To: Distribution

From: Paul Green

Date: 05/08/75

Subject: A random word generator for Multics

Morrie Gasser of the MITRE Corporation has written a set of programs that are capable of generating pronouncable English words at random. Enclosed with this MTB is the draft documentation for the various modules which comprise the word generator. Comments on the user interface are especially welcome; send them to Green.HDruid and Gasser.ADruid on the MIT Multics system.

The random word generator (random_word_) is a table-driven program that returns an array of numbers (units) which form a word. The units are supplied by a subroutine that is caller-specified. The standard version of this subroutine is named random_unit_, although there is no requirement that the units themselves be random.

The parameters to random_word_ are the number of letters that may appear in the generated word, and the random_unit_ subroutine. The random_word_ routine calls random_unit_ repeatedly to get units, each time determining from a "digram table" whether the returned unit may be added to the end of the word being generated, according to the rules encoded in the digram table. Units which satisfy the rules are added to the end of the generated word; units which do not satisfy the rules are ignored. Units are requested until the length in letters meets the caller's criteria.

The table that drives random_word_ is referenced as an external array with the name "digrams_". This table can be prepared by the user by creating an ASCII segment specifying the rules, and compiling it with the digram_table_compiler. The digram table is in two parts. The first part specifies one or two letter symbols that define each unit, and some flags that define various rules for each unit. The second part lists every possible pair of these units (i.e., if there are n units then there are n*n pairs), and contains several more flags for each pair that define rules about combining pairs.

Only the digram table itself is specifically English-oriented; the symbolic representation of the units and letters is unimportant to the digram_table_compiler and random_word_ (except that the number of letters in each unit is used to determine how long the generated word is). The random_word_ and random_unit_ subroutine operate upon unit indices, not the actual ASCII characters. These unit indices may be converted back to their character represenations by calling the convert_word_ subroutine.

As the word generator currently exists, the random_unit_ subroutine "knows" what units exist in the digram table, what their frequencies of occurance are, and which ones have specific attributes. Thus it does not have to reference the digram table. For that reason, if it desired to replace the digram table, the random_unit_ subroutine must also be replaced. Some of these dependencies could have been eliminated by having the random_unit_ subroutine reference the digram table on the first call to determine which units exist, but this was not done for reasons of efficiency. The only unit attribute that random_unit_ cares about is the "vowel" attribute, for the entrypoint random_unit_$random_vowel. For these reasons, a new digram table can be created (without replacing random_unit_) only if the English-letter representation of the units, and the order of the units, is not modified.

Note that only the command interface (generate_words) will be user-visible; the rest of the modules will remain internal interfaces.

Name: generate_word_

This subroutine returns a random pronounceable word as an ASCII character string. It also returns the same word split by hyphens into syllables as an aid to pronunciation.

Usage

```
     declare generate_word_ entry (char(*), char(*), fixed bin,
          fixed bin);

     call generate_word_ (word, hyphenated_word, min, max);
```

1) word	is the random word, padded on the right with blanks. This string must be long enough to hold the word (at least as long as max). (Output)
2) hyphenated_word	is the same word split into syllables. The length of this string must be greater than max to allow for the hyphens. A length of 3*max/2 + 1 will always be sufficient. (Output)
3) min	is the minimum length of the word to be generated. This value must be greater than 3 and less than 21. (Input)
4) max	is the maximum length of the word to be generated. The actual length of the word will be uniformly random between min and max. The value of max must be greater then or equal to min, and less than 21. (Input)

Note

Each call to generate_word_ should produce a different random word, regardless of when the call is made. However, as with any random generator, there is no guarantee that there will be no duplicates. The probability of duplication is greater with shorter words.

__Entry__: generate_word_$init_seed

This entry allows the user to specify a starting seed for generating random words. If a seed is specified, the exact same sequence of random words will always be generated on subsequent calls to generate_word_ providing the same values of min and max are specified. If this entry is not called in a process, the value of the clock is used as the initial seed on the first call to generate_word_, thereby "guaranteeing" different sequences of words in different processes.

