 77 / 243, 1 / 9,
  23 / 162, 44 / 243, 56 / 243, 28 / 81 ]
```

NAME	<code>Eval</code>
PURPOSE	Evaluates a polynomial at a value.
SYNTAX	$y := \text{Eval}(f, s);$ <div style="text-align: center;"> $\begin{array}{c} y \\ \text{polynomial } f \\ s \end{array}$ </div>
DESCRIPTION	This function assumes that the arithmetic with elements of the coefficient ring of the polynomial algebra of f and s is defined.

EXAMPLE First we create a polynomial (over \mathbb{Z}):

```
kash> f := x^2+2*x+1;
x^2 + 2*x + 1
```

Now we want to evaluate it at 2 and π :

```
kash> Eval(f, 2);
9
kash> Eval(f, pi);
17.15278970826894509575977776643515690370803820599
```

Finally, we want to substitute x by $x+1$:

```
kash> Eval(f, x+1);
x^2 + 4*x + 4
```

Exp

NAME	Exp
PURPOSE	Returns the exponential of a number.
SYNTAX	$y := \text{Exp}(x);$ complex y complex x
DESCRIPTION	Given a number x the function returns e^x . The computation is done in the current precision of the real (complex) field.
SEE ALSO	Log ,
EXAMPLE	

```
kash> Exp(1);
2.718281828459045235360287471352662497757247093699
kash> i := Comp(0, 1);
1*i
kash> Exp(i);
0.5403023058681397174009366074429766037323104206179222 + 0.8414709848078965066\
52502321630298999622563060798371*i
```

NAME	FF
PURPOSE	Creates a finite field.
SYNTAX	<pre> F := FF(p); F := FF(p, d); F := FF(f); finite field F integer p integer d polynomial f </pre>
DESCRIPTION	<p>A finite field can be created in three different ways:</p> <ul style="list-style-type: none"> • <code>FF(p)</code> returns the finite field \mathbb{F}_p where p is a prime. • <code>FF(p, d)</code> returns the finite field \mathbb{F}_{p^d} where p is a prime and d a positive integer. • <code>FF(f)</code> returns an extension field of a finite field: f is a monic, irreducible polynomial over a finite field \mathbb{F}_q (q a prime power) of degree $l > 1$; \mathbb{F}_{q^l} is returned.
SEE ALSO	<code>FFElt</code> , <code>FFGenerator</code> , <code>FFPrimitiveElt</code> ,

EXAMPLE Creation of finite fields:

```

kash> F2 := FF(2);
Finite field of size 2
kash> F2x := PolyAlg(F2);
Univariate Polynomial Ring in x over GF(2)

kash> f := Poly(F2x, [1, 1, 1]);
x^2 + x + 1
kash> F4 := FF(f);
Finite field of size 2^2
kash> F9 := FF(3, 3);
Finite field of size 3^3

```

NAME FFCreate

PURPOSE Creates a finite field.

SYNTAX FFCreate(0,p,k)

 order 0

 integer p

 integer k

DESCRIPTION The finite field with p^k elements is created and stored in 0. This routine differs from `FunFF` in that it only allows one way of generating the field; however, it has the advantage that one can work relatively.

SEE ALSO [kantff](#),

EXAMPLE

```
kash> 0:=OrderMaximal(Poly(Zx,[1,0,-4,0,1]));  
Generating polynomial: x^4 - 4*x^2 + 1  
Discriminant: 2304
```

```
kash> FFCreate(0, 3, 4);  
Finite field of size 3^4
```


NAME	FFToElt
PURPOSE	Returns a canonical representative of a finite field element viewed as a representative of a class of algebraic numbers.
SYNTAX	<pre>a := FFToElt(f, p);</pre> <p> finite field element f ideal p must be prime algebraic number or integer a interpreted as algebraic number </p>
DESCRIPTION	<p>This is a inverse function of EltToFFE, and a embedding</p> $\mathcal{O}/\mathfrak{p} \rightarrow \mathcal{O}.$ <p>The prime ideal must not lie over a number which divides the degree of the order.</p>
SEE ALSO	IdealResidueField , IdealResidueFieldIsomorphism , EltToFFE ,

EXAMPLE

```
kash> O := OrderMaximal(Order(Poly(Zx,[1,4,1,-4,-3,7])));
Generating polynomial: x^5 + 4*x^4 + x^3 - 4*x^2 - 3*x + 7
Discriminant: 28442269
```

```
kash> p := Factor(7*O)[1][1];
<7, [0, 1, 0, 0, 0]>
kash> b := Elt(0,[3, 1,5,1,8]);
[3, 1, 5, 1, 8]
kash> n := EltToFFE(b, p);
3
kash> FFToElt(n, p);
3
```

NAME	FFElt
PURPOSE	Creates an element of a finite field.
SYNTAX	<pre>a := FFElt(F, n);</pre> <div> <div>finite field element</div> <div>finite field</div> <div>integer</div> <div>a</div> <div>F</div> <div>n</div> </div>
DESCRIPTION	Embeds the integer n into a finite field \mathbb{F}_q .
SEE ALSO	FFPrimitiveElt , FFGenerator , FFEltToList ,
EXAMPLE	Creation of a finite field element and some arithmetic:

```
kash> F5 := FF(5);
Finite field of size 5
kash> 1 + 4;
5
kash> a := FFElt(F5, 1) + 4;
0
```

NAME FFEltFF

PURPOSE Gives the finite field corresponding to the element.

SYNTAX `F := FFEltFF(a);`

 finite field element a
 finite field F

DESCRIPTION This function returns the finite field in which the given element is defined.

SEE ALSO `FFElt`, `FF`,

EXAMPLE

```
kash> F4 := FF(2, 2);
Finite field of size 2^2
kash> a := FFPrimitiveElt(F4);
w
kash> F := FFEltFF(a);
Finite field of size 2^2
```

NAME FFEltIsZero

PURPOSE Returns true iff the argument is of type "KANT finite field elt" AND zero.

SYNTAX `b := FFEltIsZero(a);`

 boolean b
 a

EXAMPLE

```
kash> k := FF(2,4);
Finite field of size 2^4
kash> a := FFElt(k,0);
0
kash> FFEltIsZero(a);
true
```

NAME	FFEltLog
PURPOSE	Discrete logarithm for finite fields.
SYNTAX	<pre>d := FFEltLog(a);</pre> <p> finite field element a integer d </p>
DESCRIPTION	<p>This function computes the discrete logarithm of an element a with respect to a primitive element ω, that means a non negative integer d with $a = \omega^d$. The primitive element ω is that returned by <code>FFPrimitiveElt()</code>.</p>
SEE ALSO	FFPrimitiveElt ,
EXAMPLE	Compute some discrete logarithms:

```

kash> F125 := FF(5, 3);
Finite field of size 5^3
kash> a := FFElt(F125, 3);
3
kash> d := FFEltLog(a);
93
kash> w := FFPrimitiveElt(F125);
w
kash> w^d;
3
kash> FFEltLog(w^47);
47

```

NAME	FFEltMinPoly
PURPOSE	Minimal polynomial of finite field elements.
SYNTAX	<pre>p := FFEltMinPoly(a); p := FFEltMinPoly(a, J); p := FFEltMinPoly(a, Jx); polynomial p finite field element a finite field J subfield of the finite field of <i>a</i> polynomial algebra in x over J Jx</pre>
DESCRIPTION	<p>This function computes the minimal polynomial of a finite field element over the base field of the finite field or over the finite field given as second parameter. It is also possible to specify the polynomial algebra in which the minimal polynomial should be.</p>
SEE ALSO	FFPrimitiveElt ,
EXAMPLE	

NAME	FFeltMove
PURPOSE	Move finite field element into another finite field if possible.
SYNTAX	<pre>b := FFeltMove(a, J);</pre> <pre>finite field element b finite field element a finite field J</pre>
DESCRIPTION	
SEE ALSO	FFPrimitiveElt ,
EXAMPLE	

NAME	FFEltNorm
PURPOSE	Norm of finite field elements.
SYNTAX	<pre>p := FFEltNorm(a); p := FFEltNorm(a, J);</pre> <p> polynomial p finite field element a finite field J subfield of the finite field of a </p>
DESCRIPTION	This function computes the Norm of a finite field element over the base field of the finite field or over the finite field given as second parameter.
SEE ALSO	FFPrimitiveElt ,
EXAMPLE	

NAME FFeltRoot

PURPOSE

SYNTAX **b** := FFeltRoot(a, n);

 finite field element **b**

 finite field element **a**

 small integer **n**

DESCRIPTION

EXAMPLE

NAME	FFeltToInt
PURPOSE	Converts an element of a finite prime field to the corresponding integer.
SYNTAX	<pre>i := FFeltToInt(b); finite field element b integer i</pre>

NAME	FFeltToList
PURPOSE	Returns a basis representation of a finite field element.
SYNTAX	<pre>L := FFeltToList(b); L := FFeltToList(b, k);</pre> <pre>finite field element b subfield k list L</pre>
DESCRIPTION	<p>Let F/k be an extension of finite fields of degree d and a be a generator as returned by <code>FFGenerator()</code>. Then $1, a, \dots, a^{d-1}$ is a k-basis for F. The function returns a list of the coefficients of an element $b \in F$ with respect to the powers of a. If k is omitted in the function call, it is assumed to be the base field.</p>
SEE ALSO	FFGenerator ,
EXAMPLE	

```
kash> F125 := FF(5, 3);
Finite field of size 5^3
kash> a := FFGenerator(F125);
w
kash> FFeltToList(1 + 2*a + 3*a^2);
[ 1, 2, 3 ]
```

NAME	FFEltTrace
PURPOSE	Trace of finite field elements.
SYNTAX	<pre>p := FFEltTrace(a); p := FFEltTrace(a, J);</pre> <p> polynomial p finite field element a finite field J subfield of the finite field of a </p>
DESCRIPTION	This function computes the trace of a finite field element over the base field of the finite field or over the finite field given as second parameter.
SEE ALSO	FFPrimitiveElt ,
EXAMPLE	

NAME	FFEmbed
PURPOSE	Embeds one finite field into another.
SYNTAX	<pre>FFEmbed(F1, F2); FFEmbed(F1, F2, b);</pre> <div> <div>finite field</div> <div>finite field</div> <div>finite field element</div> <div>F1</div> <div>F2</div> <div>b</div> </div>
DESCRIPTION	Embeds one finite field into another, if possible. The optional third parameter specifies the element of F_2 to which the generator of F_1 over its prime field should be mapped to.
SEE ALSO	FF ,
EXAMPLE	

```
kash> FFEmbed(FF(2,3), FF(2,15));
true
```

NAME	FFGenerator
PURPOSE	Returns a generator of a finite field.
SYNTAX	<pre> a := FFGenerator(F); a := FFGenerator(F, k); finite field element a finite field F subfield of F k </pre>
DESCRIPTION	Given a finite field extension F/k of degree d the function returns $a \in F$ such that $F = k \cdot 1 + k \cdot a + \dots + k \cdot a^{d-1}$. If k is omitted in the function call, it is assumed to be the base field.
SEE ALSO	FFPrimitiveElt , FFeltToList ,
EXAMPLE	

```

kash> F25 := FF(5, 2);
Finite field of size 5^2
kash> a := FFGenerator(F25);
w
kash> y := FFPrimitiveElt(F25);
w
kash> F5 := FF(5);
Finite field of size 5
kash> b := FFGenerator(F5);
1
kash> z := FFPrimitiveElt(F5);
2
kash> F64 := FF(2, 6);
Finite field of size 2^6
kash> FFGenerator(F64, F64);
w
kash> FFGenerator(F64, FF(2,2));
w

```

NAME	FFPrimitiveElt
PURPOSE	Returns a primitive element of a finite field.
SYNTAX	<pre>w := FFPrimitiveElt(F);</pre> <p>finite field element w finite field F</p>
DESCRIPTION	For a finite field \mathbb{F}_q ($q = p^d$ a prime power) the function returns a generator ω of \mathbb{F}_q^\times .
SEE ALSO	FFElt , FFGenerator ,
EXAMPLE	

```
kash> F25 := FF(5, 2);
Finite field of size 5^2
kash> w := FFPrimitiveElt(F25);
w
kash> w + w;
w^7
kash> w^0;
1
kash> w^-1;
w^23
```

NAME	FFSize
PURPOSE	Gives the size of the finite field \mathbb{F}_q .
SYNTAX	<pre>L := FFSize(F);</pre> <p>list L of p and d with $p^d = q$. finite field F</p>
DESCRIPTION	This function returns a list consisting of the characteristic p and the degree d of \mathbb{F}_q over \mathbb{F}_p .
EXAMPLE	

```
kash> F9 := FF(3, 2);
Finite field of size 3^2
kash> L := FFSize(F9);
[ 3, 2 ]
```


NAME	FLDin		
PURPOSE	Reads an order in the FLD format.		
SYNTAX	<pre>o := FLDin(name [, n]); o := FLDin(); o := FLDin(f [, n]);</pre>		
	file	f	open for reading
	string	name	filename
	order	o	
	integer	b	the number of fields to skip. If given, the number
			of fields is returned.

If the function is called with a string the order is read from a file with this name. If no argument is given the order is read from the standard input where twice the return key finishes the input.

The last option is a call with an already opened file. When reading multiple orders from the same file it has to be opened with modus "R" (see `Open`).

Now some Information about the structure of the FLD format:

The routine `FLDout` produces an output which can be read again by `FLDin`. Example:

Consider the following output:

```
(FLD=) 6, 2, 2, 0, 6, 2, 0, 18, 0, 0, 0, * -182099043 19.183962221%454045270
1 10 7 89 72 64
3272 0 0 0 0 0 3272
0 3272 0 0 0 0 3272
0 0 3272 0 0 0 3272
0 0 0 3272 0 0 3272
0 1636 0 0 1636 0 3272
720 2117 2543 2554 1237 1 3272
-9 -11 -32 -32 -31 41
-27 -33 -102 -102 -99 131
10 11 35 35 34 -45
18 2 3 6
2
0 2 -4 -3 -3 -1 -3 -1
0 2 -5 -4 -1 -1 -1 0
3 0
0 6
```

First line: (FLD=): command for FLDin

4: field degree N

EXFLAG(1),...,EXFLAG(10): Existence flags for field data: Meaning:

- No. 1: = 0: no integral basis known
- = 1: integral basis is power basis
- = 2: integral basis is not power basis
- = -1: integral basis is not power basis but not known
- = -2: the basis given is only probably an integral basis
- No. 2: > 0: number of independent or fundamental units
- No. 4: > 0: number of roots of unity (if computed)
- No. 5: > 0: The units are fundamental units
- No. 6: > 0: The Galois group structure is known
- No. 7: > 0: Information about the class number.
- No. 10 > 0: Information about the signature

after the asterisk:

first number = field discriminant

second number = regulator of unit system (if computed already)

Second line: coefficients of defining polynomial, decreasing and without leading coefficient (= 1)

Then: Beginning in the third line: If EXFLAG(1)=2 you find the transformation matrix of the integral basis referring to the power basis. Every line corresponds to an integral basis element where the denominator stands in the last position.

After the integral basis matrix: If computed, coefficients of the unit system, corresponding to the integral basis

After the unit system: If already computed and rank of torsion unit subgroup > 2: coefficient of a torsion unit subgroup generator, corresponding to the integral basis

After this: If known: The structure of the Galois group (six characters)

After this: If the class number is known and larger than one in the following four (or more) lines describe the class group structure:

First line: class number, number of cyclic factors, orders

Second line: number of prime ideals needed for description

Then: Prime ideals in two-element-representation, starting with the degree of inertia, the first generator, then the coefficients of the second generator (Each prime ideal in a row)

Then: Exponent vectors for the generators of the cyclic factors, corresponding to the ideals (Each generator in a row)

The above output corresponds to the totally complex sextic number field \mathcal{F} . A generating polynomial of \mathcal{F} is:

$$f(x) = x^6 + x^5 + 10x^4 + 7x^3 + 89x^2 + 72x + 64.$$

An integral basis $\omega_1, \dots, \omega_6$ (referring to a root ρ of f) is given by:

$$\begin{aligned}\omega_1 &= 3272/3272 \\ \omega_2 &= 3272\rho/3272 \\ \omega_3 &= 3272\rho^2/3272 \\ \omega_4 &= 3272\rho^3/3272 \\ \omega_5 &= (1636\rho + 1636\rho^4)/3272 \\ \omega_6 &= (720 + 2117\rho + 2543\rho^2 + 2554\rho^3 + 1237\rho^4 + \rho^5)/3272\end{aligned}$$

The field discriminant is:

$$-182099043.$$

A system of fundamental units is:

$$\begin{aligned}\eta_1 &= -9\omega_1 - 11\omega_2 - 32\omega_3 - 32\omega_4 - 31\omega_5 + 41\omega_6 \\ \eta_2 &= -27\omega_1 - 33\omega_2 - 102\omega_3 - 102\omega_4 - 99\omega_5 + 131\omega_6.\end{aligned}$$

The regulator of \mathcal{F} is 19.18396.

\mathcal{F} contains 6 roots of unity. A primitive 6th root of unity is

$$10\omega_1 + 11\omega_2 + 35\omega_3 + 35\omega_4 + 34\omega_5 - 45\omega_6.$$

The class number of \mathcal{F} is 18. The class group is isomorphic to

$$\mathbb{Z}_3 \times \mathbb{Z}_6.$$

Let

$$\begin{aligned}\wp_1 &= 2o_{\mathcal{F}} + (-4\omega_1 - 3\omega_2 - 3\omega_3 - \omega_4 - 3\omega_5 - \omega_6)o_{\mathcal{F}} \quad (\text{norm}=2) \\ \wp_2 &= 2o_{\mathcal{F}} + (-5\omega_1 - 4\omega_2 - \omega_3 - \omega_4 - \omega_5)o_{\mathcal{F}} \quad (\text{norm}=2)\end{aligned}$$

Then the cyclic factors of the class group are generated by the classes of

$$\begin{aligned}\wp_1^3 &\quad (\text{order: } 3) \\ \wp_2^6 &\quad (\text{order: } 6)\end{aligned}$$

SEE ALSO [FLDout](#), [Open](#),

EXAMPLE We give an example on how to compute tables using KASH: First, we will compute a series of orders using a small program written to check fields generated by polynomials of the form

$$f = x^3 + p * x^2 + p * x + p$$

for non-trivial class groups. Of course this is only an example. In practice you should define a function taking two arguments (**p** and the bound **c** for **p**) and the file to write to. Initialization:

```
kash> \# Open a file for printing the output
kash> outfile := Open("poly.tbl","w");
Filename: poly.tbl / Mode: w / Open (fid): 5
kash> \#
kash> p := 3; \# start
3
kash> c := 3; \# how many fields
3
```

Now do a loop over the first c primes:

```
kash> for j in [1..c] do
> p:=NextPrime(p);
> \#
> \# Assign f and create the corresponding maximal order
> \#
> f := Poly(Zx,[1,p,p,p]);
> O := OrderMaximal(Order(f));
> \#
> \# Compute the class group of o
> \#
> clg := OrderClassGroup(O);
> \#
> \# check if the class group is trivial
> \#
> if IsList(clg) then
> Print ("field generated by ", f," has class number: ", clg[1],"\n");
> \# print the field in the output file
> FLDout(0, outfile);
> fi;
> od;
field generated by  $x^3 + 5x^2 + 5x + 5$  has class number: 1
field generated by  $x^3 + 7x^2 + 7x + 7$  has class number: 3
field generated by  $x^3 + 11x^2 + 11x + 11$  has class number: 6
kash> \# closing the output file
kash> Close(outfile);
true
```

The file should look this:

```
(FLD=) 3, 2, 0, 0, 2, 0, 0, 1, 0, 0, 1, -200, *
5 5 5
8 0 0 8
0 8 0 8
4 0 4 8
(FLD=) 3, 1, 1, 0, 2, 0, 0, 3, 0, 0, 1, -3724, *
7 7 7
1 1 1
3 1 3
1
1 2 1 1 0
1
(FLD=) 3, 1, 1, 0, 2, 0, 0, 6, 0, 0, 1, -28556, *
11 11 11
1 1 1
6 1 6
2
1 5 8 1 0
1 2 1 1 0
1 1
```

Use the output as input to compute fundamental units:

```
kash> \#
kash> \# Open file for input. Use mode "R" (not "r") to loop
kash> \# over the whole file
kash> \#
kash> infile := Open("poly.tbl", "R");
Filename: poly.tbl / Mode: R / Open (fid): 5
kash> \#
kash> \# Read first order
kash> o1 := FLDin(infile);
  F[1]
  |
  F[2]
  /
  /
Q
F [ 1]      Given by transformation matrix
F [ 2]      x^3 + 5*x^2 + 5*x + 5
Discriminant: -200
```

FLDin

Class Number 1

```
kash> OrderUnitsFund(o1);  
[ [0, 1, 2] ]  
kash> \#  
kash> \# Read second order  
kash> o2 := FLDin(infile);  
Generating polynomial:  $x^3 + 7x^2 + 7x + 7$   
Discriminant: -3724
```

Class Number 3

Class Group Structure C3

Cyclic Factors of the Class Group:

<2, 2>

```
kash> OrderUnitsFund(o2);  
[ [1, 1, 1] ]  
kash> \#  
kash> \# Read third order  
kash> o3 := FLDin(infile);  
Generating polynomial:  $x^3 + 11x^2 + 11x + 11$   
Discriminant: -28556
```

Class Number 6

Class Group Structure C6

Cyclic Factors of the Class Group:

<10, 10>

```
kash> OrderUnitsFund(o3);  
[ [1, 1, 1] ]  
kash> Close(infile);  
true
```

NAME	FLDin_DB														
PURPOSE	Reads an order in the FLD format for database "kate" without fundamental units and only number of generators of the class group and their orders														
SYNTAX	<pre>o := FLDin(name [, n]); o := FLDin(); o := FLDin(f [, n]);</pre> <table><tr><td>file</td><td>f</td><td>open for reading</td></tr><tr><td>string</td><td>name</td><td>filename</td></tr><tr><td>order</td><td>o</td><td></td></tr><tr><td>integer</td><td>b</td><td>the number of fields to skip. If given, the number of fields is returned.</td></tr></table>			file	f	open for reading	string	name	filename	order	o		integer	b	the number of fields to skip. If given, the number of fields is returned.
file	f	open for reading													
string	name	filename													
order	o														
integer	b	the number of fields to skip. If given, the number of fields is returned.													
SEE ALSO	FLDout, Open,														

FLDout

NAME	FLDout
PURPOSE	Writes an order using the FLD format.
SYNTAX	<pre>FLDout(o [, name file]);</pre> <pre>Order o String name filename to use File file opened for writing</pre>
DESCRIPTION	<p>Dumps an order using the FLD format. Depending on the (optional) second parameter it is written to the standard output stream (your terminal) or to a file.</p> <p>Be careful when giving a file name: the file will be opened for writing and possibly its contents destroyed.</p> <p>If you want to append your order to an existing file you have to open it using mode="a" to append (see <code>Open</code>).</p> <p>Combined with <code>FLDin</code> this function may be used to produce tables of orders. Using the FLD format you are able to store information like an integral basis, units and the class group, and to retrieve it in a later KASH session.</p> <p>For information about the FLD format see <code>FLDin</code>.</p>
SEE ALSO	<code>Open</code> , <code>Close</code> , <code>LOFILES</code> , <code>FLDin</code> , <code>ECHOon</code> , <code>ECHOoff</code> ,
EXAMPLE	See <code>FLDin</code> for an example.

NAME Factor

PURPOSE Returns the factorization of the given argument.

SYNTAX `F := Factor(d);`
`F := Factor(f);`
`F := Factor(f, p);`
`F := Factor(a);`
`F := Factor(alpha);`
`F := Factor(0, d);`
`F := Factor(0, a);`
`F := Factor(0, alpha);`

list	F
prime	p
integer	d
polynomial	f
ideal	a
algebraic element	alpha
order	0

DESCRIPTION

At the moment 4 different objects can be factored:

Factor(d) Returns the factorization of the integer $d \in \mathbb{N}$. d must be greater 1.

Factor(f) Returns the factorization of the polynomial $f \in \mathbb{Z}[x], \mathbb{Q}[x], \mathbb{O}[x], FF[x]$.

Factor(f, p) Returns the factorization of the polynomial $f \in \mathbb{Z}[x] \bmod p\mathbb{Z}[x]$.

Factor(a) Let \mathcal{O} be the maximal order where \mathfrak{a} has been defined within. The function returns the factorization of \mathfrak{a} in \mathcal{O} .

Factor(alpha) Let \mathcal{O} be the maximal order where α has been defined within. The function returns the factorization of the principal ideal $\alpha\mathcal{O}$ in \mathcal{O} .

Factor(0,d) Returns the factorization of the principal ideal $d\mathcal{O}$ in the maximal order \mathcal{O} .

Factor(0,alpha) Returns the factorization of the principal ideal $\alpha\mathcal{O}$ in the maximal order \mathcal{O} . α must be defined within \mathcal{O} . Same as **Factor(alpha)**.

Factor(0,a) Returns the factorization of the ideal \mathfrak{a} in the maximal order \mathcal{O} . \mathfrak{a} must be defined within \mathcal{O} . Same as **Factor(a)**.

Factor

The factorization of the argument $A = u \prod |i = 1^k \pi| i^{e|i}$ with $A \in \{d, f, \mathfrak{a}, \alpha\}$, a unit u , $k \in \mathbb{N}|0$ and prime elements $\pi|i, 1 \leq i \leq k$ is returned in a list of the following form: $[[\pi|1, e|1], \dots, [\pi|k, e|k]]$.

SEE ALSO [IntFactor](#), [EltFactor](#), [PolyFactor](#), [IdealFactor](#),

EXAMPLE Factorization of 8274626472648264826427648723648276:

```
kash> Factor(8274626472648264826427648723648276);  
[ [ 2, 2 ], [ 11, 1 ], [ 41, 1 ], [ 86423, 1 ],  
  [ 53074086409412759917474753, 1 ] ]
```

NAME Factorial

PURPOSE The factorial of an integer

SYNTAX `a := Factorial(b);`

 integer a
 positive integer b

SEE ALSO [Gamma](#),

EXAMPLE

```
kash> a := Factorial(5);  
120
```

FilePosition

NAME	FilePosition
PURPOSE	Sets or reads the file position, allowing thus random access of FLD files.
SYNTAX	<pre>o := FilePosition(file [, pos]);</pre> <pre>file f open for reading</pre>
SEE ALSO	FLDout , Open ,

NAME	FindMaximalCentralField		
PURPOSE	Finds the maximal factorgroup where a list of automorphisms acts trivial.		
SYNTAX	<pre>sg := FindMaximalCentralField(o I [, inf][, aut]); sg := FindMaximalCentralField(G [, aut]);</pre>		
	Matrix	sg	describing the additional relations
	order	o	
	integral ideal	I	
	list	inf	of infinite places
	AbelianGroup	G	must be from RayClassGroupToAbelianGroup
	list	aut	of automorphisms of o, if omitted OrderAutomorphisms(o, []) is used.

DESCRIPTION

EXAMPLE We'll investigate the ray class field mod $m := (6)\mathfrak{p}|1^\infty\mathfrak{p}|2^\infty$ in $k := \mathbb{Q}(\sqrt{10})$:

```
kash> o := OrderMaximal(Z, 2, 10);
Generating polynomial: x^2 - 10
Discriminant: 40

kash> OrderAutomorphisms(o);
[ [0, 1], [0, -1] ]
kash> G := RayClassGroupToAbelianGroup(6*o, [1, 2]);
RayClassGroupToAbelianGroup(<6>, [ 1, 2 ])
Group with relations:
[2 0 0]
[0 4 0]
[0 0 2]
kash> rcg := FindMaximalCentralField(G);
[2 0 0]
[0 2 0]
[0 0 2]
[0 0 0]
[0 0 0]
[0 0 0]
```

Since `rcg` corresponds not to the whole group, we can construct two fields that are normal over \mathbb{Q} : the ray class field modulo m and the maximal central extension of k :

```
kash> cf1 := RayClassField(6*o, [1, 2], rcg);
[ x^2 - 6, x^2 - 2, x^2 + 2 ]
kash> cf2 := RayClassField(6*o, [1, 2]);
[ x^2 - 6, x^2 + 2, x^4 + [-8, 4]*x^2 + [28, -8] ]
```

Finally we'll compute defining equations and the automorphisms:

```
kash> O1 := RayClassFieldAuto(cf1);
[      F[1]
      /
      /
      E1[1]
      /
      /
      Q
F [ 1]      x^8 - 24*x^6 + 248*x^4 + 288*x^2 + 2704
E 1[ 1]      x^2 - 10
Discriminant: <17352869066524232277088370178392064>
, [ a, a*b, a*b*c, a*b*c*d, a*b*d, a*c, a*c*d, a*d, b, b*c, b*c*d, b*d,
    c, c*d, d, e ] ]
```

...and see that $O1$ is abelian over \mathbb{Q} whereas $O2$ is non-abelian. Since k is cyclic over \mathbb{Q} , central extensions coincide with absolute abelian ones.

```
kash> a := O1[2][Position(O1[2], "a")];
a
kash> b := O1[2][Position(O1[2], "b")];;
kash> c := O1[2][Position(O1[2], "c")];;
kash> d := O1[2][Position(O1[2], "d")];;
kash> a*b = b*a; a*c = c*a; a*d = d*a;
true
true
true
kash> b*c = c*b; b*d = d*b; c*d = d*c;true
true
> true
```

NAME FindQuotientOfShapeEnumInit

PURPOSE Initializes an environment to enumerate certain subgroups

SYNTAX `s := FindQuotientOfShapeEnumInit(G, L);`

record s
AbelianGroup G
list L of integers describing the shape of the subgroup

DESCRIPTION

SEE ALSO [FindQuotientOfShapeEnumNext](#),

EXAMPLE Let $G := C^2|8 \times C|6^2 \times C|2$. We'll enumerate all quotients $Q = G/U$ of G such that $Q \cong C|4 \times C|2$:

```
kash> G := AbelianGroupCreate(MatDiag(Z, [8, 8, 6, 6, 2]));
```

Group with relations:

```
[8 0 0 0 0]
```

```
[0 8 0 0 0]
```

```
[0 0 6 0 0]
```

```
[0 0 0 6 0]
```

```
[0 0 0 0 2]
```

```
kash> s := FindQuotientOfShapeEnumInit(G, [4, 2]);
```

Record of type FindQuotientOfShapeEnum

```
kash> while FindQuotientOfShapeEnumNext(s) and s.no < 3 do
```

```
> Print("Number ", s.no, "\n", s.el, "\n");
```

```
> l := AbelianQuotientGroup(G, AbelianSubGroup(G, s.el));
```

```
> gs := AbelianGroupEnumInit(l);
```

```
> l := [];
```

```
> while AbelianGroupEnumNext(gs) do Add(l, gs.el); od;
```

```
> Apply(l, x->AbelianGroupEltMove(x, G));
```

```
> Print("Containing: ", l, "\n");
```

```
> od;
```

Number 1

```
[2 0 4 0 1]
```

```
[6 0 1 0 0]
```

```
[2 4 4 3 0]
```

```
[0 2 0 4 0]
```

```
[4 0 2 0 0]
```

```

[0 0 0 0 0]
[0 0 0 0 0]
[0 0 0 0 0]
[0 0 0 0 0]
[0 0 0 0 0]
Containing: [ [0 0 0 0 0], [6 7 0 5 0], [1 0 0 0 0], [7 7 0 5 0],
              [2 0 0 0 0], [0 7 0 5 0], [3 0 0 0 0], [1 7 0 5 0] ]
Number 2
[0 0 0 0 1]
[6 0 1 0 0]
[2 4 4 3 0]
[0 2 0 4 0]
[4 0 2 0 0]
[0 0 0 0 0]
[0 0 0 0 0]
[0 0 0 0 0]
[0 0 0 0 0]
[0 0 0 0 0]
Containing: [ [0 0 0 0 0], [6 7 0 5 0], [1 0 0 0 0], [7 7 0 5 0],
              [2 0 0 0 0], [0 7 0 5 0], [3 0 0 0 0], [1 7 0 5 0] ]
kash> while FindQuotientOfShapeEnumNext(s) do
> i := 1;
> od;
kash> Print("Total: ", s.no, "\n");
Total: 360

```


NAME	FindQuotientOfShapeEnumNext		
PURPOSE	Steps through the subgroups.		
SYNTAX	flag := FindQuotientOfShapeEnumNext(s [, 1]);		
	boolean	flag	
	record	s	generated by FindQuoti
	if present find subgroups instead of quotients. 1		
DESCRIPTION			
SEE ALSO	FindQuotientOfShapeEnumInit,		
EXAMPLE	See FindQuotientOfShapeEnumInit for an example.		

Floor

NAME Floor

PURPOSE Returns the maximal integer less than or equal to a given number.

SYNTAX `y := Floor(x);`

 integer y

 real x

DESCRIPTION Given an x in \mathbb{Z} , \mathbb{Q} or \mathbb{R} the function returns

$$\max\{k \in \mathbb{Z} : k \leq x\}.$$

The computation is done in the current precision of the real (complex) field.

SEE ALSO **Trunc**, **Round**, **Ceil**,

EXAMPLE

```
kash> Floor(3);
```

```
3
```

```
kash> Floor(-3);
```

```
-3
```

```
kash> Floor(3.5);
```

```
3
```

```
kash> Floor(-3.5);
```

```
-4
```

NAME	GPin
PURPOSE	Reads an order in the pari/gp format.
SYNTAX	<pre>o := GPin(name); o := GPin(); o := GPin(f); file f file opened for reading string name filename order o **</pre>
DESCRIPTION	<p>If the function is called with a string the order is read from a file with this name. If no argument is given the order is read from the standard input where twice the return key finishes the input.</p> <p>The last option is a call with an already opened file. When reading multiple order from the same file it has to be opened with modus "R" (see <code>Open</code>).</p>

EXAMPLE We read from the standard input:

```
kash> o:=GPin();
false
```

NAME	Galois
PURPOSE	Computation of Galois groups.
SYNTAX	<pre> Galois (f [,p] [, "fast"]); Galois (o [,p] [, "fast"]); Galois (f , "complex" [, n]); Galois (o , "complex" [, n]); GaloisT (f [,p] [, "fast"]); GaloisT (o [,p] [, "fast"]); GaloisT (f , "complex" [, n]); GaloisT (o , "complex" [, n]); polynomial f order o prime number p positive integer n precision </pre>
DESCRIPTION	<p>This function computes the Galois group of an irreducible polynomial with coefficients in \mathbb{Q} and with degree up to 23.</p> <p>The first two callings are a p-adic version of the method of Stauduhar. The optional integer p gives the prime number to be used for the p-adic computations. Not using the parameter "fast" the p-adic version returns a proven result. The optional parameter "fast" is able to speed up the computation considerably, but incorrect results may occur. The option "fast" makes use of lower bounds during the inclusion test of a Galois group computation. The next two callings are the complex version of the method of Stauduhar. The optional integer n gives the precision to be used. The minimal precision is 100.</p> <p><code>Galois()</code> returns the name of the Galois group whereas <code>GaloisT()</code> returns its number in T-notation as classified in GAP [S+97].</p> <p>The implementation of the <code>Galois</code> function [Gei97] in KASH bases on the algorithm of R.P. Stauduhar [Sta73].</p>
SEE ALSO	GaloisGlobals , GaloisGroupsPossible , GaloisModulo , GaloisTree , GaloisRoots , GaloisNumberToName , GaloisBlocks ,
EXAMPLE	

```
kash> Galois(x^12-2);
"D(4)[x]S(3)"
kash> o := Order(Z, 12, 2);
Generating polynomial: x^12 - 2

kash> GaloisModulo(o, 100);
[ 28, 81, 83, 86, 125, 134, 141, 143, 156, 185, 186, 193, 208, 209, 213, 217,
  222, 239, 240, 248, 250, 258, 260, 267, 268, 270, 274, 281, 283, 288, 289,
  292, 293, 294, 299, 301 ]
kash> f := x^15 + 2*x^9 - 5*x^6 + 2*x^3 - 1;
x^15 + 2*x^9 - 5*x^6 + 2*x^3 - 1
kash> GaloisT(f);
61
```

NAME	GaloisBlocks
PURPOSE	Excluding impossible Galois groups by blocks.
SYNTAX	<p>GaloisBlocks(o);</p> <p>GaloisBlocks(o, true);</p> <p>order o</p>
DESCRIPTION	<p>This function computes the lengths of the blocks of the Galois group of the order using subfields. Transitive groups with other block systems are excluded from the list of possible Galois groups. Is the second parameter used, this function computes the Galois groups of the subfields and excludes transitive groups with other Galois groups of the subfields from the list of possible Galois groups.</p>
SEE ALSO	GaloisGroupsPossible , GaloisModulo , Galois ,
EXAMPLE	

```
kash> o := Order(x^15+5*x^14+8*x^13+7*x^12+8*x^11-3*x^9+7*x^8-2*x^7+7*x^6-2*x^5-3*x^4+5*x^3-2*x+1);
```

```
Generating polynomial: x^15 + 5*x^14 + 8*x^13 + 7*x^12 + 8*x^11 - 3*x^9 + 7*x^8 - 2*x^7 + 7*x^6 - 2*x^5 - 3*x^4 + 5*x^3 - 2*x + 1
```

```
kash> GaloisGroupsPossible(o);
```

```
[ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21,
 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40,
 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59,
 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78,
 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97,
 98, 99, 100, 101, 102, 103, 104 ]
```

```
kash> GaloisBlocks(o);
```

```
[ 1, 2, 3, 4, 6, 7, 8, 11, 16, 22, 23, 24, 29 ]
```

```
kash> GaloisBlocks(o, true);
```

```
[ 23 ]
```

NAME	GaloisGlobals
PURPOSE	Prints the current settings in the Galois group computation.
SYNTAX	GaloisGlobals(o); order o
DESCRIPTION	This function prints some settings for the current Galois group computation, for example the used padic or complex fields, the number of transitive permutation groups and the number of computed Tschirnhausen transformations. It is intended for internal use.
SEE ALSO	Galois , GaloisTree , GaloisRoots ,
EXAMPLE	

```

kash> o := Order(Z, 12, 2);
Generating polynomial: x^12 - 2

kash> GaloisT(o, "complex", 150);
28
kash> GaloisGlobals(o);
***** galois globals begin *****
degree is: 12
polynomial is: x^12 - 2
Fields are:
Galois ring of type real over
Complex Field of precision 152

gg_p_bound is: 100
gg_tschirn is: 0
gg_num_groups is: 301
gg_prec is: 150
***** galois globals end *****
kash> o := Order(Z, 12, 2);
Generating polynomial: x^12 - 2

kash> GaloisT(o);
28
kash> GaloisGlobals(o);
***** galois globals begin *****

```

```
degree is: 12
polynomial is: x^12 - 2
Fields are:
Galois ring of type padic over
Generating polynomial: x^2 - 3

for p = 17,
    k = 4,
    m = 48661191875666868481

gg_p_bound is: 100
gg_tschirn is: 0
gg_num_groups is: 301
gg_prec is: 0
***** galois globals end *****
```


NAME	GaloisGroupKnown
PURPOSE	This function returns the Galois group if it is already set in the order or the algebraic function field. If the Galois group is not set <code>false</code> will be returned.
SYNTAX	<pre>b:= GaloisGroupKnown(K); int or false b order or algebraic function field K</pre>
DESCRIPTION	
SEE ALSO	GaloisGlobals , GaloisGroupsPossible , GaloisModulo , GaloisTree , GaloisRoots , GaloisNumberToName , GaloisBlocks ,

EXAMPLE

```
kash> o1 := Order(x^15 - 240*x + 224);
Generating polynomial:  $x^{15} - 240x + 224$ 

kash> o2 := Order(x^15+2*x^9-2*x^6-x^3+2);
Generating polynomial:  $x^{15} + 2x^9 - 2x^6 - x^3 + 2$ 

kash> g := Galois(o1);
"A15"

kash> b := GaloisGroupKnown(o2);
false

kash> b := GaloisGroupKnown(o1);
103
```

NAME	GaloisGroupOrder
PURPOSE	Returns the order of the transitive permutation group.
SYNTAX	<pre>GaloisGroupOrder(n, k);</pre> <pre>integer n representing degree</pre> <pre>integer k representing group in T-notation</pre>
DESCRIPTION	This function returns the order of a transitive permutation group up to degree 20.
SEE ALSO	Galois ,
EXAMPLE	

```
kash> GaloisGroupOrder(18, 354);  
1944
```

NAME GaloisGroupSet

PURPOSE Sets the number of the Galois group in the order or algebraic function field.

SYNTAX GaloisGroupSet(K, G);

 order or algebraic function field K
 int G

DESCRIPTION

SEE ALSO [GaloisGlobals](#), [GaloisGroupsPossible](#), [GaloisModulo](#), [GaloisTree](#),
 [GaloisRoots](#), [GaloisNumberToName](#), [GaloisBlocks](#),

EXAMPLE

```
kash> o := Order(x^13-2);
Generating polynomial: x^13 - 2
```

```
kash> GaloisGroupSet(o,6);
```

NAME	GaloisGroupsPossible
PURPOSE	Handling of possible Galois groups.
SYNTAX	<pre> L := GaloisGroupsPossible(o); GaloisGroupsPossible(o, G, flag); GaloisGroupsPossible(o, L, flag); order o integer G representing transitive group in T-notation list L of integers G boolean flag whether to add or remove groups </pre>
DESCRIPTION	<p>This function is intended for use before calling <code>Galois()</code>. At the beginning of an Galois group computation there is a set of possible Galois groups which will be reduced by computing possible cycle types or block systems for the Galois group. This function either returns the list of current possible Galois groups or is able to remove from or add groups to the list.</p>
SEE ALSO	Galois , GaloisTree , GaloisNumberToName ,
EXAMPLE	

```

kash> o := Order(Z, 8, 2);
Generating polynomial: x^8 - 2

```

```

kash> GaloisGroupsPossible(o);
[ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21,
  22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40,
  41, 42, 43, 44, 45, 46, 47, 48, 49, 50 ]
kash> GaloisGroupsPossible(o, 49, false);
kash> GaloisGroupsPossible(o, [1, 2, 3, 4, 5], false);
kash> GaloisGroupsPossible(o);
[ 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25,
  26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44,
  45, 46, 47, 48, 50 ]

```

NAME	GaloisMSetPol
PURPOSE	Computing of a polynomial of degree $\binom{n}{k}$ which roots are products of k distinct roots of f .
SYNTAX	<pre>g:= GaloisMSetPol(f, k);</pre> <p>int k must be positive polynomial f, g</p>
DESCRIPTION	Let f be a monic polynomial of degree n in \mathbb{Q} , $\mathbb{Q}(x)$ or a simple relative order. This function computes a primitive polynomial of degree $\binom{n}{k}$. The roots of GaloisMSetPol are the products of k distinct roots of f .
SEE ALSO	GaloisGlobals , GaloisGroupsPossible , GaloisModulo , GaloisTree , GaloisRoots , GaloisNumberToName ,
EXAMPLE	

```
kash> Zxy := PolyAlg(Zx);;
kash> y := Poly(Zxy, [1,0]);;
kash> f := y^5-x*(5*y-4);
y^5 - 5*x*y + 4*x
kash> g := GaloisMSetPol(f,3);
y^10 + 20*x^2*y^7 - 125*x^3*y^6 + 128*x^3*y^5 - 400*x^4*y^4 + 1280*x^5*y^2 + 4\
096*x^6
```

NAME	GaloisMSumPol
PURPOSE	Computing of a polynomial of degree $\binom{n}{k}$ which roots are sums of k distinct roots of f .
SYNTAX	<pre>g:= GaloisMSumPol(f, k);</pre> <p>int k must be positive polynomial f,g</p>
DESCRIPTION	Let f be a monic polynomial of degree n in $\mathbb{Q}, \mathbb{Q}(x)$ or a simple relative order. This function computes a primitive polynomial of degree $\binom{n}{k}$. The roots of GaloisMSumPol are the sums of k distinct roots of f .
SEE ALSO	GaloisGlobals , GaloisGroupsPossible , GaloisModulo , GaloisTree , GaloisRoots , GaloisNumberToName , GaloisBlocks ,

EXAMPLE

```
kash> o := OrderMaximal(Z, 2, 2);
```

```
Generating polynomial: x^2 - 2
```

```
Discriminant: 8
```

```
kash> ox := PolyAlg(o);
```

```
Univariate Polynomial Ring in x over Generating polynomial: x^2 - 2
```

```
Discriminant: 8
```

```
kash> f := Poly(ox, [Elt(o,1), Elt(o,[74,-16] / 73), Elt(o,[5988,-2368]
```

```
> / 5329), Elt(o, [160667739, -69612720] / 133432831), Elt(o,
```

```
> [98807254537, -54032137568] / 68184176641]));
```

```
x^4 + [74, -16] / 73*x^3 + [5988, -2368] / 5329*x^2 + [160667739, -69612720] /\
133432831*x + [98807254537, -54032137568] / 68184176641
```

```
kash> g := GaloisMSumPol(f,2);
```

```
x^6 + (222/73*$.1 - 48/73*$.2)*x^5 + (29940/5329*$.1 - 11840/5329*$.2)*x^4 + (\
2594440/389017*$.1 - 1355200/389017*$.2)*x^3 + (42642475310/68184176641*$.1 - \
42196712560/68184176641*$.2)*x^2 + (-16578528081908/4977444894793*$.1 + 988909\
5753408/4977444894793*$.2)*x - 35497382888701141/17804320388674561*$.1 + 22361\
191523984800/17804320388674561*$.2
```

NAME	GaloisMissionS
PURPOSE	Returns all non-trivial subfields of given degree m . If no m is specified, all subfields are calculated.
SYNTAX	<pre>L := GaloisMissionS(o); L := GaloisMissionS(o, m);</pre> <p> <code>list</code> <code>L</code> list of suborders <code>order</code> <code>0</code> the given order <code>small integer</code> <code>m</code> the prescribed degree of subfields </p>
DESCRIPTION	<p>This function calculates all non-trivial subfields of given degree m. If no m is specified, all non-trivial subfields are calculated. The result is a list L, which contains the calculated subfields as orders. The function uses the algorithms described in [Klü95, KP97, Klü97]</p> <p>It is possible that this function returns the same order for two subfields. In this case, these subfields are isomorphic, but not identical. In the example below, the three subfields are isomorphic. If the running time for this function seems too long, the user may want to try OrderSubfieldSub.</p>
SEE ALSO	GaloisMissionS ,

EXAMPLE Computation of subfields of given degree.

```
kash> O:=Order(Poly(Zx,[1,-3,5,-5,5,-3,1]));
Generating polynomial: x^6 - 3*x^5 + 5*x^4 - 5*x^3 + 5*x^2 - 3*x + 1

kash> L:=GaloisMissionS(0,3);
[ Generating polynomial: x^3 - 3*x^2 + 2*x - 1
  , Generating polynomial: x^3 - 2*x^2 + 3*x - 1
  , Generating polynomial: x^3 - 3*x^2 + 2*x - 1
  ]
kash> elt:=Elt(L[1],[1,1,1]);
[1, 1, 1]
kash> EltMove(elt,0);
[2, -3, 3, -5, 3, -2]
```

NAME	GaloisModulo
PURPOSE	Excluding impossible Galois groups by cycle types.
SYNTAX	<pre>GaloisModulo(o, b); order o integer b bound</pre>
DESCRIPTION	This function computes cycle types of the Galois group by factoring the generating polynomial modulo some prime numbers up to a given bound. Transitive groups without the cycle types are excluded from the list of possible Galois groups and the list is returned.
SEE ALSO	Galois , GaloisBlocks , GaloisGroupsPossible , GaloisTree ,
EXAMPLE	

```
kash> o := Order(Z, 8, 2);
```

```
Generating polynomial: x^8 - 2
```

```
kash> GaloisGroupsPossible(o);
```

```
[ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21,
  22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40,
  41, 42, 43, 44, 45, 46, 47, 48, 49, 50 ]
```

```
kash> GaloisModulo(o, 100);
```

```
[ 6, 8, 15, 23, 26, 27, 35, 40, 43, 44, 47, 50 ]
```


NAME	GaloisNumberToName
PURPOSE	Returns name of transitive permutation group.
SYNTAX	<pre>GaloisNumberToName(n, k);</pre> <p> integer n representing degree integer k representing group in T-notation </p>
DESCRIPTION	<p>This function returns the name of the transitive permutation group which is given by an integer according to its T-notation as in GAP [S⁺97]. For the names of the permutation groups see also “On transitive permutation groups” by John H. Conway, Alexander Hulpke and John McKay [CHM98].</p>
SEE ALSO	Galois ,
EXAMPLE	

```
kash> GaloisNumberToName(8, 50);
"S8"
```

NAME	GaloisRing
PURPOSE	missing shortdoc
SYNTAX	$L := \text{GaloisRing}(o);$ <div> <p><code>order</code> <code>o</code></p> <p><code>list</code> <code>L</code> of either the p-adic order, the prime (ideal) p and the exponent k or the complex field and the precesion</p> </div>
DESCRIPTION	Returns the p -adic order, the prime p and the exponent k of the precesion p^k or in the complex case the complex field with the current precision.
SEE ALSO	Galois ,
EXAMPLE	

```

kash> o := Order(Z, 8, 2);
Generating polynomial: x^8 - 2

kash> Galois(o);
"2D_8(8)"
kash> GaloisRing(o);
[ Generating polynomial: x^2 - 5
  , 23, 3 ]

```

NAME	GaloisRoots
PURPOSE	Returns current root ordering.
SYNTAX	<pre>L := GaloisRoots(o); L := GaloisRoots(o,k);</pre> <p> order o int k list L of list of roots and permutation </p>
DESCRIPTION	<p>This function returns a list of the roots of the generating polynomial in the current ordering and a permutation giving this ordering in view of the standard ordering as returned by <code>Solve()</code>. After computation by <code>Galois()</code> the action of the Galois group on this root ordering is equivalent to the computed transitive permutation group as classified in GAP [S⁺97]. Is the second parameter k used, the roots are returned with precision k for the complex case and p^k in p-adic case.</p>
SEE ALSO	Galois ,
EXAMPLE	

```
kash> o := Order(Z, 8, 2);
Generating polynomial: x^8 - 2
```

```
kash> Galois(o);
"2D_8(8)"
kash> GaloisRoots(o);
[ [ -171, [1171, 5618], [0, 384], [1171, -5618], 171, [-1171, -5618],
  [0, -384], [-1171, 5618] ], (2,5)(4,6,7) ]
```

NAME	GaloisSymb								
PURPOSE	Computation of Galois groups								
SYNTAX	$G := \text{GaloisSymb}(o);$ $G := \text{GaloisSymb}(F);$ $G := \text{GaloisSymb}(f);$								
	<table> <tr> <td>string</td><td>G</td></tr> <tr> <td>order</td><td>o</td></tr> <tr> <td>algebraic function field</td><td>F</td></tr> <tr> <td>polynomial</td><td>f</td></tr> </table>	string	G	order	o	algebraic function field	F	polynomial	f
string	G								
order	o								
algebraic function field	F								
polynomial	f								
DESCRIPTION	<p>This function computes the Galois group of an irreducible polynomial with coefficients in \mathbb{Q} or $\mathbb{Q}(x)$ and with degree up to 7. This function uses factorization of absolute resolvents as described in the articles [CM94] resp. [SM85], so the results are unconditional. The name of the Galois group is returned.</p>								
SEE ALSO	Galois ,								
EXAMPLE	<pre> kash> Zxy := PolyAlg(Zx);; kash> y := Poly(Zxy, [1,0]);; kash> f := y^6 - 9/(x^2 + 1)*y^2 + 12/(x^2 + 1)*y - 4/(x^2 + 1); y^6 - 9/(x^2 + 1)*y^2 + 12/(x^2 + 1)*y - 4/(x^2 + 1) kash> GaloisSymb(f); "F_36(6)" </pre>								

NAME	GaloisSymb
PURPOSE	Computation of the Galois group.
SYNTAX	$G := \text{GaloisSymb}(o F f);$ <div> <div>string</div> <div>order</div> <div>algebraic function field</div> <div>polynomial</div> </div> <div> <div>G</div> <div>o</div> <div>F</div> <div>f</div> </div>
DESCRIPTION	<p>This function computes the Galois group of an irreducible polynomial with coefficients in \mathbb{Q} or $\mathbb{Q}(x)$ and simple relative orders with degree up to 7. This function uses factorization of absolute resolvents as described in the articles [CM94] resp. [SM85], so the results are unconditional. The name of the Galois group is returned.</p>
SEE ALSO	Galois ,

EXAMPLE

```
kash> Zxy := PolyAlg(Zx);;
kash> y := Poly(Zxy, [1,0]);;
kash> f := y^6 - 9/(x^2 + 1)*y^2 + 12/(x^2 + 1)*y - 4/(x^2 + 1);
y^6 - 9/(x^2 + 1)*y^2 + 12/(x^2 + 1)*y - 4/(x^2 + 1)
kash> GaloisSymb(f);
"F_36(6)"
```

```
kash> E1 := OrderMaximal(Z, 2, 2);;
kash> E1x:= PolyAlg(E1);;
kash> F1:= Order(Poly(E1x,[Elt(E1,1),0,-4,0, Elt(E1,[2,-1])]));
      F[1]
      /
      /
      E1[1]
      /
      /
      Q
```

```
F  [ 1]      x^4 - 4*x^2 + [2, -1]
E 1[ 1]      x^2 - 2
```

```
kash> GaloisSymb(F1);
"C(4)"
```

NAME	GaloisSymbT								
PURPOSE	Computation of Galois groups								
SYNTAX	$G := \text{GaloisSymbT}(o);$ $G := \text{GaloisSymbT}(F);$ $G := \text{GaloisSymbT}(f);$								
	<table> <tr> <td>int</td><td>G</td></tr> <tr> <td>order</td><td>o</td></tr> <tr> <td>algebraic function field</td><td>F</td></tr> <tr> <td>polynomial</td><td>f</td></tr> </table>	int	G	order	o	algebraic function field	F	polynomial	f
int	G								
order	o								
algebraic function field	F								
polynomial	f								
DESCRIPTION	This function computes the Galois group of an irreducible polynomial with coefficients in \mathbb{Q} or $\mathbb{Q}(x)$ and with degree up to 7. This function uses factorization of absolute resolvents as described in the articles [CM94] resp. [SM85], so the results are unconditional. The number of the Galois group is returned.								
SEE ALSO	Galois ,								

EXAMPLE

```

kash> Qx := PolyAlg(Q);
Univariate Polynomial Ring in x over Rational Field

kash> Qxy := PolyAlg(Qx);
Univariate Polynomial Ring in y over Univariate Polynomial Ring in x over Rational Field

kash> y:= Poly(Qxy, [1,0]);
kash> f := y^7-3*y^6-y^5+3*y^4+(-x + 1)*y^3+(x + 1)*y^2-5*y+4;
y^7 - 3*y^6 - y^5 + 3*y^4 + (-x + 1)*y^3 + (x + 1)*y^2 - 5*y + 4
kash> F := Alff(f);
Algebraic function field defined by
$.1^7 - 3*$.1^6 - $.1^5 + 3*$.1^4 - $.1^3*$.2 + $.1^3 + $.1^2*$.2 + $.1^2 - 5*$.1 + 4
over
Univariate rational function field over Rational Field
Variables: x

kash> GaloisSymbT(F);
5

```

NAME	GaloisSymbT
PURPOSE	Computation of the Galois group.
SYNTAX	$G := \text{GaloisSymbT}(o F f);$ <div> <div>int</div> <div>order</div> <div>algebraic function field</div> <div>polynomial</div> </div> <div> <div>G</div> <div>o</div> <div>F</div> <div>f</div> </div>
DESCRIPTION	<p>This function computes the Galois group of an irreducible polynomial with coefficients in \mathbb{Q} or $\mathbb{Q}(x)$ and simple relative orders with degree up to 7. This function uses factorization of absolute resolvents as described in the articles [CM94] resp. [SM85], so the results are unconditional. The number of the Galois group is returned.</p>
SEE ALSO	Galois ,
EXAMPLE	

```

kash> Qx := PolyAlg(Q);
Univariate Polynomial Ring in x over Rational Field

kash> Qxy := PolyAlg(Qx);
Univariate Polynomial Ring in y over Univariate Polynomial Ring in x over Rati\
onal Field

kash> y:= Poly(Qxy, [1,0]);;
kash> f := y^7-3*y^6-y^5+3*y^4+(-x + 1)*y^3+(x + 1)*y^2-5*y+4;
y^7 - 3*y^6 - y^5 + 3*y^4 + (-x + 1)*y^3 + (x + 1)*y^2 - 5*y + 4
kash> F := Alff(f);
Algebraic function field defined by
$.1^7 - 3*$.1^6 - $.1^5 + 3*$.1^4 - $.1^3*$.2 + $.1^3 + $.1^2*$.2 + $.1^2 - 5*\
$.1 + 4
over
Univariate rational function field over Rational Field
Variables: x

kash> GaloisSymbT(F);
5

```

NAME	GaloisT
PURPOSE	Computation of Galois groups.
SYNTAX	
DESCRIPTION	This function computes the Galois group of an irreducible polynomial with coefficients in \mathbb{Q} and with degree up to 15. The number of the Galois group is returned in T-notation as classified in GAP [S ⁺ 97]. For description see <code>Galois()</code> .
SEE ALSO	<code>Galois</code> , <code>GaloisGlobals</code> , <code>GaloisGroupsPossible</code> , <code>GaloisModulo</code> , <code>GaloisTree</code> , <code>GaloisRoots</code> , <code>GaloisNumberToName</code> , <code>GaloisBlock</code> ,

NAME	GaloisTree
PURPOSE	Output of possible Galois groups as subgroup lattice.
SYNTAX	<pre>L := GaloisTree(o); order o list L list of two lists of integers</pre>
DESCRIPTION	This function prints the subgroup lattice of possible Galois groups as non-connected and directed tree and returns a list with two entries which contain the roots of the tree of odd respectively even transitive groups.
SEE ALSO	Galois , GaloisTreeRoots , GaloisGroupsPossible , GaloisNumberToName ,
EXAMPLE	

```
kash> o := Order(Z, 6, 2);
Generating polynomial: x^6 - 2
```

```
kash> GaloisTree(o);
***** tree of possible groups *****
```

```
rootSn:
16: 13 11 14
13: 9
9: 5 3
5: 2 1
2:
1:
3: 2 1
11: 8 6 3
8: 2
6: 1
14:
```

```
rootAn:
15: 10 7 12
10:
7: 4
4:
```

12:

[[16], [15]]

NAME	GaloisTreeRoots
PURPOSE	Return the roots of the tree of possible Galois groups.
SYNTAX	$L := \text{GaloisTreeRoots}(o);$ <p> <code>order</code> <code>o</code> <code>list</code> <code>L</code> list of two lists of integers </p>
DESCRIPTION	This function returns a list with two entries which contain the roots of the tree of odd respectively even transitive groups. The groups are given by their number in T-notation.
SEE ALSO	GaloisTree , Galois , GaloisGroupsPossible , GaloisNumberToName ,
EXAMPLE	

```
kash> o := Order(Z, 8, 2);
Generating polynomial: x^8 - 2
```

```
kash> GaloisTreeRoots(o);
[ [ 50 ], [ 49 ] ]
kash> GaloisGroupsPossible(o, [49, 50], false);
kash> GaloisTreeRoots(o);
[ [ 47, 44, 43 ], [ 45, 39, 48 ] ]
```

NAME	GaloisTwoSequencePol
PURPOSE	Computing of a polynomial of degree $n \cdot (n - 1)$ which roots are products of two distinct roots of f .
SYNTAX	<pre>g:= GaloisTwoSequencePol(f);</pre> <p>polynomial f, g</p>
DESCRIPTION	Let f be a monic polynomial of degree n in $\mathbb{Q}, \mathbb{Q}(x)$ or a simple relative order. This function computes a primitive polynomial of degree $n(n - 1)$. The roots of <code>GaloisTwoSequencePol</code> are the products of the form $x i + 2x j$, where $x i$ and $x j$ are distinct roots of f .
SEE ALSO	GaloisGlobals , GaloisGroupsPossible , GaloisModulo , GaloisTree , GaloisRoots , GaloisNumberToName , GaloisBlocks ,
EXAMPLE	

```
kash> f := x^6-3*x^5+6*x^4-7*x^3+2*x^2+x-4;
x^6 - 3*x^5 + 6*x^4 - 7*x^3 + 2*x^2 + x - 4
kash> g := GaloisTwoSequencePol(f);
x^30 - 45*x^29 + 1026*x^28 - 15687*x^27 + 179817*x^26 - 1639404*x^25 + 1232376\
0*x^24 - 78227505*x^23 + 426372238*x^22 - 2019887529*x^21 + 8394234972*x^20 - \
30824981910*x^19 + 100623894335*x^18 - 293550986223*x^17 + 769208456704*x^16 -\
1819602767877*x^15 + 3905435727190*x^14 - 7640849227485*x^13 + 13675751675898\
*x^12 - 22435288075854*x^11 + 33744397630017*x^10 - 46517472600975*x^9 + 58815\
885814020*x^8 - 68378015808513*x^7 + 73157522800460*x^6 - 71284825965465*x^5 +\
61112931042866*x^4 - 43100727314520*x^3 + 22477430709080*x^2 - 7271427323616*\
x + 984235529312
```

Gamma

NAME Gamma

PURPOSE Returns the value of the Γ -function.

SYNTAX `y := Gamma(x);`

`real y`

`real x`

EXAMPLE

```
kash> Gamma(3);
```

```
2
```

```
kash> Gamma(4);
```

```
6
```

```
kash> Gamma(5);
```

```
24
```

```
kash> Gamma(5.5);
```

```
52.342777784553520181149008492418193679490132376102
```

```
kash> Gamma(6);
```

```
120
```

NAME	Gcd								
PURPOSE	Returns greatest common divisor of list of arguments.								
SYNTAX	<pre>gcd := Gcd(L); gcd := Gcd(a, b);</pre> <table> <tr> <td>integer or polynomial or ideal</td><td>gcd</td></tr> <tr> <td>list of integers of polynomials or ideals</td><td>L</td></tr> <tr> <td>integer or polynomial or ideal</td><td>a</td></tr> <tr> <td>integer or polynomial or ideal</td><td>b</td></tr> </table>	integer or polynomial or ideal	gcd	list of integers of polynomials or ideals	L	integer or polynomial or ideal	a	integer or polynomial or ideal	b
integer or polynomial or ideal	gcd								
list of integers of polynomials or ideals	L								
integer or polynomial or ideal	a								
integer or polynomial or ideal	b								
DESCRIPTION	Returns the greatest common divisor of list of arguments or simply the greatest common divisor of two arguments. Supported are integers, ideals and polynomials over S where S is a field or \mathbb{Z} .								
SEE ALSO	IntGcd , PolyGcd ,								
EXAMPLE	Gcd of integers:								

```
kash> Gcd([18, 12, 4, 6]);
2
kash> Gcd(18, 12);
6
kash> Gcd([18, 12]);
6
```

GetEnvironment

NAME GetEnvironment

PURPOSE Returns the current value of the environment variable name (\$name).

SYNTAX `st := GetEnvironment(name);`

`string st`
 `string name`

EXAMPLE

```
kash> GetEnvironment(HOSTNAME);  
"kantorowitsch"
```


NAME	HermiteUpperBound
PURPOSE	Returns a upper bound for Hermite's constant γ_n^n (according to Blichfeldt [Bli14]).
SYNTAX	<pre>y := HermiteUpperBound(n); real y integer n</pre>

EXAMPLE

[illegible]

HurwitzZeta

NAME HurwitzZeta

PURPOSE Returns the value of the Hurwitz Zeta-function.

SYNTAX `c := HurwitzZeta(s,v,m0);`

complex s
real v
integer m0
complex c

DESCRIPTION Given $s \in \mathbb{C} \setminus \{1\}$ and $v \in \mathbb{R}$. Then `HurwitzZeta(s,v)` returns the value of the Hurwitz Zeta-function at s .

EXAMPLE

```
kash> c := HurwitzZeta(Comp(2.4,3.1),0.4,6);  
-8.511982 + 2.304635*i  
kash> HurwitzZeta(Comp(2.4,3.1),0.4);  
-8.51198187508935436273191718773675923621648069199 + 2.30463482701114878654193\  
3445673918516011077291233*i
```

NAME	Ideal																								
PURPOSE	Creates an ideal defined by the arguments.																								
SYNTAX	<pre>I := Ideal(e); I := Ideal(o, n); I := Ideal(e, f); I := Ideal(n, e); I := Ideal(o, M, d); I := Ideal(o, MO);</pre> <table><tr><td>Ideal</td><td>I</td><td></td></tr><tr><td>algebraic element</td><td>e</td><td></td></tr><tr><td>algebraic element</td><td>f</td><td></td></tr><tr><td>order over Z</td><td>o</td><td></td></tr><tr><td>matrix</td><td>M</td><td>of integers</td></tr><tr><td>integer</td><td>d</td><td>denominator of M</td></tr><tr><td>integer</td><td>n</td><td></td></tr><tr><td>module</td><td>MO</td><td>over the coefficient order of o</td></tr></table>	Ideal	I		algebraic element	e		algebraic element	f		order over Z	o		matrix	M	of integers	integer	d	denominator of M	integer	n		module	MO	over the coefficient order of o
Ideal	I																								
algebraic element	e																								
algebraic element	f																								
order over Z	o																								
matrix	M	of integers																							
integer	d	denominator of M																							
integer	n																								
module	MO	over the coefficient order of o																							

DESCRIPTION

EXAMPLE All examples will be in the equation order of $x^3 + 6x^2 + 6x + 6$. Here we will create `alpha*o` using different methods:

```

kash> o := Order(Poly(Zx, [1, 6, 6, 6]));;
kash> alpha := Elt(o, [0,1,0]);
[0, 1, 0]
kash> I := Ideal(alpha);
<[0, 1, 0]>
kash> I := alpha*o;
<[0, 1, 0]>

```

NAME	<code>Ideal2EltAssure</code>
PURPOSE	The 2-element representation of the ideal is computed if it is not given yet.
SYNTAX	<code>Ideal2EltAssure(I);</code> <code>ideal I</code>
DESCRIPTION	This procedure is for clarity, <code>IdealGenerators</code> serves for the same purpose but returns the generators. This procedure returns nothing but adds a representation of the ideal.
SEE ALSO	IdealGenerators , IdealBasis ,
EXAMPLE	We consider an ideal in the equation order of $x^3 + 6x^2 + 6x + 6$ which is the result of a multiplication which usually yields an ideal in basis representation. We obtain the 2-element representation

```

kash> O := Order(Poly(Zx, [1, 6, 6, 6]));;
kash> I := Ideal(6,Elt(0,[0,0,1]))*Ideal(3,Elt(0,[0,1,1]));
<
[6 0 0]
[0 6 0]
[0 0 3]
>

kash> Ideal2EltAssure(I);
kash> I;
<6, [24, 12, 33]>

```

NAME	<code>Ideal2EltIntAssure</code>
PURPOSE	The 2-element representation where the first generator is a rational integer of the ideal is computed if it is not given yet.
SYNTAX	<code>Ideal2EltIntAssure(I);</code> <code>ideal I</code>
DESCRIPTION	
SEE ALSO	Ideal2EltAssure , Ideal2EltNormalAssure ,
EXAMPLE	

```

kash> O := Order(Poly(Zx, [1, 6, 6, 6]));;
kash> alpha := Elt(0, [0, 1, 0]);;
kash> beta := Elt(0, [2, 2, 2]);;
kash> I := O*alpha + O*beta;
<[0, 1, 0], [2, 2, 2]>
kash> Ideal2EltIntAssure(I);
kash> I;
<2, [0, 3, 0]>

```

NAME	<code>Ideal2EltKnown</code>
PURPOSE	Returns <code>true</code> if the ideal is given in two element representation.
SYNTAX	<pre>b := Ideal2EltKnown(I);</pre> <p> Boolean b ideal I </p>
DESCRIPTION	<p>The representation is not computed, this is only a check. This function is useful when programming in KASH if an algorithm can handle both basis and 2-element-representation but prefers the 2-element-representation. The computation of the 2-element-representation can be cumbersome in large orders because it involves a trial-and-error algorithm. So it can be handy just to check if the 2-element-representation is already present.</p>
SEE ALSO	IdealGenerators ,
EXAMPLE	

```

kash> O := OrderMaximal(Order(Poly(Zx, [1, 6, 6, 6])));
kash> Lp := Factor(6*O);
[ [ <2, [0, 1, 0]>, 3 ], [ <3, [0, 1, 0]>, 3 ] ]
kash> I := Lp[1][1]*Lp[2][1];
<
[6 0 0]
[0 1 0]
[0 0 1]
>

kash> Ideal2EltKnown(I);
false
kash> IdealGenerators(I);
[ 6, [0, 1, 35] ]
kash> Ideal2EltKnown(I);
true

```

NAME	<code>Ideal2EltNormalAssure</code>
PURPOSE	Calculates a 2 element normal representation of an ideal.
SYNTAX	<code>Ideal2EltNormalAssure(I);</code> <code>ideal I</code>
DESCRIPTION	This is a special representation to speed up multiplication and inverting an ideal. The two generators are computed if not already present and then transformed to a normal representation via a probabilistic algorithm. Multiplying and inverting use normal representations on appropriate occasions.
SEE ALSO	IdealGenerators ,
EXAMPLE	

```

kash> O := OrderMaximal (Order (Poly (Zx, [1,0,0,0,657])));;
kash> I := 5*Elt (O,[345,76,345,46])*O;
<[1725, 380, 1725, 230]>
kash> fac := Filtered (Flat (Factor (I)), IsIdeal);
[ <2, [1, 1, 0, 0]>, <3, [0, 7, 3, 7]>, <5>, <7, [1, 1, 0, 0]>,
  <41, [3, 1, 0, 0]>, <41, [27, 1, 0, 0]>, <197603671, [39148097, 1, 0, 0]> ]
kash> for I in fac do
> Ideal2EltNormalAssure (I);
> Print (I,"\n");
> od;
<2, [1, 1, 0, 0]>
<3, [0, 7, 3, 7]>
<5>
<7, [1, 1, 0, 0]>
<41, [3, 1, 0, 0]>
<41, [27, 1, 0, 0]>
<197603671, [39148097, 1, 0, 0]>

```

NAME	Ideal2EltNormalKnown
PURPOSE	True if the ideal is given with normal two elements representation.
SYNTAX	<pre>b := Ideal2EltNormalKnown(I);</pre> <pre>Boolean b ideal I</pre>
DESCRIPTION	The representation is not computed, this is only a check.
SEE ALSO	IdealGenerators ,
EXAMPLE	See function Ideal2EltNormalAssure

NAME	<code>IdealAutomorphism</code>
PURPOSE	Applies an automorphism to an ideal.
SYNTAX	<pre> B := IdealAutomorphism (A,i); ideal B ideal A integer i </pre>
DESCRIPTION	<p>Let \mathcal{F} be a normal number field with \mathbb{Q}-automorphisms $\sigma_1, \dots, \sigma_n$ and let A be an ideal in \mathcal{F}. The <code>IdealAutomorphism</code> function returns the ideal $\sigma_i(A)$. Before calling <code>IdealAutomorphism</code> the automorphisms $\sigma_1, \dots, \sigma_n$ of \mathcal{F} must be computed by the <code>OrderAutomorphisms</code> routine.</p>
SEE ALSO	EltAutomorphism , OrderAutomorphisms ,

NAME	<code>IdealBasis</code>
PURPOSE	Returns the basis of an ideal.
SYNTAX	<pre>M := IdealBasis(I);</pre> <pre>ideal I list M</pre> <p>two elements, the denominator and the representation matrix</p>
DESCRIPTION	<p>Ideals are given either in Z-basis representation or 2 element representation. If the ideal is given in basis representation this representation is simply returned. Otherwise it will be computed. In this case the ideal has got 2 equivalent representations stored afterwards. This can be of advantage because some algorithms prefer 2 element representations, others prefer Z-basis representations. Once a basis is fixed (entered or calculated) it will never change even if simplification is possible.</p>
SEE ALSO	IdealGenerators ,
EXAMPLE	

```
kash> o := Order(Poly(Zx, [1, 6, 6, 6]));;
kash> alpha := Elt(o, [0, 1, 0]);;
kash> I := Ideal(6, alpha);
<6, [0, 1, 0]>
kash> IdealBasis(I);
[ 1, [6 0 0]
  [0 1 0]
  [0 0 1] ]
```

NAME IdealBasisKnown

PURPOSE Returns `true` if the ideal is given via a \mathbb{Z} basis.

SYNTAX `b := IdealBasisKnown(I);`

 Boolean b
 ideal I

DESCRIPTION The basis representation is not computed, this is only a check.

SEE ALSO [IdealBasis](#),

EXAMPLE

```
kash> o := Order(Poly(Zx, [1, 6, 6, 6]));;
kash> alpha := Elt(o, [0, 1, 0]);;
kash> I := Ideal(6, alpha);
<6, [0, 1, 0]>
kash> IdealBasisKnown(I);
false
kash> IdealBasis(I);
[ 1, [6 0 0]
  [0 1 0]
  [0 0 1] ]
kash> IdealBasisKnown(I);
true
```

NAME	IdealBasisLowerHNF
PURPOSE	The basis of the ideal transformed in lower HNF
SYNTAX	<p><code>M:=IdealBasisLowerHNF(I);</code></p> <p><code>list</code> <code>M</code> two elements, the denominator and the basis matrix</p> <p><code>ideal</code> <code>I</code></p>
DESCRIPTION	This returns the basis of the ideal transformed in lower HNF. The basis is computed from the actual (which can be seen with <code>IdealBasis</code>). Once computed the basis in lower HNF is stored in the ideal data structure so it has not to be computed again.
SEE ALSO	IdealBasisUpperHNF , IdealLowerHNFTrans ,

EXAMPLE

```

kash> O:=OrderMaximal(Order(Poly(Zx,[1,4,1,-4,-3,7])));
Generating polynomial: x^5 + 4*x^4 + x^3 - 4*x^2 - 3*x + 7
Discriminant: 28442269

kash> I:=Ideal(0,Mat(Z,[[1,2,1,4,6],[3,8,4,6,2],[6,4,3,3,8],
> [3,2,7,9,5],[0,2,8,8,4]]),1);
<
[1 2 1 4 6]
[3 8 4 6 2]
[6 4 3 3 8]
[3 2 7 9 5]
[0 2 8 8 4]
>

kash> IdealBasisLowerHNF(I);
[ 1, [ 1 0 0 0 0]
      [ 0 1 0 0 0]
      [ 0 0 1 0 0]
      [ 0 1 0 3 0]
      [ 512 910 858 1672 1678] ]

```

NAME	<code>IdealBasisUpperHNF</code>
PURPOSE	The basis of the ideal transformed in upper HNF.
SYNTAX	<pre>M := IdealBasisUpperHNF(I);</pre> <p>list M two elements, the denominator and the basis matrix ideal I</p>
DESCRIPTION	This returns the basis of the ideal transformed in (right) upper HNF. Either it is computed from a plain basis (which can be seen with <code>IdealBasis</code>) or from a 2-element representation. Once computed the basis in upper HNF is stored in the ideal data structure so it has not to be computed again. In most cases <code>IdealBasis</code> and <code>IdealBasisUpperHNF</code> are identical.
SEE ALSO	IdealBasisLowerHNF , IdealUpperHNFTTrans ,
EXAMPLE	

```
kash> o := Order(Poly(Zx, [1, 6, 6, 6]));;
kash> alpha := Elt(o, [0, 1, 0]);;
kash> I := Ideal(6, alpha);
<6, [0, 1, 0]>
kash> IdealBasisUpperHNF(I);
[ 1, [6 0 0]
      [0 1 0]
      [0 0 1] ]
```

NAME	IdealChineseRemainder
PURPOSE	Chinese Remainder Theorem for number fields
SYNTAX	$\text{beta} := \text{IdealChineseRemainder}(\text{a1}, \text{a2}, \text{alpha1}, \text{alpha2})$ <div> ideals a1, a2 algebraic numbers alpha1, alpha2 algebraic number beta </div>
DESCRIPTION	Computes an algebraic number β in the order \mathfrak{o} , such that $\alpha i - \beta$ is contained in the ideal $\mathfrak{a} i$, $i = 1, 2$.
SEE ALSO	EltApproximation , RayCantoneseRemainder ,
EXAMPLE	

```

kash> O := OrderMaximal (Order (Poly(Zx,[1,0,73,-280,-2399])));
kash> a1 := Factor (2*O)[1][1];
<2, [1, 2, 0, 1]>
kash> a2 := Factor (NextPrime (4343343)*O)[1][1];
<4343357, [1353610, 1860090, 2, 0]>
kash> beta := IdealChineseRemainder (a1, a2, Elt (O,[1,2,3,5]), Elt (O, 4));
[4, 4343357, 0, 0]
kash> Valuation (a1, beta - Elt (O,[1,2,3,5]));
2
kash> Valuation (a2, beta - Elt (O,4));
1

```

NAME	<code>IdealClassRep</code>
PURPOSE	Computes a representation of the ideal class over the cyclic generators of the class group.
SYNTAX	<pre> L := IdealClassRep(I); L := IdealClassRep(I, "gen"); list L ideal I </pre>
DESCRIPTION	<p>If $\mathfrak{a} 1, \dots, \mathfrak{a} k$ are the cyclic generators of the class group as returned by <code>OrderClassGroupCyclicFactors</code>, for an ideal \mathfrak{b} this function computes a representation of the form</p> $\mathfrak{b} = \alpha \prod_i i = 1^k \mathfrak{a} ^{s i}$ <p>with α an algebraic element. The result is stored in a list of the form $\{\alpha, \{\mathfrak{a} 1, s 1\}, \dots, \{\mathfrak{a} k, s k\}\}$, where α is omitted when the second parameter "gen" is not given.</p>
SEE ALSO	OrderClassGroup , IdealIsPrincipal , IdealRayClassRep ,

EXAMPLE

```

kash> o := OrderMaximal(Order(x^3-117));;
kash> OrderClassGroup(o, euler);
[ 3, [ 3 ] ]
kash> OrderClassGroupCyclicFactors(o);
[ [ <2, [1, 1, 3]>, 3 ] ]
kash> p := Factor(1021*o)[1][1];
<1021, [499, 1, 0]>
kash> L := IdealClassRep(p, "gen");
[ [287, 59, 36] / 2, [ <2, [1, 1, 3]>, 1 ] ]
kash> alpha := L[1];
[287, 59, 36] / 2
kash> a1 := L[2][1];
<2, [1, 1, 3]>
kash> s1 := L[2][2];
1
kash> alpha*a1^s1/p;
<
[1 0 0]
[0 1 0]

```

[0 0 1]

>

NAME	<code>IdealCollection</code>
PURPOSE	Solves a special equation.
SYNTAX	$M := \text{IdealCollection}(I1, I2);$ <p> <code>list</code> <code>M</code> two lists of two elements of the order <code>ideals</code> <code>I1, I2</code> integral, over a maximal order </p>
DESCRIPTION	<p>Let $\alpha 1, \alpha 2$ be two algebraic numbers and $\mathfrak{I} 1, \mathfrak{I} 2$ two ideals over the same order \mathcal{O}. The result of this function is a matrix M which satisfies</p> $\alpha 1\mathfrak{I} 1 + \alpha 2\mathfrak{I} 2 = \beta 1\mathcal{O} + \beta 2\mathfrak{I} 1\mathfrak{I} 2$ <p>where</p> $\begin{pmatrix} \beta 1 \\ \beta 2 \end{pmatrix} = \begin{pmatrix} \alpha 1 \\ \alpha 2 \end{pmatrix} M.$
SEE ALSO	ModuleSteinitz ,
EXAMPLE	<pre> kash> O := OrderMaximal(Poly(Zx, [1, 6, 6, 6]));; kash> I1:=Ideal(6,Elt(0,[0,0,1])); <6, [0, 0, 1]> kash> I2:=Ideal(3,Elt(0,[0,1,1])); <3, [0, 1, 1]> kash> M := IdealCollection(I1,I2); [[[0, 0, 1], 0], [-1, [0, 5, 1] / 6]] kash> M[1][1]*M[2][2]-M[1][2]*M[2][1]; 1 kash> a1 := Elt(0,[2,1,0]); [2, 1, 0] kash> a2 := Elt(0,[0,1,-1]); [0, 1, -1] kash> b1 := a1 * M[1][1] + a2 * M[1][2];; kash> b2 := a1 * M[2][1] + a2 * M[2][2];; kash> a1*I1+a2*I2 = b1*O+b2*I1*I2; true </pre>

NAME	<code>IdealDegree</code>
PURPOSE	Calculates the degree of inertia of a prime ideal.
SYNTAX	<pre>d := IdealDegree(I);</pre> <p>integer d ideal I must be prime</p>
DESCRIPTION	<p>Let \mathcal{O} be a maximal order. Let p be a prime number and \mathfrak{p} a prime \mathcal{O}-ideal over this prime number p. The index</p> $[\mathcal{O}/\mathfrak{p} : \mathbb{Z}[p]]$ <p>is called the degree of inertia of \mathfrak{p} over p.</p>
SEE ALSO	IdealRamIndex ,
EXAMPLE	

```
kash> O := OrderMaximal(Order(Poly(Zx,[1,4,1,-4,-3,7])));
Generating polynomial: x^5 + 4*x^4 + x^3 - 4*x^2 - 3*x + 7
Discriminant: 28442269
```

```
kash> I := 6*O;
<6>
kash> p := Factor(I)[2][1];
<2, [1, 0, 0, 1, 1]>
kash> IdealDegree(p);
4
```

NAME	<code>IdealDen</code>
PURPOSE	The denominator of the ideal.
SYNTAX	<pre>d := IdealDen(I);</pre> <pre>ideal I integer d</pre>
DESCRIPTION	<p>The denominator of a fractional ideal \mathfrak{a} is the smallest positive integer d that $d\mathfrak{a}$ is an integral ideal. Integral ideals are just fractional ideals with denominator 1 and have no separate internal data structure.</p> <p>Once computed the denominator is stored in the ideal data structure so it has not to be computed again.</p>
SEE ALSO	<code>IdealIsIntegral</code> ,
EXAMPLE	

```
kash> O := OrderMaximal(Poly(Zx,[1,4,1,-4,-3,7]));
Generating polynomial: x^5 + 4*x^4 + x^3 - 4*x^2 - 3*x + 7
Discriminant: 28442269
```

```
kash> a := Elt(0,[1,5,1,8,0]);
[1, 5, 1, 8, 0]
kash> b := Elt(0,[3, 1,5,1,8]);
[3, 1, 5, 1, 8]
kash> I := Ideal(a, b);
<[1, 5, 1, 8, 0], [3, 1, 5, 1, 8]>
kash> IdealDen(I^-1);
3
```

NAME	<code>IdealDivisors</code>
PURPOSE	Computes all divisors of an ideal.
SYNTAX	<pre>L := IdealDivisors (A); list L ideal A</pre>
DESCRIPTION	The <code>IdealDivisors</code> function computes a list containing all divisors of an integral ideal. The underlying order of A must be known to be maximal.
EXAMPLE	

```
kash> o := OrderMaximal(Z,2,-5);
Generating polynomial: x^2 + 5
Discriminant: -20

kash> A := 6*o;
<6>
kash> IdealDivisors(A);
[ <1>, <2, [1, 1]>, <4, [0, 2]>, <3, [1, 1]>, <
  [6 1]
  [0 1]
>
, <6, [2, 2]>, <3, [2, 1]>, <
  [6 5]
  [0 1]
>
, <6, [4, 2]>, <3>, <
  [6 3]
  [0 3]
>
, <6> ]
```

NAME	<code>IdealFactor</code>
PURPOSE	Returns the factorization of the given ideal over a maximal order.
SYNTAX	<pre>F := IdealFactor(a);</pre> <pre>list F ideal a</pre>
DESCRIPTION	<p>Because a maximal order \mathcal{O} is a Dedekind ring every integral \mathcal{O}-ideal has a unique factorization in prime ideals. More generally, it is possible to factor ideals in prime ideals over arbitrary orders above the equation order at least in case they do not divide the discriminant of the equation order.</p> <p>The factorization can simply be extended to fractional ideals: if d is the denominator of \mathfrak{a} and \mathfrak{b}, the integral ideal $d\mathfrak{a}$, the ideal \mathfrak{b} and the ideal $d\mathcal{O}$ is factorized. The exponents of the factorization of $d\mathcal{O}$ are negated and multiplied to the factorization of \mathfrak{b} simplifying exponents of identical prime ideals.</p>
SEE ALSO	Factor ,

EXAMPLE Factorization of the principal ideal (3) in the maximal order of $\mathbb{Q}(\sqrt{-110})$:

```
kash> o := Order(Z,2,-110);
Generating polynomial: x^2 + 110

kash> O := OrderMaximal(o);
Generating polynomial: x^2 + 110
Discriminant: -440

kash> IdealFactor(3*O);
[ [ <3, [1, 1]>, 1 ], [ <3, [2, 1]>, 1 ] ]
kash> a:=Ideal(0,3);
<3>
kash> IdealFactor(a);
[ [ <3, [1, 1]>, 1 ], [ <3, [2, 1]>, 1 ] ]
```

NAME	<code>IdealGen</code>
PURPOSE	missing shortdoc
SYNTAX	<pre>g := IdealGen(I, i);</pre> <p> algebraic element <code>g</code> ideal <code>I</code> small integer <code>i</code> $\in \{1, 2\}$ </p>
DESCRIPTION	<p>Any ideal I of a maximal order \mathfrak{o} may be generated using two elements g_1 and g_2 (i.e. $I = g_1\mathfrak{o} + g_2\mathfrak{o}$). This functions returns g_i ($i \in \{1, 2\}$) after computing suitable generators if necessary.</p> <p>The restriction to maximal orders is only necessary to compute the generators. Note that the generators are not unique! In fact since the algorithm is probabilistic, they may change between two sessions.</p>
SEE ALSO	IdealGenerators , IdealBasis ,
EXAMPLE	<pre>kash> O := Order(Poly(Zx, [1, 6, 6, 6]));; kash> alpha := Elt(O, [0, 1, 0]);; kash> I := Ideal(6, alpha); <6, [0, 1, 0]> kash> IdealGen(I, 1); 6 kash> IdealGen(I, 2); [0, 1, 0]</pre>

NAME	<code>IdealGenerators</code>
PURPOSE	Returns the 2 generators of the 2-element-representation of the ideal.
SYNTAX	<pre>L := IdealGenerators(I);</pre> <p> <code>ideal</code> <code>I</code> <code>list</code> <code>L</code> of two algebraic numbers </p>
DESCRIPTION	<p>Ideals are given either in Z-basis representation or 2 element representation. If the ideal is given in 2 element representation this representation is simply returned. Otherwise it will be computed. In this case the ideal has got 2 equivalent representations stored afterwards. This can be of advantage because some algorithms prefer 2 element representations, others prefer Z-basis representations.</p>
SEE ALSO	IdealBasis ,
EXAMPLE	<pre>kash> O := OrderMaximal(Order(Poly(Zx, [1, 6, 6, 6])));; kash> Lp:=Factor(6*O); [[<2, [0, 1, 0]>, 3], [<3, [0, 1, 0]>, 3]] kash> I := Lp[1][1]*Lp[2][1]; < [6 0 0] [0 1 0] [0 0 1] > kash> IdealGenerators(I); [6, [0, 1, 35]]</pre>

NAME	<code>IdealIdempotents</code>
PURPOSE	Returns elements of the comaximal ideals which sum to 1.
SYNTAX	$E := \text{IdealIdempotents}(L)$ <p> <code>list</code> <code>L</code> of integral ideals over the same order <code>false or list</code> <code>E</code> of algebraic elements over the same order </p>
DESCRIPTION	<p>Let L be the list of ideals $[I 1, \dots, I n]$.</p> <p>This function tries to find algebraic elements $a i$ in $I i$ such that: $\sum a i = 1$. The function will return false if this is not possible (iff $I 1, \dots, I n$ are NOT comaximal).</p> <p>Otherwise it returns the list $[a 1, \dots, a n]$ of elements which sum to one. Each entry of E is an element of the ideal in L with the same index.</p> <p>Fractional ideals are allowed and considered as the biggest integral ideal which contains the fractional ideal, i.e. disregarding the denominator.</p>

EXAMPLE

```

kash> O := OrderMaximal(Order(Z,2,- 45));;
kash> L := List( Factor(210*O), x->x[1]);
[ <2, [1, 3]>, <3, [1, 2]>, <3, [7, 7]>, <5, [0, 3]>, <7, [2, 3]>,
  <7, [5, 3]> ]
kash> E := IdealIdempotents([L[2]*L[6]*L[3],L[2]*L[4],L[1]*L[4]*L[6]]);
[ 966, -1035, 70 ]

```


NAME	<code>IdealImprove</code>
PURPOSE	Improves the generators of the ideal given in two element representation.
SYNTAX	$I1 := \text{IdealImprove}(I2);$ $\text{ideal } I1, I2$
DESCRIPTION	<p>The ideal can be improved if</p> <ul style="list-style-type: none"> • one of the generators is an rational integer. Then every basis coefficient of the other generator can be reduced modulo this integer. • the minimum is given. Coefficients of both generators can be reduced modulo this integer. This procedure is fast and simple therefore it does not compute the minimum of the ideal. For more possible improvement call <code>IdealMin</code> first.
SEE ALSO	IdealMin ,

EXAMPLE

```

kash> O:=Order(Poly(Zx,[1,0,2,4,58]));
Generating polynomial: x^4 + 2*x^2 + 4*x + 58

kash> E:=Elt(0,[1,2,-1,-2]);
[1, 2, -1, -2]
kash> I:=Ideal(3,E^7);
<3, [-470899145529, -57698441418, 37053141409, 29584796218]>
kash> IdealImprove(I);
<3, [-470899145529, -57698441418, 37053141409, 29584796218]>

```

NAME	IdealIntegrity
PURPOSE	Checks the ideals for various inconsistencies
SYNTAX	<pre>errcount = IdealIntegrity(I);</pre> <p> <code>ideal</code> <code>I</code> <code>small integer</code> <code>errcount</code> the number of detected errors </p>
DESCRIPTION	<p>The most important application of this function is this: Not all matrices (square and with the correct dimension) over the coefficient order of the ideal form an valid ideal. If an ideal is created with a basis matrix it is not checked if this matrix forms a correct ideal basis (because it is much too expensive to check this for every ideal creation by default.) Such an <i>wrong</i> ideal can cause all sorts of follow-up errors which might be difficult to detect. If You are not sure if the ideal is correct apply this function.</p> <p>The ideal invariants which are stored in the ideal data structure are checked for their validity as well, including factorization, norm, minimum, principal generator, prime element, primality.</p> <p>If an inconsistency is detected a warning message is printed to the screen but the break loop is not entered (except if it is not possible to find a 2 element representation where the break loop is entered). If the return value is 0 the ideal is (very likely) correct.</p>

EXAMPLE Correct ideal

```
kash> O := OrderMaximal(Order(Poly(Zx, [1, 6, 6, 6])));
kash> I := Ideal(O, Mat(Z, [[2,0,0],[0,1,0],[0,0,1]],1);
<
[2 0 0]
[0 1 0]
[0 0 1]
>

kash> IdealIntegrity(I);
0
```

NAME	<code>IdealIsIntegral</code>
PURPOSE	Checks if a (fractional) ideal is an integral (or true) ideal.
SYNTAX	<pre><code>b := IdealIsIntegral(I);</code></pre> <p> <code>boolean</code> <code>b</code> <code>Ideal</code> <code>I</code> </p>
DESCRIPTION	A fractional ideal \mathfrak{a} of \mathcal{O} is an \mathcal{O} -module $\mathfrak{a} \subset Q(\mathcal{O})$. An integral ideal is an \mathcal{O} -ideal which is an \mathcal{O} -module $\mathfrak{a} \subset \mathcal{O}$.
SEE ALSO	IdealDen ,
EXAMPLE	

```
kash> o := Order (Poly (Zx,[1,6,6,6]));
Generating polynomial: x^3 + 6*x^2 + 6*x + 6
```

```
kash> I := Elt (o,[1,0,1])*o + 5*o;
<5, [1, 0, 1]>
kash> IdealIsIntegral (I);
true
kash> I := Elt (o,[1,0,1]/5)*o + 5*o;
<5, [1, 0, 1] / 5>
kash> IdealIsIntegral (I);
false
```

NAME	<code>IdealIsPrime</code>
PURPOSE	Checks whether an ideal is a prime ideal. Returns <code>true</code> or <code>false</code> .
SYNTAX	<pre>b := IdealIsPrime(I);</pre> <p> boolean b ideal I integral </p>
DESCRIPTION	The order over which the prime ideal is defined must be maximal.
SEE ALSO	orderidealisprimeideal ,

EXAMPLE

```
kash> o := OrderMaximal (Order (Poly (Zx,[1,6,6,6])));
Generating polynomial: x^3 + 6*x^2 + 6*x + 6
Discriminant: -1836

kash> IdealIsPrime (5*o);
false
kash> Factor (5*o);
[ [ <5, [1, 1, 0]>, 1 ], [ <5, [2, 1, 0]>, 1 ], [ <5, [3, 1, 0]>, 1 ] ]
kash> IdealIsPrime (11*o);
true
kash> Factor (11*o);
[ [ <11>, 1 ] ]
```

NAME	<code>IdealIsPrincipal</code>
PURPOSE	Returns a principal generator of an ideal, if it is a principal ideal and false otherwise.
SYNTAX	<pre> g := IdealIsPrincipal(I); g := IdealIsPrincipal(I, classgroup); ideal I false or algebraic number g principal generator </pre>
DESCRIPTION	<p>There are two different algorithms used by this function. The standard procedure is not an easy check and can include expensive enumerations. If a second argument "classgroup" is given, it uses precomputed data from class group calculations. In this case it is often quite fast. Once computed, the result is stored and the ideal arithmetic uses the principal generator. A warning: The standard procedure for number fields uses real arithmetic and the correctness of the results depends on the precision defined by <code>OrderPrec()</code> whereas the second method works unconditional.</p>

EXAMPLE We will test the prime ideal lying over 2 in the maximal order of $x^3 + 6x^2 + 6x + 6$.

