1549

Bell Laboratories

Cover Sheet for Technical Memorandum

The information contained herein is for the use of employees of Bell Laboratories and is not for publication (see GEI 13.9-3)

Title- Hi - Plus

Date- April 8, 1980

TM- 80-9544-1

Other Keywords-

Author(s)	Location and Room	Extension	Charging Case- 49059-6
R. J. Yanofchick	MH 5A-139	7380	Filing Case- 49059-1

ABSTRACT

Hi is an interactive hierarchical data management system that runs under the UNIX* operating system. It was originally built to experiment with small to moderate sized data bases and with a language similar to that of the UNIX text editor. The objective was to learn more about hierarchical data management and to evolve a system that could accomodate a variety of users.

As part of the process, new concepts are being added and old ones are being replaced. Some added concepts include the ability to reference specific fields in a record, the ability to easily define or change field delimiters, a simple mathematical package which includes addition, subtraction, multiplication, division, column summation, cross products of columns etc., the ability to specify an alternate output file (which may be a UNIX file or another device) and an alternate input file or files, the ability to extract sets of fields from different nodes in the hierarchy and create new records which can then be managed separately or added to the data base.

A general purpose, screen oriented Data Entry and Update System is available for use with the system.

Pages Text 21 Other 5 Total 26

No. Figures 2 No. Tables 0 No. Refs. 0

Address Label

Bell Laboratories

subject: Hi - Plus
Charging Case - 49059-6
Filing Case - 49059-1

date: April 8, 1980

from: R. J. Yanofchick
TM 80-9544-1

MEMORANDUM FOR FILE

INTRODUCTION

Hi[1] is an interactive hierarchical data management system that runs under the UNIX* operating system. It was originally built to experiment with small to moderate sized data bases and with a language similar to that of the UNIX text editor. The objective was to learn more about hierarchical data management and to evolve a system that could accommodate a variety of users. The system provides the capability to structure nodes or record types into a hierarchy and then to manipulate records in the hierarchy with a language similar to that of the UNIX text editor. In the original version of Hi every record in the data base is treated as a character string and the system does not explicitly distinguish fields in a record. Instead, the user addresses the contents of each record with the same regular expressions allowed in the UNIX text editor. The schema is completely specified by creating a file with the names of the records in a table of contents format which reflects their hierarchical structure. The system treats the hierarchy as a

1 Kayel, R.G., "Hi - An Interactive Hierarchical Data Management System for UNIX".

* UNIX is a trademark of Bell Laboratories.

flat file, allows M:N relationships between record types without physical redundancy, and facilitates dynamic restructuring of the hierarchy.

Hi-Plus subsumes the original system and provides several new concepts which allow it to satisfy the needs of a larger user community. They include the ability to reference specific fields in a record, the ability to easily define and change field delimiters, a simple mathematical package which includes addition, subtraction, multiplication, division, column summation, cross products of column, etc., the ability to specify an alternate output file (which may be a UNIX file or another device) and an alternate input file or files, the ability to extract sets of fields from different nodes in the hierarchy and create a new record which can then be managed separately or added to the data base.

Hi-Plus integrates well with UNIX. The output of a query is a structured stream of characters which can be used as input to other UNIX tools, grep, awk, etc., or to user supplied tools.

Shell procedures containing Hi query statements can be put together to make a simple but effective application or report generation subsystem. Current applications use shell procedures containing query statements to produce parameterized canned query and report generation facilities.

The query language provides the user with the power and flexibility to view data stored in nodes at different levels of a

hierarchy as though they were contained in a single record type. Consider a tree with a root node A having children B and C. The language features "extract" and "pipe" can be used to produce two files of the form b_1, a_1 and b_1, c_1 which can be fed to the relational "join" command of UNIX to produce b_1, a_1, c_1. On the other hand, these same two features of Hi-Plus can simulate this "join" directly by "extracting" and "piping" data to a new file containing records of the form a_1, b_1, c_1 directly from the stored hierarchy.

One application is now using this system to manage a data base of about 32K records. This data base currently contains about 500K bytes of data and is still growing.

