```
   CCCCCCCCCCC  LLL              IIIIIIIII     UUU         UUU  TTTTTTTTTTTTTTT  LLL
   CCCCCCCCCCC  LLL              IIIIIIIII     UUU         UUU  TTTTTTTTTTTTTTT  LLL
   CCCCCCCCCCC  LLL              IIIIIIIII     UUU         UUU  TTTTTTTTTTTTTTT  LLL
CCC             LLL                 III      UUU         UUU        TTT        LLL
CCC             LLL                 III      UUU         UUU        TTT        LLL
CCC             LLL                 III      UUU         UUU        TTT        LLL
CCC             LLL                 III      UUU         UUU        TTT        LLL
CCC             LLL                 III      UUU         UUU        TTT        LLL
CCC             LLL                 III      UUU         UUU        TTT        LLL
CCC             LLL                 III      UUU         UUU        TTT        LLL
CCC             LLL                 III      UUU         UUU        TTT        LLL
CCC             LLL                 III      UUU         UUU        TTT        LLL
CCC             LLL                 III      UUU         UUU        TTT        LLL
CCC             LLL                 III      UUU         UUU        TTT        LLL
CCC             LLL                 III      UUU         UUU        TTT        LLL
CCC             LLL                 III      UUU         UUU        TTT        LLL
   CCCCCCCCCCC  LLLLLLLLLLLLLLL     IIIIIIIII     UUUUUUUUUUUUUUU        TTT        LLLLLLLLLLLLLLL
   CCCCCCCCCCC  LLLLLLLLLLLLLLL     IIIIIIIII     UUUUUUUUUUUUUUU        TTT        LLLLLLLLLLLLLLL
   CCCCCCCCCCC  LLLLLLLLLLLLLLL     IIIIIIIII     UUUUUUUUUUUUUUU        TTT        LLLLLLLLLLLLLLL
```

FILEID**PRONOUNCE

```
PPPPPPPP    RRRRRRRR      000000    NN      NN    000000    UU      UU  NN      NN    CCCCCCCC  EEEEEEEEEE
PPPPPPPP    RRRRRRRR      000000    NN      NN    000000    UU      UU  NN      NN    CCCCCCCC  EEEEEEEEEE
PP      PP  RR      RR    00    00  NN      NN    00    00  UU      UU  NN      NN    CC        EE
PP      PP  RR      RR    00    00  NN      NN    00    00  UU      UU  NN      NN    CC        EE
PP      PP  RR      RR    00    00  NNNN    NN    00    00  UU      UU  NNNN    NN    CC        EE
PP      PP  RR      RR    00    00  NNNN    NN    00    00  UU      UU  NNNN    NN    CC        EE
PPPPPPPP    RRRRRRRR      00    00  NN  NN  NN    00    00  UU      UU  NN  NN  NN    CC        EEEEEEEE
PPPPPPPP    RRRRRRRR      00    00  NN  NN  NN    00    00  UU      UU  NN  NN  NN    CC        EEEEEEEE
PP          RR   RR       00    00  NN    NNNN    00    00  UU      UU  NN    NNNN    CC        EE
PP          RR   RR       00    00  NN    NNNN    00    00  UU      UU  NN    NNNN    CC        EE
PP          RR    RR      00    00  NN      NN    00    00  UU      UU  NN      NN    CC        EE                ....
PP          RR    RR      00    00  NN      NN    00    00  UU      UU  NN      NN    CC        EE                ....
PP          RR      RR      000000  NN      NN      000000    UUUUUUUUUU  NN      NN      CCCCCCCC  EEEEEEEEEE    ....
PP          RR      RR      000000  NN      NN      000000    UUUUUUUUUU  NN      NN      CCCCCCCC  EEEEEEEEEE    ....

LL              IIIIII      SSSSSSSS
LL              IIIIII      SSSSSSSS
LL                II      SS
LL                II      SS
LL                II      SS
LL                II      SS
LL                II        SSSSSS
LL                II        SSSSSS
LL                II              SS
LL                II              SS
LL                II              SS
LL                II              SS
LLLLLLLLLL      IIIIII    SSSSSSSS
LLLLLLLLLL      IIIIII    SSSSSSSS
```

```
/*
        IDENT = V04-000

*****************************************************************************
*                                                                           *
*  COPYRIGHT (c) 1978, 1980, 1982, 1984 BY                                  *
*  DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASSACHUSETTS.                   *
*  ALL RIGHTS RESERVED.                                                     *
*                                                                           *
*  THIS SOFTWARE IS FURNISHED UNDER A LICENSE AND MAY BE USED AND COPIED    *
*  ONLY IN  ACCORDANCE WITH  THE  TERMS  OF  SUCH  LICENSE  AND WITH THE    *
*  INCLUSION OF THE ABOVE COPYRIGHT NOTICE. THIS SOFTWARE OR  ANY  OTHER    *
*  COPIES THEREOF MAY NOT BE PROVIDED OR OTHERWISE MADE AVAILABLE TO ANY    *
*  OTHER PERSON.  NO TITLE TO AND OWNERSHIP OF  THE  SOFTWARE IS  HEREBY    *
*  TRANSFERRED.                                                             *
*                                                                           *
*  THE INFORMATION IN THIS SOFTWARE IS  SUBJECT TO CHANGE WITHOUT NOTICE    *
*  AND  SHOULD  NOT  BE  CONSTRUED AS  A COMMITMENT BY DIGITAL EQUIPMENT    *
*  CORPORATION.                                                             *
*                                                                           *
*  DIGITAL ASSUMES NO RESPONSIBILITY FOR THE USE  OR  RELIABILITY OF ITS    *
*  SOFTWARE ON EQUIPMENT WHICH IS NOT SUPPLIED BY DIGITAL.                  *
*                                                                           *
*                                                                           *
*****************************************************************************

++
 FACILITY:

        SET PASSWORD

 ABSTRACT:

        This module contains support routines for SET PASSWORD/GENERATE.