```

kash> O := OrderMaximal(Order(Poly(Zx, [1, 6, 6, 6])));
Generating polynomial: x^3 + 6*x^2 + 6*x + 6
Discriminant: -1836

```

```

kash> p := Factor(O, 2*O)[1][1];
<2, [0, 1, 0]>
kash> IdealIsPrincipal(p);
false
kash> IdealIsPrincipal(p^3);
2

```

NAME	<code>IdealLLL</code>
PURPOSE	Creates an ideal with LLL-reduced real basis
SYNTAX	<code>I := IdealLLL(a);</code> <code>ideal I, a</code>
DESCRIPTION	The function calculates a real basis for a given ideal \mathfrak{a} and performs a LLL reduction on it.
SEE ALSO	IdealBasisIdealGenerators ,
EXAMPLE	

```
kash> o:=OrderMaximal(Z,2,2);
Generating polynomial: x^2 - 2
Discriminant: 8
```

```
kash> id:=Factor(7*o)[1][1]^44;
<15286700631942576193765185769276826401, [11817265640666745415214483233, 83560\
68669598246694295007788]>
kash> IdealLLL(id);
<
[ 1985257763218632743  4216024203981766227]
[ 3100640983600199485 -1115383220381566742]
>
```

NAME	IdealLcm
PURPOSE	Least common multiplier of 2 ideals.
SYNTAX	$I := \text{IdealLcm}(I1, I2);$ Ideals $I, I1, I2$
DESCRIPTION	The lcm of \mathfrak{a} and \mathfrak{b} is computed as $\frac{\mathfrak{a}\mathfrak{b}}{\mathfrak{a}+\mathfrak{b}}$. The order over which the ideals are defined must be maximal.
EXAMPLE	<pre> kash> O:=OrderMaximal(Order(Poly(Zx,[1,6,6,6]))); Generating polynomial: x^3 + 6*x^2 + 6*x + 6 Discriminant: -1836 kash> I1:=Ideal(18,Elt(0,[0,0,1])); <18, [0, 0, 1]> kash> I2:=Ideal(24,Elt(0,[0,0,1])); <24, [0, 0, 1]> kash> IdealLcm(I1,I2); < [6 0 0] [0 6 0] [0 0 1] > </pre>

NAME	IdealLowerHNFTrans
PURPOSE	the transformation matrix from the original basis to the basis in lower HNF
SYNTAX	<pre>M:=IdealLowerHNFTrans(I);</pre> <p>matrix M</p> <p>ideal I</p>
DESCRIPTION	This returns the transformation matrix when transforming the original basis (which can be seen with IdealBasis) to the basis in lower HNF (which can be seen with IdealBasisLowerHNF). Once computed the transformation matrix is stored in the ideal data structure so it has not to be computed again.
SEE ALSO	IdealBasisLowerHNF ,
EXAMPLE	

```
kash> O:=OrderMaximal(Order(Poly(Zx,[1,4,1,-4,-3,7])));
Generating polynomial: x^5 + 4*x^4 + x^3 - 4*x^2 - 3*x + 7
Discriminant: 28442269
```

```
kash> I:=Ideal(0,Mat(Z,[[1,2,1,4,6],[3,8,4,6,2],[6,4,3,3,8],
> [3,2,7,9,5],[0,2,8,8,4]]),1);
<
[1 2 1 4 6]
[3 8 4 6 2]
[6 4 3 3 8]
[3 2 7 9 5]
[0 2 8 8 4]
>
```

```
kash> IdealBasis(I)[2]*IdealLowerHNFTrans(I);
[ 1 0 0 0 0]
[ 0 1 0 0 0]
[ 0 0 1 0 0]
[ 0 1 0 3 0]
[ 512 910 858 1672 1678]
kash> IdealBasisLowerHNF(I);
[ 1, [ 1 0 0 0 0]
      [ 0 1 0 0 0]
      [ 0 0 1 0 0]
      [ 0 1 0 3 0]
      [ 512 910 858 1672 1678] ]
```


NAME	IdealMakeCoprime
PURPOSE	missing shortdoc
SYNTAX	$c := \text{IdealMakeCoprime}(A,B);$ <div style="display: flex; justify-content: space-between;"> ideal A,B </div> <div style="display: flex; justify-content: space-between;"> algebraic number c </div>
DESCRIPTION	Let \mathfrak{a} and \mathfrak{b} integral ideals. The procedure finds c so that $c\mathfrak{a}$ is an integral ideal coprime to \mathfrak{b} . This algorithm is due to a paper by H. Cohen [Coh96].
SEE ALSO	IdealMakeInvCoprime , IdealClassRep , OrderClassGroup ,
EXAMPLE	

```
kash> O := OrderMaximal(x^2-10);
```

```
Generating polynomial: x^2 - 10
```

```
Discriminant: 40
```

```
kash> OrderClassGroup(O,500,"euler","fast");
```

```
[ 2, [ 2 ] ]
```

```
kash> A := OrderClassGroupCyclicFactors(O)[1][1];
```

```
<2, [0, 1]>
```

```
kash> B := 2*O;
```

```
<2>
```

```
kash> c := IdealMakeCoprime(A,B);
```

```
[0, 1] / 2
```

```
kash> D := c*A;
```

```
<5, [0, 1]>
```

```
kash> IdealClassRep(D);
```

```
[ [ <2, [0, 1]>, 1 ] ]
```

NAME	IdealMakeInvCoprime
PURPOSE	Subroutine of OrderRelNfColl
SYNTAX	<pre>a := IdealMakeInvCoprime(I1,I2);</pre> <p>ideals A, B must be integral algebraic number a</p>
DESCRIPTION	Let $\mathfrak{a}, \mathfrak{b}$ be integral ideals. The procedure finds an algebraic number c so that $c * \mathfrak{a}^{-1}$ is an integral ideal coprime to \mathfrak{b} .
SEE ALSO	IdealMakeCoprime ,
EXAMPLE	

```
kash> O := OrderMaximal(Order(Poly(Zx,[1,-2,57,-56,602])));
```

```

F[1]
|
F[2]
/
/
Q
F [ 1]      Given by transformation matrix
F [ 2]      x^4 - 2*x^3 + 57*x^2 - 56*x + 602
Discriminant: 529984
```

```

kash> LP := List(Factor(210*O), x->x[1]);
[ <2, [0, 1, 0, 0]>, <2, [1, 1, 0, 0]>, <3, [1, 0, 1, 0]>, <3, [2, 1, 1, 0]>,
  <5, [3, 0, 1, 0]>, <5, [4, 3, 1, 0]>, <7, [0, 1, 0, 0]>, <7, [6, 1, 0, 0]> ]
```

```
kash> A := LP[1]^2*LP[3]*LP[6];
```

```

<
[30  0  4 17]
[ 0 30 18  2]
[ 0  0  1  0]
[ 0  0  0  1]
>
```

```
kash> B := LP[1]^2*LP[4]*LP[5];
```

```

<
[30  0  8 17]
[ 0 30 10 28]
>
```

```
[ 0  0  1  0]
```

```
[ 0  0  0  1]
```

```
>
```

```
kash> c := IdealMakeInvCoprime(A,B);
```

```
[-195, -206, 11, -7]
```

```
kash> D := c/A;
```

```
<
```

```
[-195 -382  -67  -26]
```

```
[-206 -442  -88 -103]
```

```
[ 11 -555   99  -92]
```

```
[  -7  724 -376   -9]
```

```
>
```

```
kash> IdealIdempotents([D,B]);
```

```
[ 1623181771, -1623181770 ]
```

NAME	<code>IdealMin</code>
PURPOSE	The minimum of the ideal.
SYNTAX	<pre>m:=IdealMin(I);</pre> <pre>integer m ideal I</pre>
DESCRIPTION	<p>The intersection of the ideal with the coefficient ring (which is \mathbb{Z} for absolute ideals and the coefficient order for relative ideals) defines an ideal over the coefficient ring.</p> <p>In case of absolute ideals the principal generator of this ideal is returned (which is the smallest positive rational integer in the ideal), in case of relative ideals the ideal itself is returned.</p> <p>Iterative application of <code>IdealMin</code> to a relative ideal eventually leads to the smallest positive rational integer in the ideal.</p> <p>Once computed the minimum is stored in the ideal data structure so it has not to be computed again.</p>
SEE ALSO	IdealGen ,
EXAMPLE	for absolute ideals

```
kash> O := OrderMaximal(Order(Poly(Zx,[1,4,1,-4,-3,7])));
Generating polynomial: x^5 + 4*x^4 + x^3 - 4*x^2 - 3*x + 7
Discriminant: 28442269
```

```
kash> a := Elt(0,[1,5,1,8,0]);
[1, 5, 1, 8, 0]
kash> I := Ideal(a);
<[1, 5, 1, 8, 0]>
kash> IdealMin(I);
3563523
```

NAME	IdealMove						
PURPOSE	Embeds one or more ideals into a given order.						
SYNTAX	<pre>I2 := IdealMove(I1, o);</pre> <table> <tr> <td>same type as I1</td><td>I2</td></tr> <tr> <td>ideal list of ideals</td><td>I1</td></tr> <tr> <td>order</td><td>o</td></tr> </table>	same type as I1	I2	ideal list of ideals	I1	order	o
same type as I1	I2						
ideal list of ideals	I1						
order	o						
DESCRIPTION	<p>If I1 is an ideal, it is moved into o. This is done via a 2-element representation of the ideal. We assume that KASH already knows an embedding of the coefficient order of I1 into o.</p> <p>If I1 is a (factor) list of ideals all ideals contained are moved.</p>						
SEE ALSO	EltMove ,						
EXAMPLE	We will create an order \mathcal{O} , search (and find) an order \mathfrak{o} created by a better polynomial (smaller index or coefficients), factor 2 in \mathfrak{o} where it is no index divisor and move it to \mathcal{O} .						

```
kash> O := OrderMaximal(Order(x^6 - 9*x^4 - 4*x^3 + 27*x^2 - 36*x - 23));;
kash> o := OrderShort(O);;
kash> IdealMove(2*o, O);
<2>
kash> IdealMove(Factor(2*o), O);
[ [ <2, [1, -2, 2, 1, 4, -4]>, 6 ] ]
```

NAME	<code>IdealNorm</code>
PURPOSE	Returns the norm of an ideal.
SYNTAX	<pre>n := IdealNorm(I);</pre> <p> <code>ideal</code> <code>I</code> <code>rational number</code> <code>n</code> </p>
DESCRIPTION	<p>If the ideal \mathfrak{a} is an ideal over an absolute order \mathcal{O} the norm is the number of elements of the finite ring \mathcal{O}/\mathfrak{a}.</p> <p>If the ideal \mathfrak{a} is an ideal over a relative order the relative norm of this ideal is returned which is the ideal generated by the norms of all elements of the ideal \mathfrak{a}.</p> <p>Iterative application of <code>IdealNorm</code> eventually leads to the absolute norm of an relative ideal.</p> <p>Once computed the minimum is stored in the ideal data structure so it has not to be computed again.</p>

SEE ALSO [Norm](#), [EltNorm](#),

EXAMPLE Computing the norm of $2 * O + (1 + \rho) * O$ where O is the maximal order of $x^2 - 2$ and $\rho^2 = 2$.

```
kash> O := OrderMaximal(Order(Poly(Zx, [1, 0, -2])));
Generating polynomial: x^2 - 2
Discriminant: 8
```

```
kash> IdealNorm(Elt(0, [1, 4])*O);
31
kash> IdealNorm (Elt (0, [3, 6]/2)*O);
63/4
```

NAME	IdealOrder
PURPOSE	This function returns the order of an ideal.
SYNTAX	<pre>O := IdealOrder(I);</pre> <pre>Order O</pre> <pre>Ideal I</pre>
DESCRIPTION	This function returns the order of an ideal under which it is defined. The order is included in the KASH data structure ideal , no calculation occurs.

EXAMPLE

```
kash> O:=Order(Z,2,-3);
Generating polynomial: x^2 + 3
```

```
kash> I:=3*O;
<3>
kash> IdealOrder(I);
Generating polynomial: x^2 + 3
```

NAME IdealPrimeCountInit

PURPOSE Initializes an environment for enumerating prime ideals

SYNTAX `s := IdealPrimeCountInit(o[, m]);`

`record s`
 `order o`
 `integer m` only primes coprime to `m` are used.

DESCRIPTION

SEE ALSO [IdealPrimeCountNext](#),

EXAMPLE See `IdealPrimeCountNext` for examples.

NAME	<code>IdealPrimeCountNext</code>
PURPOSE	Enumerates the next prime ideal.
SYNTAX	<pre>p := IdealPrimeCountNext(s [, 1]);</pre> <p>prime ideal record if present, p will be a pair $[p, f]$ with p a prime of the coefficient order and</p>
DESCRIPTION	Be careful with relative extensions: index divisors won't work here!
SEE ALSO	IdealPrimeCountInit ,
EXAMPLE	We'll work in the field $\mathbb{Q}(\sqrt{2})$:

```
kash> o := OrderMaximal(Z, 2, 2);
Generating polynomial: x^2 - 2
Discriminant: 8
```

```
kash> s := IdealPrimeCountInit(o);
Record of type IdealPrimeCount
```

```
kash> repeat
> p := IdealPrimeCountNext(s);
> Print(p, "\n");
> until IdealMin(p)>20;
<2, [0, 1]>
<3>
<5>
<7, [3, 1]>
<7, [4, 1]>
<11>
<13>
<17, [6, 1]>
<17, [11, 1]>
<19>
<23, [5, 1]>
```

Next, only the factorization shape of ideals coprime to 2 is investigated:

```
kash> s := IdealPrimeCountInit(o, 2);
```

```
Record of type IdealPrimeCount
```

```
kash> repeat
```

```
> fs := IdealPrimeCountNext(s, 1);
```

```
> Print(fs, "\n");
```

```
> until fs[1]>20;
```

```
[ 3, 2 ]
```

```
[ 5, 2 ]
```

```
[ 7, 1 ]
```

```
[ 11, 2 ]
```

```
[ 13, 2 ]
```

```
[ 17, 1 ]
```

```
[ 19, 2 ]
```

```
[ 23, 1 ]
```

NAME	<code>IdealPrimeElt</code>				
PURPOSE	primitive element of an ideal				
SYNTAX	<code>e:=IdealPrimeElt(I);</code> <table> <tr> <td>algebraic number</td><td><code>e</code></td></tr> <tr> <td>Ideal</td><td><code>I</code></td></tr> </table>	algebraic number	<code>e</code>	Ideal	<code>I</code>
algebraic number	<code>e</code>				
Ideal	<code>I</code>				
DESCRIPTION	<p>An element of an ideal is called primitive if it is not contained in the square of the ideal. Once computed, the primitive element is stored in the ideal data structure so it will not be recomputed.</p> <p>This function is implemented only for absolute ideals.</p>				
SEE ALSO	IdealMin , IdealGen ,				
EXAMPLE	<pre> kash> O := OrderMaximal(Order(Poly(Zx, [1, 6, 6, 6]))); kash> Lp := Factor(6*O); [[<2, [0, 1, 0]>, 3], [<3, [0, 1, 0]>, 3]] kash> I := Lp[1][1]*Lp[2][1]; < [6 0 0] [0 1 0] [0 0 1] > kash> IdealPrimeElt(I); [0, 1, 35] </pre>				

NAME IdealRadical

PURPOSE Computes a radical.

SYNTAX `r := IdealRadical (A, 0);`

`order 0`

`ideal A`

`ideal r`

DESCRIPTION

EXAMPLE

```
kash> o := OrderMaximal(Z,2,-5);
```

```
Generating polynomial: x^2 + 5
```

```
Discriminant: -20
```

```
kash> A := 6*o;
```

```
<6>
```

```
kash> IdealDivisors(A);
```

```
[ <1>, <2, [1, 1]>, <4, [0, 2]>, <3, [1, 1]>, <
```

```
  [6 1]
```

```
  [0 1]
```

```
>
```

```
, <6, [2, 2]>, <3, [2, 1]>, <
```

```
  [6 5]
```

```
  [0 1]
```

```
>
```

```
, <6, [4, 2]>, <3>, <
```

```
  [6 3]
```

```
  [0 3]
```

```
>
```

```
, <6> ]
```

NAME	<code>IdealRamIndex</code>
PURPOSE	Calculates the ramification index of a prime ideal.
SYNTAX	<pre>d := IdealRamIndex(I);</pre> <p>ideal I must be prime integer d</p>
DESCRIPTION	Let \mathcal{O} a maximal order. Let p be a prime number and \mathfrak{p} a prime \mathcal{O} -ideal over this prime number p . The natural number e which satisfies $\mathfrak{p}^e \mid p\mathcal{O}$ and $\mathfrak{p}^{e+1} \nmid p\mathcal{O}$ is called the ramification index of \mathfrak{p} over p .
SEE ALSO	IdealDegree ,
EXAMPLE	

```
kash> O := OrderMaximal(Order(Poly(Zx,[1,4,1,-4,-3,7])));
Generating polynomial: x^5 + 4*x^4 + x^3 - 4*x^2 - 3*x + 7
Discriminant: 28442269
```

```
kash> P := Factor(6*O)[2][1];
<2, [1, 0, 0, 1, 1]>
kash> IdealDegree(P);
4
kash> IdealRamIndex(P);
1
```

NAME	<code>IdealRayClassRep</code>
PURPOSE	Returns a representative of the class of an ideal in the ray class group.
SYNTAX	<pre> r := IdealRayClassRep(I,m0,minf); matrix r ideal I ideal m0 list minf of integers/infinite primes </pre>
DESCRIPTION	<p>Decomposes an ideal, or more exactly, the class of this ideal (that need to be coprime to m_0) into a power product of the classes generating the whole ray class group.</p> <p>Based on the solutions for the discrete logarithm problem for class groups ([Heß96]) and for ray residue rings (see <code>EltRayResidueRingRep</code>), an exponent vector relative to the generators is computed.</p>
SEE ALSO	<code>EltCon</code> , <code>OrderClassGroup</code> , <code>RayClassGroup</code> , <code>RayClassGroupCyclicFactors</code> ,

EXAMPLE

```

kash> O := OrderMaximal(Order(x^2-2*x-5));
Generating polynomial: x^2 - 2*x - 5
Discriminant: 24

kash> OrderClassGroup(0,500,"euler","fast");
[ 1, [ 1 ] ]
kash> m0 := 27*O;;
kash> minf := [2];
kash> L := RayClassGroupCyclicFactors(m0,minf);
[ [ <[1344, 388]>, 3 ], [ <[1612, 444]>, 9 ] ]
kash> I := L[1][1];
<[1344, 388]>
kash> IdealRayClassRep(I,m0,minf);
[1 0]

```

NAME	<code>IdealRemainderSet</code>
PURPOSE	Computes a remainder system of an ideal
SYNTAX	$L := \text{IdealRemainderSet}(I);$ <div style="margin-left: 100px;"> <code>list</code> <code>L</code> <code>ideal</code> <code>I</code> </div>
DESCRIPTION	This function returns a list with the complete remainder set of an ideal in an order
SEE ALSO	IdealNorm ,
EXAMPLE	

```

kash> O:=Order(Poly(Zx,[1,0,2,4,58]));
Generating polynomial: x^4 + 2*x^2 + 4*x + 58

kash> E:=Elt(0,[1,2,-1,-2]);
[1, 2, -1, -2]
kash> I:=Ideal(3,E^7);
<3, [-470899145529, -57698441418, 37053141409, 29584796218]>
kash> IdealRemainderSet(I);
[ -1, 0, 1 ]

```

NAME	IdealResidueField
PURPOSE	Returns the finite field defined by a prime ideal.
SYNTAX	$K := \text{IdealResidueField}(p);$ <div style="display: flex; justify-content: space-around; margin-top: 10px;"> <div style="text-align: center;"> ideal finite field </div> <div style="text-align: center;"> p K </div> </div>
DESCRIPTION	<p>The field can be written as</p> $\mathcal{O}/\mathfrak{p},$ <p>where \mathfrak{p} is the prime ideal and \mathcal{O} the defining order.</p>
SEE ALSO	IdealResidueFieldIsomorphism , EltToFFE , FFToElt , RayResidueRing ,
EXAMPLE	

```
kash> O := OrderMaximal(Order(Poly(Zx,[1,4,1,-4,-3,7])));
Generating polynomial: x^5 + 4*x^4 + x^3 - 4*x^2 - 3*x + 7
Discriminant: 28442269
```

```
kash> p := Factor(7*O)[1][1];
<7, [0, 1, 0, 0, 0]>
kash> IdealResidueField(p);
Finite field of size 7
kash> a := Elt (0, [1,234,54,57,4]);
[1, 234, 54, 57, 4]
kash> EltToFFE (a, p);
1
kash> a := Elt (0, [4,234,54,57,4]);
[4, 234, 54, 57, 4]
kash> b:=EltToFFE (a, p);
4
kash> TYPE (b);
"KANT finite field elt"
```


NAME	<code>IdealResidueFieldIsomorphism</code>						
PURPOSE	Calculates the isomorphism between the residue fields of two prime ideals.						
SYNTAX	<pre>alpha := IdealResidueFieldIsomorphism(a,b);</pre> <table> <tr> <td>algebraic element</td><td>alpha</td></tr> <tr> <td>ideal</td><td>a</td></tr> <tr> <td>ideal</td><td>b</td></tr> </table>	algebraic element	alpha	ideal	a	ideal	b
algebraic element	alpha						
ideal	a						
ideal	b						
DESCRIPTION	This function calculates an algebraic element α which corresponds to the isomorphism between the residue fields of the prime ideals.						
SEE ALSO	IdealResidueField ,						
EXAMPLE							

```
kash> O:=OrderMaximal(Order(Poly(Zx,[1,0,-4,0,1])));
```

```
Generating polynomial: x^4 - 4*x^2 + 1
```

```
Discriminant: 2304
```

```
kash> L:=Factor(5*O);
```

```
[ [ <5, [1, 1, 1, 0]>, 1 ], [ <5, [1, 4, 1, 0]>, 1 ] ]
```

```
kash> IdealResidueFieldIsomorphism(L[1][1],L[2][1]);
```

```
[0, 4, 0, 0]
```

NAME IdealRingOfMultipliers

PURPOSE Computes the ring of multipliers of an ideal.

SYNTAX `o := IdealRingOfMultipliers (A);`

`order o`

`ideal A`

DESCRIPTION

EXAMPLE

```
kash> o := OrderMaximal(Z,2,-5);
```

```
Generating polynomial: x^2 + 5
```

```
Discriminant: -20
```

```
kash> A := 6*o;
```

```
<6>
```

```
kash> IdealDivisors(A);
```

```
[ <1>, <2, [1, 1]>, <4, [0, 2]>, <3, [1, 1]>, <
```

```
  [6 1]
```

```
  [0 1]
```

```
>
```

```
, <6, [2, 2]>, <3, [2, 1]>, <
```

```
  [6 5]
```

```
  [0 1]
```

```
>
```

```
, <6, [4, 2]>, <3>, <
```

```
  [6 3]
```

```
  [0 3]
```

```
>
```

```
, <6> ]
```

NAME	<code>IdealUpperHNFTrans</code>
PURPOSE	the transformation matrix from the original basis to the basis in upper HNF
SYNTAX	<code>M:=IdealUpperHNFTrans(I);</code> matrix M ideal I
DESCRIPTION	This returns the transformation matrix when transforming the original basis (which can be seen with <code>IdealBasis</code>) to the basis in upper HNF (which can be seen with <code>IdealBasisUpperHNF</code>). Once computed the transformation matrix is stored in the ideal data structure so it has not to be computed again.
SEE ALSO	<code>IdealBasisUpperHNF</code> ,
EXAMPLE	

```
kash> O:=OrderMaximal(Order(Poly(Zx,[1,4,1,-4,-3,7])));
Generating polynomial: x^5 + 4*x^4 + x^3 - 4*x^2 - 3*x + 7
Discriminant: 28442269
```

```
kash> I:=Ideal(O,Mat(Z,[[1,2,1,4,6],[3,8,4,6,2],[6,4,3,3,8],
> [3,2,7,9,5],[0,2,8,8,4]]),1);
<
[1 2 1 4 6]
[3 8 4 6 2]
[6 4 3 3 8]
[3 2 7 9 5]
[0 2 8 8 4]
>
```

```
kash> IdealBasis(I)[2]*IdealUpperHNFTrans(I);
[839 755 349 421 449]
[ 0  3  0  1  0]
[ 0  0  1  0  0]
[ 0  0  0  1  0]
[ 0  0  0  0  2]
kash> IdealBasisUpperHNF(I);
[ 1, [839 755 349 421 449]
    [ 0  3  0  1  0]
    [ 0  0  1  0  0]
    [ 0  0  0  1  0]
    [ 0  0  0  0  2] ]
```

NAME	IdealValuation
PURPOSE	Computes the valuation of an ideal at a prime ideal.
SYNTAX	<pre>val := IdealValuation(p, I);</pre> <p> ideal P must be prime ideal I integer val </p>
DESCRIPTION	<p>Consider the factorization of I in prime ideals. This function returns the exponent of P in the factorization (possibly 0). The prime ideal must not be an index divisor.</p> <p>The ideal may be fractional, see IdealFactor for interpretation.</p> <p>This function is only implemented for ideals over absolute orders.</p>
SEE ALSO	IdealFactor ,
EXAMPLE	

```
kash> O := OrderMaximal (Order (Poly (Zx,[1,6,6,6])));
Generating polynomial: x^3 + 6*x^2 + 6*x + 6
Discriminant: -1836
```

```
kash> P := Factor(5*O)[1][1];
<5, [1, 1, 0]>
kash> IdealValuation (P, Elt (O, [5,5,5])*O);
1
kash> P := Factor(2*O)[1][1];
<2, [0, 1, 0]>
kash> IdealValuation (P, Elt (O, [8,8,8])*O);
9
kash> IdealValuation (P, 2*O);
3
```

NAME	IdealWithNorm
PURPOSE	Computed all ideals of \mathfrak{o} of a given norm n .
SYNTAX	$L := \text{IdealWithNorm}(n, \mathfrak{o});$
	<div> <div>List</div> <div>integer</div> <div>MaximalOrder</div> </div> <div> <div>L</div> <div>$n > 1$</div> <div>\mathfrak{o}</div> </div>

EXAMPLE We have for example:

```
kash> o := OrderMaximal(Z, 2, 3);
Generating polynomial:  $x^2 - 3$ 
Discriminant: 12
```

```
kash> IdealWithNorm(2,o);
[ <2, [1, 1]> ]
kash> IdealWithNorm(11*13,o);
[ <
  [143  82]
  [  0   1]
>
, <
[143 126]
[  0   1]
>
, <
[143  17]
[  0   1]
>
, <
[143  61]
[  0   1]
>
]
```

IdemLift

NAME	IdemLift
PURPOSE	Lifts an idempotent.
SYNTAX	$\alpha := \text{IdemLift}(a, p, k);$ algebraic element α algebraic element a integer p integer k
DESCRIPTION	a is an idempotent for a prime ideal \mathfrak{p} over p . This function calculates the idempotent α corresponding to the ideal \mathfrak{p}^{2^k} .
EXAMPLE	

```
kash> O:=OrderMaximal(Order(Poly(Zx,[1,0,-4,0,1])));  
Generating polynomial: x^4 - 4*x^2 + 1  
Discriminant: 2304  
  
kash> L:=Factor(5*O);  
[ [ <5, [1, 1, 1, 0]>, 1 ], [ <5, [1, 4, 1, 0]>, 1 ] ]  
kash> L1:=IdealIdempotents([L[1][1],L[2][1]]);  
[ [-2, 0, 0, 2], [3, 0, 0, -2] ]  
kash> IdemLift(L1[1],5,3);  
[-195312, 112715, 0, -22543]
```

NAME Im

PURPOSE Returns the imaginary part of a complex number.

SYNTAX `a := Im(z);`

`real a`
 `complex z`

SEE ALSO [Re](#),

EXAMPLE Compute the imaginary part of $1 + 2i$:

```
kash> z := Comp(1, 2);  
1 + 2*i  
kash> Im(z);  
2
```

NAME ImQuadFormCreate

PURPOSE Generates a quadratic form.

SYNTAX $g := \text{ImQuadFormCreate}(D, p);$
 $g := \text{ImQuadFormCreate}(D, [a, b, c]);$

DESCRIPTION Let D be a fundamental discriminant and p be a prime or $[a, b, c]$ a list of integers. This function generates the quadratic form with $a=p$ and discriminant D if there exists one or proves if the discriminant of the suggested form $[a, b, c]$ is D . The group operations is written additively.

EXAMPLE

EXAMPLE

NAME	ImQuadHilbert
PURPOSE	Determines the Hilbert class field of an imaginary quadratic field.
SYNTAX	<pre> O := ImQuadHilbert(o [,repr] [,"roots"]); </pre> <p> order o imaginary quadratic field list repr representants for the ideal classes of o order O Hilbert class field </p>
DESCRIPTION	<p>Let K be an imaginary quadratic field and $K (1)$ its Hilbert class field. Possible generators for $K (1)$ over K are for example quotients of the Dedekind η-function of the form</p> $\left(\frac{\eta(a)\eta(b)}{\eta(ab)\eta(\mathcal{O} K)} \right)^n, \quad n \in \mathbb{N},$ <p>where the integral $\mathcal{O} K$-ideals a and b have to be suitably normalized (this is implicitly contained in [Sch90b]). A defining equation for $\mathcal{O} K (1)$ is calculated using complex approximations one gets by evaluating the defining infinite series of the Dedekind η-function.</p> <p>If "roots" is specified, also the roots of the minimal polynomial in the ordering given by the ordering of the used ideal class representants (that correspond to the Galois automorphisms by use of the Artin map) are returned.</p>
SEE ALSO	OrderHilbert , ImQuadRayField , RayClassField ,
EXAMPLE	<pre> kash> o := OrderMaximal(Z, 2, -47); F[1] F[2] / / Q F [1] Given by transformation matrix F [2] x^2 + 47 Discriminant: -47 kash> oK1 := ImQuadHilbert(o); used real precision is 52 places F[1] </pre>

```

      /
      /
E1[1]
 |
E1[2]
 /
 /
Q
F [ 1]      x^5 + 3*x^2 + 2*x - 1
E 1[ 1]      Given by transformation matrix
E 1[ 2]      x^2 + 47

```

Test for correctness (in other examples the coefficients of the generating polynomial one gets by using the routine `ImQuadHilbert` are usually quite small):

```

kash> O := OrderMaximal(OrderAbs(oK1));
      F[1]
      |
      F[2]
      /
      /
Q
F [ 1]      Given by transformation matrix
F [ 2]      x^10 + 235*x^8 + 6*x^7 + 22094*x^6 - 284*x^5 + 1037299*x^4 - 65318\
*x^3 + 24355069*x^2 - 1890344*x + 229780651
Discriminant: -229345007

kash> Factor(Abs(OrderDisc(O)));
[ [ 47, 5 ] ]

```

NAME	ImQuadRayField
PURPOSE	Determines the ray class field modulo ideal f over an imaginary quadratic field.
SYNTAX	<pre>0 := ImQuadRayField(f [, "char" "field" "maxord"]);</pre> <p> ideal f integral ideal in imaginary quadratic field order 0 ray class field modulo f </p>
DESCRIPTION	<p>Let K be a imaginary quadratic field with maximal order $\mathcal{O} K$, f an integral $\mathcal{O} K$-ideal and $K (f)$ the ray class field over K corresponding to f. Integral generators for $K (f)$ over the Hilbert class field $K (1)$ are for example elements of the form</p> $\Theta = \alpha \left(P(1 f) - D \right),$ <p>where $P(1 f)$ is a normalized singular value of the Weierstrass \wp-function and α and D are suitable choosen elements of $K (1)$ (see [Sch90b, Sch90a]). Furthermore we have an integral basis for $K (f)/K (1)$ given by</p> $\begin{aligned} \mathcal{O} K (f) = \mathcal{O} K (1) + \Theta \mathcal{O} K (1) &+ \alpha^{-1} \Theta^2 \mathcal{O} K (1) + \\ &+ \dots + \alpha^{-n+2} \Theta^{n-1} \mathcal{O} K (1) \end{aligned}$ <p>The minimal polynomial for Θ is calculated using complex approximations one gets by evaluating the defining infinite series of the Dedekind η-function and the Weierstrass \wp-function.</p> <p>If "char" is specified only the characteristic polynomial for Θ over K is computed (which might be reducible). In case of "field" also the Hilbert class field (and the representation of the ray class field over the Hilbert class field) is determined, and if "maxord" or nothing is specified (but that might cause a long output) one gets the maximal order of the ray class field as an extension of the maximal order of the Hilbert class field.</p> <p><i>Note:</i> For numerical reasons don't use prime ideal powers as a module whenever possible. So for example if 2 is not inert in K, $p 2$ is a prime ideal in K above 2 and f is prime to $p 2$, then we have $K (f) = K (p 2f)$ and one should use <code>ImQuadRayField(p 2f)</code> instead of <code>ImQuadRayField(f)</code>.</p>
SEE ALSO	RayClassField , ImQuadHilbert , OrderHilbert ,
EXAMPLE	

```
kash> o := OrderMaximal(Z, 2, -7);
F[1]
|
```

```

      F[2]
    /
  /
Q
F [ 1]      Given by transformation matrix
F [ 2]      x^2 + 7
Discriminant: -7

kash> f := Factor(2*o)[1][1] * Factor(7*o)[1][1];
<
[14  3]
[ 0  1]
>

kash> O := ImQuadRayField(f, "maxord");
resulting degrees:
  [RayField:Hilbert]=3 , [Hilbert:Imaginary]=1

calculation of Hilbert class field ...
used real precision is 52 places

calculation of ray class field ...
used real precision is 52 places

calculation of representation with integral relative basis...
      F[1]
    /
  /
  E1[1]
  |
  E1[2]
  /
  /
Q
F [ 1]      x^3 + [0, 1]*x^2 + [-1, 1]*x - 1
E 1[ 1]      Given by transformation matrix
E 1[ 2]      x^2 + 7

```

Here Hilbert class field and imaginary field are the same (E1). F[1] is the maximal order of the ray class field, compare also the corresponding discriminants:

```
kash> Factor(OrderDisc(OrderCoefOrder(0)));  
[ [ -1, 1 ], [ 7, 1 ] ]  
kash> Factor(OrderDisc(0));  
[ [ <7, [-1, 2]>, 2 ] ]
```

Index

NAME	Index
PURPOSE	Computes the index of an algebraic element, an order or an alff order
SYNTAX	$I := \text{Index } (a);$ $I := \text{Index } (o);$ coefficient ring element I algebraic element a order alff order o
DESCRIPTION	For an algebraic element a in an arbitrary order o over \mathbb{Z} this function returns the modul index ($o : \mathbb{Z}[\alpha]$) Given an order this function returns the index of the suborder o_1 of o in o . Given an alff order this function returns the index of the equation order in the given order.
SEE ALSO	EltIndexOrderIndexAlffOrderIndex ,
EXAMPLE	

```
kash> o := Order(Z,5,3);  
Generating polynomial: x^5 - 3
```

```
kash> a := Elt(o, [0,0,1,1,1]);  
[0, 0, 1, 1, 1]  
kash> I := Index(a);  
3501
```

NAME	InftyGcd
PURPOSE	Computes a greatest common divisor with respect to the degree valuation.
SYNTAX	<code>c := InftyGcd(a, b);</code> quotient field elements <code>a,b,c</code>
DESCRIPTION	Let $a, b \in \mathbb{F}_q(x)$. This function returns $x^{-\min(\nu_\infty(a), \nu_\infty(b))}$.
SEE ALSO	InftyVal , InftyQuotRem , InftyLcm ,
EXAMPLE	

```

kash> k := FF(5, 2);
Finite field of size 5^2
kash> kx := PolyAlg(k);
Univariate Polynomial Ring in x over GF(5^2)

kash> x := Poly(kx, [1,0]);
x
kash> InftyGcd(x^2+1, x^3+1);
x^3

```

NAME	InftyLcm
PURPOSE	Computes a least common multiple with respect to the degree valuation.
SYNTAX	$c := \text{InftyLcm}(a, b);$ quotient field elements a, b, c
DESCRIPTION	Let $a, b \in \mathbb{F}_q(x)$. This function returns $x^{-\max(\nu_\infty(a), \nu_\infty(b))}$.
SEE ALSO	InftyVal , InftyQuotRem , InftyGcd ,
EXAMPLE	

```

kash> k := FF(5, 2);
Finite field of size 5^2
kash> kx := PolyAlg(k);
Univariate Polynomial Ring in x over GF(5^2)

kash> x := Poly(kx, [1,0]);
x
kash> InftyLcm(x^2+1, x^3+1);
x^2

```


NAME	InftyQuotRem
PURPOSE	Computes quotient and remainder with respect to the degree valuation.
SYNTAX	$L := \text{InftyQuotRem}(a, b);$ <p>list L of q and r quotient field elements a, b</p>
DESCRIPTION	Let $a, b \in \mathbb{F}_q(x)$. The function computes $q, r \in \mathbb{F}_q(x)$ such that $a = qb + r$ and $\nu_\infty(q) \geq 0$. The rest r is chosen such that $r = \sum_{i=\nu_\infty(a)}^{\nu_\infty(b)-1} c_i x^{-i}$ and $c_i \in \mathbb{F}_q$. By these conditions q, r are uniquely determined.
SEE ALSO	InftyVal , InftyGcd , InftyLcm ,
EXAMPLE	

```
kash> k := FF(5, 2);
Finite field of size 5^2
kash> kx := PolyAlg(k);
Univariate Polynomial Ring in x over GF(5^2)
```

```
kash> x := Poly(kx, [1,0]);
x
kash> a := x + 2 + 3/x + 4/x^2;
(x^3 + 2*x^2 + 3*x + 4)/x^2
kash> InftyQuotRem(a, x);
[ (x^3 + 2*x^2 + 3*x + 4)/x^3, 0 ]
kash> InftyQuotRem(a, x/x);
[ (2*x^2 + 3*x + 4)/x^2, x ]
kash> InftyQuotRem(a, 1/x);
[ (3*x + 4)/x, x + 2 ]
kash> InftyQuotRem(a, 1/x^2);
[ 4, (x^2 + 2*x + 3)/x ]
kash> InftyQuotRem(a, 1/x^3);
[ 0, (x^3 + 2*x^2 + 3*x + 4)/x^2 ]
```

NAME	InftyVal
PURPOSE	Returns the degree valuation of a rational function or a Puiseux series defined over a finite field.
SYNTAX	<pre>n := InftyVal(a);</pre> <p>integer n quotient field element a of $\mathbb{F}_q(x)$</p>
DESCRIPTION	This function computes the degree valuation $\nu_\infty(a)$ of an element $a \in \mathbb{F}_q(x)^\times$ or $a \in \mathbb{F}_q((x^{-1/e}))$ (the negative degree).
SEE ALSO	InftyQuotRem , InftyGcd , InftyLcm , AlffRoots ,
EXAMPLE	

```
kash> k := FF(5, 2);
Finite field of size 5^2
kash> kx := PolyAlg(k);
Univariate Polynomial Ring in x over GF(5^2)

kash> x := Poly(kx, [1,0]);
x
kash> InftyVal(x);
-1
kash> InftyVal((x^2+x)^-1);
2
```

NAME Insert

PURPOSE Insert an element into a list at a given position.

SYNTAX Insert(L, a, pos);

list	L
arbitrary object	a
integer	pos

EXAMPLE

```
kash> L := [1, 2, 3];  
[ 1, 2, 3 ]  
kash> Insert(L, 7, 1);  
kash> L;  
[ 7, 1, 2, 3 ]  
kash> Insert(L, 8, 5);  
kash> L;  
[ 7, 1, 2, 3, 8 ]
```

NAME IntDivisors

PURPOSE Returns a sorted list of all non-negative divisors of a rational integer.

SYNTAX L := IntDivisors(d);

list L

integer d

DESCRIPTION

SEE ALSO [IntFactor](#),

EXAMPLE Divisors of 30030:

```
kash> IntDivisors(30030);
```

```
[ 1, 2, 3, 5, 6, 7, 10, 11, 13, 14, 15, 21, 22, 26, 30, 33, 35, 39, 42, 55,
  65, 66, 70, 77, 78, 91, 105, 110, 130, 143, 154, 165, 182, 195, 210, 231,
  273, 286, 330, 385, 390, 429, 455, 462, 546, 715, 770, 858, 910, 1001,
  1155, 1365, 1430, 2002, 2145, 2310, 2730, 3003, 4290, 5005, 6006, 10010,
  15015, 30030 ]
```

NAME IntEulerPhi

PURPOSE Computes the Euler- ϕ function of n , i.e. the number of coprime integers less than n .

SYNTAX `a := IntEulerPhi(n);`

`integer a`

`integer n`

DESCRIPTION

EXAMPLE

EXAMPLE

```
kash> IntEulerPhi(3);
2
kash> IntEulerPhi(81);
54
kash> IntEulerPhi(134);
66
```

NAME IntFactor

PURPOSE

SYNTAX `F := IntFactor(d);`

`list F`
 `integer d`

DESCRIPTION Returns the factorization of the given positive integer $d \in \mathbb{N}$.

SEE ALSO [Factor](#),

EXAMPLE Factorization of 8274626472648264826427648723648276:

```
kash> Factor(8274626472648264826427648723648276);
[ [ 2, 2 ], [ 11, 1 ], [ 41, 1 ], [ 86423, 1 ],
  [ 53074086409412759917474753, 1 ] ]
```

NAME IntGcd

PURPOSE Returns the non-negative greatest common divisor of two integers.

SYNTAX gcd := IntGcd (a1,a2);

 integer gcd

 integer a1

 integer a2

DESCRIPTION IntGcd($a,0$) or IntGcd($0,a$) is the absolute value of a . IntGcd($0,0$) is 0.

SEE ALSO [IntXGcd](#), [IntLcm](#),

EXAMPLE

```
kash> IntGcd(9, 0);
```

```
9
```

```
kash> IntGcd(-9, 0);
```

```
9
```

```
kash> IntGcd(0, 7);
```

```
7
```

```
kash> IntGcd(0, -7);
```

```
7
```

```
kash> IntGcd(2345, -505);
```

```
5
```

```
kash> IntGcd(2345, 505);
```

```
5
```

NAME IntIsPrime

PURPOSE Returns `true` iff the argument is a rational prime.

SYNTAX `b := IntIsPrime(n)`

 boolean `b`

 integer `n`

DESCRIPTION

SEE ALSO [NextPrime](#),

EXAMPLE

```
kash> IntIsPrime(1);  
false  
kash> IntIsPrime(2);  
true
```


NAME IntIsSquare

PURPOSE Returns `true` iff the argument is a square.

SYNTAX `b := IntIsSquare(n)`

 boolean `b`

 integer `n`

DESCRIPTION

EXAMPLE

```
kash> IntIsSquare(1);
true
kash> IntIsSquare(2);
false
```

NAME IntLcm

PURPOSE Returns the least positive common multiplier of two integers and zero if one of both is zero.

SYNTAX `c := IntLcm(a, b);`

 integer c

 integer a

 integer b

DESCRIPTION

SEE ALSO [IntGcd](#), [IntGcdEx](#),

EXAMPLE

```
kash> IntLcm(345, 6540);
150420
kash> IntLcm(-345, 6540);
150420
kash> IntLcm(-345, -6540);
150420
kash> IntLcm(345, -6540);
150420
kash> IntLcm(0, 1);
0
kash> IntLcm(1, 0);
0
kash> IntLcm(0, 0);
0
```

NAME IntMoebiusMy

PURPOSE Computes the Moebius μ -function.

SYNTAX `a := IntMoebiusMy(n);`

`integers a, n`

SEE ALSO [IntDivisors](#),

EXAMPLE

```
kash> IntMoebiusMy(1);
```

```
1
```

```
kash> IntMoebiusMy(2*3);
```

```
1
```

```
kash> IntMoebiusMy(2*3*5);
```

```
-1
```

```
kash> IntMoebiusMy(2^2*3*5);
```

```
0
```

NAME IntMoebiusMy

PURPOSE Computes the Moebius μ -function.

SYNTAX `a := IntMoebiusMy(n);`

`integers a, n`

SEE ALSO [IntDivisors](#),

EXAMPLE

```
kash> IntMoebiusMy(1);
```

```
1
```

```
kash> IntMoebiusMy(2*3);
```

```
1
```

```
kash> IntMoebiusMy(2*3*5);
```

```
-1
```

```
kash> IntMoebiusMy(2^2*3*5);
```

```
0
```

NAME IntPowerMod

PURPOSE Returns the power of a given integer modulo an integer.

SYNTAX `y := IntPowerMod(x, n, m);`

 integer y

 integer x

 integer n

 integer m

DESCRIPTION This function computes $y = x^n \bmod m$ where n is non-negative and m greater than one.

EXAMPLE

```
kash> IntPowerMod(3, 1000, 17);
```

```
16
```

NAME IntPrimeDivisors

PURPOSE Returns a sorted list of all prime divisors of a rational integer.

SYNTAX L := IntPrimeDivisors(d);

list L

integer d

DESCRIPTION

SEE ALSO [IntFactor](#),

EXAMPLE Divisors of 30030:

```
kash> IntPrimeDivisors(30030);  
[ 2, 3, 5, 7, 11, 13 ]
```

NAME	IntQuo
PURPOSE	Returns the integer quotient of two integers.
SYNTAX	<pre>q := IntQuo(a, b); integer q integer a integer b</pre>
DESCRIPTION	Returns q of $a = q * b + r$ subject to $ r < b $. The integer b must not be zero.
EXAMPLE	<pre>kash> IntQuo(20, 3); 6 kash> IntQuo(-20, 3); -7 kash> IntQuo(20, -3); -7 kash> IntQuo(-20, -3); 6</pre>

NAME IntRandomBits

PURPOSE Returns a random integer in the range 0 to $2^n - 1$. n must be ≥ 0 .

SYNTAX `y := IntRandomBits(n);`

integer y
small integer n
H

DESCRIPTION

EXAMPLE

```
kash> y:=IntRandomBits(100);
64802451788490783625902680364
```


NAME IntRoot

PURPOSE Returns $\lfloor \sqrt[n]{n} \rfloor$.

SYNTAX `m := IntRoot(r, n)`

 boolean `m`

 integer `n`

 integer `r`

DESCRIPTION

EXAMPLE

```
kash> IntRoot(8, 3);
```

```
2
```

```
kash> IntRoot(82, 4);
```

```
3
```

NAME IntToChar

PURPOSE Returns the integer with (ASCII) code i , where $0 \leq i < 256$

SYNTAX `c := IntToChar(i);`

 character `c`

 integer `i`

SEE ALSO [CharToInt](#),

EXAMPLE

```
kash> IntToChar(123);  
'{'
```

NAME	IntValuation
PURPOSE	Returns the p -adic valuation of an integer
SYNTAX	$v := \text{IntValuation}(p, n);$ integer v integer p integer n
DESCRIPTION	Returns the p -adic valuation v of an integer $n \neq 0$, i.e. for a rational prime p it returns a non-negative, maximal v subject to $p^v n$.
SEE ALSO	IdealValuation , EltValuation ,
EXAMPLE	

```
kash> IntValuation(1208724496185624589, 1180992730069637804054573);  
1
```

NAME	IntXGcd
PURPOSE	Extended Euclidean algorithm.
SYNTAX	$G := \text{IntXGcd } (a_1, a_2);$ $G := \text{IntXGcd } (L);$ <div style="margin-left: 40px;"> list G integer a_1 integer a_2 list L </div>
DESCRIPTION	<p>The <code>IntXGcd</code> function computes the greatest common divisor g of integers a_1, \dots, a_n. g is the first element of the list G which will be returned by <code>IntXGcd</code>. The second element of G is another list which contains the representation of g by a_1, \dots, a_n.</p> <p><code>IntXGcd (a1,a2);</code> computes the greatest common divisor of a_1, a_2 and its representation.</p> <p><code>IntXGcd (L);</code> Let L be a list of rational integers a_1, \dots, a_n. The <code>IntXGcd</code> computes the greatest common divisor of a_1, \dots, a_n and its representation.</p>
SEE ALSO	IntGcd ,

EXAMPLE

```

kash> G := IntXGcd(2345,-505);
[ 5, [ 14, 65 ] ]
kash> G[2][1]*2345+G[2][2]*(-505);
5
kash> G := IntXGcd([12,18,21]);
[ 3, [ 3, -3, 1 ] ]
kash> G[2][1]*12+G[2][2]*18+G[2][3]*21;
3

```


NAME	IntegralPoints
PURPOSE	Computes all integral points on an elliptic curve in normal form.
SYNTAX	<pre> L := IntegralPoints(k); L := IntegralPoints(a,b); list L integer k integer a integer b </pre>
DESCRIPTION	<p>The IntegralPoints function computes all integral points on an elliptic curve in normal form over the rational integers. The result is returned as a list of all x-y pairs solving the equation.</p> <p>IntegralPoints(k) computes all integral points of Mordell's equation $y^2 = x^3 + k$, where k is a rational integer. In this case the IntegralPoints function uses the algorithm described in [Wil].</p> <p>IntegralPoints(a,b) computes all integral points of an elliptic curve in short Weierstraß form $y^2 = x^3 + ax + b$ with rational integers a, b. The computation of all integral points is reduced to another problem in which finitely many associated quartic Thue are solved [Str84, TdW89].</p>

EXAMPLE Compute all integral points of $y^2 = x^3 + 17$.

```

kash> IntegralPoints(17);
[ [ -2, -3 ], [ -2, 3 ], [ -1, -4 ], [ -1, 4 ], [ 2, -5 ], [ 2, 5 ],
  [ 4, -9 ], [ 4, 9 ], [ 8, -23 ], [ 8, 23 ], [ 43, -282 ], [ 43, 282 ],
  [ 52, -375 ], [ 52, 375 ], [ 5234, -378661 ], [ 5234, 378661 ] ]

```

NAME	IsAlff
PURPOSE	Returns whether an object is an algebraic function field.
SYNTAX	<pre>b := IsAlff(F);</pre> <pre>boolean b</pre> <pre>object F</pre>
DESCRIPTION	Given an object this function returns whether this object is an algebraic function field.
SEE ALSO	Alff ,
EXAMPLE	

```
kash> AlffInit(FF(7,2));
"Defining global variables: k, w, kT, kTf, kTy, T, y, AlffGlobals"
kash> F := Alff((T^2+1)*y^3+y+T^4+1);
Algebraic function field defined by
$.1^3*$.2^2 + $.1^3 + $.1 + $.2^4 + 1
over
Univariate rational function field over GF(7^2)
Variables: T

kash> IsAlff(F);
true
kash> IsAlff(1);
false
```

NAME IsAlffDiff

PURPOSE Returns whether argument is an alff differential.

SYNTAX b := IsAlffDiff(a);

 boolean b
 alff differential a

SEE ALSO [AlffDiff](#),

EXAMPLE

```
kash> AlffInit(FF(5,1));;
kash> AlffOrders(y^2 + T^3 + 1);
"Defining global variables: F, o, oi, one"
kash> dT := AlffDiff(AlffEltGenT(F));
Alff Differential
[ 1, 0 ] d[ T, 0 ]
kash> IsAlffDiff(dT);
true
```


NAME	IsAlffDivisor
PURPOSE	Returns true if and only if the parameter is an algebraic function field divisor.
SYNTAX	<pre>b := IsAlffDivisor(D);</pre> <p>boolean b arbitrary object D</p>
SEE ALSO	IsAlffPlace , IsBound , Unbind ,
EXAMPLE	

```
kash> AlffInit(FF(2,4));
"Defining global variables: k, w, kT, kTf, kTy, T, y, AlffGlobals"
kash> F := Alff(y^3+T^3*y+T);
Algebraic function field defined by
$.1^3 + $.1*$.2^3 + $.2
over
Univariate rational function field over GF(2^4)
Variables: T

kash> P := AlffPlaceSplit(F, T+1)[1];
Alff place < [ T + 1, 0, 0 ] >
kash> IsAlffDivisor(P);
false
kash> D := AlffDivisor(P);
Alff divisor
[ [ Alff place < [ T + 1, 0, 0 ] >, 1 ] ]

kash> IsAlffDivisor(D);
true
```

NAME	IsAlffElt
PURPOSE	Returns true iff the argument is of type "KANT function field order elt".
SYNTAX	<pre>b := IsAlffElt(T);</pre> <div> <div>boolean</div> <div>b</div> </div> <div> <div>function field order element</div> <div>T</div> </div>

EXAMPLE

```

kash> AlffInit(Q);
"Defining global variables: k, w, kT, kTf, kTy, T, y, AlffGlobals"
kash> AlffOrders(y^3+T^4+1);
"Defining global variables: F, o, oi, one"
kash> a:=AlffElt(o,[0,1,0]);
[ 0, 1, 0 ]
kash> IsAlffElt(a);
true

```

NAME	IsAlffPlace
PURPOSE	Returns whether an object is an algebraic function field place.
SYNTAX	<pre>b := IsAlffPlace(P);</pre> <p>boolean b arbitrary object P</p>
SEE ALSO	IsBound , Unbind ,
EXAMPLE	

```
kash> H := AlffHermitianFunField(2,3);
Algebraic function field defined by
$.1^8 + $.1 + $.2^9
over
Univariate rational function field over GF(2^6)
Variables: T

kash> P := AlffPlacesDegOne(H)[1];
Alff place < [ 1/T, 0, 0, 0, 0, 0, 0, 0 ], [ w^60/T, (w^7*T + w^44)/T, (w^14*T\
+ w^26)/T, (w^37*T + w^19)/T, (w^8*T + w^12)/T, (w^33*T + w^21)/T, (w^12*T + \
w^30)/T, (w^21*T + w^57)/T ] >
kash> IsAlffPlace(P);
true
```

NAME IsChar

PURPOSE Returns true if the argument is a character, false otherwise.

SYNTAX b := IsChar(c);

 character c

 boolean b

SEE ALSO **IsString**,

EXAMPLE

```
kash> IsChar('a');
```

```
true
```

```
kash> IsChar("a");
```

```
false
```

```
kash> IsChar(1);
```

```
false
```

NAME	IsEcc
PURPOSE	Returns true iff the argument is an elliptic curve.
SYNTAX	<code>b := IsEcc(E);</code>
	<div>boolean b</div> <div> E</div>

DESCRIPTION

EXAMPLE

EXAMPLE

```

kash> p:=65112*2^144-1;
1452046121366725933991673688168680114377396846591
kash> IsPrime(p);
true
kash> E := EccInit(p,-3,49);
[ 1452046121366725933991673688168680114377396846588, 49 ]
kash> IsEcc(E);
true

```

NAME IsElt

PURPOSE Returns true iff the argument is of type "KANT algebraic element".

SYNTAX b := IsElt(x);

 boolean b
 x

EXAMPLE

```
kash> o := Order(Z, 2, 5);;
kash> IsElt(1);
false
kash> IsElt(Elt(o, 1));
true
```

NAME	IsFF
PURPOSE	Returns whether object is a finite field or not.
SYNTAX	<pre>b := IsFF(k);</pre> <pre>boolean b</pre> <pre>object k</pre>
DESCRIPTION	
SEE ALSO	FF ,
EXAMPLE	

```
IsFF(2);
IsFF(FF(2));
```

NAME	IsFFElt
PURPOSE	Returns true iff the argument is of type "KANT finite field elt".
SYNTAX	<pre>b := IsFFElt(a);</pre> <pre>boolean b</pre> <pre> a</pre>

EXAMPLE

```
kash> k := FF(2,4);  
Finite field of size 2^4  
kash> a := FFElt(k,9);  
1  
kash> IsFFElt(a);  
true
```

NAME	IsIdeal
PURPOSE	Returns true iff the argument is an ideal.
SYNTAX	$b := \text{IsIdeal}(x);$ boolean b x

EXAMPLE We have for example:

```
kash> o := Order(Z, 2, 3);  
Generating polynomial:  $x^2 - 3$ 
```

```
kash> IsIdeal(o);  
false  
kash> IsIdeal(1*o);  
true
```

NAME IsInt

PURPOSE Returns `true` iff the argument is an integer.

SYNTAX `b := IsInt(x);`

```

boolean  b
         x

```

DESCRIPTION

SEE ALSO [TYPE](#), [IsIdeal](#), [IsPoly](#), [IsOrder](#), [IsElt](#), ,

EXAMPLE

```

kash> x := Poly(Zx, [1, 0]);
x
kash> IsInt(x);
false
kash> IsInt(2*3);
true

```

NAME	IsLat
PURPOSE	Returns true iff the argument is of type "KANT lattice".
SYNTAX	<code>b := IsLat(a);</code>
	<pre> boolean b a </pre>

EXAMPLE

```
kash> lambda := Lat(MatId(Z,8));
```

```
Basis:
```

```
[1 0 0 0 0 0 0 0]
```

```
[0 1 0 0 0 0 0 0]
```

```
[0 0 1 0 0 0 0 0]
```

```
[0 0 0 1 0 0 0 0]
```

```
[0 0 0 0 1 0 0 0]
```

```
[0 0 0 0 0 1 0 0]
```

```
[0 0 0 0 0 0 1 0]
```

```
[0 0 0 0 0 0 0 1]
```

```
kash> IsLat(lambda);
```

```
true
```



```
kash> M[1];  
[1 2]  
kash> M[1][1];  
1  
kash> IsMat(false=IsMat(IsMat(IsMat(IsMat))));  
false
```

NAME	IsMat
PURPOSE	Returns true iff the argument is of type "KANT matrix".
SYNTAX	<pre>b := IsMat(x);</pre> <pre>boolean b</pre> <pre> x</pre>

EXAMPLE

```

kash> M := MatSym(Z, [[1,2,3,4], [5,6,7], [8,9], [10]] );
[ 1  2  3  4]
[ 2  5  6  7]
[ 3  6  8  9]
[ 4  7  9 10]
kash> IsMat(M);
true
kash> M := [ [1,2], [3,4] ];
[ [ 1, 2 ], [ 3, 4 ] ]
kash> IsMat(M);
true
kash> M;
[1 2]
[3 4]
kash> M[1];
[1 2]
kash> M[1][1];
1
kash> IsMat(false=IsMat(IsMat(IsMat(IsMat))));
false

```

NAME IsModule

PURPOSE Returns true iff the argument is a module.

SYNTAX **b** := IsModule(**m**);

 boolean **b**

 object **m**

EXAMPLE

```
kash> o := OrderMaximal(Z, 3, 3);;
kash> IsModule(o);
false
kash> IsModule(ModuleId(o, 4));
true
```

NAME IsOrder

PURPOSE Returns true iff the argument is an order.

SYNTAX b := IsOrder(x);

 boolean b
 x

EXAMPLE

```
kash> IsOrder(Z);  
false  
kash> IsOrder(Order(Z, 2, 3));  
true
```


NAME IsPoly

PURPOSE Returns true iff the argument is a polynomial.

SYNTAX `b := IsPoly(f);`

 boolean b
 f

EXAMPLE

```
kash> f := Poly(Zx, [1, 2, 1, 2]);  
x^3 + 2*x^2 + x + 2  
kash> IsPoly(f);  
true  
kash> IsPoly(3);  
false
```

NAME IsPrime

PURPOSE Returns `true` iff the argument is either a prime number or a prime ideal.

SYNTAX `b := IsPrime(x);`

 boolean b
 x

EXAMPLE First some rational integers

```
kash> IsPrime(1997);  
true  
kash> IsPrime(4);  
false
```

Now some ideals in $\mathbb{Q}(\sqrt{3})$

```
kash> o := Order(Z, 2, 3);;  
kash> OrderMaximal(o);;  
kash> IsPrime(5*o);  
true  
kash> IsPrime(6*o);  
false  
kash> IsPrime(Factor(6*o)[1][1]);  
true
```

NAME	IsQf
PURPOSE	Returns whether object is a rational function field or not.
SYNTAX	<code>b := IsQf(k);</code> boolean b object k
SEE ALSO	QuotientField , QfeQf ,
EXAMPLE	

```
kash> IsQf(1);  
false  
kash> IsQf(Zx);  
false  
kash> IsQf(QuotientField(Zx));  
true
```

NAME	IsQp
PURPOSE	Checks whether k is an element of a p -adic field.
SYNTAX	<pre>b := IsQp(k);</pre> <p> p-adic element k boolean b </p>
SEE ALSO	Qp , QpElt , QpEltToQ , QpEltQp , QpPrec , QpExp , QpLog , QpPrime , QpSqrt , QpValuation ,

EXAMPLE

```
kash> F := Qp(2);
2-adic Field mod 2^20
kash> k := QpElt(F, 23);
1 + 2 + 2^2 + 2^4
kash> b := IsQp(k);
true
```

NAME	IsQpElt
PURPOSE	Returns true if an element is a p -adic field element.
SYNTAX	<pre>b := IsQpELt(a);</pre> <p> p-adic element a boolean b </p>
SEE ALSO	IsQp , Qp , QpElt , QpEltToQ , QpEltQp , QpPrec , QpExp , QpLog , QpPrime , QpValuation ,

NAME	IsRecType
PURPOSE	Returns true iff the argument is an record with an Type enry containing the correct string.
SYNTAX	<pre>flag := IsRecType(a, type);</pre> <pre>boolean flag anything a string type</pre>
DESCRIPTION	Useful in User-supplied functions, see the Abelian group package as an example.

EXAMPLE

```
kash> a := rec(Type := "Test", data := 1);  
rec(  
  Type := "Test",  
  data := 1 )  
kash> IsRecType(a, "Test");  
true  
kash> IsRecType(a, "Paul");  
false  
kash> IsRecType(2, "two");  
false
```

NAME IsThue

PURPOSE Returns `true` iff the argument is a `Thue` object.

SYNTAX `b := IsThue(x);`

 boolean b
 x

EXAMPLE

```
kash> t := Thue([1,2,2,2,2]);  
X^4 + 2 X^3 Y + 2 X^2 Y^2 + 2 X Y^3 + 2 Y^4  
kash> IsThue(t);  
true
```

JBessel

NAME JBessel

PURPOSE Returns the value of the J-Bessel function.

SYNTAX `z := JBessel(nu,x);`

 integer nu

 real x

 real z

DESCRIPTION Given a real x and an integer nu the function returns $K_n u(x)$.

EXAMPLE

```
kash> z := JBessel(2,1.5);
```

```
0.2320876721442147272377765399247074797469350619464579
```


NAME JacobiSymbol

PURPOSE Returns the Jacobi symbol of two integers.

SYNTAX `y := JacobiSymbol(n, m);`

 integer n

 integer m

DESCRIPTION

EXAMPLE

```
kash> JacobiSymbol(1000, 17);
```

```
-1
```

KASHLEVEL

NAME	KASHLEVEL
PURPOSE	Reads or sets different defaults for KASH.
SYNTAX	<pre>x := KASHLEVEL(s); x := KASHLEVEL(s,level); integer x string s integer level</pre>
DESCRIPTION	<p>The Order command creates a new order and checks if the polynomial which creates the order is irreducible. Furthermore it calculates the real basis of the new order. It is possible to change these defaults. The real basis is only calculated if KASH_REAL_BASIS_CALC is not equal 0. If KASH_ORDER_POLY_FACTOR is 0 the test of irreducibility is omitted. In the case KASH_ORDER_POLY_FACTOR equals 1 kash tests the polynomial only in the case that the polynomial is given over \mathbb{Z}_x.</p>

EXAMPLE

```
kash> KASHLEVEL("KASH_ORDER_POLY_FACTOR",2);  
2  
kash> O:=Order(x^2-1);  
Error, polynomial is not irreducible  
kash> KASHLEVEL("KASH_ORDER_POLY_FACTOR",0);  
0  
kash> O:=Order(x^2-1);  
Generating polynomial: x^2 - 1
```

NAME	KBessel
PURPOSE	Returns the value of the K-Bessel function.
SYNTAX	<pre>z := KBessel(s,x); complex s real x complex z</pre>
DESCRIPTION	Given a complex s and a real x the function returns $K_s(x)$.
EXAMPLE	<pre>kash> s := Comp(0.5, 1); 0.5 + 1*i kash> z := KBessel(s,1); 0.2988249890873913480778254742871879580176108573144423 + 0.1189446943013590937\ 37231871224012813256270260507153*i</pre>

LOFILES

NAME	LOFILES
PURPOSE	Lists all open files, for debugging purposes only.
SYNTAX	LOFILES();
DESCRIPTION	

NAME	Lat
PURPOSE	Creates a (relative) lattice.
SYNTAX	<pre> Lambda := Lat(M, ["basis" "gram"]); Lambda := Lat(o [, "mink" "unit"]); Lambda := Lat(a); Lambda := Lat(Delta, L); Lambda := Lat(Delta, M [, "trans" "basis" "gram"]); Lambda := Lat(Module); lattice Lambda matrix M order o ideal a list L lattice Delta </pre>
DESCRIPTION	<p>At the moment there are nine different ways to define a lattice in KASH.</p> <p>Lat(M) (or Lat(M, "basis")) Let $M \in \mathbb{R}^{n \times k}$ be a matrix of rank k. Denote the columns of M by b_1, \dots, b_k. The Lat function returns the lattice $\Lambda = \mathbb{Z}b_1 + \dots + \mathbb{Z}b_k$.</p> <p>If M is a matrix over an absolute extension o of \mathbb{Q}, a relative lattice will be created. This will be an o-module $\Lambda \subset (\mathbb{R} \otimes o)^n$ together with the canonical quadratic form on $(\mathbb{R} \otimes o)^n$.</p> <p>Lat(M, "gram") Let $M \in \mathbb{R}^{k \times k}$ be a positive definite matrix. The Lat function returns the lattice Λ corresponding to the positive definite quadratic form</p> $(x_1, \dots, x_k) \cdot M \cdot \begin{pmatrix} x_1 \\ \vdots \\ x_k \end{pmatrix}.$ <p>Lat(o) (or Lat(o, "mink")) Let $\omega_1, \dots, \omega_n$ be the \mathbb{Z}-basis of the order \mathfrak{o}. We denote by $\psi : \mathfrak{o} \mapsto \mathbb{R}^n$ the Minkowski map (see EltMinkowski). The Lat function returns the lattice</p> $\Lambda = \mathbb{Z}\psi(\omega_1) + \dots + \mathbb{Z}\psi(\omega_w).$ <p>Λ has discriminant $\sqrt{ d }$ where d is the discriminant of the order \mathfrak{o}.</p>

For a relative extension $O/o/\mathbb{Q}$ given via a pseudo-basis (a module, cf. `Module`) $O = \sum_{i=1}^n \alpha_i \mathfrak{a}_i$ consider the canonical embedding

$$\iota : O \rightarrow (\mathbb{R} \otimes O) \rightarrow (\mathbb{R} \otimes o)^n,$$

where the first part of the embedding corresponds to the canonical embedding into \mathbb{C} . The relative lattice Λ created by this function is the image of O under this embedding together with the canonical quadratic form on $(\mathbb{R} \otimes o)^n$.

Lat(o, "unit")

Let $\varepsilon_1, \dots, \varepsilon_r$ be the current system of independent units of the order \mathfrak{o} . The **Lat** function returns the lattice

$$\Lambda = \mathbb{Z} \begin{pmatrix} \log |\varepsilon_1^{(1)}| \\ \vdots \\ \log |\varepsilon_1^{(r+1)}| \end{pmatrix} + \dots + \mathbb{Z} \begin{pmatrix} \log |\varepsilon_r^{(1)}| \\ \vdots \\ \log |\varepsilon_r^{(r+1)}| \end{pmatrix}.$$

Lat(a)

Let $\omega_1, \dots, \omega_n$ be the \mathbb{Z} -basis of the ideal \mathfrak{a} . We denote by $\psi : \mathfrak{o} \mapsto \mathbb{R}^n$ the Minkowski map (see `EltMinkowski`). The **Lat** function returns the lattice

$$\Lambda = \mathbb{Z} \psi(\omega_1) + \dots + \mathbb{Z} \psi(\omega_w).$$

For an ideal in a relative extension, the relative lattice is similar to the lattice corresponding to the order, i.e. the image under the canonical embedding from O to $(\mathbb{R} \otimes o)^n$.

Lat(Delta, L)

Let Δ be a lattice with basis $b_1, \dots, b_k \in \mathbb{R}^n$ and let L be a list of k positive real numbers $\lambda_1, \dots, \lambda_k$. The **Lat** function returns the lattice

$$\Lambda = \mathbb{Z} \begin{pmatrix} \lambda_1 & & \\ & \ddots & \\ & & \lambda_k \end{pmatrix} \cdot b_1 + \dots + \mathbb{Z} \begin{pmatrix} \lambda_1 & & \\ & \ddots & \\ & & \lambda_k \end{pmatrix} \cdot b_k.$$

Notice that elements can be moved between the lattices Δ and Λ .

Lat(Delta, M) (or **Lat(Delta, M, "trans")**)

Let Δ be a lattice with basis $b_1, \dots, b_k \in \mathbb{R}^n$ and let $M \in \mathbb{Z}^{k \times k}$ a unimodular matrix. The **Lat** function returns the lattice Λ with basis $(b_1, \dots, b_k) \cdot M$. Notice that elements can be moved between the lattices Δ and Λ .

Lat(Delta, M, "basis")

This calling sequence creates the same lattice as the calling sequence **Lat(M)**. However, this call enables moving elements between the lattices Δ and Λ .

`Lat(Delta,M,"gram")`

This calling sequence creates the same lattice as the calling sequence `Lat(M,"gram")`. However, this call enables moving elements between the lattices Δ and Λ .

`Lat(module)`

Basically the same as `Lat(mat)` except, that special care is taken for the coeff. ideals of the module.

EXAMPLE Creating the lattice

$$\Lambda = \mathbb{Z} \begin{pmatrix} 1 \\ 2 \end{pmatrix} + \mathbb{Z} \begin{pmatrix} 2 \\ 1 \end{pmatrix} :$$

```
kash> B := Mat(Z,[[1,2],[2,1]]);
```

```
[1 2]
```

```
[2 1]
```

```
kash> Lambda := Lat(B);
```

```
Basis:
```

```
[1 2]
```

```
[2 1]
```

NAME	LatBasis
PURPOSE	Return the basis of a lattice.
SYNTAX	<pre>M := LatBasis(Lambda);</pre> <pre>matrix M lattice Lambda</pre>
DESCRIPTION	If the basis is known it is returned as a matrix. Each column of this matrix represents a basis vector. In case of a rel. lattice, a module is returned.
EXAMPLE	

```
kash> o := Order(Z, 2, 3);;
kash> Lambda := Lat(o);;
kash> M := LatBasis(Lambda);
[1 1.73205080756887729352744634150587236694280525381]
[1 -1.73205080756887729352744634150587236694280525381]
kash> L := OrderBasis(o);;
kash> List(L, x->EltToList(EltMinkowski(x)));
[ [ 59/3277, 296/16385 ], [ 297/16385, 298/16385 ] ]
```

NAME	LatDisc
PURPOSE	Computes the discriminant of a lattice.
SYNTAX	<pre>disc := LatDisc(Lambda); real disc lattice Lambda</pre>

EXAMPLE

```
kash> Lambda := Lat(5*MatId(Z, 5));
```

```
Basis:
```

```
[5 0 0 0 0]
```

```
[0 5 0 0 0]
```

```
[0 0 5 0 0]
```

```
[0 0 0 5 0]
```

```
[0 0 0 0 5]
```

```
kash> LatDisc(Lambda);
```

```
3125
```

NAME	LatElt
PURPOSE	Creates a lattice element.
SYNTAX	<pre> a := LatElt(Lambda, L); a := LatElt(Lambda, v); lattice elt a lattice Lambda list of reals L vector over reals v </pre>
DESCRIPTION	<p>Let Λ be a lattice with basis $b_1, \dots, b_k \in \mathbb{R}^n$.</p> <p>LatElt(Lambda, L) Let L be a list containing the entries $a_1, \dots, a_k \in \mathbb{R}$. The LatElt function returns the lattice element $a_1b_1 + \dots + a_kb_k$.</p> <p>LatElt(Lambda, v) Let $v \in \mathbb{R}^n \cap [\Lambda]$. After computing $v_1, \dots, v_k \in \mathbb{R}$ with $v = v_1b_1 + \dots + v_kb_k$ the LatElt function returns the lattice element $v_1b_1 + \dots + v_kb_k$.</p>

EXAMPLE

```

kash> o := Order(Poly(Zx, [1, 13, 13, 13, 169]));;
kash> Lambda1 := Lat(o);;
kash> Lambda2 := LatLLL(Lambda1);
Basis:
[1 -8.898972742509405137989481976338606158885263693444 10.69685215937833774951\
8381293192142719370918398672 -8.3929181756421331436558159231442716924315445312\
98]
[1 0.134261593289898000224869399556756476905813297935 -23.31066585811776801515\
6201504078952000030073676501 79.4175118311570330970156479070963376256057136484\
54]
[1.414213562373095048801688724209698078569671875377 5.490479907567903997165793\
011467947079779898762665 8.919313203002418011662432704182923884009727755229 -5\
0.221971804833625921569542397881159401643756122457]
[0 2.890521001958810373589806560427662461469013267521 36.896665660756767606327\
137523226704052168032822311 64.55872298086831805701859966901857310820073028551\
9]

kash> a := LatElt(Lambda1, [-13, -3, -12, -1]);
[ -13 -3 -12 -1 ]

kash> EltMove(a, Lambda2);

```

[0 0 0 -1]

```
kash> b := LatElt(Lambda2, Mat(R, [[-8.3929181756421277, 79.4175118311570333, -50.221971804833618,
[3.8982415889784072959401304801339719125234441844412e-15 -1.250940443446586219\
619498224288088927319302801639e-16 5.27881561428295035666070314675828437059321\
869618135e-18 0.999999999999999552314203468322215678323061495939787]
```

```
kash> EltMove(b, Lambda1);
[13.000000000000002940967920453250310436100992792267 2.9999999999999979866718\
8452950496664814830634314 11.99999999999999468055859776269609170648376941886 \
0.999999999999999552314203468322215678323061495939787]
```

NAME	LatEltLength
PURPOSE	Computes the length of a lattice element.
SYNTAX	<pre>r := LatEltLength(a);</pre> <div> <div>real</div> <div>r</div> </div> <div> <div>lattice elt</div> <div>a</div> </div>

EXAMPLE

```
kash> o := Order(Poly(Zx, [1, 13, 13, 13, 169]));;
kash> Lambda := Lat(o);;
kash> a := LatElt(Lambda, [1, 2, 3, 4]);
[ 1 2 3 4 ]

kash> LatEltLength(a);
40167001.09020786180114727023964280465933321626335782
kash> elt := Elt(o, [1,2,3,4]);;
kash> EltT2(elt);
40167001.09020786180114727023964280465933321626335817
```

NAME	LatEltToList
PURPOSE	missing shortdoc
SYNTAX	$L := \text{LatEltToList}(a);$ <div style="display: flex; justify-content: space-between; margin-top: 10px;"> <div>list</div> <div>L</div> </div> <div style="display: flex; justify-content: space-between; margin-top: 10px;"> <div>lattice element</div> <div>a</div> </div>
DESCRIPTION	Let b_1, \dots, b_k be the basis of a certain lattice. Given a lattice element $a_1 b_1 + \dots + a_k b_k$ the <code>LatEltToList</code> function returns a list containing a_1, \dots, a_k .
EXAMPLE	

```
kash> Lambda := Lat(MatId(Z, 4));;
kash> a := LatElt(Lambda, [1, 2, 3, 4]);
[ 1 2 3 4 ]
```

```
kash> LatEltToList(a);
[ 1, 2, 3, 4 ]
```

NAME	LatEltVec				
PURPOSE	Computes the corresponding point of the \mathbb{R}^n .				
SYNTAX	<code>v := LatEltVec(le);</code>				
	<table> <tr> <td>vector</td><td>v</td></tr> <tr> <td>lattice element</td><td>le</td></tr> </table>	vector	v	lattice element	le
vector	v				
lattice element	le				

DESCRIPTION

EXAMPLE

```

kash> o := Order(Poly(Zx, [1, 13, 13, 13, 169]));;
kash> l := Lat(o);;
kash> a := LatElt(l, [1, 2, 3, 4]);
[ 1 2 3 4 ]

kash> LatEltVec(a);
[-6336.931799991655223227849180410447496200770397385772]
[ -74.233116314471776820846681903034918196904380465164]
[ -69.17985111789101908102124937760718501968771790037]
[ -0.212376860558238865590608061711096102287121114234]

```

NAME	LatEnum
PURPOSE	Enumeration of lattice points.
SYNTAX	<pre>ok := LatEnum(Lambda); boolean ok lattice Lambda</pre>
DESCRIPTION	Please refer to “KASH – A User’s Guide” for a detailed description of the LatEnum function.

EXAMPLE Let ρ be a zero of the polynomial $x^3 + x^2 + x - 1 \in \mathbb{Z}[x]$. We will enumerate several elements $\alpha \in \mathbb{Z}[\rho]$ with small T_2 norms. Those elements are likely to have a small norm. By assigning 4.2 to the lower bound we avoid enumeration of lattice elements which correspond to torsion units.