NEW CONCEPTS

1. Fields and Field Delimiters

Each data record may be considered to be divided into fields. Fields are normally delimited by a tab character; however, the field delimiter may be changed, as described below. The command "del?" will display the value of the current field delimiter, and the command "delim" allows the user to specify or change the current delimiter. A delimiter may be any string of up to 12 characters. For example:

```
del?
 current delimiter is TAB
```

Change the current delimiter to the pattern 'pass='

```
delim
 delimiter = pass=
```

```
del?
 current delimiter is pass=
```

Fields are referred to as $1, $2, etc., where $1 is the first field, $2 is the second field. This concept of fields may be applied to both print commands (p and P) and all qualified sequencing commands (see Appendix 1 for a complete list of commands). The following examples make use of the presidential data base shown in Figure 1.

A record will not be printed unless the command "p" or "p" is given (the only difference is that the "P" will show the internal identifier for the record). The command "rpP" will get the first presidential record and print it without the internal identifier and then with the identifier.

```
rpP

pres:       Eisenhower        10-14-1890      03-28-1969      Rep
pres(1): Eisenhower     10-14-1890      03-28-1969      Rep
```

But suppose we only wanted to print the name and the party of the president , the command would be

```
rp$1,4P$1,4
```

```
pres: Eisenhower        10-14-1890      03-28-1969      Rep
      elect: 1952       votes=442       lsr=Stevenson   Dem
      elect: 1956       votes=457       lsr=Stevenson   Dem
      admin: 49 inaugdate=01-20-1953    vp=Nixon
      admin: 50 inaugdate=01-20-1957    vp=Nixon
            state: Alaska       pop=407000      votes=3
            state: Hawaii       pop=895000      votes=4
      cong: 83  SRep%=50        SDem%=49        HRep%=49        HDem%=50
      cong: 84  SRep%=49        SDem%=51        HRep%=47        HDem%=53
      cong: 85  SRep%=49        SDem%=51        HRep%=46        HDem%=54
      cong: 86  SRep%=34        SDem%=66        HRep%=35        HDem%=65
pres: Kennedy   05-29-1917      11-22-1963      Dem
      elect: 1960       votes=303       lsr=Nixon       Rep
      admin: 51 inaugdate=01-20-1961    vp=Johnson
      cong: 87  SRep%=36        SDem%=64        HRep%=40        HDem%=60
      cong: 88  SRep%=33        SDem%=67        HRep%=41        HDem%=59
pres: Johnson   08-27-1908      01-22-1973      Dem
      elect: 1964       votes=486       lsr=Goldwater   Rep
      admin: 52 inaugdate=11-22-1963    vp=NULL
      admin: 53 inaugdate=01-20-1965    vp=Humphrey
      cong: 88  SRep%=33        SDem%=67        HRep%=41        HDem%=59
      cong: 89  SRep%=32        SDem%=68        HRep%=33        HDem%=67
      cong: 90  SRep%=36        SDem%=64        HRep%=43        HDem%=57
pres: Nixon     01-09-1913      NULL    Rep
      elect: 1968       votes=301       lsr=Humphrey    Dem
      elect: 1968       votes=301       lsr=Wallace     3rdParty
      elect: 1972       votes=520       lsr=McGovern    Dem
      admin: 54 inaugdate=01-20-1969    vp=Agnew
      admin: 55 inaugdate=01-20-1973    vp=Agnew
      admin: 55 inaugdate=01-20-1973    vp=Ford
      cong: 91  SRep%=43        SDem%=57        HRep%=44        HDem%=56
      cong: 92  SRep%=44        SDem%=54        HRep%=41        HDem%=59
      cong: 93  SRep%=42        SDem%=56        HRep%=44        HDem%=56
pres: Ford      07-14-1913      NULL    Rep
      admin: 56 inaugdate=08-09-1974    vp=Rockefeller
      cong: 93  SRep%=42        SDem%=56        HRep%=44        HDem%=56
      cong: 94  SRep%=37        SDem%=60        HRep%=33        HDem%=66
```

Figure 1

which would produce

```
pres:
      (1) Eisenhower
      (4) Rep
 pres(1):
         (1) Eisenhower
         (4) Rep
```

Notice that we did not have to repeat the "$" for each field we printed. "$1,2,3,4" will reference fields 1,2,3 and 4.