__Usage__

```
declare generate_word_$init_seed entry (fixed bin(35));

call generate_word_$init_seed (seed);
```

1) seed is the initial seed value. If zero, the system clock will be used as the seed. (Input)

Name: generate_words, gw

This command will print random pronounceable "words" on the user's terminal.

Usage

    generate_words -control_args-

1) control_args may be selected from the following:

nwords	is the number of words to print. If not specified, one word is printed.
-min *n*	specifies the minimum length, in characters, of the words to be generated.
-max *n*	specifies the maximum length of the words to be generated.
-length *n*, -ln *n*	specifies the length of the words to be generated. If this argument is specified, all words will be this length, and -min or -max may not be specified.
-hyphenate, -hph	causes the hyphenated form (divided into syllables) of each word to be printed alongside the original word.
-seed SEED	On the first call to generate_words in a process, the system clock is used to obtain a starting "seed" for generating random words. This seed is updated for every word generated, and subsequent values of the seed depend on previous values (in a rather complex way). If the -seed argument is specified, SEED must be a positive decimal integer. For a given value of SEED, the sequence of random words will always be the same providing the same length values are specified. When no -seed argument is specified, the last value of the updated

seed from the previous call to generate_words will be used. To revert back to using the system clock as the seed, specify a zero value for SEED, i.e., -seed 0.

Notes

If neither -min, -max, nor -length are specified, the defaults are -min 6 and -max 8. In all other cases, the defaults are -min 4 and -max 20.

If -length is not specified, the lengths of the random words will be uniformly distributed between min and max. Words generated are printed one per line, with the hyphenated forms, if specified, lined up in a column alongside the original words.

Name: convert_word_

This subroutine is used to convert the random word array returned by random_word_ to ASCII.

Usage

```
dcl convert_word_ entry ((0:*) fixed bin, (0:*) bit(1)
     aligned, fixed bin, char(*), char(*));

call convert_word_ (word, hyphenated_word, word_length,
     ascii_word, ascii_hyphenated_word);
```

1) word — Array of random units returned from a previous call to random_word_. (Input)

2) hyphenated_word — Array of bits indicating where hyphens are to be placed, returned from random_word_. (Input)

3) word_length — Number of units in word, returned from random_word_. (Input)

4) ascii_word — This string will contain the word, left justified, with trailing blanks. This string should be long enough to hold the longest word that may be returned. This is normally the value of "maximum" supplied to random_word_. (Output)

5) ascii_hyphenated_word — This string will contain the word, with hyphens between the syllables, left justified within the string. The length of this string should be at least 3*maximum/2+1 to guarantee that the hyphenated word will fit. (Output)

Entry: convert_word_$no_hyphens

This entry can be used to obtain the ASCII form of a random word without the hyphenated form.

Usage

```
dcl convert_word_$no_hyphens ((0:*) fixed bin, fixed bin,
     char(*));

call convert_word_$no_hyphens (word, word_length,
     ascii_word);
```

Arguments are the same as above.

Name: convert_word_char_

This subroutine facilitates printing of the hyphenated word returned from a call to hyphenate_.

Usage

```
     dcl convert_word_char_ entry (char(*), (*) bit(1) aligned,
          fixed bin, char(*) varying);

     call convert_word_char_ (word, hyphens, last, result);
```

1) word	This string is the word to be hyphenated. (Input)
2) hyphens	This is the array returned from a call to hyphenate_ that marks characters in word after which hyphens are to be inserted. (Input)
3) last	This is the status code returned from hyphenate_. If negative, the result will be the original word, unhyphenated, with ** following it. If positive, the word will be returned hyphenated, but with an asterisk preceding the last'th character. If zero, the word will be returned hyphenated without any asterisks. (Input)
4) result	This string contains the resultant hyphenated word. (Output)

Name: digram_table_compiler, dtc

This command compiles a source segment containing the digrams for the random word generator and produces an object segment with the name "digrams_".