```
kash> o := Order(Poly(Zx, [1,1,1,-1]));;
kash> Lambda := Lat(o);;
kash> LatEnumLowerBound(Lambda, 4.2);
4.2
kash> LatEnumUpperBound(Lambda, 2*EltT2(OrderBasis(o)[3]));
13.706659122393255422774258033027130293898341365588
kash> while LatEnum(Lambda) do
> elt := EltMove(LatEnumElt(Lambda), o);
> Print(elt, "\t", EltT2(elt), "\t", Norm(elt), "\n");
> od;
[-2, -2, -1] 12.03572053128496679531111538791971960254507247658 -1
[-1, -2, -1] 9.03572053128496679531111538791971960254507247658 -4
[0, -2, -1] 12.035720531284966795311115387919719602545072476584 -7
[-2, -1, -1] 9.470353793290390217264420480058530559999788639047 -2
[-1, -1, -1] 4.470353793290390217264420480058530559999788639048 -1
[0, -1, -1] 5.470353793290390217264420480058530559999788639051 -2
[1, -1, -1] 12.47035379329039021726442048005853055999978863905 1
[-1, 0, -1] 7.853329561196627711387129016513565146949170682804 -4
[0, 0, -1] 6.853329561196627711387129016513565146949170682794 -1
[1, 0, -1] 11.853329561196627711387129016513565146949170682793 4
[-2, -1, 0] 11.974171252950407036084701722158111814747332940641 -7
[-1, -1, 0] 4.974171252950407036084701722158111814747332940639 -2
[1, -1, 0] 8.974171252950407036084701722158111814747332940643 2
-2 12 -8
```


NAME	LatEnumElt
PURPOSE	Returns the current element of an enumeration process.
SYNTAX	<pre>a := LatEnumElt(Lambda); lattice elt a lattice Lambda</pre>
EXAMPLE	See LatEnum for an example.

NAME LatEnumLowerBound

PURPOSE Reads or sets the lower bound for enumeration.

SYNTAX lbound := LatEnumLowerBound(Lambda [,lbound]);

 real lbound

 lattice Lambda

EXAMPLE See LatEnum for an example.

NAME	LatEnumPrec
PURPOSE	Sets the precision used by the enumeration function <code>LatEnum</code> .
SYNTAX	<pre> s := LatEnumPrec(Lambda); s := LatEnumPrec(Lambda,"short"); s := LatEnumPrec(Lambda,"long"); list s lattice Lambda </pre>
DESCRIPTION	If precision problems trouble the validity of the results of <code>LatEnum</code> a higher precision is necessary. Only 2 levels of precision are supported: 'short' and 'long'.

NAME	LatEnumRefVec
PURPOSE	Reads or sets a reference vector for enumeration.
SYNTAX	<pre> v := LatEnumRefVec(Lambda [,v]); vector v lattice Lambda </pre>
DESCRIPTION	<p>A reference vector of a lattice $l \subset \mathbb{R}^n$ is a vector $v \in \mathbb{R} \otimes_{\mathbb{Z}} l$. If the reference vector is set, we will enumerate points in $v - l$. Typically it is used to find points with minimal distance to the reference vector.</p>
EXAMPLE	<pre> kash> Lambda := Lat(MatId(Z, 2));; kash> LatEnumRefVec(Lambda, LatElt(Lambda, [1.7, 3.4])); [1.7 3.4] kash> LatShortestElt(Lambda); [[2 3]] </pre>

NAME	LatEnumReset
PURPOSE	Resets the enumeration environment of a lattice.
SYNTAX	<code>LatEnumReset(Lambda);</code> <code>lattice Lambda</code>
SEE ALSO	LatEnum ,

NAME	LatEnumUpperBound
PURPOSE	Reads or sets the upper bound for enumeration.
SYNTAX	<pre>ubound := LatEnumUpperBound(Lambda [,ubound]); real ubound lattice Lambda</pre>
EXAMPLE	See LatEnum for an example.

NAME	LatFinckeReduce
PURPOSE	Performs a reduction on a lattice specially suited for enumeration.
SYNTAX	<pre> Lambda2 := LatFinckeReduce(Lambda1); lattice Lambda2 lattice Lambda1 </pre>

EXAMPLE

```

kash> o := Order(Poly(Zx, [1, 13, 13, 13, 169]));;
kash> Lambda1 := Lat(o);
Basis:
[1 -11.898972742509405137989481976338606158885263693444 141.585552326981794267\
402683032916810467108819026556 -1684.72262787189544893851956638913017882108158\
1769638]
[1 -2.865738406710101999775130600443243523094186702065 8.212456615693353982370\
235100796726754005980046214 -23.5347523370329086921017815011346528531834867999\
19]
[1.414213562373095048801688724209698078569671875377 1.247839220448618850760726\
838838852844070883136534 -4.806918221932389346705562523044457400769986746645 -\
14.66724711384104594780692605259030414602229895222]
[0 2.890521001958810373589806560427662461469013267521 5.1009346392098534968392\
6535852241697600888687958 -5.324055695526355025822004314533417988312952072004]

kash> Lambda2 := LatFinckeReduce(Lambda1);
Basis:
[1 -8.898972742509405137989481976338606158885263693444 10.69685215937833774951\
8381293192142719370918398672 -8.3929181756421331436558159231442716924315445312\
98]
[1 0.134261593289898000224869399556756476905813297935 -23.31066585811776801515\
6201504078952000030073676501 79.4175118311570330970156479070963376256057136484\
54]
[1.414213562373095048801688724209698078569671875377 5.490479907567903997165793\
011467947079779898762665 8.919313203002418011662432704182923884009727755229 -5\
0.221971804833625921569542397881159401643756122457]
[0 2.890521001958810373589806560427662461469013267521 36.896665660756767606327\
137523226704052168032822311 64.55872298086831805701859966901857310820073028551\
9]

```

NAME LatGram

PURPOSE Computes the Gram matrix of a lattice.

SYNTAX `M := LatGram(Lambda);`

 matrix M
 lattice Lambda

EXAMPLE

```
kash> o := Order(Z, 2, 3);;  
kash> Lambda := Lat(o);;  
kash> LatGram(Lambda);  
[2 0]  
[0 6]
```


NAME	LatLLL
PURPOSE	Performs a LLL-reduction on a lattice.
SYNTAX	<p><code>Lambda2 := LatLLL(Lambda1);</code></p> <p><code>lattice Lambda2</code></p> <p><code>lattice Lambda1</code></p>

EXAMPLE

```

kash> o := Order(Poly(Zx, [1, 13, 13, 13, 169]));;
kash> Lambda1 := Lat(o);
Basis:
[1 -11.898972742509405137989481976338606158885263693444 141.585552326981794267\
402683032916810467108819026556 -1684.72262787189544893851956638913017882108158\
1769638]
[1 -2.865738406710101999775130600443243523094186702065 8.212456615693353982370\
235100796726754005980046214 -23.5347523370329086921017815011346528531834867999\
19]
[1.414213562373095048801688724209698078569671875377 1.247839220448618850760726\
838838852844070883136534 -4.806918221932389346705562523044457400769986746645 -\
14.66724711384104594780692605259030414602229895222]
[0 2.890521001958810373589806560427662461469013267521 5.1009346392098534968392\
6535852241697600888687958 -5.324055695526355025822004314533417988312952072004]

kash> Lambda2 := LatLLL(Lambda1);
Basis:
[1 -8.898972742509405137989481976338606158885263693444 10.69685215937833774951\
8381293192142719370918398672 -8.3929181756421331436558159231442716924315445312\
98]
[1 0.134261593289898000224869399556756476905813297935 -23.31066585811776801515\
6201504078952000030073676501 79.4175118311570330970156479070963376256057136484\
54]
[1.414213562373095048801688724209698078569671875377 5.490479907567903997165793\
011467947079779898762665 8.919313203002418011662432704182923884009727755229 -5\
0.221971804833625921569542397881159401643756122457]
[0 2.890521001958810373589806560427662461469013267521 36.896665660756767606327\
137523226704052168032822311 64.55872298086831805701859966901857310820073028551\
9]

```

NAME	LatShortestElt
PURPOSE	Finds shortest elements in a lattice.
SYNTAX	<pre> L := LatShortestElt(Lambda); L := LatShortestElt(Lambda,m); L := LatShortestElt(Lambda,"all"); list of lattice elements L lattice Lambda integer m </pre>
DESCRIPTION	<p>Please refer to “KASH – A User’s Guide” for a general description of the LatShortestElt function.</p> <p>If m is specified, the LatShortestElt function tries to find m shortest elements. If the second argument is "all", the LatShortestElt function returns all shortest vectors.</p>
EXAMPLE	See Lat for an example.

NAME	LatSuccMins
PURPOSE	Computes the successive minima of a lattice.
SYNTAX	<pre>L := LatSuccMins(Lambda);</pre> <pre>list L</pre> <pre>lattice Lambda</pre>
DESCRIPTION	The LatSuccMins function computes the successive minima of a lattice and a set of corresponding lattice elements.
EXAMPLE	

```
kash> o := Order(Poly(Zx, [1, 13, 13, 13, 169]));;
kash> Lambda := Lat(o);;
kash> LatSuccMins(Lambda);
[ [ -1 0 0 0 ]
  , 4, [ -3 -1 0 0 ]
  , 117.710223325529930087190601154781784396363864536319, [ 0 -11 -1 0 ]
  , 2098.727873763335276613028969537316945979798627539881, [ -13 -3 -12 -1 ]
  , 13067.65742584099018040789043545504811128068694483 ]
```

Lcm

NAME Lcm

PURPOSE Returns least common multiple of list of arguments.

SYNTAX lcm := Lcm(L);
 lcm := Lcm(a, b);

integer or polynomial or ideal	lcm
list of integers of polynomials or ideals	L
integer or polynomial or ideal	a
integer or polynomial or ideal	b

DESCRIPTION Returns the least common multiple of list of arguments or simply the least common multiple of two arguments. Supported are integers, ideals and polynomials over S where S is a field or \mathbb{Z} .

SEE ALSO [IntLcm](#), [Ideallcm](#),

EXAMPLE Lcm of integers:

```
kash> Lcm([18, 12 ,4]);
```

```
36
```

```
kash> Lcm(18, 12);
```

```
36
```

```
kash> Lcm([18, 12]);
```

```
36
```

NAME	Li
PURPOSE	Returns the value of the logarithmic integral.
SYNTAX	<pre>a := Li(b);</pre> <div style="margin-left: 100px;"> <pre>real a</pre> <pre>real or integer b</pre> </div>
DESCRIPTION	This function computes an approximation of the logarithmic integral. It gives a good idea of the number of prime ideals of norm less than or equal to the argument.
EXAMPLE	

```
kash> Li(2);
1.045163780117492784844588889194613136522615578151
kash> Li(3);
2.163588594667191972876922367347721366542116212451
kash> Li(7.5);
5.009500143078375327886075408809948575232609368221
kash> Li(10000);
1246.137215899388459692771107529059792486534653513971
kash> l := []; p := 2;
[ ]
2
kash> while p <= 10000 do Add(l, p); p := NextPrime(p); od;
kash> Length(l);
1229
kash> o := OrderMaximal(x^2+164);;
kash> l := []; p := 2;
[ ]
2
kash> while p <= 10000 do
> p1 := Filtered(Factor(p*o), x->IdealNorm(x[1]) <= 10000);
> Append(l, p1);
> p := NextPrime(p);
> od;
kash> Length(l);
1209
```

NAME	ListApplyListAdd
PURPOSE	Computes a linear combination.
SYNTAX	$a := \text{ListApplyListAdd}(L, S);$ <p> element a lists L, S of elements </p>
DESCRIPTION	Given lists $L := [a_1, \dots, a_n]$ and $S := [\lambda_1, \dots, \lambda_n]$ this function returns $a := \sum_{i=1}^n \lambda_i a_i$.
SEE ALSO	ListApplyMatAdd ,
EXAMPLE	

```

kash> L := [1, 2, 3];
[ 1, 2, 3 ]
kash> S := [1, 3, 4];
[ 1, 3, 4 ]
kash> ListApplyListAdd(L, S);
19

```

NAME	ListApplyMatAdd
PURPOSE	Computes linear combinations.
SYNTAX	$S := \text{ListApplyMatAdd}(L, M);$ <p> lists S, L of elements matrix M for linear combinations </p>
DESCRIPTION	<p>Given a list $L := [a_1, \dots, a_n]$ and an $n \times m$ matrix $M = (\lambda_{i,j})_{i,j}$ this function returns a list $S := [\sum_{i=1}^n \lambda_{i,j} a_i]_j$.</p>
SEE ALSO	ListApplyListAdd ,
EXAMPLE	

```

kash> M := MatTrans(Z, [ [ 1, 0 ], [1, 1], [0, 2] ]);
[1 1 0]
[0 1 2]
kash> L := [ 1, 2 ];
[ 1, 2 ]
kash> ListApplyMatAdd(L, M);
[ 1, 3, 4 ]

```

NAME `ListSplit`

PURPOSE Splits a list into equal parts. All parts but the last have equal length as specified by the parameter $n \geq 1$.

SYNTAX `S := ListSplit(L, n);`

list S of lists of length $\leq n$

list L to be split

integer n block length

SEE ALSO [IntDivisors](#),

EXAMPLE

```
kash> L := [1, 2, 3, 4, 5];
[ 1, 2, 3, 4, 5 ]
kash> ListSplit(L, 1);
[ [ 1 ], [ 2 ], [ 3 ], [ 4 ], [ 5 ] ]
kash> ListSplit(L, 2);
[ [ 1, 2 ], [ 3, 4 ], [ 5 ] ]
kash> ListSplit(L, 101);
[ [ 1, 2, 3, 4, 5 ] ]
```


NAME	Log
PURPOSE	Returns the principal value of the natural logarithm.
SYNTAX	$y := \text{Log}(x);$ $y := \text{Log}(b, x);$ <div style="display: flex; justify-content: space-between;"> complex or real y </div> <div style="display: flex; justify-content: space-between;"> complex or real or rational or integer x </div> <div style="display: flex; justify-content: space-between;"> complex or real or rational or integer b </div>
DESCRIPTION	<p>Given $x > 0$ in \mathbb{Z}, \mathbb{Q} or \mathbb{R} the function returns the natural logarithm of x. For $x = x e^{i\phi} \in \mathbb{C} \setminus \mathbb{R}^{\geq 0}$, $\phi \in (-\pi, \pi]$, the function returns $y = \log x + i \arg x$. The computation is done in the current precision of the real (complex) field. If two arguments are given the first is used as the base of the logarithm.</p>
SEE ALSO	Exp ,

EXAMPLE

```

kash> Log(2);
0.6931471805599453094172321214581765680755001343602553
kash> Log(-2);
0.6931471805599453094172321214581765680755001343605 + 3.1415926535897932384626\
43383279502884197169399375*i
kash> i := Comp(0, 1);
1*i
kash> Log(i);
1.570796326794896619231321691639751442098584699688*i

```

NAME	Mat
PURPOSE	Creates a matrix.
SYNTAX	$M := \text{Mat}([S,] L);$ <div> matrix M ring S list L </div>
DESCRIPTION	Creates the matrix M over the coefficient ring S. L is a list of row vectors. If only a list L is given the function tries to extract a common coefficient ring from the entries.
SEE ALSO	IsMat ,
EXAMPLE	Create the integer matrix $\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$.

```
kash> Mat(Z, [[1,2],[3,4]]);
[1 2]
[3 4]
```

NAME	MatCharPoly
PURPOSE	Returns the characteristic polynomial of a matrix.
SYNTAX	<pre>f := MatCharPoly(M);</pre> <div style="margin-left: 100px;"> matrix M Polynomial f </div>
SEE ALSO	MatMinPoly ,

EXAMPLE Compute the characteristic polynomial of

$$\begin{pmatrix} 0 & 1 & 0 & -1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ -1 & 0 & 1 & 0 \end{pmatrix} \in \mathbb{Z}^{4 \times 4}.$$

```
kash> M := Mat(Z, [[0,1,0,-1],[1,0,1,0],[0,1,0,1],[-1,0,1,0]]);
[ 0  1  0 -1]
[ 1  0  1  0]
[ 0  1  0  1]
[-1  0  1  0]
kash> MatCharPoly(M);
x^4 - 4*x^2 + 4
```

NAME MatCoef

PURPOSE Returns the coefficient ring of a matrix.

SYNTAX `S := MatCoef(M);`

 ring S
 matrix M

SEE ALSO [Mat](#),

EXAMPLE

NAME	MatCols
PURPOSE	Returns the number of columns of a matrix.
SYNTAX	<code>n := MatCols(M);</code> integer n matrix M
SEE ALSO	MatRows ,
EXAMPLE	

```
kash> M := Mat(Z, [[67,4,98,1], [-1,5,3,4], [6,-1,7,8]]);  
[67  4 98  1]  
[-1  5  3  4]  
[ 6 -1  7  8]  
kash> MatCols(M);  
4
```

NAME	MatDet
PURPOSE	Computes the determinant of a square matrix.
SYNTAX	<pre>d := MatDet(M); d := MatDet(M, bound);</pre> <p>element of the ring the matrix entries come from d matrix M</p>
DESCRIPTION	If bound is given and M is over an absolute numberfield we use an modular algorithm. If bound ≤ 0 we will compute a suitable bound.

EXAMPLE Compute the determinant of the integer matrix $\begin{pmatrix} -1 & 5 \\ 6 & -1 \end{pmatrix}$.

```
kash> M := Mat(Z, [[-1,5],[6,-1]]);
[-1  5]
[ 6 -1]
kash> MatDet(M);
-29
```

NAME	MatDiag
PURPOSE	missing shortdoc
SYNTAX	$M := \text{MatDiag}(S, L);$ matrix M ring S list L diagonal entries
DESCRIPTION	Creates the diagonal matrix M over the coefficient ring S. L is a list containing the diagonal entries.

EXAMPLE Create the diagonal matrix

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 3 & 0 & 0 \\ 0 & 0 & 0 & 4 & 0 \\ 0 & 0 & 0 & 0 & 5 \end{pmatrix} \in \mathbb{Z}^{5 \times 5}.$$

```
kash> MatDiag(Z, [1, 2, 3, 4, 5]);
[1 0 0 0 0]
[0 2 0 0 0]
[0 0 3 0 0]
[0 0 0 4 0]
[0 0 0 0 5]
```

NAME	MatEchelon
PURPOSE	Returns the rank and an upper row echelon form of a matrix together with a transformation matrix. If the echelon form cannot be computed over the ring, it is computed over its field of fractions.
SYNTAX	<pre>L := MatEchelon(M); list L matrix M</pre>

NAME	MatElt
PURPOSE	Returns or modifies a matrix entry or row.
SYNTAX	<pre> a := MatElt(M, row, col); a := M[row][col]; M[row][col] := a; r := M[row]; M[row] := L; </pre> <p> element of the ring the matrix entries come from a matrix M small integer row small integer col an object of the type "KANT matrow" r either a list, a "KANT matrix", or a "KANT matrow" L </p>
DESCRIPTION	<p>The first two syntax variants are equivalent and return the (row, col)-th entry of the matrix M.</p> <p>The third variant sets a single entry of the matrix. The object must be convertible to an element of the ring over which the matrix is defined.</p> <p>The fourth syntax retrieves a matrix row. This is an object of the type "KANT matrow". This object can be converted to a proper matrix (with only one row) with a call of <code>IsMat</code>. It can also be used as a row for another matrix.</p> <p>The fifth syntax sets a row of the matrix. For this purpose the matrix is converted into a list of rows (of the type "KANT matrix"). Then the row-th entry of this list is either replaced by or set to the right hand side of the assignment. Now M is still a list a should be converted to a matrix with a call of <code>IsMat</code>. <code>IsMat</code> also determines a ring all entries of the matrix belong to. So by setting a new row or modifying an existing one the matrix might change its ring.</p>

EXAMPLE Get certain entries of the matrix

$$\begin{pmatrix} 3 & 7 & 10 & 2 \\ 90 & 101 & 87 & 2 \\ 12 & 34 & 2 & 0 \\ 20 & 25 & 40 & 1 \end{pmatrix}$$

and modify them.

```

kash> M := Mat(Z, [[3,7,10,2],[90,101,87,2],[12,34,2,0],[20,25,40,1]]);
[ 3  7 10  2]
[ 90 101 87  2]

```

```
[ 12  34   2   0]
[ 20  25  40   1]
kash> MatElt(M,1,1);
3
kash> M[2][2];
101
kash> r:=M[4];
[20 25 40  1]
kash> TYPE(r);
"KANT matrow"
kash> IsMat(r);
true
kash> r;
[20 25 40  1]
kash> TYPE(r);
"KANT matrix"
kash> M[4][4]:=22;;M;
[  3   7  10   2]
[ 90 101  87   2]
[ 12  34   2   0]
[ 20  25  40  22]
kash> M[3]:=[-1,2,-1,4];;M;
[ [ 3   7  10   2], [ 90 101  87   2], [ -1, 2, -1, 4 ], [20 25 40 22] ]
kash> IsMat(M);
true
kash> M;
[  3   7  10   2]
[ 90 101  87   2]
[ -1   2  -1   4]
[ 20  25  40  22]
```

NAME	MatHermiteColLower
PURPOSE	missing shortdoc
SYNTAX	$H := \text{MatHermiteColLower}(M);$ matrix H matrix M
DESCRIPTION	Computes the lower column Hermite normal form of the integer matrix M.

EXAMPLE Compute the lower column Hermite normal form of

$$\begin{pmatrix} 4 & 6 & 2 \\ 3 & 9 & 12 \end{pmatrix}.$$

```
kash> M := Mat(Z, [[4,6,2],[3,9,12]]);
[ 4  6  2]
[ 3  9 12]
kash> MatHermiteColLower(M);
[2 0 0]
[0 3 0]
```

NAME	MatHermiteColLowerTrans
PURPOSE	Computes the lower column Hermite normal form together with a transformation matrix.
SYNTAX	<pre>L := MatHermiteColLowerTrans(M);</pre> <p>list L</p> <p>matrix M</p>
DESCRIPTION	The <code>MatHermiteColLowerTrans</code> routine computes the lower column Hermite normal form H of the integer matrix M . Additionally it computes the rank of M and a transformation matrix T such that $H = M * T$. The result is a list L which contains the rank, the Hermite normal form H and the transformation matrix T .

EXAMPLE Compute the lower column Hermite normal form of

$$\begin{pmatrix} 3 & 7 & 10 \\ 20 & 25 & 40 \end{pmatrix}.$$

```
kash> M := Mat(Z, [[3,7,10],[20,25,40]]);
[ 3  7 10]
[20 25 40]
kash> L := MatHermiteColLowerTrans(M);
kash> L[1];
2
kash> L[2];
[1 0 0]
[0 5 0]
kash> L[3];
[ 1  1  6]
[ 4  1 16]
[-3 -1 -13]
kash> M*L[3];
[1 0 0]
[0 5 0]
```

NAME	MatHermiteColModUpper
PURPOSE	Computes the modular upper column Hermite normal form of an integer matrix.
SYNTAX	$H := \text{MatHermiteColModUpper}(k, M);$ <div style="margin-left: 100px;"> $\text{matrix } H$ $\text{integer } k$ $\text{matrix } M$ </div>
DESCRIPTION	<p>Let $M \in \mathbb{Z}^{m \times n}$. The MatHermiteColModUpper routine computes the upper column Hermite normal form of the matrix</p>

$$(M \mid k \cdot I | m) \in \mathbb{Z}^{m \times (n+m)}$$

and returns the rightmost n columns of the Hermite normal form.

EXAMPLE Compute the upper modular column Hermite normal form of

$$\begin{pmatrix} 3 & 7 & 10 \\ 20 & 25 & 40 \\ 47 & 61 & 90 \end{pmatrix}$$

with respect to the modulus 5.

```
kash> M := Mat(Z, [[3,7,10],[20,25,40],[47,61,90]]);
[ 3  7 10]
[20 25 40]
[47 61 90]
kash> MatHermiteColModUpper(5,M);
[1 0 0]
[0 5 0]
[0 0 1]
```

NAME	MatHermiteColUpper
PURPOSE	missing shortdoc
SYNTAX	<pre>H := MatHermiteColUpper(M);</pre> <pre>matrix H</pre> <pre>matrix M</pre>
DESCRIPTION	Computes the upper column Hermite normal form of the integer matrix M.

EXAMPLE Compute the upper column Hermite normal form of

$$\begin{pmatrix} 4 & 6 & 2 \\ 3 & 9 & 12 \end{pmatrix}.$$

```
kash> M := Mat(Z, [[4,6,2],[3,9,12]]);
[ 4  6  2]
[ 3  9 12]
kash> MatHermiteColUpper(M);
[0 2 0]
[0 0 3]
```

NAME	MatHermiteColUpperTrans
PURPOSE	Computes the upper column Hermite normal form together with a transformation matrix.
SYNTAX	<pre>L := MatHermiteColUpperTrans(M [, "int"]);</pre> <p>list L matrix M</p>
DESCRIPTION	<p>The MatHermiteColUpperTrans routine computes the upper column Hermite normal form H of the integer matrix M. Additionally it computes the rank of M and a transformation matrix T such that $H = M * T$. The result is a list L which contains the rank, the Hermite normal form H and the transformation matrix T.</p> <p>If the matrix is defined over a local field and has integral coefficients, the Hermite Normal Form over the respective valuation ring is returned.</p>

EXAMPLE Compute the upper column Hermite normal form of

$$\begin{pmatrix} 3 & 7 & 10 \\ 20 & 25 & 40 \end{pmatrix}.$$

```
kash> M := Mat(Z, [[3,7,10],[20,25,40]]);
[ 3  7 10]
[20 25 40]
kash> L := MatHermiteColUpperTrans(M);
kash> L[1];
2
kash> L[2];
[0 1 0]
[0 0 5]
kash> L[3];
[ -6   1  -5]
[-16   4 -15]
[ 13  -3  12]
kash> M*L[3];
[0 1 0]
[0 0 5]
```

NAME MatHermiteRowMod

PURPOSE Computes the modular row Hermite normal form of an integer matrix.

SYNTAX H := MatHermiteRowMod(k, M);

matrix H

integer k

matrix M

DESCRIPTION Let $M \in \mathbb{Z}^{m \times n}$. The MatHermiteRowMod routine computes the row Hermite normal form of the matrix

$$\begin{pmatrix} M \\ k \cdot I_n \end{pmatrix} \in \mathbb{Z}^{(m+n) \times n}$$

and returns the upper m rows of the Hermite Normal form.

EXAMPLE Compute the row Hermite normal form of

$$\begin{pmatrix} 3 & 7 & 10 \\ 20 & 25 & 40 \end{pmatrix}$$

with respect to the modulus 5.

```
kash> M := Mat(Z, [[3,7,10],[20,25,40]]);
```

```
[ 3  7 10]
```

```
[20 25 40]
```

```
kash> MatHermiteRowMod(5,M);
```

```
[1 4 0]
```

```
[0 5 0]
```

```
[0 0 5]
```


NAME	MatId
PURPOSE	Returns the identity matrix of given dimension.
SYNTAX	$I := \text{MatId}(S, n);$ <div> <div>matrix</div> <div>ring</div> <div>small integer</div> <div>I</div> <div>S</div> <div>n</div> </div>
DESCRIPTION	Returns the identity matrix $I n$ of dimension n over the (unitary) ring S .
EXAMPLE	Create $I 3$ over $\mathbb{Z}[\sqrt[4]{-1}]$.

```
kash> O := Order(Z,4,-1);
Generating polynomial: x^4 + 1
```

```
kash> MatId(O,3);
[1 0 0]
[0 1 0]
[0 0 1]
```

NAME	MatIndex
PURPOSE	Computes the determinant of a row normal form of an integral matrix.
SYNTAX	<pre>n := MatIndex(m);</pre> <p>integer n index matrix m over \mathbb{Z}</p>
DESCRIPTION	Suppose we have \mathbb{Z} modules $M 1 \subset M 2$ given via a basis of $M 2$ and a set of elements generating $M 1$. Furthermore let m be the matrix which rows contain the aforementioned generators. MatIndex (m) will compute the index of $M 1$ in $M 2$. If the index is not finite, a 0 will be returned.

EXAMPLE

```
kash> m := RandomMatrix(Z, 3);
[ 19  11   5]
[ -6 -18  -6]
[ 17  18   5]
kash> MatIndex(m);
540
kash> m := m{[1..2]}; IsMat(m);
[ 19  11   5]
[ -6 -18  -6]
true
kash> MatIndex(m);
0
kash> m := Concatenation(m, RandomMatrix(Z, 3)); IsMat(m);
[ [19 11  5], [-6 -18 -6], [-16 -1 -16], [-4 -3 13], [17 11 -9] ]
true
kash> MatIndex(m);
1
```

NAME	MatInv
PURPOSE	Returns the inverse matrix (if possible).
SYNTAX	$A := \text{MatInv}(M);$ <div style="margin-left: 100px;"> <code>matrix</code> <code>A</code> <code>matrix</code> <code>M</code> </div>
DESCRIPTION	Returns the inverse matrix <code>A</code> of <code>M</code> . Returns an error if <code>M</code> is not invertible.

EXAMPLE Compute the inverse of the integer matrix $\begin{pmatrix} 1 & 2 \\ 1 & 1 \end{pmatrix}$.

```
kash> M := Mat(Z, [[1,2],[1,1]]);
[1 2]
[1 1]
kash> MatInv(M);
[-1  2]
[ 1 -1]
```

NAME `MatKernel`

PURPOSE Returns a basis for the kernel of the given matrix.

SYNTAX `K := MatKernel(M); K := MatKernel(M, d);`

`matrix` `K`
`matrix` `M`
`integer` `d`

DESCRIPTION Let M be a $m \times n$ -matrix. The `MatKernel` routine computes a basis K for the linear subspace

$$\{x = (x|1, \dots, x|m) : x \cdot M = (0, \dots, 0)\}.$$

.

If a second parameter d is used, the kernel over $\mathbb{Z}/d\mathbb{Z}$ will be computed. The used algorithm is based on [BN96].

EXAMPLE Compute the kernel of the integer matrix $\begin{pmatrix} 2 & 3 & 4 \\ 3 & 4 & 5 \\ 4 & 5 & 6 \\ 6 & 7 & 8 \end{pmatrix}$.

```
kash> M := Mat(Z, [[2,3,4],[3,4,5],[4,5,6],[6,7,8]]);
[2 3 4]
[3 4 5]
[4 5 6]
[6 7 8]
kash> MatKernel(M);
[ 1  0 -2  1]
[ 0  2 -3  1]
```

Compute the kernel modulo 3

```
kash> MatKernel(M, 3);
[0 2 0 1]
[1 1 1 0]
```

Compute kernel

```
kash> o := Order(Z, 2,3);;
kash> a := Elt(o, [1,2/19]);;
kash> one := Elt(o, [1,0]);;
kash> m := Mat(o, [[a,2*a,3*a^2],[a^2,2*a^2,3*a^3],[one,2*one,3*a]]);
[[19, 2] / 19 [38, 4] / 19 [1119, 228] / 361]
[[373, 76] / 361 [746, 152] / 361 [22629, 6570] / 6859]
[1 2 [57, 6] / 19]
kash> k := MatKernel(m);
[1 0 [-19, -2] / 19]
[0 1 [-373, -76] / 361]
kash> k*m;
[0 0 0]
[0 0 0]
```

NAME	MatLLL
PURPOSE	Performs an LLL reduction on the columns of an integer or real matrix.
SYNTAX	<pre> L := MatLLL(M [,r]); L := MatLLL(M,"short" "long" [,r]); list of A, T L matrix M LLL constant (default 0.75) r </pre>
DESCRIPTION	<p>Performs an LLL reduction on the columns of an integer or real matrix M. The result is a list containing the reduced matrix A and the unimodular transformation matrix T such that $M = A * T$. The <code>MatLLL</code> function requires that the columns of M are linearly independent.</p> <p>If M is an integer matrix the <code>MatLLL</code> function normally omits any floating-point arithmetic. When the "SHORT" option is specified the <code>MatLLL</code> function uses floating-point arithmetic basing on machine doubles.</p> <p>If M is a real matrix the <code>MatLLL</code> function normally makes use of the long floating-point arithmetic of KASH. When the "SHORT" option is specified the <code>MatLLL</code> function uses floating-point arithmetic basing on machine doubles</p>
EXAMPLE	<pre> kash> M := Mat(Z, [[234, 469], [-6546, -13126], > [-6312, -12657]]); [234 469] [-6546 -13126] [-6312 -12657] kash> L := MatLLL(M); [[1 42] [-34 -18] [-33 24], [-2 385] [1 -192]] kash> L[1]; [1 42] [-34 -18] [-33 24] kash> M*L[2]; [1 42] [-34 -18] [-33 24] </pre>

NAME	MatMLLL
PURPOSE	Performs an MLLL reduction on the columns of an integer or real matrix.
SYNTAX	$L := \text{MatMLLL}(M);$ <code>list L</code> <code>matrix M</code>
DESCRIPTION	<p>Performs a MLLL reduction on the columns of an integer or real matrix M. Let $M \in \mathbb{R}^{m \times (n+1)}$ and let $a 1, \dots, a n+1 \in \mathbb{R}^m$ be the columns of M. The <code>MatMLLL</code> function expects that $a 1, \dots, a n$ are linearly independent and that $a n+1$ has a non-zero entry. It computes columns $b 1, \dots, b n+1 \in \mathbb{R}^m$ of B such that</p> $\mathbb{Z} \cdot a 1 + \dots + \mathbb{Z} \cdot a n+1 = \mathbb{Z} \cdot b 1 + \dots + \mathbb{Z} \cdot b n+1$ <p>and additionally a transformation matrix $T \in \mathbb{R}^{(n+1) \times (n+1)}$ with $B = M \cdot T$. Furthermore a flag is returned. If it is true, $b 1, \dots, b n+1$ are linearly independent. Otherwise all entries of $b n+1$ are zero. These results are stored in a list containing the reduced matrix B, the unimodular transformation matrix T and the flag.</p>

EXAMPLE MLLL-Reduction of a matrix:

```

kash> M := Mat(Z, [ [ 234, 469, -54411 ],
> [ -6546, -13126, 1531763 ], [ -6312, -12657, 1477352 ] ]);
[ 234 469 -54411]
[ -6546 -13126 1531763]
[ -6312 -12657 1477352]
kash> L := MatMLLL(M);
[ [ 1 1 0]
  [-1 0 0]
  [ 0 1 0], [ -794943 -1327888 4198061]
  [ 427600 714271 -2258136]
  [ 267 446 -1410], true ]
kash> L[1];
[ 1 1 0]
[-1 0 0]
[ 0 1 0]
kash> M*L[2];
[ 1 1 0]
[-1 0 0]
[ 0 1 0]

```

NAME MatMinPoly

PURPOSE Returns the minimal polynomial of a matrix.

SYNTAX `f := MatMinPoly(M);`

 matrix M

 Polynomial f

SEE ALSO [EltMinPoly](#), [MatCharPoly](#),

EXAMPLE Compute the minimal polynomial of

$$\begin{pmatrix} 0 & 1 & 0 & -1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ -1 & 0 & 1 & 0 \end{pmatrix} \in \mathbb{Z}^{4 \times 4}.$$

```
kash> M := Mat(Z, [[0,1,0,-1],[1,0,1,0],[0,1,0,1],[-1,0,1,0]]);
```

```
[ 0  1  0 -1]
```

```
[ 1  0  1  0]
```

```
[ 0  1  0  1]
```

```
[-1  0  1  0]
```

```
kash> MatMinPoly(M);
```

```
x^2 - 2
```

```
kash> M^2;
```

```
[2 0 0 0]
```

```
[0 2 0 0]
```

```
[0 0 2 0]
```

```
[0 0 0 2]
```


NAME	MatMove
PURPOSE	missing shortdoc
SYNTAX	$V := \text{MatMove}(M, r);$ <pre>matrix V matrix M ring r</pre>
DESCRIPTION	The <code>MatMove</code> function casts the matrix <code>M</code> to a matrix over the ring <code>R</code> .
SEE ALSO	PolyMove , EltMove ,
EXAMPLE	Compute the representation of an interger matrix in $\mathbb{Z}/23\mathbb{Z}$.

```
kash> M := Mat(Z, [[0,23,0,45456],[35,33,85,14],[86,79,80,-3],[-1,0,1,0]]);
[  0   23   0 45456]
[ 35   33   85   14]
[ 86   79   80   -3]
[ -1    0    1    0]
kash> F23 := FF (23);
Finite field of size 23
kash> MatMove (M, F23);
[ 0  0  0  8]
[12 10 16 14]
[17 10 11 20]
[22  0  1  0]
```

NAME MatRows

PURPOSE Returns the number of rows of a matrix.

SYNTAX `n := MatRows(M);`

 integer n

 matrix M

SEE ALSO [MatCols](#),

EXAMPLE

```
kash> M := Mat(Z, [[67,4,98,1],[-1,5,3,4],[6,-1,7,8]]);
[67  4 98  1]
[-1  5  3  4]
[ 6 -1  7  8]
kash> MatRows(M);
3
```

NAME	MatSmith
PURPOSE	missing shortdoc
SYNTAX	$L := \text{MatSmith}(M);$ list L matrix M
DESCRIPTION	Computes the Smith normal form and the rank of the integer matrix M. The result is a list L which contains the rank and the Smith normal form S of M.
EXAMPLE	Compute the Smith normal form of

$$\begin{pmatrix} 4 & 6 & 2 \\ 3 & 9 & 12 \end{pmatrix}.$$

```

kash> M := Mat(Z, [[4,6,2],[3,9,12]]);
[ 4  6  2]
[ 3  9 12]
kash> L := MatSmith(M);
kash> rank := L[1];
2
kash> S := L[2];
[1 0 0]
[0 6 0]

```

NAME	MatSmithTrans
PURPOSE	Computes the Smith normal form together with transformation matrices.
SYNTAX	<pre>L := MatSmithTrans(M);</pre> <pre>list L</pre> <pre>matrix M</pre>
DESCRIPTION	<p>The MatSmithTrans routine computes the Smith normal form S of the integer matrix M. Additionally it computes the rank of M and transformation matrices A and B such that $S = A * M * B$. The result is a list L which contains the rank, the Smith normal form S and the transformation matrices A and B.</p>

EXAMPLE Compute the upper column Smith normal form of

$$\begin{pmatrix} 3 & 7 & 10 \\ 20 & 25 & 40 \\ 17 & 6 & 9 \end{pmatrix}.$$

```
kash> M := Mat(Z, [[3,7,10],[20,25,40],[17,6,9]]);
```

```
[ 3  7 10]
```

```
[20 25 40]
```

```
[17  6  9]
```

```
kash> L := MatSmithTrans(M);;
```

```
kash> rank := L[1];
```

```
3
```

```
kash> S := L[2];
```

```
[ 1  0  0]
```

```
[ 0  1  0]
```

```
[ 0  0 405]
```

```
kash> A := L[3];
```

```
[ 1  0  0]
```

```
[-56 -11  4]
```

```
[-70 -14  5]
```

```
kash> B := L[4];
```

```
[ 1  1 -318]
```

```
[ 4  1 -308]
```

```
[ -3 -1 311]
```

```
kash> A*M*B;
```

```
[ 1  0  0]
```

```
[ 0  1  0]
```

```
[ 0  0 405]
```

NAME	MatSolve								
PURPOSE	Solves a linear equation.								
SYNTAX	$L := \text{MatSolve}(A, b);$ $L := \text{MatSolve}(A, b, N);$								
	<table> <tr> <td>list</td><td>L</td></tr> <tr> <td>matrix</td><td>A</td></tr> <tr> <td>matrix</td><td>b</td></tr> <tr> <td>integer</td><td>N a module</td></tr> </table>	list	L	matrix	A	matrix	b	integer	N a module
list	L								
matrix	A								
matrix	b								
integer	N a module								
DESCRIPTION	<p>Given matrix A and vector b, the MatSolve function finds the vector x which satisfies the matrix equation $A \cdot x = b$. The MatSolve function returns a list whose first entry is x. The second of L is the null space of A. If $A \cdot x = b$ has no solution, then FALSE is returned. If the module N is given, solutions modulo N are returned.</p>								

EXAMPLE

```

kash> M := Mat(Z, [[1,0],[0,1]]);
[1 0]
[0 1]
kash> b := Mat(Z, [[1],[2]]);
[1]
[2]
kash> MatSolve(M, b);
[ [1]
  [2], Matrix with 2 rows and 0 columns ]
kash> M := Mat(Z, [[1,0,1],[0,1,1]]);
[1 0 1]
[0 1 1]
kash> MatSolve(M, b);
[ [1]
  [2]
  [0], [ 1]
  [ 1]
  [-1] ]

```

NAME	MatSym
PURPOSE	missing shortdoc
SYNTAX	<pre>M := MatSym(S, L); matrix M ring S list L</pre>
DESCRIPTION	Creates the symmetric matrix M over the coefficient ring S.

EXAMPLE Create the symmetric matrix

$$\begin{pmatrix} 1 & 2 & 3 & 4 \\ 2 & 5 & 6 & 7 \\ 3 & 6 & 8 & 9 \\ 4 & 7 & 9 & 10 \end{pmatrix} \in \mathbb{Z}^{4 \times 4}.$$

```
kash> MatSym(Z, [[1,2,3,4], [5,6,7], [8,9], [10]]);
[ 1  2  3  4]
[ 2  5  6  7]
[ 3  6  8  9]
[ 4  7  9 10]
```

NAME	MatSymDiag
PURPOSE	missing shortdoc
SYNTAX	$L := \text{MatSymDiag}(A);$ <div style="margin-left: 100px;">list L matrix A</div>
DESCRIPTION	<p>Let A be a symmetric matrix over \mathbb{Z}, \mathbb{Q} or \mathbb{R}. The <code>MatSymDiag</code> function computes Δ, T and λ such that $\Delta = \lambda T^t A T$ is a diagonal matrix. The diagonal entries of Δ are squarefree integers if the coefficient ring of A is \mathbb{Z} or \mathbb{Q}. When the coefficient ring of A equals \mathbb{R} each diagonal entry of Δ is ± 1 or 0.</p>

EXAMPLE

```

kash> A := Mat(Z, [[2,3,4],[3,1,4],[4,4,0]]);
[2 3 4]
[3 1 4]
[4 4 0]
kash> L := MatSymDiag(A);
kash> Delta := L[1];
[ 7  0  0]
[ 0 -1  0]
[ 0  0 -6]
kash> lambda := L[2];
14/1
kash> T := L[3];
[ 1/2 -3/14 -2/7]
[   0   1/7 -1/7]
[   0    0  1/4]
kash> Delta;
[ 7  0  0]
[ 0 -1  0]
[ 0  0 -6]
kash> lambda * MatTrans(T) * A * T;
[ 7  0  0]
[ 0 -1  0]
[ 0  0 -6]

```

NAME MatToColList

PURPOSE Returns a list of the columns.

SYNTAX L := MatToColList(M);

list L

matrix M

EXAMPLE

```
kash> M := Mat(Z, [[1,2,3], [4,5,6], [7,8,9], [10,11,12]]);  
[ 1  2  3]  
[ 4  5  6]  
[ 7  8  9]  
[10 11 12]  
kash> L := MatToColList(M);  
[ [ 1, 4, 7, 10 ], [ 2, 5, 8, 11 ], [ 3, 6, 9, 12 ] ]
```


NAME MatToRowList

PURPOSE Returns a list of the rows.

SYNTAX L := MatToRowList(M);

list L

matrix M

EXAMPLE

```
kash> M := Mat(Z, [[1,2,3], [4,5,6], [7,8,9], [10,11,12]]);  
[ 1  2  3]  
[ 4  5  6]  
[ 7  8  9]  
[10 11 12]  
kash> L := MatToRowList(M);  
[ [ 1, 2, 3 ], [ 4, 5, 6 ], [ 7, 8, 9 ], [ 10, 11, 12 ] ]
```

The rows can also be accessed through:

```
kash> M[1];  
[1 2 3]  
kash> IsList(M);  
true
```

NAME MatTrace

PURPOSE Computes the trace of a matrix.

SYNTAX `t := MatTrace(M);`

 element of the ring the matrix elements come from t
matrix M

EXAMPLE Compute the trace of the integer matrix $\begin{pmatrix} 1 & 1 \\ 2 & 2 \end{pmatrix}$.

```
kash> M := Mat(Z, [[1,1],[2,2]]);
```

```
[1 1]
```

```
[2 2]
```

```
kash> MatTrace(M);
```

```
3
```

NAME MatTrans

PURPOSE Returns the tranpose of a matrix.

SYNTAX A := MatTrans(M);

 matrix A

 matrix M

DESCRIPTION Returns the transposed matrix A of M.

EXAMPLE Compute the transpose of the integer matrix $\begin{pmatrix} 1 & 2 & 4 \\ 1 & 3 & 9 \end{pmatrix}$.

```
kash> M := Mat(Z, [[1,2,4],[1,3,9]]);
```

```
[1 2 4]
```

```
[1 3 9]
```

```
kash> MatTrans(M);
```

```
[1 1]
```

```
[2 3]
```

```
[4 9]
```

NAME Min

PURPOSE Computes the minimum of an ideal.

SYNTAX `n := Min(a);`

`int | rational | ideal` `n`
`ideal` `a`

DESCRIPTION

EXAMPLE All examples take place in $\mathbb{Q}(\sqrt{5})$:

```
kash> O := OrderMaximal(Order(x^3 - 10*x^2 - 3*x - 2));;
kash> o := Order(O, 3, 3);;
kash> IdealMin(Ideal(5, Elt(O, [1, 1, 0])) / 4);
5/4
kash> IdealMin(Ideal(Elt(O, [3, 1, 0]), Elt(O, [1, 1, 0])));
2
kash> IdealMin(5*o + Elt(o, [3, 1, 0])*o);
<5>
```

NAME	MinPoly
PURPOSE	Computes the minimal polynomial of an algebraic element, a matrix, or an alff element
SYNTAX	<pre> p := MinPoly (a [,PA]); p := MinPoly (a [,0]); p := MinPoly(M); p := MinPoly(a); </pre> <p> polynomial p polynomial algebra PA suborder 0 algebraic element alff order element a matrix M </p>
DESCRIPTION	<p>For an algebraic element a in an Order O and a polynomial algebra PA over another order o or this order, this function computes the minimal polynomial of a over o i.e. the returned polynomial is contained in PA.</p> <p>Given a matrix this function returns its minimal polynomial.</p> <p>Given a alff element this function computes the minimal polynomial</p>
SEE ALSO	EltMinPolyMatMinPolyAlffEltMinPoly ,
EXAMPLE	

```

kash> o := Order(Z,2,2);
Generating polynomial: x^2 - 2

kash> a := Elt(o,[0,1]);
[0, 1]
kash> p := MinPoly(a);
x^2 - 2
kash> M := Mat(o,[[1,a],[1,-a]]);
[1 [0, 1]]
[1 [0, -1]]
kash> p := MinPoly(M);
x^2 + [-1, 1]*x + [0, -2]

```

NAME	MinVec								
PURPOSE	Returns a list of minimal vectors of a real Humbert form in two variables and his minimum								
SYNTAX	<p>$M := \text{MinVec}(\text{HF}, o, n);$</p> <table> <tr> <td>list</td><td>M</td></tr> <tr> <td>Humbert form with $S_i \in \overline{Q}^{2 \times 2}$</td><td>HF=[S1,S2]</td></tr> <tr> <td>maximal order of an quadratic field</td><td>o</td></tr> <tr> <td>integer</td><td>n</td></tr> </table>	list	M	Humbert form with $S_i \in \overline{Q}^{2 \times 2}$	HF=[S1,S2]	maximal order of an quadratic field	o	integer	n
list	M								
Humbert form with $S_i \in \overline{Q}^{2 \times 2}$	HF=[S1,S2]								
maximal order of an quadratic field	o								
integer	n								
DESCRIPTION	<p>Given a real Humbert form in two variables it returns a list of minimal vectors and the minimum of the real Humbert form. If $n=0$, then MinVec uses the default value which is always computed to get minimal vectors. In the other case, n would be</p> <ul style="list-style-type: none"> • the value for computing minimal vectors or • the value to receive an upper bound for the minimum <p>The minimal vectors are conjugated in the same order KANT conjugates an element of the given order o.</p>								
SEE ALSO	EutacticCoef ,								

EXAMPLE

```
kash> o := OrderMaximal(x^2-3);
Generating polynomial: x^2 - 3
Discriminant: 12
```

```
kash> ur := Re(EltCon(Elt(o, [2, 1]))[1][1]);
3.73205080756887729352744634150587236694280525381
kash> urs := Re(EltCon(Elt(o, [2, 1]))[1][2]);
0.26794919243112270647255365849412763305719474619
```

```
kash> S1 := Mat(R, [ [ 1, ur/2 ], [ur/2, ur ] ]);
[1 1.866025403784438646763723170752936183471402626905]
[1.866025403784438646763723170752936183471402626905 3.732050807568877293527446\
34150587236694280525381]
kash> S2 := Mat(R, [ [ 1, urs/2 ], [urs/2, urs] ]);
[1 0.133974596215561353236276829247063816528597373095]
[0.133974596215561353236276829247063816528597373095 0.267949192431122706472553\
```


NAME	Module
PURPOSE	Creates a module over an order.
SYNTAX	<pre> M1 := Module([IL,] M); M1 := Module(EL); module M1 list IL of ideals over a maximal order \mathcal{O} matrix M of algebraic elements over \mathcal{O} list EL of relative algebraic elements of \mathfrak{o} </pre>
DESCRIPTION	<p>Let $\mathfrak{a} 1, \dots, \mathfrak{a} m$ be fractional \mathcal{O}-ideals (the list IL, they are called the coefficient ideals) and $A 1, \dots, A m$ be the columns of the matrix M of dimension n over the quotient field of \mathcal{O}. This function creates a pseudomatrix which is a representation of the \mathcal{O}-module</p> $M 1 = A 1\mathfrak{a} 1 + \dots + A m\mathfrak{a} m.$ <p>If the list of coefficient ideals (IL) is omitted, the module with trivial coefficient ideals ($1\mathcal{O}$) will be created.</p> <p>If Module is invoked with a list of relative algebraic elements, the representations of those in a relative order \mathfrak{o} as column vectors over \mathcal{O}, the coefficient order of \mathfrak{o}. Again the ideals are assumed to be trivial.</p>
SEE ALSO	ModuleIdeals , ModuleMatrix , ModuleOrder ,

EXAMPLE

```

kash> O:=OrderMaximal(Poly(Zx,[1, 5, -6, -53, 3, 206, 244]));;
kash> IL:=List(Factor(2*O),f->f[1]);;
kash> List(IL,IdealGenerators);
[ [ 2, [0, 2, 0, 3, 0, 2] ], [ 2, [0, 0, 3, 0, 1, 2] ],
  [ 2, [3, 3, 1, 2, 0, 0] ] ]
kash> EL1:=List(Factor(5*O),f->IdealGen(f[1],2));
[ [12, 17, 15, 7, 9, 5], [19, 18, 15, 1, 10, 24], [4, 22, 4, 10, 23, 4] ]
kash> EL2:=List(Factor(13*O),f->IdealGen(f[1],2));
[ [10, 8, 0, 1, 0, 0], [1, 12, 5, 1, 0, 0] ]
kash> EL2[3]:=IdealGen(Factor(7*O)[1][1],2);
[1, 1, 0, 0, 0, 0]
kash> EL3:=List(Factor(3*O),f->IdealGen(f[1],2));
[ [1, 1, 0, 0, 0, 0], [1, 0, 1, 0, 0, 0] ]
kash> EL3[3]:=IdealGen(Factor(7*O)[2][1],2);

```



```
[3, 1, 0, 0, 0, 0]
kash> M:=Module(IL,Mat(0,[EL1,EL2,EL3]));
{<2, [0, 2, 0, 3, 0, 2]><2, [0, 0, 3, 0, 1, 2]><2, [3, 3, 1, 2, 0, 0]>
[[12, 17, 15, 7, 9, 5] [19, 18, 15, 1, 10, 24] [4, 22, 4, 10, 23, 4]]
[[10, 8, 0, 1, 0, 0] [1, 12, 5, 1, 0, 0] [1, 1, 0, 0, 0, 0]]
[[1, 1, 0, 0, 0, 0] [1, 0, 1, 0, 0, 0] [3, 1, 0, 0, 0, 0]]
}
```

NAME	ModuleDen
PURPOSE	Returns the denominator of the given module.
SYNTAX	<pre>den := ModuleDen(M); integer den module M</pre>
DESCRIPTION	The denominator of a module M is defined as the smallest natural number d that the module dM is an integral module. An integral module over the Dedekind ring R is a subset of R^n where n is the degree of the module.
EXAMPLE	<pre>kash> O := OrderMaximal(Poly(Zx,[1, 5, -6, -53, 3, 206, 244]));; kash> Ids:=[Ideal(5, Elt(0,[-27, 1, -2, 1, -3, 15])), > Ideal(1, Elt(0,[-18, 0, -3, 3, -1, 0]/5))]; [<5, [-27, 1, -2, 1, -3, 15]>, <1, [-18, 0, -3, 3, -1, 0] / 5>] kash> M := Module(Ids,Mat(0,[[1,2],[2,1]])/15); {<5, [-27, 1, -2, 1, -3, 15]><1, [-18, 0, -3, 3, -1, 0] / 5> [1 / 15 2 / 15] [2 / 15 1 / 15] } kash> ModuleDen(M); 75</pre>

NAME	ModuleDet
PURPOSE	Returns the determinant of the given module.
SYNTAX	<pre>det := ModuleDet(M) module M ideal det</pre>
DESCRIPTION	<p>If the module is represented by a square pseudomatrix, the product of the coefficient ideals multiplied by the determinant of the matrix is computed. Otherwise there are more columns than rows in the representation. In this case the gcd of the determinants of all n-subpseudomatrices of the pseudomatrix is computed. For further information have a look at [Hop98].</p> <p>If there are many more columns than rows it would be impractical to compute all determinants (because there are exponentially many of them) so this function considers just a few (set <code>PRINTLEVEL(ORDER_MODULE_DET,1)</code> to see how many) determinants and gets a reasonable multiple of the degree minor gcd.</p>
SEE ALSO	ModuleModul ,
EXAMPLE	

```
kash> O := OrderMaximal(Poly(Zx,[1, 5, -6, -53, 3, 206, 244]));;
kash> Ids:=[ Ideal(5, Elt(0,[-27, 1, -2, 1, -3, 15])),
> Ideal(1, Elt(0,[-18, 0, -3, 3, -1, 0]/5)) ];
[ <5, [-27, 1, -2, 1, -3, 15]>, <1, [-18, 0, -3, 3, -1, 0] / 5> ]
kash> M := Module(Ids,Mat(0,[[1,2],[2,1]]/15);
{<5, [-27, 1, -2, 1, -3, 15]><1, [-18, 0, -3, 3, -1, 0] / 5>
[1 / 15 2 / 15]
[2 / 15 1 / 15]
}

kash> ModuleDet(M);
<
[-5  0  0 -1 -4  0]
[ 0 -5  0 -1 -4  0]
[ 0  0 -5 -2 -1  0]
[ 0  0  0 -1  0  0]
[ 0  0  0  0 -1  0]
[ 0  0  0  0  0 -5]
/375>
```

NAME	ModuleDual
PURPOSE	Computes the dual module of the given module.
SYNTAX	$M2 := \text{ModuleDual}(M1);$ $\text{modules } M1, M2$
DESCRIPTION	The dual module is defined as $M 2 = \{x \in Q(\mathcal{O})M 1; \text{Tr}(xM 1) \subseteq \mathcal{O}\}$. The module $M 1$ must be of full rank. For details see [Hop98] .
SEE ALSO	ModuleIntersection ,
EXAMPLE	

```

kash> O:=OrderMaximal(Poly(Zx,[1, 5, -6, -53, 3, 206, 244]));;
kash> IL:=List(Factor(2*O),f->f[1]);;
kash> List(IL,IdealGenerators);;
kash> EL1:=List(Factor(5*O),f->IdealGen(f[1],2));;
kash> EL2:=List(Factor(13*O),f->IdealGen(f[1],2));;
kash> EL2[3]:=IdealGen(Factor(7*O)[1][1],2);;
kash> EL3:=List(Factor(3*O),f->IdealGen(f[1],2));;
kash> EL3[3]:=IdealGen(Factor(7*O)[2][1],2);;
kash> M:=Module(IL,Mat(O,[EL1,EL2,EL3]));
{<2, [0, 2, 0, 3, 0, 2]><2, [0, 0, 3, 0, 1, 2]><2, [3, 3, 1, 2, 0, 0]>
[[12, 17, 15, 7, 9, 5] [19, 18, 15, 1, 10, 24] [4, 22, 4, 10, 23, 4]]
[[10, 8, 0, 1, 0, 0] [1, 12, 5, 1, 0, 0] [1, 1, 0, 0, 0, 0]]
[[1, 1, 0, 0, 0, 0] [1, 0, 1, 0, 0, 0] [3, 1, 0, 0, 0, 0]]
}

kash> ModuleDual(M);
{<1, [688333593256040823590763075194430223058992998640350230425, 4157727723404\
9412961809251677, 688333593256040823590763075213190812482229202013839062953, 4\
005210759058806859873928718, 88005633548175729093569707143, 3] / 2623611238838\
6371287536615054><1 / 2><1, [0, 3, 3, 3, 0, 2] / 2>
[1 0 0]
[[10218246369603710638941628710, 0, -1, 0, 0, -1] / 2 1 0]
[[-35538667259348990574164865443, 10218246369603710638941616121, 1021824636960\
3710638941640551, -10218246369603710638941625039, -204364927392074212778832286\
83, -66089] / 4 [-2, 1, 1, -1, -2, 0] / 2 1]
}
```

NAME	ModuleId
PURPOSE	Returns the trivial module over an order with a certain degree.
SYNTAX	<pre>m := ModuleId(o, n);</pre> <div> Module m order o positive integer n the degree </div>
DESCRIPTION	Creates the trivial module \mathfrak{o}^n .
EXAMPLE	

```
kash> o := OrderMaximal(Poly(Zx, [1, 1, 10]));;
kash> ModuleId(o, 4);
{<1><1><1><1>
 [1 0 0 0]
 [0 1 0 0]
 [0 0 1 0]
 [0 0 0 1]
 }
```

NAME	ModuleIdeals
PURPOSE	Retrieves the coefficient ideals of the module representation.
SYNTAX	<pre>L := ModuleIdeals(M);</pre> <p>list L of ideals module M</p>
DESCRIPTION	<p>The module is represented by a pseudomatrix over a Dedekind ring and consists of a matrix and a list of ideals (one for each column). See description of <code>Module</code> for details.</p> <p>If the module is a zero module false is returned.</p>
SEE ALSO	Module , ModuleMatrix ,

EXAMPLE

```
kash> O:=OrderMaximal(Poly(Zx,[1, 5, -6, -53, 3, 206, 244]));;
kash> IL:=List(Factor(2*O),f->f[1]);;
kash> List(IL,IdealGenerators);;
kash> EL1:=List(Factor(5*O),f->IdealGen(f[1],2));;
kash> EL2:=List(Factor(13*O),f->IdealGen(f[1],2));;
kash> EL2[3]:=IdealGen(Factor(7*O)[1][1],2);;
kash> EL3:=List(Factor(3*O),f->IdealGen(f[1],2));;
kash> EL3[3]:=IdealGen(Factor(7*O)[2][1],2);;
kash> M:=Module(IL,Mat(O,[EL1,EL2,EL3]));
{<2, [0, 2, 0, 3, 0, 2]><2, [0, 0, 3, 0, 1, 2]><2, [3, 3, 1, 2, 0, 0]>
[[12, 17, 15, 7, 9, 5] [19, 18, 15, 1, 10, 24] [4, 22, 4, 10, 23, 4]]
[[10, 8, 0, 1, 0, 0] [1, 12, 5, 1, 0, 0] [1, 1, 0, 0, 0, 0]]
[[1, 1, 0, 0, 0, 0] [1, 0, 1, 0, 0, 0] [3, 1, 0, 0, 0, 0]]
}

kash> ModuleIdeals(M);
[ <2, [0, 2, 0, 3, 0, 2]>, <2, [0, 0, 3, 0, 1, 2]>, <2, [3, 3, 1, 2, 0, 0]> ]
```

NAME	ModuleIntersection
PURPOSE	Computes the intersection of two modules.
SYNTAX	<pre>M :=ModuleIntersection(M1, M2 [,d I] [,"PBNF"] [,"lower"]);</pre> <p> modules M, M1, M2 integer d used for reduction ideal I used for reduction </p>
DESCRIPTION	<p>This function computes the intersection $M_1 \cap M_2$ of the given modules. Both modules must be of full rank.</p> <p>The additional parameters control the normal form algorithm application. For their description see ModuleNF.</p> <p>The algorithm computes the dual module of the sum of the dual modules of the input, it is described in [Hop98].</p>
SEE ALSO	ModuleConcat , ModuleNF ,

EXAMPLE

```
kash> O:=OrderMaximal(Poly(Zx, [1,-10,-3,-2]));;
kash> o:=OrderMaximal(O,3,3);;
kash> Oa:=OrderMaximal(OrderAbs(o));;
kash> L:=List(Factor(10*Oa),i->i[1]);;
kash> M1:=IdealBasis(IdealMove(L[2]*L[6], o));
{<10, [26, 1, 0]><2, [0, 0, 1]><1>
 [1 -2 -1]
 [0 1 1]
 [0 0 1]
 }

kash> M2:=IdealBasis(IdealMove(L[1]*L[6], o));
{<10, [-12714, -20615, -67306]><1><1>
 [1 3 1]
 [0 1 0]
 [0 0 1]
 }

kash> ModuleIntersection(M1,M2);
{<10, [88, 2, 99]><2, [0, 2, 1]><2, [2, 2, 1]>
 [1 -2 -1]
```

```
[0 1 1]
[0 0 1]
}
```


NAME ModuleIntersectionVS

PURPOSE Computes the intersection of a module and a VS.

SYNTAX M :=ModuleIntersectionVS(M1, mat);

modules	M, M1, M2	
integer	d	used for reduction
ideal	I	used for reduction

DESCRIPTION

SEE ALSO [ModuleConcat](#), [ModuleNF](#),

EXAMPLE

```
kash> O:=OrderMaximal(Poly(Zx, [1,-10,-3,-2]));;
kash> o:=OrderMaximal(0,3,3);;
kash> Oa:=OrderMaximal(OrderAbs(o));;
kash> L:=List(Factor(10*Oa),i->i[1]);;
kash> M1:=IdealBasis(IdealMove(L[2]*L[6], o));
{<10, [26, 1, 0]><2, [0, 0, 1]><1>
 [1 -2 -1]
 [0 1 1]
 [0 0 1]
 }

kash> M2:=IdealBasis(IdealMove(L[1]*L[6], o));
{<10, [-12714, -20615, -67306]><1><1>
 [1 3 1]
 [0 1 0]
 [0 0 1]
 }

kash> ModuleIntersection(M1,M2);
{<10, [88, 2, 99]><2, [0, 2, 1]><2, [2, 2, 1]>
 [1 -2 -1]
 [0 1 1]
 [0 0 1]
 }
```

NAME	ModuleMap
PURPOSE	Computes the image and kernel under the multiplication by map
SYNTAX	<pre>L :=ModuleMap(M1, mat [, "nf" "kernel"]);</pre> <p>modules M, M1, M2</p> <p>integer d used for reduction</p> <p>ideal I used for reduction</p>

DESCRIPTION

SEE ALSO [ModuleConcat](#), [ModuleNF](#),

EXAMPLE

```
kash> O:=OrderMaximal(Poly(Zx, [1,-10,-3,-2]));;
kash> o:=OrderMaximal(O,3,3);;
kash> Oa:=OrderMaximal(OrderAbs(o));;
kash> L:=List(Factor(10*Oa),i->i[1]);;
kash> M1:=IdealBasis(IdealMove(L[2]*L[6], o));
{<10, [26, 1, 0]><2, [0, 0, 1]><1>
 [1 -2 -1]
 [0 1 1]
 [0 0 1]
 }

kash> M2:=IdealBasis(IdealMove(L[1]*L[6], o));
{<10, [-12714, -20615, -67306]><1><1>
 [1 3 1]
 [0 1 0]
 [0 0 1]
 }

kash> ModuleIntersection(M1,M2);
{<10, [88, 2, 99]><2, [0, 2, 1]><2, [2, 2, 1]>
 [1 -2 -1]
 [0 1 1]
 [0 0 1]
 }
```

NAME	ModuleMatrix
PURPOSE	Retrieves the matrix of the module representation.
SYNTAX	<pre>m := ModuleMatrix(M);</pre> <pre>matrix m</pre> <pre>module M</pre>
DESCRIPTION	<p>The module is represented by a pseudomatrix over a Dedekind ring and consists of a matrix and a list of ideals (one for each column). See description of <code>Module</code> for details.</p> <p>If the module is a zero module false is returned.</p>
SEE ALSO	Module ,
EXAMPLE	<pre>kash> O:=OrderMaximal(Poly(Zx,[1, 5, -6, -53, 3, 206, 244]));; kash> IL:=List(Factor(2*O),f->f[1]);; kash> List(IL,IdealGenerators);; kash> EL1:=List(Factor(5*O),f->IdealGen(f[1],2));; kash> EL2:=List(Factor(13*O),f->IdealGen(f[1],2));; kash> EL2[3]:=IdealGen(Factor(7*O)[1][1],2);; kash> EL3:=List(Factor(3*O),f->IdealGen(f[1],2));; kash> EL3[3]:=IdealGen(Factor(7*O)[2][1],2);; kash> M:=Module(IL,Mat(O,[EL1,EL2,EL3])); {<2, [0, 2, 0, 3, 0, 2]><2, [0, 0, 3, 0, 1, 2]><2, [3, 3, 1, 2, 0, 0]> [[12, 17, 15, 7, 9, 5] [19, 18, 15, 1, 10, 24] [4, 22, 4, 10, 23, 4]] [[10, 8, 0, 1, 0, 0] [1, 12, 5, 1, 0, 0] [1, 1, 0, 0, 0, 0]] [[1, 1, 0, 0, 0, 0] [1, 0, 1, 0, 0, 0] [3, 1, 0, 0, 0, 0]] } kash> ModuleMatrix(M); [[12, 17, 15, 7, 9, 5] [19, 18, 15, 1, 10, 24] [4, 22, 4, 10, 23, 4]] [[10, 8, 0, 1, 0, 0] [1, 12, 5, 1, 0, 0] [1, 1, 0, 0, 0, 0]] [[1, 1, 0, 0, 0, 0] [1, 0, 1, 0, 0, 0] [3, 1, 0, 0, 0, 0]]</pre>

NAME	ModuleMember
PURPOSE	Tests if a column vector is element of the given module.
SYNTAX	$L := \text{ModuleMember}(M, V EL);$ <div style="margin-left: 40px;"> <code>list</code> <code>L</code> see below <code>module</code> <code>M</code> <code>vector</code> <code>V</code> <code>list</code> <code>EL</code> of n algebraic elements of \mathcal{O} </div>
DESCRIPTION	<p>The column vector v is a member of the module iff it is a linear combination of the columns of the matrix of M with coefficients in the corresponding coefficient ideals.</p> <p>This function returns a list of a boolean value (if the element is indeed a member of the module) and a representation column vector x where $Mx = v$. x is returned even if its entries are not members of the corresponding ideals of the module.</p>

EXAMPLE

```

kash> O:=OrderMaximal(Poly(Zx,[1,-10,-3,-2]));;
kash> M:=Module([2*O,2*O,1*O],Mat(O,[[1,0,1],[0,1,1],[0,0,1]]));
{<2><2><1>
 [1 0 1]
 [0 1 1]
 [0 0 1]
 }

kash> V:=MatTrans(O,[[1,3,1]]);
[1]
[3]
[1]
kash> L:=ModuleMember(M,V);
[ true, [0]
      [2]
      [1] ]
kash> ModuleMatrix(M)*L[2];
[1]
[3]
[1]

```

NAME	ModuleModul
PURPOSE	Returns an ideal which can be used for reduction.
SYNTAX	$\mathfrak{a} := \text{ModuleModul}(M)$ <pre>ideal a module M</pre>
DESCRIPTION	<p>Let R be a Dedekind ring, $n \in \mathbb{N}$, $M \subset \mathbb{Q}(R)^n$ be a module of degree n over the ring R. The ideal \mathfrak{a} guarantees $\mathfrak{a}R^n \subset M$ where R^n denotes the 1-module of degree n over R.</p> <p>It is computed as follows: Let d be the denominator of M, $\mathfrak{a} := \frac{1}{d} \det(dM)$.</p>
SEE ALSO	ModuleDet , ModuleDen ,
EXAMPLE	<pre>kash> O:=OrderMaximal(Poly(Zx, [1,-10,-3,-2]));; kash> o:=OrderMaximal(O,3,3);; kash> Oa:=OrderMaximal(OrderAbs(o));; kash> L:=List(Factor(10*Oa),i->i[1]);; kash> M:=IdealBasis(IdealMove(L[2]*L[6]/2, o)); {<5, [4, 1, 2] / 2><1, [2, 3, 2] / 2><1 / 2> [1 -2 -1] [0 1 1] [0 0 1] }</pre> <pre>kash> I := ModuleModul(M); < [20 6 4] [0 1 0] [0 0 1] /16></pre> <pre>kash> I*ModuleId(0,3) < M; false</pre>

NAME	ModuleMove						
PURPOSE	Changes the coefficient order of the module.						
SYNTAX	<p><code>M2 := ModuleMove(M1, o);</code></p> <table> <tr> <td>same type as M1</td><td>M2</td></tr> <tr> <td>module list of modules</td><td>M1</td></tr> <tr> <td>order</td><td>o</td></tr> </table>	same type as M1	M2	module list of modules	M1	order	o
same type as M1	M2						
module list of modules	M1						
order	o						
DESCRIPTION	<p>Suppose M1 is an \mathfrak{o}_1-module and there is an order \mathfrak{o}_2 such that there is an homomorphism from $\phi : \mathfrak{o}_1 \rightarrow \mathfrak{o}_2$ this function will compute $\phi(M1)$. This is done simply by applying ϕ to all the matrix elements and to the coefficient ideals of M1. ϕ must be known to the system before using this function.</p> <p>If M1 is a list of modules all modules contained are moved.</p>						
SEE ALSO	EltMove , IdealMove ,						
EXAMPLE							

```
kash> O:=Order(Poly(Zx,[1,1,-4,-14,3,1]));;
kash> o:=OrderMaximal(O);;
kash> M:=Module([Ideal(2,Elt(o,[0, 0, 0, 1, 1])),1*o],
> Mat(o,[[1,Elt(o, [2, 2, 0, 2, 0])],[0,1]]));
{<2, [0, 0, 0, 1, 1]><1>
 [1 [2, 2, 0, 2, 0]]
 [0 1]
 }
```

```
kash> M1:=ModuleMove(M, O);
{<2, [2, 1, 1, 1, 1] / 2><1>
 [1 [3, 3, 1, 1, 0]]
 [0 1]
 }
```

```
kash> M2:=ModuleMove(M1, o);
{<2, [0, 0, 0, 1, 1]><1>
 [1 [2, 2, 0, 2, 0]]
 [0 1]
 }
```

```
kash> M=M2;
true
```

NAME	ModuleNF
PURPOSE	Computes a normal form of the given module.
SYNTAX	<pre>M2 := ModuleNF(M1 [,d I] [,"PBNF"] [,"lower"]);</pre> <p> modules M1, M2 integer d used for reduction ideal I used for reduction </p>
DESCRIPTION	<p>The module will be represented by a pseudomatrix whose matrix is in triangular form with ones on the diagonal.</p> <p>Two different algorithms are implemented. One is according to [Coh96] (the default), the other one according to [BP91] (which is invoked by the string parameter PBNF).</p> <p>The pseudomatrix will be either in upper triangular form which is the default or in lower triangular form if requested by the string parameter lower.</p> <p>Modular reduction is implemented using either an integer or an ideal.</p>
SEE ALSO	Module ,

EXAMPLE

```

kash> O:=OrderMaximal(Poly(Zx,[1, 5, -6, -53, 3, 206, 244]));;
kash> IL:=List(Factor(2*0),f->f[1]);;
kash> List(IL,IdealGenerators);;
kash> EL1:=List(Factor(5*0),f->IdealGen(f[1],2));;
kash> EL2:=List(Factor(13*0),f->IdealGen(f[1],2));;
kash> EL2[3]:=IdealGen(Factor(7*0)[1][1],2);;
kash> EL3:=List(Factor(3*0),f->IdealGen(f[1],2));;
kash> EL3[3]:=IdealGen(Factor(7*0)[2][1],2);;
kash> M:=Module(IL,Mat(0,[EL1,EL2,EL3]));;
{<2, [0, 2, 0, 3, 0, 2]><2, [0, 0, 3, 0, 1, 2]><2, [3, 3, 1, 2, 0, 0]>
[[12, 17, 15, 7, 9, 5] [19, 18, 15, 1, 10, 24] [4, 22, 4, 10, 23, 4]]
[[10, 8, 0, 1, 0, 0] [1, 12, 5, 1, 0, 0] [1, 1, 0, 0, 0, 0]]
[[1, 1, 0, 0, 0, 0] [1, 0, 1, 0, 0, 0] [3, 1, 0, 0, 0, 0]]
}

kash> ModuleNF(M);
{<26236112388386371287536615054, [294144153883328, 688333593256040823590763075\
215506079571362956748104814491, 106413296427452, 30897359779322, 2714188044413\
95, 688333593256040823590763075215506079571362445884846952856]><2><2, [3, 2, 1\

```

```
, 2, 2, 0]>  
[1 [-10218246369603710638941628710, 0, 1, 0, 0, 1] / 2 [7551087260070784648140\  
818381, -1, 1, 1, 1, 1] / 2]  
[0 1 [2, -1, -1, 1, 2, 0] / 2]  
[0 0 1]  
}
```


NAME	ModuleOrder
PURPOSE	Retrieves the order over which the module is defined.
SYNTAX	<pre>o := ModuleOrder(M); order o module M</pre>
SEE ALSO	Module ,

EXAMPLE

```
kash> O:=OrderMaximal(Poly(Zx,[1, 5, -6, -53, 3, 206, 244]));;
kash> IL:=List(Factor(2*O),f->f[1]);;
kash> List(IL,IdealGenerators);;
kash> EL1:=List(Factor(5*O),f->IdealGen(f[1],2));;
kash> EL2:=List(Factor(13*O),f->IdealGen(f[1],2));;
kash> EL2[3]:=IdealGen(Factor(7*O)[1][1],2);;
kash> EL3:=List(Factor(3*O),f->IdealGen(f[1],2));;
kash> EL3[3]:=IdealGen(Factor(7*O)[2][1],2);;
kash> M:=Module(IL,Mat(O,[EL1,EL2,EL3]));
{<2, [0, 2, 0, 3, 0, 2]><2, [0, 0, 3, 0, 1, 2]><2, [3, 3, 1, 2, 0, 0]>
[[12, 17, 15, 7, 9, 5] [19, 18, 15, 1, 10, 24] [4, 22, 4, 10, 23, 4]]
[[10, 8, 0, 1, 0, 0] [1, 12, 5, 1, 0, 0] [1, 1, 0, 0, 0, 0]]
[[1, 1, 0, 0, 0, 0] [1, 0, 1, 0, 0, 0] [3, 1, 0, 0, 0, 0]]
}

kash> ModuleOrder(M);
F[1]
|
F[2]
/
/
Q
F [ 1]      Given by transformation matrix
F [ 2]      x^6 + 5*x^5 - 6*x^4 - 53*x^3 + 3*x^2 + 206*x + 244
Discriminant: -182099043
```

NAME ModuleSmith

PURPOSE Computes the structure of the quotient module.

SYNTAX L :=ModuleSmith(M1, M2, ["U" | "V" | "UV"]);

modules M1, M2
integer d used for reduction
ideal I used for reduction

DESCRIPTION

SEE ALSO [ModuleConcat](#), [ModuleNF](#),

EXAMPLE

```
kash> O:=OrderMaximal(Poly(Zx, [1,-10,-3,-2]));;
kash> o:=OrderMaximal(O,3,3);;
kash> Oa:=OrderMaximal(OrderAbs(o));;
kash> L:=List(Factor(10*Oa),i->i[1]);;
kash> M1:=IdealBasis(IdealMove(L[2]*L[6], o));
{<10, [26, 1, 0]><2, [0, 0, 1]><1>
 [1 -2 -1]
 [0 1 1]
 [0 0 1]
 }

kash> M2:=IdealBasis(IdealMove(L[1]*L[6], o));
{<10, [-12714, -20615, -67306]><1><1>
 [1 3 1]
 [0 1 0]
 [0 0 1]
 }

kash> ModuleIntersection(M1,M2);
{<10, [88, 2, 99]><2, [0, 2, 1]><2, [2, 2, 1]>
 [1 -2 -1]
 [0 1 1]
 [0 0 1]
 }
```

NAME	ModuleSteinitz
PURPOSE	Computes a Steinitz form of the given module.
SYNTAX	<pre>M2 := ModuleSteinitz(M1); modules M1, M2</pre>
DESCRIPTION	<p>The representation of a module is called to be in Steinitz form if all but possibly the last coefficient ideal are trivial. The ideal class of this non-trivial ideal is called the Steinitz class of the module, and is uniquely determined by the module.</p> <p>The given module must be represented by a square pseudomatrix, <code>ModuleNF</code> could be applied to receive a square pseudomatrix.</p> <p>Even if the given matrix was in triangular form the result is usually not in triangular form. A description of the algorithm can be found in [Coh96, Hop98].</p>
SEE ALSO	Module ,

EXAMPLE

```
kash> O:=OrderMaximal(Poly(Zx,[1, 5, -6, -53, 3, 206, 244]));;
kash> IL:=List(Factor(2*O),f->f[1]);;
kash> List(IL,IdealGenerators);;
kash> EL1:=List(Factor(5*O),f->IdealGen(f[1],2));;
kash> EL2:=List(Factor(13*O),f->IdealGen(f[1],2));;
kash> EL2[3]:=IdealGen(Factor(7*O)[1][1],2);;
kash> EL3:=List(Factor(3*O),f->IdealGen(f[1],2));;
kash> EL3[3]:=IdealGen(Factor(7*O)[2][1],2);;
kash> M:=Module(IL,Mat(O,[EL1,EL2,EL3]));;
{<2, [0, 2, 0, 3, 0, 2]><2, [0, 0, 3, 0, 1, 2]><2, [3, 3, 1, 2, 0, 0]>
[[12, 17, 15, 7, 9, 5] [19, 18, 15, 1, 10, 24] [4, 22, 4, 10, 23, 4]]
[[10, 8, 0, 1, 0, 0] [1, 12, 5, 1, 0, 0] [1, 1, 0, 0, 0, 0]]
[[1, 1, 0, 0, 0, 0] [1, 0, 1, 0, 0, 0] [3, 1, 0, 0, 0, 0]]
}

kash> ModuleSteinitz(M);
{<1><1><1>
[[4394728, -1966075, 1823048, 571045, 4429120, -10180704] [-886494713833578911\
17763, 49309593904741566858502, -52070513627197717051692, -1650569801283810864\
2379, -105665618835177578724552, 250254135800356303073304] [149910328613511789\
69776672140483616, -22967225283143150022165285807093973, -34673488886088036107\
```

```
962483287508403, -4437224226897232365918195124044250, -74602028023951564691914\
894717311488, 197126665681095115272266885027081077]]
[[-329720, 131751, -131932, -39035, -319020, 734938] [58213067248083905543048,\
-27005759280469352852993, 25386500301627438878953, 7941216786374593082408, 59\
825169409273038698515, -138245397998170775484193] [-11853849383701969927973650\
52296325118, 544037966810989998767976423878766838, -48534301452621879401121006\
6434824596, -146424099146002933785473535422180681, -11324954441447246556135245\
21832008039, 2614242618416499310431986234953432438]]
[[-7556, 2694, -2857, -791, -7018, 16204] [3258642871112552540430, -1479582178\
071356246319, 1346137259868262612650, 426410933164074407916, 32313579395021170\
17165, -7449146675255990899196] [-170193889031977664076567799469094156, 748152\
08747439211909974778952022006, -62668187018577464668197075689018408, -19947714\
481520745192370616578138265, -151478708262378372278211079885467801, 3490309531\
12850724274499079221773238]]
}
```

NAME	ModuleUnion
PURPOSE	Computes the union of two modules.
SYNTAX	<pre>M :=ModuleUnion(M1, M2 [,d I] [,"PBNF"] [,"lower"]);</pre> <p> modules M, M1, M2 integer d used for reduction ideal I used for reduction </p>
DESCRIPTION	<p>The result of this function is the union or sum of the two modules in normal form.</p> <p>For a detailed description of the parameters for the modular normal form computation see <code>ModuleNF</code>. This function is basically a <code>ModuleConcat</code> of the modules and a <code>ModuleNF</code> on the result.</p> <p>The operator '+' on modules is identical to <code>ModuleUnion</code> except the additional parameters for the normal form computation.</p>
SEE ALSO	<code>ModuleConcat</code> , <code>ModuleNF</code> ,
EXAMPLE	<pre>kash> O:=OrderMaximal(Poly(Zx, [1,-10,-3,-2]));; kash> o:=OrderMaximal(Order(0,3,3));; kash> Oa:=OrderMaximal(OrderAbs(o));; kash> L:=List(Factor(10*Oa),i->i[1]);; kash> M1:=IdealBasis(IdealMove(L[2]*L[6], o)); {<10, [26, 1, 0]><2, [0, 0, 1]><1> [1 -2 -1] [0 1 1] [0 0 1] } kash> M2:=IdealBasis(IdealMove(L[1]*L[6], o)); {<10, [-12714, -20615, -67306]><1><1> [1 3 1] [0 1 0] [0 0 1] } kash> ModuleUnion(M1,M2); {<5, [0, 2, 2]><1><1>}</pre>

```
[1 -2 1]
[0 1 0]
[0 0 1]
}
```

NAME	Move
PURPOSE	Moves an object to another structure (order, ring)
SYNTAX	<pre> a := Move(L, 0); N := Move(M, r); a := Move(f, r) </pre> <p> integer or polynomial or ideal a list of integers of polynomials or ideals L Order 0 matrix N, M polynomial f ring r </p>
DESCRIPTION	<p>Moves algebraic numbers (Elt), ideals (Ideal), pseudomatrices (Module), or lists of those to the order 0.</p> <p>Moves a matrix (Mat) to a matrix over another ring R.</p> <p>Tries to compute a representation of a polynomial in a different polynomial algebra.</p>
SEE ALSO	EltMove , IdealMove , ModuleMove , MatMove , PolyMove , FFEltMove ,

EXAMPLE This is the same example as IdealMove apart from using the function Move.