 ENVIRONMENT:

        Vax native

--

 AUTHOR: Brian Bailey  , CREATION DATE: Summer 83

 MODIFIED BY:

        V03-001 SHZ0001         Stephen H. Zalewski     01-feb-1984
                Extensive rewriting to implement /GENERATE and incorporate
                into SET PASSWORD.

**/
```

```
53          /* ROUTINE pronounceable_
54          |
55          | FUNCTIONAL DESCRIPTION:
56          |
57          | This procedure tests a word supplied by the caller for pronounceability.
58          |
59          | The word is tested by using random_word_ and whatever existing digram
60          | table is in use by random_word_ to determine the syllabification and
61          | pronounceability of the word supplied.
62          |
63          | INPUT PARAMETERS:
64          |       word -  A word consisting of ASCII letters to be tested.
65          |               All characters must be lowercase.
66          |
67          |       returned_hyphens -
68          |               A '1' bit in this array means that the corresponding
69          |               character in word is to have a hyphen after it.
70          |
71          |       n_units - number of units in unit table.
72          |
73          |       d_ptr - pointer to digram table
74          |       r_ptr - pointer to rules table
75          |       l_ptr - pointer to letters table
76          |
77          | OUTPUT PARAMETERS:
78          |       NONE
79          |
80          | ROUTINE VALUE:
81          |       pronounceability - set if the word is legal according to
82          |                          the random_word_ algorithm and the
83          |                          digram table.
84          |
85          | SIDE EFFECTS:
86          |       NONE
87          |
88          */
89
90          pronounceable_: procedure (word, returned_hyphens, d_ptr, l_ptr, r_ptr, n_units)  returns (bit(1));
91      1
92      1   dcl  word char(*);                                  /* PARAMETER: word being tested */
93      1   dcl  returned_hyphens(*) bit(1) aligned;           /* PARAMETER: hyphens for word */
94      1   dcl  pronounceability bit(1) aligned;              /* RETURNS VALUE: set if word is legal */
95      1
96      1
97      1   dcl  word_length_in_chars fixed bin static;       /* length of word in characters */
98      1   dcl  word_array(20) fixed bin static;             /* word spread out into units */
99      1   dcl  word_length fixed bin static;                /* length of word_array in units */
100     1   dcl  word_index fixed bin static;                 /* index into word_array */
101     1
102     1
103     1   dcl  random_word_ entry ((*) fixed bin, (*) bit(1) aligned, fixed bin,  /* algorithm used to test the */
104     1           fixed bin, entry, entry, ptr, ptr, ptr, fixed bin);            /* pronounceablilty of word.  */
105     1   dcl  returned_word(0:20) fixed bin;               /* word returned by random_word_ */
106     1   dcl  hyphenated_word(0:20) bit(1) aligned;        /* hyphens for word returned from random_word_ */
107     1   dcl  returned_length fixed bin;                   /* dummy argument for random_word_, since */
108 ;   1                                                     /* length of word is already known.       */
```

```
109     1
110     1
111     1       dcl  new_unit fixed bin;                        /* unit currently being tested in random_unit */
112     1       dcl  last_good_unit fixed bin static;           /* word_index of last good unit */
113     1       dcl  split_point fixed bin;                     /* index of 2-letter unit to be split into */
114   ;  1                                                      /* single letter units                     */
115     1       dcl  vowel_flag bit(1) aligned;                 /* set when random_vowel is called */
116     1
117     1
118   ;  1       /* this array contains information about all possible pairs of units */
119     1
120     1       dcl 1 digrams(n_units, n_units) based (d_ptr),
121     1             2 begin bit(1),           /* on if this pair must begin syllable */
122     1             2 not_begin bit(1),       /* on if this pair must not begin */
123     1             2 end bit(1),             /* on if this pair must end syllable */
124     1             2 not_end bit(1),         /* on if this pair must not end */
125     1             2 break bit(1),           /* on if this pair is a break pair */
126     1             2 prefix bit(1),          /* on if vowel must precede this pair in same syllable */
127     1             2 suffix bit(1),          /* on if vowel must follow this pair in same syllable */
128     1             2 illegal_pair bit(1);    /* on if this pair may not appear */
129     1
130   ;  1       /* this array contains left justified 1 or 2-letter pairs representing each unit */
131     1
132     1       dcl letters(0:n_units) char(2) based (l_ptr);
133     1
134   ;  1       /* this is the same as letters, but allows reference to individual characters */
135     1
136     1       dcl 1 letters_split(0:n_units) based (l_ptr),
137     1             2 first char(1),
138     1             2 second char(1);
139     1
140   ;  1       /* this array has rules for each unit */
141     1
142     1       dcl 1 rules(n_units) based (r_ptr),
143     1             2 no_final_split bit(1),          /* can't be the only vowel in last syllable */
144     1             2 not_begin_syllable bit(1),      /* can't begin a syllable */
145     1             2 vowel bit(1),                   /* this is a vowel */
146     1             2 alternate_vowel bit(1);         /* this is an alternate vowel, (i.e., "y") */
147     1
148     1       dcl n_units fixed bin;                  /* PARAMETER: number of units in unit table */
149     1       dcl d_ptr ptr;                          /* PARAMETER: pointer to digram table */
150     1       dcl l_ptr ptr;                          /* PARAMETER: pointer to unit letters */
151     1       dcl r_ptr ptr;                          /* PARAMETER: pointer to unit rules */
152     1
153     1
154     1       dcl  chars char(2);
155     1       dcl  char char(1);
156     1       dcl  i fixed bin;
157     1       dcl  j fixed bin;
158     1
159     1
160     1               split_point = 0;
161     1               goto continue;
162     1
163     1       pronounceable_$split: entry (word, returned_hyphens, splitpoint, d_ptr, l_ptr,
