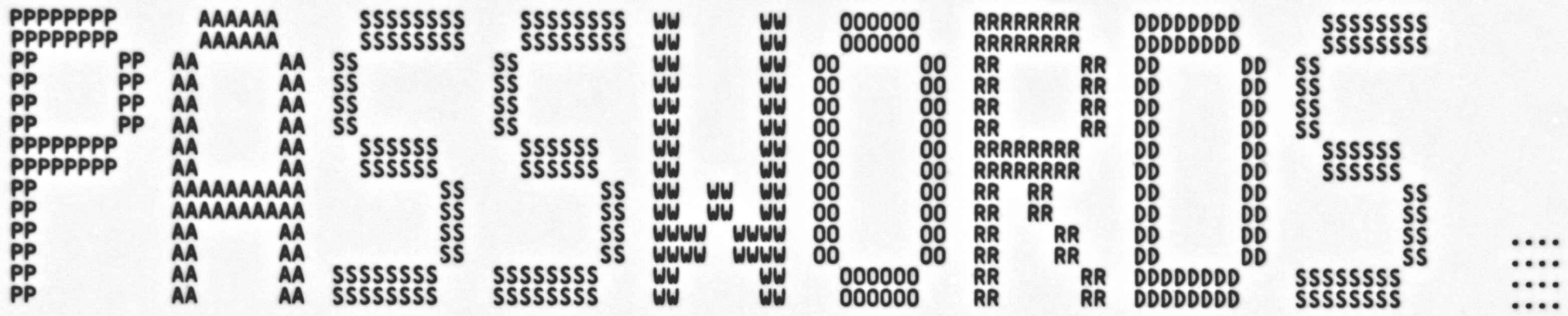
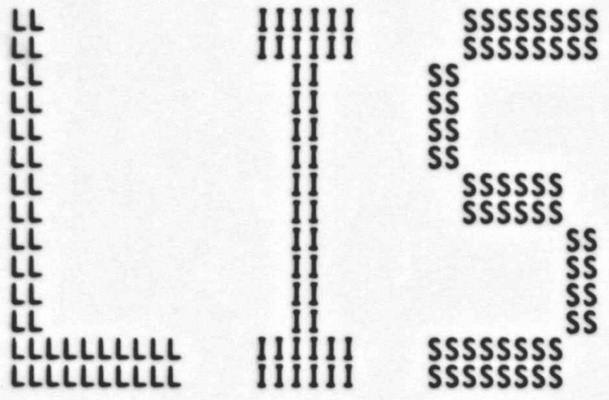
```
   CCCCCCCCCCC  LLL              IIIIIIIII     UUU         UUU  TTTTTTTTTTTTTTT  LLL
   CCCCCCCCCCC  LLL              IIIIIIIII     UUU         UUU  TTTTTTTTTTTTTTT  LLL
   CCCCCCCCCCC  LLL              IIIIIIIII     UUU         UUU  TTTTTTTTTTTTTTT  LLL
CCC             LLL                 III      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
```


PPPPPPPP AAAAAA SSSSSSSS SSSSSSSS WW WW OOOOOO RRRRRRRR DDDDDDDD SSSSSSSS
PPPPPPPP AAAAAA SSSSSSSS SSSSSSSS WW WW OOOOOO RRRRRRRR DDDDDDDD SSSSSSSS
PP PP AA AA SS SS WW WW OO OO RR RR DD DD SS
PP PP AA AA SS SS WW WW OO OO RR RR DD DD SS
PP PP AA AA SS SS WW WW OO OO RR RR DD DD SS
PP PP AA AA SS SS WW WW OO OO RR RR DD DD SS
PPPPPPPP AA AA SSSSSS SSSSSS WW WW OO OO RRRRRRRR DD DD SSSSSS
PPPPPPPP AA AA SSSSSS SSSSSS WW WW OO OO RRRRRRRR DD DD SSSSSS
PP AAAAAAAAAA SS SS WW WW WW OO OO RR RR DD DD SS
PP AAAAAAAAAA SS SS WW WW WW OO OO RR RR DD DD SS
PP AA AA SS SS WWWW WWWW OO OO RR RR DD DD SS
PP AA AA SS SS WWWW WWWW OO OO RR RR DD DD SS
PP AA AA SSSSSSSS SSSSSSSS WW WW OOOOOO RR RR DDDDDDDD SSSSSSSS
PP AA AA SSSSSSSS SSSSSSSS WW WW OOOOOO RR RR DDDDDDDD SSSSSSSS


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

```
1       /*
2       :       IDENT = V04-000
3       :
4       :*****************************************************************************
5       :*                                                                           *
6       :*  COPYRIGHT (c) 1978, 1980, 1982, 1984 BY                                  *
7       :*  DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASSACHUSETTS.                   *
8       :*  ALL RIGHTS RESERVED.                                                     *
9       :*                                                                           *
10      :*  THIS SOFTWARE IS FURNISHED UNDER A LICENSE AND MAY BE USED AND COPIED    *
11      :*  ONLY IN  ACCORDANCE WITH  THE  TERMS  OF  SUCH  LICENSE  AND WITH THE   *
12      :*  INCLUSION OF THE ABOVE COPYRIGHT NOTICE. THIS SOFTWARE OR  ANY  OTHER    *
13      :*  COPIES THEREOF MAY NOT BE PROVIDED OR OTHERWISE MADE AVAILABLE TO ANY    *
14      :*  OTHER PERSON.  NO TITLE TO AND OWNERSHIP OF  THE  SOFTWARE IS  HEREBY    *
15      :*  TRANSFERRED.                                                             *
16      :*                                                                           *
17      :*  THE INFORMATION IN THIS SOFTWARE IS  SUBJECT TO CHANGE WITHOUT NOTICE    *
18      :*  AND  SHOULD  NOT  BE  CONSTRUED AS  A COMMITMENT BY DIGITAL EQUIPMENT    *