```

kash> 0 := OrderMaximal(Order(x^6 - 9*x^4 - 4*x^3 + 27*x^2 - 36*x - 23));;
kash> o := OrderShort(0);;
kash> Move(2*o, 0);
<2>
kash> Move(Factor(2*o), 0);
[ [ <2, [1, -2, 2, 1, 4, -4]>, 6 ] ]

```

MultiCounterInc

NAME MultiCounterInc

PURPOSE Increments a multi counter subject to a bound.

SYNTAX **b** := MultiCounterInc(L, n);

 boolean **b**

 list **L**

 integer | list **n**

DESCRIPTION Increments a multi counter L. Incrementation is done if $L[i] < n$ for some i , if n is an integer or $L[i] < n[i]$ for some i , if n is a list. In case the counter was incremented, “true” is returned. Otherwise “false” is returned.

SEE ALSO [MultiCounterInit](#),

EXAMPLE A counter with 2 items:

```
kash> L := MultiCounterInit(2);;
kash> while MultiCounterInc(L, 2) do
> Print(L, " ");
> od; Print("\n");[ 0, 0 ] [ 1, 0 ] [ 2, 0 ] [ 0, 1 ] [ 1, 1 ] [ 2, 1 ] [ 0, 2 ] [ 1, 2 ]
[ 2, 2 ] >
```


NAME	MultiCounterInit
PURPOSE	Initializes a multi counter.
SYNTAX	<pre>L := MultiCounterInit(n);</pre> <pre>list L integer n</pre>
DESCRIPTION	Initializes a list L of length n to be used by MultiCounterInc.
SEE ALSO	MultiCounterInc ,
EXAMPLE	A counter with 2 items:

```
kash> L := MultiCounterInit(2);;
kash> while MultiCounterInc(L, 2) do
> Print (L, " ");
> od; Print("\n");[ 0, 0 ] [ 1, 0 ] [ 2, 0 ] [ 0, 1 ] [ 1, 1 ] [ 2, 1 ] [ 0, 2 ] [ 1, 2 ]
[ 2, 2 ] >
```

NextPrime

NAME NextPrime

PURPOSE Returns the smallest rational prime greater a given rational integer.

SYNTAX p := NextPrime(n);

 integer p

 integer n

DESCRIPTION

EXAMPLE

```
kash> NextPrime(-1000);
```

```
2
```

```
kash> NextPrime(2);
```

```
3
```

```
kash> NextPrime(NextPrime(NextPrime(NextPrime(0))));
```

```
7
```

NAME Nice

PURPOSE If called with no parameter, this function returns the current priority (nice) value. Otherwise the priority (nice) value is set to the parameter.

SYNTAX Nice(v);
 v := Nice();

 small integer v

EXAMPLE

```
kash> Nice();  
0
```

Norm

NAME	Norm
PURPOSE	Computes the norm of an algebraic number, an ideal, a polynomial or an alff element or divisor.
SYNTAX	<pre> n := Norm(a); n := Norm(b, o); int rational algebraic number polynomial n algebraic element ideal polynomial alff element ff element a algebraic element ff element b order o </pre>
DESCRIPTION	Depending on the type of a or b the norm of an algebraic element, the norm of an ideal, polynomial or an alff element or divisor is computed.

EXAMPLE This examples take place in $\mathbb{Q}(\sqrt{5})$:

```

kash> o := Order(Z, 2, 5);;
kash> Norm(Elt(o, [1, 2]));
-19
kash> u := OrderUnitsFund(o);
[ [2, 1] ]
kash> Norm(u[1]);
-1
kash> Norm(2*o);
4
kash> Norm(Ideal(2, Elt(o, [2, 3 ])));
1

```

NAME	Num														
PURPOSE	Returns the numerator of an object.														
SYNTAX	<pre> d := Num(q); d := Num(a); d := Num(I); </pre> <table> <tr> <td>integer</td><td>d</td></tr> <tr> <td>quotient field element or polynomial</td><td>d</td></tr> <tr> <td>rational</td><td>q</td></tr> <tr> <td>algebraic function field order element</td><td>a</td></tr> <tr> <td>algebraic element</td><td>a</td></tr> <tr> <td>quotient field element or polynomial</td><td>q</td></tr> <tr> <td>ideal</td><td>I</td></tr> </table>	integer	d	quotient field element or polynomial	d	rational	q	algebraic function field order element	a	algebraic element	a	quotient field element or polynomial	q	ideal	I
integer	d														
quotient field element or polynomial	d														
rational	q														
algebraic function field order element	a														
algebraic element	a														
quotient field element or polynomial	q														
ideal	I														
DESCRIPTION	<p>For a rational number the function returns the numerator of the rational argument.</p> <p>For an algebraic number a or for a fractional ideal \mathfrak{a} the function returns $d \cdot a$ or $d \cdot \mathfrak{a}$. d is the smallest integer that $d \cdot a$ is an algebraic integral or an integral ideal</p> <p>For an algebraic function field order element see <code>AlffEltDen</code>.</p>														

EXAMPLE An absolute example for an algebraic element:

```

kash> o := Order (x^3 - 23*x^2 + 146*x - 244);;
kash> a := Elt (o, [1/2,3/5,45/5657567]);
[28287835, 33945402, 450] / 56575670
kash> Num ( a );
[28287835, 33945402, 450]

```

Open

NAME **Open**

PURPOSE (Re)opens a file.

SYNTAX **f := Open(name, mode);**
 ok := Open(f);

File **f**
string **name**
string **mode**
boolean **ok**

DESCRIPTION The **Open** function opens the file specified in the argument **name**. The type of operations you intend to perform on the file must be given in the argument **mode**. The following table explains the values that the **mode** string can take.

Access mode	Interpretation
"r"	Opens file for read operations.
"w"	Opens a new file for writing. If the file exists, its content is destroyed.
"a"	Opens file for appending. A new file is created if the file does not exist.
"R"	Opens file for <i>unbuffered</i> read operations. This mode should be used in connection with the FLDin function.

If the file is successfully opened, **Open** returns a **File** object **f**, an object to be used in subsequent I/O operations on the file. The **File** object is a structure which contains information about the name of the file, the access mode and the status indicating whether the file is opened or closed. In case of an error, **Open** returns **false**.

SEE ALSO **BagRead, BagWrite, Close, ECHOon, ECHOoff, FLDin, FLDout, LOFILES,**

EXAMPLE We will open, close and reopen a file named "dummy".

```
kash> f := Open("dummy","w");
Filename: dummy / Mode: w / Open (fid): 5
kash> Close(f);
true
kash> f;
Filename: dummy / Mode: w / Closed
```

```
kash> Open(f);  
true  
kash> f;  
Filename: dummy / Mode: w / Open (fid): 5
```

NAME	Order
PURPOSE	Returns the order defined by the given arguments.
SYNTAX	<pre> o1 := Order (f); o1 := Order (o,d,alpha); o1 := Order (o,T,d); o1 := Order (o,T,L); o1 := Order (o,L); order o1 order o polynomial f integer d matrix T algebraic element alpha list L </pre>
DESCRIPTION	<p>In the following \mathfrak{o} may be an arbitrary order or the ring of rational integers \mathbb{Z}. At the moment there are no checks to ensure that the created module is a ring. So it is possible that a module is returned which is not a ring. In this case the representation of the module can be used as a representation of the defined number field. Basic arithmetic with the elements of this number field is supported. But be careful when using this module as an argument in functions which require an order. Usually these functions do not check this, so it may lead to an error or even to wrong results (for example calling <code>OrderMaximal</code>).</p>

At the moment there are five different ways of defining an order:

`Order (f)`

creates an equation order $\mathfrak{o}[1] = \mathfrak{o}[\beta]$ with $f \in \mathfrak{o}[t]$ and $f(\beta) = 0$. The polynomial f must have degree greater than 1 and must be irreducible. If f is not monic, $\mathfrak{o}[1]$ may not be a ring (see above).

`Order (o,d,alpha)`

returns the order $\mathfrak{o}[1] = \mathfrak{o}[\sqrt[d]{\alpha}]$, where $d > 1$ is a rational integer and α is an element of the number field defined by \mathfrak{o} . If α is not integral, this leads to the difficulties mentioned above.

`Order (o,T,d)`

T must be an $n \times n$ -matrix over the coefficient ring of \mathfrak{o} and should be non-singular, $d \neq 0$ must be a rational integer.

In the absolute case (coefficient ring is \mathbb{Z}) an order $\mathfrak{o}[1]$ with basis

$$(\omega|1, \dots, \omega|n) = \frac{1}{d}(a|1, \dots, a|n)T$$

is returned, where $a|1, \dots, a|n$ is the basis of \mathfrak{o} . It is possible that $\omega|1, \dots, \omega|n$ is not a basis for an order (see above).

In the relative case an order $\mathfrak{o}|1$ with pseudo basis $\mathfrak{a}|1 \cdot \omega|1, \dots, \mathfrak{a}|n \cdot \omega|n$ is returned, where

$$(\omega|1, \dots, \omega|n) = \frac{1}{d}(a|1, \dots, a|n)T,$$

and $\mathfrak{a}|1 \cdot a|1, \dots, \mathfrak{a}|n \cdot a|n$ is the pseudo basis of \mathfrak{o} . For a detailed description of relative orders and pseudo bases we refer to [Fri97]. What happens if $\mathfrak{a}|1\omega|1, \dots, \mathfrak{a}|n\omega|n$ is not a pseudo basis of a relative order is explained above.

Order (\mathfrak{o}, T, L)

This is for the relative case: \mathfrak{o} must be a relative order with coefficient ring $\bar{\mathfrak{o}}$, a maximal order, and relative degree n , T a $n \times n$ -Matrix over the number field defined by $\bar{\mathfrak{o}}$ and $L = \{\mathfrak{a}|1, \dots, \mathfrak{a}|n\}$ a list of (fractional) ideals defined in $\bar{\mathfrak{o}}$. A relative order $\mathfrak{o}|1$ with pseudo basis $\mathfrak{a}|1\omega|1, \dots, \mathfrak{a}|n\omega|n$ is returned, where

$$(\omega|1, \dots, \omega|n) = (a|1, \dots, a|n)T$$

and $\mathfrak{a}|1a|1, \dots, \mathfrak{a}|na|n$ is the pseudo basis of \mathfrak{o} . For a detailed description of relative orders and pseudo bases we refer to [Fri97]. What happens if $\mathfrak{a}|1\omega|1, \dots, \mathfrak{a}|n\omega|n$ is not a pseudo basis of a relative order is explained above.

Order (\mathfrak{o}, L)

with $L = \{a|1, \dots, a|m\}$, a list of algebraic elements, the function returns the order $\mathfrak{o}|1 = \mathfrak{o} + a|1\mathfrak{o} + \dots + a|m\mathfrak{o}$. In the case that $\mathfrak{o} + a|1\mathfrak{o} + \dots + a|m\mathfrak{o}$ does not define an order, see above.

EXAMPLE Creation of $\mathbb{Q}(\sqrt{2})(\sqrt[3]{3})$ by polynomials.

```
kash> o := Order (Poly (Zx, [1,0,-2]));
Generating polynomial: x^2 - 2

kash> ox := PolyAlg (o);
Univariate Polynomial Ring in x over Generating polynomial: x^2 - 2

kash> zero := Elt (o,0);
0
kash> o1 := Order (Poly (ox, [Elt(o,1),zero,zero,Elt (o,-3)]));
F[1]
      /
      /
      E1[1]
      /
      /
      Q
F [ 1]      x^3 - 3
```

Order

Order

E 1[1] $x^2 - 2$

NAME	<code>OrderAbs</code>
PURPOSE	Creates an absolute extension from a simple relative extension or creates a simple relative extension from a double relative extension.
SYNTAX	<pre> Oa := OrderAbs(O); Oa := OrderAbs(O,"no hom"); order Oa order O </pre>
DESCRIPTION	<p>Given a simple relative order \mathcal{O} – the coefficient ring (<code>OrderCoefOrder</code>) of the coefficient ring of \mathcal{O} is \mathbb{Z} – this function computes an absolute equation order $\mathcal{O} a$ with quotient field isomorphic to the quotient field of \mathcal{O} over \mathbb{Q}.</p> <p>Given a double relative order \mathcal{O} – the coefficient ring of the coefficient ring of \mathcal{O}, denoted by \mathfrak{o}, has the coefficient ring \mathbb{Z} – this function computes a relative equation order $\mathcal{O} a$ with quotient field isomorphic to the quotient field of \mathcal{O} over the quotient field of \mathfrak{o}. \mathfrak{o} has to be maximal.</p> <p>Usually the homomorphisms are computed and installed (cf. <code>OrderInstallHom</code>), so that elements can be moved between the orders \mathcal{O} and $\mathcal{O} a$. But this may take a while. If you want to omit the computation of the homomorphism, use the second optional argument "no hom".</p>
SEE ALSO	<code>OrderInstallHom</code> ,

EXAMPLE The following function merges two absolute orders

```

kash> Merge := function(o1,o2)
> local o1x, f, fl;
> o1x := PolyAlg(o1);
> f := PolyMove(OrderPoly(Zx, o2), o1x);
> fl := Factor (f);
> return OrderAbs(Order(fl[Length(fl)][1]));
> end;
function ( o1, o2 ) ... end

```

We create $\mathbb{Q}(\sqrt{5}, \sqrt{3})$:

```
kash> o := Order(Z, 2, 3);
Generating polynomial: x^2 - 3
```

```
kash> O := Order(o, 2, 5);
```

```
      F[1]
      /
      /
    E1[1]
    /
    /
  Q
F  [ 1]      x^2 - 5
E 1[ 1]      x^2 - 3
```

```
kash> Oa := OrderAbs(O);
Generating polynomial: x^4 - 16*x^2 + 4
```

The element $\sqrt{3}$ of \mathfrak{o} is represented in $\mathcal{O}|a$:

```
kash> a:=EltMove(Elt(o, [0,1]), Oa);
[0, -14, 0, 1] / 4
kash> EltMinPoly(a, Zx);
x^2 - 3
```

A double relative extension:

```
kash> o := OrderMaximal(Z,8,2);
Generating polynomial: x^8 - 2
Discriminant: -2147483648
```

```
kash> oo := Order(o,2,5);
```

```
      F[1]
      /
      /
    E1[1]
    /
    /
  Q
```

```

F [ 1]      x^2 - 5
E 1[ 1]      x^8 - 2

```

```
kash> O := Order(oo,2,7);
```

```

      F[1]
      /
      /
      E2[1]
      /
      /
      E1[1]
      /
      /
Q

```

```

F [ 1]      x^2 - 7
E 2[ 1]      x^2 - 5
E 1[ 1]      x^8 - 2

```

```
kash> Oa := OrderAbs(O);
```

```

      F[1]
      /
      /
      E1[1]
      /
      /
Q

```

```

F [ 1]      x^4 - 24*x^2 + 4
E 1[ 1]      x^8 - 2

```

```
kash> Oaa:= OrderAbs(Oa);
```

```

Generating polynomial: x^32 - 192*x^30 + 16160*x^28 - 779520*x^26 + 23611832*x\
^24 - 461456256*x^22 + 5726131264*x^20 - 42165517824*x^18 + 155056995992*x^16 \
- 167363594496*x^14 + 79016151424*x^12 + 17149424640*x^10 + 206341230816*x^8 -\
1685897697792*x^6 - 781759831296*x^4 - 194736998400*x^2 + 415926965776

```

NAME	OrderAutomorphisms
PURPOSE	Computes or stores automorphisms of the given extension.
SYNTAX	<pre>aut := OrderAutomorphisms(o); aut := OrderAutomorphisms(o, L); aut := OrderAutomorphisms(o, "normal"); aut := OrderAutomorphisms(o, "abel");</pre> <p> list aut list of automorphisms order o the given order list L list of some known automorphisms </p>
DESCRIPTION	<p>This function computes or stores the automorphisms of the given extension. The automorphisms are represented by algebraic numbers which are zeros of the generating polynomials of the given extension. They can be applied to algebraic numbers with the function <code>EltAutomorphism</code>.</p> <p>In the case that some automorphisms are known in the list L, they can be stored with <code>OrderAutomorphisms(o, L);</code>.</p> <p>In the other cases the automorphisms will be computed. The option "normal" can be omitted. With the option "abel" you can turn on the more efficient algorithm for abelian extensions. The computation of automorphisms is only possible for absolute normal extensions. In the case that an extension is not normal the function will return <code>false</code>. The algorithms are described in [Klü97, AK99].</p>
SEE ALSO	<code>EltAutomorphism</code> , <code>OrderAutomorphismsAbel</code> , <code>OrderAutomorphismsNormal</code> ,

EXAMPLE Store an automorphism:

```
kash> o := Order(x^4-4*x^2+1);;
kash> a := Elt(o, [0,-1,0,0]);;
kash> OrderAutomorphisms(o, [a]);
[ [0, 1, 0, 0], [0, -1, 0, 0] ]
```

Compute the automorphisms:

```
kash> o := Order(x^4-4*x^2+1);;
kash> OrderAutomorphisms(o);
[ [0, 1, 0, 0], [0, -4, 0, 1], [0, 4, 0, -1], [0, -1, 0, 0] ]
```

Compute the automorphisms in the abelian case (faster):

```
kash> o := Order(x^4-4*x^2+1);;  
kash> OrderAutomorphisms(o, "abel");  
[ [0, 1, 0, 0], [0, -4, 0, 1], [0, 4, 0, -1], [0, -1, 0, 0] ]
```

NAME	OrderAutomorphismsAbel
PURPOSE	Computes of the given Abelian extension.
SYNTAX	<pre>aut := OrderAutomorphismsAbel(o);</pre> <p> logical aut IsAbelian order o the given order </p>
DESCRIPTION	<p>This function computes the automorphisms of the given Abelian extension. The automorphisms are represented by algebraic numbers which are zeros of the generating polynomials of the given extension. They can be applied to algebraic numbers with the function <code>EltAutomorphism</code>.</p> <p>The computation of automorphisms is only possible for absolute Abelian extensions. In the case that an extension is Abelian the function will return <code>true</code> otherwise <code>false</code>. Using the function <code>OrderAutomorphisms</code> one gets the explicit automorphisms. The algorithms are described in [Klü97, AK99].</p>
SEE ALSO	<code>EltAutomorphism</code> , <code>OrderAutomorphisms</code> , <code>OrderAutomorphismsNormal</code> ,

EXAMPLE Compute the automorphisms:

```
kash> o := Order(x^4-4*x^2+1);;
kash> OrderAutomorphismsAbel(o);
true
```


NAME	<code>OrderAutomorphismsNormal</code>
PURPOSE	Computes automorphisms of the given normal extension.
SYNTAX	<pre>aut := OrderAutomorphismsNormal(o);</pre> <p> <code>list</code> <code>aut</code> list of automorphisms <code>order</code> <code>o</code> the given order </p>
DESCRIPTION	<p>This function computes the automorphisms of the given normal extension. The automorphisms are represented by algebraic numbers which are zeros of the generating polynomials of the given extension. They can be applied to algebraic numbers with the function <code>EltAutomorphism</code>.</p> <p>The computation of automorphisms is only possible for absolute normal extensions. In the case that an extension is normal the function will return <code>true</code> otherwise <code>false</code>. Using the function <code>OrderAutomorphisms</code> one gets the explicit automorphisms. The algorithms are described in [Klü97, AK99]. If it is known that the given extension is Abelian it is strongly recommended to use the function <code>OrderAutomorphismsAbel</code> which is much faster.</p>
SEE ALSO	<code>EltAutomorphism</code> , <code>OrderAutomorphisms</code> , <code>OrderAutomorphismsAbel</code> ,

EXAMPLE Compute the automorphisms:

```
kash> o := Order(x^4-4*x^2+1);;
kash> OrderAutomorphismsNormal(o);
true
```

NAME	OrderBach
PURPOSE	Returns the value of the floor function of the Bach bound of the algebraic number field which is generated by the given order.
SYNTAX	<pre>b := OrderBach(O);</pre> <pre>integer b</pre> <pre>order O</pre>
DESCRIPTION	<p>The given order \mathfrak{o} must be maximal, otherwise an error is returned.</p> <p>A theorem of Bach asserts (under the assumption of GRH) that all prime ideals with norm below this bound generate the class group.</p>
SEE ALSO	OrderMinkowski ,

EXAMPLE Compute the floor of the Bach bound of the algebraic number field $\mathbb{Q}(\sqrt[17]{2})$.

```
kash> o := Order(Z,17,2);
Generating polynomial: x^17 - 2

kash> O := OrderMaximal(o);
Generating polynomial: x^17 - 2
Discriminant: 54214017802982966177103872

kash> OrderBach(O);
42133
```

NAME	OrderBasis
PURPOSE	Returns a basis whose elements are elements of the given order with a denominator.
SYNTAX	<pre>L := OrderBasis(o); L := OrderBasis(o, 0);</pre> <p>list of algebraic elements L order o order 0</p>
DESCRIPTION	If the second order is omitted, the basis returned consists of elements of the equation order. Otherwise the elements are moved into the second order. In the relative case the basis elements of the pseudo basis are returned.
SEE ALSO	OrderCoefIdeals ,
EXAMPLE	A basis of the maximal order of $x^4 + 73x^2 - 280x - 2399$ represented in the equation order and in the maximal order itself.

```
kash> O := OrderMaximal (Poly (Zx,[1,0, 73, -280, -2399]));
  F[1]
  |
  F[2]
  /
  /
Q
F [ 1]      Given by transformation matrix
F [ 2]      x^4 + 73*x^2 - 280*x - 2399
Discriminant: -997975

kash> OrderBasis (O);
[ 1, [0, 1, 0, 0], [1, 1, 1, 0] / 2, [414, 857, 795, 1] / 1646 ]
kash> OrderBasis (O, 0);
[ 1, [0, 1, 0, 0], [0, 0, 1, 0], [0, 0, 0, 1] ]
```

NAME	OrderBasisIsPower
PURPOSE	Returns <code>true</code> iff the given order is an equation order.
SYNTAX	<pre>B := OrderBasisIsPower(o);</pre> <pre>boolean B order o</pre>
DESCRIPTION	<code>OrderBasisIsPower</code> checks whether in the actual KANT-representation a given order <code>o</code> is an equation order or not. It does not check whether a given order could be represented as an equation order.
SEE ALSO	OrderBasisIsRel , OrderEquationOrder ,
EXAMPLE	Check, whether or not a certain order is an equation order.

```
kash> o := Order (Poly (Zx,[1,0,-186,0,13097,0,-412704,0,4946176]));
Generating polynomial: x^8 - 186*x^6 + 13097*x^4 - 412704*x^2 + 4946176
```

```
kash> O := OrderMaximal (o);
F[1]
|
F[2]
/
/
Q
F [ 1]      Given by transformation matrix
F [ 2]      x^8 - 186*x^6 + 13097*x^4 - 412704*x^2 + 4946176
Discriminant: 60050488100625
```

```
kash> OrderBasisIsPower (O);
false
kash> OrderBasisIsPower (OrderEquationOrder (O));
true
```

NAME	OrderBasisIsRel
PURPOSE	Returns true iff the basis of an order is given by a transformation matrix.
SYNTAX	<pre>b := OrderBasisIsRel(o);</pre> <p>boolean b</p> <p>order o</p>
SEE ALSO	OrderBasisIsPower , OrderEquationOrder ,
EXAMPLE	Checks, whether a certain order is an equation order or given by a transformation matrix.

```
kash> O:= Order (Poly (Zx,[1,0,-186,0,13097,0,-412704,0,4946176]));
Generating polynomial: x^8 - 186*x^6 + 13097*x^4 - 412704*x^2 + 4946176
```

```
kash> OrderBasisIsRel(OrderEquationOrder(O));
false
kash> OrderBasisIsRel(O);
false
```

NAME	OrderClassGroup
PURPOSE	Computes the class group.
SYNTAX	<pre>L := OrderClassGroup(O [,b] [,"fast"] [,"Euler"]);</pre> <p>list L order O integer b</p>
DESCRIPTION	<p>The OrderClassGroup function computes the class group of the algebraic number field which is generated by the order \mathcal{O}. It returns a list L whose first entry is the class number. The second entry is a list which contains the orders of the cyclic subgroups.</p> <p>Notice that the class group is stored in the order \mathcal{O}. To get more information on the class group print out the order.</p> <p>Several (optional) arguments can be passed to this function: The first argument has always to be the order. As a second argument a bound b for prime ideals to be considered may be given. If it is omitted, OrderClassGroup uses the Minkowski bound for the computation of the class group. The Minkowski bound always guarantees correct results. However, when the field discriminant is <i>large</i>, the Minkowski bound requires very time consuming computations. Additionally, the following options can be passed to speed up the computation: "fast" indicates a fast computation without checking the results "Euler" uses the Euler product for computation. The check omitted when using "fast" may be done later by using OrderClassGroupCheck.</p>
SEE ALSO	OrderClassGroupCheck , OrderClassGroupCyclicFactors , IdealClassRep , IdealIsPrincipal ,

EXAMPLE We are going to compute the class group of $\mathcal{F} = \mathbb{Q}(\sqrt[4]{-65})$.

```
kash> o := OrderMaximal(Z,4,-65);;
kash> Time(true);
true
Time: 0 ms
kash> OrderClassGroup(o);
[ 128, [ 2, 4, 4, 4 ] ]
Time: 440 ms
```

The class group is of order 128 and is isomorphic to $C_2 \times C_4 \times C_4 \times C_4$.

In the example above, the time display was activated. Compare the runtime difference when computing the class group of \mathcal{F} by taking 30 as a bound for the prime ideals. Notice, that the Minkowski bound is 1274.

```
kash> o := OrderMaximal(Z,4,-65);;  
kash> OrderMinkowski(o);  
1274  
Time: 0 ms  
kash> OrderClassGroup(o,30);  
[ 128, [ 2, 4, 4, 4 ] ]  
Time: 180 ms
```

NAME	OrderClassGroupCheck
PURPOSE	Checks the results of class group computations done by using the Euler product.
SYNTAX	<pre>OrderClassGroupCheck(o, [[lb,] ub] [p, "pmax"]);</pre> <p> <code>order</code> <code>o</code> <code>integer</code> <code>ub</code> upper resp. lower bound <code>integer</code> <code>p</code> prime number </p>
DESCRIPTION	<p>This function checks the computed class group to be the ideal class group generated by all prime ideals with norm below the used ideal bound. The units are proved to be fundamental. You may specify by <code>lb</code> and <code>ub</code> a lower and upper bound for primes p for which the units are checked to be p-maximal. In this way you may compute a class group using the option "fast" first and then perform this test later on. This function returns after quite a while if the results are wrong. It is therefore sometimes convenient to switch on output using the <code>PRINTLEVEL</code> function.</p> <p>The last calling sequence is used to ensure that subsequently computed S-units are p-maximal (special case of the previous calling sequences).</p>
SEE ALSO	OrderClassGroup ,

EXAMPLE Of $\mathbb{Q}(\sqrt[6]{-13})$ we check the results:

```
kash> o := OrderMaximal(x^6+13);;
kash> OrderClassGroup(o, 500, fast);
[ 6, [ 6 ] ]
kash> OrderClassGroupCheck(o);
true
```


NAME	<code>OrderClassGroupCyclicFactors</code>
PURPOSE	Returns a list with the generators of the cyclic factors.
SYNTAX	<pre>L := OrderClassGroupCyclicFactors(0);</pre> <pre>list L order 0</pre>
DESCRIPTION	<p>Returns a list with the generators of the cyclic factors of the class group together with their orders in the class group. The list contains sublists and each sublist contains two items. The first is an ideal, and the second item is the order of this ideal in the class group.</p> <p>This function does not compute the class group so that <code>OrderClassGroup</code> has to be called in advance.</p>
SEE ALSO	<code>OrderClassGroup</code> , <code>OrderClassGroupCyclicFactorsPrincipal</code> ,
EXAMPLE	Compute the class group structure of $\mathbb{Q}(\sqrt[4]{-65})$.

```
kash> O := OrderMaximal(Z,4,-65);;
kash> OrderClassGroup (O, 100);
[ 128, [ 2, 4, 4, 4 ] ]
kash> OrderClassGroupCyclicFactors (O);
[ [ <30, [885, 896, 0, 899]>, 2 ], [ <3, [2, 1, 0, 0]>, 4 ],
  [ <3, [1, 0, 1, 0]>, 4 ], [ <2, [1, 1, 0, 0]>, 4 ] ]
```

NAME	OrderClassGroupCyclicFactorsPrincipal
PURPOSE	Returns a list of the generators of the cyclic factors of the class group to the power of their orders.
SYNTAX	<pre>L := OrderClassGroupCyclicFactorsPrincipal(0, ["raw"]);</pre> <pre>list L</pre> <pre>order 0</pre>
DESCRIPTION	<p>This function computes the generating elements of the powers of the cyclic factors of the class group to their orders (which are principal integral ideals). Given a second string parameter "raw" a power product representation is returned.</p> <p>This function does not compute the class group so that <code>OrderClassGroup</code> has to be called in advance.</p>
SEE ALSO	OrderClassGroup , OrderClassGroupCyclicFactors ,

EXAMPLE Compute the class group structure of $\mathbb{Q}(\sqrt[4]{-65})$.

```
kash> O := OrderMaximal(Z,4,-65);;
kash> OrderClassGroup (O, 100);
[ 128, [ 2, 4, 4, 4 ] ]
kash> L1 := OrderClassGroupCyclicFactors(O);
[ [ <30, [885, 896, 0, 899]>, 2 ], [ <3, [2, 1, 0, 0]>, 4 ],
  [ <3, [1, 0, 1, 0]>, 4 ], [ <2, [1, 1, 0, 0]>, 4 ] ]
kash> L2 := OrderClassGroupCyclicFactorsPrincipal(O);
[ [5, 0, -1, 0], [-2, -1, 0, 0], [4, 0, 1, 0], 2 ]
kash> L1[2][1]^4/L2[2] = 1*0;
true
```

NAME	OrderClassGroupFactorBasisProve
PURPOSE	Checks whether a small factorbasis generates the same subgroup of the class-group as a large factorbasis.
SYNTAX	<pre>r := OrderClassGroupFactorBasisProve(o, lb, ub);</pre> <p> boolean r order o integer lb, ub lower resp. upper bound </p>
DESCRIPTION	<p>This function is used to check whether a given small factorbasis generates the same subgroup of the classgroup as a large one. It is recommended first to compute the class group using a small bound, i. e. a small factorbasis, and then afterwards to check if the class group was computed itself.</p> <p>Actually this function is not really needed since one has a good criterion by the euler product but which is not fully proven. This function is only able to detect the equality of the factor bases. It should be used only for small fields and may take a very long time.</p>

SEE ALSO [OrderClassGroup](#),

EXAMPLE Compute the class group structure of $\mathbb{Q}(\sqrt[4]{-65})$.

```
kash> O := OrderMaximal(Z,4,-65);;
kash> OrderClassGroup (O, 100, "Euler");
[ 128, [ 2, 4, 4, 4 ] ]
kash> OrderMinkowski(O);
1274
kash> OrderClassGroupFactorBasisProve(O, 100, 1274);
true
```

NAME	OrderCoefIdeals
PURPOSE	Returns the list of coefficient ideals of a relative order.
SYNTAX	<pre>L := OrderCoefIdeals(o);</pre> <p>list of ideals L relative order o</p>
DESCRIPTION	<p>Let \mathfrak{o} be a relative order over a maximal order \mathcal{O}. Then \mathfrak{o} can be represented via a pseudo basis:</p> $\mathfrak{o} = \mathfrak{a} 1\xi 1 + \dots + \mathfrak{a} n\xi n$ <p>where the $\mathfrak{a} i$ are \mathcal{O}-ideals and the ξi (possibly fractional) elements of \mathcal{O}. The $\mathfrak{a} i$ are the coefficient ideals (returned by this function), the ξi are the basis elements (returned by <code>OrderBasis</code>).</p> <p>Returns an error if the order is not relative.</p>
SEE ALSO	OrderBasis ,
EXAMPLE	The coefficient ideals of a relative maximal order.

```
kash> O := OrderMaximal(Z, 2, 10);
Generating polynomial: x^2 - 10
Discriminant: 40
```

```
kash> o := OrderMaximal(O, 2, 5);
      F[1]
      |
      F[2]
      /
      /
      E1[1]
      /
      /
      Q
F  [ 1]    Given by transformation matrix
F  [ 2]    x^2 - 5
E 1[ 1]    x^2 - 10
Discriminant: <1>
Coef. Ideals are: <1>, <1, [5, 1] / 10>
```

```
kash> OrderCoefIdeals(o);
[ <1>, <1, [5, 1] / 10> ]
```

NAME	OrderCoefOrder
PURPOSE	Returns the coefficient ring of a given order.
SYNTAX	<pre>c := OrderCoefOrder(o);</pre> <div> <div>order, ring of integers</div> <div>c</div> <div>order</div> <div>o</div> </div>
DESCRIPTION	Returns the coefficient ring c of the given order o .
SEE ALSO	OrderEquationOrder , OrderSubOrder ,
EXAMPLE	A simple relative extension

```
kash> o1:=Order(Z,2,2);
Generating polynomial: x^2 - 2
```

```
kash> o2:=Order(o1,2,3);
```

```

      F[1]
      /
      /
E1[1]
/
/
Q
F [ 1]      x^2 - 3
E 1[ 1]      x^2 - 2
```

```
kash> OrderCoefOrder(o2);
Generating polynomial: x^2 - 2
```

NAME OrderCyclotomic

PURPOSE Computes the n 'th cyclotomic field.

SYNTAX `o := OrderCyclotomic(n);`

 order o
 integer n

DESCRIPTION

EXAMPLE

```
kash> o := OrderCyclotomic(15);
Generating polynomial: x^8 - x^7 + x^5 - x^4 + x^3 - x + 1

kash> OrderIsMaximal(o);
true
kash> OrderDisc(o);
1265625
kash> OrderAutomorphisms(o, []);
[ [0, 1, 0, 0, 0, 0, 0, 0], [0, 0, 1, 0, 0, 0, 0, 0],
  [0, 0, 0, 0, 1, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 1],
  [-1, 1, 0, -1, 1, -1, 0, 1], [0, -1, 0, 0, 0, 0, -1, 0],
  [1, -1, 0, 0, -1, 1, 0, -1], [1, 0, -1, 1, -1, 0, 1, -1] ]
```

NAME `OrderCyclotomicRealSubfield`

PURPOSE Computes the maximal real subfield of the n 'th cyclotomic field.

SYNTAX `o := OrderCyclotomicRealSubfield(n);`

 order o
 integer n

DESCRIPTION

EXAMPLE

```
kash> o := OrderCyclotomicRealSubfield(11);  
Generating polynomial:  $x^5 + x^4 - 4x^3 - 3x^2 + 3x + 1$ 
```

NAME	OrderDeg
PURPOSE	Returns the relative degree of the given order.
SYNTAX	<pre>d := OrderDeg(o);</pre> <p>integer d order o</p>
DESCRIPTION	Returns the degree $[\mathbb{Q}(\mathfrak{o}) : \mathbb{Q}(C(\mathfrak{o}))]$ of $\mathbb{Q}(\mathfrak{o})$ as vector space over $\mathbb{Q}(C(\mathfrak{o}))$, where $C(\mathfrak{o})$ is the coefficient ring of the given order.
SEE ALSO	OrderDegAbs ,
EXAMPLE	Some relative degrees of extensions.

```
kash> F := Order (Poly (Zx, [1,0,-186,0,13097,0,-412704,0,4946176]));
Generating polynomial: x^8 - 186*x^6 + 13097*x^4 - 412704*x^2 + 4946176
```

```
kash> E := Order (F, 2, 7);
      F[1]
      /
      /
      E1[1]
      /
      /
      Q
F  [ 1]      x^2 - 7
E 1[ 1]      x^8 - 186*x^6 + 13097*x^4 - 412704*x^2 + 4946176
```

```
kash> OrderDeg (E);
2
kash> OrderDeg (F);
8
```


NAME	OrderDegAbs
PURPOSE	Returns the absolute degree of the given order.
SYNTAX	<pre>d := OrderDegAbs(o);</pre> <p>integer d order o</p>
DESCRIPTION	Returns $[\mathbb{Q}(\mathfrak{o}) : \mathbb{Q}]$, the absolute degree of the given order \mathfrak{o} .
SEE ALSO	OrderDeg ,
EXAMPLE	Absolute degrees of extensions.

```
kash> F := Order (Poly (Zx, [1,0,-186,0,13097,0,-412704,0,4946176]));
Generating polynomial: x^8 - 186*x^6 + 13097*x^4 - 412704*x^2 + 4946176
```

```
kash> E := Order (F, 2, 7);
```

```

      F[1]
      /
      /
    E1[1]
    /
    /
  Q
F  [ 1]      x^2 - 7
E 1[ 1]      x^8 - 186*x^6 + 13097*x^4 - 412704*x^2 + 4946176
```

```
kash> OrderDegAbs (E);
```

```
16
```

```
kash> OrderDegAbs (F);
```

```
8
```

NAME	OrderDisc
PURPOSE	Returns the discriminant of the given order.
SYNTAX	$D := \text{OrderDisc}(o);$ <div style="display: flex; justify-content: space-between;"> <div>integer or ideal</div> <div>D</div> </div> <div style="display: flex; justify-content: space-between;"> <div>order</div> <div>\mathfrak{o} over \mathbb{Z} or over a maximal order</div> </div>
DESCRIPTION	Returns the discriminant of the order \mathfrak{o} given over \mathbb{Z} or over a maximal order. The discriminant of an absolute order is a rational integer, of a relative order an ideal.
SEE ALSO	OrderDisc , PolyDisc ,
EXAMPLE	Compute the discriminant of an order over \mathbb{Z} .

```
kash> o := Order (Poly (Zx, [1,0,-186,0,13097,0,-412704,0,4946176]));
Generating polynomial: x^8 - 186*x^6 + 13097*x^4 - 412704*x^2 + 4946176
```

```
kash> OrderDisc (o);
822201012939308277619413178871929896960000
kash> o := OrderMaximal (o);;
kash> OrderDisc (o);
60050488100625
```

NAME	OrderEquationOrder
PURPOSE	Returns the equation suborder of the given order.
SYNTAX	<pre>oe := OrderEquationOrder(o); order oe order o</pre>
DESCRIPTION	<code>OrderEquationOrder</code> returns the equation suborder $\mathfrak{o} E$ of a given order \mathfrak{o} .
SEE ALSO	OrderBasisIsPower , OrderBasisIsRel ,
EXAMPLE	Extracting the equation order of an arbitrary order.

```
kash> o := Order (Poly (Zx,[1,0,-186,0,13097,0,-412704,0,4946176]));
Generating polynomial: x^8 - 186*x^6 + 13097*x^4 - 412704*x^2 + 4946176
```

```
kash> O := OrderMaximal (o);
      F[1]
      |
      F[2]
      /
      /
Q
F [ 1]      Given by transformation matrix
F [ 2]      x^8 - 186*x^6 + 13097*x^4 - 412704*x^2 + 4946176
Discriminant: 60050488100625
```

```
kash> oe := OrderEquationOrder (O);
Generating polynomial: x^8 - 186*x^6 + 13097*x^4 - 412704*x^2 + 4946176
Discriminant: 822201012939308277619413178871929896960000
```

NAME	OrderExcepSequence
PURPOSE	Computes a sequence of exceptional units of maximal length in a given order.
SYNTAX	$L := \text{OrderExcepSequence } (o);$ <p>list L exceptional sequence order o</p>
DESCRIPTION	<p>Let \mathfrak{o} be an order over \mathbb{Z} and let $\omega_1, \dots, \omega_m$ ($m \geq 2$) be a sequence of elements of \mathfrak{o}. We say that this sequence is an exceptional sequence if $\omega_1 = 0, \omega_2 = 1$ and all the mutual differences $\omega_i - \omega_j$ ($1 \leq i < j \leq m$) are units. Note that for such an exceptional sequence the elements $\omega_3, \dots, \omega_m$ are exceptional units in the terminology of T. Nagell [Nag69].</p> <p>The <code>OrderExcepSequence</code> function returns an exceptional sequence of maximal length.</p>
SEE ALSO	<code>OrderUnitsExcep</code> ,

EXAMPLE Compute an exceptional sequence of maximal length in $\mathbb{Z}[\zeta_7]$:

```
kash> o := Order(Poly(Zx,[1,1,1,1,1,1,1]));
Generating polynomial: x^6 + x^5 + x^4 + x^3 + x^2 + x + 1

kash> OrderExcepSequence(o);
[ 0, 1, [3, 0, 2, 1, 1, 2], [0, -1, 0, -1, 0, -1], [0, 0, -1, 0, 0, -1],
  [1, 0, 1, 0, 0, 1], [-2, 0, -2, -1, -1, -2] ]
```

NAME	OrderFincke
PURPOSE	Computes or sets the Fincke constants.
SYNTAX	<pre>OrderFincke(o); OrderFincke(o, x); OrderFincke(o, x, y);</pre> <pre>order o real x real y</pre>
DESCRIPTION	<p>Computes the optimal λ and $\gamma 1$ as described in [PZ89, p. 341]. λ is defined as the unique zero of a function.</p> <p>If the second optional argument x is specified, the constant λ is set to be the value of x and $\gamma 1$ is computed in an iteration beginning with the third optional argument y or 0 otherwise.</p>
SEE ALSO	OrderNormEquation ,
EXAMPLE	

```
kash> Time(true);
true
Time: 0 ms
kash> o:=Order(Poly(Zx,[1,0,-4,0,1]));
Generating polynomial: x^4 - 4*x^2 + 1

Time: 10 ms
kash> OrderUnitsFund(o);
kash> OrderNormEquation(o, 3, 1, "abs");
[ ]
Time: 220 ms
kash> OrderNormEquation(o, 3, 1, "abs");
[ ]
Time: 0 ms
kash> OrderFincke(o, 10.0);
kash> OrderNormEquation(o, 3, 1, "abs");
[ ]
Time: 0 ms
kash> Time(false);
false
```

NAME	OrderGalois
PURPOSE	Computation of Galois groups.
SYNTAX	
DESCRIPTION	This function computes the Galois group of an irreducible polynomial with coefficients in \mathbb{Q} and with degree up to 15. For description see <code>Galois()</code> .
SEE ALSO	<code>Galois</code> , <code>GaloisT</code> , <code>GaloisGlobals</code> , <code>GaloisGroupsPossible</code> , <code>GaloisModulo</code> , <code>GaloisTree</code> , <code>GaloisRoots</code> , <code>GaloisNumberToName</code> , <code>GaloisBlocks</code> ,

NAME	OrderIndex
PURPOSE	Returns the index of the given order relative to its suborder.
SYNTAX	$I := \text{OrderIndex}(o);$ <div style="margin-left: 100px;"> integer I order o </div>
DESCRIPTION	<p>This function returns the index of the suborder $\mathfrak{o} 1$ of \mathfrak{o} in \mathfrak{o}. The index equals $\sqrt{\text{disc}(\mathfrak{o} 1)/\text{disc}(\mathfrak{o})}$. In the absolute case the index is a rational integer, otherwise it is an ideal. If \mathfrak{o} has no suborder 1 resp. the ideal generated by the coefficient order is returned.</p>
SEE ALSO	OrderDisc ,

EXAMPLE Computing and checking the index of a maximal order.

```
kash> o := Order (Poly (Zx, [1,0,-186,0,13097,0,-412704,0,4946176]));
Generating polynomial: x^8 - 186*x^6 + 13097*x^4 - 412704*x^2 + 4946176
```

```
kash> OrderIndex (o);
1
kash> O := OrderMaximal (o);
  F[1]
  |
  F[2]
  /
  /
Q
F [ 1]      Given by transformation matrix
F [ 2]      x^8 - 186*x^6 + 13097*x^4 - 412704*x^2 + 4946176
Discriminant: 60050488100625
```

```
kash> OrderIndex (O);
117012089012224
kash> OrderDisc (o) / OrderDisc (O) - OrderIndex (O)^2;
0
```

NAME	OrderIndexFormEquation
PURPOSE	Solves an index form equation.
SYNTAX	$L := \text{OrderIndexFormEquation}(o, \text{index});$ <div> list L order o integer index </div>
DESCRIPTION	<p>Let \mathfrak{o} be an order over \mathbb{Z}. We say that algebraic integers $a, b \in \mathfrak{o}$ are \mathbb{Z}-equivalent iff $a - b$ or $a + b$ is a rational integer. The <code>OrderIndexFormEquation</code> function computes a full set of pairwise inequivalent $\alpha \in \mathfrak{o}$ such that the module index $(\mathfrak{o} : \mathbb{Z}[\alpha])$ equals <code>index</code>. Note that this set is always finite due to a result of K. Györy.</p> <p>At present, the <code>OrderIndexFormEquation</code> function can only handle cubic and quartic fields. Algorithms are taken from [GPP93, GPP96, GS89].</p>

SEE ALSO [EltIndex](#),

EXAMPLE Compute - up to \mathbb{Z} -equivalence - all generators of power integral bases of $\mathbb{Z}[\sqrt[3]{2}]$
:

```
kash> o := Order(Z,3,2);
Generating polynomial: x^3 - 2
```

```
kash> OrderIndexFormEquation(o,1);
[ [0, 1, 0], [0, 1, 1] ]
```


NAME	OrderInstallHom
PURPOSE	Installs an embedding from the first given order into the second given order.
SYNTAX	OrderInstallHom(o1,o2,alpha); OrderInstallHom(o1,o2,b); <div style="display: flex; justify-content: space-between;"> <div>order</div> <div>o1</div> </div> <div style="display: flex; justify-content: space-between;"> <div>order</div> <div>o2</div> </div> <div style="display: flex; justify-content: space-between;"> <div>algebraic element</div> <div>alpha</div> </div> <div style="display: flex; justify-content: space-between;"> <div>boolean</div> <div>b</div> </div>
DESCRIPTION	Installs the embedding given via the image of the primitive element of $\mathfrak{o} 1$ in $\mathfrak{o} 2$ which is defined to be α in case $\mathfrak{o} 1$ is an equation order. If the third argument is a boolean instead of an algebraic element, the corresponding number fields are assumed to be isomorphic, iff the boolean is ‘true’ and this will be used in the sequel.

EXAMPLE Moving of an element from an absolute extension into a relative one.

```

kash> o1 := Order (Z,2,3);;
kash> o2 := Order (o1,2,2);
      F[1]
      /
      /
      E1[1]
      /
      /
      Q
      F [ 1]      x^2 - 2
      E 1[ 1]     x^2 - 3

kash> gen1 := OrderBasis (OrderEquationOrder (o1))[2];
      [0, 1]
kash> gen2 := OrderBasis (OrderEquationOrder (o2))[2];
      [0, 1]
kash> gen := EltMove (gen1, o2) + gen2;
      [[0, 1], 1]
kash> o3 := Order (MatMinPoly (EltRepMat(gen, Z)[2]));
Generating polynomial: x^4 - 10*x^2 + 1

kash> OrderInstallHom (o3, o2, gen);
kash> EltMove (Elt (o3, [1,2,3,4]), o2);

```

```
[[16, 38], [46, 6]]
```

```
kash> EltMove (Elt (o3, [1,2,3,4]/2), o2);
```

```
[[8, 19], [23, 3]]
```

NAME	OrderIsMaximal
PURPOSE	Tests whether the given order is known to be maximal or not.
SYNTAX	$B := \text{OrderIsMaximal}(o);$ <div style="margin-left: 100px;"> boolean B order o </div>
DESCRIPTION	This function checks if the given order \mathfrak{o} is known to be maximal according to the OrderIsMaximal flag.
SEE ALSO	OrderSetMaximal ,

EXAMPLE The order $\mathbb{Z}[\alpha]$ with $a^4 + 73a^2 - 280a - 2399 = 0$ is not known to be maximal.

```

kash> o := Order (Poly(Zx,[1,0,73,-280,-2399]));;
kash> OrderIsMaximal (o);
false
kash> O := OrderMaximal (o);
      F[1]
      |
      F[2]
      /
      /
      Q
F  [ 1]      Given by transformation matrix
F  [ 2]      x^4 + 73*x^2 - 280*x - 2399
Discriminant: -997975

kash> OrderIsMaximal (O);
true

```

NAME	OrderIsRelative
PURPOSE	Tests whether the given order is given as a relative extension.
SYNTAX	<pre>B := OrderIsMaximal(o);</pre> <pre>boolean B order o</pre>
SEE ALSO	OrderSetMaximal ,

NAME	OrderIsSubfield
PURPOSE	Tests $Q(o 1) \subseteq Q(o 2)$.
SYNTAX	OrderIsSubfield(o1, o2); <div style="margin-left: 100px;">order o1 order o2</div>
DESCRIPTION	This function checks whether $Q(o 1) \subseteq Q(o 2)$ and returns true or false. If true is returned, a homomorphism between $o 1$ and $o 2$ is saved (globally) for further usage in, say, moving elements from $o 1$ and $o 2$.

EXAMPLE Check a certain subfield.

```

kash> o1 := Order (Z,2,3);
Generating polynomial: x^2 - 3

kash> o2 := Order (o1, 2, 2);
      F[1]
      /
      /
      E1[1]
      /
      /
      Q
F  [ 1]      x^2 - 2
E 1[ 1]      x^2 - 3

kash> M := EltRepMat (Elt (o2, [[0,1], 1]),Z)[2];
[0 3 2 0]
[1 0 0 2]
[1 0 0 3]
[0 1 1 0]
kash> o3 := Order (MatMinPoly (M));
Generating polynomial: x^4 - 10*x^2 + 1

kash> EltMove (Elt (o1,[0,1]), o3);
[0, -11, 0, 1] / 2
kash> OrderIsSubfield (o1, o3);
true

```

NAME	OrderKextGenAbs
PURPOSE	Computes a set of absolute generators for the ring of integers.
SYNTAX	<p><code>Gen := OrderKextGenAbs(F);</code></p> <p>list Gen of algebraic elements order F</p>
DESCRIPTION	Computes absolute generators for the ring of integers of a Kummer extension of prime degree [Dab95].

EXAMPLE Absolute generators in $\mathbb{Q}(\sqrt{10}, \sqrt{5})$ over $\mathbb{Q}(\sqrt{10})$

```
kash> E := OrderMaximal (Z,2,10);;
kash> F := Order (E,2,5);
      F[1]
      /
      /
      E1[1]
      /
      /
      Q
F  [ 1]      x^2 - 5
E 1[ 1]      x^2 - 10

kash> OrderKextGenRel(F);
[ 1, [1, 1] / 2, [[0, 5], [0, 9]] / 10 ]
```

NAME	OrderKextGenRel
PURPOSE	Computes relative generators for the ring of integers.
SYNTAX	<code>Gen := OrderKextGenRel(F);</code> <code>list</code> <code>Gen</code> of algebraic elements <code>order</code> <code>F</code>
DESCRIPTION	Computes a set of relative generators for the ring of integers of a Kummer extension of prime degree [Dab95].
SEE ALSO	<code>OrderKextGenRel</code> , <code>OrderKextDisc</code> ,
EXAMPLE	Relative generators in $\mathbb{Q}(\sqrt{10}, \sqrt{5})$ over $\mathbb{Q}(\sqrt{10})$

```
kash> E := OrderMaximal (Z,2,10);;
kash> F := Order (E,2,5);
      F[1]
      /
      /
      E1[1]
      /
      /
      Q
      F [ 1]      x^2 - 5
      E 1[ 1]     x^2 - 10

kash> OrderKextGenAbs(F);
[ 0, [[0, 2], 0], [1, 1] / 2, [[0, 5], [0, 1]] / 10 ]
```

NAME	OrderKextModularPower
PURPOSE	Computes a maximal exponent.
SYNTAX	$L := \text{OrderKextModularPower}(\mathfrak{pI}, \mu);$ <div style="margin-left: 100px;"> list L containing α and l ideal \mathfrak{pI} a prime ideal algebraic element μ </div>
DESCRIPTION	<p>Given an prime ideal $\mathfrak{p} \subset \mathfrak{o}$ and an integral element $\mu \in \mathfrak{o} \setminus \mathfrak{p}$, such that $\zeta p \in \mathfrak{o}$ for the prime number p dividing \mathfrak{p}. This function will compute the maximal exponent $l \leq p \frac{v_{\mathfrak{p}}(p)}{p-1}$ such that there is an α subject to $\alpha^p \equiv \mu \pmod{\mathfrak{p}^l}$. α and l will be returned.</p> <p>This function is useful mainly in the context of Kummer extensions, where conditions like the above are used to compute the discriminant and the splitting of \mathfrak{p}. This is based on [Dab95, p 50].</p>

EXAMPLE

```

kash> o := OrderCyclotomic(7);;
kash> pI := Factor(7*o)[1][1];
<7, [6, 1, 0, 0, 0, 0]>
kash> mu := Elt(o, [1,2,3,4,5,8]);
[1, 2, 3, 4, 5, 8]
kash> L := OrderKextModularPower(pI, mu);
[ 2, 1 ]
kash> EltIsInIdeal(mu - L[1]^7, pI^L[2]);
true

```


NAME	OrderLLL
PURPOSE	Creates an order with LLL-reduced basis.
SYNTAX	<pre>o1 := OrderLLL(0); order o1 order o</pre>
DESCRIPTION	Creates an overorder $\mathfrak{o} 1$ of the given order \mathfrak{o} via transformation, so that the basis of the new order $\mathfrak{o} 1$ is LLL-reduced.

EXAMPLE

```
kash> o := Order (Poly(Zx,[1,0,73,-280,-2399]));;
kash> O1 := OrderMaximal (o);
      F[1]
      |
      F[2]
      /
      /
Q
F [ 1]      Given by transformation matrix
F [ 2]      x^4 + 73*x^2 - 280*x - 2399
Discriminant: -997975

kash> O2 := OrderSimplify (OrderLLL (O1));
      F[1]
      |
      F[2]
      /
      /
Q
F [ 1]      Given by transformation matrix
F [ 2]      x^4 + 73*x^2 - 280*x - 2399
Discriminant: -997975

kash> OrderBasis (O1);
[ 1, [0, 1, 0, 0], [1, 1, 1, 0] / 2, [414, 857, 795, 1] / 1646 ]
kash> OrderBasis (O2);
[ 1, [409, -34, 28, -1] / 1646, [-4045, 1775, 39, 28] / 1646,
  [2055, -1680, 28, -1] / 1646 ]
```

NAME	OrderMaximal
PURPOSE	Returns the maximal overorder of an order.
SYNTAX	<pre> O := OrderMaximal(def); O := OrderMaximal(def, str); order O see below def up to 4 optional strings str </pre>
DESCRIPTION	<p>The OrderMaximal function returns the maximal overorder of the order \mathfrak{o} by using the algorithm and techniques which are specified in the optional strings. The argument <i>def</i> may be the order \mathfrak{o} or one of the argument lists of the function Order which defines the order \mathfrak{o}.</p> <p>The optional argument list <i>str</i> (1–4 strings) is used to control the algorithm: You can choose the "Round2" XOR "Round4" algorithm for the computation with additional techniques "Split" to use the algebra splitting due to the Round4-Algorithm or "NoSplit" to avoid this. The reduced discriminant can be useful to factorize the discriminant of the order \mathfrak{o}, which is necessary if you want to compute the maximal order. "RD" uses the reduced discriminant and "NoRD" avoids the reduced discriminant. The Dedekind-test allows a short-cut if \mathfrak{o} is an equation order, a simple test if \mathfrak{o} is already p-maximal. Using the Round2-Algorithm, we implemented the extended Dedekind-test which computes an overorder of the equation order if it is not p-maximal. If you want the Dedekind-test use "Dedekind", otherwise type "NoDedekind". A special order of these strings is not necessary, but you can use just one of the above mentioned devices in each string. If you leave one of them unspecified, the algorithm tries to find the best solution (with respect to the running time). At present we use "Round4" and "Split" only for absolute equation orders and RD only for equation orders, so that these devices are changed to "Round2", "NoSplit" and "NoRD" in the other cases.</p> <p>If you want further information about the Round2-Algorithm and the Dedekind-test, see [Fri97], about the Round4-Algorithm, algebra-splitting and the reduced-discriminant see [Bai96].</p>
SEE ALSO	OrderPMaximal , Order ,
EXAMPLE	<p>Compute the maximal order of $x^4 + 73x^2 - 280x - 2399$ with the Round4-Algorithm</p> <pre> kash> f := Poly (Zx,[1,0, 73, -280, -2399]); x^4 + 73*x^2 - 280*x - 2399 </pre>

```

kash> O := OrderMaximal (f, "Round4");
F[1]
  |
  F[2]
  /
  /
Q
F [ 1]      Given by transformation matrix
F [ 2]       $x^4 + 73x^2 - 280x - 2399$ 
Discriminant: -997975

```

NAME	OrderMerge
PURPOSE	Computes extension fields which contain the given ones.
SYNTAX	$L := \text{OrderMerge}(o1, o2);$ <pre> list L return value order o1 order o2 </pre>
DESCRIPTION	<p>This function computes fields which contain the given ones. It is possible to get more than one field (see example). The output is a list of generating polynomials for these fields.</p> <p>Let αi, βj be the zeroes of the defining polynomials of $o 1$ and $o 2$. After a substitution of the form $\beta i \mapsto \beta i + s$, $s \in \mathbb{N}$ appropriate, all $\alpha i + \beta j$ are pairwise distinct. The polynomial $f := \prod i, j(x - \alpha i - \beta j)$ is computed, and the irreducible factors are returned.</p>

EXAMPLE Compute the following splitting field.

```

kash> o:=Order(Z,3,2);
Generating polynomial: x^3 - 2

kash> L:=OrderMerge(o,o);
[ Generating polynomial: x^3 - 54
  , Generating polynomial: x^6 + 108
  ]

```

NAME	OrderMinIdeal
PURPOSE	Computes a non-zero integral ideal of smallest norm.
SYNTAX	<pre>a := OrderMinIdeal (o); ideal a order o</pre>
DESCRIPTION	The <code>OrderMinIdeal</code> function computes an ideal $\mathfrak{a} \neq \{0\}$ of \mathfrak{o} with smallest norm. Note that \mathfrak{a} is always a prime ideal.

EXAMPLE

```
kash> o := Order(Poly(Zx,[1,-1,-6,4,10,-4,-4,1]));
Generating polynomial: x^7 - x^6 - 6*x^5 + 4*x^4 + 10*x^3 - 4*x^2 - 4*x + 1
```

```
kash> a := OrderMinIdeal(o);
```

```
<7, [5, 1, 0, 0, 0, 0, 0]>
```

```
kash> Norm(a);
```

```
7
```

NAME	OrderMinkowski
PURPOSE	Returns the floor of the Minkowski bound of the algebraic number field which is generated by the given order.
SYNTAX	<pre>m := OrderMinkowski(O); integer m order O</pre>
DESCRIPTION	Returns the floor of the Minkowski bound of the algebraic number field which is generated by the maximal order \mathcal{O} . Returns an error if \mathcal{O} is not known to be maximal.
SEE ALSO	OrderBach ,

EXAMPLE Compute the floor of the Minkowski bound of the algebraic number field $\mathbb{Q}(\sqrt[4]{-27})$.

```
kash> o := Order(Z,4,-27);
Generating polynomial: x^4 + 27

kash> O := OrderMaximal(o);
F[1]
|
F[2]
/
/
Q
F [ 1]      Given by transformation matrix
F [ 2]      x^4 + 27
Discriminant: 432

kash> OrderMinkowski(O);
3
```

NAME	OrderNormEquation
PURPOSE	Solves a (relative) norm equation.
SYNTAX	$L := \text{OrderNormEquation}(o, a [,n "all" [, "exact" "abs" "ineq"]]);$ <div style="display: flex; justify-content: space-between; margin-top: 10px;"> <div> $list$ int element of the coefficient ring int </div> <div> L a n </div> </div>
DESCRIPTION	<p>Searches for elements $\alpha \in \mathfrak{o}$ with norm a. The third argument specifies the number of different non associated solutions. <code>OrderNormEquation</code> tries to find n, all or, if unspecified, 1 solution.</p> <p>The default for the fourth argument is <code>"exact"</code>: the function solves the above mentioned equation. If <code>"abs"</code> is given, the norm equation $\text{Norm}(\alpha) = \epsilon a$ is solved, where ϵ is a torsion unit of the coefficient ring of the given order. The <code>"ineq"</code>-Option is supported only in the absolute case, here the norm equation $0 \leq \text{Norm}(\alpha) \leq a$ is solved.</p> <p>Remember that solving relative norm equations may take quite a long time. Due to the used algorithm it is faster to call the <code>OrderNormEquation</code> function using the <code>"abs"</code>-Option than using the <code>"exact"</code>-Option or the default value. For further information about the algorithms we refer the reader to [Fin84] for the absolute case and to [Fie97] otherwise.</p>
SEE ALSO	Solve ,

EXAMPLE First we solve an absolute norm equation in $\mathbb{Z}[\sqrt[3]{5}]$.

```
kash> o := Order(Z, 3, 5);
Generating polynomial: x^3 - 5

kash> L := OrderNormEquation(o, 4);
[ [29, 17, 10] ]
kash> L := OrderNormEquation(o, 4, "all");
[ [29, 17, 10], [-1, -1, 1] ]
kash> EltNorm(L[1]);
4
kash> EltNorm(L[2]);
4
```

Now a relative norm equation is solved in the maximal order of $\mathbb{Q}(\sqrt[3]{5}, \sqrt{2})$.

```
kash> O := Order(o, 2, 2);;  
kash> EltNorm(Elt(0, [[1, 2, 3], [3, 2, 2]]));  
[-37, -15, -22]  
kash> a:=OrderNormEquation(0, Elt(o, [-37, -15, -22]), 1, "abs");  
[ [[-5, -2, -1], [2, 0, -1]] ]  
kash> EltNorm(a[1]);  
[37, 15, 22]
```


NAME	OrderPMaximal
PURPOSE	Computes the p -maximal overorder of an order.
SYNTAX	<pre> op := OrderPMaximal(o,p,str); op := OrderPMaximal(o,p,b,str); order op order o rational prime or prime ideal p integer b up to 3 optional strings str </pre>
DESCRIPTION	<p>Let \mathfrak{o} be an arbitrary order and let p be a prime (in the absolute case: a rational prime number, in the relative case a prime ideal). OrderPMaximal computes the p-maximal overorder \mathfrak{o}_p of \mathfrak{o} subject to the constraint $\nu_p(\mathfrak{o}_p : \mathfrak{o}) \leq b$. If the third argument b is omitted, its default value is ∞.</p> <p>You can control the algorithm and additional techniques using the optional argument list str (1–3 strings) with the values "Round2" XOR "Round4", "Split" XOR "NoSplit", "Dedekind" XOR "NoDedekind". Their behavior is explained in the description of OrderMaximal.</p> <p>A special order of these strings is not necessary, but you can use just one of these devices in each string. If you leave one of them unspecified, the algorithm tries to find the best solution (with respect to the running time).</p> <p>If you want further information about the Round2-Algorithm and the Dedekind-test, see [Fri97], about the Round4-Algorithm and algebra-splitting see [Bai96].</p>
SEE ALSO	OrderMaximal , Order ,

EXAMPLE Compute a maximal order by computing p -maximal orders:

```

kash> o := Order (Poly (Zx,[1,0, 73, -280, -2399]));
Generating polynomial: x^4 + 73*x^2 - 280*x - 2399

kash> Factor (Abs (Disc (o)));
[ [ 2, 4 ], [ 5, 2 ], [ 11, 1 ], [ 19, 1 ], [ 191, 1 ], [ 823, 2 ] ]
kash> O := OrderPMaximal (o, 2, 4,"Round4");;
kash> O := OrderPMaximal (O, 5, 2,"Round2", "NoSplit");;
kash> O := OrderPMaximal (O, 823);;
kash> Disc (O);
-997975

```

We test our results:

```
kash> 0 := OrderMaximal (0);;  
kash> Disc (0);  
-997975
```

NAME	OrderPoly
PURPOSE	Returns the defining polynomial of the associated equation order
SYNTAX	<div><div><div>f := OrderPoly(P,o);</div><div>f := OrderPoly(o);</div></div><div><div>polynomial</div><div>polynomial algebra</div><div>order algebraic function field order</div></div><div><div>f</div><div>P</div><div>o</div></div></div>
DESCRIPTION	<div>Returns the polynomial defining the equation order of \mathfrak{o} represented in the polynomial algebra P.</div> <div>The polynomial algebra P has to be defined over the coefficient ring of \mathfrak{o}. If the polynomial algebra P is omitted, the polynomial f is represented over the coefficient ring of the order \mathfrak{o}.</div> <div>For an order of an algebraic function field, the polynomial f is an element of $k[T, y]$ or $\mathcal{O}_\infty[y]$.</div> <div>IDEAL order; equation; defining polynomial equation; order; defining polynomial polynomial; defining; of an equation order</div>
SEE ALSO	OrderEquationOrder , AlffOrderDeg , Alff ,
EXAMPLE	

NAME	OrderPolyHenselLift
PURPOSE	Calculates a p -adic factorization of the defining polynomial of an order.
SYNTAX	<pre>L := OrderPolyHenselLift(o, d, p, k);</pre> <pre>list L</pre> <pre>order o</pre> <pre>integer d</pre> <pre>prime p</pre> <pre>integer k</pre>
DESCRIPTION	Let o be an order defined by a polynomial $f \in \mathbb{Z}[x]$ and let $p \in \mathbb{Z}$ be a prime which does not divide the discriminant of f . This function computes the factorization of f over the unramified extension of $\mathbb{Q} p$ of degree d . It returns a list containing the factors modulo p^{2^k} .

EXAMPLE

```
kash> o :=Order(Poly(Zx,[1,0,-4,0,1]));
```

```
Generating polynomial: x^4 - 4*x^2 + 1
```

```
kash> P :=OrderPolyHenselLift(o,2,5,2);
```

```
[ x + [258, 312], x + [-258, 312], x + [-258, -312], x + [258, -312] ]
```

```
kash> P[1]*P[2]*P[3]*P[4];
```

```
x^4 - 522504*x^2 + 16415759376
```

```
kash> P :=OrderPolyHenselLift(o,3,7,2);
```

```
[ x^2 - 235*x - 1, x^2 + 235*x - 1 ]
```

```
kash> P[1]*P[2];
```

```
x^4 - 55227*x^2 + 1
```

NAME	OrderPrec
PURPOSE	Sets or returns the internal precision for calculations in orders.
SYNTAX	<pre> P := OrderPrec(p); P := OrderPrec(); L := OrderPrec(o,p); L := OrderPrec(o,p,u); L := OrderPrec(o); small integer P small integer p order o small integer u list L precisions of both real rings belonging to o </pre>
DESCRIPTION	<p>To every order in KASH internally there belong two "real rings". One of these rings is used for computations with (real) approximations to the elements of \mathfrak{o}, the other ring is used for computations in which units and their logarithms are involved. When a <i>new</i> order is created, both rings obtain the same precision which is taken from an internal KANT constant named <code>Real_Std_Prec</code>.</p> <p>Note however, that all precisions are automatically enlarged to a multiple of 4.</p> <p>OrderPrec(p) Sets the internal KANT constant <code>Real_Std_Prec</code> and returns the new value of this constant. Already existing orders are not affected.</p> <p>OrderPrec() Returns the current value of <code>Real_Std_Prec</code>.</p> <p>OrderPrec(o,p) The precisions of both "real rings" in the order \mathfrak{o} are set to p. A list L containing the precisions of both "real rings" of \mathfrak{o} is returned.</p> <p>OrderPrec(o,p,u) The precision for the first "real ring" of the order \mathfrak{o} is set to p and the precision of the second is set to u. A list L containing the precisions of both "real rings" of \mathfrak{o} is returned.</p> <p>OrderPrec(o) Returns a list L containing the precisions of both "real rings" of \mathfrak{o}.</p>
SEE ALSO	Prec ,

EXAMPLE We compute the regulator of the maximal order \mathcal{O} in $\mathbb{Q}(\sqrt[4]{2})$. First, we use the default precision of 50 digits. Then we increase the precision of the order and print the regulator again. Remark that we must use the `Prec` function to increase the precision in order to print all 60 digits.

```
kash> OrderPrec();
50
kash> O := OrderMaximal(Order(Z,4,2));
Generating polynomial: x^4 - 2
Discriminant: -2048

kash> OrderPrec(O);
[ 52, 52 ]
kash> OrderUnitsFund(O);
[ [1, 0, 1, 0], [1, -1, 0, 0] ]
kash> OrderReg(O);
2.15800131645680564826065544584339217422724449652
kash> OrderPrec(O,60);
[ 60, 60 ]
kash> OrderReg(O);
2.158001316456805648260655445843392174227244496522
kash> Prec(60);
60
kash> OrderReg(O);
2.15800131645680564826065544584339217422724449652236075411
```

NAME	OrderPrintFlags																																											
PURPOSE	Set or get Flags to reduce or extend the output level of print_order.																																											
SYNTAX	<pre>OrderPrintFlags(a); a := OrderPrintFlags(); record a</pre>																																											
DESCRIPTION	<p>OrderPrint can be used to set or get Flags, which are saved in a record. The record entries can be the following</p> <table> <tr> <th>name</th><th>type</th><th>description</th></tr> <tr> <td>disc</td><td>int</td><td>information about discriminant</td></tr> <tr> <td>reg</td><td>int</td><td>information about regulator</td></tr> <tr> <td>fac_disc</td><td>int</td><td>information about factorizing</td></tr> <tr> <td>unit</td><td>int</td><td>information about units</td></tr> <tr> <td>index</td><td>int</td><td>information about index</td></tr> <tr> <td>class_group</td><td>bool</td><td>information about class group</td></tr> <tr> <td>class</td><td>bool</td><td>information about class number</td></tr> <tr> <td>trafo</td><td>bool</td><td>information about transform. Matrix</td></tr> <tr> <td>trafo_inv</td><td>bool</td><td>information about inverse Trafomatrix</td></tr> <tr> <td>trafo_den</td><td>bool</td><td>information about Matrix denominator</td></tr> <tr> <td>galois</td><td>bool</td><td>information about galois groups</td></tr> <tr> <td>sign</td><td>bool</td><td>information about sign</td></tr> <tr> <td>autom</td><td>bool</td><td>information about automorphisms</td></tr> </table> <p>where</p> <p>number = -1: maximal output number = 0: no output number = \langle any number \rangle: prints if length of output string is less or equal then number, otherwise prints “Known”</p> <p>and</p> <p>bool = true: prints if known bool = false: prints nothing</p> <p>To get the settings call OrderPrintFlags();</p>		name	type	description	disc	int	information about discriminant	reg	int	information about regulator	fac_disc	int	information about factorizing	unit	int	information about units	index	int	information about index	class_group	bool	information about class group	class	bool	information about class number	trafo	bool	information about transform. Matrix	trafo_inv	bool	information about inverse Trafomatrix	trafo_den	bool	information about Matrix denominator	galois	bool	information about galois groups	sign	bool	information about sign	autom	bool	information about automorphisms
name	type	description																																										
disc	int	information about discriminant																																										
reg	int	information about regulator																																										
fac_disc	int	information about factorizing																																										
unit	int	information about units																																										
index	int	information about index																																										
class_group	bool	information about class group																																										
class	bool	information about class number																																										
trafo	bool	information about transform. Matrix																																										
trafo_inv	bool	information about inverse Trafomatrix																																										
trafo_den	bool	information about Matrix denominator																																										
galois	bool	information about galois groups																																										
sign	bool	information about sign																																										
autom	bool	information about automorphisms																																										
EXAMPLE	Set output level of unit and signature to 0 (prints nothing) even if unit or signature are known.																																											

```
kash> OrderPrintFlags(rec(unit:=0,sign:=false));
```

```
kash> a:=OrderPrintFlags();
```

```
rec(
  disc := -1,
  reg := -1,
  fac_disc := 0,
  unit := 0,
  index := 0,
  class_group := true,
  class := false,
  trafo_inv := false,
  trafo_den := false,
  trafo := false,
  galois := false,
  sign := false,
  autom := false,
  coef_ideals := true )
```

```
kash> a.unit:=0;
```

```
0
```

```
kash> a.sign:=false;
```

```
false
```

```
kash> OrderPrintFlags(a);
```

```
kash> q:=Order(Z, 2, 2^15);; OrderUnitsFund(q);;
```

```
kash> o:= OrderMaximal(Z, 4, 2);; O := Order(o, 5, 3);;
```

```
kash> OrderSig(O);;
```

```
kash> q;
```

Generating polynomial: $x^2 - 32768$

Discriminant: 131072

Regulator: 112.815819138501507229773993597413415555604522017489

```
kash> O;
```

```
  F[1]
```

```
  /
  /
```

```
  E1[1]
```

```
  /
  /
```

```
Q
```

```
F [ 1]      x^5 - 3
```

```
E 1[ 1]     x^4 - 2
```



```

kash> OrderPrintFlags(rec(unit:=-1,sign:=true));
kash> q;
Generating polynomial:  $x^2 - 32768$ 
Discriminant: 131072
Regulator: 112.815819138501507229773993597413415555604522017489
Fundamental unit:
[4946041176255201878775086487573351061418968498177, -2732327543560891540641290\
5094301272808109095411]
Signature: [2, 0]

```

```

kash> 0;
      F[1]
      /
      /
      E1[1]
      /
      /
Q
F  [ 1]       $x^5 - 3$ 
E 1[ 1]       $x^4 - 2$ 
Signature: [ [1, 2]  [1, 2]  [5, 0] ]

```

```

kash> OrderPrintFlags(rec(unit:=3));
kash> q;
Generating polynomial:  $x^2 - 32768$ 
Discriminant: 131072
Regulator: 112.815819138501507229773993597413415555604522017489
Fundamental unit: Known
Signature: [2, 0]

```

NAME	OrderReg
PURPOSE	Returns the regulator of the <i>current</i> maximal system of independent units.
SYNTAX	<pre> x := OrderReg(o); x := OrderReg(o,"classgroup"); x := OrderReg(o, reg); real x order o real reg </pre>
DESCRIPTION	<p>Let \mathfrak{o} be an absolute order with unit rank $r \geq 1$.</p> <p>OrderReg(o) Let $\mathcal{U} = \{\varepsilon_1, \dots, \varepsilon_k\}$ ($0 \leq k \leq r$) be the current system of independent units which is stored in \mathfrak{o}. If k equals r we say that the system \mathcal{U} is <i>maximal</i>. In this case the regulator of the system \mathcal{U} can be computed by the OrderReg function.</p> <p>The regulator $\text{Reg}_{\mathfrak{o}}$ of the order \mathfrak{o} is defined as the regulator of a system of fundamental units of the order \mathfrak{o}. To obtain $\text{Reg}_{\mathfrak{o}}$, a system of fundamental units for \mathfrak{o} must be computed first, which can be done by the OrderUnitsFund function.</p> <p>OrderReg(o,"classgroup") The OrderReg function returns the regulator of the units found by class group computations using the "euler" option.</p> <p>OrderReg(o, reg) Sets the regulator of the order \mathfrak{o} to the value of <i>reg</i> and returns it.</p> <p>SEE ALSO OrderRegLowBound, OrderUnitsFund, OrderUnitsIndep,</p>

EXAMPLE Let \mathcal{O} denote the maximal order of $\mathbb{Q}(\rho)$, where $\rho^4 + 73\rho^2 - 280\rho - 2399 = 0$. First, we compute the regulator of a maximal system of independent units in \mathcal{O} .

```

kash> O := OrderMaximal (Poly(Zx,[1,0,73,-280,-2399]));
      F[1]
      |
      F[2]
      /
      /
      Q

```

```
F [ 1]      Given by transformation matrix
F [ 2]       $x^4 + 73x^2 - 280x - 2399$ 
Discriminant: -997975
```

```
kash> OrderUnitsIndep (0);
[ [0, 0, 1, -1], [109589, -13889, 743946, -766419] ]
kash> OrderReg (0);
12.399824579954190488548880525089080810892337788058
```

Next we compute the regulator of a fundamental system of units in \mathcal{O} .

```
kash> OrderUnitsFund (0);
[ [-1, 0, -1, 1], [-253011, -13889, 519847, -542320] ]
kash> OrderReg (0);
12.39982457995419048854888052508908081089233778807
```

From this result we find that the system of independent units which we had computed in the first step was fundamental.

Finally, we set the regulator of the maximal order to 12.4:

```
kash> OrderReg (0, 12.4);
12.4
kash> OrderReg (0);
12.4
```

NAME	OrderRegLowBound
PURPOSE	Returns a lower bound for the regulator of the given order.
SYNTAX	<pre> x := OrderRegLowBound(o); x := OrderRegLowBound(o, reg); real x order o real reg </pre>
DESCRIPTION	<p>Let \mathfrak{o} be an absolute order.</p> <p><code>OrderRegLowBound(o)</code> computes a lower bound for the regulator of the order \mathfrak{o}.</p> <p><code>OrderRegLowBound(o, reg)</code> sets the regulator of the order \mathfrak{o} to the value of <code>reg</code> and returns it.</p>
SEE ALSO	OrderReg ,

EXAMPLE Compute a lower bound for the regulator of $\mathbb{Q}(\rho)$ where $\rho^4 + 73\rho^2 - 280\rho - 2399 = 0$:

```

kash> O := OrderMaximal (Order (Poly(Zx,[1,0,73,-280,-2399])));
  F[1]
  |
  F[2]
  /
  /
Q
F [ 1]      Given by transformation matrix
F [ 2]      x^4 + 73*x^2 - 280*x - 2399
Discriminant: -997975

```

```

kash> OrderRegLowBound (O);
2.590616643223093069474635679775415624303102251639

```

Set the lower bound to 2.0:

```

kash> OrderRegLowBound (O, 2.0);
2
kash> OrderRegLowBound (O);
2

```

NAME	OrderRelNormEq
PURPOSE	Calculates an element of given norm.
SYNTAX	<pre>elt := OrderRelNormEq(O,n [, "true"]);</pre> <p> order 0 list elt list of elements in O element n element in coefficient ring of O </p>
DESCRIPTION	<p>Let o be the coefficient order of O and O a simple relative extension. This means that o is an absolute extension. (Thus not \mathbb{Z}!) Let f and F be the quotient fields of o and O. This function computes for a given $n \in o$ an element $elt \in O$ with $N^{F/f}(elt) = n$.</p> <p>If no such element exists an empty list is returned. If "true" is set we try to find an integral element. Sometimes two solutions are returned. That is if the reduced version is different from the non-reduced version.</p>

EXAMPLE

```
kash> o:= Order(Z, 2, 2);
Generating polynomial: x^2 - 2

kash> O:= Order(o, 2, 3);
      F[1]
      /
      /
      E1[1]
      /
      /
      Q
F  [ 1]      x^2 - 3
E 1[ 1]      x^2 - 2

kash> n := Elt(o, [-3856938, 1576756]);
[-3856938, 1576756]
kash> elt := OrderRelNormEq(O,n);
[ [[-3961, -4022], [3679, 1372]], [[-449, 140], [361, -786]] ]
```

NAME	OrderRelUnits
PURPOSE	Computes the relative units.
SYNTAX	<pre>L := OrderRelUnits(o [,S I] [,"raw"]);</pre> <p> list L of L1, T, hs or L2, hs list L1 of algebraic integers list L2 of S-units matrix T transformation matrix integer hs S-class number order o list S of pairwise distinct prime ideals. ideal I </p>
DESCRIPTION	<p>The OrderRelUnits function returns a basis of the S-units modulo split into torsion units and other S-units. You may specify S either as a list of prime ideals or as an ideal \mathfrak{J}. In this case S consists of all primes dividing \mathfrak{J}. Omitting the argument S means to choose for S the ideal $1*o$.</p> <p>If the last argument is "raw", this function will return some algebraic elements $\alpha 1, \dots, \alpha s$ and two transformation matrices T and U such that</p> $(\alpha 1, \dots, \alpha s) T$ <p>interpreted as power products gives a basis of the relative torsion units and</p> $(\alpha 1, \dots, \alpha s) U$ <p>interpreted as power products gives a basis of the relative S-units.</p> <p>In both cases it will also return the cardinality of the subgroup of the class group which is generated by the prime ideals of S.</p>

SEE ALSO **OrderSUnits**,

EXAMPLE We have for example:

```
kash> o:= Order(Poly(Zx,[1,1,1,1,1]));
Generating polynomial: x^4 + x^3 + x^2 + x + 1

kash> zeta:= Elt(o,[0,1,0,0]);
[0, 1, 0, 0]
kash> ox:=PolyAlg(o);
Univariate Polynomial Ring in x over Generating polynomial: x^4 + x^3 + x^2 + \
x + 1
```

```

kash> O:= Order(Poly(ox,[1,0,-zeta,0,zeta^2]));
      F[1]
      /
      /
      E1[1]
      /
      /
Q
F  [ 1]      x^4 + [0, -1, 0, 0]*x^2 + [0, 0, 1, 0]
E 1[ 1]      x^4 + x^3 + x^2 + x + 1

kash> I:=31*o;
<31>
kash> m:=OrderRelUnits(O,I);
[ [ [0, [-2, -2, -1, -1], 0, [0, 1, 0, -1]],
    [0, [-1, -1, -1, 0], 0, [-1, -1, -2, -1]],
    [[0, -1, 0, 0], [0, 0, -1, -1], [1, 1, 1, 0], [0, 1, 1, 1]],
    [[0, -1, 0, 0], [-1, 0, -1, -1], [1, 0, 0, 1], [0, 0, 0, -1]],
    [[-1, -1, -1, -1], [0, 1, 0, 0], [0, 0, 0, -1], -1],
    [[0, 1, 0, 0], [1, 1, 0, 1], 0, [0, 0, 0, 1]],
    [[-10, -19, -24, -44], 0, [-14, -8, 16, -12], 0] / 31,
    [[17, 10, 22, -3], 0, [-4, -37, -7, -23], 0] / 31,
    [[-66, -67, -137, -77], 0, [6, -39, -58, -117], 0] / 31,
    [[-19, -26, 84, 56], 0, [27, -43, -59, -91], 0] / 31 ],
[ [[0, 0, 1, 1], 0, [-1, -1, -1, 0], 0],
  [0, [0, -1, -1, 1], 0, [2, 1, 1, 1]],
  [[-49, -38, -43, -52], 0, [27, -5, 18, -15], 0] / 31,
  [[-2, -18, 71, 8], 0, [34, -4, 33, 81], 0] / 31,
  [[-13, 69, -19, -103], 0, [66, 67, 137, -16], 0] / 31 ] ]

```

NAME	OrderRelativeOrder
PURPOSE	Computes a relative representation of the given orders.
SYNTAX	<pre>Or := OrderRelativeOrder(0, o);</pre> <p> order Or order 0 order o </p>
DESCRIPTION	This function computes a relative representation for 0 and o where o is a suborder of 0. The homomorphisms are installed.
SEE ALSO	OrderAbs ,
EXAMPLE	A simple example

```
kash> 0:=Order(x^4-4*x^2+1);
Generating polynomial: x^4 - 4*x^2 + 1
```

```
kash> o:=OrderSubfield(0)[2];
Generating polynomial: x^2 + 4*x + 1
```

```
kash> Or:=OrderRelativeOrder(0, o);
F[1]
```

```

      /
     /
    E1[1]
   /
  /
 Q
F [ 1]      x^2 + [0, 1]
E 1[ 1]     x^2 + 4*x + 1
```


NAME	OrderSUnits
PURPOSE	Computes the S -unit group.
SYNTAX	$L := \text{OrderSUnits}(o \text{ [, } S \text{ } I] \text{ [, "raw"]});$ <p> <code>list</code> <code>L</code> of <code>L1</code>, <code>T</code>, <code>hs</code> or <code>L2</code>, <code>hs</code> <code>list</code> <code>L1</code> of algebraic integers <code>list</code> <code>L2</code> of S-units <code>matrix</code> <code>T</code> transformation matrix <code>integer</code> <code>hs</code> S-class number <code>order</code> <code>o</code> <code>list</code> <code>S</code> of pairwise distinct prime ideals. <code>ideal</code> <code>I</code> </p>
DESCRIPTION	<p>The OrderSUnits function returns a basis of the S-units modulo torsion units. You may specify S either as a list of prime ideals or as an ideal \mathfrak{J}. In this case S consists of all primes dividing \mathfrak{J}. Omitting the argument S means to choose for S the factor basis used during class group computation. The first elements of this basis are fundamental units.</p> <p>If the last argument is "raw", this function will return some algebraic integers $\alpha 1, \dots, \alpha s$ and a transformation matrix T such that</p> $(\alpha 1, \dots, \alpha s) T$ <p>interpreted as power products gives a basis of the S-unit group modulo torsion units.</p> <p>In both cases it will also return the cardinality of the subgroup of the class group which is generated by the prime ideals of S.</p> <p>The first basis elements are not fully proven to be fundamental units. This may be verified using OrderClassGroupCheck.</p>

SEE ALSO **OrderClassGroup**, **OrderClassGroupCheck**, **OrderSUnitsPositive**,

EXAMPLE For $\mathbb{Q}(\sqrt[3]{333})$ we have for example:

```
kash> o := OrderMaximal(Z, 3, 333);;
kash> OrderClassGroup(o, 500, euler, fast);
[ 3, [ 3 ] ]
kash> OrderSUnits(o, 7*11*13*o);
[ [ [-7309383190, 288532187, 331545960], -7, [256, 37, 16], [-13, -12, 6],
    [3422, -161, -144], [4852, 700, 303], [16, 0, -1] ], 3 ]
```

NAME	OrderSUnitsPositive
PURPOSE	Returns a list of positive S-units
SYNTAX	$P := \text{OrderSUnitsPositive}(L);$ list P list L
DESCRIPTION	Given a list of independent S-units this function returns a list consisting of a list of positive S-units and the index in the original S-units.
SEE ALSO	OrderSUnits , OrderUnitsFund , EltMinkowski ,
EXAMPLE	

```

kash> o := OrderMaximal(x^3 + x^2 - 6*x - 1);
Generating polynomial: x^3 + x^2 - 6*x - 1
Discriminant: 985

kash> OrderSig(o);
[ 3, 0 ]
kash> L := OrderUnitsFund(o);
[ [0, 1, 0], [2, -1, 0] ]
kash> List(L, EltMinkowski);
[ [ -0.1629618567775299422224210388659244025428534549274, -2.93080160017275807\
0023094586767733368753328418998, 2.0937634569502880122455156256336577712961818\
73926],
  [ 2.162961856777529942222421038865924402542853454927, 4.93080160017275807002\
3094586767733368753328418998, -0.093763456950288012245515625633657771296181873\
926] ]
kash> P := OrderSUnitsPositive(L);
[ [ [0, 1, 0], [2, -1, 0] ], 1 ]
kash> List(P[1], EltMinkowski);
[ [ -0.1629618567775299422224210388659244025428534549274, -2.93080160017275807\
0023094586767733368753328418998, 2.0937634569502880122455156256336577712961818\
73926],
  [ 2.162961856777529942222421038865924402542853454927, 4.93080160017275807002\
3094586767733368753328418998, -0.093763456950288012245515625633657771296181873\
926] ]

```

NAME	OrderSetMaximal
PURPOSE	Sets the <code>OrderIsMaximal</code> flag in the given order.
SYNTAX	<pre>OrderSetMaximal(o); OrderSetMaximal(o,flag);</pre> <p> <code>order</code> <code>o</code> <code>optional boolean</code> <code>flag</code> </p>
DESCRIPTION	Sets the <code>OrderIsMaximal</code> flag in the order <code>o</code> to <code>flag</code> . If <code>flag</code> is not given, the order is assumed to be maximal.
SEE ALSO	OrderIsMaximal ,
EXAMPLE	Define the order $\mathbb{Z}[\frac{1+\sqrt{5}}{2}]$ as maximal.