To display the presidential records for all Republican presidents we would execute

```
g/$4=Rep/p
```

to produce

```
 pres: Eisenhower        10-14-1890       03-28-1969       Rep
 pres: Nixon     01-09-1913       NULL     Rep
 pres: Ford      07-14-1913       NULL     Rep
```

An alternate approach would be

```
g/$4!=Dem/p
```

which would exclude all Democratic presidents and produce

```
 pres: Eisenhower        10-14-1890       03-28-1969       Rep
 pres: Nixon     01-09-1913       NULL     Rep
```

```
pres: Ford      07-14-1913      NULL    Rep
```

The allowable conditional operators are: =, !=, >, >=, < and <= . The general form for a qualified search of this type is

```
/$i<conditional operator>/
```

2. Arithmetic Operations

Hi provides the user with a simple computational subsystem. This subsystem was designed to allow the user to apply a limited set of arithmetic operations to data stored in the data base or to perform stand-alone computations. The general form of a computational statement is:

(1) C [< expression >]

or

(2) C "label" [< expression >] .

The interpretation of the expression proceeds from left to right. An operator is executed if its operator precedence is greater than or equal to the operator to its right. Balancing parentheses may be used to alter the order of the execution.

The operations currently implemented are:

	Operator	Meaning
(1)	+	sum
(2)	- (underscore)	difference
(3)	*	product
(4)	/	quotient
(5)	!	summation

Operators 1 through 4 have the normal algebraic precedence. The summation (!) operator really has no precedence; it simply indicates to Hi that the numbered field must be summed before any other operator is applied.

The following examples make use of the data base shown in Figure 2. This data base has two nodes which will be used in the computations. At the Header node we will be interested in field 1, the company name and field 3, the product name. At the Movrev node we will be interested in field 2, the number of units installed during the period and field 3, the number of units removed from service during the period.

Consider the following query. Identify those companies which provide the product d100 and compute the net gain (or net loss) for this product during the year 1978. Net gain (or loss) can be computed by summing fields 2 and 3, then taking the difference, total field 2 - total field 3. The command

```
g/$3=d100/ p c{Movrev} g/.*/ C[!$2_!$3]
```

```
Header: sc 03 d100 01 17v+
      Movrev: 1q78 70 3 0 541 7
      Movrev: 2q78 98 2 0 618 7
      Movrev: 3q78 86 1 3 666 7
      Movrev: 4q78 94 3 3 714 7

Header: sw 02 d100 01 17v+
      Movrev: 1q78 75 3 0 451 8
      Movrev: 2q78 92 2 0 861 9
      Movrev: 3q78 80 1 0 555 6
      Movrev: 4q78 53 4 0 417 7
```

Figure 2

Which has the structure

(1) find each Header record having product d100
 (a) print the record
 (b) change to its' Movrev children
  (b.1) for each child sum fields 2 and 3
(2) subtract the sum of field 3 from the sum of field 2
 and print the result

would produce —

```
Header: sc 03 d100 01 17v+
Header: sw 02 d100 01 17v+
 629.000000
```

Using the second form of the compute statement shown above we can label the result with the following statement

```
g/$3=d100/ p c{Movrev} g/.*/ C"Net change for product d100
="[!$2_!$3]
```

to produce

```
Header: sc 03 d100 01 17v+
Header: sw 02 d100 01 17v+
Net change for product d100 = 629.000000
```

Another command which can be useful when performing field summation is the 'H' command. 'H' allows the user to specify field or column titles to be used when printing the results of a computation. The general form of the 'H' command is

H"<specified field titles>"

The results of field summations are printed in columns 1, 16, 32, 48, etc. . This should be kept in mind when trying to align titles. The following example uses the 'H' command in conjunction with a command to sum field 2, sum field 3 and then take their difference.

```
H"Installations Removals        Net Change"

g/$3=d100/ p c{Movrev} g/.*/ C[!$2!$3(!$2_!$3)]

 Header: sc 03 d100 01 17v+
 Header: sw 02 d100 01 17v+

 Installations  Removals        Net Change

 648.000000     19.000000       629.000000
```

As mentioned above this subsystem can also be used to perform stand-alone computations. For example, the command

```
C[2+2]
```

would produce

```
 4.000000
```

Again we can now label the result of the computation

```
C"Something is "[((1-2)*(4+5)/2)]

 to produce
```

```
Something is  4.500000
```

The 'H' command has no effect in stand-alone computation.