164     1                                    r_ptr, n_units)  returns (bit(1));
165     1
```

```
166  1     dcl  splitpoint fixed bin;              /* index of 2-letter unit to be split */
167  1          split_point = splitpoint;
168  1
169  1     continue:
170  1
171  1     /* Now that we have the word we want to hyphenate, we try to divide it up into units as defined          */
172  1     /* in the digram table.  We start with the first two letters in the word, and see if they are equal to any  */
173  1     /* of the 2-letter units.  If they are, we store the index of that unit in the word_array, and increment    */
174  1     /* our word_index by 2.  If they are not, we see if the first letter is equal to any of the 1-letter units. */
175  1     /* If it is, we store that unit and increment the word_index by 1.  If still not found, the character is    */
176  1     /* not defined as a unit in the digram table and the word is illegal.  Of course, the word may still not be */
177  1     /* "legal" according to random_word_ rules of pronunciation and the digram table, but we'll find that out   */
178  1     /* later.                                                                                                   */
179  1
180  1          word_length_in_chars = length (word);
181  1          word_index = 1;
182  1          do i = 1 to word_length_in_chars;
183  2              chars = substr (word, i, min (2, word_length_in_chars - i + 1));
184  2              j = 1;
185  2              do j = 1 to n_units while (chars ^= letters (j));   /* look for 2-letter unit match */
186  3                  end;
187  2              if j <= n_units & word_index ^= split_point
188  2                  then do;                                       /* match found */
189  3                      word_array (word_index) = j;               /* store 2-letter unit index */
190  3                      word_index = word_index + 1;
191  3                      i = i + 1;                                 /* skip over next unit */
192  3                      end;
193  2                  else do;                                       /* two-letter unit not found, search for 1-letter unit */
194  3                      char = substr (chars, 1, 1);
195  3                      j = 1;
196  3                      do j = 1 to n_units while (char ^= letters (j));
197  4                          end;
198  3                      if j <= n_units
199  3                          then do;                               /* match found */
200  4                              word_array (word_index) = j;       /* store 1-letter unit index */
201  4                              word_index = word_index + 1;
202  4                              end;
203  3                          else do;                               /* not found, unit is illegal */
204  4                              pronounceability = '0'b;
205  4                              return (pronounceability);
206  4                              end;
207  3                      end;
208  2                  end;
209  1          word_length = word_index - 1;
210  1          word_index = 0;
211  1
212  1     /* Now call random_word_, trying to get the word hyphenated.  Special versions of random_unit and */
213  1     /* random_vowel are supplied that return units of the word we are trying to hyphenate rather than */
214  1     /* random units.                                                                                   */
215  1
216  1          call random_word_ (returned_word, hyphenated_word, word_length_in_chars,
217  1                             returned_length, random_unit, random_vowel,
218  1                             d_ptr, l_ptr, r_ptr, n_units);
219  1          goto accepted;
220  1
221  1     /* If random_unit ever finds that random_word_ did not accept a unit from the word to be hyphenated,     */
222  1     /* a nonlocal goto directly to this label (which pops random_word_ off the stack) is made, and we        */
```

```
223  1     /* abort the whole operation.  If the last unit tried (i.e. the one not accepted) was a 2-letter unit,   */
224  1     /* we might be able to make the word legal by splitting that unit up into two 1-letter units and         */
225  1     /* starting all over.  Unfortunately, this is a lot of code and complication for a relatively rare case. */
226  1
227  1     not_accepted:
228  1             word_index = word_index - 1;                    /* index of last unit accepted */
229  1
230  1     accepted:
231  1             j = 1;
232  1             returned_hyphens = '0'b;
233  1             do i = 1 to word_length;
234  2                 if i > word_index & word_index < word_length        /* we never got done with the word */
235  2                 then do;
236  3                     pronounceability = '0'b;
237  3                     if letters_split (word_array (i)).second ^= ' '     /* was it not accepted because of */
238  3                     & split_point = 0                                   /* an illegal 2-letter unit?      */
239  3                       then if pronounceable_$split (word, returned_hyphens, i,
240  3                                 d_ptr, l_ptr, r_ptr, n_units)           /* try again with split pair */
241  3
242  3     /* Note: in even rarer cases, the unit that might be split to make this word legal is not the    */
243  3     /* unit that was rejected, but a previous unit.  It's too hard to deal with this case, so we'll  */
244  3     /* refuse the word, even though it might be legal.  As an example, using the standard digram     */
245  3     /* table, "preeg-hu-o" is a legal word.  However, our first attempt was to supply p-r-e-e-gh-u-o */
246  3     /* units.  Random_word_ rejects the "u" because it may not follow a "gh" unit in this context.   */
247  3     /* Since "u" is not a 2-letter unit, we can't try to split it up, so the word is thrown out.     */
248  3     /* However, p-r-e-e-g-h-u-o would have been acceptable to random_word_.  This is the only case   */
249  3     /* where a word that could have been produced by random_word_ will be rejected by this routine.  */
250  3
251  3                           then pronounceability = '1'b;       /* word was legal when 2-letter unit was split */
252  3                     return (pronounceability);
253  3                     end;