19      :*  CORPORATION.                                                             *
20      :*                                                                           *
21      :*  DIGITAL ASSUMES NO RESPONSIBILITY FOR THE USE  OR  RELIABILITY OF ITS    *
22      :*  SOFTWARE ON EQUIPMENT WHICH IS NOT SUPPLIED BY DIGITAL.                  *
23      :*                                                                           *
24      :*                                                                           *
25      :*****************************************************************************
26      :
27      :++
28      : FACILITY:
29      :
30      :       SET PASSWORD
31      :
32      : ABSTRACT:
33      :
34      :       This is main routine for the /GENERATE qualifier on SET PASSWORD.
35      :
36      : ENVIRONMENT:
37      :
38      :       Vax native
39      :
40      :--
41      :
42      : AUTHOR: Brian Bailey  , CREATION DATE: Summer 83
43      :
44      : MODIFIED BY:
45      :
46      :       V03-004 SHZ0004         Stephen H. Zalewski,    27-Mar-1984
47      :               Change maximum password length from 80 to 32 because that
48      :               is what the calling routine assumes, and this routine blank
49      :               fills the password buffer.
50      :
51      :       V03-003 SHZ0003         Stephen H. Zalewksi,    21-Mar-1984
52      :               Add more words to the bad word list.
53      :
54      :       V03-002 SHZ0002         Stephen H. Zalewski     10-Feb-1984
55      :               Call SYS$EXIT when user types CNTRL/Z to new password prompt
```

```
    56 !           !               in order to prevent the "PLI Finish Condition" from being
    57 !           !               signaled.
    58 !           !
    59 !           !       V03-001 SHZ0001         Stephen H. Zalewski     01-feb-1984
    60 !           !               Extensive rewriting to implement /GENERATE and incorporate
    61 !           !               into SET PASSWORD.
    62 !           !
    63 !           !**
    64 !           */
    65
    66
```

```
        /* ROUTINE set_password_generate
        |
        | FUNCTIONAL DESCRIPTION:
        |
        |       This is the main routine for SET PASSWORD/GENERATE.  It call the
        |       routines to generate the passwords and returns when a password
        |       has been chosen by the user.
        |
        | INPUT PARAMETERS:
        |       mini - The minimum password length to generate. (The passwords generated
        |              are of length mini to mini+2.)
        |
        | OUTPUT PARAMETERS:
        |       new_password - An ASCIC string of the password chosen by the user.