```
kash> o := Order (Poly (Zx,[1,0,-5]));
Generating polynomial: x^2 - 5
```

```
kash> o1 := Order (o,[Elt(o,1),Elt (o,[1,1]/2)]);
F[1]
|
F[2]
/
/
Q
F [ 1]      Given by transformation matrix
F [ 2]      x^2 - 5
```

```
kash> OrderIsMaximal(o1);
false
kash> OrderSetMaximal (o1);
kash> OrderIsMaximal(o1);
true
```

NAME	OrderSetTorsionUnit							
PURPOSE	Sets a generator of the torsion unit group in the given order.							
SYNTAX	<div>OrderSetTorsionUnit(o,alpha,r);</div> <table><tr><td>order</td><td>o</td></tr><tr><td>algebraic element</td><td>alpha</td></tr><tr><td>integer</td><td>r</td><td>rank</td></tr></table>	order	o	algebraic element	alpha	integer	r	rank
order	o							
algebraic element	alpha							
integer	r	rank						
DESCRIPTION	Sets the generator for the torsion units in the order \mathfrak{o} to be α of exponent r , no check is done, so be very careful.							
SEE ALSO	OrderTorsionUnit , OrderTorsionUnitRank ,							
EXAMPLE								

```
kash> o := Order(Poly(Zx,[1,1,1,1,1]));
Generating polynomial: x^4 + x^3 + x^2 + x + 1

kash> zeta := Elt(o,[0,-1,0,0]);
[0, -1, 0, 0]
kash> OrderSetTorsionUnit(o,zeta,10);
```

NAME	OrderShort
PURPOSE	Tries to find a "better" primitive polynomial for the given order.
SYNTAX	<pre>o1 := OrderShort(o); o1 := OrderShort(o, modus); o1 := OrderShort(o, modus, iterations);</pre> <pre>order o1 order o integer modus integer iterations</pre>
DESCRIPTION	<p>Tries to find a polynomial for which the corresponding equation order is of smaller index. If modus is given (and not zero), the search is for a polynomial for which modus does not divide the index of the equation order. When using on large fields, this routine may take a long time ($O(2^{deg})$). You can specify an upper bound for the number of iterations with the optional third argument. If an invalid or no argument is given, this bound is set to 15. Note that this function only tries to find a better polynomial, it may fail, even if there is a better one.</p>
SEE ALSO	Order ,
EXAMPLE	This function is useful, if you merge two fields. The final equation order is a maximal order.

```
kash> o1 := Order(Z,2,2);
Generating polynomial: x^2 - 2
```

```
kash> o2:= Order(o1,2,3);
```

```
      F[1]
      /
      /
E1[1]
/
/
Q
F [ 1]      x^2 - 3
E 1[ 1]      x^2 - 2
```

```
kash> o3:=OrderAbs(o2);
Generating polynomial: x^4 - 10*x^2 + 1
```

```
kash> o4:=OrderShort(o3);  
Generating polynomial:  $x^4 - 4x^2 + 1$   
Discriminant: 2304
```

NAME	OrderShortAbs
PURPOSE	Creates an absolute extension from a simple relative extension and tries to find a "good" primitive polynomial for the given order.
SYNTAX	<pre>Oa := OrderShortAbs(O);</pre> <pre>order Oa</pre> <pre>order 0</pre>
DESCRIPTION	<p>Given a simple relative order \mathcal{O} – the coefficient ring (<code>OrderCoefOrder</code>) of the coefficient ring of \mathcal{O} is \mathbb{Z} – this function computes an absolute equation order $\mathcal{O} a$ with quotient field isomorphic to the quotient field of \mathcal{O} over \mathbb{Q}.</p> <p>At the same time this algorithm tries to find a polynomial for which the corresponding equation order is of small index.</p>
SEE ALSO	OrderShort , OrderAbs ,

EXAMPLE The following example creates the absolute extension of a relative extension:

```
kash> o:=OrderMaximal(x^2 + 19*x + 11);
Generating polynomial: x^2 + 19*x + 11
Discriminant: 317

kash> ox:=PolyAlg(o);
Univariate Polynomial Ring in x over Generating polynomial: x^2 + 19*x + 11
Discriminant: 317

kash> O:=Order(Poly(ox,[1,-18,-6]));
      F[1]
      /
      /
      E1[1]
      /
      /
      Q
      F [ 1]      x^2 - 18*x - 6
      E 1[ 1]      x^2 + 19*x + 11

kash> Oa:=OrderShortAbs(O);
F[1]
|
```

```

      F[2]
    /
  /
Q
F [ 1]      Given by transformation matrix
F [ 2]      x^4 - 2*x^3 - 331*x^2 + 332*x - 23

```


NAME	OrderSig
PURPOSE	Returns the signature of the number field defined by the given order.
SYNTAX	<pre>L := OrderSig(o);</pre> <div> list L order o </div>
DESCRIPTION	Computes the signature of the polynomial which defines an equation suborder of the given order <code>o</code> . The two entries of <code>L</code> are the number of real roots and one half of the number of complex roots of the corresponding generating polynomial. For further information see the description of <code>PolySig</code> .
SEE ALSO	<code>PolySig</code> ,

EXAMPLE Compute the signature of $x^4 + 73x^2 - 280x - 2399$.

```
kash> o := Order (Poly(Zx,[1,0, 73, -280, -2399]));
Generating polynomial: x^4 + 73*x^2 - 280*x - 2399
```

```
kash> L := OrderSig (o);
[ 2, 1 ]
```

NAME	OrderSimplify
PURPOSE	Returns the given order in a simplified representation.
SYNTAX	<pre>o1 := OrderSimplify(o); order o1 order o</pre>
DESCRIPTION	The order is simplified in the sense that the number of suborders might be reduced.
SEE ALSO	OrderTransformationMatrix , Order ,
EXAMPLE	Simplification of an order.

```
kash> o := Order (Poly(Zx,[1,0,73,-280, -2399]));
Generating polynomial: x^4 + 73*x^2 - 280*x - 2399
```

```
kash> M1 := Mat (Z, [[2,0,-1,-2],[0,2,-1,0],[0,0,1,0],[0,0,0,1]]);
[ 2  0 -1 -2]
[ 0  2 -1  0]
[ 0  0  1  0]
[ 0  0  0  1]
kash> M2 := Mat (Z, [[823,0,0,-409],[0,823,0,-789],[0,0,823,-28],[0,0,0,1]]);
[ 823    0    0 -409]
[   0  823    0 -789]
[   0    0  823 -28]
[   0    0    0   1]
kash> o := Order (o, M1, 2);
  F[1]
  |
  F[2]
  /
  /
Q
F [ 1]      Given by transformation matrix
F [ 2]      x^4 + 73*x^2 - 280*x - 2399
```

```
kash> o := Order (o, M2, 823);
  F[1]
  |
```

```

      F[2]
      |
      F[3]
      /
      /
Q
F  [ 1]      Given by transformation matrix
F  [ 2]      Given by transformation matrix
F  [ 3]      x^4 + 73*x^2 - 280*x - 2399

```

```
kash> O := OrderSimplify (o);
```

```

F[1]
  |
  F[2]
  /
  /
Q
F  [ 1]      Given by transformation matrix
F  [ 2]      x^4 + 73*x^2 - 280*x - 2399

```

NAME	OrderSplittingField
PURPOSE	Computes the normal closure of an algebraic number field.
SYNTAX	$O := \text{OrderSplittingField}(o);$ $\text{order } O$ $\text{order } o$
DESCRIPTION	Let \mathfrak{o} be an absolute order. The <code>OrderSplittingField</code> function computes an equation order of the normal closure of the quotient field of \mathfrak{o} .
EXAMPLE	Computation of an equation order \mathcal{O} of the normal closure of the quotient field of the absolute order \mathfrak{o} and the factorization of the generating polynomial f for \mathfrak{o} in \mathcal{O}

```
kash> o := Order(Z,3,2);
```

```
Generating polynomial: x^3 - 2
```

```
kash> O := OrderSplittingField(o);
```

```
Generating polynomial: x^6 - 3*x^5 + 5*x^3 - 3*x + 1
```

```
Discriminant: -34992
```

```
kash> f := PolyMove(x^3-2, O );
```

```
x^3 - 2
```

```
kash> Factor(f);
```

```
[ [ x + [-1, -1, 1, 0, 0, 0], 1 ], [ x + [5, -5, -10, 3, 5, -2], 1 ],
  [ x + [-4, 6, 9, -3, -5, 2], 1 ] ]
```

NAME	OrderSubOrder
PURPOSE	Returns a suborder of the given order.
SYNTAX	<pre>os := OrderSubOrder(o);</pre> <p> order or boolean os order o </p>
DESCRIPTION	<p>If the given order \mathfrak{o} is an overorder of $\mathfrak{o} s$, for example \mathfrak{o} is defined via transformation (matrix and denominator or matrix and list of ideals; see Order) over $\mathfrak{o} s$, this order will be returned. This function will return false if \mathfrak{o} is an equation order.</p>
SEE ALSO	OrderEquationOrder , OrderCoefOrder ,

EXAMPLE Compute suborder $\mathfrak{o}|s$ of an equation order and of a maximal overorder.

```
kash> o:=Order(Z,2,5);
Generating polynomial: x^2 - 5

kash> OrderSubOrder(o);
false
kash> O:=OrderMaximal(o);
F[1]
|
F[2]
/
/
Q
F [ 1]      Given by transformation matrix
F [ 2]      x^2 - 5
Discriminant: 5

kash> OrderSubOrder(O);
Generating polynomial: x^2 - 5
Discriminant: 20
```

NAME	OrderSubfield									
PURPOSE	Returns all non-trivial subfields of given degree m . If no m is specified, all subfields are calculated.									
SYNTAX	$L := \text{OrderSubfield}(o);$ $L := \text{OrderSubfield}(o, m);$									
	<table><tr><td>list</td><td>L</td><td>list of suborders</td></tr><tr><td>order</td><td>0</td><td>the given order</td></tr><tr><td>small integer</td><td>m</td><td>the prescribed degree of subfields</td></tr></table>	list	L	list of suborders	order	0	the given order	small integer	m	the prescribed degree of subfields
list	L	list of suborders								
order	0	the given order								
small integer	m	the prescribed degree of subfields								
DESCRIPTION	<p>This function calculates all non-trivial subfields of given degree m. If no m is specified, all non-trivial subfields are calculated. The result is a list L, which contains the calculated subfields as orders. The function uses the algorithms described in [Klü95, KP97, Klü97]</p> <p>It is possible that this function returns the same order for two subfields. In this case, these subfields are isomorphic, but not identical. In the example below, the three subfields are isomorphic. If the running time for this function seems too long, the user may want to try <code>OrderSubfieldSub</code>.</p>									
SEE ALSO	OrderSubfieldSub ,									

EXAMPLE Computation of subfields of given degree.

```
kash> O:=Order(Poly(Zx,[1,-3,5,-5,5,-3,1]));
Generating polynomial: x^6 - 3*x^5 + 5*x^4 - 5*x^3 + 5*x^2 - 3*x + 1

kash> L:=OrderSubfield(0,3);
[ Generating polynomial: x^3 - 3*x^2 + 2*x - 1
  , Generating polynomial: x^3 - 2*x^2 + 3*x - 1
  , Generating polynomial: x^3 - 3*x^2 + 2*x - 1
]
kash> elt:=Elt(L[2],[1,1,1]);
[1, 1, 1]
kash> EltMove(elt,0);
[1, 1, 0, -2, 1, 0]
```

NAME	OrderSubfieldSub
PURPOSE	Calculates subfields with specified block systems.
SYNTAX	<pre> L := OrderSubfieldSub(o, p); L := OrderSubfieldSub(o, p, d); L := OrderSubfieldSub(o, p, d, L1); list L order o integer p integer d list L1 </pre>
DESCRIPTION	<p>This function calculates non-trivial subfields of $\mathbb{Q}(o)$ using a special prime number p, which may not divide the discriminant of the equation order of o. If d is specified, only subfields of relative degree d are calculated. $L1$ is a potential block system of the form $[block 1, \dots, block r]$.</p> <p>The computation time is dependent on the cycle decomposition corresponding to the prime p. It is very difficult to choose a “good” prime. In some cases it is better to use this function instead of <code>OrderSubfield</code>.</p>
SEE ALSO	OrderSubfield ,
EXAMPLE	

```

kash> o:=Order(Poly(Zx,[1,0,-4,0,1]));
Generating polynomial: x^4 - 4*x^2 + 1

kash> OrderSubfieldSub(o,5);
[ Generating polynomial: x^2 - 4*x - 2
  , Generating polynomial: x^2 - 2
  , Generating polynomial: x^2 + 4*x + 1
]
kash> o:=Order(Poly(Zx,[1,0,-4,0,1]));
Generating polynomial: x^4 - 4*x^2 + 1

kash> OrderSubfieldSub(o,5);
[ Generating polynomial: x^2 - 4*x - 2
  , Generating polynomial: x^2 - 2
  , Generating polynomial: x^2 + 4*x + 1
]

```

NAME	OrderTorsionUnit
PURPOSE	Returns a generator for the group of torsion units of the given order.
SYNTAX	<pre>u := OrderTorsionUnit(o);</pre> <p>algebraic element u order o</p>
DESCRIPTION	\mathfrak{o} must be an absolute order. <code>OrderTorsionUnit</code> returns an algebraic element u , which is a generator for the group of roots of unity in the order \mathfrak{o} .
SEE ALSO	OrderSetTorsionUnit , OrderTorsionUnitRank ,
EXAMPLE	Computing generators for the groups of roots of unity in an equation order and the corresponding maximal order.

```
kash> o := Order (Poly (Zx,[1,0,-186,0,13097,0,-412704,0, 4946176]));
Generating polynomial: x^8 - 186*x^6 + 13097*x^4 - 412704*x^2 + 4946176
```

```
kash> OrderTorsionUnit (o);
-1
kash> O := OrderMaximal (o);;
kash> OrderTorsionUnit (O);
[15, -80, 10, -9, 10, -20, -51, 102]
```


NAME	OrderTorsionUnitRank
PURPOSE	Returns the number of roots of unity in the given order.
SYNTAX	<pre>r := OrderTorsionUnitRank(o);</pre> <p>integer r number of roots of unity order o</p>
DESCRIPTION	\mathfrak{o} must be an absolute order. <code>OrderTorsionUnitRank</code> returns the number of roots of unity in the order \mathfrak{o} , the rank of the stored generator of the roots of unity in the order \mathfrak{o} .
SEE ALSO	OrderTorsionUnit , OrderSetTorsionUnit ,
EXAMPLE	Compute the number of roots of unity in $\mathbb{Z}[\sqrt{-3}]$ and in $\mathbb{Z}[\frac{1+\sqrt{-3}}{2}]$

```
kash> o := Order(Z,2,-3);
Generating polynomial: x^2 + 3

kash> O := OrderMaximal(o);
  F[1]
  |
  F[2]
  /
  /
Q
F [ 1]      Given by transformation matrix
F [ 2]      x^2 + 3
Discriminant: -3

kash> OrderTorsionUnitRank(o);
2
kash> OrderTorsionUnitRank(O);
6
```

NAME	OrderTraceMat
PURPOSE	Returns the trace matrix of the order.
SYNTAX	$M := \text{OrderTraceMat}(o);$ <div style="margin-left: 100px;"> Matrix M order o </div>
DESCRIPTION	<p>OrderTraceMat returns the the trace matrix of the given order \mathfrak{o}. Let o be given by a (pseudo) basis $\omega 1, \dots, \omega n$ (we ignore the coefficient ideals here). The OrderTraceMat $M = (\text{Tr}(\omega i\omega j))_{i,j}$.</p>

EXAMPLE Extracting the equation order of an arbitrary order.

```
kash> O := OrderMaximal(Poly (Zx,[1,0,-186,0,13097,0,-412704,0,4946176]));
      F[1]
      |
      F[2]
      /
      /
Q
F [ 1]      Given by transformation matrix
F [ 2]      x^8 - 186*x^6 + 13097*x^4 - 412704*x^2 + 4946176
Discriminant: 60050488100625
```

```
kash> M := OrderTraceMat(O);
[      8      0      186      4      1120      657      774      387]
[      0      372      186      2426      186      24657      0      5304]
[     186      186      4294      1306      24471      26711      16664      10984]
[      4      2426      1306      15364      1773      147638      387      32097]
[     1120      186      24471      1773      133394      91728      89382      47343]
[      657      24657      26711      147638      91728      1282603      53410      298835]
[      774      0      16664      387      89382      53410      59566      29783]
[      387      5304      10984      32097      47343      298835      29783      74701]
```

NAME	OrderTransformationMatrix
PURPOSE	Returns a list containing information about the transformation from a suborder of the given order to the given order.
SYNTAX	<pre>L := OrderTransformationMatrix(O);</pre> <p>list L order 0</p>
DESCRIPTION	<p>In the absolute case the list L contains the denominator d and the $n \times n$-matrix T with</p> $(\omega 1, \dots, \omega n) = \frac{1}{d}(a 1, \dots, a n)T,$ <p>where $\omega 1, \dots, \omega n$ is a basis of the given order \mathcal{O} and $a 1, \dots, a n$ is a basis of a suborder of the given order. If the order \mathcal{O} is the equation order, simply $d = 1$ and $T = I n$ are returned.</p> <p>In the relative case the list L contains a list of ideals (1 or n, where n is the relative degree of \mathcal{O}), called the coefficient ideals, and an $n \times n$-matrix T.</p> <p>If we get only one coefficient ideal \mathfrak{a}, the pseudo basis of the order \mathcal{O} is $\mathfrak{o}\omega 1, \dots, \mathfrak{o}\omega n-1, \mathfrak{a}\omega n$, where</p> $(\omega 1, \dots, \omega n) = (a 1, \dots, a n)T,$ <p>$a 1, \dots, a n$ are the basis elements of the pseudo basis of the suborder of \mathcal{O} and \mathfrak{o} is the coefficient order of \mathcal{O}. In this case the relative order is called “free” ($\mathfrak{a} = \mathfrak{o}$) or “almost free” ($\mathfrak{a} \neq \mathfrak{o}$).</p> <p>If we get n coefficient ideals $\mathfrak{a} 1, \dots, \mathfrak{a} n$, the pseudo basis of the order \mathcal{O} is $\mathfrak{a} 1\omega 1, \dots, \mathfrak{a} n\omega n$, with $\omega 1, \dots, \omega n$ as above.</p> <p>A relative equation order \mathcal{O} is “free”, hence <code>OrderTransformationMatrix(O)</code> returns the ideal \mathfrak{o} (the coefficient order) and the $n \times n$ identity matrix.</p>

SEE ALSO [OrderBasis](#), [OrderCoefIdeals](#), [Order](#),

EXAMPLE An absolute extension:

```
kash> o := Order(Z,2,165);
Generating polynomial: x^2 - 165
```

```
kash> O := OrderMaximal(o);
F[1]
|
F[2]
/
```

```
/
Q
F [ 1]      Given by transformation matrix
F [ 2]      x^2 - 165
Discriminant: 165
```

```
kash> OrderTransformationMatrix(0);
[ 2, [2 1]
  [0 1] ]
```

NAME	OrderUnitsAreFund
PURPOSE	Tests whether the current unit system of the given order is known to be fundamental or not.
SYNTAX	<pre> b := OrderUnitsAreFund(o); b := OrderUnitsAreFund(o,c); boolean b, c order o </pre>
DESCRIPTION	<p>The order \mathfrak{o} must be an absolute order. The <code>OrderUnitsAreFund</code> function can be used to check whether the current unit system of the order \mathfrak{o} is known to be fundamental.</p> <p>Additionally, the <code>OrderUnitsAreFund</code> function makes it possible to set an internal flag in the order \mathfrak{o} which indicates that the current unit system is known to be fundamental. This can be achieved by invoking the <code>OrderUnitsAreFund</code> function with second optional argument c.</p>
SEE ALSO	OrderUnitsFund ,

EXAMPLE In a CM field it is quite easy to compute fundamental units.

```

kash> O := OrderMaximal (Order (x^3 + x^2 - 16*x - 8));;
kash> units := OrderUnitsFund (O);
[ [1, 2, -1], [1, 2, 1] ]
kash> Ox := PolyAlg (O);;
kash> f := Poly (Ox, [1,1,1]);
x^2 + x + 1
kash> O2 := OrderMaximal (OrderAbs (Order (f)));
      F[1]
      |
      F[2]
      /
      /
Q
F [ 1]      Given by transformation matrix
F [ 2]      x^6 + 5*x^5 - 20*x^4 - 95*x^3 + 136*x^2 + 465*x + 475
Discriminant: -502020288

```

The fundamental units of \mathcal{O} are moved to $\mathcal{O}|2$, the index of the lifted group in $U(\mathcal{O}|2)$ is shown not to be divisible by 2.

```
kash> Apply (units, u -> EltMove (u, 02));
kash> for u in units do OrderUnitsMerge (02, u); od;
kash> OrderReg (02);
28.915447106812858929100272734448621763738691622816
kash> OrderUnitsPFund (02, 2);
[ [6, -1, -1, 1, 1, -1], [-8, 1, 1, -1, -1, 1] ]
kash> OrderReg (02);
28.915447106812858929100272734448621763738691622816
kash> OrderUnitsAreFund (02);
false
kash> \# Mark the units as fundamental
kash> OrderUnitsAreFund (02,true);
true
```

NAME	OrderUnitsEquation
PURPOSE	Solves a unit equation.
SYNTAX	$L := \text{OrderUnitsEquation}(\alpha, \beta, \gamma);$ $L := \text{OrderUnitsEquation}(\alpha, \beta);$ $\text{list} \quad L$ $\text{algebraic elements} \quad \alpha, \beta, \gamma$
DESCRIPTION	<p>Let α, β, γ be non-zero elements of a certain order \mathfrak{o}. We denote by U the unit group of \mathfrak{o}. The <code>OrderUnitsEquation</code> function computes all $(\varepsilon_1, \varepsilon_2) \in U \times U$ satisfying</p> $\alpha \cdot \varepsilon_1 + \beta \cdot \varepsilon_2 = \gamma.$ <p>If the third argument is omitted, $\gamma = 1$ is used by default.</p>
SEE ALSO	OrderUnitsExcep ,

EXAMPLE Compute all units $\varepsilon_1, \varepsilon_2 \in \mathbb{Z}[\sqrt[4]{5}]$ with $1 \cdot \varepsilon_1 + 2 \cdot \varepsilon_2 = 3$.

```
kash> o := Order(Z,4,5);
Generating polynomial: x^4 - 5

kash> one := Elt(o,1);
1
kash> OrderUnitsEquation(one,2*one,3*one);
[ [ 1, 1 ] ]
```

NAME	OrderUnitsExcep
PURPOSE	Computes all exceptional units of an order.
SYNTAX	<pre> L := OrderUnitsExcep(o); L := OrderUnitsExcep(o,"orbits" "list"); n := OrderUnitsExcep(o,"number"); list L integer n order o </pre>
DESCRIPTION	<p>Let \mathfrak{o} be an arbitrary order. In the terminology of T. Nagell [Nag69] a unit $\varepsilon \in \mathfrak{o}$ is exceptional if $1 - \varepsilon$ is also a unit.</p> <p>For an exceptional unit $\varepsilon \in \mathfrak{o}$ all the units</p> $\varepsilon_1 = \varepsilon, \quad \varepsilon_2 = \frac{1}{\varepsilon}, \quad \varepsilon_3 = 1 - \varepsilon, \quad \varepsilon_4 = \frac{1}{1 - \varepsilon}, \quad \varepsilon_5 = \frac{\varepsilon - 1}{\varepsilon}, \quad \varepsilon_6 = \frac{\varepsilon}{\varepsilon - 1}$ <p>are exceptional. The set formed by $\varepsilon_1, \dots, \varepsilon_6$ is called the orbit of ε.</p> <p><code>OrderUnitsExcep(o)</code> or <code>OrderUnitsExcep(o,"orbits")</code> returns a list whose entries represent all orbits.</p> <p><code>OrderUnitsExcep(o,"number")</code> returns the number of exceptional units in \mathfrak{o}.</p> <p><code>OrderUnitsExcep(o,"list")</code> returns the list L of all exceptional units in \mathfrak{o}.</p>
SEE ALSO	EltExcepUnitOrbit , OrderExcepSequence , OrderUnitsEquation ,

EXAMPLE Compute exceptional units in $\mathbb{Z}[\zeta_7]$.

```

kash> O := Order(Poly(Zx,[1,1,1,1,1,1,1]));
Generating polynomial: x^6 + x^5 + x^4 + x^3 + x^2 + x + 1

kash> OrderUnitsExcep(O);
[ [0, -1, 0, 0, 0, 0], [0, -1, -1, 0, 0, 0], [0, 0, -1, 0, 0, 0],
  [0, -1, -1, -1, 0, 0], [0, -1, -1, -1, -1, 0], [-1, 0, 0, -1, -1, 0],
  [0, 0, 0, -1, -1, 0], [1, 0, 0, -1, -1, 0],
  [-684, 0, -549, -244, -244, -549], [-7, 0, -6, -3, -3, -6],
  [-3, 0, -3, -1, -1, -3], [-2, 0, -2, -1, -1, -2] ]
kash> OrderUnitsExcep(O,"number");

```


72

```

kash> OrderUnitsExcep(0,"list");
[ [0, -1, 0, 0, 0, 0], [1, 1, 0, 0, 0, 0], [0, -1, -1, -1, -1, -1],
  [0, -1, 0, -1, 0, -1], [1, 1, 0, 1, 0, 1], [1, 1, 1, 1, 1, 1],
  [0, -1, -1, 0, 0, 0], [1, 1, 1, 0, 0, 0], [0, 0, -1, 0, -1, 0],
  [0, -1, 0, 0, -1, 0], [1, 1, 0, 0, 1, 0], [1, 0, 1, 0, 1, 0],
  [0, 0, -1, 0, 0, 0], [1, 0, 1, 0, 0, 0], [0, -1, 0, 0, -1, -1],
  [0, 0, 0, 0, 0, -1], [1, 0, 0, 0, 0, 1], [1, 1, 0, 0, 1, 1],
  [0, -1, -1, -1, 0, 0], [0, 0, 0, -1, 0, 0], [1, 0, 0, 1, 0, 0],
  [1, 1, 1, 1, 0, 0], [0, 0, 0, 0, -1, 0], [1, 0, 0, 0, 1, 0],
  [0, -1, -1, -1, -1, 0], [1, 1, 1, 1, 1, 0], [0, -1, -1, 0, -1, -1],
  [0, 0, 0, -1, 0, -1], [1, 0, 0, 1, 0, 1], [1, 1, 1, 0, 1, 1],
  [-1, 0, 0, -1, -1, 0], [2, 0, 0, 1, 1, 0], [1, 0, -1, -2, -2, -1],
  [-1, 0, -1, -1, -1, -1], [2, 0, 1, 1, 1, 1], [0, 0, 1, 2, 2, 1],
  [0, 0, 0, -1, -1, 0], [1, 0, 0, 1, 1, 0], [0, 0, -1, -1, -1, -1],
  [0, 0, -1, 0, 0, -1], [1, 0, 1, 0, 0, 1], [1, 0, 1, 1, 1, 1],
  [1, 0, 0, -1, -1, 0], [0, 0, 0, 1, 1, 0], [-1, 0, -1, 0, 0, -1],
  [2, 0, -1, 1, 1, -1], [-1, 0, 1, -1, -1, 1], [2, 0, 1, 0, 0, 1],
  [-684, 0, -549, -244, -244, -549], [136, 0, -305, -549, -549, -305],
  [441, 0, -244, 305, 305, -244], [-440, 0, 244, -305, -305, 244],
  [-135, 0, 305, 549, 549, 305], [685, 0, 549, 244, 244, 549],
  [-7, 0, -6, -3, -3, -6], [2, 0, -3, -6, -6, -3], [5, 0, -3, 3, 3, -3],
  [-4, 0, 3, -3, -3, 3], [-1, 0, 3, 6, 6, 3], [8, 0, 6, 3, 3, 6],
  [-3, 0, -3, -1, -1, -3], [1, 0, -2, -3, -3, -2], [3, 0, -1, 2, 2, -1],
  [-2, 0, 1, -2, -2, 1], [0, 0, 2, 3, 3, 2], [4, 0, 3, 1, 1, 3],
  [-2, 0, -2, -1, -1, -2], [1, 0, -1, -1, -1, -1], [1, 0, -1, 0, 0, -1],
  [0, 0, 1, 0, 0, 1], [0, 0, 1, 1, 1, 1], [3, 0, 2, 1, 1, 2] ]

```

NAME	OrderUnitsFund
PURPOSE	Returns a list whose entries are a maximal set of fundamental units (algebraic numbers) of the given order.
SYNTAX	<pre>L := OrderUnitsFund(o);</pre> <pre>list L order o</pre>
DESCRIPTION	The order \mathfrak{o} must be an absolute order. For further information about units, the underlying theory and the implemented algorithms, we refer the reader to [Wil93]
SEE ALSO	OrderSUnits , OrderSUnitsPositive ,

EXAMPLE A set of fundamental units in the maximal order of $x^4 + 73x^2 - 280x - 2399$.

```
kash> O := OrderMaximal (Order (Poly(Zx,[1,0,73,-280,-2399])));
  F[1]
  |
  F[2]
  /
  /
Q
F [ 1]      Given by transformation matrix
F [ 2]      x^4 + 73*x^2 - 280*x - 2399
Discriminant: -997975
```

```
kash> OrderUnitsFund (O);
[ [-1, 0, -1, 1], [109589, -13889, 743946, -766419] ]
```

NAME	OrderUnitsIndep
PURPOSE	Computes a maximal system of independent units.
SYNTAX	<pre> L := OrderUnitsIndep(o); L := OrderUnitsIndep(o,"classgroup"); L := OrderUnitsIndep(o,"classgroup","list"); list L order o </pre>
DESCRIPTION	<p>Let \mathfrak{o} be an absolute order with unit rank $r \geq 1$.</p> <p>OrderUnitsIndep(o) The OrderUnitsIndep function computes a maximal system $\varepsilon_1, \dots, \varepsilon_r$ of independent units of \mathfrak{o} which are returned in a list.</p> <p>OrderUnitsIndep(o,"classgroup") The OrderUnitsIndep function returns a list containing a maximal system $\varepsilon_1, \dots, \varepsilon_r$ of independent units of \mathfrak{o} found during class group computations done by using the "euler" option. Note: It is not necessary to call the function OrderClassGroup first. It is done automatically.</p> <p>When the class group computation is done without the "fast" option, the units are proven to be p-maximal for primes p dividing the class number of \mathfrak{o}.</p> <p>An empty list is returned if a system of fundamental units has been computed before the computation of the class group.</p> <p>If a system of units has been computed before the computation of the class group, OrderUnitsIndep(o,"classgroup") merges the two systems of units. This resulting system of units is stored in the order \mathfrak{o} and returned. No merging occurs when giving "list" as additional parameter.</p> <p>For further information about units, the underlying theory and the implemented algorithms, we refer the reader to [Wil93]</p>
SEE ALSO	OrderUnitsFund , OrderClassGroup ,

EXAMPLE A set of independent units in the maximal order of $x^4 + 73x^2 - 280x - 2399$.

```

kash> O := OrderMaximal (Order (Poly(Zx,[1,0,73,-280,-2399])));
      F[1]
      |
      F[2]
      /
      /

```

Q

F [1] Given by transformation matrix

F [2] $x^4 + 73x^2 - 280x - 2399$

Discriminant: -997975

kash> OrderUnitsIndep (0);

[[0, 0, 1, -1], [109589, -13889, 743946, -766419]]

NAME	OrderUnitsLLL
PURPOSE	Computes a LLL reduced system of independent units.
SYNTAX	OrderUnitsLLL (0); order 0
DESCRIPTION	The given absolute order \mathcal{O} has to be maximal. Computes a LLL reduced representation of the units given in \mathcal{O} . If no units have been computed so far, an arbitrary system of independent units is computed in advance.
SEE ALSO	OrderLLL , OrderUnitsPFund ,
EXAMPLE	

```
kash> O := OrderMaximal(x^3+x^2-2*x-1);
Generating polynomial: x^3 + x^2 - 2*x - 1
Discriminant: 49
```

```
kash> OrderUnitsLLL(O);
[ [-1, 1, 0], [5, -1, -2] ]
```

NAME	OrderUnitsMerge
PURPOSE	Extends a system of units of the given order.
SYNTAX	$B := \text{OrderUnitsMerge}(o, \eta); \quad B := \text{OrderUnitsMerge}(o, \eta, \text{"append"});$ <div style="margin-left: 100px;"> $\text{boolean} \quad B$ $\text{order} \quad o$ $\text{algebraic element} \quad \eta \quad \text{a unit in } o$ </div>
DESCRIPTION	<p>The OrderUnitsMerge function performs a MLLL reduction to expand the current unit group of the given absolute order o by merging η in it. If the rank of the unit group increases OrderUnitsMerge will return TRUE. Otherwise it will return FALSE.</p> <p>OrderUnitsMerge($o, \eta, \text{"append"}$) just appends the unit η to the system of units of the order o, MLLL is not performed.</p>

EXAMPLE Extending a system of units of an order.

```
kash> o := Order (Poly (Zx,[1,0,-186,0,13097,0,-412704,0,4946176]));
Generating polynomial: x^8 - 186*x^6 + 13097*x^4 - 412704*x^2 + 4946176

kash> O := OrderMaximal (o);;
kash> OrderUnitsMerge (O,Elt (O,[-14, 5, 1, 0, -1, 1, 3, -5]));
true
kash> OrderUnitsMerge (O,Elt (O,[177, -301, 26, -31, 39, -74, -190, 377]));
true
kash> OrderReg (O);
Error, no maximal set of independent units known
kash> OrderUnitsMerge (O,Elt (O,[227, -555, 57, -63, 74, -140, -361, 714]));
true
kash> a := Elt (O,[-14, 5, 1, 0, -1, 1, 3, -5]);;
kash> b := Elt (O,[227, -555, 57, -63, 74, -140, -361, 714]);;
kash> OrderUnitsMerge (O,a*b);
false
kash> OrderReg (O);
121.297795062765195428716897829517493027623458635
```

NAME	OrderUnitsPFund
PURPOSE	Computes the p maximal overgroup of the units in the given order.
SYNTAX	<p>OrderUnitsPFund (O,p);</p> <p>order 0</p> <p>rational prime p</p>
DESCRIPTION	<p>The given absolute order \mathcal{O} must be maximal. This function computes the p maximal overgroup of the units given in \mathcal{O}. If no units have been computed so far, an arbitrary system of independent units is computed in advance. For further information we refer the reader to [Wil93].</p>
SEE ALSO	OrderUnitsFund ,

EXAMPLE Let ρ be a root of $t^3 + 15t^2 + 15t + 15$. We consider the order $\mathbb{Z}[\rho]$, which is already maximal with signature $[1, 1]$.
 $2729 + 573\rho + 27\rho^2$ is a unit in this order.

```
kash> O := OrderMaximal(Order(Poly(Zx,[1,15,15,15])));
Generating polynomial: x^3 + 15*x^2 + 15*x + 15
Discriminant: -110700
```

```
kash> a := Elt(O,[2729, 573, 27]);
[2729, 573, 27]
kash> OrderUnitsMerge(O,a);
true
kash> OrderReg(O);
15.630874731280170083241764717443071385772563284107
kash> OrderUnitsPFund(O,2);
[ [2729, 573, 27] ]
kash> OrderUnitsPFund(O,3);
[ [-14, -1, 0] ]
kash> OrderReg(O);
5.210291577093390027747254905814357128590623830734
```

PRINTLEVEL

NAME	PRINTLEVEL
PURPOSE	Reads or sets the printlevel for a certain function.
SYNTAX	<pre>x := PRINTLEVEL(s); x := PRINTLEVEL(s,level); x := PRINTLEVEL("all",level); small integer x string s small integer level</pre>
DESCRIPTION	<p>The PRINTLEVEL function reads or sets the printlevel for a certain function, i.e. the level of how much information of a computation is displayed for the KANT function specified by the debug string s. For valid debug strings refer to the DEBUG entries in the reference manual. To read the printlevel corresponding to s enter PRINTLEVEL(s). A printlevel can be set by calling the PRINTLEVEL function with two arguments the second one being the new printlevel. In both cases the current printlevel will be returned.</p> <p>The command PRINTLEVEL("all",level) sets the printlevel for all KANT functions to level. Please note, that this function should be used carefully, e.g. for a first try it is useful to start with PRINTLEVEL("all", 0).</p>

EXAMPLE Compute a maximal order with printlevel 2:

```
kash> f := Poly(Zx,[1,0,-2750]);
x^2 - 2750
kash> O := Order(f);
Generating polynomial: x^2 - 2750
```

```
kash> PRINTLEVEL("ROUND2",2);
2
kash> OrderMaximal(O,"round2");
```

Factorization of discriminant: $2^3 * 5^3 * 11^1$

Factors of discriminant with (possibly) $R(p) \neq \mathbb{Z}\mathbb{Z}[a] : 5^3$
Calculation of p-maximal overorder...
Prime: 5 maximal exponent: 3
relative degree found: 5
Index exponent found until now: 1
relative degree found: 1

Transformation for 5...
Index of order: 5
(FLD=) 2, -2, 0, 0, 0, 0, 0, 0, 0, 0, 2, *
0 -2750
5 0 5
0 1 5
F[1]
|
F[2]
/
/
Q
F [1] Given by transformation matrix
F [2] $x^2 - 2750$
Generating polynomial: $x^2 - 2750$
Discriminant: 440

NAME	Poly
PURPOSE	Creates a polynomial.
SYNTAX	<pre>f := Poly(A, L);</pre> <pre> polynomial f polynomial algebra A list L </pre>
DESCRIPTION	<p>Given a polynomial algebra A and a list of elements (coefficients) $L := [c_n, c_{n-1}, \dots, c_0]$ the function returns the polynomial $f(x) := \sum_{i=0}^n c_i x^i$. The c_i must be elements of the coefficient ring of A.</p>
SEE ALSO	PolyAlg ,
EXAMPLE	Creation of the polynomial $x^5 + 2 * x^4 + 3 * x^3 + 4 * x^2 + 5 * x + 6 \in \mathbb{Z}[x]$:

```
kash> f := Poly(Zx, [1, 2, 3, 4, 5, 6]);
x^5 + 2*x^4 + 3*x^3 + 4*x^2 + 5*x + 6
```

NAME	PolyAlg
PURPOSE	Creates a univariate polynomial algebra over a ring or returns the polynomial algebra of a polynomial.
SYNTAX	<pre>Sx := PolyAlg(S [, name]);</pre> <p> polynomial algebra Sx ring or polynomial S string name </p>
DESCRIPTION	<p>Creates a univariate polynomial algebra $S[x]$ over a ring S. Zx is a predefined constant for the polynomial algebra over the integers. A name for the variable can be given as a second parameter.</p> <p>If S is a polynomial the polynomial algebra of S is returned.</p>
SEE ALSO	Poly , PolyAlgCoef , Zx ,
EXAMPLE	Creating a polynomial algebra over an order and computing a minimal polynomial:
	<pre>kash> o := Order(Poly(Zx, [1 ,0 , 73, -280, -2399])); Generating polynomial: x^4 + 73*x^2 - 280*x - 2399</pre> <pre>kash> ox := PolyAlg(o); Univariate Polynomial Ring in x over Generating polynomial: x^4 + 73*x^2 - 280\ *x - 2399</pre> <pre>kash> M := Mat (o,[[1,2],[2,3]]); [1 2] [2 3] kash> MatMinPoly (M); x^2 - 4*x - 1</pre>

NAME	PolyAlgCoef
PURPOSE	Returns the coefficient ring of a polynomial algebra.
SYNTAX	$S := \text{PolyAlgCoef}(Sx);$ <div style="display: flex; justify-content: space-between;"> ring S </div> <div style="display: flex; justify-content: space-between;"> polynomial algebra Sx </div>
DESCRIPTION	Returns the coefficient ring S of a polynomial algebra $S[x]$.
SEE ALSO	PolyAlg ,
EXAMPLE	

```
kash> o := Order(Poly(Zx, [1 , 0, 73, -280, -2399]));
```

```
Generating polynomial: x^4 + 73*x^2 - 280*x - 2399
```

```
kash> ox := PolyAlg(o);
```

```
Univariate Polynomial Ring in x over Generating polynomial: x^4 + 73*x^2 - 280\
*x - 2399
```

```
kash> PolyAlgCoef(ox);
```

```
Generating polynomial: x^4 + 73*x^2 - 280*x - 2399
```

NAME PolyDeg

PURPOSE Returns the degree of a given polynomial.

SYNTAX `n := PolyDeg(f);`

 integer n
 polynomial f

DESCRIPTION Returns the degree n of a polynomial f . The degree of the zero-polynomial is -1 .

EXAMPLE

```
kash> f := Poly(Zx, [1, 2, 3, 4, 5, 6]);  
x^5 + 2*x^4 + 3*x^3 + 4*x^2 + 5*x + 6  
kash> PolyDeg(f);  
5  
kash> f := Poly(Zx, [0]);  
0  
kash> PolyDeg(f);  
-1
```

NAME	PolyDeriv
PURPOSE	Returns the derivative of a given polynomial.
SYNTAX	$h := \text{PolyDeriv}(f);$ <div>$\begin{array}{ll} \text{polynomial} & h \\ \text{polynomial} & f \end{array}$</div>
DESCRIPTION	Returns the derivative $h \in S[x]$ of a polynomial $f \in S[x]$.

EXAMPLE The derivative of $x^4 + 73 * x^2 - 280 * x - 2399$:

```
kash> f := Poly(Zx, [1, 0, 73, -280, -2399]);  
x^4 + 73*x^2 - 280*x - 2399  
kash> PolyDeriv(f);  
4*x^3 + 146*x - 280
```

NAME	PolyDisc
PURPOSE	Returns the discriminant of a polynomial.
SYNTAX	$d := \text{PolyDisc}(f);$ <div style="display: flex; justify-content: space-between;"> discriminant (integer or finite field element) d </div> <div style="display: flex; justify-content: space-between;"> polynomial f </div>
DESCRIPTION	Returns the discriminant $d \in \mathbb{Z}$ of a polynomial $f \in \mathbb{Z}[x], \mathbb{F}_q[x]$.
SEE ALSO	PolyRedDisc ,

EXAMPLE The discriminant of $x^4 + 73 * x^2 - 280 * x - 2399$:

```
kash> f := Poly(Zx, [1, 0, 73, -280, -2399]);
x^4 + 73*x^2 - 280*x - 2399
kash> PolyDisc(f);
-10815318540400
```

NAME	PolyFactor
PURPOSE	Returns the factorization of the given polynomial.
SYNTAX	<pre> F := PolyFactor(f); F := PolyFactor(f, p); F := PolyFactor(f, p, m); list F polynomial f prime number p integer m </pre>
DESCRIPTION	<p>This function returns the factorization of the given polynomial over its coefficient ring. Supported ring types are: \mathbb{Z}, \mathbb{Q}, $\mathbb{F}[p]$, \mathcal{O}, and \mathbb{Q}_p.</p> <p>If the polynomial has coefficients in \mathbb{Z}, a second argument p may be specified to obtain the factorization modulo (the prime) p.</p> <p>If the polynomial has coefficients in \mathbb{Z} or \mathbb{Q} and it is monic an approximation to a p-adic factorization can be obtained by giving a prime number p and a precision m.</p>
SEE ALSO	Factor ,

EXAMPLE Factorization of the polynomial $f(x) = 4x^7 + 6x^6 + 12x^5 + 14x^4 + 27x^3 + 24x^2 + 30x + 9 \in \mathbb{Z}[x]$:

```

kash> f := Poly(Zx, [4,6,12,14,27,24,30,9]);
4*x^7 + 6*x^6 + 12*x^5 + 14*x^4 + 27*x^3 + 24*x^2 + 30*x + 9
kash> Factor(f);
[ [ 2*x^2 + x + 3, 1 ], [ 2*x^5 + 2*x^4 + 2*x^3 + 3*x^2 + 9*x + 3, 1 ] ]

```


NAME	PolyGcd
PURPOSE	Returns the gcd of two polynomials.
SYNTAX	<pre>g := PolyGcd(f, h);</pre> <p> polynomial g polynomial f polynomial h </p>
DESCRIPTION	Given two polynomials $f, h \in S[x]$ where S is a field or \mathbb{Z} the function returns the gcd g of f and h .
SEE ALSO	PolyXGcd ,
EXAMPLE	

```
kash> f := Poly(Zx, [1, 2, 3, 4, 5, 6]);
x^5 + 2*x^4 + 3*x^3 + 4*x^2 + 5*x + 6
kash> h := Poly(Zx, [1, 4, 10, 16, 22, 28, 27, 18]);
x^7 + 4*x^6 + 10*x^5 + 16*x^4 + 22*x^3 + 28*x^2 + 27*x + 18
kash> PolyGcd(f, h);
x^5 + 2*x^4 + 3*x^3 + 4*x^2 + 5*x + 6
```

NAME	PolyHenselLift
PURPOSE	Lifts a factorization of a polynomial modulo a prime ideal to a factorization modulo the prime ideal to a given exponent.
SYNTAX	<pre>PolyHenselLift(f, A, n);</pre> <pre>polynomial over an order f ideal of the same order A integer n</pre>
DESCRIPTION	

NAME PolyIsIrreducible

PURPOSE missing shortdoc

SYNTAX `b := PolyIsIrreducible(f);`

 boolean b
 polynomial f

DESCRIPTION Checks if a given polynomial is irreducible.

EXAMPLE

```
kash> f := Poly(Zx, [1, 0, -4, 0, 1]);  
x^4 - 4*x^2 + 1  
kash> PolyIsIrreducible(f);  
true
```

NAME	PolyIsSquarefree
PURPOSE	missing shortdoc
SYNTAX	<pre>b := PolyIsSquarefree(f);</pre> <pre>boolean b polynomial f</pre>
DESCRIPTION	Checks if a given polynomial is square free
EXAMPLE	

```
kash> f := Poly(Zx, [1, 0, -4, 0, 1]);  
x^4 - 4*x^2 + 1  
kash> PolyIsSquarefree(f);  
true
```

NAME PolyIsZero

PURPOSE Returns true iff the argument is a polynomial and equal to the zero polynomial

SYNTAX `b := PolyIsZero(f);`

 boolean b
 f

EXAMPLE

```
kash> f := Poly(Zx, [1, 2, 1, 2]);  
x^3 + 2*x^2 + x + 2  
kash> PolyIsZero(f);  
false  
kash> f := Poly(Zx, [0, 0, 0]);  
0  
kash> PolyIsZero(f);  
true
```

NAME PolyMakeMonicInOrder

PURPOSE Make $f(x)$ monic by rational transformation, so that the resulting monic polynomial $g(x)$ is defined over \mathfrak{o} and creates the same algebra as $f(x)$. $g(x)$ may be the same as $f(x)$.

SYNTAX `g:= PolyMakeMonicInOrder(f, o);`

`order` `o`

`polynomial` `f`

`polynomial` `g`

DESCRIPTION

EXAMPLE

```
kash> f:= -1/3*x^3+7;
```

```
-1/3*x^3 + 7
```

```
kash> g:= PolyMakeMonicInOrder(f, Z);
```

```
x^3 - 21
```

NAME PolyMakeMonicInZ

PURPOSE Given $f(x)$ in $Q[x]$, the function returns $g(x)$ monic in $Z[x]$, such that $g(x)$ generates the same algebra as $f(x)$.

SYNTAX `g:= PolyMakeMonicInZ(f);`

 polynomial f

 polynomial g

DESCRIPTION

EXAMPLE

```
kash> f:= -1/3*x^3+7;
-1/3*x^3 + 7
kash> g:= PolyMakeMonicInZ(f);
x^3 - 21
```

NAME	PolyMove
PURPOSE	Tries to compute a representation of a polynomial in a different polynomial algebra.
SYNTAX	<pre>g := PolyMove (f, S); polynomial g polynomial f ring S</pre>
DESCRIPTION	Given a polynomial and a polynomial algebra or a ring, this function computes a representation of the polynomial either in the given polynomial algebra or in the polynomial algebra associated to the ring. In case only a ring is given, the function automatically generates the polynomial algebra, which is then accessible by the function call PolyAlg (g).
SEE ALSO	PolyAlg ,

EXAMPLE We will factor a polynomial over its equation order.

```
kash> o := Order(Z, 4, 5);;
kash> ox := PolyAlg(o);;
kash> f := PolyMove(OrderPoly(Zx, o), ox);
x^4 - 5
kash> Factor(f);
[ [ x + [0, -1, 0, 0], 1 ], [ x + [0, 1, 0, 0], 1 ],
  [ x^2 + [0, 0, 1, 0], 1 ] ]
```


NAME	PolyMoveIntegral
PURPOSE	Returns the polynomial moved to integral coefficients.
SYNTAX	<pre>g := PolyMoveIntegral(f);</pre> <p> polynomial f polynomial g </p>
DESCRIPTION	Given a polynomial defined over the quotient field of R such that the denominators of the coefficients are all one, the function returns this polynomial defined over R .
SEE ALSO	PolyMove ,
EXAMPLE	

```
kash> a := 2*(x/2);
x
kash> PolyAlg(a);
Univariate Polynomial Ring in x over Rational Field

kash> b := PolyMoveIntegral(a);
x
kash> PolyAlg(b);
Univariate Polynomial Ring in x over Integer Ring
```

NAME	PolyNewtonLift
PURPOSE	Lifts an algebraic element with the Newton lifting method.
SYNTAX	<pre> beta := PolyNewtonLift(f, alpha, k); beta := PolyNewtonLift(f, alpha, k, a); algebraic element alpha integer (0 if omitted) a polynomial f integer k polynomial beta </pre>
DESCRIPTION	Given an algebraic element α with $f(\alpha) \equiv 0 \pmod{(\mathfrak{t} - \mathfrak{a})}$, this function calculates an algebraic element $\beta \in o[t]$ with $f(\beta) \equiv 0 \pmod{(\mathfrak{t} - \mathfrak{a})^k}$.
EXAMPLE	<pre> kash> AlffInit(Q,"t","X"); "Defining global variables: k, w, kt, ktf, ktX, t, X, AlffGlobals" kash> f:=X^4-2+t; X^4 + t - 2 kash> o:=Order(Z,4,2); Generating polynomial: x^4 - 2 kash> alpha:=OrderBasis(o)[2]; [0, 1, 0, 0] kash> beta:=PolyNewtonLift(f,alpha,5); [0, -77, 0, 0] / 32768*t^4 + [0, -7, 0, 0] / 1024*t^3 + [0, -3, 0, 0] / 128*t^2 + [0, -1, 0, 0] / 8*t + [0, 1, 0, 0] </pre>

NAME	PolyNorm
PURPOSE	Returns the norm of a polynomial.
SYNTAX	$n := \text{PolyNorm}(f);$ $\begin{array}{ll} \text{norm}(\text{polynomial}) & n \\ \text{polynomial} & f \end{array}$
DESCRIPTION	Returns the norm $n \in \mathfrak{o}[x]$ of a polynomial $f \in \mathcal{O}[x]$. \mathfrak{o} is the coefficient ring of the order \mathcal{O} .
EXAMPLE	

```

kash> O := Order(Z,2,3);
Generating polynomial: x^2 - 3

kash> Ox := PolyAlg(O);
Univariate Polynomial Ring in x over Generating polynomial: x^2 - 3

kash> f := Poly(Ox,[1,0,Elt(0,[1,4]),Elt(0,[3,4]),9]);
x^4 + [1, 4]*x^2 + [3, 4]*x + 9
kash> PolyNorm(f);
x^8 + 2*x^6 + 6*x^5 - 29*x^4 - 90*x^3 - 21*x^2 + 54*x + 81

```

NAME	PolyPowerMod
PURPOSE	missing shortdoc
SYNTAX	<pre>h := PolyPowerMod(f, n, g);</pre> <pre>polynomial g</pre> <pre>polynomial f</pre> <pre>polynomial h</pre> <pre>positive integer n</pre>
DESCRIPTION	Returns the remainder of a power product of a polynomial modulo another polynomial.

NAME	PolyPrimeList
PURPOSE	Returns a list containing all monic prime polynomials of degree d in $k[x]$ for a finite field k .
SYNTAX	<div>L := PolyPrimeList(kx, d);</div> <div><div>list</div><div>polynomial algebra</div><div>integer</div><div>L</div><div>kx</div><div>d</div><div>over finite field k</div></div>
SEE ALSO	PolyPrimeNum , PolyPrimeRandom ,
EXAMPLE	

NAME	PolyPrimeNum
PURPOSE	Return the number of monic prime polynomials over a finite field.
SYNTAX	<pre>f := PolyPrimeNum(kx, d);</pre> <p> polynomial f polynomial algebra over k kx integer d </p>
DESCRIPTION	For given degree d and finite field k this function returns the number of monic prime polynomials of degree d in $k[x]$.
EXAMPLE	

```
kash> k := FF(5, 2);
Finite field of size 5^2
kash> kx := PolyAlg(k, "x");
Univariate Polynomial Ring in x over GF(5^2)

kash> PolyPrimeNum(kx, 11);
216744162819600
```

NAME	PolyPrimeRandom						
PURPOSE	Return a random prime polynomial over a finite field.						
SYNTAX	<pre>f := PolyPrimeRandom(kx, d);</pre> <table> <tr> <td>polynomial</td><td>f</td></tr> <tr> <td>polynomial algebra over k</td><td>kx</td></tr> <tr> <td>integer</td><td>d</td></tr> </table>	polynomial	f	polynomial algebra over k	kx	integer	d
polynomial	f						
polynomial algebra over k	kx						
integer	d						
DESCRIPTION	For given degree d and finite field k this function returns a random prime polynomial of degree d in $k[x]$.						
EXAMPLE	<pre>kash> k := FF(5, 2); Finite field of size 5^2 kash> kx := PolyAlg(k, "x"); Univariate Polynomial Ring in x over GF(5^2) kash> PolyPrimeRandom(kx, 11); x^11 + 4*x^10 + w^14*x^9 + x^8 + 3*x^7 + 2*x^6 + w^22*x^5 + w^19*x^4 + w^15*x^3 + w^16*x^2 + w^23*x + 4</pre>						

NAME	PolyQuotRem
PURPOSE	missing shortdoc
SYNTAX	$L := \text{PolyGcd}(f, g);$ polynomial g polynomial f list L
DESCRIPTION	Returns the quotient and remainder of two polynomials.
EXAMPLE	

```
kash> f := Poly(Zx, [1, 2, 3, 4, 5, 6]);  
x^5 + 2*x^4 + 3*x^3 + 4*x^2 + 5*x + 6  
kash> h := Poly(Zx, [1, 4, 10, 16, 22, 28, 27, 18]);  
x^7 + 4*x^6 + 10*x^5 + 16*x^4 + 22*x^3 + 28*x^2 + 27*x + 18  
kash> PolyQuotRem(f, h);  
[ 0, x^5 + 2*x^4 + 3*x^3 + 4*x^2 + 5*x + 6 ]
```


NAME	PolyRedDisc
PURPOSE	Returns the reduced discriminant of a separable polynomial.
SYNTAX	<pre>d := PolyRedDisc(f);</pre> <p>integer d polynomial f</p>
DESCRIPTION	Returns the reduced discriminant $d \in \mathbb{Z}$ of a separable polynomial $f \in \mathbb{Z}[x], \mathbb{F}_q[x]$.
SEE ALSO	PolyDisc ,

EXAMPLE The discriminant of $x^4 + 73 * x^2 - 280 * x - 2399$:

```
kash> f := Poly(Zx, [1, 0, 73, -280, -2399]);
x^4 + 73*x^2 - 280*x - 2399
kash> PolyRedDisc(f);
270382963510
```

NAME	PolyResultant
PURPOSE	Computes the resultant of the two given polynomials.
SYNTAX	<pre> r := PolyResultant (f, g); polynomial r polynomial f polynomial g </pre>
DESCRIPTION	$\text{PolyResultant}(f,g) = 0$ if and only if f and g have a common factor which has positive degree in x
SEE ALSO	PolyGcd ,
EXAMPLE	

```

kash> Zxy := PolyAlg(Zx);
Univariate Polynomial Ring in y over Univariate Polynomial Ring in x over Inte\
ger Ring

kash> f := Poly(Zxy,[1,2*x+4,x^2 +x,6*x]);
y^3 + (2*x + 4)*y^2 + (x^2 + x)*y + 6*x
kash> f_ := PolyDeriv(f);
3*y^2 + (4*x + 8)*y + x^2 + x
kash> PolyResultant(f,f_);
-12*x^5 - 64*x^4 + 460*x^3 + 2828*x^2 + 1536*x

```

NAME	PolyRoundFour
PURPOSE	Returns the factorization of the given polynomial over the p-adic integers and certificates for the irreducibility of the factors.
SYNTAX	<pre> L := PolyRoundFour(f); L := PolyRoundFour(f, p); L := PolyRoundFour(f, p, m); list L polynomial f prime number p integer m </pre>
DESCRIPTION	This function returns the factorization of the given polynomial over the p-adic integers. Supported ring types are: \mathbb{Z} , \mathbb{Q} , and \mathbb{Q}_p . If the polynomial is not given over \mathbb{Q}_p a prime number p as a second parameter is needed. The desired precision can be given as a third parameter.
SEE ALSO	Factor , PolyFactor , OrderMaximal ,

NAME	PolySig
PURPOSE	Computes the signature of a monic squarefree polynomial.
SYNTAX	<pre>s := PolySig(f);</pre> <p>polynomial f list s</p>
DESCRIPTION	<p>Computes the signature of a monic squarefree polynomial $f(x)$ of degree n over \mathbb{Z} or over an absolute order o.</p> <p>In the first case the PolySig function returns a list containing the number r_1 of real roots and the number r_2 of pairs of complex roots of the given polynomial. In the second case the PolySig function returns a list s containing lists of the type above.</p> <p>Let $[r_1, r_2]$ be the signature of the generating polynomial of o and for $1 \leq i \leq r_1$ let $f^{(i)}(x) \in \mathbb{R}[x]$ be the polynomial received by applying the i-th real embedding of $Q(o)$ in \mathbb{C} to the coefficients of f. The i-th entry of s is the signature of the polynomial $f^{(i)}(x) \in \mathbb{R}[x]$ ($1 \leq i \leq r_1$). For $r_1 + 1 \leq i \leq r_1 + r_2$ the i-th entry of s is $[n, 0]$.</p> <p>The algorithm is based on [Coh95, Algorithm 4.1.11].</p>

EXAMPLE

```
kash> f:= Poly(Zx,[1,7,4,9,6,1]);
x^5 + 7*x^4 + 4*x^3 + 9*x^2 + 6*x + 1
kash> s:= PolySig(f);
[ 3, 1 ]
kash> p:= Poly(Zx,[1,1,1,0,0,-2]);
x^5 + x^4 + x^3 - 2
kash> o:= Order(p);
Generating polynomial: x^5 + x^4 + x^3 - 2

kash> ox:= PolyAlg(o);
Univariate Polynomial Ring in x over Generating polynomial: x^5 + x^4 + x^3 - \
2

kash> f:= Poly(ox,[Elt(o,[1,0,0,0,0]),Elt(o,[-12,13,3,20,-19]),
> Elt(o,[-1,12,-5,7,-12])]);
x^2 + [-12, 13, 3, 20, -19]*x + [-1, 12, -5, 7, -12]
kash> PolySig(f);
[ [ 2, 0 ], [ 2, 0 ], [ 2, 0 ] ]
```

NAME	PolySwapVars
PURPOSE	Changes the variables of the given bivariate polynomial.
SYNTAX	<pre>g := PolySwapVars (f); polynomial g polynomial f</pre>
DESCRIPTION	
EXAMPLE	<pre>kash> Zxy := PolyAlg(Zx); Univariate Polynomial Ring in y over Univariate Polynomial Ring in x over Inte\ ger Ring kash> f := Poly(Zxy,[1,2*x+4,x^2 +x,6*x]); y^3 + (2*x + 4)*y^2 + (x^2 + x)*y + 6*x kash> PolySwapVars(f); x*y^2 + (2*x^2 + x + 6)*y + x^3 + 4*x^2</pre>

NAME	PolyToList
PURPOSE	Returns the coefficients of a polynomial in a list.
SYNTAX	$L := \text{PolyToList}(f);$ <div> list L polynomial f </div>
DESCRIPTION	Given a polynomial $f(x) := \sum_{i=0}^n c_i x^i$ the function returns $L := [c_n, c_{n-1}, \dots, c_0]$.
SEE ALSO	Poly ,

EXAMPLE List of coefficients of the polynomial $x^5 + 2x^4 + 3x^3 + 4x^2 + 5x + 6 \in \mathbb{Z}[x]$:

```
kash> f := Poly(Zx, [1, 2, 3, 4, 5, 6]);
x^5 + 2*x^4 + 3*x^3 + 4*x^2 + 5*x + 6
kash> PolyToList(f);
[ 1, 2, 3, 4, 5, 6 ]
```

NAME	PolyXGcd
PURPOSE	Returns the extended gcd of two polynomials.
SYNTAX	<pre>l := PolyXGcd(f, h);</pre> <div> list of polynomials l polynomial f polynomial h </div>
DESCRIPTION	Given two polynomials $f, h \in S[x]$ where S is a field the function returns the gcd g of f and h together with polynomials a, b such that $g = af + bh$.
SEE ALSO	PolyGcd ,
EXAMPLE	

```
kash> Qx := PolyAlg(Q);
Univariate Polynomial Ring in x over Rational Field

kash> f := Poly(Qx, [1, 2, 3, 4, 5, 6]);
x^5 + 2*x^4 + 3*x^3 + 4*x^2 + 5*x + 6
kash> h := Poly(Qx, [1, 4, 10, 16, 22, 28, 27, 18]);
x^7 + 4*x^6 + 10*x^5 + 16*x^4 + 22*x^3 + 28*x^2 + 27*x + 18
kash> PolyXGcd(f, h);
Error, Panic: can't eval bag of type 116
```

NAME	PolyZeros
PURPOSE	Computes roots of a polynomial.
SYNTAX	<pre>L := PolyZeros (f [[, "complex" "int"] [, p [, m]]]);</pre> <p>list L</p> <p>polynomial f</p>
DESCRIPTION	<p>The PolyZeros function returns a list containing roots of a polynomial.</p> <p>PolyZeros(f) (or PolyZeros(f,"complex")) Let f be a polynomial over $\mathbb{Z}, \mathbb{Q}, \mathbb{R}$ or \mathbb{C}. The PolyZeros function returns a list containing all complex roots of f.</p> <p>PolyZeros(f,"int") Let f be a polynomial with integral coefficients, The PolyZeros function returns a list containing all integral roots of f.</p> <p>PolyZeros(f,p) Let f be a polynomial with integral coefficients, the functions returns p-adic approximations of the roots.</p> <p>PolyZeros(f,p,m) Let f be a polynomial with integral coefficients, then functions returns p-adic approximations of the roots to a p-adic precision m.</p>
EXAMPLE	Compute all complex roots of $x^3 + 2$:

```
kash> f := Poly(Zx,[1,0,0,2]);
x^3 + 2
kash> PolyZeros(f);
[ -1.259921049894873164767210607278228350570251464701,
  0.6299605249474365823836053036391141752851257323507919 + 1.09112363597172140\
356007261418980888132587333874*i,
  0.6299605249474365823836053036391141752851257323507919 - 1.09112363597172140\
356007261418980888132587333874*i ]
```


NAME	<code>Prec</code>
PURPOSE	Sets and gets the precision for real and complex computations in the shell.
SYNTAX	<pre>Prec(n); Prec(); integer n</pre>
DESCRIPTION	<p>All real and complex computations in the shell are performed in a global precision. The <code>Prec</code> function sets and gets this global precision. Remark that the global precision has no influence at all on computations done in internal KANT functions. The only way to set this is using <code>OrderPrec</code>.</p> <p><code>Prec(n)</code> Sets the global precision to <code>n</code> and returns the new global precision. This is the smallest positive integer which is divisible by 4 and greater than or equal to $\max(n, 12)$.</p> <p><code>Prec()</code> Returns the current global precision.</p>
SEE ALSO	<code>OrderPrec</code> ,

EXAMPLE Compute Euler's constant with different precisions:

```
kash> e;
2.718281828459045235360287471352662497757247093699
kash> Prec(100);
100
kash> e;
2.7182818284590452353602874713526624977572470936999595749669676277240766303535\
47594571382178525166
```

NAME PvmClear

PURPOSE Remove all broadcast-jobs, all failed-jobs and all slave-jobs.

SYNTAX PvmClear();

EXAMPLE

```
PvmInit();
PvmStartSlave(["nicky","julia","markov"]);
PvmSendAll("h:=GetEnvironment(HOSTNAME);;\n");
PvmShowBroadcastJobs();
PvmClear();
PvmShowBroadcastJobs();
PvmExit();
Exec("echo halt | pvm"); \# kill pvmd on all hosts
```

NAME PvmExit

PURPOSE Stops KASH-Pvm.

SYNTAX PvmExit

EXAMPLE see function PvmStartSlave

NAME	PvmGet
PURPOSE	Receives data from PVM.
SYNTAX	<pre>L := PvmGet();</pre> <p>List L</p>
DESCRIPTION	<p>If there is (valid) data PvmGet will receive one (complete) item, otherwise the return value is false. If it successfully receives data it will return a list containing a (small) integer as first entry as a flag. If flag equals 1 the other list entries are (exactly) as sent e.g. by PvmSend.</p> <p>If flag equals 2 this is send by Error(fmt) on a slave. The second entry will contain the hostname, the third the output of Error(fmt).</p> <p>If flag equals 4 the data comes from an ordinary Print() on the slave. The second parameter will be the hostname, the third the data string.</p> <p>Attention, it is possible to send arbitrary data with flag of 2 and 4 using PvmSlavePrint or PvmSlaveError. Data sent this way may not contain the hostname.</p> <p>In the examples given below we will NOT describe how to set up the pvm-network. Lines beginning with slave> indicate commands executed on the slave. They will not work an the master.</p>
EXAMPLE	

```

PvmInit();
PvmStartSlave("nicky");
Sleep(5);
PvmGet();
PvmSendNext("PvmSlaveSend(1,2,3,4,5,\"Hallo\");\n");
\# Sending this, the Slave will call PvmSlaveSend(...);
\# don't forget the \n
PvmGet();
PvmGet();

```

NAME	PvmGetAnswer
PURPOSE	Waits for a valid answer to arrive from the slave.
SYNTAX	<pre>L := PvmGetAnswer(true false);</pre> <p>list L first entry is in {1,2,4}, all others arbitrary</p>
DESCRIPTION	<p>This function performs a while-loop until a valid answer arrives (i. e. a data list starting with 1).</p> <p>Master only: if the parameter is true all incoming data is echoed to the screen. The function will return after the first answer arrived.</p>
SEE ALSO	PvmGet ,
EXAMPLE	See PvmClassgroup (in PvmClassgroup.kash) for an example.

NAME	PvmGetB
PURPOSE	Waits for any data to arrive.
SYNTAX	<pre>L := PvmGetB(); list L</pre>
DESCRIPTION	Calls alternately PvmGet and Sleep(1) until valid data arrives.
SEE ALSO	PvmGet ,
EXAMPLE	See PvmGet or PvmClassgroup (in PvmClassgroup.kash) for an example.

NAME PvmGetEval

PURPOSE

SYNTAX *not intended to be called by a user*

DESCRIPTION This function serves both as an example for a slave program and to provide a minimal slave. This function performs a typical cycle for the slave: It waits for any data to arrive, evaluates it and returns an answer. You may stop this loop by sending a simple `true`.

EXAMPLE See `PvmClassgroup` in `PvmClassgroup.kash` for an example.

NAME `PvmInit`

PURPOSE Starts KASH-Pvm on the master.

SYNTAX `PvmInit();`

EXAMPLE see function `PvmStartSlave`

NAME PvmKashIsSlave

PURPOSE true iff we are running as slave, false otherwise.

SYNTAX s := PvmKashIsSlave();

 boolean s

EXAMPLE

```
PvmKashIsSlave;
PvmInit();
PvmStartSlave("nicky");
Sleep(15);
PvmGet();
PvmGet();
PvmGet();
PvmSendNext("PvmKashIsSlave;\n");
Sleep(5);
PvmGet();
PvmExit();
Exec("echo halt | pvm"); \# kill pvmd on all hosts
```

NAME	PvmLengthOfQueue
PURPOSE	If called with no parameter, this function returns the maximal number of jobs that can be stored in queue. Otherwise the number is set to the parameter.
SYNTAX	<pre>PvmLengthOfQueue(n1); n2 := PvmLengthOfQueue(); integer n1, n2</pre>

NAME	PvmMaxRestartSlave
PURPOSE	If called with no parameter, this function returns the maximal number of times a slave may be restarted by security system. Otherwise the number is set to the parameter.
SYNTAX	<pre>PvmMaxRestartSlave(n1); n2 := PvmMaxRestartSlave(); integer n1, n2</pre>

NAME	PvmMaxRetransmitJob
PURPOSE	If called with no parameter, this function returns the maximal number of times a job may be retransmitted by security system. Otherwise the number is set to the parameter.
SYNTAX	<pre>PvmMaxRetransmitJob(n1); n2 := PvmMaxRetransmitJob(); integer n1, n2</pre>

NAME PvmPread

PURPOSE Reads and prints all arrived data in plain style, i.e. as provided by KASH.

SYNTAX PvmPread();

DESCRIPTION

SEE ALSO PvmGet,

EXAMPLE

NAME	<code>PvmRead</code>
PURPOSE	Reads (and prints) all data already received from the slave(s).
SYNTAX	<code>PvmRead()</code> ;
DESCRIPTION	The incoming data is “beautified” before printing.
SEE ALSO	PvmGet ,
EXAMPLE	

NAME	PvmSecurity
PURPOSE	If called with no parameter, this function returns the current status of the security system (activated or not). Otherwise it is set to the parameter.
SYNTAX	<pre>PvmSecurity(true false); b := PvmSecurity(); boolean b</pre>

NAME `PvmSendAll`

PURPOSE Sends data to all slaves.

SYNTAX `PvmSendAll(data, data, ...);`

DESCRIPTION Sends data to all slaves, corresponds to a broadcast. Only data is being send, no jobs, so that the slaves are not allowed to send an answer.
Master only: Note that no evaluation is done on the master, so if you send 1+2 the addition is performed on all slaves! The job count on the slaves will be unaffected.

EXAMPLE see function `PvmSendNext`

NAME	PvmSendLast
PURPOSE	Sends all parameters to the next free slave.
SYNTAX	<code>ok := PvmSendLast(data, data, ...);</code>
DESCRIPTION	<p>Master only: Note that no evaluation is done on the master, so if you send 1+2 the addition is performed on the slave!</p> <p>Data sent this way is not regarded as a new job for the security system, the slave is not allowed to send an answer to this. If there is no last slave it will return false, true otherwise.</p>
SEE ALSO	PvmSendNext ,
EXAMPLE	see function PvmSendNext

NAME	PvmSendNext
PURPOSE	Sends all parameters to the next free slave.
SYNTAX	PvmSendNext(data, data, ...);
DESCRIPTION	<p>Master only: Note that no evaluation is done on the master, so if you send 1+2 the addition is performed on the slave!</p> <p>Data sent this way will cause a new job to be started. The job count will be incremented by 1.</p> <p>For each job received from a PvmSendNext exactly one answer has to be send back.</p>
EXAMPLE	<pre>kash> PvmInit(); Error, KANT failure kash> PvmStartSlave(["marlin","julia"]);</pre>

NAME PvmSetPrintLevel

PURPOSE Set the `pvm_master->print_level` to `lev`. Only for debugging.

SYNTAX PvmSetPrintLevel(`lev`);

 small integer `lev`

EXAMPLE

```
PvmInit();  
PvmSetPrintLevel(1);  
PvmExit();
```

NAME `PvmShowBroadcastJobs`

PURPOSE Returns the formatstrings of the broadcast—job as a list. Intendend maily for debugging.

SYNTAX `PvmShowBroadcastJobs()` ;

EXAMPLE see function `PvmSendNext` and `PvmClear`

NAME	PvmSlaveError
PURPOSE	Slave only: Sends all data given as parameter to the master.
SYNTAX	<code>KashPvmError(data, data, ...);</code>
DESCRIPTION	Data send this way will not be regarded as an answer, no updates of the watch and the security system take place! Flag will be set to 2 on the receiving system.
SEE ALSO	PvmGet ,
EXAMPLE	see function PvmGet

NAME `PvmSlaveInfo`

PURPOSE Returns information of all Slaves in a list of records.

SYNTAX `PvmSlaveInfo()`;

EXAMPLE see function `PvmStartSlave` and `PvmStopSlave`

NAME	PvmSlavePrint
PURPOSE	Slave only: Sends all data given as parameter to the master.
SYNTAX	PvmSlavePrint(data, data, ...);
DESCRIPTION	Data send this way will not be regarded as an answer, no updates of the watch and the security system take place! Flag will be set to 4 on the receiving system.
SEE ALSO	PvmGet ,
EXAMPLE	see function PvmGet

NAME	PvmSlaveSend
PURPOSE	Slave only: Sends all data given as parameter to the master.
SYNTAX	PvmSlaveSend(data, data, ...);
DESCRIPTION	The data send this way is regarded as an answer, so the job count is decremented by one, and the watch (and the security system) is updated. Flag will be set to 1 on the receiving system.
SEE ALSO	PvmGet ,
EXAMPLE	see function PvmGet

NAME	PvmStartSlave		
PURPOSE	Starts the slave(s).		
SYNTAX	<pre>noSlaves := PvmStartSlave(num); noSlaves := PvmStartSlave(host);</pre>		
	integer	noSlaves	the number of started slaves
	integer	num	the number of slaves to start. Use 0 to start as many as possible.
	string	host	start one slave at host.
	list	L	List of hostnames to start one slave at.

EXAMPLE

```
PvmInit();
PvmStartSlave(["nicky","marlin"]);
PvmSlaveInfo();
PvmStartSlave("julia");
List(PvmSlaveInfo(),slave->[slave.host,slave.arch,slave.tid]);
PvmExit();
Exec("echo halt | pvm"); \# kill pvmd on all hosts
```

NAME	PvmStopSlave		
PURPOSE	Kills the slaves. If no argument is given, all slaves will be killed.		
SYNTAX	noSlaves := PvmStopSlave(); noSlaves := PvmStopSlave(stdid); noSlaves := PvmStopSlave(L);		
	integer	noSlaves	the number of killed slaves
	integer	stdid	slave-tid to kill
	list	L	List of slave-tids to kill

EXAMPLE

```

PvmInit();
PvmStartSlave(["markov","markov","julia","daisy","goofy"]);
a:=List(PvmSlaveInfo(),slave->[slave.host,slave.tid]);
a1:=a[1][2];
PvmStopSlave(a1);
a:=List(PvmSlaveInfo(),slave->[slave.host,slave.tid]);
PvmStopSlave([a[1][2],a[4][2]]);
a:=List(PvmSlaveInfo(),slave->[slave.host,slave.tid]);
PvmStopSlave();
a:=List(PvmSlaveInfo(),slave->[slave.host,slave.tid]);
PvmExit(); \# kill all kash-slaves
Exec("echo halt | pvm"); \# kill pvmd on all hosts

```

NAME	PvmStoreOrders
PURPOSE	Toggels/ Queries some internal flags.
SYNTAX	PvmStoreOrders

NAME PvmUse

PURPOSE Enables or disables the use of KANT-PVM. Initial state is false. If no argument is given, this function gets the status of PvmUse.

SYNTAX PvmUse(true|false);
 s := PvmUse();

 boolean s

EXAMPLE Enable KANT-PVM.

```
PvmUse(true);  
PvmUse();
```

NAME PvmUseMastersHost

PURPOSE Enables or disables the use of the master's host for PvmStartSlave. Initial state is false. If no argument is given, it gets the status of PvmUseMastersHost.

SYNTAX PvmUseMastersHost(true|false);
 s := PvmUseMastersHost();

 boolean s

EXAMPLE Enable PvmUseMastersHost.

```
PvmUseMastersHost(true);  
PvmUseMastersHost();
```

NAME	PvmUseMsg
PURPOSE	Enables or disables output for debugging.
SYNTAX	<pre>PvmUseMsg(true false); s := PvmUseMsg(); boolean s</pre>
DESCRIPTION	Initial state is false. If no argument is given, it gets the status of PvmUseMsg. The output is written to files of the name “pvm_master_uid.old” or “pvm_slave_uid.old” in the directory /tmp. <i>uid</i> is the user-id of the particular PVM-process.
EXAMPLE	Enable PvmUseMsg.

```
PvmUseMsg(true);  
PvmUseMsg();
```

NAME PvmUseWatch

PURPOSE Enables or disables the PVM-Watch for debugging. If h is given, the PVM-Watch starts on this host. Initial state is false. If no argument is given, it gets the status of PvmUseWatch.

SYNTAX PvmUseWatch(x, h);
 s := PvmUseWatch(x);
 s := PvmUseWatch();

 boolean x
 string h hostname
 boolean s

EXAMPLE Enable and disable the PVM-Watch.