254  2
255  2     /* set returned_hyphens bits corresponding to character in word.  Note that   */
256  2     /* hyphens returned from random_word_ (hyphenated_word array) point to units, */
257  2     /* not characters.                                                            */
258  2
259  2                 if letters_split (word_array (i)).second ^= ' '
260  2                     then j = j + 2;
261  2                     else j = j + 1;
262  2                 returned_hyphens (j-1) = hyphenated_word (i);
263  2                 end;
264  1             pronounceability = '1'b;
265  1             return (pronounceability);
266  1
```

```
267 ;  1      /* The internal procedures random_unit and random_vowel keep track of the */
268 ;  1      /* acceptance or rejection of units they are supplying to random_word_.   */
269    1
270    1      random_unit:    proc (returned_unit);
271    2      dcl  returned_unit fixed bin;                    /* a unit from the word being tested */
272    2
273    2              vowel_flag = '0'b;
274    2              goto generate;
275    2
276    2      random_vowel:   entry (returned_unit);
277    2
278    2              vowel_flag = '1'b;
279    2
280    2
281    2      generate:
282    2
283 ;  2      /* get the next unit of the word being tested */
284    2
285    2              if returned_unit < 0 | (returned_unit = 0 & word_index ^= 0)
286    2                  then goto not_accepted;              /* if last unit was not accepted */
287    2              word_index = word_index + 1;
288    2              new_unit = word_array (word_index);      /* get next unit from word */
289    2              if vowel_flag                            /* if random_word_ wanted a vowel, and this next */
290    2                  then if ^rules.vowel (new_unit)      /* unit is not one, then we have to give up      */
291    2                      then if ^rules.alternate_vowel (new_unit)       /* can't give random_word_ a non-vowel */
292    2                          then goto not_accepted;                     /* when it expects a vowel             */
293    2              returned_unit = new_unit;
294    2              return;
295    2
296    2          end;
297    1
298    1
299    1      end pronounceable_;
300
301
```

```
302          /* ROUTINE random_word_
303          |
304          | FUNCTIONAL DESCRIPTION:
305          |
306          | This procedure generates a pronounceable random word of caller specified length
307          | and returns the word and the hyphenated (divided into syllables) form of the
308          | word.
309          |
310          | INPUT PARAMETERS:
311          |       hyphens -       position of hyphens, bit on indicates hyphen appears
312          |                       after corresponding unit in "word".
313          |
314          |       length -        length of word to be generated in letters.
315          |
316          |
317          |       random_unit -   routine to be called to generate a random unit.
318          |
319          |       random_vowel -  routine to be called to generate a random vowel.
320          |
321          |       d_ptr -
322          |       l_ptr -         pointers to digram table.
323          |       r_ptr -
324          |
325          |       n_units -       size of digram table (n_units x n_units).
326          |
327          | OUTPUT PARAMETERS:
328          |       word -          random word, 1 unit per array element.
329          |       word_length -   actual length of word in units.
330          |
331          | ROUTINE VALUE:
332          |       NONE
333          |
334          | SIDE EFFECTS:
335          |       NONE
336          |
337          */
338
339          random_word_: procedure (password, hyphenated_word, length, word_length,
340                                   random_unit, random_vowel, d_ptr, l_ptr, r_ptr,
341                                   n_units);
342        1
343        1 dcl  password(*) fixed bin;                          /* PARAMETER: unit number coded form of word */
344        1 dcl  hyphenated_word(*) bit(1) aligned;              /* PARAMETER: position of hyphens in word */
345        1 dcl  length fixed bin;                               /* PARAMETER: length of word in letters */
346        1 dcl  word_length fixed bin;                          /* PARAMETER: length of word in units */
347        1
348        1 dcl  n_units fixed bin;                              /* PARAMETER: number of units in unit table */
349        1 dcl  d_ptr ptr;                                      /* PARAMETER: pointer to digram table */
350        1 dcl  l_ptr ptr;                                      /* PARAMETER: pointer to unit letters table */
351        1 dcl  r_ptr ptr;                                      /* PARAMETER: pointer to unit rules table */
352        1
353        1
354      : 1 /* this array contains information about all possible pairs of units */
355        1
356        1 dcl 1 digrams(n_units, n_units) based(d_ptr),
357        1       2 begin bit(1),                /* on if this pair must begin syllable */
```

```
358    1            2 not_begin bit(1),       /* on if this pair must not begin */
359    1            2 end bit(1),             /* on if this pair must end syllable */
360    1            2 not_end bit(1),         /* on if this pair must not end */
361    1            2 break bit(1),           /* on if this pair is a break pair */
362    1            2 prefix bit(1),          /* on if vowel must precede this pair in same syllable */
363    1            2 suffix bit(1),          /* on if vowel must follow this pair in same syllable */
364    1            2 illegal_pair bit(1);    /* on if this pair may not appear */
365    1
366    1
367  ! 1       /* this array contains left justified 1 or 2-letter pairs representing each unit */
368    1
369    1       dcl letters(0:n_units) char(2) based(l_ptr);
370    1
371    1
372  ! 1       /* this is the same as letters, but allows reference to individual characters */
373    1
374    1       dcl 1 letters_split(0:n_units) based(l_ptr),
375    1             2 first char(1),
376    1             2 second char(1);
377    1
378    1
379  ! 1       /* this array has rules for each unit */
380    1
381    1       dcl 1 rules(n_units) based(r_ptr),
382    1             2 no_final_split bit(1),          /* can't be the only vowel in last syllable */