        |
        | ROUTINE VALUE:
        |       NONE
        |
        | SIDE EFFECTS:
        |       NONE
        |
        */

        set_password_generate: proc (new_password,mini);

    1   dcl new_password char(32) var;                  /* password to be entered by user */
    1   dcl mini fixed bin;                             /* minimum password length */
    1
    1   dcl generate_passwords entry ((*) char(20) var, /* generate the passwords */
            (*) char(40) var, fixed bin, fixed bin, fixed bin);
    1
    1   dcl sys$exit external entry (fixed bin(31) value);/* called by enfile to exit */
    1   dcl (ss$_normal) globalref fixed bin value;
    1
    1   dcl password(100) char(20) var;                 /* passwords to be generated */
    1   dcl hyph_word(100) char(40) var;                /* hyphenate form of passwords to help */
    1                                                   /* the user with their pronunciation   */
    1
    1   dcl more bit(1) aligned;                        /* set when more passwords are to be generated */
    1   dcl on_list bit(1) aligned;                     /* set when new_password matches one of  */
    1                                                   /* the passwords in passwords(1:n_words) */
    1
    1   dcl n_words fixed bin init(5);                  /* number of passwords to generate */
    1   dcl maxi fixed bin;                             /* maximum password length */
    1
    1   dcl collate builtin;
    1
    1   dcl translation character(128);                 /* Used to translate upper to lower case       */
    1
    1   dcl aligning_blanks char(16) var;               /* align the words for printing */
    1   dcl (i, j, b, v) fixed bin;
    1
    1
    1       on endfile(sysin) call sys$exit(ss$_normal);
    1
    1      /* set up translation buffer to change upper case letters to lower case.     */
```

```
123 :  1          /* this is used to change the password entered by the user.                    */
124    1           translation = collate;
125    1           do i=66 to 91;                          /* replace upper with lower */
126    2               substr(translation,i,1) = substr(collate,i+32,1);
127    2               end;
128    1
129 :  1          /* generate the words */
130    1           maxi = mini + 2;                                    /* set maximum word length */
131    1           more = '1'b;
132    1           on_list = '0'b;
133    1           do while (more);
134    2               call generate_passwords (password, hyph_word, mini, maxi, n_words);
135    2               do while(^on_list);
136    3                   put skip(2)
137    3                       edit('Choose a password from this list, or press RETURN to get a new list') (a);
138    3                   put skip;
139    3                   get list(new_password) options(prompt('New password: '), no_echo);
140    3                   put skip;
141    3                   if new_password = ''
142    3                       then do;
143    4                           put skip;
144    4                           call generate_passwords(password, hyph_word, mini, maxi, n_words);
145    4                           end;
146    3                       else do;
147 :  4                           /* see if the new_password matches any of the passwords on the list */
148    4                           new_password = translate(new_password,translation);
149    4                           i = 1;
150    4                           do while(^on_list  &  i <= n_words);
151    5                               if new_password = password(i)
152    5                                   then on_list = '1'b;
153    5                               i = i + 1;
154    5                               end;
155    4                           if ^on_list
156    4                               then do;
157    5                                   put skip edit('That word is not on this list: ') (a);
158    5                                   put skip;
159    5                                   do i = 1 to n_words;
160    6                                       aligning_blanks = copy(' ',maxi-length(password(i)));
161    6                                       put skip edit(password(i)||aligning_blanks,hyph_word(i)) (a,x(4),a);
162    6                                       end;
163    5                                   end;
164    4                               else
165    4                                   more = '0'b;
166    4                           end;
167    3                   end;
168    2               end;
169    1
170    1       end set_password_generate;
171
172
```

```
173 ¦         /* ROUTINE generate_passwords
174 ¦          ¦
175 ¦          ¦ FUNCTIONAL DESCRIPTION:
176 ¦          ¦
177 ¦          ¦ Generate a number of pronounceable words according to the rules given in the
178 ¦          ¦ digram table and in the algorithm in random_word_:  First generate an evenly
179 ¦          ¦ distributed random letter string.  Then test it using the random_word_ routine.
180 ¦          ¦ If it is legal, keep it.  If it is not, generate another and test it.  Repeat
181 ¦          ¦ this process until the desired number of pronounceable words is obtained.
182 ¦          ¦
183 ¦          ¦ This algorithm requires an initial seed before it generates any numbers.  In
184 ¦          ¦ order to make this initial seed as secure from detection as possible, and also
185 ¦          ¦ to make this program transportable to all types of VAXen, a longword counter
186 ¦          ¦ is incremented in a tight loop, and the lowest two bits are read 32 times at
187 ¦          ¦ 10 millisecond intervals and concatenated together to form the 64-bit initial
188 ¦          ¦ seed.