Usage

```
     digram_table_compiler pathname -option-
```

1) pathname	is the pathname of the source segment. If the suffix ".dtc" does not appear, it will be assumed. Regardless of the name of the source segment, the output segment will always be given the name "digrams_" and will be placed in the working directory.
2) -option-	may be the following:
-list, -ls	lists the compiled table on the terminal. The table will be printed in columns to fit the terminal line length. If file_output is being used, lines will be 132 characters long.
-list n, -ls n	lists the table as above, but uses n as the number of columns to print. Each column occupies 14 positions, thus a value of 5 will cause 5 columns to be printed, each line being 70 characters long. This option is useful when file_output is being used, so that the lines produced are not too long to fit on the terminal to be used to print the output file.

Notes

The compiler makes an attempt to detect inconsistent combinations of attributes, as well as syntax errors. If an error is encountered during compilation, processing of the source segment will continue if possible. The digrams segment in case of an error will be left in an undefined state.

During compilation, the ALM assembler is used. At that point the letters "ALM" will be printed on the terminal. If compilation was successful, no other messages should appear.

The listing produced by digram_table_compiler is in a format suitable for printing on the terminal -- not for dprinting. This is because blank lines are used for page breaks, instead of the "new page" character as recognized by dprint.

Syntax

The syntax of the source segment is specified below. Spaces are meaningful to this compiler and a space is only allowed where specified as <space>. The new line character is indicated as <new line>.

```
<digram table>::= <unit specs>;[<new line>]...<digram specs>$
<unit specs>::= <unit spec>[<delim><unit spec>]...
<digram specs>::= <digram spec>[<delim><digram spec>]...
<delim>::= ,[<new line>]|<new line>
<unit spec>::= <unit name>[<not begin word>[<no final split>]]
<digram spec>::= [<begin><not begin><break><prefix>]
                 <unit name><unit name>[<suffix>[<end>[<not end>]]]
<unit name>::= <letter>[<letter>]
<letter>::= a|b|c|d|e|f|g|h|i|j|k|l|m|n|o|p|q|r|s|t|u|v|w|x|y|z
<not begin word>::= <bit>
<no final split>::= <bit>
<begin>::= <bit>
<not begin>::= <bit>
<break>::= <bit>
<prefix>::= <space>|-
<suffix>::= <space>|-|+
<end>::= <bit>
<not end>::= <bit>
<bit>::= <space>|1
```

The first part of the <digram table> consists of definitions of the various units that are to be used and their attributes. The units are defined as one or two-letter pairs, and the order in which they are defined is unimportant. For each unit, the attributes <not begin word> and <no final split> may be specified. In addition, if <unit name> is a, e, i, o, or u, the "vowel" attribute is set. If the unit is y, the "alternate vowel" attribute is set. A <bit> is assumed to be zero if specified as <space>, or one if specified as 1.

The second part of <digram table> specifies all possible pairs of units and the attributes for each pair. The order in which these pairs must be specified depends on the order of the <unit specs> as follows:

Number the <unit spec>s from 1 to n in the order in which they appeared in <unit specs>. The first <digram spec> must consist of the pair of units numbered (1,1), the second <digram spec> is the pair (1,2), etc., and the last <digram spec> is the pair (n,n). All pairs must be specified, i.e., there must be n*n <digram spec>s. The <bit>s preceding or following each pair set the attributes for that pair as shown. The <prefix> and <suffix> indicators are set to 1 if specified as "-". If <suffix> is specified as "+", the "illegal pair" indicator will be set, and no other attributes may be specified for that <digram spec>.

Example

The following is a very short example of a <digram table>. Only four units are defined, "a", "b", "sh" and "e". The letter "e" is given the "no final split" attribute, the pair "aa" is given "illegal pair", the pair "ae" is given the "not begin", "break", and "not end" attributes, etc.

```
a,b,sh,e 1;
aa+,ab,ash, 11 ae  1
ba, 1  bb, 11 bsh  1,be
sha, 11 shb  1,shsh+,she,ea,eb,esh,ee
$
```

Assume the above segment was named "dt.dtc". Below is an example of the command used to compile and list the table produced for dt.

```
digram_table_compiler dt -ls
ALM
```

```
  1 a  0010     2 b  0000     3 sh 0000     4 e  0110

000  aa +00   000  ba   00   000 sha   00   000  ea  00
000  ab  00   010  bb   00   011 shb  01    000  eb  00
000  ash 00   011  bsh 01    000 shsh+00    000  esh 00
011  ae  01   000  be   00   000 she   00   000  ee  00
```