383    1             2 not_begin_syllable bit(1),      /* can't begin a syllable */
384    1             2 vowel bit(1),                   /* this is a vowel */
385    1             2 alternate_vowel bit(1);         /* this is an alternate vowel, (i.e., "y") */
386    1
387    1
388    1       dcl  random_unit entry (fixed bin);     /* get a unit */
389    1       dcl  random_vowel entry (fixed bin);    /* get a vowel unit */
390    1       dcl  unit fixed bin;                    /* a unit number from random_unit or random_vowel */
391    1
392    1       dcl  nchars fixed bin;                  /* number of characters in password */
393    1       dcl  index fixed bin init(1);           /* index of current unit in password */
394    1       dcl (first, second) fixed bin init(1);  /* index into digram table for current unit pair */
395    1       dcl  syllable_length fixed bin init(1); /* 1 when next unit is 1st in syllable, 2 if 2nd, etc. */
396    1
397    1       dcl  vowel_found bit(1) aligned;        /* set if vowel was found somewhere in syllable before this unit */
398    1       dcl  last_vowel_found aligned bit(1);   /* set if previous unit in this syllable was a vowel */
399    1       dcl  cons_count fixed bin init(0);      /* count of consecutive consonants in syllable preceeding current unit */
400    1
401    1       dcl  debug bit(1) aligned init('0'b);   /* debugging switch */
402    1       dcl  i fixed bin;
403    1
404    1               do i = 0 to length;
405    2                   password (i) = 0;
406    2                   hyphenated_word (i) = '0'b;
407    2                   end;
408    1               nchars = length;
409    1
410  ! 1       /* get rest of units in password */
411    1
412    1               unit = 0;
413    1               do index = 1 by 1 while (index <= nchars);
414    2                   if syllable_length = 1
```

```
                        then do;                        /* on first unit of a syllable, use any unit */
keep_trying:        unit = abs (unit);                  /* last unit was accepted (or first in word), make positive */
                    goto first_time;
retry:              unit = -abs (unit);                 /* last unit was not accepted, make negative */
first_time:
                    if index = nchars                   /* if last unit of word must be a syllable, it must be a vowel */
                        then call random_vowel (unit);
                        else call random_unit (unit);
                    password (index) = abs (unit);              /* put actual unit in word */
                    if index ^= 1
                        then if digrams (password (index-1), password (index)).illegal_pair
                            then goto retry;                    /* this pair is illegal */
                    if rules (password (index)).not_begin_syllable
                        then goto retry;
                    if letters_split.second (password (index)) ^= ' '
                        then if index = nchars
                            then goto retry;
                            else if index = nchars-1 & ^rules (password (index)).vowel
                                    & ^rules (password (index)).alternate_vowel
                                then goto retry;                /* last unit was a double-letter unit and not a vowel */
                                else if unit < 0
                                    then goto keep_trying;
                                    else nchars = nchars - 1;
                        else if unit < 0
                            then goto keep_trying;
                    syllable_length = 2;
                    if rules (password (index)).vowel | rules (password (index)).alternate_vowel
                        then do;
                            cons_count = 0;
                            vowel_found = '1'b;
                            end;
                        else do;
                                else do;
                                    cons_count = 1;
                                    vowel_found = '0'b;
                                    end;
                            last_vowel_found = '0'b;
                            end;
                        else do;
                            call generate_unit;
                            if second = 0 then goto all_done;       /* we have word already */
                            end;
                        end;

    /* enter here at end of word */

    all_done:
            word_length = index - 1;
            return;

    end;
```

```
/* ROUTINE procedure generate_unit

!
! FUNCTIONAL DESCRIPTION:
!
! generate next unit to password, making sure that it follows these rules:
! 1. Each syllable must contain exactly 1 or 2 consecutive vowels,
!     where y is considered a vowel.
! 2. Syllable end is determined as follows:
!    a. Vowel is generated and previous unit is a consonant and
!       syllable already has a vowel.  In this case new syllable is
!       started and already contains a vowel.
!    b. A pair determined to be a "break" pair is encountered.
!       In this case new syllable is started with second unit of this pair.
!    c. End of password is encountered.
!    d. "begin" pair is encountered legally.  New syllable is started
!       with this pair.
!    e."end" pair is legally encountered.  New syllable has nothing yet.
! 3. Try generating another unit if:
!    a. third consecutive vowel and not y.
!    b. "break" pair generated but no vowel yet in current syllable
!       or previous 2 units are "not_end".
!    c. "begin" pair generated but no vowel in syllable preceeding
!       begin pair, or both previous 2 pairs are designated "not_end".
!    d. "end" pair generated but no vowel in current syllable or in "end" pair.
!    e. "not_begin" pair generated but new syllable must begin
!       (because previous syllable ended as defined in 2 above).
!    f. vowel is generated and 2a is satisfied, but no syllable break is
!        possible in previous 3 pairs.
!    g. Second & third units of syllable must begin, and first unit is
!         "alternate_vowel".
!
! The done routine checks for required prefix vowels & end of word conditions.
!
! INPUT PARAMETERS:
!       NONE
!
! OUTPUT PARAMETERS:
!       NONE
!
! ROUTINE VALUE:
!       NONE
!
! SIDE EFFECTS:
!       NONE
!