189 ¦          ¦
190 ¦          ¦ INPUT PARAMETERS:
191 ¦          ¦       n_words          - number of words to generate
192 ¦          ¦       min_length       - minimum length of words to be generated
193 ¦          ¦       max_length       - maximum length of words to be generated
194 ¦          ¦
195 ¦          ¦
196 ¦          ¦ OUTPUT PARAMETERS:
197 ¦          ¦       password         - legal words in both normal and hyphenated form
198 ¦          ¦       hyph_password      (Output words are printed 1 per line as well
199 ¦          ¦                          being passed back to the calling procedure)
200 ¦          ¦
201 ¦          ¦ ROUTINE VALUE:
202 ¦          ¦       NONE
203 ¦          ¦
204 ¦          ¦ SIDE EFFECTS:
205 ¦          ¦       NONE
206 ¦          ¦
207 ¦         */
208
209           generate_passwords: proc (password, hyph_password, min_length, max_length, n_words);
210     1
211     1     %replace n_units by 34;
212     1
213     1     dcl  password(*) char(20) var;                  /* PARAMETER: list of words after they have been generated */
214     1     dcl  hyph_password(*) char(40) var;             /* PARAMETER: hyphenated form of passwords in above list   */
215     1     dcl  min_length fixed bin;                      /* PARAMETER: min length of word to be generated */
216     1     dcl  max_length fixed bin;                      /* PARAMETER: max length of word to be generated */
217     1     dcl  n_words fixed bin;                         /* PARAMETER: number of words to be generated */
218     1     dcl  word_count fixed bin;                      /* number of words generated so far */
219     1
220     1
221     1     dcl  random_chars_ entry (char(*) var, fixed bin);      /* get a string of random letters */
222     1     dcl  word char(20) var;                                 /* letter string to be tested */
223     1     dcl  word_length fixed bin;                             /* length of word to be tested */
224     1     dcl  s bit(31) aligned;                                 /* temporary binary form of word_length */
225     1     dcl  n fixed bin;                                       /* temporary integer form of word_length */
226     1     dcl  aligning_blanks char(16) var;                      /* blanks to align word for printing */
227     1
228     1
```

```
229  1      dcl  pronounceable_ entry (char(*), (*) bit(1) aligned, /* test the pronounceability of word */
230  1           ptr, ptr, ptr, fixed bin)  returns (bit(1));
231  1      dcl  word_is_pronounceable bit(1) aligned;              /* status of word after being tested */
232  1
233  1
234  1      dcl  get_random_bits entry(bit(64) aligned);    /* random number generator */
235  1      dcl  seed bit(64) aligned ext;                  /* random number generator seed */
236  1      dcl  seed_bits bit(2) aligned;                  /* initial random number seed bits */
237  1      dcl  next_field fixed bin;                      /* location in seed string of 2-bit field which */
238  1                                                      /* will be filled by next seed_bits string      */
239  1
240  1
241  1      dcl  counter fixed bin;                         /* longword counter whose lowest two */
242  1                                                      /* bits will be stored in seed_bits  */
243  1      dcl  read_flag bit(1) aligned ext;              /* set when counter value is to be read */
244  1      dcl  delta_time bit(64) aligned;                /* time value for sys$setimr */
245  1      dcl  ten_msec char(13) init('0 00:00:00.01');   /* 10 millisecond delta_time value */
246  1
247  1
248  1      dcl sys$bintim entry (                          /* convert ASCII string to binary time value */
249  1              char(*),                                /*     - ASCII string:          ten_msec     */
250  1              bit(64) aligned)                        /*     - system time value:     delta_time   */
251  1          returns (fixed binary(31));
252  1
253  1      dcl sys$setimr entry (                          /* timer request with AST interrupt           */
254  1              fixed bin(31) value,                    /*     - event flag number:     NOT USED      */
255  1              bit(64) aligned,                        /*     - time value:            delta_time    */
256  1              entry value,                            /*     - AST procedure:         set_read_flag */
257  1              fixed bin(31) value)                    /*     - AST parameter:         NOT USED      */
258  1          options(variable) returns(fixed bin(31));
259  1
260  1      dcl sys$cantim entry(any,any)                   /* cancel the current timer request */
261  1          options(variable) returns(fixed bin(31));
262  1
263  1      dcl sts$value fixed binary(31);                 /* system service return status value */
264  1      dcl 1 sts$fields based (addr(sts$value)),       /* sts$value broken down into bits */
265  1            2 sts$success bit(1),                             /* set when system service completed successfully */
266  1            2 sts$pad bit(31),                                /* padding to make 1 longword */
267  1            2 sts$align char(0);                              /* for byte alignment */
268  1
269  1      /* possible values of sts$value */
270  1
271  1      dcl (ss$_normal,
272  1           ss$_accvio,