3. Alternate Output/Input Files

Sometimes the results of a set of interactive queries can be useful, as inputs to another program, or, with some minor reformatting as a small report. Hi provides a command 'pipe' which is similar in function to the UNIX 'tee' command. The form of this command is:

pipe < filename >

where <filename> can either be the name of a file, a path name (/usr/x/y/file), or another device (ttyxx). When in use the 'pipe' command routes all output from Hi to the specified file as well as the user terminal. If the file specified does not exist it will be created; if the file does exist, all current output will be appended at the end of the file. The 'pipe' command remains in effect until the end of the terminal session or until a 'depipe' command is issued. For example

pipe junk

all output from Hi will be copied to file junk.

depipe

stops output from being copied to file junk.

It is also sometimes useful to be able to accept input from a source other than the user terminal. Hi also provides the ability to specify a file from which input should be accepted. The form of this command is

use < filename >

The 'use' command can be especially useful if a specified set of Hi commands is to be executed repeatedly (e.g. canned queries), 'use' is recursive in that up to ten alternate input files can be open simultaneously. When a 'use' statement is encountered, Hi accepts input from the file specified until an end-of-file or another use command is found. When an end-of-file is encountered Hi checks to see if any other alternate files had been specified, if so input is accepted from the most recently specified file. This process continues until all alternate files have been used and control is returned to the user terminal.

4. Extracting Fields from Records

At times it is useful to be able to view a collection of data items which are stored at different levels of a hierarchy as though they were contained in a single record. Hi provides this capability with the command 'x'. The general form of this command is

x < $i1, i2, i3, ..., in >

where i1, i2, ..., in are the fields which are to be extracted from the current record. Referring once again to the presidential data base shown in Figure 1, suppose we wanted to create a

new record containing the presidents' name, his party affiliation, his administrations and the names of his vice-presidents. The data items of interest are stored at two different levels in this hierarchy, presidents' name and party are at the "pres" node, administrations and vice-presidents are at the "admin" node. To create the records as we want to see them we could execute:

```
g/.*/x$1,4 c{admin} g/.*/x$1,3 p
```

to produce

```
Eisenhower      Rep     49      vp=Nixon
Eisenhower      Rep     50      vp=Nixon
Kennedy         Dem     51      vp=Johnson
Johnson         Dem     52      vp=NULL
Johnson         Dem     53      vp=Humphrey
Nixon   Rep     54      vp=Agnew
Nixon   Rep     55      vp=Agnew
Nixon   Rep     55      vp=Ford
Ford    Rep     56      vp=Rockefeller
```

When used in conjunction with the 'pipe' command these records could be copied to a file and used in some later processing.

5. Some Other Useful Commands

5.1 Preorder Traversal

When dealing with a tree structure it is sometimes convenient to be able to traverse the tree in some orderly fashion. Hi provides the user with the ability to traverse the entire data base, or some subset of the data base in a preorder fashion. The command 'G' allows the user to start at any node of the tree and traverse the remainder of the tree in a preorder fashion. Figure 1 of this paper was obtained by positioning the cursor at the root node of the presidential data base and executing the 'G' command. If an interrupt is detected before 'G' reaches the bottom of the tree the cursor is positioned back to the node from which the command was issued. This allows the user to get a local view of the data base for reference purposes. For example, referring to Figure 1, suppose the cursor was on the node 'pres: Kennedy...' and the user issued the 'G' command, if when the node 'pres: Johnson...' printed, the user issued an interrupt (shift-del), output would cease and the cursor would be re-positioned at node 'pres: Kennedy...'.

```
 pres: Kennedy  05-29-1917       11-22-1963       Dem

G

 pres: Kennedy  05-29-1917       11-22-1963       Dem
      elect: 1960       votes=303       lsr=Nixon       Rep
```

```
      admin: 51        inaugdate=01-20-1961     vp=Johnson
      cong: 87  SRep%=36  SDem%=64  HRep%=40  HDem%=60
      cong: 88  SRep%=33  SDem%=67  HRep%=41  HDem%=59
 pres: Johnson  08-27-1908       01-22-1973       Dem
```