```
PvmUseWatch(true);  
PvmUseWatch();  
PvmUseWatch(false);
```

Q

NAME	Q
PURPOSE	Predefined constant: Rational field \mathbb{Q} .
SYNTAX	Q; ring Q
DESCRIPTION	This is a predefined constant for the rational field \mathbb{Q} and is referenced by the variable Q. It is of arbitrary precision.
SEE ALSO	Z , R , C ,
EXAMPLE	

```
kash> Q;  
Rational Field
```


NAME	QfName
PURPOSE	missing shortdoc
SYNTAX	<pre>s := QfName(S);</pre> <p> string s polynomial algebra or quotient field S </p>
DESCRIPTION	Given a polynomial algebra or its quotient field this function returns the name of the (outmost) variable.

EXAMPLE

```
kash> Zx;
Univariate Polynomial Ring in x over Integer Ring
```

```
kash> QfName(Zx);
"x"
```

NAME	QfRank
PURPOSE	missing shortdoc
SYNTAX	<pre>n := QfRank(S);</pre> <p>integer n polynomial algebra or quotient field S</p>
DESCRIPTION	Given a polynomial algebra or its quotient field this function returns the number of variables.
EXAMPLE	

```
kash> Zxy := PolyAlg(Zx);
Univariate Polynomial Ring in y over Univariate Polynomial Ring in x over Integer Ring
```

```
kash> QfRank(Zxy);
2
```

NAME	QfScalarRing
PURPOSE	missing shortdoc
SYNTAX	<pre>A := QfScalarRing(S);</pre> <div> <div>ring</div> <div>A</div> <div>polynomial algebra or quotient field</div> <div>S</div> </div>
DESCRIPTION	Given a polynomial algebra or its quotient field this function returns its ring of scalars.
SEE ALSO	PolyAlgCoef ,
EXAMPLE	

```
kash> Zxy := PolyAlg(Zx);
Univariate Polynomial Ring in y over Univariate Polynomial Ring in x over Integer Ring
```

```
kash> QfScalarRing(Zxy);
Integer Ring
```

```
kash> QfScalarRing(QuotientField(Zxy));
Rational Field
```

NAME **QfeDen**

PURPOSE This function returns the denominator of a rational function (quotient field element). Works also for polynomials.

SYNTAX **d := QfeDen(f);**

 quotient field elements d,f

SEE ALSO **QfeNum, Num, Den,**

EXAMPLE

```
kash> k := FF(5, 2);
Finite field of size 5^2
kash> kx := PolyAlg(k);
Univariate Polynomial Ring in x over GF(5^2)

kash> x := Poly(kx, [1,0]);
x
kash> f := x;
x
kash> g := x+1;
x + 1
kash> h := f/g;
x/(x + 1)
kash> QfeDen(h);
x + 1
```

NAME	QfeDeriv
PURPOSE	Computes the derivation of an rational function.
SYNTAX	<pre>dhdT := QfeDeriv(h); qf elements dhdT, h</pre>
DESCRIPTION	This function computes dh/dT for $h \in k(T)$.
SEE ALSO	AlffDiff , PolyDeriv ,
EXAMPLE	

```
kash> AlffInit(FF(5,1));;
kash> h := 1/T;
1/T
kash> QfeDeriv(h);
4/T^2
```

NAME	QfeNum
PURPOSE	This function returns the numerator of a rational function (quotient field element). Works also for polynomials.
SYNTAX	<code>d := QfeNum(f);</code> quotient field elements d,f
SEE ALSO	QfeDen , Num , Den ,

EXAMPLE

```

kash> k := FF(5, 2);
Finite field of size 5^2
kash> kx := PolyAlg(k);
Univariate Polynomial Ring in x over GF(5^2)

kash> x := Poly(kx, [1,0]);
x
kash> f := x;
x
kash> g := x+1;
x + 1
kash> h := f/g;
x/(x + 1)
kash> QfeNum(h);
x

```

NAME	QfePthRoot
PURPOSE	Returns the p -th root of a rational function with scalars in a finite field of characteristic p , if existent.
SYNTAX	$r := \text{QfePthRoot}(a);$ <div style="display: flex; justify-content: space-between; margin-top: 10px;"> <div> $qf \text{ element or bool}$ $qf \text{ element}$ </div> <div> r a </div> <div> r p-th root of a or false </div> </div>
SEE ALSO	AlffEltPthRoot ,

EXAMPLE

```

kash> k := FF(3);
Finite field of size 3
kash> kx := PolyAlg(k);
Univariate Polynomial Ring in x over GF(3)

kash> kxy := PolyAlg(kx);
Univariate Polynomial Ring in y over Univariate Polynomial Ring in x over GF(3\
)

kash> X := Poly(kx, [1,0]);
x
kash> y := Poly(kxy, [1,0]);
y
kash> QfePthRoot(1/X^3);
1/x
kash> QfePthRoot(y);
false
kash> QfePthRoot(1/X + y^3);
false
kash> QfePthRoot(1/X^3 + y^3);
y + 1/x
kash> QfePthRoot(1/y^3);
1/y

```

NAME	QfeQf
PURPOSE	Returns the quotient field in which a rational function (quotient field element) is defined.
SYNTAX	<pre>F := QfeQf(a);</pre> <div> <div>quotient field</div> <div>quotient field element</div> <div>F</div> <div>a</div> </div>

SEE ALSO [QuotientField](#),

EXAMPLE

```
kash> k := FF(5, 2);
Finite field of size 5^2
kash> kx := PolyAlg(k);
Univariate Polynomial Ring in x over GF(5^2)

kash> x := Poly(kx, [1,0]);
x
kash> QfeQf(1/x);
Univariate rational function field over GF(5^2)
Variables: x
```


NAME	QfeVal
PURPOSE	Computing valuations in a rational function field.
SYNTAX	$v := \text{QfeVal}(p, a);$ <div style="margin-left: 100px;"> integer v qf elements p, a </div>
DESCRIPTION	Given the rational function field $k(x)$ and a prime element $p \in k(x)$, that is a prime polynomial or $1/x$, compute the valuation $v_p(a)$ of a at the place given by p .
SEE ALSO	InftyVal , PolyFactor ,
EXAMPLE	

```

kash> k := FF(5, 2);
Finite field of size 5^2
kash> kx := PolyAlg(k);
Univariate Polynomial Ring in x over GF(5^2)

kash> x := Poly(kx, [1,0]);
x
kash> QfeVal(x, x^3);
3
kash> QfeVal(1/x, x^3);
-3
kash> QfeVal(x, (x+1)^3);
0
kash> QfeVal(1/x, (x+1)^3);
-3

```

NAME	Qp
PURPOSE	Creates the p -adic field \mathbb{Q}_p
SYNTAX	<pre>F := Qp(p[,n]);</pre> <p> p-adic field F prime number p integer n </p>
DESCRIPTION	This function creates the p -adic Field \mathbb{Q}_p with given precision n . If no precision is given it is set to 20.
SEE ALSO	IsQp , QpElt , QpEltToQ , QpEltQp , QpPrec , QpExp , QpLog , QpPrime , QpSqrt , QpValuation ,
EXAMPLE	

```
kash> F := Qp(2);
2-adic Field mod 2^20
```

NAME	QpElt
PURPOSE	Returns an element of \mathbb{Q}_p .
SYNTAX	$k := \text{QpElt}(F, n);$ <div style="display: flex; justify-content: space-between;"> <div>p-adic field</div> <div>F</div> </div> <div style="display: flex; justify-content: space-between;"> <div>integer or rational</div> <div>n</div> </div> <div style="display: flex; justify-content: space-between;"> <div>p-adic element</div> <div>k</div> </div>
DESCRIPTION	This function returns the series of an integral or rational number, e.g. an element k of the p -adic field.
SEE ALSO	IsQp , Qp , QpEltToQ , QpEltQp , QpExp , QpLog , QpPrec , QpPrime , QpSqrt , QpValuation ,

EXAMPLE

```

kash> F := Qp(3);
3-adic Field mod 3^20
kash> q := 1/2;
1/2
kash> k := QpElt(F, q);
2 + 3 + 3^2 + 3^3 + 3^4 + 3^5 + 3^6 + 3^7 + 3^8 + 3^9 + 3^10 + 3^11 + 3^12 + 3\
^13 + 3^14 + 3^15 + 3^16 + 3^17 + 3^18 + 3^19 + 0(3^20)

```

NAME QpEltQp

PURPOSE This function creates the p -adic field of the element k .

SYNTAX F := QpEltQp(k);

p -adic field element k

p -adic field F

SEE ALSO IsQp, Qp, QpElt, QpEltToQ, QpPrec, QpExp, QpLog, QpPrime,
QpSqrt, QpValuation,

EXAMPLE

```
kash> F := Qp(3);
3-adic Field mod 3^20
kash> k := QpElt(F, 44);
2 + 2*3 + 3^2 + 3^3
kash> F := QpEltQp(k);
3-adic Field mod 3^20
```

NAME	QpEltToQ
PURPOSE	Computes the rational number q of an element in \mathbb{Q}_p .
SYNTAX	<pre>q := QpEltToQ(k);</pre> <p> p-adic field F p-adic element k rational number q </p>
SEE ALSO	IsQp , Qp , QpElt , QpEltQp , QpExp , QpLog , QpPrec , QpPrime , QpSqrt , QpValuation ,
EXAMPLE	

```

kash> F := Qp(3);
3-adic Field mod 3^20
kash> k := QpElt(F, 27);
3^3
kash> q := QpEltToQ(k);
27

```

NAME	QpExp
PURPOSE	Computes the exponent function of an element in \mathbb{Q}_p .
SYNTAX	<pre>exp:= QpExp(k);</pre> <p style="margin-left: 40px;"> p-adic element k p-adic element exp </p>
DESCRIPTION	The exponent function <i>exp</i> of the element k of a p -adic field is returned. If $prec$ is the used precision, the result is correct modulo p^{prec} .
SEE ALSO	IsQp , Qp , QpElt , QpEltToQ , QpEltQp , QpLog , QpPrec , QpPrime , QpSqrt , QpValuation ,

EXAMPLE

```
kash> F := Qp(3);
3-adic Field mod 3^20
kash> k := QpElt(F, 18);
2*3^2
kash> exp := QpExp(k);
1 + 2*3^2 + 2*3^4 + 3^5 + 3^6 + 2*3^7 + 2*3^9 + 3^10 + 3^11 + 2*3^13 + 3^17 + \
2*3^18 + 2*3^19 + 0(3^20)
```

NAME	QpLog
PURPOSE	Computes the logarithm of an element in \mathbb{Q}_p .
SYNTAX	$l := \text{QpLog}(k);$ <p style="margin-left: 40px;"> p-adic element k p-adic element l </p>
DESCRIPTION	This function returns the logarithm l of the element k of a p -adic field. If $prec$ is the used precision, the result is correct modulo p^{prec} .
SEE ALSO	IsQp , Qp , QpElt , QpEltToQ , QpEltQp , QpPrec , QpExp , QpPrime , QpSqrt , QpValuation ,
EXAMPLE	

```

kash> F := Qp(3);
3-adic Field mod 3^20
kash> k := QpElt(F, 18);
2*3^2
kash> l := QpLog(k);
2*3 + 2*3^2 + 3^5 + 3^6 + 2*3^8 + 3^9 + 2*3^10 + 3^12 + 2*3^13 + 3^14 + 2*3^16\
+ 2*3^17 + 0(3^21)

```

NAME QpPrec

PURPOSE Prints or sets the precision of a p -adic field.

SYNTAX `k:= QpPrec(F [,n]);`

 p-adic field F

 integer n

 integer k

SEE ALSO [IsQp](#), [Qp](#), [QpElt](#), [QpEltToQ](#), [QpEltQp](#), [QpExp](#), [QpLog](#), [QpPrime](#),
[QpSqrt](#), [QpValuation](#),

EXAMPLE

```
kash> F:= Qp(2);
2-adic Field mod 2^20
kash> k:= QpPrec(F, 30);
30
```


NAME	QpPrime
PURPOSE	This function creates the prime number which defines the p -adic field.
SYNTAX	<pre>p := QpPrime(F);</pre> <p>p-adic field F prime number p</p>
SEE ALSO	IsQp, Qp, QpElt, QpEltToQ, QpEltQp, QpPrec, QpExp, QpLog, QpSqrt, QpValuation,
EXAMPLE	

```
kash> F := Qp(2);
2-adic Field mod 2^20
kash> p := QpPrime(F);
2
```

NAME	QpSqrt
PURPOSE	Computes the square root of an element in \mathbb{Q}_p .
SYNTAX	<pre>s := QpSqrt(k);</pre> <p><i>p</i>-adic element k <i>p</i>-adic element s</p>
DESCRIPTION	This function returns the square root <i>s</i> of the element <i>k</i> of a <i>p</i> -adic field. If <i>prec</i> is the used precision, the result is correct modulo p^{prec} .
SEE ALSO	IsQp , Qp , QpElt , QpEltToQ , QpEltQp , QpPrec , QpExp , QpLog , QpPrime , QpValuation ,

EXAMPLE

```
kash> F := Qp(3);
3-adic Field mod 3^20
kash> k := QpElt(F, 19);
1 + 2*3^2
kash> s := QpSqrt(k);
1 + 3^2 + 3^4 + 3^5 + 2*3^8 + 3^9 + 3^10 + 2*3^11 + 3^13 + 2*3^15 + 3^16 + 2*3^18 + 2*3^19 + 0(3^20)
```

NAME	QpValuation
PURPOSE	Computes the valuation of an element in \mathbb{Q}_p .
SYNTAX	$v := \text{QpValuation}(F, k);$ <div style="margin-left: 100px;"> p-adic field F p-adic element k integer v </div>
SEE ALSO	IsQp , Qp , QpElt , QpEltToQ , QpEltQp , QpExp , QpLog , QpPrec , QpPrime , QpSqrt ,
EXAMPLE	

```

kash> F := Qp(3);
3-adic Field mod 3^20
kash> k := QpElt(F, 27);
3^3
kash> v := QpValuation(k);
3

```

QuotientField

NAME `QuotientField`

PURPOSE missing shortdoc

SYNTAX `QF := QuotientField(S);`

 quotient field `QF`
 ring `S`

DESCRIPTION Creates the quotient field of a ring, intended for polynomial algebras.

SEE ALSO [PolyAlg](#),

EXAMPLE Creating the quotient field of a polynomial algebra over a finite field:

```
kash> Fp := FF(5);  
Finite field of size 5  
kash> Fpx := PolyAlg(Fp);  
Univariate Polynomial Ring in x over GF(5)  
  
kash> Fpxf := QuotientField(Fpx);  
Univariate rational function field over GF(5)  
Variables: x
```

NAME	R
PURPOSE	Predefined constant: Real field \mathbb{R} .
SYNTAX	R ; ring R
DESCRIPTION	This is a predefined constant for the real field \mathbb{R} and is referenced by the variable R . The precision can be set or displayed with Prec . Default precision is 50.
SEE ALSO	Z , Q , C , Prec ,
EXAMPLE	

```
kash> R;  
Real Field of precision 52
```

NAME RandomEcc

PURPOSE Returns a random elliptic curve in Weierstrass form.

SYNTAX `E := RandomEcc(F);`

elliptic curve E

finite field F

DESCRIPTION

EXAMPLE

EXAMPLE

```
kash> p:=65112*2^144-1;
1452046121366725933991673688168680114377396846591
kash> IsPrime(p);
true
kash> RandomEcc(FF(p));
[ 1042675512671975645588756792447528436479728692810,
  932028267535118580676757976094182914034159852790 ]
```

NAME	RandomElt		
PURPOSE	Computes a “random” element of a given order.		
SYNTAX	$\alpha := \text{RandomElt}(R \text{ [, } l \text{ } b, \text{ deg degl}]);$		
	element	α	of o
	ring	R	may be an order, an ideal, \mathbb{Z} , a module, a polynomial ring or a function field order.
	list	l	of integers
	integer	b	equivalent to $l := [-b..b]$
	integer	deg	degree of a random element of the polynomial algebra R
	list	degl	of positive integers
DESCRIPTION	<p>A “random” element <i>alpha</i> with coefficients in l will be returned (by default $l := [-20, 20]$). Of course, it is not possible to do a truly random element from an infinite set.</p> <p>If R is a polynomial algebra, and you want to create a random element with a certain degree deg, then you also have to give l or b as a parameter to <code>RandomElt</code>. If deg is a list, then the degree of <i>alpha</i> will be an element of deg. If R is a function field order, then deg is the range of the degree of the coefficients of α, and l the range of coefficients of these coefficients (only for absolute extensions in the moment). By default the degree is an integer between 0 and 20.</p>		
EXAMPLE	<pre> kash> o := Order(Z, 5, 3); Generating polynomial: x^5 - 3 kash> RandomElt(o); [19, 11, 5, -6, -18] kash> RandomElt(o, 100); [-29, 86, 87, 22, -77] kash> RandomElt(o, [-1,1]); [-1, -1, -1, -1, 1] </pre>		

NAME	RandomIdeal
PURPOSE	A random ideal over a given order is computed.
SYNTAX	<pre>id := RandomIdeal(o); order o ideal id</pre>
DESCRIPTION	
EXAMPLE	<pre>kash> o := OrderMaximal(Z, 5, 3); Generating polynomial: x^5 - 3 Discriminant: 253125 kash> id := RandomIdeal(o); < [30221858653155826767789393356661370243131978936076711612754461465704355739193\ 612475566786777906963891586524359057988695861475664454763392079742685961456156\ 9623232086679843444703859644740972080771484375 8520476934232591240674785026940\ 581068274373216041187587415209522620502901537032050010693225189975621019021915\ 524691019487833146776006257482144719065601654421429750197586366266003190664428\ 5115888671875 2381637602482023254700118888715903986676201167796568301136750691\ 825768378884621678290168539796005136081164082124091096917619085410227738247306\ 34060844190397661169593082466208639999978144531926289062500 205138702125977431\ 920342332195820956347160383190056413001838875304965300318916534427589020374104\ 635233384856226444046873115022718843925669667353437476686014979730785133912425\ 46257200244792525193359375 277769736762008522310759066210171971717262570056649\ 960023794302025128023430166336388999617905688010694521398459742446347915727835\ 287627768010124548394338139315936251011410336321991985185180593656250000] [0 617906921611055298621727122045948267927573408240201614029487224472609818283\ 702859352888052619378799942003010050802073046059120389872921984888882826142517\ 12668121431813823271484375 0 2471627686444221194486908488183793071710293632960\ 806456117948897890439273134811437411552210477515199768012040203208292184236481\ 559491687939555313045700685067248572725529308593750 1523614512402451042975727\ 550080670511383123929388023010516649023488237055499904667962148812869101469800\ 3988506012737671733219655573087713568099319630429480716279606342336164062500] [0 0 6179069216110552986217271220459482679275734082402016140294872244726098182\ 837028593528880526193787999420030100508020730460591203898729219848888828261425\ 1712668121431813823271484375 1235813843222110597243454244091896535855146816480\</pre>


```
403228058974448945219636567405718705776105238757599884006020101604146092118240\  
7797458439697777656522850342533624286362764654296875 5236738144655066600424429\  
584652494977523477732689116091261137434737377598973805051150653892245491459124\  
0224151737804141789496164296487346917600159181767975872641726792257248046875]  
[0 0 0 12358138432221105972434542440918965358551468164804032280589744489452196\  
365674057187057761052387575998840060201016041460921182407797458439697777656522\  
850342533624286362764654296875 16425823889565787100790840234242390693672473764\  
862421365217535087150211602695303929742010807150070841791067627570060156602027\  
46487079386135363732060428168073754840037437089843750]  
[0 0 0 0 803628477662010217173357389370142650855419626318440491520629646672245\  
144719795607173644668491194018865334378398584170064876074355033808838298070701\  
7644456828732421875]
```

>

NAME RandomMatrix

PURPOSE Using RandomMatrix a square matrix with “random” entries is returned.

SYNTAX `m := RandomMatrix(R, n, l);`

`matrix` `m` in $R^{n \times n}$
 `ring` `R` may be an order, an ideal, \mathbb{Z} , a module or an function field
 order.
 `integer` `n`
 `list` `l` of possible (coefficients of the) entries.

DESCRIPTION

EXAMPLE

```
kash> m := RandomMatrix(Z, 5);
[ 19  11   5  -6 -18]
[ -6  17  18   5 -16]
[ -1 -16  -4  -3  13]
[ 17  11  -9  11  18]
[ -5   3  -7  12  -3]
kash> m := RandomMatrix(Z, 5);
[  7  17  -6  -9   5]
[ 11   3 -10 -18  10]
[ 17  16 -14   3 -15]
[ -2   0   3  -9  10]
[  1 -11   6   9  -4]
```

NAME	RandomOrder
PURPOSE	Using RandomOrder a random order is computed.
SYNTAX	<pre>o := RandomOrder(R, n [, 1]);</pre> <p> order o ring R may be \mathbb{Z}, an order or an functions field order. integer n degree list 1 </p>

DESCRIPTION

EXAMPLE

```
kash> o := RandomOrder(Z, 4);
Generating polynomial: x^4 + 19*x^3 + 11*x^2 + 5*x - 6
```

NAME	RandomPoly
PURPOSE	Produces a monic random polynomial of given degree.
SYNTAX	$f := \text{RandomPoly}(R, d [, 1]);$ <div> <div>polynomial</div> <div>ring</div> <div>integer</div> <div>list</div> </div> <div> <div>f</div> <div>R</div> <div>d</div> <div>1</div> </div> <div> <div>in $R[x]$</div> <div>may be an order, an ideal, \mathbb{Z}, a module or an function field order.</div> <div>degree of f</div> <div>of possible (coefficients of the) coefficients.</div> </div>

DESCRIPTION

EXAMPLE

```

kash> f := RandomPoly(Z, 3);
x^3 + 19*x^2 + 11*x + 5
kash> o := Order(f);
Generating polynomial: x^3 + 19*x^2 + 11*x + 5

kash> g := RandomPoly(o, 4);
x^4 + [-6, -18, -6]*x^3 + [17, 18, 5]*x^2 + [-16, -1, -16]*x + [-4, -3, 13]
```

NAME	RationalReconstruct
PURPOSE	Lifts an integer modulo m to a rational.
SYNTAX	<pre>q := RationalReconstruct (u,m);</pre> <pre> rational q integer u integer m </pre>
DESCRIPTION	<p>Given positive integers u, m, the function <code>RationalReconstruct</code> computes a rational number $q = \frac{a}{b}$ such that $a \equiv bu \pmod{m}$ and $0 \leq a , b < \sqrt{\frac{m}{2}}, b \neq 0$, if such a pair exists. Otherwise the 0 is returned.</p>
SEE ALSO	EltReconstruct ,
EXAMPLE	

```
kash> RationalReconstruct(17,49);
2/3
```

NAME	RayCantoneseRemainder										
PURPOSE	missing shortdoc										
SYNTAX	<pre>elt := RayCantoneseRemainder(m0,minf,elt0,sig);</pre> <table> <tr> <td>ideal</td><td>m0</td></tr> <tr> <td>list of integers/infinite primes</td><td>minf</td></tr> <tr> <td>algebraic number</td><td>elt0</td></tr> <tr> <td>list</td><td>sig</td></tr> <tr> <td>algebraic number</td><td>elt</td></tr> </table>	ideal	m0	list of integers/infinite primes	minf	algebraic number	elt0	list	sig	algebraic number	elt
ideal	m0										
list of integers/infinite primes	minf										
algebraic number	elt0										
list	sig										
algebraic number	elt										

DESCRIPTION Returns an algebraic number `elt`, which is congruent to `elt0` (mod `m0`) and whose real conjugates have the signature as chosen by `sig` at the places of `minf`. The algorithm is described in [Pau96]. `sig` is a vector/matrix, which consists of 0's and 1's. A 1 in the *i*'th position means that `elt` has a negative conjugate at the *i*'th place of `minf` (which must be sorted!), a 0 means that it has positive conjugate.

SEE ALSO [EltApproximation](#), [EltCon](#), [IdealChineseRemainder](#), [OrderSig](#), [RayResidueRing](#),

EXAMPLE

```
kash> O := OrderMaximal(Z,4,5);
      F[1]
      |
      F[2]
      /
      /
      Q
F  [ 1]      Given by transformation matrix
F  [ 2]      x^4 - 5
Discriminant: -2000

kash> m0 := 12*O;; minf := [1,2];;
kash> elt0 := Elt(O,[1,2,3,4]);;sig := [1,0];;IsMat(sig);;
kash> elt := RayCantoneseRemainder(m0,minf,elt0,sig);
[5029, -2590, 3591, -2300]
kash> elt mod m0;
[1, 2, 3, 4]
kash> EltCon(elt);
```

$$\begin{aligned} & [-5910.979315905677215736970806158202922913622694351555 \ 12964.7430748579057138 \backslash \\ & 0121401249028054286746099661 \ 640.118120523885750967878396833961190023080848871 \backslash \\ & 131 - 1747.347688152938188731912066849790464428137141477991*i \ 640.118120523885 \backslash \\ & 750967878396833961190023080848871131 + 1747.3476881529381887319120668497904644 \backslash \\ & 28137141477991*i] \end{aligned}$$

NAME	RayClassFieldAbelianTest
PURPOSE	Tests if a class field is also abelian over \mathbb{Q} .
SYNTAX	<pre>a:=RayClassFieldIsAbelian(G [,m]);</pre> <p>AbelianGroup G defining a class group boolean a integer m</p>
DESCRIPTION	Let \mathcal{O} be the ring of integers of an abelian extension over \mathbb{Q} . \mathfrak{m} is a multiple of its conductor $\mathfrak{f} \in \mathbb{Z}$. G is a class group in \mathcal{O} , K the abelian extension over \mathcal{O} defined by G . The function determines if K is also abelian over \mathbb{Q} . The methode is described in [Has70, II, p.24].

EXAMPLE

```
kash> o:=OrderMaximal(x^2-10);
Generating polynomial: x^2 - 10
Discriminant: 40

kash> G:=RayClassGroupToAbelianGroup(9*o);
RayClassGroupToAbelianGroup(<9>, [ ])
Group with relations:
[2 0]
[0 3]

kash> RayClassFieldAbelianTest(G);
true
```

Let's check this. First we'll compute the RayClassField

```
kash> L:=RayClassField(G);
[ x^2 - 2, x^3 - 3*x - 1 ]
```

The only critical part is the cubic polynomial, therefore we'll show that this polynomial defines a $C|3$ extension over \mathbb{Q} .


```
kash> g:=PolyMove(L[2], Z);
x^3 - 3*x - 1
kash> OrderAutomorphismsAbel(Order(g));
true
kash> o:=OrderMaximal(x^4-2);
Generating polynomial: x^4 - 2
Discriminant: -2048

kash> G:=RayClassGroupToAbelianGroup(9*o);
RayClassGroupToAbelianGroup(<9>, [ ])
Group with relations:
[3 0]
[0 3]
kash> RayClassFieldIsAbelian(G);
false
```

NAME	RayClassFieldArtin
PURPOSE	Given an ideal, this function computes the corresponding automorphism.
SYNTAX	<pre>aut := RayClassFieldArtin(id, 0);</pre> <p>an automorphism aut ideal id coprime to the defining module order 0 output of RayClassFieldAuto</p>
DESCRIPTION	<p>This function essentially provides $(\mathfrak{a}, O/o) \in \text{Gal}(O/o)$ for given unramified ideals. Note, that this function won't check if the conductor of the abelian extension is known. Therefore it is only possible to compute automorphisms for ideals coprime to the defining module.</p> <p>Note, that this function won't work on an arbitrarily defined abelian extensions O of o. It is necessary to compute O using RayClassFieldAuto.</p>
SEE ALSO	RayClassFieldAuto , RayClassField ,

EXAMPLE

```
kash> o := OrderMaximal(Z, 2, 10);
Generating polynomial: x^2 - 10
Discriminant: 40

kash> i := 9*o;
<9>
kash> O := RayClassFieldAuto(RayClassField(i))[1];
      F[1]
      /
      /
      E1[1]
      /
      /
      Q
F  [ 1]      x^6 - 12*x^4 - 2*x^3 + 21*x^2 - 6*x - 1
E 1[ 1]      x^2 - 10
Discriminant: <17901347328>

kash> l := RayClassGroupCyclicFactors(i);
[ [ <[-4, -6], [-30, -2]>, 2 ], [ <[-2, -3]>, 3 ] ]
kash> ap := RayClassFieldArtin(l[1][1], 0);
```

```
a
kash> ap2 := RayClassFieldArtin(1[1][1]^2, 0);
e
kash> ap^2 = ap2;
true
kash> OrderAutomorphisms(o);
[ [0, 1], [0, -1] ]
kash> RayClassFieldArtin(1[2][1], 0);
b*b
kash> RayClassFieldArtin(IdealAutomorphism(1[2][1], 2), 0);
b*b
```

NAME	RayClassFieldAuto
PURPOSE	Given the output of <code>RayClassField</code> , this function computes a primitive element for the Ray Class Field together with the automorphisms.
SYNTAX	<pre>L := RayClassFieldAuto(c);</pre> <p>list L containing 0 and a list of automorphisms list c output of <code>RayClassField</code></p>
DESCRIPTION	<p>Starting with the data computed by <code>RayClassField</code>, this function first computes a primitive element for \mathbb{Q}/\mathfrak{o}. Next all the automorphisms of \mathbb{Q}/\mathfrak{o} are computed, together with information allowing one to use the Artin isomorphism.</p> <p>If automorphisms of the base field (\mathfrak{o}) are known (prior to the call of <code>RayClassField</code>), and if they extend to \mathbb{Q}/\mathbb{Q} they to are computed.</p>
SEE ALSO	RayClassField ,
EXAMPLE	We will consider various extensions of $\mathbb{Q}(\sqrt{10})$ to demonstrate some effects and possibilities:

```
kash> o := OrderMaximal(Z, 2, 10);
Generating polynomial: x^2 - 10
Discriminant: 40
```

NAME	RayClassFieldIsAbelian
PURPOSE	Tests if a class field is also abelian over \mathbb{Q} .
SYNTAX	$a := \text{RayClassFieldIsAbelian}(G \text{ [,m]});$ AbelianGroup G defining a class group boolean a integer m
DESCRIPTION	Let \mathcal{O} be the ring of integers of an abelian extension over \mathbb{Q} . m is a multiple of its conductor $f \in \mathbb{Z}$. G is a class group in \mathcal{O} , K the abelian extension over \mathcal{O} defined by G . The function determines if K is also abelian over \mathbb{Q} .

EXAMPLE

```

kash> o:=OrderMaximal(x^2-10);
Generating polynomial: x^2 - 10
Discriminant: 40

kash> G:=RayClassGroupToAbelianGroup(9*o);
RayClassGroupToAbelianGroup(<9>, [ ])
Group with relations:
[2 0]
[0 3]
kash> m:=Disc(o);
40
kash> RayClassFieldIsAbelian(G,m);
true

```

Let's check this. First we'll compute the RayClassField

```

kash> L:=RayClassField(G);
[ x^2 - 2, x^3 - 3*x - 1 ]

```

The only critical part is the cubic polynomial, therefore we'll show that this polynomial defines a $C|3$ extension over \mathbb{Q} .

```
kash> g:=PolyMove(L[2], Z);  
x^3 - 3*x - 1  
kash> OrderAutomorphismsAbel(Order(g));  
true  
kash> o:=OrderMaximal(x^4-2);  
Generating polynomial: x^4 - 2  
Discriminant: -2048  
  
kash> G:=RayClassGroupToAbelianGroup(9*o);  
RayClassGroupToAbelianGroup(<9>, [  ])  
Group with relations:  
[3 0]  
[0 3]  
kash> RayClassFieldIsAbelian(G);  
false
```

NAME	RayClassFieldIsCentral		
PURPOSE	Tests whether the Ray Class Field defined by the given data will be a central extension.		
SYNTAX	flag := RayClassFieldIsCentral(G [, 1]);		
	boolean	flag	
	AbelianGroup	G	a quotient from RayClassGroupToAbelianGroup
	list	1	of automorphisms, if not present all known automorphisms are used.

DESCRIPTION

EXAMPLE We'll investigate the ray class field mod $m := (6)\mathfrak{p}|1^\infty\mathfrak{p}|2^\infty$ in $k := \mathbb{Q}(\sqrt{10})$:

```
kash> o := OrderMaximal(Z, 2, 10);
Generating polynomial: x^2 - 10
Discriminant: 40

kash> OrderAutomorphisms(o);
[ [0, 1], [0, -1] ]
kash> G1 := RayClassGroupToAbelianGroup(6*o, [1, 2]);
RayClassGroupToAbelianGroup(<6>, [ 1, 2 ])
Group with relations:
[2 0 0]
[0 4 0]
[0 0 2]
kash> rcg := FindMaximalCentralField(G1);
[2 0 0]
[0 2 0]
[0 0 2]
[0 0 0]
[0 0 0]
[0 0 0]
kash> G2 := AbelianQuotientGroup(G1, AbelianSubGroup(G1, rcg));
Group with relations:
[2 0 0]
[0 2 0]
[0 0 2]
[0 0 0]
[0 0 0]
[0 0 0]
```

[2 0 0]

[0 4 0]

[0 0 2]

kash> RayClassFieldIsCentral(G1);

false

kash> RayClassFieldIsCentral(G2);

true

NAME	RayClassFieldIsNormal		
PURPOSE	Tests whether the Ray Class Field defined by the given data will be a normal extension.		
SYNTAX	<pre>flag := RayClassFieldIsNormal(G [, 1]);</pre>		
	boolean	flag	
	AbelianGroup	G	a quotient from RayClassGroupToAbelianGroup
	list	1	of automorphisms, if not present all known automorphisms are used.
DESCRIPTION			
EXAMPLE	See RayClassFieldSplittingField for an example.		

NAME	RayClassFieldSplittingField		
PURPOSE	Computes data to get the splitting field of the Ray Class Field.		
SYNTAX	sg := RayClassFieldSplittingField(G, 1);		
	AbelianGroup	sg	
	AbelianGroup	G	quotient of RayClassGroupToAbelianGroup
	list	1	of automorphisms. Must be complete, not just generators.

DESCRIPTION

EXAMPLE

```

kash> o := OrderMaximal(Z, 2, 10);
Generating polynomial: x^2 - 10
Discriminant: 40

kash> OrderAutomorphisms(o);
[ [0, 1], [0, -1] ]
kash> p3 := Factor(3*o)[2][1];
<3, [2, 1]>
kash> G1 := RayClassGroupToAbelianGroup(p3, [2]);
RayClassGroupToAbelianGroup(<3, [2, 1]>, [ 2 ])
Group with relations:
[2]
kash> G2 := RayClassFieldSplittingField(G1);
Group with relations:
[2 0]
[0 1]
[4 0]
[0 2]
kash> RayClassFieldIsNormal(G1);
true
kash> RayClassFieldIsNormal(G2);
true
kash> O1 := RayClassField(G1);
[ x^2 - 2 ]
kash> Galois(OrderAbs(Order(O1[1])));
"E(4)"

```

NAME	RayClassGroup
PURPOSE	Returns the ray class number and the orders of the generators of the ray class group modulo a congruence module.
SYNTAX	<pre> L := RayClassGroup(m0 [,minf]); L := RayClassGroup(o [,minf]); list L ideal m0 list minf of integers/infinite primes order o </pre>
DESCRIPTION	<p>Computes the ray class group modulo a congruence module $\mathfrak{m} = \mathfrak{m}_0 \mathfrak{m}_\infty$, where \mathfrak{m}_0 is an integral ideal of a maximal order \mathcal{O} and \mathfrak{m}_∞ is a formal product of primes at infinity — here a subset of the real embeddings of \mathcal{O}. \mathfrak{m}_∞ is represented by a list of integers, where each entry represents a real embedding (as given by <code>EltCon</code>). <code>RayClassGroup</code> returns a list consisting of the ray class number and a list of the orders of the cyclic factors of the ray class group.</p> <p>It might be necessary to use a higher precision (set by <code>OrderPrec</code>) in order to get correct results when working with primes at infinity. Computing the class group and the units of the maximal order first can speed up the computation of the ray class group, as it is possible to influence the calculation of the class group, for instance by setting a lower bound for the regulator. For a description of the algorithm see [Pau96, PP98].</p>
SEE ALSO	<code>EltCon</code> , <code>IdealRayClassRep</code> , <code>OrderClassGroup</code> , <code>OrderPrec</code> , <code>RayClassGroupCyclicFactors</code> , <code>RayConductor</code> ,

EXAMPLE

```

kash> O := OrderMaximal(Order(Z,2,10));;
kash> OrderClassGroup(O,500,"euler","fast");
[ 2, [ 2 ] ]
kash> m0 := 27*O;;
kash> minf := [1,2];;
kash> L := RayClassGroup(m0,minf);
[ 72, [ 36, 2 ] ]
kash> o:=OrderMaximal(Poly(Zx,[1,1,1]));
Generating polynomial: x^2 + x + 1
Discriminant: -3

kash> L:=RayClassGroup(o);
[ 1, [ 1 ] ]

```

compute the ray class group of an ideal in \mathbb{Z}

```
kash> a := ZIdealCreate(9);  
<9>  
kash> RayClassGroup(a);  
[ 3, [ 3 ] ]
```

NAME	RayClassGroupCyclicFactors
PURPOSE	Returns the generators and the orders of the generators of the ray class group modulo a congruence module as computed using RayClassGroup.
SYNTAX	<pre>L := RayClassGroupCyclicFactors(m0 [,minf]);</pre> <p>list L</p> <p>ideal m0</p> <p>list minf of integers/infinite primes</p>
SEE ALSO	EltCon , IdealRayClassRep , OrderClassGroup , RayClassGroup ,

EXAMPLE

```
kash> O := OrderMaximal(Order(Z,2,10));;
kash> OrderClassGroup(O,500,"euler");
[ 2, [ 2 ] ]
kash> m0 := 7*13*11*O;;
kash> L := RayClassGroupCyclicFactors(m0);
[ [ <[184, 728], [3640, 92]>, 12 ], [ <[463, -77]>, 12 ],
  [ <[92, 364]>, 120 ] ]
kash> IdealRayClassRep(L[1][1],m0);
[1 0 0]
kash> IdealRayClassRep(L[2][1],m0);
[0 1 0]
kash> IdealRayClassRep(L[3][1],m0);
[0 0 1]
```

NAME	RayClassGroupToAbelianGroup																	
PURPOSE	Returns the ray class group for a congruence module.																	
SYNTAX	<pre>g := RayClassGroupToAbelianGroup(m0 [, minf] [, rels expo]);</pre> <table><tr><td>group</td><td>g</td><td></td></tr><tr><td>ideal</td><td>m0</td><td></td></tr><tr><td>list of integers</td><td>minf</td><td>infinite primes</td></tr><tr><td>matrix of integers</td><td>rels</td><td>gives additional relations for a quotient</td></tr><tr><td>integer</td><td>expo</td><td>equivalent to rels = expo*MatId.</td></tr></table>			group	g		ideal	m0		list of integers	minf	infinite primes	matrix of integers	rels	gives additional relations for a quotient	integer	expo	equivalent to rels = expo*MatId.
group	g																	
ideal	m0																	
list of integers	minf	infinite primes																
matrix of integers	rels	gives additional relations for a quotient																
integer	expo	equivalent to rels = expo*MatId.																
DESCRIPTION	Returns the ray class group for a congruence module as an abstract group g . For more information see RayClassGroup .																	
SEE ALSO	RayClassGroup ,																	
EXAMPLE																		

```
kash> o := OrderMaximal(x^2+6*x+2);
Generating polynomial: x^2 + 6*x + 2
Discriminant: 28
```

```
kash> m0 :=Elt(o, [0,3])*o;
<[0, 3]>
kash> g := RayClassGroupToAbelianGroup(m0, [1,2]);
RayClassGroupToAbelianGroup(<[0, 3]>, [ 1, 2 ])
Group with relations:
[2 0]
[0 2]
```

NAME	RayConductor
PURPOSE	Calculates the conductor of the ray class group modulo a congruence module.
SYNTAX	<pre>L := RayConductor(m0 [, minf] [, rels]);</pre> <p> list L ideal m0 list minf of integers/infinite primes matrix rels relation matrix over \mathbb{Z} </p>
DESCRIPTION	<p>Calculates the minimal ideal and the minimal set of infinite primes with the same ray class group as the given congruence module. The conductor is represented by a list of an ideal and a list of primes at infinity — the numbers correspond to the real embeddings of the order.</p> <p>This is done by a careful analysis of the structure of the ray class group as described in [Pau96, PP98].</p>
SEE ALSO	EltCon , EltRayResidueRingRep , RayResidueRing , RayClassGroup ,
EXAMPLE	

```

kash> O := OrderMaximal(Order(x^3+9*x^2-8*x-9));;
kash> m0 := 94*O;;
kash> minf:= [1,2];;
kash> L := RayConductor(m0,minf);
[ <47>, [ ] ]
kash> OrderClassGroup(O,500,euler,fast);
[ 1, [ 1 ] ]
kash> RayClassGroup(m0,minf);
[ 23, [ 23 ] ]
kash> RayClassGroup(L[1],L[2]);
[ 23, [ 23 ] ]

```

NAME	RayConductorTest
PURPOSE	Tests iff the given module is the true conductor.
SYNTAX	<pre>b := RayConductorTest(m0 [, minf] [, rels]);</pre> <p> boolean b ideal m0 list minf of integers/infinite primes matrix rels relation matrix over \mathbb{Z} </p>
SEE ALSO	EltCon , EltRayResidueRingRep , RayResidueRing , RayClassGroup ,
EXAMPLE	

```

kash> O := OrderMaximal(Order(x^3+9*x^2-8*x-9));;
kash> m0 := 94*O;;
kash> minf := [1,2];;
kash> RayConductorTest(m0,minf);
false
kash> L := RayConductor(m0,minf);
[ <47>, [ ] ]
kash> OrderClassGroup(0,500,euler,fast);
[ 1, [ 1 ] ]
kash> RayClassGroup(m0,minf);
[ 23, [ 23 ] ]
kash> RayClassGroup(L[1],L[2]);
[ 23, [ 23 ] ]

```


NAME	RayDiscSig
PURPOSE	Returns the relative discriminant and the absolute signature of the ray class field belonging to a congruence module.
SYNTAX	<pre>L := RayDiscSig(m0 [,minf] [,rels]);</pre> <p> <code>ideal</code> <code>m0</code> <code>list</code> <code>minf</code> of integer/infinite primes <code>matrix</code> <code>rels</code> relation matrix over \mathbb{Z} <code>list</code> <code>L</code> </p>
DESCRIPTION	<p>A list which contains the relative discriminant and the signature in the form $[r1,r2]$ is returned.</p> <p>If the units of the maximal order or the multiplicative group of the residue ring modulo the ideal or the class group of the order are not known, they will be calculated. The algorithm is based on the formulas for the discriminant and the signature in [CDO96, CDO97].</p>

SEE ALSO [EltCon](#), [OrderDisc](#), [RayConductor](#), [RayClassGroup](#), [RayClassField](#),

EXAMPLE

```
kash> O := OrderMaximal(Order(x^3+9*x^2-8*x-9));
Generating polynomial: x^3 + 9*x^2 - 8*x - 9
Discriminant: 42953

kash> m0 := 3*O;
<3>
kash> minf:= [1,2];
[ 1, 2 ]
kash> RayDiscSig(m0,minf);
[ <
  [27  0  9]
  [ 0 27  0]
  [ 0  0  9]
  >
  , [ 8, 2 ] ]
```

NAME	RayResidueRing
PURPOSE	This function computes the multiplicative group of the residue class ring.
SYNTAX	$L := \text{RayResidueRing}(m_0 \text{ [,minf]});$ <p> list L ideal m0 list minf of integers/infinite primes </p>
DESCRIPTION	<p>This function computes the multiplicative group of the residue class ring of an ideal or a congruence module $\mathfrak{m} = \mathfrak{m}_0 \mathfrak{m}_\infty$, where \mathfrak{m}_0 is an integral ideal of a maximal order \mathcal{O} and \mathfrak{m}_∞ is a formal product of primes at infinity. \mathfrak{m}_∞ is represented by a list of integers, where each entry represents a real embedding (as given by <code>EltCon</code>). <code>RayResidueRing</code> returns a list of the order of the multiplicative group and a list of the orders of the cyclic factors.</p> <p>This structure can be used to solve multiplicative congruences which may involve infinite places. This algorithm bases on the presentation of the Einseinheiten in [Has63] and is described in [Pau96].</p>
SEE ALSO	<code>EltCon</code> , <code>EltRayResidueRingRep</code> , <code>RayResidueRingRepToElt</code> , <code>RayResidueRingCyclicFactors</code> ,
EXAMPLE	

```

kash> O := OrderMaximal(Order(Z,2,10));
Generating polynomial: x^2 - 10
Discriminant: 40

kash> P3 := Factor(3*O)[1][1];
kash> IdealDegree(P3);
1
kash> IdealRamIndex(P3);
1
kash> m0 := P3^2;; L := RayResidueRing(m0);
[ 6, [ 2, 3 ] ]
kash> m0 := P3^4;; L := RayResidueRing(m0);
[ 54, [ 2, 27 ] ]
kash> P5 := Factor(5*O)[1][1];
kash> IdealDegree(P5);
1
kash> IdealRamIndex(P5);

```

```

2
kash> m0 := P5^2;; L := RayResidueRing(m0);
[ 20, [ 4, 5 ] ]
kash> m0 := P5^4;; L := RayResidueRing(m0);
[ 500, [ 4, 25, 5 ] ]
kash> P7 := 7*0;;
kash> IdealDegree(P7);
2
kash> IdealRamIndex(P7);
1
kash> m0 := P7^2;; RayResidueRing(m0);
[ 2352, [ 48, 7, 7 ] ]
kash> m0 := P7^4;; RayResidueRing(m0);
[ 5647152, [ 48, 343, 343 ] ]
kash> m0 := P5^4*P7^2;; L := RayResidueRing(m0);
[ 1176000, [ 4, 25, 5, 48, 7, 7 ] ]
kash> minf := [2];; L := RayResidueRing(1*0,minf);
[ 2, [ 1, 2 ] ]
kash> L := RayResidueRing(m0,minf);
[ 2352000, [ 4, 25, 5, 48, 7, 7, 2 ] ]

```

compute the ray residue ring of an ideal in \mathbb{Z}

```

kash> a := ZIdealCreate(9);
<9>
kash> RayResidueRing(a);
[ 6, [ 6 ] ]

```

NAME	RayResidueRingCyclicFactors
PURPOSE	Returns generators (and their orders) for the multiplicative group of the residue class ring of an ideal or a congruence module, as computed by RayResidueRing.
SYNTAX	<pre>L := RayResidueRingCyclicFactors(m0 [,minf]);</pre> <p>list L</p> <p>ideal m0</p> <p>list minf of integers/infinite primes</p>
DESCRIPTION	A list containing generators and their orders is computed. This structure can be used to solve multiplicative congruences which may involve infinite places.
SEE ALSO	EltCon , IdealRayClassRep , OrderClassGroup , RayClassGroup ,
EXAMPLE	

```

kash> O := OrderMaximal(Order(Z,2,10));;
kash> m0 := 7^3*0;; minf := [1,2];;
kash> L := RayResidueRingCyclicFactors(m0,minf);
[ [ [243657, 43632], 48 ], [ 15, 49 ], [ [48021, 357], 49 ], [ [1, -343], 2 ],
  [ [1, 343], 2 ] ]
kash> EltRayResidueRingRep(L[1][1],m0,minf);
[1 0 0 0 0]
kash> EltCon(L[1][1]);
[381633.49886846672701377572313068837506325162983903 105680.501131533272986224\
27686931162493674837016097]
kash> EltRayResidueRingRep(L[2][1]^2*L[3][1],m0,minf);
[0 2 1 0 0]
kash> EltRayResidueRingRep(L[4][1],m0,minf);
[0 0 0 1 0]
kash> L[4][1] mod m0;
1
kash> EltRayResidueRingRep(L[2][1]^2*L[3][1]*L[4][1],m0,minf);
[0 2 1 1 0]

```

NAME	RayResidueRingRepToElt
PURPOSE	Returns a canonical representative of an element of the multiplicative group of the residue class ring of a congruence module.
SYNTAX	<pre>b := RayResidueRingRepToElt(r, m0, minf);</pre> <p>algebraic element b</p> <p>matrix r</p> <p>ideal m0</p> <p>list minf of integers/infinite primes</p>
DESCRIPTION	<p>This is the inverse to <code>EltRayResidueRingRep</code>. The function will compute the following product:</p> $\prod i = 1^l a i^r ^i$ <p>with $a i$ the multiplicative basis for $(\mathfrak{o}/\mathfrak{m}_0)^*$ as returned by <code>RayResidueRingCyclicFactors</code>.</p>
SEE ALSO	RayResidueRing , RayResidueRingCyclicFactors , EltRayResidueRingRep ,

EXAMPLE

```
kash> O := OrderMaximal(Order(Poly(Zx,[1,3,-4,7])));;
kash> m0 := 15*O;;
kash> a := Elt(0,[7,1,8]);;
kash> r := EltRayResidueRingRep(a,m0);
[20 65]
kash> b := RayResidueRingRepToElt(r,m0);
[7, 1, -7]
```

NAME	RayResidueRingToAbelianGroup
PURPOSE	Returns the ray residue ring modulo a congruence module.
SYNTAX	<pre>g := RayResidueRingToAbelianGroup(m0 [, minf]);</pre> <div> <div>group</div> <div>ideal</div> <div>list of integers</div> </div> <div> <div>g</div> <div>m0</div> <div>minf infinite primes</div> </div>
DESCRIPTION	Returns the multiplicative group g of the ray residue ring modulo a congruence module. For more information see RayResidueRing .
SEE ALSO	RayResidueRing ,
EXAMPLE	

```
kash> o := OrderMaximal(Poly(Zx, [1,6,2]));
```

```
Generating polynomial: x^2 + 6*x + 2
```

```
Discriminant: 28
```

```
kash> m0 := Ideal(Elt(o, [0,3]));
```

```
<[0, 3]>
```

```
kash> g := RayResidueRingToAbelianGroup(m0, [1,2]);
```

```
Group with relations:
```

```
[2 0 0 0]
```

```
[0 2 0 0]
```

```
[0 0 2 0]
```

```
[0 0 0 2]
```

NAME	Re
PURPOSE	Returns the real part of a complex number.
SYNTAX	$a := \text{Re}(z);$ real a complex z
SEE ALSO	Im ,

EXAMPLE Compute the real part of $1 + 2i$:

```
kash> z := Comp(1, 2);  
1 + 2*i  
kash> Re(z);  
1
```

Read

NAME	Read
PURPOSE	Read a file containing KASH commands.
SYNTAX	<code>Read(name);</code> <code>string name</code>
DESCRIPTION	The file "name" must be both existing and readable. KASH looks first in the given path, then in the current directory and finally in LIBNAME../src.

EXAMPLE Assume that we have a file named "in" in our current working directory. First, we take a look at its contents:

```
kash> Exec("cat in");
Time(true);
x := Poly(Zx, [1,0]);
Palg := function(0)
local px;
px := PolyAlg(0);
x := Poly(px, [1,0]);
return px;
end;
```

```
kash> Read("in");
kash> x;
x
Time: 0 ms
```


NAME	ReadLib
PURPOSE	The same as Read, but here the file should be in the KASH lib directory and it must have the extension “.g”.
SYNTAX	<code>ReadLib(name);</code> <code>string name</code>
SEE ALSO	Read ,

EXAMPLE We will read the neq.g library. This will define a function solving norm equations by suitable principal ideal tests.

```
kash> ReadLib("neq");
kash> Neq(o, 7);
Prime 7: 1 [ 2 ]
No solution
false
Time: 40 ms
kash> Neq(o, 11);
Prime 11: 1 [ 1, 1 ]
[ [ [ [ <11, [5, 2]>, 1 ], [ <11, [8, 2]>, 1 ] ], [ 0, 1 ], [ 1, 0 ] ] ]
at most 2 different solutions!
Ideals : [ [ <11, [5, 2]>, 1 ], [ <11, [8, 2]>, 1 ] ]
Ideal Basis of all (possible) solutions: [ [ <11, [8, 2]>, <11, [5, 2]> ] ]
[ [1, 3], [-2, -3] ]
Time: 260 ms
kash> OrderNormEquation(o, 11, -1);
[ [-2, -3], [-1, -3] ]
Time: 180 ms
```

Round

NAME Round

PURPOSE Returns the integer closest to a given number. Note that if the decimal part of the number given is .5, then the number is rounded up. If the number is negative, it is rounded down.

SYNTAX `y := Round(x);`

 integer y
 real x

SEE ALSO [Trunc](#), [Floor](#), [Ceil](#),

EXAMPLE

```
kash> Round(1.4);  
1  
kash> Round(-1.4);  
-1  
kash> Round(1.5);  
2  
kash> Round(-1.5);  
-2
```

NAME	SPrint				
PURPOSE	Creates a string instead of printing on the screen.				
SYNTAX	$s := \text{SPrint}(\text{obj1}, \text{obj2}, \dots);$ <table><tr><td>string</td><td>s</td></tr><tr><td>string or kash object</td><td>obj1, obj2</td></tr></table>	string	s	string or kash object	obj1, obj2
string	s				
string or kash object	obj1, obj2				
DESCRIPTION					
SEE ALSO	Print , SScan ,				
EXAMPLE					

EXAMPLE converting a polynomial into a string

```
kash> f := Poly(Zx, [1,0,2]);  
x^2 + 2  
kash> s := SPrint("f equals ",f);;  
kash> Print(s,"\n");  
f equals x^2 + 2
```

convert an algebraic element into a string

```
kash> o := OrderMaximal(f);;  
kash> alpha := Elt(o, [0,1]);  
[0, 1]  
kash> s := SPrint("alpha equals ",alpha);;  
kash> Print(s,"\n");  
alpha equals [0, 1]
```

SScan

NAME **SScan**

PURPOSE Reads kash objects out of a string.

SYNTAX **L := SScan(s, fmt, R1, R2, ...);**

list L containing the kash objects
string s
string fmt format string
rings R1, R2

DESCRIPTION Supported kash objects are: integers (%Z), reals (%R), logicals (%B), polynomials (%P) and algebraic elements (%E); also the usual standard C-format is supported. If you want to scan an algebraic element resp. a polynomial you have to pass the order resp. the polynomial algebra as parameter to the **SScan** function. For a better understanding see the detailed list of examples below.

SEE ALSO **SPrint**, **SScan**,

EXAMPLE

EXAMPLE reading a list of integers

```
kash> L := SScan("111, 12, 13", "%Z,%Z,%Z");  
[ 111, 12, 13 ]
```

reading an algebraic element

```
kash> o := OrderMaximal(Z,2,5);;  
kash> L := SScan("alpha: [0,27]/12 ", "alpha: %E", o);  
[ [0, 9] / 4 ]
```

or

```
kash> L := SScan("alpha: [0,27]/12 ", "%s %E", o);  
[ "alpha:", [0, 9] / 4 ]
```

reading a logical and a polynomial and a real

```
kash> s := "true, 17*x^3 + 14*x +2, 2.1382";;  
kash> fmt := "%B,%P,%R";;  
kash> L := SScan(s, fmt, Zx);  
[ true, 17*x^3 + 14*x + 2, 2.1382 ]
```

and now an interesting example

```
kash> s := "foobar : foobar 17*x^3 + 14*x +2, 2.1382";;  
kash> fmt := "%s : %s %P,%R";;  
kash> L := SScan(s, fmt, Zx);  
[ "foobar", "foobar", 17*x^3 + 14*x + 2, 2.1382 ]
```

```
list of coordinates of one integral point L
simplex s
```

NAME	SimplexInit
PURPOSE	Initializes the enumeration of all integral points contained in a bounded simplex.
SYNTAX	<pre>s := SimplexInit(A, b [, delta]);</pre> <p> Simplex s real matrix A $A \in \mathbb{R}^{m \times n}$ real matrix b $b \in \mathbb{R}^m$ real delta should be $1 + \epsilon$ </p>
DESCRIPTION	<p>Initializes the enumeration of all integral points contained in an bounded simplex. This function performs Fourier-Motzkin elimination to produce a triangular system. Be careful: The size of the system may be exponential in the number of initial constraints.</p> <p>A simplex s is defined as $\mathbf{s} := \{x \in \mathbb{R}^n \mid \mathbf{Ax} \leq \mathbf{b}\}$ where \leq means \leq componentwise. If delta is given, it is used as a correction for round-off errors. You may get points outside of s.</p>
SEE ALSO	SimplexNext , SimplexElt , SimplexReInit ,
EXAMPLE	Find all x, y in $\mathbb{Z}^2 \geq 0$ subject to $x + y \leq 3$:

```

kash> A := Mat(R, [[1,1],[-1, 0],[0,-1]]);
[ 1  1]
[-1  0]
[ 0 -1]
kash> b := MatTrans(Mat(R, [[3, 0, 0]]));
[3]
[0]
[0]
kash> s := SimplexInit(A, b);;
kash> while SimplexNext(s) do Print(SimplexElt(s), "\n"); od;
[ 0, 0 ]
[ 1, 0 ]
[ 2, 0 ]
[ 3, 0 ]
[ 0, 1 ]
[ 1, 1 ]
[ 2, 1 ]
[ 0, 2 ]
[ 1, 2 ]
[ 0, 3 ]

```

NAME	SimplexNext
PURPOSE	Tries to find a “next” integral point in the given simplex.
SYNTAX	<pre>ok := SimplexNext(s); boolean ok simplex s</pre>
DESCRIPTION	Tries to find a “next” integral point in the given simplex. If there is no next point, it will return false; otherwise true.
SEE ALSO	SimplexInit , SimplexElt , SimplexReInit ,

NAME	SimplexReInit
PURPOSE	Reinitializes a given simplex. May be used to set <code>delta</code> .
SYNTAX	<pre>SimplexReInit(s [, delta]);</pre> <pre>simplex s real delta</pre>
DESCRIPTION	Restarts the enumeration of the integral points of the given simplex. In addition you may give a (<code>delta</code>) that is used as a scaling factor on the bounds of the enumeration environment. It is intended to be used to correct round-off errors for border points.
SEE ALSO	SimplexNext , SimplexElt , SimplexInit ,

EXAMPLE Find all x, y in $\mathbb{Z}^2 | \geq 0$ subject to $x + y \leq 2$:

```
kash> A := Mat(R, [[1,1],[-1, 0],[0,-1]]);;
kash> b := MatTrans(Mat(R, [[2, 0, 0]]));;
kash> s := SimplexInit(A, b);;
kash> while SimplexNext(s) do Print(SimplexElt(s), "\n"); od;
[ 0, 0 ]
[ 1, 0 ]
[ 2, 0 ]
[ 0, 1 ]
[ 1, 1 ]
[ 0, 2 ]
```

Sin

NAME Sin

PURPOSE Returns the sine of a number.

SYNTAX `y := Sin(x);`

 complex y

 complex x

DESCRIPTION Given an `x` (in radians) the function returns the sine of `x`. The computation is done in the current precision of the real (complex) field.

SEE ALSO [Cos](#), [Tan](#),

EXAMPLE

```
kash> Sin(.5);
0.4794255386042030002732879352155713880818033679406006
kash> i := Comp(0, 1);
1*i
kash> Sin(i);
1.175201193643801456882381850595600815155717981334*i
```

NAME Sleep

PURPOSE Lets kash sleep for nSec seconds. Useful for KashPvm

SYNTAX Sleep(nSec);

 small integer nSec

EXAMPLE

```
kash> Date();
22.2.2004 16:17:06
kash> Sleep(5);
0
kash> Date();
22.2.2004 16:17:11
```

Solve

NAME Solve

PURPOSE Solves an equation or computes the roots of a polynomial.

SYNTAX L := Solve(t, A);
 L := Solve(o, a);
 L := Solve(f);

list	L
Thue object	t
int	A
order	o
int algebraic element	a
polynomial	f

DESCRIPTION At the moment two kinds of equations can be solved using **Solve**. Furthermore it is possible to compute the roots of a polynomial.

Solve(t,A) Solves a Thue equation. The command **Solve(t,A)** is tantamount to the calling sequence **ThueSolve(t,A)**.

Solve(o,a) Solves a norm equation. The command **Solve(o,a)** is tantamount to the calling sequence **OrderNormEquation(o,a)**.

Solve(f) Computes the roots of a polynomial.

SEE ALSO **OrderNormEquation**, **ThueSolve**, **PolyZeros**,

EXAMPLE Compute all $x, y \in \mathbb{Z}$ with $x^3 + x^2y - 6xy^2 + 2y^3 = 2$:

```
kash> t := Thue( [1, 1, -6, 2] );  
X^3 + X^2 Y - 6 X Y^2 + 2 Y^3  
kash> Solve(t, 2);  
[ [ -724, -411 ], [ -4, -11 ], [ -3, 1 ], [ -1, -1 ], [ 0, 1 ], [ 2, 1 ] ]
```

NAME	Sqrt
PURPOSE	Returns a square root of a number.
SYNTAX	$y := \text{Sqrt}(x);$ <div style="display: flex; justify-content: space-between;"> complex or real y </div> <div style="display: flex; justify-content: space-between;"> complex or real or rational or integer x </div>
DESCRIPTION	<p>Given x in \mathbb{Z}, \mathbb{Q} or \mathbb{R} the function returns \sqrt{x} if $x > 0$ or $i\sqrt{ x }$, otherwise. For $x = x e^{i\phi} \in \mathbb{C}$, $\phi \in (-\pi, \pi]$, the function returns $\sqrt{ x }e^{i\frac{\phi}{2}}$. The computation is done in the current precision of the real (complex) field.</p>

EXAMPLE

```

kash> Sqrt(2);
1.414213562373095048801688724209698078569671875377
kash> Sqrt(-2);
1.414213562373095048801688724209698078569671875377*i
kash> i := Comp(0, 1);
1*i
kash> Sqrt(i);
0.7071067811865475244008443621048490392848359376883159 + 0.7071067811865475244\
008443621048490392848359376886321*i
kash> Sqrt(-i);
0.7071067811865475244008443621048490392848359376883159 - 0.7071067811865475244\
008443621048490392848359376886321*i

```

NAME	SubfieldAdd
PURPOSE	Adds a subfield to the given order.
SYNTAX	<p>SubfieldAdd(o, sub, alpha);</p> <p> order o order sub subfield algebraic element alpha primitive element of sub in o </p>
DESCRIPTION	This function adds a subfield to the given order. It checks if this subfield was known. In this case, true is returned. The subfield must be given as an equation order.
SEE ALSO	OrderSubfield , OrderSubfieldSub , SubfieldGet ,
EXAMPLE	<pre> kash> o:=Order(Z,2,2); Generating polynomial: x^2 - 2 kash> O:=Order(o,2,3); F[1] / / E1[1] / / Q F [1] x^2 - 3 E 1[1] x^2 - 2 kash> Oa:=OrderAbs(O); Generating polynomial: x^4 - 10*x^2 + 1 kash> SubfieldAdd(Oa,o,EltMove(OrderBasis(o)[2],Oa)); true </pre>

NAME	SubfieldGet
PURPOSE	Returns all computed subfields.
SYNTAX	$L := \text{SubfieldGet}(o);$ <div> <div>list of subfields</div> <div>L</div> </div> <div> <div>order</div> <div>o</div> </div>
DESCRIPTION	This function returns all computed subfields. Use <code>OrderSubfield</code> if you want to get all subfields.
SEE ALSO	<code>OrderSubfield</code> , <code>OrderSubfieldSub</code> , <code>SubfieldAdd</code> ,
EXAMPLE	

```
kash> o:=OrderMaximal(Order(Poly(Zx,[1,0,-4,0,1])));
Generating polynomial: x^4 - 4*x^2 + 1
Discriminant: 2304
```

```
kash> OrderSubfield(o);;
kash> L:=SubfieldGet(o);
[ Generating polynomial: x^2 - 2
  , Generating polynomial: x^2 + 4*x + 1
  , Generating polynomial: x^2 - 4*x - 2
]
```

NAME	SubfieldSetDegreeMax
PURPOSE	Sets the maximal number of subfield of degree d .
SYNTAX	$\text{SubfieldSetDegreeMax}(0, d)$ $\text{order} \quad 0$ $\text{integer} \quad d$
DESCRIPTION	This function sets the maximal number of degree d of a given field to the number of known subfields of degree d . This function is very useful if one knows from theory that there are only a specified number of subfields which can be computed by a special algorithm. One has to be very carefull in using this function. If the number of computed subfields of degree d is smaller than the number of subfields of degree d , the Subfield command has no chance to find the other ones.

NAME	Tan
PURPOSE	Returns the tangent of a number.
SYNTAX	<pre>y := Tan(x);</pre> <pre>complex y</pre> <pre>complex x</pre>
DESCRIPTION	Given an $x \in \mathbb{C} \setminus \{k\pi + \pi/2 : k \in \mathbb{Z}\}$ (in radians) the function returns the tangent of x . The computation is done in the current precision of the real (complex) field.
SEE ALSO	Sin , Cos ,
EXAMPLE	

```
kash> Tan(1);
1.557407724654902230506974807458360173087250772381
kash> i := Comp(0, 1);
1*i
kash> Tan(i);
0.761594155955764888119458282604793590412768597257912*i
```

NAME	Thue
PURPOSE	Creates a thue object for solving Thue equations.
SYNTAX	<pre> t := Thue(o); t := Thue(L); t := Thue(f); Thue object t order o list L polynomial f </pre>
DESCRIPTION	<p>One classical object of number theory is the Diophantine equation of Thue</p> $f(X, Y) = a,$ <p>where $f(X, Y) \in \mathbb{Z}[X, Y]$ is a homogeneous polynomial of degree $n \leq 3$ and a is an integer.</p> <p>Before solving a Thue equation in KASH the form f on the left side has to be created by invoking the Thue function.</p> <p>Thue(o) takes the coefficients of the form from the polynomial defining the order \mathbf{o}.</p> <p>Thue(L) takes the coefficients from the list \mathbf{L}.</p> <p>Thue(f) takes the coefficients from the polynomial \mathbf{f}.</p>
SEE ALSO	ThueEval , ThueSolve ,

EXAMPLE Create the Thue object corresponding to the form

$$f(X, Y) = X^3 + X^2Y - 6XY^2 + 2Y^3.$$

```

kash> o := Order(Poly(Zx,[1,1,-6,2]));
Generating polynomial: x^3 + x^2 - 6*x + 2

```

```

kash> t := Thue(o);
X^3 + X^2 Y - 6 X Y^2 + 2 Y^3
kash> L := [1,1,-6,2];
[ 1, 1, -6, 2 ]
kash> t := Thue(L);

```

```
X^3 + X^2 Y - 6 X Y^2 + 2 Y^3
kash> f := Poly(Zx,[1,1,-6,2]);
x^3 + x^2 - 6*x + 2
kash> t := Thue(f);
X^3 + X^2 Y - 6 X Y^2 + 2 Y^3
```

NAME	ThueEval
PURPOSE	missing shortdoc
SYNTAX	<pre>a := ThueEval(t,x,y); int a Thue object t int x int y</pre>
DESCRIPTION	Let $f(X,Y) \in \mathbb{Z}[X,Y]$ be the homogeneous polynomial of the Thue object t generated by the KASH function Thue. The ThueEval function evaluates f at (x,y) .
SEE ALSO	Thue , ThueSolve ,
EXAMPLE	With $f(X,Y) = X^3 + X^2Y - 6XY^2 + 2Y^3$ compute $f(1,1)$, $f(3,1)$ and $f(-724,-411)$.

```
kash> t := Thue([1,1,-6,2]);
X^3 + X^2 Y - 6 X Y^2 + 2 Y^3
kash> ThueEval(t,1,1);
-2
kash> ThueEval(t,3,1);
20
kash> ThueEval(t,-724,-411);
2
```

NAME	ThueSolve
PURPOSE	Solves a Thue equation.
SYNTAX	$L := \text{ThueSolve}(t, a \text{ [, "exact" "abs"]});$ <div style="margin-left: 100px;"> $\text{list} \quad L$ $\text{Thue object} \quad t$ $\text{int} \quad a$ </div>
DESCRIPTION	<p>Let $f(X, Y) \in \mathbb{Z}[X, Y]$ be the homogeneous polynomial of the Thue object t generated by the KASH function <code>Thue</code>. The <code>ThueSolve</code> function determines all $x, y \in \mathbb{Z}$ such that $f(x, y) = a$.</p> <p>If the third argument equals "abs" the <code>ThueSolve</code> function will compute all $x, y \in \mathbb{Z}$ with $f(x, y) = a$. The calling sequence <code>ThueSolve(t, a, "exact")</code> is tantamount to <code>ThueSolve(t, a)</code>.</p> <p>The implementation of the <code>ThueSolve</code> function in KASH bases on the algorithm of Y. Bilu and G. Hanrot [BH96].</p>
SEE ALSO	<code>Solve</code> , <code>Thue</code> , <code>ThueEval</code> ,

EXAMPLE Compute all $x, y \in \mathbb{Z}$ with $x^3 + x^2y - 6xy^2 + 2y^3 = 2$.

```
kash> t := Thue([1,1,-6,2]);
X^3 + X^2 Y - 6 X Y^2 + 2 Y^3
kash> ThueSolve(t,2);
[ [ -724, -411 ], [ -4, -11 ], [ -3, 1 ], [ -1, -1 ], [ 0, 1 ], [ 2, 1 ] ]
```

Time

NAME	Time
PURPOSE	Toggles the time display.
SYNTAX	$b := \text{Time}([true false]);$ $\text{boolean } b \quad \text{current status of time display}$
DESCRIPTION	Activates or deactivates the time display. Default is false . If no argument is given the mode will be toggled. The CPU-time used is displayed after a result of a computation is printed. If there is no printed result, the time will not be printed even if the time display is on.

EXAMPLE Toggling of time display:

```
kash> o := Order(Z,2,110);
Generating polynomial: x^2 - 110
```

```
kash> Time(true);
true
Time: 0 ms
kash> 0 := OrderMaximal(o);
Generating polynomial: x^2 - 110
Discriminant: 440
```

```
Time: 0 ms
kash> OrderUnitsFund(0);
[ [21, 2] ]
Time: 0 ms
kash> Time();
false
kash> OrderClassGroup(0);
[ 2, [ 2 ] ]
```

NAME	Trace						
PURPOSE	Computes the trace of an algebraic number or alff element or a matrix						
SYNTAX	<pre>n := Trace(a); n := Trace(b, o);</pre> <table> <tr> <td>algebraic element alff elt matrix ff element</td><td>a</td></tr> <tr> <td>algebraic element ff element</td><td>b</td></tr> <tr> <td>order</td><td>o</td></tr> </table>	algebraic element alff elt matrix ff element	a	algebraic element ff element	b	order	o
algebraic element alff elt matrix ff element	a						
algebraic element ff element	b						
order	o						
DESCRIPTION	Depending on the type of a the trace of an algebraic element, alff element or matrix is computed.						

EXAMPLE All examples take place in $\mathbb{Q}(\sqrt{5})$:

```
kash> o := Order(Z, 2, 5);
Generating polynomial: x^2 - 5
```

```
kash> Trace(Elt(o, [1, 2]));
```

```
2
```

TrialDivision

NAME TrialDivision

PURPOSE

SYNTAX F := TrialDivision(d, b);

list F
integer d
integer b

DESCRIPTION Finds all factors up to b. Returns the rest.

EXAMPLE

```
kash> TrialDivision(8274626472648264826427648723648276,10^6);  
[ [ [ 2, 2 ], [ 11, 1 ], [ 41, 1 ], [ 86423, 1 ],  
  [ 53074086409412759917474753, 1 ] ], 1 ]
```


NAME	Trunc
PURPOSE	Returns the integer part of a number.
SYNTAX	<pre>y := Trunc(x);</pre> $\begin{array}{ll} \text{integer} & y \\ \text{real} & x \end{array}$
DESCRIPTION	<p>Given an x in \mathbb{Z}, \mathbb{Q} or \mathbb{R} the function returns</p> $\begin{cases} \max\{k \in \mathbb{Z} : k \leq x\} & : x \geq 0 \\ \min\{k \in \mathbb{Z} : k \geq x\} & : x < 0 \end{cases}.$ <p>The computation is done in the current precision of the real (complex) field.</p>
SEE ALSO	Round , Floor , Ceil ,
EXAMPLE	<pre>kash> Trunc(1); 1 kash> Trunc(-1); -1 kash> Trunc(1.1); 1 kash> Trunc(-1.1); -1 kash> Trunc(1.6); 1 kash> Trunc(-1.6); -1</pre>

Valuation

NAME	Valuation								
PURPOSE	missing shortdoc								
SYNTAX	<p><code>d := Valuation([P,] a);</code> SHOTDOC Computes the valuation of the argument at a prime (if possible).</p> <table><tr><td><code>integer</code></td><td><code>d</code></td></tr><tr><td><code>ideal integer alff order ideal</code></td><td><code>P</code> must be prime.</td></tr><tr><td><code>ideal algebraic element integer</code></td><td><code>a</code> to valuate.</td></tr><tr><td><code>alff order ideal</code></td><td><code>a</code> to valuate.</td></tr></table>	<code>integer</code>	<code>d</code>	<code>ideal integer alff order ideal</code>	<code>P</code> must be prime.	<code>ideal algebraic element integer</code>	<code>a</code> to valuate.	<code>alff order ideal</code>	<code>a</code> to valuate.
<code>integer</code>	<code>d</code>								
<code>ideal integer alff order ideal</code>	<code>P</code> must be prime.								
<code>ideal algebraic element integer</code>	<code>a</code> to valuate.								
<code>alff order ideal</code>	<code>a</code> to valuate.								
DESCRIPTION	<p>Returns the valuation of a p-adic number, an integer, an algebraic element, an (fractional) ideal at a prime ideal, a rational prime or an alff order ideal at a prime ideal. If the first argument is an integer then the second argument must also be an integer. For alff order ideals they have to be defined in the same order which must be maximal.</p> <p>This function calls <code>EltValuation</code>, <code>IdealValuation</code>, <code>IntValuation</code>, or <code>AlffIdealValuation</code>.</p>								

EXAMPLE Computation of the valuation of an algebraic number.

```
kash> O := OrderMaximal(Order(Z, 2, 3));
Generating polynomial: x^2 - 3
Discriminant: 12
```

```
kash> L := Factor(2*O);
[ [ <2, [1, 1]>, 2 ] ]
kash> P := L[1][1];
<2, [1, 1]>
kash> Valuation(P, Elt(0, 2));
2
kash> Valuation(P, Elt(0, [1, 1]));
1
```

NAME	Vec
PURPOSE	Returns a vector (matrix with one column).
SYNTAX	<pre>v := Vec(r,L);</pre> <pre>ring r list L matrix M</pre>
DESCRIPTION	Returns a matrix with one column, used for VecDotProduct for example.
EXAMPLE	

```
kash> v := Vec(Z, [1,2,3]);
[1]
[2]
[3]
```

VecDotProduct

NAME VecDotProduct

PURPOSE Computes the dot product of two vectors.

SYNTAX `r := VecDotProduct (u,v);`

list r
matrix u
matrix v

EXAMPLE

```
kash> u := Mat(Z,[[1],[2],[3]]);  
[1]  
[2]  
[3]  
kash> v := Mat(Z,[[4],[5],[6]]);  
[4]  
[5]  
[6]  
kash> VecDotProduct(u,v);  
32  
kash> v := Comp(0,1)*v;  
[4*i]  
[5*i]  
[6*i]  
kash> VecDotProduct(u,v);  
-32*i
```

NAME	WeierstrassP
PURPOSE	Calculates the value of the Weierstrass \wp -function related to the given lattice.
SYNTAX	<pre>u := WeierstrassP(z, w1, w2);</pre> <p> <code>complex</code> <code>u</code> <code>complex</code> <code>z</code> a non-zero element of the complex torus $\mathbb{C}/\mathbb{Z}w_1 \oplus \mathbb{Z}w_2$ <code>complex</code> <code>w1,w2</code> complex values with $\text{Im}(w_1/w_2) > 0$ </p>
DESCRIPTION	<p>Let z, w_1 and w_2 be complex numbers with $\text{Im}(w_1/w_2) > 0$ and with $z \notin \mathbb{Z}w_1 \oplus \mathbb{Z}w_2$. The value of the Weierstrass \wp-function $\wp(z; \mathbb{Z}w_1 \oplus \mathbb{Z}w_2)$ is then a well-defined complex number. It is calculated by use of the infinite q-expansion of \wp described in [Lan87, Chapt. 4.2, Prop. 3], where the precision of the calculated value depends on that of the defining complex field.</p>

EXAMPLE Compute $\wp(\sqrt{-11}; (1 + \sqrt{-11})\mathbb{Z} \oplus 12\mathbb{Z})$:

```
kash> z := Sqrt(-11);
3.316624790355399849114932736670686683927088545589*i
kash> w1 := 1+Sqrt(-11);
1 + 3.316624790355399849114932736670686683927088545589*i
kash> w2 := Comp(12,0);
12
kash> WeierstrassP(z,w1,w2);
1.018219365756141747667075554911574076310206321785 + 0.03622971784813503226915\
102443845385206504756332*i
```

WeilPairing

NAME	WeilPairing
PURPOSE	Calculates the value of the Weil-Pairing on given places p,q
SYNTAX	<pre>a := WeilPairing(p,q,m);</pre> <p>place of degree one in a global function field of genus 1 p place of degree one in a global function field of genus 1 q integer m finite field element or false a</p>
DESCRIPTION	<p>Return $e_m(p,q)$ if p,q have m-torsion in the classgroup and m is coprime to the characteristic of the ground field OR return false if it is not possible to compute the pairing using the below algorithm (this may happen when the curve has very few elements over the ground field (e.g. 4)). $e_m(p,q)$ is of type FFElt.</p> <p>Raise an error if:</p> <ul style="list-style-type: none">• Either p or q have not m torsion• p,q have no common function field,• Either p or q are not of degree 1,• Either p or q are not of type alffPlace• m is not of type int or negative,• F is not global <p>Raise a warning if the infinite place of the underlying rational field splits in F (-> No canonical choice can be made for a neutral element of the point group).</p>

EXAMPLE

```
kash> prime := 5;
5
kash> d := 3;
3
kash> k:=FF(prime,d);
Finite field of size 5^3
kash> w:=FFPrimitiveElt(k);
w
kash> kT := PolyAlg(k, "T");
Univariate Polynomial Ring in T over GF(5^3)
```

```

kash> kTy := PolyAlg(kT, "y");
Univariate Polynomial Ring in y over Univariate Polynomial Ring in T over GF(5\
^3)

kash> L:=[1,2,3,7,4];
[ 1, 2, 3, 7, 4 ]
kash> f:= y^2 + L[1]*T*y + L[3]*y - (T^3 + L[2]*T^2 + L[4]*T + L[5]);
y^2 + (T + 3)*y + 4*T^3 + 3*T^2 + 3*T + 1
kash> AlffPolyIsIrrSep(f);
true
kash> F:=Alff(f);
Algebraic function field defined by
$.1^2 + $.1*$.2 + 3*$.1 + 4*$.2^3 + 3*$.2^2 + 3*$.2 + 1
over
Univariate rational function field over GF(5^3)
Variables: T

kash> AlffGenus(F);
1
kash> P := AlffPlaces(F,1);;
kash> p := P[1];
Alff place < [ 1/T, 0 ], [ 0, 1 ] >
kash> q := P[2];
Alff place < [ T, 0 ], [ 4, 1 ] >
kash> WeilPairing(p,q,18);
1

```

World

NAME World

PURPOSE Returns some information on objects.

SYNTAX World (arg1, arg2, ...);

 object 1 arg1

 ⋮ ⋮

 object n argn

DESCRIPTION

EXAMPLE Information on integers:

```
kash> i:=2^28;
268435456
kash> World(i);
Big integer: 268435456
kash> i:=2^28-1;
268435455
kash> World(i);
Small integer:
268435455
```


NAME	Z
PURPOSE	Predefined constant: Integer ring \mathbb{Z} .
SYNTAX	<code>Z;</code> <code>ring Z</code>
DESCRIPTION	This is a predefined constant for the integer ring \mathbb{Z} and is referenced by the variable Z. It is of arbitrary precision.
SEE ALSO	Q , R , C ,
EXAMPLE	

```
kash> Z;  
Integer Ring
```

ZIdealCreate

NAME ZIdealCreate

PURPOSE Returns the ideal in \mathbb{Z} generated by the given integer.

SYNTAX I := IdealRayClassRep(a);

 ideal in \mathbb{Z} I
 integer a

SEE ALSO [Ideal](#),

EXAMPLE

```
kash> I := ZIdealCreate(6);  
<6>
```

NAME	Zx
PURPOSE	Predefined constant: polynomial algebra over integer ring.
SYNTAX	Zx ;
SEE ALSO	PolyAlg , PolyAlgCoef ,
EXAMPLE	

```
kash> Zx;  
Univariate Polynomial Ring in x over Integer Ring
```

NAME e

PURPOSE Predefined constant: Euler's constant in the current precision of the real field.

SYNTAX e;

 real e

SEE ALSO [pi](#), [Prec](#), [R](#),

EXAMPLE

```
kash> e;
2.718281828459045235360287471352662497757247093699
kash> Prec(100);
100
kash> e;
2.7182818284590452353602874713526624977572470936999595749669676277240766303535\
47594571382178525166
```

NAME	mod
PURPOSE	The remainder modulo an ideal or an element (in case of principal ideal domains)
SYNTAX	<pre>a := b mod c;</pre> <div> <pre>ring element</pre> <pre>ring element</pre> <pre>ideal or (in the case of principal ideal domains) ring element</pre> </div> <div> <pre>a</pre> <pre>b</pre> <pre>c</pre> </div>
DESCRIPTION	The function computes the relative discriminant of a Kummer extension of prime degree using results of [Dab95]. The function will first check, whether or not F is such an extension. The Discriminant is an ideal in the coefficient ring of F.
SEE ALSO	EltIdealReduce ,

EXAMPLE Discriminant of $\mathbb{Q}(\sqrt{10}, \sqrt{5})$ over $\mathbb{Q}(\sqrt{10})$

```
kash> E := OrderMaximal (Z,2,10);;
kash> F := Order (E,2,5);
      F[1]
      /
      /
      E1[1]
      /
      /
      Q
F  [ 1]      x^2 - 5
E 1[ 1]      x^2 - 10

kash> OrderKextDisc (F);
<1, 1>
```

undefined function name

NAME undefined function name

PURPOSE missing shortdoc

SYNTAX

NAME undefined function name

PURPOSE missing shortdoc

SYNTAX

pi

NAME pi

PURPOSE Predefined constant: π in the current precision of the real field.

SYNTAX pi;

 real pi

SEE ALSO [e](#), [Prec](#), [R](#),

EXAMPLE

```
kash> pi;
3.141592653589793238462643383279502884197169399375
kash> Prec(100);
100
kash> pi;
3.1415926535897932384626433832795028841971693993751058209749445923078164062862\
08998628034825342117
```


Chapter 2

The Programming Language

This chapter describes the GAP programming language. It should allow you in principle to predict the result of each and every input. In order to know what we are talking about, we first have to look more closely at the process of interpretation and the various representations of data involved.

First we have the input to GAP, given as a string of characters. How those characters enter GAP is operating system dependent, e.g., they might be entered at a terminal, pasted with a mouse into a window, or read from a file. The mechanism does not matter. This representation of expressions by characters is called the **external representation** of the expression. Every expression has at least one external representation that can be entered to get exactly this expression.

The input, i.e., the external representation, is transformed in a process called **reading** to an internal representation. At this point the input is analyzed and inputs that are not legal external representations, according to the rules given below, are rejected as errors. Those rules are usually called the **syntax** of a programming language.

The internal representation created by reading is called either an **expression** or a **statement**. Later we will distinguish between those two terms, however now we will use them interchangeably. The exact form of the internal representation does not matter. It could be a string of characters equal to the external representation, in which case the reading would only need to check for errors. It could be a series of machine instructions for the processor on which GAP is running, in which case the reading would more appropriately be called compilation. It is in fact a tree-like structure.

After the input has been read it is again transformed in a process called **evaluation** or **execution**. Later we will distinguish between those two terms too, but for the moment we will use them interchangeably. The name hints at the nature of this process, it replaces an expression with the value of the expression. This works recursively, i.e., to evaluate an expression first the subexpressions are evaluated and then the value of the expression is computed according to rules given below from those values. Those rules are usually called the **semantics** of a programming language.

The result of the evaluation is, not surprisingly, called a **value**. The set of values is of course a much smaller set than the set of expressions; for every value there are several expressions that will evaluate to this value. Again the form in which such a value is represented internally does not matter. It is in fact a tree-like structure again.

The last process is called **printing**. It takes the value produced by the evaluation and creates an external representation, i.e., a string of characters again. What you do with this external representation is up to you. You can look at it, paste it with the mouse into another window, or write it to a file.

Lets look at an example to make this more clear. Suppose you type in the following string of 8 characters

```
1 + 2 * 3;
```

GAP takes this external representation and creates a tree like internal representation, which we can picture as follows

```

      +
     / \
    1   *
      / \
     2   3

```

This expression is then evaluated. To do this GAP first evaluates the right subexpression *'2*3'*. Again to do this GAP first evaluates its subexpressions 2 and 3. However they are so simple that they are their own value, we say that they are self-evaluating. After this has been done, the rule for *'*'* tells us that the value is the product of the values of the two subexpressions, which in this case is clearly 6. Combining this with the value of the left operand of the *'+'*, which is self-evaluating too gives us the value of the whole expression 7. This is then printed, i.e., converted into the external representation consisting of the single character *'7'*.

In this fashion we can predict the result of every input when we know the syntactic rules that govern the process of reading and the semantic rules that tell us for every expression how its value is computed in terms of the values of the subexpressions. The syntactic rules are given in sections "Lexical Structure", "Symbols", "Whitespaces", "Keywords", "Identifiers", and "The Syntax in BNF", the semantic rules are given in sections "Expressions", "Variables", "Function Calls", "Comparisons", "Operations", "Statements", "Assignments", "Procedure Calls", "If", "While", "Repeat", "For", "Functions", and the chapters describing the individual data types.

2.1 Lexical Structure

The input of GAP consists of sequences of the following characters.

Digits, uppercase and lowercase letters, <space>, <tab>, <newline>, and the special characters

"	'	()	*	+	,	-
.	/	:	;	<	=	>	~
[\]	^	_	{	}	&

Other characters will be signalled as illegal. Inside strings and comments the full character set supported by the computer is allowed.

2.2 Symbols

The process of reading, i.e., of assembling the input into expressions, has a subprocess, called **scanning**, that assembles the characters into symbols. A **symbol** is a sequence of characters that form a lexical unit. The set of symbols consists of keywords, identifiers, strings, integers, and operator and delimiter symbols.

A keyword is a reserved word consisting entirely of lowercase letters (see "Keywords"). An identifier is a sequence of letters and digits that contains at least one letter and is not a keyword (see "Identifiers"). An integer is a sequence of digits. A string is a sequence of arbitrary characters enclosed in double quotes.

Operator and delimiter symbols are

+	-	*	/	^	~
=	<>	<	<=	>	>=
:=	.	..	->	,	;
[]	{	}	()

Note that during the process of scanning also all whitespace is removed (see "Whitespaces").

2.3 Whitespaces

The characters <space>, <tab>, <newline>, and <return> are called **whitespace characters**. Whitespace is used as necessary to separate lexical symbols, such as integers, identifiers, or keywords. For example 'Thorondor' is a single identifier, while 'Th or ondor' is the keyword 'or' between the two identifiers 'Th' and 'ondor'. Whitespace may occur between any two symbols, but not within a symbol. Two or more adjacent whitespaces are equivalent to a single whitespace. Apart from the role as separator of symbols, whitespaces are otherwise insignificant. Whitespaces may also occur inside a string, where they are significant. Whitespaces should also be used freely for improved readability.

A **comment** starts with the character '\#', which is sometimes called sharp or hatch, and continues to the end of the line on which the comment character appears. The whole comment, including '\#' and the <newline> character is treated as a single whitespace. Inside a string, the comment character '\#' loses its role and is just an ordinary character.

For example, the following statement

```
if i<0 then a:=-i;else a:=i;fi;
```

is equivalent to

```
if i < 0 then      & if i is negative
    a := -i;      &   take its inverse
else              & otherwise
    a := i;       &   take itself
fi;
```

(which by the way shows that it is possible to write superfluous comments). However the first statement is not equivalent to

```
ifi<0thena:=-i;elsea:=i;fi;
```

since the keyword 'if' must be separated from the identifier 'i' by a whitespace, and similarly 'then' and 'a', and 'else' and 'a' must be separated.

2.4 Keywords

Keywords are reserved words that are used to denote special operations or are part of statements. They must not be used as identifiers. The keywords are

and	do	elif	else	end	fi
for	function	if	in	local	mod
not	od	or	repeat	return	then
until	while	quit			

Note that all keywords are written in lowercase. For example only 'else' is a keyword; 'Else', 'eLsE', 'ELSE' and so forth are ordinary identifiers. Keywords must not contain whitespace, for example 'el if' is not the same as 'elif'.

2.5 Identifiers

An identifier is used to refer to a variable (see "Variables"). An identifier consists of letters, digits, and underscores '_', and must contain at least one letter or underscore. An identifier is terminated by the first character not in this class. Examples of valid identifiers are

a	foo	aLongIdentifier
hello	Hello	HELLO
x100	100x	_100
some_people_prefer_underscores_to_separate_words		
WePreferMixedCaseToSeparateWords		

Note that case is significant, so the three identifiers in the second line are distinguished.

The backslash \ can be used to include other characters in identifiers; a backslash followed by a character is equivalent to the character, except that this escape sequence is considered to be an ordinary letter. For example G\ (2\,5\) is an identifier, not a call to a function 'G'.

An identifier that starts with a backslash is never a keyword, so for example * and \mod are identifier.

The length of identifiers is not limited, however only the first 1023 characters are significant. The escape sequence \<newline> is ignored, making it possible to split long identifiers over multiple lines.

2.6 Expressions

An **expression** is a construct that evaluates to a value. Syntactic constructs that are executed to produce a side effect and return no value are called **statements** (see "Statements"). Expressions appear as right hand sides of assignments (see "Assignments"), as actual arguments in function calls (see "Function Calls"), and in statements.

Note that an expression is not the same as a value. For example `'1 + 11'` is an expression, whose value is the integer 12. The external representation of this integer is the character sequence `'12'`, i.e., this sequence is output if the integer is printed. This sequence is another expression whose value is the integer 12. The process of finding the value of an expression is done by the interpreter and is called the **evaluation** of the expression.

Variables, function calls, and integer, permutation, string, function, list, and record literals (see "Variables", "Function Calls", "Functions",), are the simplest cases of expressions.

Expressions, for example the simple expressions mentioned above, can be combined with the operators to form more complex expressions. Of course those expressions can then be combined further with the operators to form even more complex expressions. The **operators** fall into three classes. The **comparisons** are `'='`, `'$<>$'`, `'$<=$'`, `'$>$'`, `'$>=$'`, and `'in'` (see "Comparisons" and "In"). The **arithmetic operators** are `'+'`, `'-'`, `'*'`, `'/'`, `'mod'`, and `'\^'` (see "Operations"). The **logical operators** are `'not'`, `'and'`, and `'or'`.

```
gap> 2 * 2;      # a very simple expression with value
4
gap> 2 * 2 + 9 = Fibonacci(7) and Fibonacci(13) in Primes;
true           # a more complex expression
```

2.7 Variables

A **variable** is a location in a GAP program that points to a value. We say the variable is **bound** to this value. If a variable is evaluated it evaluates to this value.

Initially an ordinary variable is not bound to any value. The variable can be bound to a value by **assigning** this value to the variable (see "Assignments"). Because of this we sometimes say that a variable that is not bound to any value has no assigned value. Assignment is in fact the only way by which a variable, which is not an argument of a function, can be bound to a value. After a variable has been bound to a value an assignment can also be used to bind the variable to another value.

A special class of variables are **arguments** of functions. They behave similarly to other variables, except they are bound to the value of the actual arguments upon a function call (see "Function Calls").

Each variable has a name that is also called its **identifier**. This is because in a given scope an identifier identifies a unique variable (see "Identifiers"). A **scope** is a lexical part of a program text. There is the global scope that encloses the entire program text, and there are local scopes that range from the `'function'` keyword, denoting the beginning of a function definition, to the corresponding

'end' keyword. A local scope introduces new variables, whose identifiers are given in the formal argument list and the 'local' declaration of the function (see "Functions"). Usage of an identifier in a program text refers to the variable in the innermost scope that has this identifier as its name. Because this mapping from identifiers to variables is done when the program is read, not when it is executed, GAP is said to have lexical scoping. The following example shows how one identifier refers to different variables at different points in the program text.

```

g := 0;           & global variable g
x := function ( a, b, c )
  local  y;
  g := c;         & c refers to argument c of function x
  y := function ( y )
    local  d, e, f;
    d := y;       & y refers to argument y of function y
    e := b;       & b refers to argument b of function x
    f := g;       & g refers to global variable g
    return d + e + f;
  end;
  return y( a ); & y refers to local y of function x
end;

```

It is important to note that the concept of a variable in GAP is quite different from the concept of a variable in programming languages like PASCAL. In those languages a variable denotes a block of memory. The value of the variable is stored in this block. So in those languages two variables can have the same value, but they can never have identical values, because they denote different blocks of memory. (Note that PASCAL has the concept of a reference argument. It seems as if such an argument and the variable used in the actual function call have the same value, since changing the argument's value also changes the value of the variable used in the actual function call. But this is not so; the reference argument is actually a pointer to the variable used in the actual function call, and it is the compiler that inserts enough magic to make the pointer invisible.) In order for this to work the compiler needs enough information to compute the amount of memory needed for each variable in a program, which is readily available in the declarations PASCAL requires for every variable.

In GAP on the other hand each variable just points to a value.

2.8 Function Calls

```

'<function-var>()'
'<function-var>( <arg-expr> \{',' <arg-expr>\} )'

```

The function call has the effect of calling the function <function-var>. The precise semantics are as follows.

First GAP evaluates the <function-var>. Usually <function-var> is a variable, and GAP does nothing more than taking the value of this variable. It is allowed though that <function-var> is a more

complex expression, namely it can for example be a selection of a list element '`<list-var>[<int-expr>]`', or a selection of a record component '`<record-var>.<ident>`'. In any case GAP tests whether the value is a function. If it is not, GAP signals an error.

Next GAP checks that the number of actual arguments `<arg-expr>`s agrees with the number of formal arguments as given in the function definition. If they do not agree GAP signals an error. An exception is the case when there is exactly one formal argument with the name '`arg`', in which case any number of actual arguments is allowed.

Now GAP allocates for each formal argument and for each formal local a new variable. Remember that a variable is a location in a GAP program that points to a value. Thus for each formal argument and for each formal local such a location is allocated.

Next the arguments `<arg-expr>`s are evaluated, and the values are assigned to the newly created variables corresponding to the formal arguments. Of course the first value is assigned to the new variable corresponding to the first formal argument, the second value is assigned to the new variable corresponding to the second formal argument, and so on. However, GAP does not make any guarantee about the order in which the arguments are evaluated. They might be evaluated left to right, right to left, or in any other order, but each argument is evaluated once. An exception again occurs if the function has only one formal argument with the name '`arg`'. In this case the values of all the actual arguments are stored in a list and this list is assigned to the new variable corresponding to the formal argument '`arg`'.

The new variables corresponding to the formal locals are initially not bound to any value. So trying to evaluate those variables before something has been assigned to them will signal an error.

Now the body of the function, which is a statement, is executed. If the identifier of one of the formal arguments or formal locals appears in the body of the function it refers to the new variable that was allocated for this formal argument or formal local, and evaluates to the value of this variable.

If during the execution of the body of the function a '`return`' statement with an expression (see "Return") is executed, execution of the body is terminated and the value of the function call is the value of the expression of the '`return`'. If during the execution of the body a '`return`' statement without an expression is executed, execution of the body is terminated and the function call does not produce a value, in which case we call this call a procedure call (see "Procedure Calls"). If the execution of the body completes without execution of a '`return`' statement, the function call again produces no value, and again we talk about a procedure call.