273  1           ss$_exquota,
274  1           ss$_insfmem ) globalref fixed bin value;
275  1
276  1
277  1      dcl collate builtin;
278  1
279  1      dcl translation character(128);                 /* used to encrypt generated passwords */
280  1
281  1      dcl number_bad_words fixed bin init(93);
282  1      dcl bad_word_string(93) char(10) var
283  1          init(
284  1              'tiju','gvl','gvdl','btti','ifmm',
285  1              'dvou','cjudi','ujut','ojhhfs','gvr',
```

```
286   1              'mvm','qjl','tojllfm','lmppu','{bl',
287   1              'usvu','ipfs','tmfu','efm','uvu',
288   1              'ofvl','obbj','hfjm','lf{fo','tuspou',
289   1              'lvu','usvu','gmjllfs','qppu','qpu',
290   1              'njfu','ipnp','lpou','hbu','cjm',
291   1              'sffu','bbst','lfwfs','wfsepnnf','cbmmf',
292   1              'ekfwfm','esjuf','esjuu','gbfo','gbo',
293   1              'gboefo','gjuuf','gpsqvmu','ifmwfuf','kvllf',
294   1              'lovmm','lvl','qjll','qspng','qvm',
295   1              'qvmu','sbtt','spol','svol','twjo',
296   1              'ujtqf','ujtt','qvub','qvubt','qpzb',
297   1              'qpzbt','qpmmb','qpmmbt','hjmjqpzbt','hjmjqpmmbt',
298   1              'dbqvmmp','dbqvmmpt','dpqpo','dbhbs','nfbs',
299   1              'qjt','njfseb','djqpuf','kpefs','gpmmbs',
300   1              'dbhbs','nbsjdb','qfep','dbdb','dvmp',
301   1              'ufub','nbsjdpo','dpkpoft','qjdib','nbnpo',
302   1              'dbcspo','iptujb','iptujbt');
303   1
304   1
305   1      /* This is the structure needed to obtain the digram table.          */
306   1      dcl d_ptr ptr static init(null());          /* location of digram table */
307   1      dcl l_ptr ptr static init(null());          /* location of unit letters */
308   1      dcl r_ptr ptr static init(null());          /* location of unit rules */
309   1
310   1
311   1      /* this array contains information about all possible pairs of units */
312   1      dcl 1 digrams(n_units, n_units) globalref,
313   1            2 begin bit(1),           /* on if this pair must begin syllable */
314   1            2 not_begin bit(1),       /* on if this pair must not begin */
315   1            2 end bit(1),             /* on if this pair must end syllable */
316   1            2 not_end bit(1),         /* on if this pair must not end */
317   1            2 break bit(1),           /* on if this pair is a break pair */
318   1            2 prefix bit(1),          /* on if vowel must precede this pair in same syllable */
319   1            2 suffix bit(1),          /* on if vowel must follow this pair in same syllable */
320   1            2 illegal_pair bit(1),    /* on if this pair may not appear */
321   1            2 align char(0);          /* dummy variable to force byte alignment */
322   1
323   1
324   1      /* this array contains left justified 1 or 2-letter pairs representing each unit */
325   1
326   1      dcl letters(0:n_units) globalref char(2);
327   1
328   1
329   1      /* this array has rules for each unit */
330   1
331   1      dcl 1 rules(n_units) based (r_ptr),
332   1            2 no_final_split bit(1),          /* can't be the only vowel in last syllable */
333   1            2 not_begin_syllable bit(1),      /* can't begin a syllable */
334   1            2 vowel bit(1),                   /* this is a vowel */
335   1            2 alternate_vowel bit(1);         /* this is an alternate vowel, (i.e., "y") */
336   1
337   1
338   1      dcl i fixed bin;
339   1      dcl j fixed bin;
340   1
341   1      /**** SLEEZE STRUCTURE */
342   1      dcl r(17) char(1) globalref;
```

```
343   1
344 ! 1       /* on the first call to generate_passwords, initialize pointers and     */
345 ! 1       /* generate the initial random number seed.                             */
346   1
347   1               if d_ptr = null()
348   1                 then
349   1       first_call:     do;
350   2                       d_ptr = addr(digrams);
351   2                       r_ptr = addr(r);
352   2                       l_ptr = addr(letters);
353   2
354   2                       do i=98 to 123;                 /* replace lower with lower + 1 */
355   3                           substr(translation,i,1) = substr(collate,i+1,1);
356   3                           end;
357   2
358 ! 2                   /* get the initial random seed:  Increment a counter in a tight loop.  Set a */
359 ! 2                   /* timer to interrupt (with an AST) every 10 milliseconds.  When this AST    */
360 ! 2                   /* occurs, read the last two bits of the counter and append them to the seed */
361 ! 2                   /* until we have 64 bits.                                                   */
362   2
363 ! 2                       /* set delta_time to 10 milliseconds binary time value */
364   2                       sts$value = sys$bintim(ten_msec, delta_time);
365   2                       if ^sts$success
366   2                           then do;
367   3                               put skip edit('fatal error:  invalid time string') (a);
368   3                               stop;
369   3                               end;
370   2
371   2                       next_field = 1;
372   2                       do while(next_field < 64);
373   3                           counter = 0;
374   3                           read_flag = '0'b;