The first line of output lists the individual units. The number preceeding the unit is the unit index. The four bits following the unit are respectively:

not begin syllable
no final split
vowel
alternate vowel

Following the unit specifications are the digram specifications. Preceeding each digram are three bits and a space (or possibly a "-") with meanings corresponding to those specified in the source segment as follows:

begin
not begin
break
prefix (if "-" appears)

Immediately following each digram is a field which may be blank, "-", or "+". If "+", the "illegal pair" flag is set. Otherwise, the meaning of the "-" and following two bits are as follows:

suffix (if "-" appears)
end
not end

Names: hyphen_test

This command uses the random word generator (the same one used by generate_words) to divide words into syllables. Words are printed on the terminal with hyphens between the syllables.

Usage

```
hyphen_test -control_arg- -word1- ... -wordn-
```

1) control_arg	may be -probability (-pb), specifying that the probability of each of the words that follows be printed alongside the hyphenated word.
2) wordi	are one or more words to be hyphenated. A word may consist of three to twenty alphabetic characters, only the first of which may be uppercase.

Notes

The control argument may appear anywhere in the command line. However, it only applies to words that follow. Words preceding the option will be hyphenated but no probabilities will be calculated.

If a word contains any illegal characters, or is not of three to twenty characters in length, the word will be printed unhyphenated, followed by **.

If the word could not be completely hyphenated because it was considered unpronounceable, an asterisk (*) will be printed out in front of the first character that was not accepted. The part of the word before the asterisk will be properly hyphenated.

The calculated probability is the probability that the word would have been generated by generate_words, assuming generate_words was requested to generate a word of that length only. If a range of lengths is requested of generate_words, each length has equal probability. For example, if generate_words is called to generate words of 6, 7, or 8 characters, there is a 33% probability that a given word will have 8 characters. If hyphen_test is then asked to calculate the probability of a given 8 letter word, that probability should be divided by 3 to obtain the correct probability for the case of three possible lengths.

Name: hyphenate_

This subroutine attempts to hyphenate a word into syllables.

Usage

```
dcl hyphenate_ entry (char(*), (*) bit(1) aligned, fixed
     bin);

call hyphenate_ (word, hyphens, code);
```

1) word	This is a left justified ASCII string, 3 to 20 characters in length. This string must contain all lowercase alphabetic characters, except the first character may be uppercase. Trailing blanks are not permitted in this string. (Input)
2) hyphens	This array will contain a "1"b for every character in the word that is to have a hyphen following it. (Output)
3) code	This is a status code, as follows: 0 word has been successfully hyphenated. -1 word contains illegal (non alphabetic or uppercase) characters. -2 word was not from three to twenty characters in length. Any positive value of code means that the word couldn't be completely hyphenated. In this case, code is the position of the first character in word that was not acceptable. The part of the word before code will be properly hyphenated. (Output)

Notes

This subroutine uses random_word_ to provide the hyphenation. It does this by calling random_word_$give_up and supplying its own version of random_unit and random_vowel that return specified units (of the particular word to be hyphenated) instead of random units.

The word supplied to hyphenate_ is first transformed into units by translating pairs of letters into single units if a 2-letter unit is defined for the pair, and then by translating the remaining single letters into units. See the subroutine description of random_word_ and random_unit_ for a description of units. If any units of the word are rejected by random_word_, hyphenate_ tries to determine if the refused letter was a 2-letterr unit. If this is the case, the 2-letter unit is broken into two 1-letter units and random_word_ is called again. In rare cases, hyphenate_ is not able to determine which 2-letter unit is at fault, and will return a status code indicating that the word is unpronounceable, when, in fact, it could have been properly divided by breaking up a 2-letter unit.

Entry: hyphenate_$probability

This entry returns information as above, but also supplies the probability of the word having been generated at random by generate_word_ or random_word_generator_. The assumption is made that generate_word_ or random_word_generator_ was asked to supply a word of exactly the same length as the word given to hyphenate_, rather than a range of lengths. If a range of lengths was asked of generate_word_, the probability must be divided by the number of different lengths (all lengths are equally probable).