```
gap> Fibonacci( 11 );
      # a call to the function \verb|'Fibonacci'| with actual argument \verb|'11'|
89

gap> G.operations.RightCosets( G, Intersection( U, V ) );;
      # a call to the function in \verb|'G.operations.RightCosets'|
      # where the second actual argument is another function call
```

2.9 Comparisons

```
'<left-expr> = <right-expr>'
```

```
'<left-expr> <> <right-expr>'
```

The operator '=' tests for equality of its two operands and evaluates to 'true' if they are equal and to 'false' otherwise. Likewise '<>' tests for inequality of its two operands. Note that any two objects can be compared, i.e., '=' and '<>' will never signal an error. For each type of objects the definition of equality is given in the respective chapter. Objects of different types are never equal, i.e., '=' evaluates in this case to 'false', and '<>' evaluates to 'true'.

```
'<left-expr> < <right-expr>'
```

```
'<left-expr> > <right-expr>'
```

```
'<left-expr> <= <right-expr>'
```

```
'<left-expr> >= <right-expr>'
```

'<' denotes less than, '<=' less than or equal, '>' greater than, and '>=' greater than or equal of its two operands. For each type of objects the definition of the ordering is given in the respective chapter. The ordering of objects of different types is as follows. Rationals are smallest, next are cyclotomics, followed by finite field elements, permutations, words, words in solvable groups, boolean values, functions, lists, and records are largest.

Comparison operators, which includes the operator 'in' (see "In") are not associative, i.e., it is not allowed to write '<a> = <> <c> = <d>', you must use '(<a> =) <> (<c> = <d>)' instead. The comparison operators have higher precedence than the logical operators, but lower precedence than the arithmetic operators (see "Operations"). Thus, for example, '<a> = <c> and <d>' is interpreted, '(<a>) = <c>) and <d>'.

```
gap> 2 * 2 + 9 = Fibonacci(7);    # a comparison where the left
true                             # operand is an expression
```

2.10 Operations

```
'+ <right-expr>'
```

```
'- <right-expr>'
```

```
'<left-expr> + <right-expr>'
```

```
'<left-expr> - <right-expr>'
```

```
'<left-expr> *\ <right-expr>'
```

```
'<left-expr> / <right-expr>'
```

```
'<left-expr> mod <right-expr>'
```

```
'<left-expr> ^\ <right-expr>'
```

The arithmetic operators are '+', '-', '*', '/', 'mod', and '^'. The meanings (semantic) of those operators generally depend on the types of the operands involved, and they are defined in the various chapters describing the types. However basically the meanings are as follows.

'+' denotes the addition, and '-' the subtraction of ring and field elements. '*' is the multiplication of group elements, '/' is the multiplication of the left operand with the inverse of the right operand.

'mod' is only defined for integers and rationals and denotes the modulo operation. '+' and '-' can also be used as unary operations. The unary '+' is ignored and unary '-' is equivalent to multiplication by -1. '^' denotes powering of a group element if the right operand is an integer, and is also used to denote operation if the right operand is a group element.

The *precedence* of those operators is as follows. The powering operator '^' has the highest precedence, followed by the unary operators '+' and '-', which are followed by the multiplicative operators '*', '/', and 'mod', and the additive binary operators '+' and '-' have the lowest precedence. That means that the expression '-2 ^ -2 * 3 + 1' is interpreted as '(-(2 ^ (-2)) * 3) + 1'. If in doubt use parentheses to clarify your intention.

The *associativity* of the arithmetic operators is as follows. '^' is not associative, i.e., it is illegal to write '2^3^4', use parentheses to clarify whether you mean '(2^3) ^ 4' or '2 ^ (3^4)'. The unary operators '+' and '-' are right associative, because they are written to the left of their operands. '*', '/', 'mod', '+', and '-' are all left associative, i.e., '1-2-3' is interpreted as '(1-2)-3' not as '1-(2-3)'. Again, if in doubt use parentheses to clarify your intentions.

The arithmetic operators have higher precedence than the comparison operators (see "Comparisons" and "In") and the logical operators. Thus, for example, '<a> = <c> and <d>' is interpreted, '(<a>) = <c>) and <d>'.

```
gap> 2 * 2 + 9;    & a very simple arithmetic expression
13
```

2.11 Statements

Assignments (see "Assignments"), Procedure calls (see "Procedure Calls"), 'if' statements (see "If"), 'while' (see "While"), 'repeat' (see "Repeat") and 'for' loops (see "For"), and the 'return' statement (see "Return") are called statements. They can be entered interactively or be part of a function definition. Every statement must be terminated by a semicolon.

Statements, unlike expressions, have no value. They are executed only to produce an effect. For example an assignment has the effect of assigning a value to a variable, a 'for' loop has the effect of executing a statement sequence for all elements in a list and so on. We will talk about *evaluation* of expressions but about *execution* of statements to emphasize this difference.

It is possible to use expressions as statements. However this does cause a warning.

```
gap> if i <> 0 then k = 16/i; fi;
Syntax error: warning, this statement has no effect
if i <> 0 then k = 16/i; fi;
^
```

As you can see from the example this is useful for those users who are used to languages where '=' instead of '^:=' denotes assignment.

A sequence of one or more statements is a statement sequence, and may occur everywhere instead of a single statement. There is nothing like PASCAL's BEGIN-END, instead each construct is terminated

by a keyword. The most simple statement sequence is a single semicolon, which can be used as an empty statement sequence.

2.12 Assignments

'<var> \:= <expr>;'

The **assignment** has the effect of assigning the value of the expressions <expr> to the variable <var>.

The variable <var> may be an ordinary variable (see "Variables"), a list element selection '<list-var>[<int-expr>]' (see "List Assignment") or a record component selection '<record-var>.<ident>' (see "Record Assignment"). Since a list element or a record component may itself be a list or a record the left hand side of an assignment may be arbitrarily complex.

Note that variables do not have a type. Thus any value may be assigned to any variable. For example a variable with an integer value may be assigned a permutation or a list or anything else.

If the expression <expr> is a function call then this function must return a value. If the function does not return a value an error is signalled and you enter a break loop. As usual you can leave the break loop with 'quit;'. If you enter 'return <return-expr>;' the value of the expression <return-expr> is assigned to the variable, and execution continues after the assignment.

```
gap> S6 := rec( size := 720 );; S6;
rec(
  size := 720 )
gap> S6.generators := [ (1,2), (1,2,3,4,5) ];; S6;
rec(
  size := 720,
  generators := [ (1,2), (1,2,3,4,5) ] )
gap> S6.generators[2] := (1,2,3,4,5,6);; S6;
rec(
  size := 720,
  generators := [ (1,2), (1,2,3,4,5,6) ] )
```

2.13 Procedure Calls

'<procedure-var>();'

'<procedure-var>(<arg-expr> \{'<arg-expr>\});'

The procedure call has the effect of calling the procedure <procedure-var>. A procedure call is done exactly like a function call (see "Function Calls"). The distinction between functions and procedures is only for the sake of the discussion, GAP does not distinguish between them.

A **function** does return a value but does not produce a side effect. As a convention the name of a function is a noun, denoting what the function returns, e.g., 'Length', 'Concatenation' and 'Order'.

A **procedure** is a function that does not return a value but produces some effect. Procedures are called only for this effect. As a convention the name of a procedure is a verb, denoting what the procedure does, e.g., 'Print', 'Append' and 'Sort'.

```
gap> Read( "myfile.g" );      & a call to the procedure Read
gap> l := [ 1, 2 ];;
gap> Append( l, [3,4,5] );    & a call to the procedure Append
```

2.14 If

```
'if <bool-expr1> then <statements1>
{ elif <bool-expr2> then <statements2> }
[ else <statements3> ]
fi;'
```

The 'if' statement allows one to execute statements depending on the value of some boolean expression. The execution is done as follows.

First the expression <bool-expr1> following the 'if' is evaluated. If it evaluates to 'true' the statement sequence <statements1> after the first 'then' is executed, and the execution of the 'if' statement is complete.

Otherwise the expressions <bool-expr2> following the 'elif' are evaluated in turn. There may be any number of 'elif' parts, possibly none at all. As soon as an expression evaluates to 'true' the corresponding statement sequence <statements2> is executed and execution of the 'if' statement is complete.

If the 'if' expression and all, if any, 'elif' expressions evaluate to 'false' and there is an 'else' part, which is optional, its statement sequence <statements3> is executed and the execution of the 'if' statement is complete. If there is no 'else' part the 'if' statement is complete without executing any statement sequence.

Since the 'if' statement is terminated by the 'fi' keyword there is no question where an 'else' part belongs, i.e., GAP has no dangling else.

```
In 'if <expr1> then if <expr2> then <stats1> else <stats2> fi; fi;'
the 'else' part belongs to the second 'if' statement, whereas in
'if <expr1> then if <expr2> then <stats1> fi; else <stats2> fi;'
the 'else' part belongs to the first 'if' statement.
```

Since an if statement is not an expression it is not possible to write

```
abs := if x > 0 then x; else -x; fi;
```

which would, even if legal syntax, be meaningless, since the 'if' statement does not produce a value that could be assigned to 'abs'.

If one expression evaluates neither to 'true' nor to 'false' an error is signalled and a break loop is entered. As usual you can leave the break loop with 'quit;'. If you enter 'return true;',

execution of the 'if' statement continues as if the expression whose evaluation failed had evaluated to 'true'. Likewise, if you enter 'return false;', execution of the 'if' statement continues as if the expression whose evaluation failed had evaluated to 'false'.

```
gap> i := 10;;
gap> if 0 < i then
>     s := 1;
>     elif i < 0 then
>         s := -1;
>     else
>         s := 0;
>     fi;
gap> s;
1          # the sign of i
```

2.15 While

```
'while <bool-expr> do <statements> od;'
```

The 'while' loop executes the statement sequence <statements> while the condition <bool-expr> evaluates to 'true'.

First <bool-expr> is evaluated. If it evaluates to 'false' execution of the 'while' loop terminates and the statement immediately following the 'while' loop is executed next. Otherwise if it evaluates to 'true' the <statements> are executed and the whole process begins again.

The difference between the 'while' loop and the 'repeat until' loop (see "Repeat") is that the <statements> in the 'repeat until' loop are executed at least once, while the <statements> in the 'while' loop are not executed at all if <bool-expr> is 'false' at the first iteration.

If <bool-expr> does not evaluate to 'true' or 'false' an error is signalled and a break loop is entered. As usual you can leave the break loop with 'quit;'. If you enter 'return false;', execution continues with the next statement immediately following the 'while' loop. If you enter 'return true;', execution continues at <statements>, after which the next evaluation of <bool-expr> may cause another error.

```
gap> i := 0;; s := 0;;
gap> while s <= 200 do
>     i := i + 1; s := s + i^2;
>     od;
gap> s;
204          # first sum of the first i squares larger than 200
```

2.16 Repeat

```
'repeat <statements> until <bool-expr>;'
```

The **'repeat'** loop executes the statement sequence `<statements>` until the condition `<bool-expr>` evaluates to **'true'**.

First `<statements>` are executed. Then `<bool-expr>` is evaluated. If it evaluates to **'true'** the **'repeat'** loop terminates and the statement immediately following the **'repeat'** loop is executed next. Otherwise if it evaluates to **'false'** the whole process begins again with the execution of the `<statements>`.

The difference between the **'while'** loop (see "While") and the **'repeat until'** loop is that the `<statements>` in the **'repeat until'** loop are executed at least once, while the `<statements>` in the **'while'** loop are not executed at all if `<bool-expr>` is **'false'** at the first iteration.

If `<bool-expr>` does not evaluate to **'true'** or **'false'** a error is signalled and a break loop is entered. As usual you can leave the break loop with **'quit;'**. If you enter **'return true;'**, execution continues with the next statement immediately following the **'repeat'** loop. If you enter **'return false;'**, execution continues at `<statements>`, after which the next evaluation of `<bool-expr>` may cause another error.

```
gap> i := 0;; s := 0;;
gap> repeat
>     i := i + 1; s := s + i^2;
> until s > 200;
gap> s;
204      # first sum of the first i squares larger than 200
```

2.17 For

```
'for <simple-var> in <list-expr> do <statements> od;'
```

The **'for'** loop executes the statement sequence `<statements>` for every element of the list `<list-expr>`.

The statement sequence `<statements>` is first executed with `<simple-var>` bound to the first element of the list `<list>`, then with `<simple-var>` bound to the second element of `<list>` and so on. `<simple-var>` must be a simple variable, it must not be a list element selection **'<list-var>[<int-expr>']** or a record component selection **'<record-var>.<ident>'**.

The execution of the **'for'** loop is exactly equivalent to the **'while'** loop

```
<loop-list> := <list>;
<loop-index> := 1;
while <loop-index> <= Length(<loop-list>) do
    <variable> := <loop-list>[<loop-index>];
    <statements>
    <loop-index> := <loop-index> + 1;
od;
```

with the exception that `<loop-list>` and `<loop-index>` are different variables for each `'for'` loop that do not interfere with each other.

The list `<list>` is very often a range.

```
'for <variable> in [<from>..<to>] do <statements> od;'
```

corresponds to the more common

```
'for <variable> from <from> to <to> do <statements> od;'
```

in other programming languages.

```
gap> s := 0;;
gap> for i in [1..100] do
>   s := s + i;
> od;
gap> s;
5050
```

Note in the following example how the modification of the `*list*` in the loop body causes the loop body also to be executed for the new values

```
gap> l := [ 1, 2, 3, 4, 5, 6 ];;
gap> for i in l do
>   Print( i, " " );
>   if i mod 2 = 0 then Add( l, 3 * i / 2 ); fi;
> od; Print( "\n" );
1 2 3 4 5 6 3 6 9 9
gap> l;
[ 1, 2, 3, 4, 5, 6, 3, 6, 9, 9 ]
```

Note in the following example that the modification of the `*variable*` that holds the list has no influence on the loop

```
gap> l := [ 1, 2, 3, 4, 5, 6 ];;
gap> for i in l do
>   Print( i, " " );
>   l := [];
> od; Print( "\n" );
1 2 3 4 5 6
gap> l;
[ ]
```

2.18 Functions

```
function ( [ <arg-ident> \{, <arg-ident>\} ] )
```

```

    [ local    <loc-ident> \{, <loc-ident>\} ; ]
    <statements>
end

```

A function is in fact a literal and not a statement. Such a function literal can be assigned to a variable or to a list element or a record component. Later this function can be called as described in "Function Calls".

The following is an example of a function definition. It is a function to compute values of the Fibonacci sequence

```

gap> fib := function ( n )
>     local  f1, f2, f3, i;
>     f1 := 1; f2 := 1;
>     for i in [3..n] do
>         f3 := f1 + f2;
>         f1 := f2;
>         f2 := f3;
>     od;
>     return f2;
> end;;
gap> List( [1..10], fib );
[ 1, 1, 2, 3, 5, 8, 13, 21, 34, 55 ]

```

Because for each of the formal arguments <arg-ident> and for each of the formal locals <loc-ident> a new variable is allocated when the function is called (see "Function Calls"), it is possible that a function calls itself. This is usually called *recursion*. The following is a recursive function that computes values of the Fibonacci sequence

```

gap> fib := function ( n )
>     if n < 3 then
>         return 1;
>     else
>         return fib(n-1) + fib(n-2);
>     fi;
> end;;
gap> List( [1..10], fib );
[ 1, 1, 2, 3, 5, 8, 13, 21, 34, 55 ]

```

Note that the recursive version needs ' $2 \cdot \text{fib}(\langle n \rangle) - 1$ ' steps to compute ' $\text{fib}(\langle n \rangle)$ ', while the iterative version of ' fib ' needs only ' $\langle n \rangle - 2$ ' steps. Both are not optimal however, the library function ' Fibonacci ' only needs on the order of ' $\text{Log}(\langle n \rangle)$ ' steps.

'<arg-ident> -> <expr>'

This is a shorthand for

```
'function ( <arg-ident> ) return <expr>; end'.
```

<arg-ident> must be a single identifier, i.e., it is not possible to write functions of several arguments this way. Also 'arg' is not treated specially, so it is also impossible to write functions that take a variable number of arguments this way.

The following is an example of a typical use of such a function

```
gap> Sum( List( [1..100], x -> x^2 ) );
338350
```

When a function <fun1> definition is evaluated inside another function <fun2>, GAP binds all the identifiers inside the function <fun1> that are identifiers of an argument or a local of <fun2> to the corresponding variable. This set of bindings is called the environment of the function <fun1>. When <fun1> is called, its body is executed in this environment. The following implementation of a simple stack uses this. Values can be pushed onto the stack and then later be popped off again. The interesting thing here is that the functions 'push' and 'pop' in the record returned by 'Stack' access the local variable 'stack' of 'Stack'. When 'Stack' is called a new variable for the identifier 'stack' is created. When the function definitions of 'push' and 'pop' are then evaluated (as part of the 'return' statement) each reference to 'stack' is bound to this new variable. Note also that the two stacks 'A' and 'B' do not interfere, because each call of 'Stack' creates a new variable for 'stack'.

```
gap> Stack := function ()
>     local  stack;
>     stack := [];
>     return rec(
>         push := function ( value )
>             Add( stack, value );
>         end,
>         pop := function ()
>             local  value;
>             value := stack[Length(stack)];
>             Unbind( stack[Length(stack)] );
>             return value;
>         end
>     );
> end;;
gap> A := Stack();
gap> B := Stack();
gap> A.push( 1 ); A.push( 2 ); A.push( 3 );
gap> B.push( 4 ); B.push( 5 ); B.push( 6 );
gap> A.pop(); A.pop(); A.pop();
3
```



```
2
1
gap> B.pop(); B.pop(); B.pop();
6
5
4
```

This feature should be used rarely, since its implementation in GAP is not very efficient.

2.19 Return

'return;'

In this form **'return'** terminates the call of the innermost function that is currently executing, and control returns to the calling function. An error is signalled if no function is currently executing. No value is returned by the function.

'return <expr>;'

In this form **'return'** terminates the call of the innermost function that is currently executing, and returns the value of the expression **<expr>**. Control returns to the calling function. An error is signalled if no function is currently executing.

Both statements can also be used in break loops. **'return;'** has the effect that the computation continues where it was interrupted by an error or the user hitting **'<ctr>C'**. **'return <expr>;'** can be used to continue execution after an error. What happens with the value **<expr>** depends on the particular error.

2.20 The Syntax in BNF

This section contains the definition of the GAP syntax in Backus-Naur form.

A BNF is a set of rules, whose left side is the name of a syntactical construct. Those names are enclosed in angle brackets and written in *<italics>*. The right side of each rule contains a possible form for that syntactic construct. Each right side may contain names of other syntactic constructs, again enclosed in angle brackets and written in *<italics>*, or character sequences that must occur literally; they are written in **'typewriter style'**.

Furthermore each righthand side can contain the following metasympols written in ***boldface***. If the right hand side contains forms separated by a pipe symbol (**|**) this means that one of the possible forms can occur. If a part of a form is enclosed in square brackets (**[]**) this means that this part is optional, i.e. might be present or missing. If part of the form is enclosed in curly braces (**{ }**) this means that the part may occur arbitrarily often, or possibly be missing.

<Ident>	= 'a' ... 'z' 'A' ... 'Z' '_' { 'a' ... 'z' 'A' ... 'Z' '0' ... '9' '_' }
<Var>	= <Ident> <Var> '.' <Ident> <Var> '.' '(' <Expr> ')' <Var> '[' <Expr> ']' <Var> '{' <Expr> '}' <Var> '(' [<Expr> { ',' <Expr> }] ')'
<List>	= '[' [<Expr>] { ',' [<Expr>] } ']' '[' <Expr> [, <Expr>] ... <Expr> ']'
<Record>	= 'rec' ([<Ident> '\:=' <Expr> { ',' <Ident> '\:=' <Expr> }])'
<Permutation>	= '(' <Expr> { ',' <Expr> } ') { '(' <Expr> { ',' <Expr> } ') }'
<Function>	= 'function' ([<Ident> { ',' <Ident> }])' ['local' <Ident> { ',' <Ident> } ';'] <Statements> 'end'
<Char>	= '\verb' <any character> '
<String>	= '\"' { <any character> } '\"'
<Int>	= '0' '1' ... '9' { '0' '1' ... '9' }
<Atom>	= <Int> <Var> '(' <Expr> ')' <Permutation> <Char> <String> <Function> <List> <Record>
<Factor>	= { '+' '-' } <Atom> ['^' { '+' '-' } <Atom>]
<Term>	= <Factor> { '*' '/' 'mod' <Factor> }
<Arith>	= <Term> { '+' '-' <Term> }
<Rel>	= { 'not' } <Arith> { '<' '>' '<=' '>=' 'in' <Arith> }
<And>	= <Rel> { 'and' <Rel> }
<Log>	= <And> { 'or' <And> }
<Expr>	= <Log> <Var> ['->' <Log>]
<Statement>	= <Expr> <Var> '\:=' <Expr> 'if' <Expr> 'then' <Statements> { 'elif' <Expr> 'then' <Statements> } ['else' <Statements>] 'fi' 'for' <Var> 'in' <Expr> 'do' <Statements> 'od' 'while' <Expr> 'do' <Statements> 'od' 'repeat' <Statements> 'until' <Expr>

```

    || 'return' [ <Expr> ]
    || 'quit'
<Statements> = { <Statement> ';' }
    || ';'

```


Chapter 3

Lists

Lists are the most important way to collect objects and treat them together. A **list** is a collection of elements. A list also implies a partial mapping from the integers to the elements. I.e., there is a first element of a list, a second, a third, and so on.

List constants are written by writing down the elements in order between square brackets '[', ']', and separating them with commas ','. An **empty list**, i.e., a list with no elements, is written as '[]'.

```
gap> [ 1, 2, 3 ];
[ 1, 2, 3 ]      # a list with three elements
gap> [ [], [ 1 ], [ 1, 2 ] ];
[ [ ], [ 1 ], [ 1, 2 ] ]      # a list may contain other lists
```

Usually a list has no holes, i.e., contain an element at every position. However, it is absolutely legal to have lists with holes. They are created by leaving the entry between the commas empty. Lists with holes are sometimes convenient when the list represents a mapping from a finite, but not consecutive, subset of the positive integers. We say that a list that has no holes is **dense**.

```
gap> l := [ , 4, 9,, 25,, 49,,,, 121 ];;
gap> l[3];
9
gap> l[4];
Error, List Element: <list>[4] must have a value
```

It is most common that a list contains only elements of one type. This is not a must though. It is absolutely possible to have lists whose elements are of different types. We say that a list whose elements are all of the same type is **homogeneous**.

```
gap> l := [ 1, E(2), Z(3), (1,2,3), [1,2,3], "What a mess" ];;
gap> l[1]; l[3]; l[5][2];
1
Z(3)
2
```

The first sections describe the functions that test if an object is a list and convert an object to a list (see "IsList" and "List").

The next section describes how one can access elements of a list (see "List Elements" and "Length").

The next sections describe how one can change lists (see "List Assignment", "Add", "Append", "Identical Lists", "Enlarging Lists").

The next sections describe the operations applicable to lists (see "Comparisons of Lists" and "Operations for Lists").

The next sections describe how one can find elements in a list (see "In", "Position", "PositionSorted", "PositionProperty").

The next sections describe the functions that construct new lists, e.g., sublists (see "Concatenation", "Flat", "Reversed", "Sublist", "Cartesian").

The next sections describe the functions deal with the subset of elements of a list that have a certain property (see "Number", "Collected", "Filtered", "ForAll", "ForAny", "First").

The next sections describe the functions that sort lists (see "Sort", "SortParallel", "Sortex", "Permuted").

The next sections describe the functions to compute the product, sum, maximum, and minimum of the elements in a list (see "Product", "Sum", "Maximum", "Minimum", "Iterated").

The final section describes the function that takes a random element from a list (see "RandomList").

Lists are also used to represent sets, subsets, vectors, and ranges.

3.1 IsList

```
'IsList( <obj> )'
```

'IsList' returns 'true' if the argument <obj>, which can be an arbitrary object, is a list and 'false' otherwise. Will signal an error if <obj> is an unbound variable.

```
gap> IsList( [ 1, 3, 5, 7 ] );
true
gap> IsList( 1 );
false
```

3.2 List

```
'List( <obj> )'
```

```
'List( <list>, <func> )'
```

In its first form 'List' returns the argument <obj>, which must be a list, a permutation, a string or a word, converted into a list. If <obj> is a list, it is simply returned. If <obj> is a permutation, 'List' returns a list where the <i>-th element is the image of <i> under the permutation <obj>. If <obj> is a word, 'List' returns a list where the <i>-th element is the <i>-th generator of the word, as a word of length 1.

```
gap> List( [1,2,3] );
[ 1, 2, 3 ]
gap> List( (1,2)(3,4,5) );
[ 2, 1, 4, 5, 3 ]
```

In its second form 'List' returns a new list, where each element is the result of applying the function <func>, which must take exactly one argument and handle the elements of <list>, to the corresponding element of the list <list>. The list <list> must not contain holes.

```
gap> List( [1,2,3], x->x^2 );
[ 1, 4, 9 ]
gap> List( [1..10], IsPrime );
[ false, true, true, false, true, false, true, false, false, false ]
```

Note that this function is called 'map' in Lisp and many other similar programming languages. This name violates the GAP rule that verbs are used for functions that change their arguments. According to this rule 'map' would change <list>, replacing every element with the result of the application <func> to this argument.

3.3 List Elements

'<list>[<pos>]'

The above construct evaluates to the <pos>-th element of the list <list>. <pos> must be a positive integer. List indexing is done with origin 1, i.e., the first element of the list is the element at position 1.

```
gap> l := [ 2, 3, 5, 7, 11, 13 ];;
gap> l[1];
2
gap> l[2];
3
gap> l[6];
13
```

If <list> does not evaluate to a list, or <pos> does not evaluate to a positive integer, or '<list>[<pos>]' is unbound an error is signalled. As usual you can leave the break loop with 'quit;'. On the other hand you can return a result to be used in place of the list element by 'return <expr>;'.

'<list>\{ <poss> \}'

The above construct evaluates to a new list <new> whose first element is '<list>[<poss>[1]]', whose second element is '<list>[<poss>[2]]', and so on. <poss> must be a dense list of positive integers, it need, however, not be sorted and may contain duplicate elements. If for any <i>, '<list>[<poss>[<i>]]' is unbound, an error is signalled.

```
gap> l := [ 2, 3, 5, 7, 11, 13, 17, 19 ];;
gap> l{[4..6]};
[ 7, 11, 13 ]
gap> l{[1,7,1,8]};
[ 2, 17, 2, 19 ]
```

The result is a new list, that is not identical to any other list. The elements of that list however are identical to the corresponding elements of the left operand (see "Identical Lists").

It is possible to nest such sublist extractions, as can be seen in the following example.

```
gap> m := [ [1,2,3], [4,5,6], [7,8,9], [10,11,12] ];;
gap> m{[1,2,3]}{[3,2]};
[ [ 3, 2 ], [ 6, 5 ], [ 9, 8 ] ]
gap> l := m{[1,2,3]};; l{[3,2]};
[ [ 7, 8, 9 ], [ 4, 5, 6 ] ]
```

Note the difference between the two examples. The latter extracts elements 1, 2, and 3 from <m> and then extracts the elements 3 and 2 from *this list*. The former extracts elements 1, 2, and 3 from <m> and then extracts the elements 3 and 2 from *each of those element lists*.

To be precise. With each selector '`<pos>`' or '`\{<poss>\}`' we associate a *level* that is defined as the number of selectors of the form '`\{<poss>\}`' to its left in the same expression. For example

	1[pos1]{poss2}{poss3}[pos4]{poss5}[pos6]
level	0 0 1 1 1 2

Then a selector '`<list>[<pos>]`' of level <level> is computed as '`ListElement(<list>,<pos>,<level>)`', where '`ListElement`' is defined as follows

```
ListElement := function ( list, pos, level )
  if level = 0 then
    return list[pos];
  else
    return List( list, elm -> ListElement(elm,pos,level-1) );
  fi;
end;
```

and a selector '`<list>\{<poss>\}`' of level <level> is computed as '`ListElements(<list>,<poss>,<level>)`', where '`ListElements`' is defined as follows

```
ListElements := function ( list, poss, level )
  if level = 0 then
    return list{poss};
  else
    return List( list, elm -> ListElements(elm,poss,level-1) );
  fi;
end;
```


3.4 Length

`'Length(<list>)'`

`'Length'` returns the length of the list `<list>`. The **length** is defined as 0 for the empty list, and as the largest positive integer `<index>` such that `'<list>[<index>]'` has an assigned value for nonempty lists. Note that the length of a list may change if new elements are added to it or assigned to previously unassigned positions.

```
gap> Length( [] );
0
gap> Length( [ 2, 3, 5, 7, 11, 13, 17, 19 ] );
8
gap> Length( [ 1, 2,,, 5 ] );
5
```

For lists that contain no holes `'Length'`, `'Number'` (see "Number"), and `'Size'` return the same value. For lists with holes `'Length'` returns the largest index of a bound entry, `'Number'` returns the number of bound entries, and `'Size'` signals an error.

3.5 List Assignment

`'<list>[<pos>] \:= <object>;'`

The list assignment assigns the object `<object>`, which can be of any type, to the list entry at the position `<pos>`, which must be a positive integer, in the list `<list>`. That means that accessing the `<pos>`-th element of the list `<list>` will return `<object>` after this assignment.

```
gap> l := [ 1, 2, 3 ];;
gap> l[1] := 3;; l; # assign a new object
[ 3, 2, 3 ]
gap> l[2] := [ 4, 5, 6 ];; l; # <object> may be of any type
[ 3, [ 4, 5, 6 ], 3 ]
gap> l[ l[1] ] := 10;; l; # <index> may be an expression
[ 3, [ 4, 5, 6 ], 10 ]
```

If the index `<pos>` is larger than the length of the list `<list>` (see "Length"), the list is automatically enlarged to make room for the new element. Note that it is possible to generate lists with holes that way.

```
gap> l[4] := "another entry";; l; # <list> is enlarged
[ 3, [ 4, 5, 6 ], 10, "another entry" ]
gap> l[ 10 ] := 1;; l; # now <list> has a hole
[ 3, [ 4, 5, 6 ], 10, "another entry",,,,,, 1 ]
```

The function 'Add' (see "Add") should be used if you want to add an element to the end of the list. Note that assigning to a list changes the list. The ability to change an object is only available for lists and records (see "Identical Lists").

If <list> does not evaluate to a list, <pos> does not evaluate to a positive integer or <object> is a call to a function which does not return a value, for example 'Print', an error is signalled. As usual you can leave the break loop with 'quit;'. On the other hand you can continue the assignment by returning a list, an index or an object using 'return <expr>;'.

```
'<list>\{ <poss> \} \:= <objects>;'
```

The list assignment assigns the object '<objects>[1]', which can be of any type, to the list <list> at the position '<poss>[1]', the object '<objects>[2]' to '<list>[<poss>[2]]', and so on. <poss> must be a dense list of positive integers, it need, however, not be sorted and may contain duplicate elements. <objects> must be a dense list and must have the same length as <poss>.

```
gap> l := [ 2, 3, 5, 7, 11, 13, 17, 19 ];;
gap> l{[1..4]} := [10..13];; l;
[ 10, 11, 12, 13, 11, 13, 17, 19 ]
gap> l{[1,7,1,10]} := [ 1, 2, 3, 4 ];; l;
[ 3, 11, 12, 13, 11, 13, 2, 19,, 4 ]
```

It is possible to nest such sublist assignments, as can be seen in the following example.

```
gap> m := [ [1,2,3], [4,5,6], [7,8,9], [10,11,12] ];;
gap> m{[1,2,3]}{[3,2]} := [ [11,12], [13,14], [15,16] ];; m;
[ [ 1, 12, 11 ], [ 4, 14, 13 ], [ 7, 16, 15 ], [ 10, 11, 12 ] ]
```

The exact behaviour is defined in the same way as for list extractions (see "List Elements"). Namely with each selector '[<pos>]' or '\{<poss>\}' we associate a *level* that is defined as the number of selectors of the form '\{<poss>\}' to its left in the same expression. For example

	l	[pos1]	{poss2}	{poss3}	[pos4]	{poss5}	[pos6]
level	0	0	1	1	1	2	

Then a list assignment '<list>[<pos>] \:= <vals>;' of level <level> is computed as 'ListAssignment(<list>[<pos>], <vals>, <level>)' where 'ListAssignment' is defined as follows

```
ListAssignment := function ( list, pos, vals, level )
  local i;
  if level = 0 then
    list[pos] := vals;
  else
    for i in [1..Length(list)] do
      ListAssignment( list[i], pos, vals[i], level-1 );
    od;
  fi;
end;
```

and a list assignment '`<list>\{<poss>\} \:= <vals>`' of level `<level>` is computed as '`ListAssignments(` where '`ListAssignments`' is defined as follows

```
ListAssignments := function ( list, poss, vals, level )
  local i;
  if level = 0 then
    list{poss} := vals;
  else
    for i in [1..Length(list)] do
      ListAssignments( list[i], poss, vals[i], level-1 );
    od;
  fi;
end;
```

3.6 Add

`'Add(<list>, <elm>)'`

`'Add'` adds the element `<elm>` to the end of the list `<list>`, i.e., it is equivalent to the assignment '`<list>[Length(<list>) + 1] \:= <elm>`'. The list is automatically enlarged to make room for the new element. `'Add'` returns nothing, it is called only for its side effect.

Note that adding to a list changes the list. The ability to change an object is only available for lists and records (see "Identical Lists").

To add more than one element to a list use `'Append'` (see "Append").

```
gap> l := [ 2, 3, 5 ];; Add( l, 7 ); l;
[ 2, 3, 5, 7 ]
```

3.7 Append

`'Append(<list1>, <list2>)'`

`'Append'` adds (see "Add") the elements of the list `<list2>` to the end of the list `<list1>`. `<list2>` may contain holes, in which case the corresponding entries in `<list1>` will be left unbound. `'Append'` returns nothing, it is called only for its side effect.

```
gap> l := [ 2, 3, 5 ];; Append( l, [ 7, 11, 13 ] ); l;
[ 2, 3, 5, 7, 11, 13 ]
gap> Append( l, [ 17,, 23 ] ); l;
[ 2, 3, 5, 7, 11, 13, 17,, 23 ]
```

Note that appending to a list changes the list. The ability to change an object is only available for lists and records (see "Identical Lists").

Note that `'Append'` changes the first argument, while `'Concatenation'` (see "Concatenation") creates a new list and leaves its arguments unchanged. As usual the name of the function that work destructively is a verb, but the name of the function that creates a new object is a substantive.

3.8 Identical Lists

With the list assignment (see "List Assignment", "Add", "Append") it is possible to change a list. The ability to change an object is only available for lists and records. This section describes the semantic consequences of this fact.

You may think that in the following example the second assignment changes the integer, and that therefore the above sentence, which claimed that only lists and records can be changed is wrong

```
i := 3;
i := i + 1;
```

But in this example not the *integer* '3' is changed by adding one to it. Instead the *variable* 'i' is changed by assigning the value of 'i+1', which happens to be '4', to 'i'. The same thing happens in the following example

```
l := [ 1, 2 ];
l := [ 1, 2, 3 ];
```

The second assignment does not change the first list, instead it assigns a new list to the variable 'l'. On the other hand, in the following example the list is changed by the second assignment.

```
l := [ 1, 2 ];
l[3] := 3;
```

To understand the difference first think of a variable as a name for an object. The important point is that a list can have several names at the same time. An assignment '`<var> \:= <list>;`' means in this interpretation that `<var>` is a name for the object `<list>`. At the end of the following example 'l2' still has the value '[1, 2]' as this list has not been changed and nothing else has been assigned to it.

```
l1 := [ 1, 2 ];
l2 := l1;
l1 := [ 1, 2, 3 ];
```

But after the following example the list for which 'l2' is a name has been changed and thus the value of 'l2' is now '[1, 2, 3]'.

```
l1 := [ 1, 2 ];
l2 := l1;
l1[3] := 3;
```

We shall say that two lists are *identical* if changing one of them by a list assignment also changes the other one. This is slightly incorrect, because if *two* lists are identical, there are actually only two names for *one* list. However, the correct usage would be very awkward and would only add to the confusion. Note that two identical lists must be equal, because there is only one list with two different names. Thus identity is an equivalence relation that is a refinement of equality.

Let us now consider under which circumstances two lists are identical.

If you enter a list literal then the list denoted by this literal is a new list that is not identical to any other list. Thus in the following example 'l1' and 'l2' are not identical, though they are equal of course.

```
l1 := [ 1, 2 ];
l2 := [ 1, 2 ];
```

Also in the following example, no lists in the list 'l' are identical.

```
l := [];
for i in [1..10] do l[i] := [ 1, 2 ]; od;
```

If you assign a list to a variable no new list is created. Thus the list value of the variable on the left hand side and the list on the right hand side of the assignment are identical. So in the following example 'l1' and 'l2' are identical lists.

```
l1 := [ 1, 2 ];
l2 := l1;
```

If you pass a list as argument, the old list and the argument of the function are identical. Also if you return a list from a function, the old list and the value of the function call are identical. So in the following example 'l1' and 'l2' are identical list

```
l1 := [ 1, 2 ];
f := function ( l ) return l; end;
l2 := f( l1 );
```

The functions 'Copy' and 'ShallowCopy' (see "Copy" and "ShallowCopy") accept a list and return a new list that is equal to the old list but that is *not* identical to the old list. The difference between 'Copy' and 'ShallowCopy' is that in the case of 'ShallowCopy' the corresponding elements of the new and the old lists will be identical, whereas in the case of 'Copy' they will only be equal. So in the following example 'l1' and 'l2' are not identical lists.

```
l1 := [ 1, 2 ];
l2 := Copy( l1 );
```

If you change a list it keeps its identity. Thus if two lists are identical and you change one of them, you also change the other, and they are still identical afterwards. On the other hand, two lists that are not identical will never become identical if you change one of them. So in the following example both 'l1' and 'l2' are changed, and are still identical.

```
l1 := [ 1, 2 ];
l2 := l1;
l1[1] := 2;
```

3.9 IsIdentical

'IsIdentical(<l>, <r>)'

'IsIdentical' returns 'true' if the objects <l> and <r> are identical. Unchangeable objects are considered identical if they are equal. Changeable objects, i.e., lists and records, are identical if changing one of them by an assignment also changes the other one, as described in "Identical Lists".

```
gap> IsIdentical( 1, 1 );
true
gap> IsIdentical( 1, () );
false
gap> l := [ \verb|h'|, \verb|a'|, \verb|l'|, \verb|l'|, \verb|o'| ];
gap> l = "hallo";
true
gap> IsIdentical( l, "hallo" );
false
```

3.10 Enlarging Lists

The previous section (see "List Assignment") told you (among other things), that it is possible to assign beyond the logical end of a list, automatically enlarging the list. This section tells you how this is done.

It would be extremely wasteful to make all lists large enough so that there is room for all assignments, because some lists may have more than 100000 elements, while most lists have less than 10 elements.

On the other hand suppose every assignment beyond the end of a list would be done by allocating new space for the list and copying all entries to the new space. Then creating a list of 1000 elements by assigning them in order, would take half a million copy operations and also create a lot of garbage that the garbage collector would have to reclaim.

So the following strategy is used. If a list is created it is created with exactly the correct size. If a list is enlarged, because of an assignment beyond the end of the list, it is enlarged by at least ' $\lceil \text{length}/8 + 4 \rceil$ ' entries. Therefore the next assignments beyond the end of the list do not need

to enlarge the list. For example creating a list of 1000 elements by assigning them in order, would now take only 32 enlargements.

The result of this is of course that the **physical length**, which is also called the size, of a list may be different from the **logical length**, which is usually called simply the length of the list. Aside from the implications for the performance you need not be aware of the physical length. In fact all you can ever observe, for example by calling `'Length'` is the logical length.

Suppose that `'Length'` would have to take the physical length and then test how many entries at the end of a list are unassigned, to compute the logical length of the list. That would take too much time. In order to make `'Length'`, and other functions that need to know the logical length, more efficient, the length of a list is stored along with the list.

A note aside. In the previous version 2.4 of **GAP** a list was indeed enlarged every time an assignment beyond the end of the list was performed. To deal with the above inefficiency the following hacks were used. Instead of creating lists in order they were usually created in reverse order. In situations where this was not possible a dummy assignment to the last position was performed, for example

```
l := [];
l[1000] := "dummy";
l[1] := first_value();
for i from 2 to 1000 do l[i] := next_value(l[i-1]); od;
```

3.11 Comparisons of Lists

```
'<list1> = <list2>'
'<list1> <> <list2>'
```

The equality operator `'='` evaluates to `'true'` if the two lists `<list1>` and `<list2>` are equal and `'false'` otherwise. The inequality operator `'<>'` evaluates to `'true'` if the two lists are not equal and `'false'` otherwise. Two lists `<list1>` and `<list2>` are equal if and only if for every index `<i>`, either both entries `'<list1>[<i>]'` and `'<list2>[<i>]'` are unbound, or both are bound and are equal, i.e., `'<list1>[<i>] = <list2>[<i>]'` is `'true'`.

```
gap> [ 1, 2, 3 ] = [ 1, 2, 3 ];
true
gap> [ , 2, 3 ] = [ 1, 2, ];
false
gap> [ 1, 2, 3 ] = [ 3, 2, 1 ];
false
```

```
'<list1> <\ <list2>', '<list1> <= <list2>' '<list1> > <list2>', '<list1> >= <list2>'
```

The operators `'<'`, `'<='`, `'>'` and `'>='` evaluate to `'true'` if the list `<list1>` is less than, less than or equal to, greater than, or greater than or equal to the list `<list2>` and to `'false'` otherwise. Lists are ordered lexicographically, with unbound entries comparing very small. That means the following. Let `<i>` be the smallest positive integer `<i>`, such that neither both entries `'<list1>[<i>]'`

and '`<list2>[<i>]`' are unbound, nor both are bound and equal. Then `<list1>` is less than `<list2>` if either '`<list1>[<i>]`' is unbound (and '`<list2>[<i>]`' is not) or both are bound and '`<list1>[<i>] <\ <list2>[<i>]`' is 'true'.

```
gap> [ 1, 2, 3, 4 ] < [ 1, 2, 4, 8 ];
true      # \verb|'<list1>[3] <\ <list2>[3]'|
gap> [ 1, 2, 3 ] < [ 1, 2, 3, 4 ];
true      # \verb|'<list1>[4]'| is unbound and therefore very small
gap> [ 1, , 3, 4 ] < [ 1, 2, 3 ];
true      # \verb|'<list1>[2]'| is unbound and therefore very small
```

You can also compare objects of other types, for example integers or permutations with lists. Of course those objects are never equal to a list. Records are greater than lists, objects of every other type are smaller than lists.

```
gap> 123 < [ 1, 2, 3 ];
true
gap> [ 1, 2, 3 ] < rec( a := 123 );
true
```

3.12 Operations for Lists

```
'<list> \*\ <obj>'
'<obj> \*\ <list>'
```

The operator '`*`' evaluates to the product of list `<list>` by an object `<obj>`. The product is a new list that at each position contains the product of the corresponding element of `<list>` by `<obj>`. `<list>` may contain holes, in which case the result will contain holes at the same positions.

The elements of `<list>` and `<obj>` must be objects of the following types; integers, rationals, cyclotomics, elements of a finite field, permutations, matrices, words in abstract generators, or words in solvable groups.

```
gap> [ 1, 2, 3 ] * 2;
[ 2, 4, 6 ]
gap> 2 * [ 2, 3,, 5,, 7 ];
[ 4, 6,, 10,, 14 ]
gap> [ (), (2,3), (1,2), (1,2,3), (1,3,2), (1,3) ] * (1,4);
[ (1,4), (1,4)(2,3), (1,2,4), (1,2,3,4), (1,3,2,4), (1,3,4) ]
```

Many more operators are available for vectors and matrices, which are also represented by lists.

3.13 In

`'<elm> in <list>'`

The `'in'` operator evaluates to `'true'` if the object `<elm>` is an element of the list `<list>` and to `'false'` otherwise. `<elm>` is an element of `<list>` if there is a positive integer `<index>` such that `'<list>[<index>]=<elm>'` is `'true'`. `<elm>` may be an object of an arbitrary type and `<list>` may be a list containing elements of any type.

It is much faster to test for membership for sets, because for sets, which are always sorted, `'in'` can use a binary search, instead of the linear search used for ordinary lists. So if you have a list for which you want to perform a large number of membership tests you may consider converting it to a set with the function `'Set'` (see "Set").

```
gap> 1 in [ 2, 2, 1, 3 ];
true
gap> 1 in [ 4, -1, 0, 3 ];
false
gap> s := Set([2,4,6,8,10,12,14,16,18,20,22,24,26,28,30,32]);;
gap> 17 in s;
false          # uses binary search and only 4 comparisons
gap> 1 in [ "This", "is", "a", "list", "of", "strings" ];
false
gap> [1,2] in [ [0,6], [0,4], [1,3], [1,5], [1,2], [3,4] ];
true
```

`'Position'` (see "Position") and `'PositionSorted'` (see "PositionSorted") allow you to find the position of an element in a list.

3.14 Position

`'Position(<list>, <elm>)'`

`'Position'` returns the position of the element `<elm>`, which may be an object of any type, in the list `<list>`. If the element is not in the list the result is `'false'`. If the element appears several times, the first position is returned.

It is much faster to search for an element in a set, because for sets, which are always sorted, `'Position'` can use a binary search, instead of the linear search used for ordinary lists. So if you have a list for which you want to perform a large number of searches you may consider converting it to a set with the function `'Set'` (see "Set").

```
gap> Position( [ 2, 2, 1, 3 ], 1 );
3
gap> Position( [ 2, 1, 1, 3 ], 1 );
2
```

```

gap> Position( [ 4, -1, 0, 3 ], 1 );
false
gap> s := Set([2,4,6,8,10,12,14,16,18,20,22,24,26,28,30,32]);
gap> Position( s, 17 );
false      # uses binary search and only 4 comparisons
gap> Position( [ "This", "is", "a", "list", "of", "strings" ], 1 );
false
gap> Position( [ [0,6], [0,4], [1,3], [1,5], [1,2], [3,4] ], [1,2] );
5

```

The 'in' operator (see "In") can be used if you are only interested to know whether the element is in the list or not. 'PositionSorted' (see "PositionSorted") can be used if the list is sorted. 'PositionProperty' (see "PositionProperty") allows you to find the position of an element that satisfies a certain property in a list.

3.15 PositionSorted

```

'PositionSorted( <list>, <elm> )'
'PositionSorted( <list>, <elm>, <func> )'

```

In the first form 'PositionSorted' returns the position of the element <elm>, which may be an object of any type, with respect to the sorted list <list>.

In the second form 'PositionSorted' returns the position of the element <elm>, which may be an object of any type with respect to the list <list>, which must be sorted with respect to <func>. <func> must be a function of two arguments that returns 'true' if the first argument is less than the second argument and 'false' otherwise.

'PositionSorted' returns <pos> such that '<list>[<pos>-1] < <elm>' and '<elm> <= <list>[<pos>]'. That means, if <elm> appears once in <list>, its position is returned. If <elm> appears several times in <list>, the position of the first occurrence is returned. If <elm> is not an element of <list>, the index where <elm> must be inserted to keep the list sorted is returned.

```

gap> PositionSorted( [1,4,5,5,6,7], 0 );
1
gap> PositionSorted( [1,4,5,5,6,7], 2 );
2
gap> PositionSorted( [1,4,5,5,6,7], 4 );
2
gap> PositionSorted( [1,4,5,5,6,7], 5 );
3
gap> PositionSorted( [1,4,5,5,6,7], 8 );
7

```

'Position' (see "Position") is another function that returns the position of an element in a list. 'Position' accepts unsorted lists, uses linear instead of binary search and returns 'false' if <elm> is not in <list>.

3.16 PositionProperty

`'PositionProperty(<list>, <func>)'`

`'PositionProperty'` returns the position of the first element in the list `<list>` for which the unary function `<func>` returns `'true'`. `<list>` must not contain holes. If `<func>` returns `'false'` for all elements of `<list>` `'false'` is returned. `<func>` must return `'true'` or `'false'` for every element of `<list>`, otherwise an error is signalled.

```
gap> PositionProperty( [10^7..10^8], IsPrime );
20
gap> PositionProperty( [10^5..10^6],
>                      n -> not IsPrime(n) and IsPrimePowerInt(n) );
490
```

`'First'` (see "First") allows you to extract the first element of a list that satisfies a certain property.

3.17 Concatenation

`'Concatenation(<list1>, <list2>..)'`

`'Concatenation(<list>)'`

In the first form `'Concatenation'` returns the concatenation of the lists `<list1>`, `<list2>`, etc. The **concatenation** is the list that begins with the elements of `<list1>`, followed by the elements of `<list2>` and so on. Each list may also contain holes, in which case the concatenation also contains holes at the corresponding positions.

```
gap> Concatenation( [ 1, 2, 3 ], [ 4, 5 ] );
[ 1, 2, 3, 4, 5 ]
gap> Concatenation( [2,3,,5,,7], [11,,13,,,17,,19] );
[ 2, 3,, 5,, 7, 11,, 13,,, 17,, 19 ]
```

In the second form `<list>` must be a list of lists `<list1>`, `<list2>`, etc, and `'Concatenation'` returns the concatenation of those lists.

```
gap> Concatenation( [ [1,2,3], [2,3,4], [3,4,5] ] );
[ 1, 2, 3, 2, 3, 4, 3, 4, 5 ]
```

The result is a new list, that is not identical to any other list. The elements of that list however are identical to the corresponding elements of the argument lists (see "Identical Lists").

Note that `'Concatenation'` creates a new list and leaves its arguments unchanged, while `'Append'` (see "Append") changes its first argument. As usual the name of the function that works destructively is a verb, but the name of the function that creates a new object is a substantive.

`'Set(Concatenation(<set1>,<set2>..))'` (see "Set") is a way to compute the union of sets, however, `'Union'` is more efficient.

3.18 Flat

`'Flat(<list>)'`

'Flat' returns the list of all elements that are contained in the list <list> or its sublists. That is, 'Flat' first makes a new empty list <new>. Then it loops over the elements <elm> of <list>. If <elm> is not a list it is added to <new>, otherwise 'Flat' appends 'Flat(<elm>)' to <new>.

```
gap> Flat( [ 1, [ 2, 3 ], [ [ 1, 2 ], 3 ] ] );
[ 1, 2, 3, 1, 2, 3 ]
gap> Flat( [ ] );
[ ]
```

3.19 Reversed

`'Reversed(<list>)'`

'Reversed' returns a new list that contains the elements of the list <list>, which must not contain holes, in reverse order. The argument list is unchanged.

```
gap> Reversed( [ 1, 4, 5, 5, 6, 7 ] );
[ 7, 6, 5, 5, 4, 1 ]
```

The result is a new list, that is not identical to any other list. The elements of that list however are identical to the corresponding elements of the argument list (see "Identical Lists").

3.20 Sublist

`'Sublist(<list>, <inds>)'`

'Sublist' returns a new list in which the <i>-th element is the element '`<list>[<inds>[<i>]]`', of the list <list>. <inds> must be a list of positive integers without holes, it need, however, not be sorted and may contains duplicate elements. If '`<list>[<inds>[<i>]]`' is unbound for an <i>, an error is signalled.

```
gap> Sublist( [ 2, 3, 5, 7, 11, 13, 17, 19 ], [4..6] );
[ 7, 11, 13 ]
gap> Sublist( [ 2, 3, 5, 7, 11, 13, 17, 19 ], [1,7,1,8] );
[ 2, 17, 2, 19 ]
```

The result is a new list, that is not identical to any other list. The elements of that list however are identical to the corresponding elements of the argument list (see "Identical Lists").

'Filtered' (see "Filtered") allows you to extract elements from a list according to a predicate.

'Sublist' has been made obsolete by the introduction of the construct '`<list>\{ <inds> \}`' (see "List Elements").

3.21 Cartesian

```
'Cartesian( <list1>, <list2>.. )'
```

```
'Cartesian( <list> )'
```

In the first form **'Cartesian'** returns the cartesian product of the lists **<list1>**, **<list2>**, etc.

In the second form **<list>** must be a list of lists **<list1>**, **<list2>**, etc., and **'Cartesian'** returns the cartesian product of those lists.

The *cartesian product* is a list **<cart>** of lists **<tup>**, such that the first element of **<tup>** is an element of **<list1>**, the second element of **<tup>** is an element of **<list2>**, and so on. The total number of elements in **<cart>** is the product of the lengths of the argument lists. In particular **<cart>** is empty if and only if at least one of the argument lists is empty. Also **<cart>** contains duplicates if and only if no argument list is empty and at least one contains duplicates.

The last index runs fastest. That means that the first element **<tup1>** of **<cart>** contains the first element from **<list1>**, from **<list2>** and so on. The second element **<tup2>** of **<cart>** contains the first element from **<list1>**, the first from **<list2>**, and so on, but the last element of **<tup2>** is the second element of the last argument list. This implies that **<cart>** is a set if and only if all argument lists are sets.

```
gap> Cartesian( [1,2], [3,4], [5,6] );
[ [ 1, 3, 5 ], [ 1, 3, 6 ], [ 1, 4, 5 ], [ 1, 4, 6 ], [ 2, 3, 5 ],
  [ 2, 3, 6 ], [ 2, 4, 5 ], [ 2, 4, 6 ] ]
gap> Cartesian( [1,2,2], [1,1,2] );
[ [ 1, 1 ], [ 1, 1 ], [ 1, 2 ], [ 2, 1 ], [ 2, 1 ], [ 2, 2 ],
  [ 2, 1 ], [ 2, 1 ], [ 2, 2 ] ]
```

The function **'Tuples'** computes the **<k>**-fold cartesian product of a list.

3.22 Number

```
'Number( <list> )'
```

```
'Number( <list>, <func> )'
```

In the first form **'Number'** returns the number of bound entries in the list **<list>**.

For lists that contain no holes **'Number'**, **'Length'** (see "Length"), and **'Size'** return the same value. For lists with holes **'Number'** returns the number of bound entries, **'Length'** returns the largest index of a bound entry, and **'Size'** signals an error.

'Number' returns the number of elements of the list **<list>** for which the unary function **<func>** returns **'true'**. If an element for which **<func>** returns **'true'** appears several times in **<list>** it will also be counted several times. **<func>** must return either **'true'** or **'false'** for every element of **<list>**, otherwise an error is signalled.

```

gap> Number( [ 2, 3, 5, 7 ] );
4
gap> Number( [, 2, 3,, 5,, 7,,, 11 ] );
5
gap> Number( [1..20], IsPrime );
8
gap> Number( [ 1, 3, 4, -4, 4, 7, 10, 6 ], IsPrimePowerInt );
4
gap> Number( [ 1, 3, 4, -4, 4, 7, 10, 6 ],
>           n -> IsPrimePowerInt(n) and n mod 2 <> 0 );
2

```

'Filtered' (see "Filtered") allows you to extract the elements of a list that have a certain property.

3.23 Collected

'Collected(<list>)'

'Collected' returns a new list <new> that contains for each different element <elm> of <list> a list of two elements, the first element is <elm> itself, and the second element is the number of times <elm> appears in <list>. The order of those pairs in <new> corresponds to the ordering of the elements <elm>, so that the result is sorted.

```

gap> Factors( Factorial( 10 ) );
[ 2, 2, 2, 2, 2, 2, 2, 2, 3, 3, 3, 3, 5, 5, 7 ]
gap> Collected( last );
[ [ 2, 8 ], [ 3, 4 ], [ 5, 2 ], [ 7, 1 ] ]
gap> Collected( last );
[ [ [ 2, 8 ], 1 ], [ [ 3, 4 ], 1 ], [ [ 5, 2 ], 1 ], [ [ 7, 1 ], 1 ] ]

```

3.24 Filtered

'Filtered(<list>, <func>)'

'Filtered' returns a new list that contains those elements of the list <list> for which the unary function <func> returns 'true'. The order of the elements in the result is the same as the order of the corresponding elements of <list>. If an element, for which <func> returns 'true' appears several times in <list> it will also appear the same number of times in the result. <list> may contain holes, they are ignored by 'Filtered'. <func> must return either 'true' or 'false' for every element of <list>, otherwise an error is signalled.

```

gap> Filtered( [1..20], IsPrime );
[ 2, 3, 5, 7, 11, 13, 17, 19 ]
gap> Filtered( [ 1, 3, 4, -4, 4, 7, 10, 6 ], IsPrimePowerInt );

```

```
[ 3, 4, 4, 7 ]
gap> Filtered( [ 1, 3, 4, -4, 4, 7, 10, 6 ],
>             n -> IsPrimePowerInt(n) and n mod 2 <> 0 );
[ 3, 7 ]
```

The result is a new list, that is not identical to any other list. The elements of that list however are identical to the corresponding elements of the argument list (see "Identical Lists").

'Sublist' (see "Sublist") allows you to extract elements of a list according to indices given in another list.

3.25 ForAll

'ForAll(<list>, <func>)'

'ForAll' returns 'true' if the unary function <func> returns 'true' for all elements of the list <list> and 'false' otherwise. <list> may contain holes. <func> must return either 'true' or 'false' for every element of <list>, otherwise an error is signalled.

```
gap> ForAll( [1..20], IsPrime );
false
gap> ForAll( [2,3,4,5,8,9], IsPrimePowerInt );
true
gap> ForAll( [2..14], n -> IsPrimePowerInt(n) or n mod 2 = 0 );
true
```

'ForAny' (see "ForAny") allows you to test if any element of a list satisfies a certain property.

3.26 ForAny

'ForAny(<list>, <func>)'

'ForAny' returns 'true' if the unary function <func> returns 'true' for at least one element of the list <list> and 'false' otherwise. <list> may contain holes. <func> must return either 'true' or 'false' for every element of <list>, otherwise 'ForAny' signals an error.

```
gap> ForAny( [1..20], IsPrime );
true
gap> ForAny( [2,3,4,5,8,9], IsPrimePowerInt );
true
gap> ForAny( [2..14],
>   n -> IsPrimePowerInt(n) and n mod 5 = 0 and not IsPrime(n) );
false
```

'ForAll' (see "ForAll") allows you to test if all elements of a list satisfies a certain propertie.

3.27 First

`'First(<list>, <func>)'`

'First' returns the first element of the list <list> for which the unary function <func> returns 'true'. <list> may contain holes. <func> must return either 'true' or 'false' for every element of <list>, otherwise an error is signalled. If <func> returns 'false' for every element of <list> an error is signalled.

```
gap> First( [10^7..10^8], IsPrime );
10000019
gap> First( [10^5..10^6],
>          n -> not IsPrime(n) and IsPrimePowerInt(n) );
100489
```

'PositionProperty' (see "PositionProperty") allows you to find the position of the first element in a list that satisfies a certain property.

3.28 Sort

`'Sort(<list>)'`

`'Sort(<list>, <func>)'`

'Sort' sorts the list <list> in increasing order, using shellsort. In the first form 'Sort' uses the operator '<' to compare the elements. In the second form 'Sort' uses the function <func> to compare elements. This function must be a function taking two arguments that returns 'true' if the first is strictly smaller than the second and 'false' otherwise.

'Sort' does not return anything, since it changes the argument <list>. Use 'ShallowCopy' (see "ShallowCopy") if you want to keep <list>. Use 'Reversed' (see "Reversed") if you want to get a new list sorted in decreasing order.

It is possible to sort lists that contain multiple elements which compare equal. It is not guaranteed that those elements keep their relative order, i.e., 'Sort' is not stable.

```
gap> list := [ 5, 4, 6, 1, 7, 5 ];; Sort( list ); list;
[ 1, 4, 5, 5, 6, 7 ]
gap> list := [ [0,6], [1,2], [1,3], [1,5], [0,4], [3,4] ];;
gap> Sort( list, function(v,w) return v*v < w*w; end ); list;
[ [ 1, 2 ], [ 1, 3 ], [ 0, 4 ], [ 3, 4 ], [ 1, 5 ], [ 0, 6 ] ]
# sorted according to the Euclidian distance from [0,0]
gap> list := [ [0,6], [1,3], [3,4], [1,5], [1,2], [0,4], ];;
gap> Sort( list, function(v,w) return v[1] < w[1]; end ); list;
[ [ 0, 6 ], [ 0, 4 ], [ 1, 3 ], [ 1, 5 ], [ 1, 2 ], [ 3, 4 ] ]
# note the random order of the elements with equal first component
```


'SortParallel' (see "SortParallel") allows you to sort a list and apply the exchanges that are necessary to another list in parallel. 'Sortex' (see "Sortex") sorts a list and returns the sorting permutation.

3.29 SortParallel

```
'SortParallel( <list1>, <list2> )'
'SortParallel( <list1>, <list2>, <func> )'
```

'SortParallel' sorts the list <list1> in increasing order just as 'Sort' (see "Sort") does. In parallel it applies the same exchanges that are necessary to sort <list1> to the list <list2>, which must of course have at least as many elements as <list1> does.

```
gap> list1 := [ 5, 4, 6, 1, 7, 5 ];;
gap> list2 := [ 2, 3, 5, 7, 8, 9 ];;
gap> SortParallel( list1, list2 );
gap> list1;
[ 1, 4, 5, 5, 6, 7 ]
gap> list2;
[ 7, 3, 2, 9, 5, 8 ]    # \verb|'[ 7, 3, 9, 2, 5, 8 ]'| is also possible
```

'Sortex' (see "Sortex") sorts a list and returns the sorting permutation.

3.30 Sortex

```
'Sortex( <list> )'
```

'Sortex' sorts the list <list> and returns the permutation that must be applied to <list> to obtain the sorted list.

```
gap> list1 := [ 5, 4, 6, 1, 7, 5 ];;
gap> list2 := Copy( list1 );
gap> perm := Sortex( list1 );
(1,3,5,6,4)
gap> list1;
[ 1, 4, 5, 5, 6, 7 ]
gap> Permuted( list2, perm );
[ 1, 4, 5, 5, 6, 7 ]
```

'Permuted' (see "Permuted") allows you to rearrange a list according to a given permutation.

3.31 Permuted

`'Permuted(<list>, <perm>)'`

`'Permuted'` returns a new list `<new>` that contains the elements of the list `<list>` permuted according to the permutation `<perm>`. That is `'<new>[<i>\^<perm>] = <list>[<i>]'`.

```
gap> Permuted( [ 5, 4, 6, 1, 7, 5 ], (1,3,5,6,4) );
[ 1, 4, 5, 5, 6, 7 ]
```

`'Sortex'` (see "Sortex") allows you to compute the permutation that must be applied to a list to get the sorted list.

3.32 Product

`'Product(<list>)'`

`'Product(<list>, <func>)'`

In the first form `'Product'` returns the product of the elements of the list `<list>`, which must have no holes. If `<list>` is empty, the integer 1 is returned.

In the second form `'Product'` applies the function `<func>` to each element of the list `<list>`, which must have no holes, and multiplies the results. If the `<list>` is empty, the integer 1 is returned.

```
gap> Product( [ 2, 3, 5, 7, 11, 13, 17, 19 ] );
9699690
gap> Product( [1..10], x->x^2 );
13168189440000
gap> Product( [ (1,2), (1,3), (1,4), (2,3), (2,4), (3,4) ] );
(1,4)(2,3)
```

`'Sum'` (see "Sum") computes the sum of the elements of a list.

3.33 Sum

`'Sum(<list>)'`

`'Sum(<list>, <func>)'`

In the first form `'Sum'` returns the sum of the elements of the list `<list>`, which must have no holes. If `<list>` is empty 0 is returned.

In the second form `'Sum'` applies the function `<func>` to each element of the list `<list>`, which must have no holes, and sums the results. If the `<list>` is empty 0 is returned.

```
gap> Sum( [ 2, 3, 5, 7, 11, 13, 17, 19 ] );
77
gap> Sum( [1..10], x->x^2 );
385
gap> Sum( [ [1,2], [3,4], [5,6] ] );
[ 9, 12 ]
```

'Product' (see "Product") computes the product of the elements of a list.

3.34 Maximum

```
'Maximum( <obj1>, <obj2>.. )'
'Maximum( <list> )'
```

'Maximum' returns the maximum of its arguments, i.e., that argument obj_i for which $obj_k \leq obj_i$ for all k . In its second form 'Maximum' takes a list <list> and returns the maximum of the elements of this list.

Typically the arguments or elements of the list respectively will be integers, but actually they can be objects of an arbitrary type. This works because any two objects can be compared using the '<' operator.

```
gap> Maximum( -123, 700, 123, 0, -1000 );
700
gap> Maximum( [ -123, 700, 123, 0, -1000 ] );
700
gap> Maximum( [ 1, 2 ], [ 0, 15 ], [ 1, 5 ], [ 2, -11 ] );
[ 2, -11 ]      # lists are compared elementwise
```

3.35 Minimum

```
'Minimum( <obj1>, <obj2>.. )'
'Minimum( <list> )'
```

'Minimum' returns the minimum of its arguments, i.e., that argument obj_i for which $obj_i \leq obj_k$ for all k . In its second form 'Minimum' takes a list <list> and returns the minimum of the elements of this list.

Typically the arguments or elements of the list respectively will be integers, but actually they can be objects of an arbitrary type. This works because any two objects can be compared using the '<' operator.

```
gap> Minimum( -123, 700, 123, 0, -1000 );
-1000
gap> Minimum( [ -123, 700, 123, 0, -1000 ] );
```

```
-1000
gap> Minimum( [ 1, 2 ], [ 0, 15 ], [ 1, 5 ], [ 2, -11 ] );
[ 0, 15 ]      # lists are compared elementwise
```

3.36 Iterated

'Iterated(<list>, <f>)'

'Iterated' returns the result of the iterated application of the function <f>, which must take two arguments, to the elements of <list>. More precisely 'Iterated' returns the result of the following application, '<f>(..<f>(<f>(<list>[1], <list>[2]), <list>[3]),...,<list>[<n>])'.

```
gap> Iterated( [ 126, 66, 105 ], Gcd );
3
```

3.37 RandomList

'RandomList(<list>)'

'RandomList' returns a random element of the list <list>. The results are equally distributed, i.e., all elements are equally likely to be selected.

```
gap> RandomList( [1..200] );
192
gap> RandomList( [1..200] );
152
gap> RandomList( [ [ 1, 2 ], 3, [ 4, 5 ], 6 ] );
[ 4, 5 ]
```

'RandomSeed(<n>)'

'RandomSeed' seeds the pseudo random number generator 'RandomList'. Thus to reproduce a computation exactly you can call 'RandomSeed' each time before you start the computation. When GAP is started the pseudo random number generator is seeded with 1.

```
gap> RandomSeed(1); RandomList([1..100]); RandomList([1..100]);
96
76
gap> RandomSeed(1); RandomList([1..100]); RandomList([1..100]);
96
76
```

'RandomList' is called by all random functions for domains.

Chapter 4

Sets

A very important mathematical concept, maybe the most important of all, are sets. Mathematically a **set** is an abstract object such that each object is either an element of the set or it is not. So a set is a collection like a list, and in fact **GAP** uses lists to represent sets. Note that this of course implies that **GAP** only deals with finite sets.

Unlike a list a set must not contain an element several times. It simply makes no sense to say that an object is twice an element of a set, because an object is either an element of a set, or it is not. Therefore the list that is used to represent a set has no duplicates, that is, no two elements of such a list are equal.

Also unlike a list a set does not impose any ordering on the elements. Again it simply makes no sense to say that an object is the first or second etc. element of a set, because, again, an object is either an element of a set, or it is not. Since ordering is not defined for a set we can put the elements in any order into the list used to represent the set. We put the elements sorted into the list, because this ordering is very practical. For example if we convert a list into a set we have to remove duplicates, which is very easy to do after we have sorted the list, since then equal elements will be next to each other.

In short sets are represented by sorted lists without holes and duplicates in **GAP**. Such a list is in this document called a proper set. Note that we guarantee this representation, so you may make use of the fact that a set is represented by a sorted list in your functions.

In some contexts, we also want to talk about multisets. A **multiset** is like a set, except that an element may appear several times in a multiset. Such multisets are represented by sorted lists with holes that may have duplicates.

The first section in this chapter describes the functions to test if an object is a set and to convert objects to sets (see `"IsSet"` and `"Set"`).

The next section describes the function that tests if two sets are equal (see `"SetIsEqual"`).

The next sections describe the destructive functions that compute the standard set operations for sets (see `"SetAdd"`, `"SetRemove"`, `"SetUnite"`, `"SetIntersect"`, and `"SetSubtract"`).

The last section tells you more about sets and their internal representation (see `"More about Sets"`).

All set theoretic functions, especially 'Intersection' and 'Union', also accept sets as arguments. Thus all functions described in the chapter Domains in the GAP-manual are applicable to sets (see "Set Functions for Sets").

Since sets are just a special case of lists, all the operations and functions for lists, especially the membership test (see "In"), can be used for sets just as well.

4.1 IsSet

'IsSet(<obj>)'

'IsSet' returns 'true' if the object <obj> is a set and 'false' otherwise. An object is a set if it is a sorted list without holes or duplicates. Will cause an error if evaluation of <obj> is an unbound variable.

```
gap> IsSet( [] );
true
gap> IsSet( [ 2, 3, 5, 7, 11 ] );
true
gap> IsSet( [, 2, 3,, 5,, 7,,, 11 ] );
false          # this list contains holes
gap> IsSet( [ 11, 7, 5, 3, 2 ] );
false          # this list is not sorted
gap> IsSet( [ 2, 2, 3, 5, 5, 7, 11, 11 ] );
false          # this list contains duplicates
gap> IsSet( 235711 );
false          # this argument is not even a list
```

4.2 Set

'Set(<list>)'

'Set' returns a new proper set, which is represented as a sorted list without holes or duplicates, containing the elements of the list <list>.

'Set' returns a new list even if the list <list> is already a proper set, in this case it is equivalent to 'ShallowCopy' (see "ShallowCopy"). Thus the result is a new list that is not identical to any other list. The elements of the result are however identical to elements of <list>. If <list> contains equal elements, it is not specified to which of those the element of the result is identical (see "Identical Lists").

```
gap> Set( [3,2,11,7,2,,5] );
[ 2, 3, 5, 7, 11 ]
gap> Set( [] );
[ ]
```

4.3 SetIsEqual

'SetIsEqual(<list1>, <list2>)'

'SetIsEqual' returns 'true' if the two lists <list1> and <list2> are equal *when viewed as sets*, and 'false' otherwise. <list1> and <list2> are equal if every element of <list1> is also an element of <list2> and if every element of <list2> is also an element of <list1>.

If both lists are proper sets then they are of course equal if and only if they are also equal as lists. Thus 'SetIsEqual(<list1>, <list2>)' is equivalent to 'Set(<list1>) = Set(<list2>)' (see "Set"), but the former is more efficient.

```
gap> SetIsEqual( [2,3,5,7,11], [11,7,5,3,2] );
true
gap> SetIsEqual( [2,3,5,7,11], [2,3,5,7,11,13] );
false
```

4.4 SetAdd

'SetAdd(<set>, <elm>)'

'SetAdd' adds <elm>, which may be an element of an arbitrary type, to the set <set>, which must be a proper set, otherwise an error will be signalled. If <elm> is already an element of the set <set>, the set is not changed. Otherwise <elm> is inserted at the correct position such that <set> is again a set afterwards.

```
gap> s := [2,3,7,11];;
gap> SetAdd( s, 5 ); s;
[ 2, 3, 5, 7, 11 ]
gap> SetAdd( s, 13 ); s;
[ 2, 3, 5, 7, 11, 13 ]
gap> SetAdd( s, 3 ); s;
[ 2, 3, 5, 7, 11, 13 ]
```

'SetRemove' (see "SetRemove") is the counterpart of 'SetAdd'.

4.5 SetRemove

'SetRemove(<set>, <elm>)'

'SetRemove' removes the element <elm>, which may be an object of arbitrary type, from the set <set>, which must be a set, otherwise an error will be signalled. If <elm> is not an element of <set> nothing happens. If <elm> is an element it is removed and all the following elements in the list are moved one position forward.

```
gap> s := [ 2, 3, 4, 5, 6, 7 ];;
gap> SetRemove( s, 6 );
gap> s;
[ 2, 3, 4, 5, 7 ]
gap> SetRemove( s, 10 );
gap> s;
[ 2, 3, 4, 5, 7 ]
```

'SetAdd' (see "SetAdd") is the counterpart of 'SetRemove'.

4.6 SetUnite

'SetUnite(<set1>, <set2>)'

'SetUnite' unites the set <set1> with the set <set2>. This is equivalent to adding all the elements in <set2> to <set1> (see "SetAdd"). <set1> must be a proper set, otherwise an error is signalled. <set2> may also be list that is not a proper set, in which case 'SetUnite' silently applies 'Set' to it first (see "Set"). 'SetUnite' returns nothing, it is only called to change <set1>.

```
gap> set := [ 2, 3, 5, 7, 11 ];;
gap> SetUnite( set, [ 4, 8, 9 ] ); set;
[ 2, 3, 4, 5, 7, 8, 9, 11 ]
gap> SetUnite( set, [ 16, 9, 25, 13, 16 ] ); set;
[ 2, 3, 4, 5, 7, 8, 9, 11, 13, 16, 25 ]
```

The function 'UnionSet' (see "Set Functions for Sets") is the nondestructive counterpart to the destructive procedure 'SetUnite'.

4.7 SetIntersect

'SetIntersect(<set1>, <set2>)'

'SetIntersect' intersects the set <set1> with the set <set2>. This is equivalent to removing all the elements that are not in <set2> from <set1> (see "SetRemove"). <set1> must be a set, otherwise an error is signalled. <set2> may be a list that is not a proper set, in which case 'SetIntersect' silently applies 'Set' to it first (see "Set"). 'SetIntersect' returns nothing, it is only called to change <set1>.

```
gap> set := [ 2, 3, 4, 5, 7, 8, 9, 11, 13, 16 ];;
gap> SetIntersect( set, [ 3, 5, 7, 9, 11, 13, 15, 17 ] ); set;
[ 3, 5, 7, 9, 11, 13 ]
gap> SetIntersect( set, [ 9, 4, 6, 8 ] ); set;
[ 9 ]
```

The function 'IntersectionSet' (see "Set Functions for Sets") is the nondestructive counterpart to the destructive procedure 'SetIntersect'.

4.8 SetSubtract

`'SetSubtract(<set1>, <set2>)'`

`'SetSubtract'` subtracts the set `<set2>` from the set `<set1>`. This is equivalent to removing all the elements in `<set2>` from `<set1>` (see `"SetRemove"`). `<set1>` must be a proper set, otherwise an error is signalled. `<set2>` may be a list that is not a proper set, in which case `'SetSubtract'` applies `'Set'` to it first (see `"Set"`). `'SetSubtract'` returns nothing, it is only called to change `<set1>`.

```
gap> set := [ 2, 3, 4, 5, 6, 7, 8, 9, 10, 11 ];;
gap> SetSubtract( set, [ 6, 10 ] ); set;
[ 2, 3, 4, 5, 7, 8, 9, 11 ]
gap> SetSubtract( set, [ 9, 4, 6, 8 ] ); set;
[ 2, 3, 5, 7, 11 ]
```

The function `'Difference'` is the nondestructive counterpart to destructive the procedure `'SetSubtract'`.

4.9 Set Functions for Sets

As was already mentioned in the introduction to this chapter all domain functions also accept sets as arguments. Thus all functions described in the chapter Domains in the GAP-manual are applicable to sets. This section describes those functions where it might be helpful to know the implementation of those functions for sets.

`'IsSubset(<set1>, <set2>)'`

This is implemented by `'SetIsSubset'`, which you can call directly to save a little bit of time. Either argument to `'SetIsSubset'` may also be a list that is not a proper set, in which case `'IsSubset'` silently applies `'Set'` (see `"Set"`) to it first.

`'Union(<set1>, <set2>)'`

This is implemented by `'UnionSet'`, which you can call directly to save a little bit of time. Note that `'UnionSet'` only accepts two sets, unlike `'Union'`, which accepts several sets or a list of sets. The result of `'UnionSet'` is a new set, represented as a sorted list without holes or duplicates. Each argument to `'UnionSet'` may also be a list that is not a proper set, in which case `'UnionSet'` silently applies `'Set'` (see `"Set"`) to this argument. `'UnionSet'` is implemented in terms of its destructive counterpart `'SetUnite'` (see `"SetUnite"`).

`'Intersection(<set1>, <set2>)'`

This is implemented by `'IntersectionSet'`, which you can call directly to save a little bit of time. Note that `'IntersectionSet'` only accepts two sets, unlike `'Intersection'`, which accepts several sets or a list of sets. The result of `'IntersectionSet'` is a new set, represented as a sorted list without holes or duplicates. Each argument to `'IntersectionSet'` may also be a list that is not a proper set, in which case `'IntersectionSet'` silently applies `'Set'` (see `"Set"`) to this argument. `'IntersectionSet'` is implemented in terms of its destructive counterpart `'SetIntersect'` (see `"SetIntersect"`).

The result of 'IntersectionSet' and 'UnionSet' is always a new list, that is not identical to any other list. The elements of that list however are identical to the corresponding elements of <set1>. If <set1> is not a proper list it is not specified to which of a number of equal elements in <set1> the element in the result is identical (see "Identical Lists").

4.10 More about Sets

In the previous section we defined a proper set as a sorted list without holes or duplicates. This representation is not only nice to use, it is also a good internal representation supporting efficient algorithms. For example the 'in' operator can use binary instead of a linear search since a set is sorted. For another example 'Union' only has to merge the sets.

However, all those set functions also allow lists that are not proper sets, silently making a copy of it and converting this copy to a set. Suppose all the functions would have to test their arguments every time, comparing each element with its successor, to see if they are proper sets. This would chew up most of the performance advantage again. For example suppose 'in' would have to run over the whole list, to see if it is a proper set, so it could use the binary search. That would be ridiculous.

To avoid this a list that is a proper set may, but need not, have an internal flag set that tells those functions that this list is indeed a proper set. Those functions do not have to check this argument then, and can use the more efficient algorithms. This section tells you when a proper set obtains this flag, so you can write your functions in such a way that you make best use of the algorithms.

The results of 'Set', 'Difference', 'Intersection' and 'Union' are known to be sets by construction, and thus have the flag set upon creation.

If an argument to 'IsSet', 'SetIsEqual', 'IsSubset', 'Set', 'Difference', 'Intersection' or 'Union' is a proper set, that does not yet have the flag set, those functions will notice that and set the flag for this set. Note that 'in' will use linear search if the right operand does not have the flag set, will therefore not detect if it is a proper set and will, unlike the functions above, never set the flag.

If you change a proper set, that does have this flag set, by assignment, 'Add' or 'Append' the set will generally lose it flag, even if the change is such that the resulting list is still a proper set. However if the set has more than 100 elements and the value assigned or added is not a list and not a record and the resulting list is still a proper set than it will keep the flag. Note that changing a list that is not a proper set will never set the flag, even if the resulting list is a proper set. Such a set will obtain the flag only if it is passed to a set function.

Suppose you have built a proper set in such a way that it does not have the flag set, and that you now want to perform lots of membership tests. Then you should call 'IsSet' with that set as an argument. If it is indeed a proper set 'IsSet' will set the flag, and the subsequent 'in' operations will use the more efficient binary search. You can think of the call to 'IsSet' as a hint to GAP that this list is a proper set.

There is no way you can set the flag for an ordinary list without going through the checking in 'IsSet'. The internal functions depend so much on the fact that a list with this flag set is indeed sorted and without holes and duplicates that the risk would be too high to allow setting the flag without such a check.

Chapter 5

Records

Records are next to lists the most important way to collect objects together. A record is a collection of **components**. Each component has a unique **name**, which is an identifier that distinguishes this component, and a **value**, which is an object of arbitrary type. We often abbreviate **value of a component** to **element**. We also say that a record **contains** its elements. You can access and change the elements of a record using its name.

Record literals are written by writing down the components in order between `'rec('` and `')`, and separating them by commas `','`. Each component consists of the name, the assignment operator `'\:='`, and the value. The **empty record**, i.e., the record with no components, is written as `'rec()'`.

```
gap> rec( a := 1, b := "2" );      # a record with two components
rec(
  a := 1,
  b := "2" )
gap> rec( a := 1, b := rec( c := 2 ) );    # record may contain records
rec(
  a := 1,
  b := rec(
    c := 2 ) )
```

Records usually contain elements of various types, i.e., they are usually not homogeneous like lists.

The first section in this chapter tells you how you can access the elements of a record (see "Accessing Record Elements").

The next sections tell you how you can change the elements of a record (see "Record Assignment" and "Identical Records").

The next sections describe the operations that are available for records (see "Comparisons of Records", "Operations for Records", "In for Records", and "Printing of Records").

The next section describes the function that tests if an object is a record (see "IsRec").

The next sections describe the functions that test whether a record has a component with a given name, and delete such a component (see "IsBound" and "Unbind"). Those functions are also applicable to lists.

The final sections describe the functions that create a copy of a record (see "Copy" and "Shallow-Copy"). Again those functions are also applicable to lists.

5.1 Accessing Record Elements

`'<rec>.<name>'`

The above construct evaluates to the value of the record component with the name `<name>` in the record `<rec>`. Note that the `<name>` is not evaluated, i.e., it is taken literal.

```
gap> r := rec( a := 1, b := 2 );;
gap> r.a;
1
gap> r.b;
2
```

`'<rec>.(<name>)'`

This construct is similar to the above construct. The difference is that the second operand `<name>` is evaluated. It must evaluate to a string or an integer otherwise an error is signalled. The construct then evaluates to the element of the record `<rec>` whose name is, as a string, equal to `<name>`.

```
gap> old := rec( a := 1, b := 2 );;
gap> new := rec();
rec(
  )
gap> for i in RecFields( old ) do
>   new.(i) := old.(i);
> od;
gap> new;
rec(
  a := 1,
  b := 2 )
```

If `<rec>` does not evaluate to a record, or if `<name>` does not evaluate to a string, or if `'<rec>.<name>'` is unbound, an error is signalled. As usual you can leave the break loop with `'quit;'`. On the other hand you can return a result to be used in place of the record element by `'return <expr>;'`.

5.2 Record Assignment

`'<rec>.<name> \:= <obj>;'`

The record assignment assigns the object `<obj>`, which may be an object of arbitrary type, to the record component with the name `<name>`, which must be an identifier, of the record `<rec>`. That

means that accessing the element with name `<name>` of the record `<rec>` will return `<obj>` after this assignment. If the record `<rec>` has no component with the name `<name>`, the record is automatically extended to make room for the new component.

```
gap> r := rec( a := 1, b := 2 );;
gap> r.a := 10;; r;
rec(
  a := 10,
  b := 2 )
gap> r.c := 3;; r;
rec(
  a := 10,
  b := 2,
  c := 3 )
```

The function `'IsBound'` (see "IsBound") can be used to test if a record has a component with a certain name, the function `'Unbind'` (see "Unbind") can be used to remove a component with a certain name again.

Note that assigning to a record changes the record. The ability to change an object is only available for lists and records (see "Identical Records").

```
'<rec>.<name> := <obj>;'
```

This construct is similar to the above construct. The difference is that the second operand `<name>` is evaluated. It must evaluate to a string or an integer otherwise an error is signalled. The construct then assigns `<obj>` to the record component of the record `<rec>` whose name is, as a string, equal to `<name>`.

If `<rec>` does not evaluate to a record, `<name>` does not evaluate to a string, or `<obj>` is a call to a function that does not return a value, e.g., `'Print'`, an error is signalled. As usual you can leave the break loop with `'quit;'`. On the other hand you can continue the assignment by returning a record in the first case, a string in the second, or an object to be assigned in the third, using `'return <expr>;'`.

5.3 Identical Records

With the record assignment (see "Record Assignment") it is possible to change a record. The ability to change an object is only available for lists and records. This section describes the semantic consequences of this fact.

You may think that in the following example the second assignment changes the integer, and that therefore the above sentence, which claimed that only records and lists can be changed, is wrong.

```
i := 3;
i := i + 1;
```

But in this example not the *integer* '3' is changed by adding one to it. Instead the *variable* 'i' is changed by assigning the value of 'i+1', which happens to be '4', to 'i'. The same thing happens in the following example

```
r := rec( a := 1 );  
r := rec( a := 1, b := 2 );
```

The second assignment does not change the first record, instead it assigns a new record to the variable 'r'. On the other hand, in the following example the record is changed by the second assignment.

```
r := rec( a := 1 );  
r.b := 2;
```

To understand the difference first think of a variable as a name for an object. The important point is that a record can have several names at the same time. An assignment '`<var> \:= <record>;`' means in this interpretation that `<var>` is a name for the object `<record>`. At the end of the following example 'r2' still has the value '`rec(a \:= 1)`' as this record has not been changed and nothing else has been assigned to 'r2'.

```
r1 := rec( a := 1 );  
r2 := r1;  
r1 := rec( a := 1, b := 2 );
```

But after the following example the record for which 'r2' is a name has been changed and thus the value of 'r2' is now '`rec(a \:= 1, b \:= 2)`'.