375   3                           sts$value = sys$cantim (,);                                  /* cancel previous timer request */
376   3                           sts$value = sys$setimr (,delta_time, set_read_flag,);        /* set timer: AST every 10 msecs. */
377   3                           if ^sts$success
378   3                               then do;
379 ! 4                               /* fatal errors:  cannot continue */
380   4                                   put skip edit('fatal error:  ') (a);
381   4                                   if sts$value = ss$_accvio
382   4                                       then put edit('can''t read delta_time') (a);
383   4                                   else if sts$value = ss$_exquota
384   4                                       then put edit('too many ASTs or timer entries') (a);
385   4                                   else if sts$value = ss$_insfmem
386   4                                       then put edit('not enough dynamic memory') (a);
387   4                                   stop;
388   4                                   end;
389   3       tight_loop:             do while(^read_flag);                   /* This loop will execute until the AST goes off. */
390   4                                   counter =  counter + 1;             /* The AST handler sets read_flag, which stops    */
391   4                                   end;                                /* the loop and transfers control to read_ctr.    */
392   3       read_ctr:               seed_bits = substr(unspec(counter), 1, 2);  /* read the lower two bits of the counter */
393   3                               substr(seed, next_field, 2) = seed_bits;    /* insert them into the seed string */
394   3                               next_field = next_field + 2;
395   3                               end;
396   2
397 ! 2                   /* AST handling procedure */
398   2
399   2                       set_read_flag: proc;
```

```
                        dcl read_flag bit(1) aligned ext;       /* set when counter value is to be read */

                            read_flag = '1'b;

                            end set_read_flag;
                        end first_call;

      /* generate 'n_words' pronounceable words */

              do word_count = 1 to n_words;
                  word_is_pronounceable = '0'b;       /* reset the status code for each legal word */
                  call get_random_bits(seed);         /* get a random bit pattern */
                  s = substr(seed, 1, 31);            /* get the length of the word from */
                  unspec(n) = s;                      /* the random bit pattern in seed  */
                  word_length = mod(n, max_length-min_length+1) + min_length;
              /* generate random letter strings and test them until a legal word is found */
                  do while (^word_is_pronounceable);
                      call random_chars_ (word, word_length);
                      call test_word;
                      end;
                  aligning_blanks = copy(' ', max_length-word_length);        /* align the words for printing */
                  put skip edit(word||aligning_blanks, hyph_password(word_count)) (a, x(4), a);
                  end;


      /*  this internal procedure tests a word against the rules of pronounce-    */
      /*  ability contained in the digram table and the random_word_ algorithm.   */
      /*  the word is also hyphenated if it is pronounceable.                     */

      test_word: proc;
      dcl fixed_word char(word_length);       /* fixed length form of word */
      dcl bad_word char(word_length);         /* encrypted word for bad word check */
      dcl hyphens (20) bit(1) aligned;        /* position of hyphens in word */
      dcl i fixed bin;
      dcl bad_word_found bit(1) aligned;

              fixed_word = word;
              if pronounceable_ (fixed_word, hyphens, d_ptr, l_ptr, r_ptr, n_units)
                  then do;
                      bad_word = translate(fixed_word,translation);   /* do not use word if naughty */
                      i = 1;
                      bad_word_found = '0'b;
                      do while (^bad_word_found & i <= number_bad_words );
                          if index(bad_word,bad_word_string(i)) ^= 0
                              then bad_word_found = '1'b;
                          i = i + 1;
                          end;
                      if ^bad_word_found
                          then do;
                              word_is_pronounceable = '1'b;
                              password(word_count) = fixed_word;
                              /* add the hyphens to the word */
                              hyph_password(word_count) = '';
                              do i = 1 to word_length;
                                  hyph_password(word_count) = hyph_password(word_count) || substr(word, i, 1);
                                  if hyphens(i)
                                      then hyph_password(word_count) = hyph_password(word_count) || '-';
```

```
457     5                                   end;
458     4                               end;
459     3                           end;
460     2
461     2           end test_word;
462     1
463     1
464     1       end generate_passwords;
465
466
```

```
467  !          /* ROUTINE  random_chars_.pli
468  !          !