**/

generate_unit: procedure;

dcl 1 x,                    /* rules for the digram currently being tested*/
      2 begin bit(1),             /* on if this pair must begin syllable */
      2 not_begin bit(1),         /* on if this pair must not begin */
      2 end bit(1),               /* on if this pair must end syllable */
      2 not_end bit(1),           /* on if this pair must not end */
      2 break bit(1),             /* on if this pair is a break pair */
```

```
                2 prefix bit(1),          /* on if vowel must precede this pair in same syllable */
                2 suffix bit(1),          /* on if vowel must follow this pair in same syllable */
                2 illegal_pair bit(1);    /* on if this pair may not appear */

        dcl  unit_count fixed bin init (1);     /* count of tries to generate this unit */
        dcl  try_for_vowel bit(1) aligned;      /* set if next unit needed is a vowel */
        dcl  v bit(1) aligned;                  /* set if last unit generated is a vowel, or an */
                                                /* alternate vowel to be treated as a vowel     */

        dcl  i fixed bin;


                first = password (index-1);

        /* on last unit of word and no vowel yet in syllable, or if previous pair */
        /* requires a vowel and no vowel in syllable, then try only for a vowel   */

                if syllable_length = 2                  /* this is the second unit of syllable */
                    then try_for_vowel = ^vowel_found & index=nchars; /* last unit of word and no vowel yet, try for vowel *
                    else                                /* this is at least the third unit of syllable */
                    if ^vowel_found ¦ digrams (password (index-2), first).not_end
                        then try_for_vowel = digrams (password (index-2), first).suffix;
                        else try_for_vowel = '0'b;
                goto keep_trying;                       /* on first try of a unit, don't make the tests below */

        /* come here to try another unit when previous one was not accepted */

        try_more:
                unit = -abs (unit);                     /* last unit was not accepted, set sign negative */
                if unit_count = 100
                    then do;
                        if debug
                            then do;
                                put edit ('100 tries failed to generate unit.', 'password so far is: ')
                                    (a, skip, a);
                                do i = 1 to index;
                                    put edit (letters (password (i)))  (a);
                                    end;
                                put skip;
                                end;
                        call random_word_ (password, hyphenated_word, length, index,
                                           random_unit, random_vowel, d_ptr, l_ptr,
                                           r_ptr, n_units);
                        second = 0;
                        return;
                        end;

        /* come here to try another unit whether last one was accepted or not */

        keep_trying:
                if try_for_vowel
                    then call random_vowel (unit);
                    else call random_unit (unit);
                second = abs (unit);                    /* save real value of unit number */
                if unit > 0
                    then unit_count = unit_count + 1;   /* count number of tries */
```

```
578 :  2      /* check if this pair is legal */
579    2
580    2            if digrams (first, second).illegal_pair
581    2                then goto try_more;
582    2                else if first = second              /* if legal, throw out 3 in a row */
583    2                    then if index > 2
584    2                        then if password (index-2) = first
585    2                            then goto try_more;
586    2            if letters_split (second).second ^= ' ' /* check if this is 2 letters */
587    2                then if index = nchars              /* then if this is the last unit of word */
588    2                    then goto try_more;             /* then a two-letter unit is illegal */
589    2                    else nchars = nchars - 1;       /* otherwise decrement number of characters */
590    2            password (index) = second;
591    2            if rules (second).alternate_vowel
592    2                then v = ^rules (first).vowel;
593    2                else v = rules (second).vowel;
594    2            x = digrams (first, second);
595    2            if syllable_length > 2                                  /* force break if last pair must be followed */
596    2                then if digrams (password (index-2), first).suffix  /* by a vowel and this unit is not a vowel   */
597    2                    then if ^v
598    2                        then break = '1'b;          /* if last pair was not_end, new_unit gave us a vowel */
599    2
600 :  2      /* In the notation to the right, the series of letters and dots stands          */
601 :  2      /* for the last n units in this syllable, to be interpreted as follows:        */
602 :  2      /*      v stands for a vowel (including alternate_vowel)                       */
603 :  2      /*      c stands for a consonant                                               */
604 :  2      /*      x stands for any unit                                                  */
605 :  2      /* the dots are interpreted as follows (c is used as example)                  */
606 :  2      /*      c...c  one or more consecutive consonants                              */
607 :  2      /*      c..c   zero or more consecutive consonants                             */
608 :  2      /*      ...c   one or more consecutive consonants from beginning of syllable   */
609 :  2      /*      ..c    zero or more consecutive consonants from beginning of syllable  */
610 :  2      /* the vertical line ¦ marks a syllable break.
611 :  2      /* The group of symbols indicates what units there are in current
612 :  2      /* syllable.  The last symbol is always the current unit.