```
r1 := rec( a := 1 );  
r2 := r1;  
r1.b := 2;
```

We shall say that two records are *identical* if changing one of them by a record assignment also changes the other one. This is slightly incorrect, because if *two* records are identical, there are actually only two names for *one* record. However, the correct usage would be very awkward and would only add to the confusion. Note that two identical records must be equal, because there is only one records with two different names. Thus identity is an equivalence relation that is a refinement of equality.

Let us now consider under which circumstances two records are identical.

If you enter a record literal then the record denoted by this literal is a new record that is not identical to any other record. Thus in the following example 'r1' and 'r2' are not identical, though they are equal of course.

```
r1 := rec( a := 1 );  
r2 := rec( a := 1 );
```

Also in the following example, no records in the list 'l' are identical.

```
l := [];  
for i in [1..10] do  
  l[i] := rec( a := 1 );  
od;
```

If you assign a record to a variable no new record is created. Thus the record value of the variable on the left hand side and the record on the right hand side of the assignment are identical. So in the following example 'r1' and 'r2' are identical records.

```
r1 := rec( a := 1 );  
r2 := r1;
```

If you pass a record as argument, the old record and the argument of the function are identical. Also if you return a record from a function, the old record and the value of the function call are identical. So in the following example 'r1' and 'r2' are identical record

```
r1 := rec( a := 1 );  
f := function ( r ) return r; end;  
r2 := f( r1 );
```

The functions 'Copy' and 'ShallowCopy' (see "Copy" and "ShallowCopy") accept a record and return a new record that is equal to the old record but that is **not** identical to the old record. The difference between 'Copy' and 'ShallowCopy' is that in the case of 'ShallowCopy' the corresponding elements of the new and the old records will be identical, whereas in the case of 'Copy' they will only be equal. So in the following example 'r1' and 'r2' are not identical records.

```
r1 := rec( a := 1 );  
r2 := Copy( r1 );
```

If you change a record it keeps its identity. Thus if two records are identical and you change one of them, you also change the other, and they are still identical afterwards. On the other hand, two records that are not identical will never become identical if you change one of them. So in the following example both 'r1' and 'r2' are changed, and are still identical.

```
r1 := rec( a := 1 );  
r2 := r1;  
r1.b := 2;
```

5.4 Comparisons of Records

```
'<rec1> = <rec2>'
'<rec1> <> <rec2>'
```

The equality operator '=' returns 'true' if the record <rec1> is equal to the record <rec2> and 'false' otherwise. The inequality operator '<>' returns 'true' if the record <rec1> is not equal to <rec2> and 'false' otherwise.

Usually two records are considered equal, if for each component of one record the other record has a component of the same name with an equal value and vice versa. You can also compare records with other objects, they are of course different, unless the record has a special comparison function (see below) that says otherwise.

```
gap> rec( a := 1, b := 2 ) = rec( b := 2, a := 1 );
true
gap> rec( a := 1, b := 2 ) = rec( a := 2, b := 1 );
false
gap> rec( a := 1 ) = rec( a := 1, b := 2 );
false
gap> rec( a := 1 ) = 1;
false
```

However a record may contain a special 'operations' record that contains a function that is called when this record is an operand of a comparison. The precise mechanism is as follows. If the operand of the equality operator '=' is a record, and if this record has an element with the name 'operations' that is a record, and if this record has an element with the name '=' that is a function, then this function is called with the operands of '=' as arguments, and the value of the operation is the result returned by this function. In this case a record may also be equal to an object of another type if this function says so. It is probably not a good idea to define a comparison function in such a way that the resulting relation is not an equivalence relation, i.e., not reflexive, symmetric, and transitive. Note that there is no corresponding '<>' function, because '<left> <> <right>' is implemented as 'not <left> = <right>'.

The following example shows one piece of the definition of residue classes, using record operations. Of course this is far from a complete implementation. Note that the '=' must be quoted, so that it is taken as an identifier (see "Identifiers").

```
gap> ResidueClassOps := rec( );
gap> ResidueClassOps.\= := function ( l, r )
>   return (l.modulus = r.modulus)
>   and (l.representative-r.representative) mod l.modulus = 0;
> end;;
gap> ResidueClass := function ( representative, modulus )
>   return rec(
```



```

>    representative := representative,
>    modulus        := modulus,
>    operations      := ResidueClassOps );
> end;;
gap> l := ResidueClass( 13, 23 );;
gap> r := ResidueClass( -10, 23 );;
gap> l = r;
true
gap> l = ResidueClass( 10, 23 );
false

```

```

'<rec1> < <rec2>'
'<rec1> <= <rec2>'
'<rec1> > <rec2>'
'<rec1> >= <rec2>'

```

The operators '<', '<=', '>', and '>=' evaluate to 'true' if the record <rec1> is less than, less than or equal to, greater than, and greater than or equal to the record <rec2>, and to 'false' otherwise.

To compare records we imagine that the components of both records are sorted according to their names. Then the records are compared lexicographically with unbound elements considered smaller than anything else. Precisely one record <rec1> is considered less than another record <rec2> if <rec2> has a component with name <name2> and either <rec1> has no component with this name or '<rec1>.<name2> < <rec2>.<name2>' and for each component of <rec1> with name '<name1> < <name2>' <rec2> has a component with this name and '<rec1>.<name1> = <rec2>.<name1>'. Records may also be compared with objects of other types, they are greater than anything else, unless the record has a special comparison function (see below) that says otherwise.

```

gap> rec( a := 1, b := 2 ) < rec( b := 2, a := 1 );
false    # they are equal
gap> rec( a := 1, b := 2 ) < rec( a := 2, b := 0 );
true     # the \verb|'a'| elements are compared first and 1 is less than 2
gap> rec( a := 1 ) < rec( a := 1, b := 2 );
true     # unbound is less than 2
gap> rec( a := 1 ) < rec( a := 0, b := 2 );
false    # the \verb|'a'| elements are compared first and 0 is less than 1
gap> rec( b := 1 ) < rec( b := 0, a := 2 );
true     # the \verb|'a'|-s are compared first and unbound is less than 2
gap> rec( a := 1 ) < 1;
false    # other objects are less than records

```

However a record may contain a special 'operations' record that contains a function that is called when this record is an operand of a comparison. The precise mechanism is as follows. If the operand of the equality operator '<' is a record, and if this record has an element with the name 'operations'

that is a record, and if this record has an element with the name ' $<$ ' that is a function, then this function is called with the operands of ' $<$ ' as arguments, and the value of the operation is the result returned by this function. In this case a record may also be smaller than an object of another type if this function says so. It is probably not a good idea to define a comparison function in such a way that the resulting relation is not an ordering relation, i.e., not antisymmetric, and transitive. Note that there are no corresponding ' $<=$ ', ' $>$ ', and ' $>=$ ' functions, since those operations are implemented as ' $\text{not } <\text{right}> < <\text{left}>$ ', ' $<\text{right}> < <\text{left}>$ ', and ' $\text{not } <\text{left}> < <\text{right}>$ ' respectively.

The following example shows one piece of the definition of residue classes, using record operations. Of course this is far from a complete implementation. Note that the ' $<$ ' must be quoted, so that it is taken as an identifier (see "Identifiers").

```
gap> ResidueClassOps := rec( );
gap> ResidueClassOps.< := function ( l, r )
>   if l.modulus <> r.modulus then
>     Error("<l> and <r> must have the same modulus");
>   fi;
>   return l.representative mod l.modulus
>         < r.representative mod r.modulus;
> end;;
gap> ResidueClass := function ( representative, modulus )
>   return rec(
>     representative := representative,
>     modulus         := modulus,
>     operations      := ResidueClassOps );
> end;;
gap> l := ResidueClass( 13, 23 );
gap> r := ResidueClass( -1, 23 );
gap> l < r;
true    # 13 is less than 22
gap> l < ResidueClass( 10, 23 );
false   # 10 is less than 13
```

5.5 Operations for Records

Usually no operations are defined for record. However a record may contain a special ' operations ' record that contains functions that are called when this record is an operand of a binary operation. This mechanism is detailed below for the addition.

'<obj> + <rec>' , '<rec> + <obj>'

If either operand is a record, and if this record contains an element with name ' operations ' that is a record, and if this record in turn contains an element with the name ' $+$ ' that is a function, then this function is called with the two operands as arguments, and the value of the addition is the value returned by that function. If both operands are records with such a function ' $\text{<rec>.operations.+}$ ',

then the function of the **right** operand is called. If either operand is a record, but neither operand has such a function '`<rec>.operations.+`', an error is signalled.

```
'<obj> - <rec>', '<rec> - <obj>'
'<obj> * <rec>', '<rec> * <obj>'
'<obj> / <rec>', '<rec> / <obj>'
'<obj> mod <rec>', '<rec> mod <obj>'
'<obj> ^ <rec>', '<rec> ^ <obj>'
```

This is evaluated similar, but the functions must obviously be called '`-`', '`*`', '`/`', '`mod`', '`^`' respectively.

The following example shows one piece of the definition of a residue classes, using record operations. Of course this is far from a complete implementation. Note that the '`*`' must be quoted, so that it is taken as an identifier (see "Identifiers").

```
gap> ResidueClassOps := rec( );;
gap> ResidueClassOps.* := function ( l, r )
>   if l.modulus <> r.modulus then
>     Error("<l> and <r> must have the same modulus");
>   fi;
>   return rec(
>     representative := (l.representative * r.representative)
>                       mod l.modulus,
>     modulus        := l.modulus,
>     operations      := ResidueClassOps );
> end;;
gap> ResidueClass := function ( representative, modulus )
>   return rec(
>     representative := representative,
>     modulus        := modulus,
>     operations      := ResidueClassOps );
> end;;
gap> l := ResidueClass( 13, 23 );;
gap> r := ResidueClass( -1, 23 );;
gap> s := l * r;
rec(
  representative := 10,
  modulus        := 23,
  operations      := rec(
    * := function ( l, r ) ... end ) )
```

5.6 In for Records

```
'<element> in <rec>'
```

Usually the membership test is only defined for lists. However a record may contain a special 'operations' record, that contains a function that is called when this record is the right operand of the 'in' operator. The precise mechanism is as follows.

If the right operand of the 'in' operator is a record, and if this record contains an element with the name 'operations' that is a record, and if this record in turn contains an element with the name 'in' that is a function, then this function is called with the two operands as arguments, and the value of the membership test is the value returned by that function. The function should of course return 'true' or 'false'.

The following example shows one piece of the definition of residue classes, using record operations. Of course this is far from a complete implementation. Note that the 'in' must be quoted, so that it is taken as an identifier (see "Identifiers").

```
gap> ResidueClassOps := rec( );
gap> ResidueClassOps.\in := function ( l, r )
>   if IsInt( l ) then
>     return (l - r.representative) mod r.modulus = 0;
>   else
>     return false;
>   fi;
> end;;
gap> ResidueClass := function ( representative, modulus )
>   return rec(
>     representative := representative,
>     modulus         := modulus,
>     operations      := ResidueClassOps );
> end;;
gap> l := ResidueClass( 13, 23 );
gap> -10 in l;
true
gap> 10 in l;
false
```

5.7 Printing of Records

'Print(<rec>)'

If a record is printed by 'Print' or by the main loop, it is usually printed as record literal, i.e., as a sequence of components, each in the format '<name> \:= <value>', separated by commas and enclosed in 'rec(' and ')'

```
gap> r := rec();; r.a := 1;; r.b := 2;;
gap> r;
rec(
```

```

a := 1,
b := 2 )

```

But if the record has an element with the name `'operations'` that is a record, and if this record has an element with the name `'Print'` that is a function, then this function is called with the record as argument. This function must print whatever the printed representation of the record should look like.

The following example shows one piece of the definition of residue classes, using record operations. Of course this is far from a complete implementation. Note that it is typical for records that mimic group elements to print as a function call that, when evaluated, will create this group element record.

```

gap> ResidueClassOps := rec( );;
gap> ResidueClassOps.Print := function ( r )
>   Print( "ResidueClass( ",
>           r.representative mod r.modulus, ", ",
>           r.modulus, " )" );
> end;;
gap> ResidueClass := function ( representative, modulus )
>   return rec(
>     representative := representative,
>     modulus         := modulus,
>     operations      := ResidueClassOps );
> end;;
gap> l := ResidueClass( 33, 23 );
ResidueClass( 10, 23 )

```

5.8 IsRec

`'IsRec(<obj>)'`

`'IsRec'` returns `'true'` if the object `<obj>`, which may be an object of arbitrary type, is a record, and `'false'` otherwise. Will signal an error if `<obj>` is a variable with no assigned value.

```

gap> IsRec( rec( a := 1, b := 2 ) );
true
gap> IsRec( IsRec );
false

```

5.9 IsBound

`'IsBound(<rec>.<name>)'`
`'IsBound(<list>[<n>])'`

In the first form `'IsBound'` returns `'true'` if the record `<rec>` has a component with the name `<name>`, which must be an ident and `'false'` otherwise. `<rec>` must evaluate to a record, otherwise an error is signalled.

In the second form `'IsBound'` returns `'true'` if the list `<list>` has a element at the position `<n>`, and `'false'` otherwise. `<list>` must evaluate to a list, otherwise an error is signalled.

```
gap> r := rec( a := 1, b := 2 );;
gap> IsBound( r.a );
true
gap> IsBound( r.c );
false
gap> l := [ , 2, 3, , 5, , 7, , , 11 ];;
gap> IsBound( l[7] );
true
gap> IsBound( l[4] );
false
gap> IsBound( l[101] );
false
```

Note that `'IsBound'` is special in that it does not evaluate its argument, otherwise it would always signal an error when it is supposed to return `'false'`.

5.10 Unbind

```
'Unbind( <rec>.<name> )'
'Unbind( <list>[<n>] )'
```

In the first form `'Unbind'` deletes the component with the name `<name>` in the record `<rec>`. That is, after execution of `'Unbind'`, `<rec>` no longer has a record component with this name. Note that it is not an error to unbind a nonexisting record component. `<rec>` must evaluate to a record, otherwise an error is signalled.

In the second form `'Unbind'` deletes the element at the position `<n>` in the list `<list>`. That is, after execution of `'Unbind'`, `<list>` no longer has an assigned value at the position `<n>`. Note that it is not an error to unbind a nonexisting list element. `<list>` must evaluate to a list, otherwise an error is signalled.

```
gap> r := rec( a := 1, b := 2 );;
gap> Unbind( r.a ); r;
rec(
  b := 2 )
gap> Unbind( r.c ); r;
rec(
  b := 2 )
```

```

gap> l := [ , 2, 3, 5, , 7, , , 11 ];;
gap> Unbind( l[3] ); l;
[ , 2,, 5,, 7,,, 11 ]
gap> Unbind( l[4] ); l;
[ , 2,,, 7,,, 11 ]

```

Note that 'Unbind' does not evaluate its argument, otherwise there would be no way for 'Unbind' to tell which component to remove in which record, because it would only receive the value of this component.

5.11 Copy

'Copy(<obj>)'

'Copy' returns a copy <new> of the object <obj>. You may apply 'Copy' to objects of any type, but for objects that are not lists or records 'Copy' simply returns the object itself.

For lists and records the result is a **new** list or record that is **not identical** to any other list or record (see "Identical Lists" and "Identical Records"). This means that you may modify this copy <new> by assignments (see "List Assignment" and "Record Assignment") or by adding elements to it (see "Add" and "Append"), without modifying the original object <obj>.

```

gap> list1 := [ 1, 2, 3 ];;
gap> list2 := Copy( list1 );
[ 1, 2, 3 ]
gap> list2[1] := 0;; list2;
[ 0, 2, 3 ]
gap> list1;
[ 1, 2, 3 ]

```

That 'Copy' returns the object itself if it is not a list or a record is consistent with this definition, since there is no way to change the original object <obj> by modifying <new>, because in fact there is no way to change the object <new>.

'Copy' basically executes the following code for lists, and similar code for records.

```

new := [];
for i in [1..Length(obj)] do
  if IsBound(obj[i]) then
    new[i] := Copy( obj[i] );
  fi;
od;

```

Note that 'Copy' recursively copies all elements of the object <obj>. If you only want to copy the top level use 'ShallowCopy' (see "ShallowCopy").

```

gap> list1 := [ [ 1, 2 ], [ 3, 4 ] ];;
gap> list2 := Copy( list1 );
[ [ 1, 2 ], [ 3, 4 ] ]
gap> list2[1][1] := 0;; list2;
[ [ 0, 2 ], [ 3, 4 ] ]
gap> list1;
[ [ 1, 2 ], [ 3, 4 ] ]

```

The above code is not entirely correct. If the object <obj> contains a list or record twice this list or record is not copied twice, as would happen with the above definition, but only once. This means that the copy <new> and the object <obj> have exactly the same structure when view as a general graph.

```

gap> sub := [ 1, 2 ];; list1 := [ sub, sub ];;
gap> list2 := Copy( list1 );
[ [ 1, 2 ], [ 1, 2 ] ]
gap> list2[1][1] := 0;; list2;
[ [ 0, 2 ], [ 0, 2 ] ]
gap> list1;
[ [ 1, 2 ], [ 1, 2 ] ]

```

5.12 ShallowCopy

'ShallowCopy(<obj>)'

'ShallowCopy' returns a copy of the object <obj>. You may apply 'ShallowCopy' to objects of any type, but for objects that are not lists or records 'ShallowCopy' simply returns the object itself. For lists and records the result is a *new* list or record that is *not identical* to any other list or record (see "Identical Lists" and "Identical Records"). This means that you may modify this copy <new> by assignments (see "List Assignment" and "Record Assignment") or by adding elements to it (see "Add" and "Append"), without modifying the original object <obj>.

```

gap> list1 := [ 1, 2, 3 ];;
gap> list2 := ShallowCopy( list1 );
[ 1, 2, 3 ]
gap> list2[1] := 0;; list2;
[ 0, 2, 3 ]
gap> list1;
[ 1, 2, 3 ]

```

That 'ShallowCopy' returns the object itself if it is not a list or a record is consistent with this definition, since there is no way to change the original object <obj> by modifying <new>, because in fact there is no way to change the object <new>.

'ShallowCopy' basically executes the following code for lists, and similar code for records.


```
new := [];  
for i in [1..Length(obj)] do  
  if IsBound(obj[i]) then  
    new[i] := obj[i];  
  fi;  
od;
```

Note that 'ShallowCopy' only copies the top level. The subobjects of the new object <new> are identical to the corresponding subobjects of the object <obj>. If you want to copy recursively use 'Copy' (see "Copy").

5.13 RecFields

```
'RecFields( <rec> )'
```

'RecFields' returns a list of strings corresponding to the names of the record components of the record <rec>.

```
gap> r := rec( a := 1, b := 2 );;  
gap> RecFields( r );  
[ "a", "b" ]
```

Note that you cannot use the string result in the ordinary way to access or change a record component. You must use the '<rec>.(<name>)' construct (see "Accessing Record Elements" and "Record Assignment").

Bibliography

- [AK99] Vincenzo Acciario and Jürgen Klüners. Computing Automorphisms of Abelian Number Fields. *Math. Comp.*, 68(227):1179–1186, 1999. 642, 644, 645
- [Bai96] Georg Baier. Zum Round 4 Algorithmus. Diplomarbeit, Technische Universität Berlin, 1996. 678, 685
- [BH96] Yuri Bilu and Guillaume Hanrot. Solving Thue equations of high degree. *J. Number Th.*, 60:373–392, 1996. 873
- [Bli14] H. F. Blichfeldt. A new principle in the geometry of numbers, with some applications. *Trans. Amer. Math. Soc.*, 15:227–235, 1914. 421
- [BN96] Johannes Buchmann and Stefan Neis. Algorithms for linear algebra problems over principal ideal rings. *Technical Report*, 1996. 584
- [BP91] Wieb Bosma and Michael E. Pohst. Computations with finitely generated modules over Dedekind domains. In Stephen M. Watt, editor, *Proceedings ISSAC'91*, pages 151–156, 1991. 619
- [CDO96] Henri Cohen, Francisco Diaz y Diaz, and Michel Olivier. Computing ray class groups, conductors and discriminants. In Henri Cohen, editor, *ANTS II*, volume 1122 of *LNCS*, pages 52–59. Springer-Verlag, 1996. 337, 845
- [CDO97] Henri Cohen, Francisco Diaz y Diaz, and Michel Olivier. Computing ray class groups, conductors and discriminants. *Submitted to Math. Comp.*, 1997. 337, 845
- [CHM98] John H. Conway, Alexander Hulpke, and John McKay. On transitive permutation groups. *LMS Journal of Computation and Mathematics*, 1:1–8, 1998. 405
- [CM94] David Casperson and John McKay. Symmetric Functions, m -Sets, and Galois Groups. *Math. Comp.*, 63:749–757, 1994. 408, 409, 411, 412
- [Coh95] Henri Cohen. *A Course in Computational Algebraic Number Theory*. Springer, 1995. 760
- [Coh96] Henri Cohen. Hermite and Smith normal form algorithms over Dedekind domains. *Math. Comp.*, 65:1681–1699, 1996. 453, 619, 623
- [Dab95] Mario Daberkow. *Über die Bestimmung der ganzen Elemente in Radikalerweiterungen algebraischer Zahlkörper*. Dissertation, Technische Universität Berlin, 1995. 674, 675, 676, 889

- [Fie97] Claus Fieker. *Über relative Normgleichungen in algebraischen Zahl-körpern*. Dissertation, Technische Universität Berlin, 1997. 683
- [Fin84] Ulrich Fincke. *Ein Ellipsoidverfahren zur Lösung von Normgleichungen in algebraischen Zahlkörpern*. Dissertation, Heinrich-Heine-Universität Düsseldorf, 1984. 683
- [Fri97] Carsten Friedrichs. Berechnung relativer Ganzheitsbasen mit dem Round-2-Algorithmus. Diplomarbeit, Technische Universität Berlin, 1997. 637, 678, 685
- [Gei97] Katharina Geißler. Zur Berechnung von Galoisgruppen. Diplomarbeit, Technische Universität Berlin, 1997. 392
- [GPP93] Istvan Gaál, Attila Pethő, and Michael E. Pohst. On the resolution of index form equations in quartic number fields. *J. Symbolic Comp.*, 16:563–584, 1993. 668
- [GPP96] Istvan Gaál, Attila Pethő, and Michael E. Pohst. Simultaneous representation of integers by a pair of ternary quadratic forms – with an application to index form equations in quartic number fields. *J. Number Th.*, 57:90–104, 1996. 668
- [GS89] Istvan Gaál and Nicole Schulte. Computing all power integral bases of cubic fields. *Math. Comp.*, 53:689–696, 1989. 668
- [Has63] Helmut Hasse. *Zahlentheorie*. Akademie Verlag, Berlin, 1963. 846
- [Has70] Helmut Hasse. *Bericht über neuere Untersuchungen und Probleme aus der Theorie der algebraischen Zahlkörper*. Physica Verlag, Berlin, 1970. 828
- [Heß96] Florian Heß. Zur Klassengruppenberechnung in algebraischen Zahlkörpern. Diplomarbeit, Technische Universität Berlin, 1996. 466
- [Heß99] Florian Heß. *Zur Divisorenklassengruppenberechnung in globalen Funktionenkörpern*. Dissertation, Technische Universität Berlin, 1999. 61, 62, 63, 66, 94, 103, 107
- [Hop98] Andreas Hoppe. *Normal forms over Dedekind domains, efficient implementation in the computer algebra system KANT*. Dissertation, Technische Universität Berlin, 1998. 607, 608, 611, 623
- [Klü95] Jürgen Klüners. Über die Berechnung von Teilkörpern algebraischer Zahlkörper. Diplomarbeit, Technische Universität Berlin, 1995. 403, 714
- [Klü97] Jürgen Klüners. *Über die Berechnung von Automorphismen und Teilkörpern algebraischer Zahlkörper*. Dissertation, Technische Universität Berlin, 1997. 403, 642, 644, 645, 714
- [KP97] Jürgen Klüners and Michael E. Pohst. On computing subfields. *J. Symbolic Comp.*, 24(3):385–397, 1997. 403, 714
- [Lan87] Serge Lang. *Elliptic Functions*. Springer-Verlag, 1987. 881
- [Maa49] Hans Maass. Über eine neue art von nichtanalytischen automorphen funktionen und die bestimmung dirichletscher reihen durch funktionalgleichungen. *Math. Ann.*, 121:141–183, 1949. 302

- [Nag69] Trygve Nagell. Sur un type particulier d'unités algébriques. *Ark. Mat.*, 8:163–184, 1969. 664, 724
- [Pau96] Sebastian Pauli. Zur Berechnung von Strahlklassengruppen. Diplomarbeit, Technische Universität Berlin, 1996. 337, 826, 839, 843, 846
- [PP98] Sebastian Pauli and Michael E. Pohst. On the computation of the multiplicative group of residue class rings. *Submitted to Math. Comp.*, 1998. 337, 839, 843
- [PZ89] Michael E. Pohst and Hans Zassenhaus. *Algorithmic Algebraic Number Theory*. Cambridge University Press, 1989. 665
- [S⁺97] Martin Schönert et al. GAP 3.4, patchlevel 4. School of Mathematical and Computational Sciences, University of St. Andrews, Scotland, 1997. 392, 405, 407, 413
- [Sch90a] Reinhard Schertz. Über die Nenner normierter Teilwerte der Weierstraßschen \wp -Funktion. *J. Number Th.*, 34:229–234, 1990. 479
- [Sch90b] Reinhard Schertz. Zur expliziten Berechnung von Ganzheitsbasen in Strahlklassenkörpern über imaginär-quadratischen Zahlkörpern. *J. Number Th.*, 34:41–53, 1990. 282, 477, 479
- [Sch96] Martin Schörmig. *Untersuchung konstruktiver Probleme in globalen Funktionenkörpern*. Dissertation, Technische Universität Berlin, 1996. 114, 175, 184, 190, 234, 244
- [SM85] Leonhard H. Soicher and John McKay. Computing Galois Groups over the rationals. *J. Number Th.*, 20:273–281, 1985. 408, 409, 411, 412
- [Sta73] Richard P. Stauduhar. The determination of Galois Groups. *Math. Comp.*, 27:981–996, 1973. 392
- [Str84] Roelof J. Stroeker. How to solve a diophantine equation. *Amer. Math. Monthly*, 91:385–392, 1984. 506
- [TdW89] Nikos Tzanakis and Benjamin M. M. de Weger. On the practical solution of the thue equation. *J. Number Th.*, 31:99–132, 1989. 506
- [Wil] Klaus Wildanger. Computing all integral points on Mordell's equation $y^2 = x^3 + k$ by solving cubic index form equations. Submitted to Acta Arith. 506
- [Wil93] Klaus Wildanger. Über Grundeinheitenberechnung in algebraischen Zahlkörpern. Diplomarbeit, Heinrich-Heine-Universität Düsseldorf, 1993. 726, 727, 731

Index

Packages

AbelianGroup.....	3–56
Aiff.....	59–250
Arc.....	251ff.
Drinf.....	285ff.
Ecc.....	291–300
Elt.....	303–349
FF.....	355–372
Find.....	385–389
Galois.....	392–417
Ideal.....	423–473
Im.....	475–479
Infty.....	483–486
Integer.....	488–506
Is.....	507–531
Lat.....	537–559
List.....	562ff.
Mat.....	566–599
Min.....	600ff.
Module.....	604–625
Order.....	636–731
Poly.....	734–764
Pvm.....	766–795
Qf.....	797–805
Qp.....	806–815
Random.....	818–824
Ray.....	826–850
Simplex.....	858–861
Subfield.....	866ff.
Thue.....	870–873

A

AbelianDualGroup.....	3
AbelianDualHom.....	5
AbelianFieldToRCF.....	6
AbelianFixPointGroup.....	7
AbelianGroup, package.....	3–56
AbelianGroupBasis.....	8

AbelianGroupCanonicalQuotient.....	9
AbelianGroupCreate.....	11
AbelianGroupCyclicFactors.....	12
AbelianGroupDirectProduct.....	13
AbelianGroupDiscreteExp.....	15
AbelianGroupDiscreteLog.....	16
AbelianGroupEltCreate.....	18
AbelianGroupEltMove.....	19
AbelianGroupEltOrder.....	20
AbelianGroupEltRandom.....	21
AbelianGroupEltReduce.....	22
AbelianGroupEnumInit.....	23
AbelianGroupEnumNext.....	24
AbelianGroupEqual.....	25
AbelianGroupExponent.....	26
AbelianGroupGenerators.....	27
AbelianGroupHomCreate.....	28
AbelianGroupHomImage.....	30
AbelianGroupHomKernel.....	31
AbelianGroupIndex.....	33
AbelianGroupIntersect.....	34
AbelianGroupIsAut.....	35
AbelianGroupIsSub.....	37
AbelianGroupMinNumberGenerators.....	38
AbelianGroupMultiHomCreate.....	39
AbelianGroupName.....	41
AbelianGroupNumberGenerators.....	42
AbelianGroupOrder.....	43
AbelianGroupPrintLevel.....	44
AbelianGroupSmithCreate.....	45
AbelianGroupTensorProduct.....	46
AbelianGroupUnite.....	47
AbelianHomGroup.....	48
AbelianMultiHomGroup.....	50
AbelianQuotientGroup.....	52
AbelianRayClassGroupAutoCreate.....	53
AbelianRayGroupImbed.....	55
AbelianSubGroup.....	56

- Abelian field
 - as ray class field..... 6
 - conversion
 - to ray class field..... 6
 - to RCF 6
- Abelian group
 - basis 8
 - check for automorphism property..... 35
 - check for equality 25
 - check for subgroup property..... 37
 - creation of an element..... 18
 - creation of a multilinear mapping 39
 - creation..... 11
 - creation of a subgroup 56
 - cyclic factors..... 12
 - direct product 13
 - discrete exponentiation..... 15
 - discrete log 16
 - dual group..... 3
 - dual homomorphism 5
 - element
 - order 20
 - elements
 - enumeration init 23
 - next enumeration 24
 - random 21
 - embedding into a ray group 55
 - exponent..... 26
 - exponent vector..... 15f.
 - fix point group 7
 - generators 27, 38, 42
 - minimal number of generators..... 38
 - number of generators 42
 - group exponent 26
 - group of fix points..... 7
 - group union..... 47
 - homomorphism
 - is automorphism 35
 - image..... 30
 - kernel..... 31
 - homomorphism group..... 48
 - index of a subgroup 33
 - intersection 34
 - multilinear mapping..... 39
 - creation..... 39
 - multimorphism group 50
 - of a list of groups 50
 - name 41
 - order..... 43
 - order of an element 20
 - printlevel 44
 - quotient group..... 9, 52
 - random element..... 21
 - representation 15f.
 - of an object in a group 15f.
 - Smith normal form..... 45
 - cyclic factors..... 12
 - SNF 45
 - is subgroup 37
 - subgroup..... 56
 - tensor product 46
 - of a list of groups 46
 - union of groups 47
- Abs** 58
- absolute degree
 - of an order..... 661
- absolute order
 - of a relative order 639
 - with small index 707
- accessing
 - list elements 915
 - record elements 944
- add
 - an element to a set..... 939
 - elements to a list..... 919
 - the elements of a list..... 934
- Alff** 59
- Alff, package* 59–250
- AlffCanonicalDivisor** 60
- AlffClassGroup** 61
- AlffClassGroupGenBound** 62
- AlffClassGroupGenBoundStrong** 63
- AlffClassGroupGens** 64
- AlffClassGroupPRank** 65
- AlffClassNumberApprox** 66
- AlffClassNumberApproxBound** 67
- AlffConstField** 68

AlffConstFieldSize	69	AlffEltGenY	119
AlffDeg	70	AlffEltInftyVals	120
AlffDiff	71	AlffEltIsInIdeal	121
AlffDiffAlff	72	AlffEltMin	122
AlffDiffCartier	73	AlffEltMinPoly	123
AlffDiffCartierMatrix	74	AlffEltMove	124
AlffDiffDivisor	75	AlffEltNewtonLift	125
AlffDifferent	80	AlffEltNorm	126
AlffDifferentDeg	81	AlffEltNum	127
AlffDifferentiation	82	AlffEltOrder	128
AlffDiffFirstKind	76	AlffEltPthRoot	129
AlffDiffResiduum	77	AlffEltRepMat	130
AlffDiffSpace	78	AlffEltResiduum	131
AlffDiffValuation	79	AlffEltRoot	132
AlffDimExactConstField	83	AlffEltToList	133
AlffDivisor	84	AlffEltToResField	134
AlffDivisorAlff	86	AlffEltTrace	137
AlffDivisorClassRep	87	AlffEltValuation	138
AlffDivisorDeg	88	AlffGapNumbers	139
AlffDivisorDegOne	89	AlffGenus	140
AlffDivisorDen	90	AlffHasseWittInvariant	141
AlffDivisorIdeals	91	AlffHermitianFunField	142f.
AlffDivisorLargeLBasisShort	98	AlffIdeal2EltAssure	144
AlffDivisorLargeLDim	99	AlffIdealAlff	145
AlffDivisorLBasis	92f.	AlffIdealBasis	146
AlffDivisorLBasisShort	94	AlffIdealBasisUpperHNF	147
AlffDivisorLDim	96	AlffIdealClassGroupUnitsInfty	148
AlffDivisorLIndex	97	AlffIdealFactor	149
AlffDivisorNorm	100	AlffIdealGenerators	150
AlffDivisorNum	101	AlffIdealIsPrime	151
AlffDivisorPlaces	102	AlffIdealIsPrimeKnown	152
AlffDivisorReduction	103	AlffIdealNorm	153
AlffDivisorsSmoothNum	107	AlffIdealOrder	154
AlffDivisorsSmoothRatio	108	AlffIdealPlace	155
AlffDivisorSupp	106	AlffIdealValuation	156
AlffEllipticFunField	109	AlffIharaBound	157
AlffElt	110	AlffInit	158f.
AlffEltAlff	111	AlffIsAbs	160
AlffEltApprox	112	AlffIsGlobal	162
AlffEltBstar	114	AlffIsGlobalAssert	163
AlffEltCharPoly	115	AlffLinearSeriesEnumElt	167
AlffEltDen	116	AlffLinearSeriesEnumEnv	169
AlffEltEval	117	AlffLinearSeriesEnumNext	170
AlffEltGenT	118	AlffLPoly	164

AlffLPolyLift	165	AlffPolyIsIrreducible	227
AlffLPolyRed	166	AlffPolyIsIrrSep	226
AlffOesterleBound	171	AlffPuisseuxCoeff	228
AlffOrderAlff	172	AlffRamDivisor	229
AlffOrderBasis	173f.	AlffRegulator	230
AlffOrderBasisValues	175	AlffResFieldEltLift	231
AlffOrderDedekindTest	176	AlffRootParams	234
AlffOrderDeg	177	AlffRoots	235
AlffOrderDisc	178	AlffSerreBound	240
AlffOrderEqFinite	179	AlffSignature	241
AlffOrderEqInfty	180	AlffSUnits	237
AlffOrderIndex	181	AlffTameInftyPlace	242
AlffOrderIsFinite	182	AlffUnitRank	243
AlffOrderLO	184	AlffUnitsFund	244
AlffOrderMaxFinite	185	AlffVarT	245
AlffOrderMaximal	187	AlffVarY	246
AlffOrderMaxInfty	186	AlffWeierstrassPlaces	247
AlffOrderPoly	189	AlffWronskian	248
AlffOrderReduce	190	AlffWronskianOrders	250
AlffOrders	193, 195	algebraic	
AlffOrderTransformationMatrix	192	– element	
AlffPlaceAlff	197	– – order	284
AlffPlaceBeta	198	– elements	
AlffPlaceDeg	199	– – characteristic polynomial of an	264
AlffPlaceIdeal	200	– – denominators	283
AlffPlaceIsFinite	201	– – numerators	633
AlffPlaceMin	202	– number	
AlffPlaceOrder	203	– – norm of an	632
AlffPlacePrimeElt	204	– number field	
AlffPlaceRam	205	– – trace of an element	875
AlffPlaceRandom	206	algebraic element	
AlffPlaceResDeg	207	– as finite field element	344
AlffPlaceResField	208	– canonical image in finite field	344
AlffPlaces	212	– index of equation suborder	318
AlffPlacesDegOne	213, 215	– modulo an integer	333
AlffPlacesDegOneNonSingFiniteNum	217	– Newton lift	331
AlffPlacesDegOneNum	218f.	– norm of an	632
AlffPlacesDegOneNumBound	220	– reduced representation	317
AlffPlacesNonSpecial	221	– representation modulo an ideal	317
AlffPlacesNum	222f.	– valuation	349
AlffPlaceSplit	209	– with certain valuation	307
AlffPlaceSplitType	210f.		
AlffPoly	224		
AlffPolyIrrIsSep	225		

- algebraic number
 - absolute logarithm height 306
 - characteristic polynomial..... 310
 - coefficient matrix 324
 - conjugates 312
 - converting lists of a.n. to matrix 324
 - creation..... 303
 - denominator 313
 - discriminant of generated module 323
 - divisors 314
 - element of an order test..... 321
 - factorization 316
 - image under automorphisms 308
 - integer test 320
 - integral test..... 321
 - primitivity test 322
 - vector of conjugates..... 305
- algebraic numbers
 - move into given order 627
- algorithm
 - Euclidean
 - – extended..... 504
- append
 - elements to a list..... 919
- approximation
 - of algebraic elements..... 307
- approximation theorem
 - infinite places of orders..... 826
- approximation theorem for algebraic numbers . 307
- Arc, package* 251ff.
- ArcCos*..... 251
- ArcSin*..... 252
- ArcTan*..... 253
- Arg*..... 254
- assignment
 - to a list 917
 - to a record..... 944
 - variable..... 902
- associativity 900
- automorphism
 - of a field
 - – corresponding group automorphism 53
 - image of an algebraic number 308
- automorphisms of a ray class field 832
- B**
- bach bound 646
- BagRead* 255
- BagWrite* 256
- basis change
 - of a function field element..... 124
- basis
 - 0-reduced
 - – function field order 190
 - of an Abelian group..... 8
 - of a finite field 367
 - of a function field ideal
 - – in Hermite normal form..... 147
 - function field order..... 173
 - of a lattice 540
 - of an order..... 647
 - – transformation matrix 719
 - presentation
 - – coefficients..... 546
- basis reduction
 - for enumeration 555
 - of a lattice 557
- Bell*..... 259
- Bernoulli*..... 260
- BernoulliMagma*..... 261
- bivariate polynomial
 - swap variables 761
- blank 895
- BNF 909
- body..... 906
- bound 897
 - for enumerating vectors in a lattice. 550, 554
- C**
- C* 262
- canonical divisor
 - computation 60

- Cartier operator 73
- representation matrix 74
- Ceil** 263
- Characteristic** 266
- characteristic polynomial
 - of a function field element 115
- characteristic polynomial of an algebraic element 310
- CharPoly** 264
- CharToInt** 265
- check
 - field
 - for finite 515
 - integer 518
 - integer is prime 492
 - integer for square 493
 - for record 530
 - for Thue 531
- check for
 - algebraic number is element of an order . 321
 - algebraic number is integer 320f.
 - algebraic number is primitive 322
 - automorphism property 35
 - element of module 616
 - equality of two Abelian group 25
- CheckArgus** 267
- chinese remainder
 - for ideals 269
 - for integers 269
- ChineseRemainder** 269
- Cholesky decomposition 541
- class field
 - ray 476
 - – artin automorphism 830
 - – automorphisms and primitive element . 832
 - – discriminant and signature 845
- class group
 - of a function field
 - computation 61
 - generators 64
 - generators degree bound 62
 - generators degree bound (strong) 63
 - – representation in given generators 87
 - ray 839
 - – conductor 843
 - – generators 841
 - – representation of an ideal 466
 - of a function field
 - p -rank 65
- class number
 - of a function field
 - approximation 66
 - bound 67
- class representation
 - of an ideal 435
- clone
 - an object 955f.
- Close** 270
- close
 - file after reading fld-format 270
- Coeff** 271
- coefficient ideals
 - of a relative order 656
- coefficient list of a polynomial 762
- coefficient matrix of algebraic numbers 324
- coefficient ring
 - of a polynomial algebra 799
- coefficients
 - eutactic 351
- coefficients of lattice basis vector 546
- coefficients list
 - of a function field element 133
- Colors** 273
- ColorString** 272
- comments 895
- Comp** 274
- comparisons
 - of lists 923
- ComplexGamma** 275
- compute
 - roots of a polynomial 864
- concatenation
 - of lists 927
- conductor
 - of a rayclass group 843
 - test 844

- Conj** 276
- conjugates of an algebraic number 312
- constant field
 - of a function field 68
 - – dimension 83
- convert
 - to a list 914
 - to a set 938
- copy
 - an object 955f.
- Cos** 277
- count
 - fields in database 279
- counting lattice points 548
- creation
 - of an Abelian group 11
 - of an element of an Abelian group 18
 - elements
 - – of finite field 358
 - finite field 355
 - function field 59
 - of a polynomial 734
- cross product of lists 929
- cyclic factors
 - of an Abelian group 12
- D**
- database
 - count fields 279
 - query 279
- Date** 278
- DbQuery** 279
- decomposition of an unit 347
- Dedekind-Test
 - of a function field equation order 176
- DedekindEta** 282
- degree
 - absolute
 - – of an order 661
 - of a divisor 88
 - of a finite field \mathbb{F}_q 372
 - of a function field extension 70
 - of an order 660
 - of a polynomial 737
- degree of inertia
 - of a prime ideal 438
- Den** 283
- denominator
 - of an algebraic element 283
 - of an algebraic number 313
 - of a divisor 90
 - of a fractional ideal 283
 - of a function field element 116
 - module 606
 - of a rational function 800
 - of a rational number 283
- derivation
 - of rational function 801
- derivative
 - of a polynomial 738
- determinant
 - of an integral matrix 582
 - module 607
- difference
 - of records 950
- different
 - of a function field
 - – computation 80
 - – degree 81
- differential
 - Cartier operator 73f.
 - divisor of 75
 - exact
 - – of a function field element 71
 - of the first kind
 - – basis 76
 - residuum
 - – computation 77
 - Weil
 - – basis 78
- differentiation
 - of a function field element 82
- dimension
 - of a divisor of a function field 99
 - of the riemann-roch space 96

direct product
 – of Abelian groups 13
Disc 284
 discrete
 – logarithm
 – – of an element of a finite field 361
 discrete exp
 – of an object of an Abelian group 15
 discrete log
 – of an object of an Abelian group 16
 discriminant
 – of an algebraic element 284
 – of a function field order 178
 – of a lattice 284, 542
 – of a number field 662
 – of an order 662
 – – of an algebraic function field 284
 – – of an algebraic number field 284
 – of a polynomial 284, 739
 – – reduced 757
 – of a ray class field 845
 discriminant of a module 323
 division
 – remainder by division of two integers 876
 divisor
 – canonical
 – – computation 60
 – of a differential 75
 – of a function field
 – – basis for a large divisor 98
 – – basis of a Riemann-Roch space 93
 – – check 509
 – – dimension 99
 – – index of speciality 97
 – – norm 100
 – – number of (n,m) -smooth divisors 107
 – – one of degree one 89
 – – places and exponents 102
 – – ratio of numbers of smooth divisors ... 108
 – – riemann-roch space 94
 – – support 106
 – function field it belongs to 86
 – list of divisors of an integer 488

– norm of a 632
 – prime divisors of an integer 498
 – reduced
 – – computation 103
 – riemann-roch space
 – – dimension 96
 divisors of an algebraic integer 314
 do 905
Drinf, package 285ff.
DrinfMPhi 285
DrinfMProduct 286
DrinfMTau 287
 dual group
 – of an Abelian group 3
 dual homomorphism
 – between two Abelian groups 5
 dual module 608

E

e 888
Ecc, package 291–300
EccDecrypt 291
EccEncrypt 293
EccInit 295
EccIntPointMult 296
EccKangaroo 297
EccPointIsOnCurve 298
EccPointsAdd 299
EccRandomPoint 300
 echelon form
 – of a matrix 572
ECH0off 288
ECH0on 289
Ei 301
EisensteinSeries 302
 Element
 – of a number field 303
 – of an order 303

- element
 - of an Abelian group
 - – order 20
 - algebraic
 - – denominators 283
 - – index of 482
 - of an algebraic field
 - – norm 632
 - of an algebraic function field
 - – trace 875
 - of an algebraic number field
 - – norm 632
 - – trace 875
 - of finite field
 - – corresponding finite field 359
 - – creation 358
 - – generator 370
 - – minimal polynomial 362
 - – primitive 371
 - of a function field
 - – B^* 114
 - – n -th root 132
 - – basis change 124
 - – check if ideal element 121
 - – coefficients list 133
 - – creation of 110
 - – definition order 128
 - – denominator 116
 - – differentiation 82
 - – exact differential 71
 - – function field of 111
 - – Newton lifting 125
 - – norm 126
 - – numerator 127
 - – p^r th root 129
 - – residuum 131
 - – trace 137
 - – valuations at infinity 120
 - – minimal polynomial 123
 - – representation matrix 130
 - of module 616
- element
 - – valuation at a place 117
- elements
 - of an Abelian group
 - – next enumeration 24
 - – random element 21
 - order 284
- elif 903
- elliptic function field
 - creation 109
- else 903
- Elt** 303
- Elt, package* 303–349
- EltAbs** 305
- EltAbsLogHeight** 306
- EltApproximation** 307
- EltAutomorphism** 308
- EltCharPoly** 310
- EltCon** 312
- EltDen** 313
- EltDivisors** 314
- EltExcepUnitOrbit** 315
- EltFactor** 316
- EltIdealReduce** 317
- EltIndex** 318
- EltIsInIdeal** 319
- EltIsInt** 320
- EltIsIntegral** 321
- EltIsPrimitive** 322
- EltListAbsDisc** 323
- EltListToMat** 324
- EltLogs** 325
- EltMatToList** 326
- EltMinkowski** 329
- EltMinPoly** 327
- EltMove** 330
- EltNewtonLift** 331
- EltNorm** 332
- EltNumberReduce** 333
- EltOrder** 334
- EltPowerMod** 335
- EltPowerProduct** 336
- EltRayResidueRingRep** 337
- EltReconstruct** 338
- EltRepMat** 339

EltRoot	341	evaluation	897
EltSimplify	343	– of a polynomial	
EltToFFE	344	– – at value	353
EltToList	345	exceptional unit, orbit of	315
EltTrace	346	execution	901
EltUnitDecompose	347	Exp	354
EltValuation	349	exponent	
embedding		– of an Abelian group	26
– integer into a finite field \mathbb{F}_q	358	exponential	
end	906	– p-adic	810
enumerated element	549	extended	
enumeration environment reset	553	– Euclidean algorithm	504
enumeration init		extended gcd	
– Abelian group elements	23	– of two polynomials	763
– ideals of an order	460		
– quotients of a shape	387	F	
enumeration		Factor	381
– Abelian group		Factorial	383
– – next element	24	Factorization	
– of Abelian group elements	23	– of an algebraic number	316
– elements of an Abelian group	23	factorization	
– ideals of an order		– of a function field ideal	149
– – init	460	– of an integer	490
– next element of an Abelian group	24	– of integers	381
– next ideal of an order	461	– of a polynomial	740
– next quotient of a shape	389	– of polynomials	381
– order		– of principal ideal	381
– – next ideal	461	FF	355
– quotients of a shape		<i>FF</i> , package	355–372
– – init	387	FFCreate	356
– shape		FFelt	358
– – next quotient	389	FFeltFF	359
environment	906	FFeltIsZero	360
equality		FFeltLog	361
– of records	948	FFeltMinPoly	362
equation		FFeltMove	363
– solve an	864	FFeltNorm	364
equation order	636	FFeltRoot	365
– of an order	663	FFeltToInt	366
Euclidean algorithm	504	FFeltToList	367
EulerGamma	350	FFeltTrace	368
EutacticCoef	351	FFEmbed	369
Eval	353	FFToElt	357

- FFGenerator**..... 370
- FFPrimitiveElt**..... 371
- FFSize**..... 372
- fi 903
- field
 - finite
 - – basis..... 367
 - – convert element of prime field to integer... 366
 - – corresponds to element..... 359
 - – creation of an element 358
 - – creation 355
 - – degree of..... 372
 - – discrete logarithm..... 361
 - – embedding of an element..... 358
 - – generator 370
 - – minimal polynomial of an element..... 362
 - – move element into another finite field . 363
 - – norm
 - – – of an element 364
 - – primitive element 371
 - – root of an element 365
 - – trace
 - – – of an element 368
 - is finite 515
 - testing for finite..... 515
- field automorphism
 - corresponding group automorphism 53
- fields
 - maximal central field..... 385
 - maximal factor group 385
 - ray class field
 - – is Abelian over \mathbb{Q} 828, 833
 - – is central extension 835
 - – is commutative over \mathbb{Q} 828, 833
 - – get splitting field..... 838
 - – maximal central field 385
 - – is normal extension 837
- file
 - open..... 634
- FilePosition**..... 384
- fincke constant 665
- find
 - an element in a list 925
 - an element in a sorted list..... 926
- Find, package* 385–389
- FindMaximalCentralField** 385
- FindQuotientOfShapeEnumInit**..... 387
- FindQuotientOfShapeEnumNext**..... 389
- finite Abelian group \nearrow Abelian group..... 23
- finite field
 - basis..... 367
 - convert element of prime field to integer . 366
 - correspond to element..... 359
 - creation..... 355
 - degree of..... 372
 - discrete logarithm of an element 361
 - element creation 358
 - embedding of an element..... 358
 - generator 370
 - minimal polynomial
 - – of an element 362
 - move element
 - – into another finite field..... 363
 - moving an algebraic element into a f. f. . 344
 - norm
 - – of an element 364
 - primitive element 371
 - root
 - – of an element 365
 - test on finite field 515
 - trace
 - – of an element 368
- finite fields
 - number of prime polynomials 754
- fix point group 7
- fld-format
 - close file after reading..... 270
 - open file for reading..... 634
 - read order in fld-format 373
 - set or read file position of fld-format files 384
 - write order in..... 380
- fld-format for database
 - read order 379
- FLDin**..... 373
- FLDin_DB**..... 379
- FLDout**..... 380

- Floor** 390
- for 905
- fractional ideal
 - denominator of a 283
 - numerators 633
- function field
 - approximating element 112
 - bound for the number of places of degree one
220
 - canonical divisor
 - computation 60
 - class group
 - computation 61
 - generators 64
 - generators degree bound 62
 - generators degree bound (strong) 63
 - p -rank 65
 - representation in given generators 87
 - class number
 - approximation 66
 - bound 67
 - constant field 68
 - dimension 83
 - size 69
 - creation 59
 - defining polynomial 189, 224
 - degree
 - of the algebraic extension 70
 - different
 - computation 80
 - degree 81
 - differential
 - Cartier operator 73f.
 - the definition function field 72
 - divisor of 75
 - of the first kind 76
 - residuum 77
 - discriminant
 - of an order 284
 - divisor
 - basis of a Riemann-Roch space 93
 - corresponding ideals 91
 - creation 84
 - degree 88
 - denominator 90
 - dimension 99
 - field it belongs to 86
 - norm 100
 - number of (n,m) -smooth divisors 107
 - numerator 101
 - places and exponents 102
 - ratio of numbers of smooth divisors ... 108
 - reduced 103
 - riemann-roch space 94
 - divisor (large)
 - basis 98
 - divisor of degree one
 - computation of one 89
 - element
 - B^* 114
 - characteristic polynomial 115
 - creation 110
 - denominator from representation matrix...
122
 - differentiation 82
 - exact differential 71
 - Newton lifting 125
 - p^r th root 129
 - residue class field representative 134
 - valuation at a place 117
 - valuation 138
 - minimal polynomial 123
 - representation matrix 130
 - residuum 131
 - elliptic
 - creation 109
 - equation order
 - Dedekind-Test 176
 - finite 179
 - infinite 180
 - extension
 - absolute or not 160
 - fundamental units 244
 - gap numbers
 - computation 139
 - genus 140

- Hasse-Witt invariant 141
- hermitian
 - creation 143
- ideal
 - corresponding place 155
 - ideal of a
 - basis 146
 - ideal class group
 - computation 148
 - Ihara bound 157
 - indeterminate T 245
 - indeterminate y 246
 - indeterminate as order element 118f.
 - index
 - of equation order in given order 181
 - index of an order 482
 - infinite place
 - test if tamely ramified 242
 - irreducible polynomial
 - test 225
 - L -polynomial 164
 - reduced 166
 - L -polynomial lift 165
 - lift element of residue class field 231
 - linear series
 - enumeration 169
 - enumeration test 170
 - maximal order
 - finite 185
 - fundamental units 244
 - infinite 186
 - norm of an element 632
 - order
 - 0-reduced basis 190
 - basis 173
 - corresponding function field 172
 - defining polynomial 189
 - degree over its coefficient ring 177
 - discriminant 178
 - finite or infinite 182
 - fundamental units 244
 - maximal overorder 187
 - place corresponding to given ideal 155
 - regulator 230
 - basis 174
 - place
 - corresponding function field 197
 - corresponding ideal 200
 - corresponding order 203
 - decomposition 209
 - degree 199
 - finite or infinite 201
 - inverse prime element 198
 - minimum 202
 - prime element 204
 - ramification index 205
 - residue class field 208
 - residue degree 207
 - test if tamely ramified 242
 - places of degree one
 - bound 220
 - polynomial
 - irreducible test 226f.
 - polynomial algebra $k[T][y]$
 - creation 159
 - rational function
 - derivation 801
 - p -th root 803
 - regulator 230
 - S -regulator
 - computation 237
 - S -units
 - basis 237
 - Serre bound 240
 - signature 241
 - strong approximation 112
 - test if it is global 162
 - test if it is not global 163
 - trace of an element 875
 - unit group
 - computation 148
 - unit rank 243
 - Weil differentials
 - basis 78
 - divisor
 - support 106

- order
- – some basis values 175
- function field element
- place
- – residue class field representative 134
- function 906
- fundamental units
- of a number field 726
- of an order 726

G

Galois group 392

- adding and removing of possible groups . 400
- block system 394
- name 409
- number 412
- possible groups as subgroup lattice 414
- setting without computation 399

Galois group computation

- internal information 395

Galois groups

- modulo computing 404

Galois 392

Galois, package 392–417

GaloisBlocks 394

GaloisGlobals 395

GaloisGroupKnown 397

GaloisGroupOrder 398

GaloisGroupSet 399

GaloisGroupsPossible 400

GaloisMissionS 403

GaloisModulo 404

GaloisMSetPol 401

GaloisMSumPol 402

GaloisNumberToName 405

GaloisRing 406

GaloisRoots 407

GaloisSymb 408f.

GaloisSymbT 411f.

GaloisT 413

GaloisTree 414

GaloisTreeRoots 416

GaloisTwoSequencePol 417

Gamma 418

gap numbers

- computation 139

Gcd 419

gcd

- extended
- – of two polynomials 763
- of ideals 419
- of integers 419
- of polynomials 419
- of two integers 491
- of two polynomials 741

generation

- of a lattice 537
- of a lattice element 543

generator

- of finite field 370

generators

- of an Abelian group 27, 42
- – minimal number 38
- of a function field ideal 150

genus

- of a function field 140

GetEnvironment 420

GPIn 391

Gram matrix 556

group

- Galois 392

H

Hasse-Witt invariant

- of a function field 141

height

- absolute logarithm h. of an algebraic number
306

Hermite normal form

- of a function field ideal basis 147

HermiteUpperBound 421

Hermitian function field

- creation 143

- homomorphism
 - of an Abelian group
 - image..... 30
 - kernel..... 31
 - of orders..... 669
- homomorphism group
 - of an Abelian group..... 48
- HurwitzZeta**..... 422
- I**
- Ideal**..... 423
- ideal
 - check for integral property..... 447
 - check for primality..... 448
 - chinese remainder for..... 269
 - class representation..... 435
 - decomposition of a principal ideal..... 316
 - degree of inertia..... 438
 - of a function field
 - basis in Hermite normal form..... 147
 - basis..... 146
 - corresponding divisor..... 91
 - definition order..... 154
 - factorization..... 149
 - function field of..... 145
 - generators..... 150
 - is known to be prime..... 152
 - norm..... 153
 - primality test..... 151
 - two element representation..... 144
 - valuation..... 156
 - gcd..... 419
 - in the integers
 - generation..... 886
 - is integral..... 447
 - lcm..... 560
 - minimum..... 600
 - move into given order..... 627
 - norm of an..... 632
 - of an order
 - next enumeration..... 461
 - is prime..... 448
 - prime ideal
 - degree of inertia..... 438
 - ramification index..... 465
 - principal
 - test..... 449
 - ramification index..... 465
 - remainder..... 889
 - representation of an i. in the ray class group
466
 - test on integral property..... 447
 - test on primality..... 448
 - with minimal norm
 - in an order..... 681
- ideal class group
 - of a function field
 - computation..... 148
- ideal representation of a module..... 610
- Ideal2EltAssure**..... 424
- Ideal2EltIntAssure**..... 425
- Ideal2EltKnown**..... 426
- Ideal2EltNormalAssure**..... 427
- Ideal2EltNormalKnown**..... 428
- Ideal*, package..... 423–473
- IdealAutomorphism**..... 429
- IdealBasis**..... 430
- IdealBasisKnown**..... 431
- IdealBasisLowerHNF**..... 432
- IdealBasisUpperHNF**..... 433
- IdealChineseRemainder**..... 434
- IdealClassRep**..... 435
- IdealCollection**..... 437
- IdealDegree**..... 438
- IdealDen**..... 439
- IdealDivisors**..... 440
- IdealFactor**..... 441
- IdealGen**..... 442
- IdealGenerators**..... 443
- IdealIdempotents**..... 444
- IdealImprove**..... 445
- IdealIntegrity**..... 446
- IdealIsIntegral**..... 447
- IdealIsPrime**..... 448
- IdealIsPrincipal**..... 449

IdealLcm 451
IdealLLL 450
IdealLowerHNFTrans 452
IdealMakeCoprime 453
IdealMakeInvCoprime 454
IdealMin 456
IdealMove 457
IdealNorm 458
IdealOrder 459
IdealPrimeCountInit 460
IdealPrimeCountNext 461
IdealPrimeElt 463
IdealRadical 464
IdealRamIndex 465
IdealRayClassRep 466
IdealRemainderSet 467
IdealResidueField 468
IdealResidueFieldIsomorphism 469
IdealRingOfMultipliers 470
ideals
– of an order
– – enumeration init 460
IdealUpperHNFTrans 471
IdealValuation 472
IdealWithNorm 473
IdemLift 474
if 903
if statement 903
Im 475
Im, package 475–479
image
– of a group homomorphism 30
ImQuadFormCreate 476
ImQuadHilbert 477
ImQuadRayField 479
in
– for records 951
independent units
– of a number field 727
– of an order 727
Index 482

index
– of an alff order 482
– of an algebraic element 482
– of an order 482
– of a subgroup 33
– – of an Abelian group 33
index of speciality
– of a divisor 97
index of a suborder 318
infinite places of orders
– approximation 826
Infty, package 483–486
InftyGcd 483
InftyLcm 484
InftyQuotRem 485
InftyVal 486
Insert 487
IntDivisors 488
integer
– check 518
– check for primality 492
– check for square 493
– chinese remainder for 269
– divisors of an integer 488
– factorization 381, 490
– gcd 419
– gcd of two integers 491
– Jacobi symbol 533
– lcm 560
– lcm of two integers 494
– move into given order 627
– p -adic valuation of an integer 503
– power modulo an integer 497
– is prime 526
– prime divisors 498
– quotient of two 499
– random 500
– remainder 889
– – division of two integers 876
– root 501
Integer, package 488–506
integers
– ideal 886
– valuation of i 349

integral matrix
 – determinant 582
IntegralPoints 506
 intersection
 – of modules 611
 – of two Abelian groups 34
 – of sets 940
IntEulerPhi 489
IntFactor 490
IntGcd 491
IntIsPrime 492
IntIsSquare 493
IntLcm 494
IntMoebiusMy 495f.
IntPowerMod 497
IntPrimeDivisors 498
IntQuo 499
IntRandomBits 500
IntRoot 501
IntToChar 502
IntValuation 503
IntXGcd 504
IntZeta 505
 irreducibility test
 – of a polynomial 743
Is, package 507–531
IsAlff 507
IsAlffDiff 508
IsAlffDivisor 509
IsAlffElt 510
IsAlffPlace 511
IsChar 512
IsEcc 513
IsElt 514
IsFF 515
IsFFElt 516
IsIdeal 517
IsInt 518
IsLat 519
IsMat 520, 522
IsModule 523
IsOrder 524
IsPoly 525
IsPrime 526

IsQf 527
IsQp 528
IsQpElt 529
IsRecType 530
IsThue 531
 iterate a function over a list 936

J

Jacobi symbol
 – of two integers 533
JacobiSymbol 533
JBessel 532

K

KASHLEVEL 534
KBessel 535

kernel
 – of a group homomorphism 31

L

L-polynomial
 – of a function field 164, 166
 – – lift 165
Lat 537
Lat, package 537–559
LatBasis 540
LatCholesky 541
LatDisc 542
LatElt 543
LatEltLength 545
LatEltToList 546
LatEltVec 547
LatEnum 548
LatEnumElt 549
LatEnumLowerBound 550
LatEnumPrec 551
LatEnumRefVec 552
LatEnumReset 553
LatEnumUpperBound 554

- LatFinckeReduce** 555
 - LatGram** 556
 - LatLLL** 557
 - LatShortestElt** 558
 - LatSuccMins** 559
 - lattice
 - basis 540
 - basis reduction for enumeration 555
 - coefficient vector 546
 - discriminant 284, 542
 - enumerated element 549
 - enumeration environment 553
 - enumeration 548
 - generation of an element 543
 - Gram matrix 556
 - LLL basis reduction 557
 - lower bound for enumeration 550
 - minimum 558
 - norm of element 545
 - precision for enumeration 551
 - reference vector for enumeration 552
 - shortest vector(s) 558
 - successive minima 559
 - upper bound for enumeration 554
 - vector embedded into Euclidean space... 547
 - vectors of bounded norm 548
 - generate 537
 - Lcm** 560
 - lcm
 - of an ideal 560
 - of an integer 560
 - of a polynomial 560
 - of two integers 494
 - least positive common multiplier
 - of two integers 494
 - length
 - of a list 917
 - Li** 561
 - lifting
 - Newton l. of an algebraic element 331
 - linear series
 - enumeration envelopment 169
 - enumeration
 - – current divisor 167
 - enumeration test 170
 - List
 - of algebraic numbers converting to a matrix . 324
 - list
 - of all open files
 - – for debugging 536
 - of divisors of integer 488
 - List, package* 562ff.
 - ListApplyListAdd** 562
 - ListApplyMatAdd** 563
 - lists
 - of coefficients of a polynomial 762
 - ListSplit** 564
 - LLL
 - lattice basis reduction 557
 - reduced basis
 - – of an order 677
 - LLL-reduced
 - units
 - – of an order 729
 - local 906
 - LOFILES** 536
 - Log** 565
 - logarithm
 - absolute l. height of an algebraic number 306
 - discrete
 - – of an element of a finite field 361
 - p-adic 811
 - for loop 905
 - loop
 - for 905
 - repeat 904
 - while 904
- ## M
- map 914
 - between orders 669
 - Mat** 566
 - Mat, package* 566–599
 - MatCharPoly** 567

- MatCoef** 568
- MatCols** 569
- MatDet** 570
- MatDiag** 571
- MatEchelon** 572
- MatElt** 573
- MatHermiteColLower** 575
- MatHermiteColLowerTrans** 576
- MatHermiteColModUpper** 577
- MatHermiteColUpper** 578
- MatHermiteColUpperTrans** 579
- MatHermiteRowMod** 580
- MatId** 581
- MatIndex** 582
- MatInv** 583
- MatKernel** 584
- MatLLL** 586
- MatMinPoly** 588
- MatMLLL** 587
- MatMove** 589
- matrix
 - characteristic polynomial of a 264
 - creation 566
 - echelon form 572
 - generated by lists of algebraic number ... 324
 - move into given ring 627
 - trace of a 875
- MatRows** 590
- MatSmith** 591
- MatSmithTrans** 592
- MatSolve** 593
- MatSym** 594
- MatSymDiag** 595
- MatToColList** 596
- MatToRowList** 597
- MatTrace** 598
- MatTrans** 599
- maximal central field 385
- maximal order 678
- maximum
 - of integers 935
 - of a list 935
- membership test
 - for records 951
- method
 - of Cholesky 541
- Min** 600
- Min, package* 600ff.
- minimal
 - vector 602
- minimal number
 - of generators
 - – of an Abelian group 38
- minimal polynomial
 - of an algebraic element 601
 - of a finite field element 362
 - of a matrix 601
- minimal vectors
 - in a lattice 558
- minimum
 - ideal 600
 - of integers 935
 - of a list 935
- minkowski bound 682
- MinPoly** 601
- MinVec** 602
- mod** 889
- Module** 604
- module
 - changing of coefficient order 618
 - coefficient order 621
 - denominator 606
 - determinant 607
 - discriminant 323
 - dual module 608
 - intersection 611
 - normal form 619
 - reduction 617
 - representation 610, 615
 - Steinitz normal form 623
 - Steinitz representation 623
 - sum 625
 - triangular representation 619
 - trivial 609
 - union 625
 - move into given order 627
- module index 318

Module, package 604–625
ModuleDen 606
ModuleDet 607
ModuleDual 608
ModuleId 609
ModuleIdeals 610
ModuleIntersection 611
ModuleIntersectionVS 613
ModuleMap 614
ModuleMatrix 615
ModuleMember 616
ModuleModul 617
ModuleMove 618
ModuleNF 619
ModuleOrder 621
ModuleSmith 622
ModuleSteinitz 623
ModuleUnion 625
 modulo
 – algebraic element modulo an ideal 317
 – algebraic element modulo an integer 333
 – power
 – of integers 497
 modulo computing of Galois groups 404
Move 627
 move
 – elements
 – of a finite field into another finite field 363
 – algebraic numbers
 – into given order 627
 – ideals
 – into given order 627
 – integer
 – into given order 627
 – matrix
 – into given ring 627
 – module
 – into given order 627
 moving
 – algebraic element into a finite field 344
 – of modules 618
MultiCounterInc 628
MultiCounterInit 629

multilinear mapping
 – of Abelian groups 39
 multimorphism group 50
 – of a list of groups 50
 multiplicative group
 – of a residue class ring 846
 – – element from representation 849
 – – generators 848
 – – representation of an element 337
 multiply
 – the elements of a list 934
 multisets 937

N

Name
 – of Galois group 405
 name
 – of an Abelian group 41
 name of Galois group 409
 newline 895
 Newton lift
 – of an algebraic element 331
 Newton lifting
 – of a function field element 125
NextPrime 630
Nice 631
Norm 632
 norm
 – of an algebraic function field element 632
 – of an algebraic number 632
 – of a divisor 632
 – of a function field 100
 – of an element
 – of a finite field 364
 – of a function field element 126
 – of a function field ideal 153
 – of an ideal 632
 – of a lattice vector 545
 – of a polynomial 632, 751
 norm equation 683
 normal form
 – of module 619
 – module in Steinitz form 623

- Num** 633
- number
 - of generators
 - – of an Abelian group 42
 - – of an Abelian group, minimal 38
 - of prime polynomials
 - – over finite fields 754
 - of torsion units
 - – of an order 717
 - of elements in a list 929
- number field
 - bach bound 646
 - creation 636
 - fundamental units 726
 - maximal order 678
 - minkowski bound 682
 - p-maximal order 685
 - regulator 694
 - regulator bound 696
 - ring of integers 678
 - signature 709
 - splitting field 712
 - units
 - – independent 727
- Number of Galois group 412
- numerator
 - of an algebraic element 633
 - of a divisor 101
 - of an fractional ideal 633
 - of a function field element 127
 - of a rational function 802
 - of a rational number 633
- O**
- od 905
- Open** 634
- open
 - file 634
- operations
 - for lists 924
- orbit of an exceptional unit 315
- Order** 636
 - of module 621
- order
 - of an Abelian group 43
 - absolute 639
 - – with small index 707
 - absolute degree 661
 - bach bound 646
 - basis 647
 - corresponding function field 172
 - degree 660
 - discriminant 284, 662
 - equation 636
 - equation order 663
 - fincke constant 665
 - finite equation
 - – of a function field 179
 - finite maximal
 - – of a function field 185
 - function field
 - – defining polynomial 189
 - – finite or infinite 182
 - – index of equation order 181
 - – maximal overorder 187
 - fundamental units 726
 - homomorphism 669
 - ideal
 - – next enumeration 461
 - – ring of multipliers 470
 - – with minimal norm 681
 - ideals
 - – enumeration init 460
 - – index of an 482
 - infinite equation
 - – of a function field 180
 - infinite maximal
 - – of a function field 186
 - LLL-reduced basis 677
 - map 669
 - is maximal 671
 - maximal 678
 - maximal overorder
 - – of a function field order 187

- merge..... 680
- minkowski bound..... 682
- norm equation..... 683
- p-maximal..... 685
- positive S-units..... 702
- power basis
 - test..... 648
- precision..... 689
- reading pari/gp-format..... 391
- reduce index..... 705
- regulator..... 694
- regulator bound..... 696
- is relative..... 672
- relative
 - coefficient ideals..... 656
- relative basis
 - test..... 649
- relative representation..... 700
- set maximal flag..... 703
- signature..... 709
- simplified representation..... 710
- splitting field..... 712
- sub..... 713
- test..... 524
- torsion unit..... 716
 - rank..... 717
 - set..... 704
- trace matrix..... 718
- transformation matrix..... 719
- units
 - are fundamental..... 721
 - independent..... 727
 - LLL-reduced..... 729
 - merge..... 730
 - p-maximal..... 731
- Order, package*..... 636–731
- OrderAbs*..... 639
- OrderAutomorphisms*..... 642
- OrderAutomorphismsAbel*..... 644
- OrderAutomorphismsNormal*..... 645
- OrderBach*..... 646
- OrderBasis*..... 647
- OrderBasisIsPower*..... 648
- OrderBasisIsRel*..... 649
- OrderClassGroup*..... 650
- OrderClassGroupCheck*..... 652
- OrderClassGroupCyclicFactors*..... 653
- OrderClassGroupCyclicFactorsPrincipal*... 654
- OrderClassGroupFactorBasisProve*..... 655
- OrderCoefIdeals*..... 656
- OrderCoefOrder*..... 657
- OrderCyclotomic*..... 658
- OrderCyclotomicRealSubfield*..... 659
- OrderDeg*..... 660
- OrderDegAbs*..... 661
- OrderDisc*..... 662
- OrderEquationOrder*..... 663
- OrderExcepSequence*..... 664
- OrderFincke*..... 665
- OrderGalois*..... 666
- OrderIndex*..... 667
- OrderIndexFormEquation*..... 668
- ordering
 - of records..... 948
- Ordering of roots..... 407
- OrderInstallHom*..... 669
- OrderIsMaximal*..... 671
- OrderIsRelative*..... 672
- OrderIsSubfield*..... 673
- OrderKextGenAbs*..... 674
- OrderKextGenRel*..... 675
- OrderKextModularPower*..... 676
- OrderLLL*..... 677
- OrderMaximal*..... 678
- OrderMerge*..... 680
- OrderMinIdeal*..... 681
- OrderMinkowski*..... 682
- OrderNormEquation*..... 683
- OrderPMaximal*..... 685
- OrderPoly*..... 687
- OrderPolyHensellift*..... 688
- OrderPrec*..... 689
- OrderPrintFlags*..... 691
- OrderReg*..... 694
- OrderRegLowBound*..... 696
- OrderRelativeOrder*..... 700

<code>OrderRelNormEq</code>	697	<code>p</code> -maximal order	685
<code>OrderRelUnits</code>	698	<code>p</code> -maximal unit overgroup.....	731
<code>OrderSetMaximal</code>	703	<code>p</code> -th root	
<code>OrderSetTorsionUnit</code>	704	– of rational function	803
<code>OrderShort</code>	705	<code>padic</code>	
<code>OrderShortAbs</code>	707	– element creation	809
<code>OrderSig</code>	709	<code>pi</code>	892
<code>OrderSimplify</code>	710	<code>place</code>	
<code>OrderSplittingField</code>	712	– corresponding function field.....	197
<code>OrderSubfield</code>	714	– degree	199
<code>OrderSubfieldSub</code>	715	– ramification index.....	205
<code>OrderSubOrder</code>	713	– residue class field	208
<code>OrderSUnits</code>	701	– residue degree.....	207
<code>OrderSUnitsPositive</code>	702	<code>places</code>	
<code>OrderTorsionUnit</code>	716	– of degree one	
<code>OrderTorsionUnitRank</code>	717	– – Ihara bound	157
<code>OrderTraceMat</code>	718	<code>Poly</code>	734
<code>OrderTransformationMatrix</code>	719	<i>Poly, package</i>	734–764
<code>OrderUnitsAreFund</code>	721	<code>PolyAlg</code>	735
<code>OrderUnitsEquation</code>	723	<code>PolyAlgCoef</code>	736
<code>OrderUnitsExcep</code>	724	<code>PolyDeg</code>	737
<code>OrderUnitsFund</code>	726	<code>PolyDeriv</code>	738
<code>OrderUnitsIndep</code>	727	<code>PolyDisc</code>	739
<code>OrderUnitsLLL</code>	729	<code>PolyFactor</code>	740
<code>OrderUnitsMerge</code>	730	<code>PolyGcd</code>	741
<code>OrderUnitsPFund</code>	731	<code>PolyHensellift</code>	742
<code>output</code>		<code>PolyIsIrreducible</code>	743
– switch stdout to file.....	289	<code>PolyIsSquarefree</code>	744
		<code>PolyIsZero</code>	745
P		<code>PolyMakeMonicInOrder</code>	746
<code>p</code> -adic		<code>PolyMakeMonicInZ</code>	747
– exponential	810	<code>PolyMove</code>	748
– logarithm	811	<code>PolyMoveIntegral</code>	749
– square root	814	<code>PolyNewtonLift</code>	750
– valuation of an integer	503	<code>is polynomial</code>	525
– valuation	815	<code>polynomial</code>	
<code>p</code> -adic field		– characteristic	
– creation.....	806	– – of an algebraic element	264
– precision.....	812	– – of a matrix	264
– prime	813	– characteristic <code>p</code> . of an algebraic element.	310
<code>p</code> -adic integers		– chinese remainder for	269
– factorization of a polynomial.....	759	– factorization of.....	381
		– Galois group.....	392
		– gcd	419

- irreducible
 - test 225
- irreducible test 226f.
- lcm 560
- minimal
 - of an algebraic element 601
 - of a matrix 601
- norm of a 632
- remainder 889
- polynomial algebra
 - coefficient ring 799
 - name of the (outmost) variable 797
 - transcendence degree 798
- polynomial with given roots 401f., 417
- polynomials
 - coefficient list 762
 - creation 734
 - degree of a polynomial 737
 - derivative 738
 - discriminant 739
 - discriminant of 284
 - evaluation
 - at value 353
 - extended gcd 763
 - factorization 740
 - factorization over p -adic integers 759
 - gcd 741
 - is irreducible 743
 - moving to integral polynomial 749
 - norm 751
 - over finite fields
 - number of prime polynomials 754
 - is polynomial 525
 - power product 752
 - quotient and remainder 756
 - random prime polynomial 755
 - reduced discriminant 757
 - representation in a different algebra 748
 - resultant 758
 - roots 764
 - signature 760
 - is square free 744
 - swap variables 761
 - test on irreducibility 743
 - test on square free property 744
 - is zero polynomial 745
 - zeros 764
 - PolyNorm** 751
 - PolyPowerMod** 752
 - PolyPrimeList** 753
 - PolyPrimeNum** 754
 - PolyPrimeRandom** 755
 - PolyQuotRem** 756
 - PolyRedDisc** 757
 - PolyResultant** 758
 - PolyRoundFour** 759
 - PolySig** 760
 - PolySwapVars** 761
 - PolyToList** 762
 - PolyXGcd** 763
 - PolyZeros** 764
 - power
 - of an integer modulo an integer 497
 - of records 950
 - power basis
 - order
 - test 648
 - Prec** 765
 - precedence 900
 - precision
 - lattice enumeration 551
 - of an order 689
 - of a p -adic field 812
 - prime
 - check integer for 492
 - finite field
 - convert element of to integer 366
 - integer is prime 526
 - rational
 - greater a given rational integer 630
 - prime divisors
 - of an integer 498
 - prime ideal decomposition of an integer ... 316
 - primitive
 - elements
 - of a finite field 371
 - primitive element of a ray class field 832

primitive element test 322
 principal ideal
 – decomposition in prime ideals 316
 – factorization of 381
PRINTLEVEL 732
 printlevel
 – of Abelian groups 44
 product
 – of records 950
Pvm, package 766–795
PvmClear 766
PvmExit 767
PvmGet 768
PvmGetAnswer 769
PvmGetB 770
PvmGetEval 771
PvmInit 772
PvmKashIsSlave 773
PvmLengthOfQueue 774
PvmMaxRestartSlave 775
PvmMaxRetransmitJob 776
PvmPread 777
PvmRead 778
PvmSecurity 779
PvmSendAll 780
PvmSendLast 781
PvmSendNext 782
PvmSetPrintLevel 783
PvmShowBroadcastJobs 784
PvmSlaveError 785
PvmSlaveInfo 786
PvmSlavePrint 787
PvmSlaveSend 788
PvmStartSlave 789
PvmStopSlave 790
PvmStoreOrders 791
PvmUse 792
PvmUseMastersHost 793
PvmUseMsg 794
PvmUseWatch 795

Q

Q 796
Qf, package 797–805
QfeDen 800
QfeDeriv 801
QfeNum 802
QfePthRoot 803
QfeQf 804
QfeVal 805
QfName 797
QfRank 798
QfScalarRing 799
Qp 806
Qp, package 806–815
QpElt 807
QpEltQp 808
QpEltToQ 809
QpExp 810
QpLog 811
QpPrec 812
QpPrime 813
QpSqrt 814
QpValuation 815
 query
 – for database 279
 quotient
 – of two integers 499
 – of records 950
 quotient group 52
 quotient and remainder
 – of two polynomials 756
QuotientField 816
 quotients
 – enumeration init 387
 – next enumeration 389

R

R 817
 ramification index
 – of a function field place 205
 – of a prime ideal 465

- random
 - access
 - of fld-format files 384
 - integer 500
 - prime polynomial 755
- Random, package* 818–824
- RandomEcc* 818
- RandomElt* 819
- RandomIdeal* 820
- RandomMatrix* 822
- RandomOrder* 823
- RandomPoly* 824
- rank
 - of the torsion unit group
 - of an order 717
- rational
 - denominator of a 283
 - numerators of a 633
- rational function
 - definition quotient field 804
 - denominator 800
 - numerator 802
 - valuation 805
- rational prime
 - smallest greater a given rational integer . 630
- RationalReconstruct* 825
- ray class field 476
 - is Abelian over \mathbb{Q} 828, 833
 - artin automorphism 830
 - automorphisms and primitive element ... 832
 - is central extension 835
 - is commutative over \mathbb{Q} 828, 833
 - discriminant and signature 845
 - get splitting field 838
 - maximal central field 385
 - is normal extension 837
- ray class group 839
 - conductor 843
 - test 844
 - generation of group automorphism 53
 - generators 841
 - representation of an ideal 466
- Ray, package* 826–850
- RayCantoneseRemainder* 826
- RayClassFieldAbelianTest* 828
- RayClassFieldArtin* 830
- RayClassFieldAuto* 832
- RayClassFieldIsAbelian* 833
- RayClassFieldIsCentral* 835
- RayClassFieldIsNormal* 837
- RayClassFieldSplittingField* 838
- RayClassGroup* 839
- RayClassGroupCyclicFactors* 841
- RayClassGroupToAbelianGroup* 842
- RayConductor* 843
- RayConductorTest* 844
- RayDiscSig* 845
- RayResidueRing* 846
- RayResidueRingCyclicFactors* 848
- RayResidueRingRepToElt* 849
- RayResidueRingToAbelianGroup* 850
- RCF \nearrow ray class field 835
- Re* 851
- Read* 852
- read
 - bag from open file 255
 - file
 - position 384
 - order
 - in database fld-format 379
 - in fld-format 373
 - order in pari/gp-format 391
- ReadLib* 853
- record
 - checks for 530
- recursion 906
- recursive functions 906
- reduced basis
 - for enumeration 555
 - of a lattice 557
- reduced discriminant
 - of a polynomial 757
- reduction
 - of an algebraic element modulo an integer ... 333
 - of a module 617

- regulator
 - of a function field
 - computation 237
 - of an order 694
 - bound 696
- relative order
 - absolute 639
 - absolute order
 - with small index 707
 - coefficient ideals 656
 - from an order 700
- remainder 889
 - division of two integers 876
- remove
 - an element from a set 939
- reopen
 - file 634
- repeat 904
- repeat loop 904
- representation of an ideal
 - ray class group 466
- representation of a module 615
- representation
 - of a polynomial
 - in a different polynomial algebra 748
 - reduced r. of algebraic elements 317
- residue class field
 - of a function field place 208
 - representative of a function field element 134
- residue class ring
 - multiplicative group 846
 - element from representation 849
 - generators 848
 - representation of an element 337
- residue degree
 - of a function field place 207
- residuum
 - of a differential
 - computation 77
 - of a function field element 131
- resultant
 - of two polynomials 758
- return 909
- reverse the elements of a list 928
- Riemann-Roch space
 - basis for a given divisor 93
- riemann-roch space
 - dimension 96
 - of a divisor 94
- ring of integers 678
- ring of multipliers of an ideal 470
- Root
 - ordering 407
- root
 - n -th root of a function field element 132
 - of an element
 - of a finite field 365
 - of an integer 501
- roots
 - of a polynomial 764
- Roots of a polynomial 407
- Round 854
- S**
- S-units
 - positive
 - of an order 702
- scope 897
- see Galois 666
- Serre bound
 - function field 240
- set
 - file
 - position 384
- shape
 - quotient
 - next enumeration 389
 - quotients
 - enumeration init 387
- shortest vector(s)
 - in a lattice 558
- signature
 - of an order 709
 - of a polynomial 760
 - of a ray class field 845

Simplex, package 858–861
SimplexElt 858
SimplexInit 859
SimplexNext 860
SimplexReInit 861
Sin 862
 size
 – of the constant field of a function field.... 69
Sleep 863
 Smith normal form
 – of an Abelian group 45
 SNF
 – of an Abelian group 45
Solve 864
 solve
 – equation 864
 sort a list 932
 space 895
 splitting field
 – of an order 712
SPrint 855
Sqrt 865
 square
 – check integer for 493
 square free property
 – of a polynomial 744
 square root
 – p-adic 814
SScan 856
 standard output
 – switch in file 289
 Steinitz normal form of a module 623
Subfield, package 866ff.
SubfieldAdd 866
SubfieldGet 867
SubfieldSetDegreeMax 868
 subgroup
 – of an Abelian group 56
 – – creation of a subgroup 56
 subgroup lattice of possible Galois groups . 414
 subgroup property 37
 suborder 713
 – index of s. 318

subtract
 – a set from another 941
 successive minima of a lattice 559
 sum
 – of modules 625
 – of records 950
 swap variables
 – of a bivariate polynomial 761
 switch
 – back to normal behaviour 288
 – output of stdout to file 289
 symbol
 – Jacobi symbol
 – – of two integers 533

T

tabulator 895
Tan 869
 tensor product 46
 – of a list of groups 46
 test
 – for a list 914
 – for list elements 953
 – for membership 925
 – for record elements 953
 – for records 953
 – for a set 938
 – for set equality 939
 – for subsets 941
 test on
 – algebraic number is element of an order . 321
 – algebraic number is integer 320f.
 – algebraic number is primitive 322
 – element of module 616
 testing
 – for finite field 515
 – integer 518
 – an object for Thue 531
 then 903
Thue 870
 – checks for 531
 thue equation 870
Thue, package 870–873

ThueEval 872
ThueSolve 873
Time 874
torsion unit
 – of an order 716
 – – rank 717
 – – set 704
Trace 875
trace
 – of an algebraic element 875
 – of an algebraic function field element 875
 – of an element
 – – of finite field 368
 – of a function field element 137
 – of a matrix 875
trace matrix
 – of an order 718
transcendence degree
 – of a polynomial algebra 798
transformation matrix
 – of an order 719
TrialDivision 876
Trunc 877
type
 – list 913
 – records 943

U

undefined function name 890f.
Union
 – of modules 625
union
 – of two Abelian groups 47
 – of sets 940
unit
 – decomposition 347
 – orbit of an exceptional u. 315
 – torsion
 – – of an order 716
 – – set 704

unit group
 – of a function field
 – – computation 148
units
 – of a function field
 – – basis 237
 – of an order
 – – are fundamental 721
 – – fundamental 726
 – – independent 727
 – – merge 730
 – – p-maximal 731
 – of and order
 – – LLL-reduced 729
until 904

V

Valuation 878
valuation
 – algebraic element with certain v. 307
 – of algebraic elements 349
 – at prime ideals 349
 – of a function field element 138
 – – at infinity 120
 – of a function field ideal 156
 – of integers 349
 – p-adic 815
 – – of an integer 503
 – in a rational function field 805
Vec 879
VecDotProduct 880
vector
 – minimal 602

W

WeierstrassP 881
WeilPairing 882
while 904
while loop 904
World 884

write

- bag to open file 256
- order in fld-format 380

Z

Z 885

- ideal 886

0-reduced basis

- function field order 190

zeros

- of a polynomial 764

ZIdealCreate 886

Zx 887