Usage

```
     dcl hyphenate_$probability entry (char(*), (*) bit(1)
          aligned, fixed bin, float bin);

     call hyphenate_$probability (word, hyphens, code,
          probability);
```

1) to 3) are as above.

4) probability is the probability as defined above. (Output)

Notes

If the supplied word is illegal (i.e. code is not zero), the probability will be returned as zero.

Entry: hyphenate_$debug_on, hyphenate_$debug_off

These entries set and reset a switch that causes hyphenate_$probability to print, on user_output, all *units* (see the subroutine descriptions of random_word_ and random_unit_ for a description of units) that are illegal in a given position of the word. This entry is useful for debugging a digram table for random_word_. It makes no assumptions about the information contained in the digram table with regards to which units are defined, their distributions, the order of the units, etc. However, it assumes that a call to random_unit_$probability will return arrays of the size digrams_$n_units containing the probabilities of the units that are defined. See the subroutine description of random_unit_ for a description of the random_unit_$probability entry, and the subroutine description of random_word_ for a description of digrams_.

Usage

```
     dcl hyphenate_$debug_on entry;
     dcl hyphenate_$debug_off entry;

     call hyphenate_$debug_on;
     call hyphenate_$debug_off;
```

Notes

An example of the output produced is as follows. The assumption is that hyphenate_$probability is invoked by the hyphen_test command using the -probability option.

```
hyphenate_$debug_on
hyphen_test -probability fish
x,ck,f; b,c,d,f,g,h,j,k,m,n,p,s,t,v,w,x,y,z,ch,gh,ph,
rh,sh,th,wh,qu,ck,i; i,rh,wh,qu,sh;
fish  6.04127576e-5
```

In the above example, the units x and ck are shown to have been illegal as the first unit of the word, and the unit f, (underlined) is the first unit of the word that was accepted. All other units that were not printed are legal as the first unit of the word. Following the semicolon after f are the units that are illegal in the second position of the word (assuming that f is the first unit). Then i is shown as the legal unit that is taken from the word "fish". This repeats for each position of the word, ending in the legal unit sh (note only one underline).

If the supplied word is illegal, the last underlined letter in the output is (usually) the letter that was not accepted. In cases where hyphenate_ has to split up a 2-letter unit, the word will be shown to start over from the beginning.

Name: print_digram_table

This entry merely prints the digram table on the terminal, assuming that it has already been compiled successfully. The segment "digrams_" is assumed to be located in the working directory.

Usage

```
print_digram_table -n-
```

1) n is the number of columns in which to print the table. If not specified, the maximum number of columns that will fit in the terminal line will be used. Each column occupies 14 positions. If file_output is being used, the terminal line width is assumed to be 132.

Notes

This entry performs the same function as the -list option of digram_table_compiler.

Name: random_unit_

This subroutine provides a random unit number for random_word_ based on a standard distribution of a given set of units. It is referenced by the generate_word_ subroutine as an entry value that is passed in the call to random_word_. This subroutine assumes that the digram table being used by random_word_ is a standard table. The digram table itself is not referenced by this subroutine.

Usage

```
declare random_unit_ entry (fixed bin);

call random_unit_ (unit);
```

1) unit is a number from 1 to 34 that corresponds to a particular unit as listed in Notes below. (Output)

Notes

The table below contains the units that are assumed specified in the digrams supplied to random_word_. Shown in the table are the unit number, the letter or letters that unit represents, and the probability of that unit number being generated.

1	a	.04739	8	h	.02844	15	o	.04739	22	w	.03792	29	rh	.00474
2	b	.03792	9	i	.04739	16	p	.02844	23	x	.00474	30	sh	.00948
3	c	.05687	10	j	.03792	17	r	.04739	24	y	.03792	31	th	.00948
4	d	.05687	11	k	.03792	18	s	.03792	25	z	.00474	32	wh	.00474
5	e	.05687	12	l	.02844	19	t	.04739	26	ch	.00474	33	qu	.00474
6	f	.03792	13	m	.02844	20	u	.02844	27	gh	.00474	34	ck	.00474
7	g	.03792	14	n	.04739	21	v	.03792	28	ph	.00474			

<u>Entry</u>: random_unit_$random_vowel

This entry returns a vowel unit number only.