613 :  2      /* The first symbol is not necessarily the first unit in the
614 :  2      /* syllable, unless preceeded by dots.  Thus, "vcc..cv" should be
615 :  2      /* interpreted as "..xvcc..cv" (i.e., add "..x" to the beginning of all
616 :  2      /* syllables unless dots begin the syllable.). */
617    2
618    2            if syllable_length = 2 & not_begin      /* pair may not begin syllable */
619    2                then goto loop;                     /* rule 3e. */
620    2            if vowel_found
621    2                then if cons_count ^= 0
622    2                    then if begin                   /* vc...cx */
623    2                        then if syllable_length ^= 3 & not_end_(3) /* vc...cx begin */
624    2                            then                    /* can we break at vc..c¦cx */
625    2                            if not_end_ (2)         /* no, try a break at vc...c¦x */
626    2                                then goto loop;     /* rule 3c. */
627    2                                else call done (v, 2);      /* vc...c¦x begin, treat as break */
628    2                            else call done (v, 3);          /* vc..c¦cx begin */
629    2                        else if not_begin                   /* vc...cx ^begin */
630    2                            then if break                   /* vc...cx not_begin */
631    2                                then if not_end_ (2)        /* vc...c¦x break */
632    2                                    then goto loop;         /* rule 3b, can't break */
633    2                                    else call done (v, 2);  /* vc...c¦x break */
634    2                                else if v                   /* vc...cx ^break not_begin */
```

```
                                then                        /* vc...cv ^break not_begin */
                                if not_end_ (2)             /* try break at vc...c!v */
                                    then goto loop;         /* rule 3f, break no good */
                                    else call done ('1'b, 2);   /* vc...c!v treat as break */
                                else if end                 /* vc...cc ^break not_begin */
                                    then call done ('0'b, 1);   /* vc...cc! end */
                                    else call done ('1'b, 0);   /* vc...cc ^break ^end not_begin */
                            else if v                       /* vc...cx ^begin ^not_begin */
                                then                        /* vc...cv rule 2a says we must break somewhere */
                                if not_end_ (3) & syllable_length ^= 3
                                    then if not_end_ (2)    /* vc..c!cv doesn't work */
                                        then if cons_count > 1      /* vc...c!v doesn't work */
                                            then                    /* vc...ccv */
                                            if not_end_ (4)         /* try vc..c!ccv */
                                            ! digrams (password (index-2), first).not_begin
                                                then goto loop;     /* rule 3f */
                                                else call done ('1'b, 4); /* vc...c!ccv */
                                            else goto loop; /* vc...c!v and vc..c!cv are no good */
                                        else call done ('1'b, 3);   /* vc...c!v treat as break */
                                    else call done ('1'b, 3);       /* vc..c!cv treat as break */
                                else call done ('1'b, 0);   /* vc...cc ^begin ^not_begin */
                        else            /* vowel found and last unit is not consonant => last unit is vowel */
                        if v & rules.vowel (password (index-2)) & index > 2
                            then goto loop;                 /* rule 3a, 3 consecutive vowels non-y */
                            else if end                     /* vx */
                                then call done ('0'b, 1);       /* vx end */
                                else if begin                   /* vx ^end */
                                    then if last_vowel_found    /* vx begin */
                                        then if v               /* v...vvx begin */
                                            then if syllable_length = 3 /* v...vvv begin */
                                                then if rules(password((index-2))).alternate_vowel /* !vvv begin */
                                                    then goto loop;     /* rule 3g, !'y'!vv is no good */
                                                    else call done('1'b, 3); /* !v!vv begin */
                                                else if not_end_ (3)    /* v...vvv begin */
                                                    then goto loop;     /* rule 3c, v...v!vv no good */
                                                    else call done('1'b, 3); /* v...v!vv begin */
                                            else if syllable_length = 3 /* v...vvc begin */
                                                then if rules.alternate_vowel(password(index-2)) /* !vvc begin */
                                                    then goto loop;     /* rule 3g, !'y'!vc is no good */
                                                    else if rules.vowel(password(index-2)) /* !x!vc begin */
                                                        then call done('1'b, 3); /* !v!vc begin */
                                                        else goto loop; /* !c!vc begin is illegal */
                                                else if not_end_ (3)    /* v...vvc begin */
                                                    then                /* v...vvc begin try to split pair */
                                                    if not_end_ (2)     /* v...vvc begin */
                                                        then goto loop; /* v...vv!c no good */
                                                        else call done('0'b, 2); /* v...vv!c */
                                                    else call done('1'b, 3); /* v...v!vc begin */
                                        else                    /* try splitting begin pair */
                                        if syllable_length > 2  /* ..cvx begin */
                                            then if not_end_ (2)    /* ...cvx begin */
                                                then goto loop;     /* rule 3c, ...cv!x no good */
                                                else call done (v, 2);  /* ...cv!x begin */
                                            else call done ('1'b, 0);   /* !vx begin */
                                else if break               /* ..xvx ^begin ^end */
                                    then if not_end_(2) & syllable_length > 2 /* ..xvx break */
                                        then goto loop;         /* rule 3b, ..xv!x is no good */
```

```
692    2                                    else call done (v, 2);      /* ..v!x break */
693    2                                else call done ('1'b, 0);       /* ..vx ^end ^begin ^break */
694    2                        else if break                           /* ...cx */
695    2                            then goto loop;                     /* rule 3b, ...c!x break no good */
696    2                            else if end                         /* ...cx ^break */
697    2                                then if v                       /* ...cx end */
698    2                                    then call done ('0'b, 1);   /* ...cv! end (new syllable) */
699    2                                    else goto loop;             /* rule 3b, ...cc! end no good */
700    2                                else if v                       /* ...cx ^end ^break */
701    2                                    then if begin & syllable_length > 2     /* ...cv ^end ^break */
702    2                                        then goto loop;                     /* c...c!cv ^end ^break begin, rule 3c */