(interrupt issued)

The cursor is now positioned back at the node 'pres: Kennedy..' which can be verified by the command

p

```
 pres: Kennedy  05-29-1917       11-22-1963       Dem
```

Another command which allows the user the traverse the data base in a preorder fashion is 'N'. Unlike 'G' which keeps retrieving nodes until the end of the tree 'N' only retrieves one record and leaves the cursor pointing to the current record. For example, to sequence through the next seven records in preorder fashion the statement:

NNNNNNN

would produce

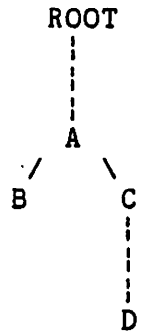
```
      elect: 1960       votes=303       lsr=Nixon       Rep
      admin: 51 inaugdate=01-20-1961     vp=Johnson
      cong: 87  SRep%=36  SDem%=64  HRep%=40  HDem%=60
      cong: 88  SRep%=33  SDem%=67  HRep%=41  HDem%=59
 pres: Johnson  08-27-1908       01-22-1973       Dem
```

elect: 1964 votes=486 lsr=Goldwater Rep

admin: 52 inaugdate=11-22-1963 vp=NULL

5.1.1 More on Traversal

Procedural query languages, like the one implemented in Hi, require the user to specify what is wanted as well as how to obtain it. Consider the following general tree structure.



If the user were positioned at the ROOT node and wanted to obtain information stored at node D, he would have to manually navigate down the tree until reaching node D and then retrieve the desired information. Using this simple example three navigation commands would be needed before retrieval could take place.

(1) c{A} change to node A

(2) c{C} change to node C

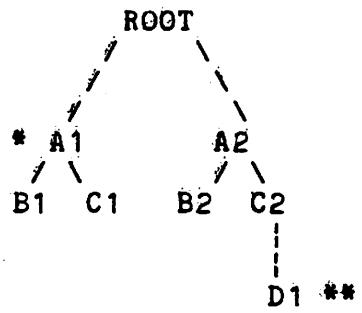
(3) c{D} change to node D

(4) retrieve information.

Hi provides the ability to proceed <u>downward</u> in the tree directly to the desired node with one command. The command

j{node name}

will position the cursor at the first occurrence of the node named in the command. The named node must be at a level in the tree lower than the current node. Consider the following example.



* indicates the cursor position before the "j" command was given
** indicates the cursor position after the "j" command is complete

If the cursor were positioned at node A1, and the command j{D} were executed, the cursor would be repositioned at node D1. Note that the current position of the cursor is in a different sub-tree than when the command was issued (A2 as opposed to A1). As the "j" command positions the cursor to the first occurrence of the requested node, this may require a jump to a different sub-tree.

5.2 l vs. p

As previously shown, the print commands 'p' and 'P' cause the record to be printed prefixed by the node name. The 'l' command acts the same as 'p' except that the node name is not included. For example,

The command

```
p
```

would produce

```
 pres: Eisenhower        10-14-1890        03-28-1969        Rep
```

while the command

```
1
```

would produce

```
 Eisenhower        10-14-1890        03-28-1969        Rep
```

6. More on Qualified Sequencing

In previous examples we have seen how to retrieve information based on the data content of a record. Hi provides another level of qualified retrieval. Referring to the presidential data base in Figure 1, suppose we wanted to know if there were any presidents who had never been elected. Since in the correct case there will be no 'elect' records, qualification based on record content will be of no use. We want to print 'pres' records only if they meet the condition that they had no 'elect' children. The general form for this type of qualified search is:

g < condition >/< qualifier >/

where < condition > is one of: =n, >n, <n and n is the number of successful retrievals which must be made before the entire query is considered successful. For instance the command:

```
r g/.*/ c{elect} g=0 /.*/ < p
```

would produce

```
pres: Ford      07-14-1913      NULL     Rep
```

alternatively

```
r g/.*/ c{elect} g<1 /.*/ < p
```

would also produce

```
pres: Ford      07-14-1913      NULL     Rep
```

These commands have the structure

(1) go to the root

(2) for each record at this level change to it's 'elect' children

(3) if there are no 'elect' children

(a) go back to the parent record

(b) print the parent record

If the query were to find all presidents who were associated with more than three congresses during their terms in office, the command

```
r g/.*/ c{cong} g>3 /.*/ < p
```

would produce

pres: Eisenhower 10-14-1890 03-28-1969 Rep

SUMMARY

Although Hi has had limited exposure to a real user environment, the exposure it has had can be considered successful. It is currently being used successfully to manage a 32k record data base containing about 500k bytes of data.