<u>Usage</u>

```
declare random_unit_$random_vowel (fixed bin);

call random_unit_$random_vowel (unit);
```

1) unit   As above. (Output)

<u>Notes</u>

Below are listed the vowel units and their distributions.

1	a	.167
5	e	.250
9	i	.167
15	o	.167
20	u	.167
24	y	.083

<u>Entry</u>: random_unit_$probabilities

This entry returns arrays containing the probabilities of the units as listed in the table on the previous page. This entry is provided for hyphenate_$probability and any other program that might require this information. The probabilities must be computed when this entry is called, so it is suggested that the call be made only once per process and the values saved in internal static storage.

<u>Usage</u>

```
declare random_unit_$probabilities entry ((*) float bin, (*)
     float bin);

call random_unit_$probabilities (unit_probs, vowel_probs);
```

1) unit_probs   This array contains the probabilities of the individual units assuming the random_unit_ entry is called to generate the random units. The value

of unit_probs(i) is the probability of unit(i). (Output)

2) vowel_probs This array contains the probabilities of the units when random_vowel is called. Since there are only 6 vowels, most of these values will be zero. (Output)

Notes

A future version of random_unit_ may use different units with different probabilities. The size of the two arrays must be large enough to hold the maximum number of values that may be returned by random_unit_ (which is currently 34). Programs should not depend on the unit_index-to-letter correspondence as shown in the table. This information can be obtained by using the include file digram_structure.incl.pl1.

<u>Name</u>: random_word_

This routine returns a single random pronounceable word of specified length. It is called by generate_word_, and allows the caller to specify the particular subroutines to be used to generate random units. For users desiring random words with an English-like distribution of letters, generate_word_ should be used.

<u>Usage</u>

```
     dcl random_word_ entry ((0:*) fixed, (0:*) bit(1) aligned,
          fixed, fixed, entry, entry);

     call random_word_(word, hyphens, char_length, unit_length,
          random_unit, random_vowel);
```

1) word	The random word will be stored in this array starting at word(1) (word(0) will always be 0). The numbers stored will correspond to a "unit index" as described in <u>Notes</u> below. This array must have a length at least equal to the value of "char_length". Unused positions in this array, up to word(char_length), will be set to zero. (Output)
2) hyphens	This array must be of length at least "char_length". A bit on in a position of this array indicates that the corresponding unit in "word" (including the very last unit) is the last unit of a syllable. (Output)
3) char_length	Length of the word to be generated, in characters. (Input)
4) unit_length	This is the length of the generated random word in <u>units</u>, i.e., the index of the last non-zero entry in the "word" array. The actual length of the word in equivalent characters will be the value of char_length. (Output)

5) random_unit This is the routine that will be called by random_word_ each time a random_unit is needed. The random_unit routine is declared as follows:

```
dcl random_unit entry (fixed bin);
```

where the value returned is a unit index between 1 and n_units. If an English-like distribution of letters is desired, the "random_unit_" subroutine may be specified here. See Notes below. (Input)

6) random_vowel

This is the routine called by random_word_ when a vowel unit is required. This routine must return the index of a unit whose "vowel" or "alternate_vowel" bits are on. See Notes below. This routine is declared as follows:

```
dcl random_vowel entry (fixed bin);
```

If desired, the subroutine "random_unit_$random_vowel" may be specified in this place. (Input)

Notes

The word array can be converted into characters by calling convert_word_.

In order to use random_word, a digram table, contained in a segment named "digrams_", must be available in the search path. This table can be created by the digram_table_compiler.