703    2                                        else call done ('1'b, 0);           /* ...cv ^end ^break ^begin */
704    2                                    else if begin                           /* ...cc ^break ^end */
705    2                                        then if syllable_length > 2         /* ..ccc begin */
706    2                                            then goto loop;                 /* rule 3c, ...ccc begin */
707    2                                            else call done ('0'b, 3);       /* !cc begin */
708    2                                        else call done ('0'b, 0);           /* ..xcc ^end ^break ^begin */
709    2
710 !  2        /* ******** return here when unit generated has been accepted ****** */
711    2
712    2                     return;
713    2
714 !  2        /* ******** enter here when unit generated was good, but we don't want to use it because ********* */
715 !  2        /* ******** it was supplied as a negative number by random_unit or random_vowel          ********* */
716    2
717    2        accepted_but_keep_trying:
718    2                if letters_split (second).second ^= ' '
719    2                    then nchars = nchars + 1;   /* pretend unit was no good */
720    2                unit = -unit;                   /* make positive to say that it would have been accepted */
721    2                goto keep_trying;
722    2
723 !  2        /* ******** enter here when unit generated is no good ******* */
724    2
725    2        loop:   if letters_split (second).second ^= ' '
726    2                    then nchars = nchars + 1;
727    2                goto try_more;
728    2
```

```
729  1  2                    /***   procedure done   ***/
730
731  1  2          /* this routine is internal to generate_unit because it can return to loop. */
732  1  2          /* call done when new unit is generated and determined to be                 */
733  1  2          /* legal. Arguments are new values of:                                       */
734  1  2          /*      vf  vowel_found                                                      */
735  1  2          /*      sl  syllable_length (number of units in syllable.                    */
736  1  2          /*                           0 means increment for this unit)                */
737  1  2
738  1  2          done:   procedure (vf, sl);
739  1  2          dcl  vf bit (1) aligned;        /* set if vowel found in this syllable before this unit */
740  1  2          dcl  sl fixed bin;              /* number of units in syllable (0 if this unit is to be */
741  1              /* added to the current syllable).                                           */
742  1  2
743  1  2          /* if we are not within first 2 units of syllable, check if */
744  1  2          /* vowel must precede this pair                             */
745  1  2
746  1  2                  if sl ^= 2
747  1  2                      then if syllable_length ^= 2
748  1  2                          then if prefix
749  1  2                              then if ^rules.vowel (password (index-2))
750  1  2                                  then                    /* vowel must precede pair but no vowel precedes pair */
751  1  2                                  if vowel_found          /* if there is a vowel in this syllable, */
752  1  2                                      then                /* we may be able to break this pair     */
753  1  2                                      if not_end_ (2)             /* check if this pair may be treated as break */
754  1  2                                          then goto loop;         /* no, previous 2 units can't end */
755  1  2                                          else do;                /* yes, break can be forced */
756  1  2                                              call done ('0'b, 2); /* ...cxx or ...cvx */
757  1  2                                              return;
758  1  2                                              end;
759  1  2                                      else goto loop;             /* no vowel in syllable */
760  1  2
761  1  2          /* Check end of word conditions.  If end of word is reached: */
762  1  2          /*      1. We must have a vowel in current syllable,         */
763  1  2          /*      2. This pair must be allowed to end syllable         */
764  1  2
765  1  2                  if sl ^= 1
766  1  2                      then if index = nchars
767  1  2                          then if not_end
768  1  2                              then goto loop;
769  1  2                              else if vf = '0'b
770  1  2                                  then goto loop;
771  1  2
772  1              /* A final "e" may not be the only vowel in the last syllable. */
773  1  2
774  1  2                  if index = nchars
775  1  2                      then if rules (second).no_final_split       /* this bit is on for "e" */
776  1  2                          then if sl ^= 1
777  1  2                              then if rules.vowel (first)         /* e preceded by vowel is ok, however */
778  1  2                                  then;
779  1  2                                  else if ^vowel_found|syllable_length<3  /* otherwise previous 2 letters must */
780  1  2                                      then goto loop;                     /* be able to end the syllable      */
781  1  2                                      else if unit < 0
782  1  2                                          then goto accepted_but_keep_trying;
783  1  2                                          else sl = 0;
784  1  2                  if unit < 0
```

```
785     3               then goto accepted_but_keep_trying;
786                 if v ¦ sl = 1
787                     then cons_count = 0;                /* this unit is a vowel or new syllable is to begin */
788                     else if sl = 0
789                         then cons_count = cons_count + 1;       /* this was a consonant, increment count */
790                         else cons_count = min(sl-1, cons_count+1);      /* a new syllable was started some letters back, */
791 ¦                                                                       /* cons_count gets incremented, but no more than */
792 ¦                                                                       /* number of units in syllable                   */
793                 if sl = 0
794                     then syllable_length = syllable_length + 1;
795                     else syllable_length = sl;
796                 if syllable_length > 3
797                     then last_vowel_found = vowel_found;
798                     else last_vowel_found = '0'b;
799                 vowel_found = vf;
800                 if index - syllable_length + 1 ^= nchars
801                     then hyphenated_word (index - syllable_length + 1) = '1'b;
802
803                 end done;
804
805     2       end generate_unit;
806     1
```

```
  807 !  1                        /***   procedure not_end_   ***/
  808    1
  809 !  1      /* not_end_(i) returns '1'b when ( password(index-i), password(index-i+1) ) may */
  810 !  1      /* not end a syllable, or when password(index-i+2) may not begin a syllable     */
  811    1
  812    1      not_end_:    procedure (i) returns (bit (1));
  813    2      dcl  i fixed bin;
  814    2
  815    2              if i = index
  816    2                  then return (^rules.vowel (password (1)));
  817    2              if i ^= 1
  818    2                  then if rules.not_begin_syllable (password (index-i+2))
  819    2                      then return ('1'b);
  820    2              return (digrams (password (index-i), password (index-i+1)).not_end);
  821    2
  822    2          end not_end_;
  823    1
  824    1
  825    1      end random_word_;
```

COMMAND LINE

```
PLI/LIS=LIS$:PRONOUNCE/OBJ=OBJ$:PRONOUNCE MSRC$:PRONOUNCE
```

PRONOUNCE
LIS

QUEMAN
LIS

MATCHKEY
LIS

PUTCLIMSG
LIS

QUEMANMSG
LIS

PASSWORDS
LIS

QUEMANSHO
LIS