While the query language of Hi may at first look somewhat forbidding, experience has shown that even the naive user quickly becomes comfortable using it. This is partly due to the fact that it so closely resembles the UNIX text editor language. Users have found the overall system easy to understand and use.

As mentioned above, the structure of the system is expected to continue to evolve. Current and future work includes the implementation of secondary indices, an english like query language and some initial form of data distribution with the ultimate goal being a fully distributed system.

MH-9544-RJY-jer R. J. Yanofchick

Atts.
Appendix

Appendix: The Language

The language for the system is a sequence of one line statements in the basic format

<stat> :: = <cmd>|<cmd><stat>

The commands are (where z stands for a cursor):

R	Set the cursor z to the first occurrence of the root.
r	Set the cursor to the root node of the current sub-tree.
p	Print the record z points to.
pk	Print the record z points to but now with k leading blanks (this overrides the usual number of leading blanks that occur corresponding to the level of the record in the original schema. This is convenient for use in restructuring the hierarchy.
P	Print the record with its internal identifier.
L	List the record at z, i.e., give such detail as names and identifiers of parent and children.
n	Move z to the next record of

("return")	the current type without changing parent record and print it.
b	Move z back to the previous record of the current type without changing parent record and print it.
c{name}	If "name" is either "<" or the parent's actual name change to the current record's parent else change z to the first occurrence of the child called "name". If "name" is ">", then change to the first occurence of the first child. If "name" is "¦", change to the first occurence of the next sibling.
/reg/	Seek to the next record of the current type under the same parent that satisfies the regular expression "reg".
g/reg/	Seek to each record of the current type under the same parent that satisfies the regular expression "reg".
T	Regard the node as a table independent of its hierarchy and go to its first record.
*n	Seek to the next record in the current node regarded as a table independent of its hierarchy.
*/reg/	Just as in *n but qualify the record
g*/reg/	Seek to each record of the current type

	across the entire data base regardless of parent that satisfies the regular expression "reg".
g(/reg/<stat>)	Do statement <stat> for every record satisfying "g/reg/" before continuing with the rest of the statement in which this command is imbedded.
g(*/reg/<stat>)	The appropriate combination of previous commands.
del?	Display the current field delimiter.
delim	Change the current delimiter.
C[<expression>]	Compute expression and display the result.
C "label" [<expression>]	Same as above except the result is prefixed by "label".
H"<titles>"	Set up column titles to be used in future summations
pipe <filename>	Copy results of subsequent queries to <filename>.
depipe	Stop copying results to <filename>.
use <filename>	Accept further commands from <filename>.
x<$i,j,k,...>	Extract the numbered fields from the current record.
G	Produce listing of data base in preorder fashion.
N	Get next record from data base in preorder fashion.
:	Same as above.
l	Print current record without node name prefix.

schema?	Display the schema for the current data base.
g<condition>/reg/	Seek records of the current type which satisfy the regular expression "reg" and then test if "condition" is satisfied.

Note that each "g/reg/" command ends with z pointing to the last record satisfying "reg".

To update the data base, one can use

a <input> .	Append <input> after the record pointed to by z.
i <input> .	Insert <input> before z.
s/str1/str2/	Substitute "str2" for "str1" in the current record.
i(a) *recordid or /text to identify rec/ .	Insert (append) a record that already exists in the data base at this point in the hierarchy.
d	Delete current record and all descendants.

d/reg/	Delete all records of the current type under the current parent that satisfy the regular expression "reg".
j{name}	change to the first occurrence of node "name"; this may require a change to a different sub-tree

Finally,

!<shell_cmd>	will escape and do the shell command <shell_cmd>, and
q	will quit.