If the user supplies his own versions of random_unit and random_vowel, these subroutines will have to supply legal units that are recognized by the random_word_ subroutine. The include file "digram_structure.incl.pl1" can be used to reference the digram table to determine which units are available. If included in the source program, appropriate references to the following variables of interest in "digrams_" will be generated:

```
dcl n_units fixed bin defined digrams_$n_units;
dcl letters(0:n_units) char(2) aligned
```

```
          based(addr(digrams_$letters));
     dcl 1 rules(n_units) aligned based(addr(digrams_$rules)),
                    2 vowel bit(1),
                    2 alternate_vowel bit(1),
                    .....
```

where:

n_units	is the number of different units.
letters(i)	contains 1 or 2 characters (left justified) for the i'th unit.
rules.vowel(i), rules.alternate_vowel(i)	One of these two bits are set for the units that may be returned by a call to random_vowel.

When random_unit is called, a number from 1 to n_units must be returned. When random_vowel is called, a number from 1 to n_units, where one of the two bits in rules(i) is marked, must be returned.

Entry: random_word_$debug_on

This entry sets a switch in random_word_ that causes printing (on user_output) of partial words that could not be completed. This entry is of interest during debugging of random_word_ or for checking the consistency of the digram table prepared by the user.

Usage

```
     dcl random_word_$debug_on entry;

     call random_word_$debug_on;
```

Entry: random_word_$debug_off

This entry resets the switch set by debug_on.

Additional notes

The random_word_ subroutine can be used for certain special applications (such as the application used by hyphenate_), and there are certain features that help support some of these applications. The features described below are of little interest to most users.

The first feature allows the caller-supplied random_unit (and random_vowel) subroutine to find out whether random_word_ "accepted" or "rejected" the previous unit supplied by random_unit. Each time random_unit is invoked by random_word_, the value of the argument passed is the index of the previous unit that random_unit_ returned (or zero on the first call to random_unit in a given invocation of random_word_). The sign of the argument will be positive if this last unit was accepted. "Accepted" means that the last unit was inserted into the random word and the word index maintained by random_word_ was incremented. Once a unit is accepted, it is never removed. Thus a positive value of the unit index passed to random_unit means that a unit for the next position of the word is requested.

If the unit index passed to random_unit has a negative sign, the last unit was rejected according to the rules used by random_word_ and information supplied in the digram table. If the unit is rejected, random_word_ does not advance its word index and calls random_unit again for another unit for that same word position. With this information random_unit can keep track of the "progress" of the word being generated.

The feature described above is used by the special random_unit routine provided by hyphenate_. Since the random_unit routine for hyphenate_ is not really supplying random units (but is supplying units of the word to be hyphenated), it must know whether any particular unit is rejected by random_word_. Rejection then implies that the word is illegal according to random_word_ rules.

The second feature allows random_unit to "try" a certain unit without committing that unit to actually be used in the random word. The sign of each unit supplied to random_word_ by random_unit is checked. If the sign of the word is positive, random_word_ will accept or reject the unit according to its rules, and will indicate this on the subsequent call to random_unit.

If the sign of the unit passed to random_word_ is negative, random_word_ will merely indicate (on the subsequent call to random_unit) whether that unit would have been accepted, but it never actually updates the word index. In other words, random_word_ always rejects the unit, but lets random_unit know whether the unit was acceptable.

This latter feature is used by hyphenate_$probability in order to determine which of all possible units are acceptable in a given position of the word. The random_unit routine used by hyphenate_$probability tries all possible units in each word position, and only allows random_word_ to accept the unit that actually appears in that position.

Name: read_table_

This subroutine is the compiler for the digram table for random_word_. It is called by digram_table_compiler.

Usage

```
declare read_table_ entry (ptr, fixed bin(24), returns
     (bit(1));

flag = read_table_ (source_ptr, bitcount);
```

1) source_ptr	is a pointer to the source segment to be compiled. (Input)
2) bitcount	is the bit count of the source segment. (Input)
3) flag	is "0"b if compilation was successful. It is "1"b if an error was encountered.

Notes

If compilation was successful, the compiled table will be placed in the working directory with the name "digrams_". If unsuccessful, the digrams segment may or may not have been created, and may be left in an inconsistent state (i.e., unusable by random_word_). Error messages are printed out on user_output as the errors are encountered, except that file system errors are printed on error_output.

This subroutine uses the ALM assembler for part of its work. As a result, the letters "ALM" will be printed on user_output sometime during the compilation.