469  !          ! FUNCTIONAL DESCRIPTION:
470  !          !
471  !          ! Form a string of random letters, given the length of the string.  The
472  !          ! letters will be the 26 letters of the english alphabet, and each letter
473  !          ! will have an equal probability of occuring in any given position in the
474  !          ! string.  These letters will be concatenated to form a string of the correct
475  !          ! length.  The input will be the length of the string, and the output will be
476  !          ! the word in character form. The table of letters is held in an internal
477  !          ! static array of 26 single-character strings.  A random pattern of 64 bits
478  !          ! will be obtained by a call to get_random_bits, and this seed will be taken
479  !          ! 1 byte at a time to generate a random number in the correct range.
480  !          !
481  !          ! INPUT PARAMETERS:
482  !          !       length -        length of string to be generated.
483  !          !
484  !          ! OUTPUT PARAMETERS:
485  !          !       chars -         string of random letters to be generated.
486  !          !
487  !          ! ROUTINE VALUE:
488  !          !       NONE
489  !          !
490  !          ! SIDE EFFECTS:
491  !          !       NONE
492  !          !
493  !          */
494
495              random_chars_: proc (chars, length);
496      1
497      1       dcl chars char(*) var;          /* PARAMETER: string of random letters to be generated */
498      1       dcl length fixed bin;           /* PARAMETER: length of string to be generated */
499      1       dcl char_count fixed bin;       /* number of characters generated so far */
500      1
501      1
502      1       dcl letters(0:25) char(1) static        /* table of letters */
503      1               init('a','b','c','d','e','f','g','h','i','j','k','l','m',
504      1                    'n','o','p','q','r','s','t','u','v','w','x','y','z');
505      1       dcl letter_index fixed bin;     /* a random number index into the letter table */
506      1       dcl char char(1);               /* a letter from the letter table */
507      1
508      1
509      1       dcl get_random_bits entry(bit(64) aligned);     /* gets a random 64-bit pattern */
510      1       dcl seed bit(64) aligned ext;   /* random 64-bit pattern obtained from get_random_bits, */
511  !   1                                       /* used to generate 8 random integers.                  */
512      1       dcl seed_byte(0:7) bit(8)       /* same as seed, but allows access to individual bytes, */
513      1               based(addr(seed));      /* which allows easier generation of random numbers     */
514      1       dcl byte_count fixed bin        /* index of next unused byte in random bit pattern */
515      1               static init(8);
516      1       dcl s bit(8) aligned;           /* temporary bit form of random number */
517      1       dcl n fixed bin(8);             /* temporary integer form of random number */
518      1       dcl f float bin(8);             /* temporary floating point form of random number */
519      1
520  !   1       /* generate the string */
521      1
522      1               chars = '';
```

```
523     1               do char_count = 1 to length;
524     2
525   ! 2               /* see if all bytes of the seed string have been used */
526     2                   if byte_count = 8
527     3                       then do;
528   ! 3                       /* all bytes have been used, get a new seed string */
529     3                           call get_random_bits(seed);
530     3                           byte_count = 0;
531     3                           end;
532     2
533   ! 2               /* We now have a random pattern of 64 bits.  Use one byte at a time to  */
534   ! 2               /* generate a random number between 0 and 25 and use this as an index   */
535   ! 2               /* to get a letter from the table, letters(0:25).                       */
536     2
537     2                   unspec(n) = seed_byte(byte_count);  /* make the byte into an integer n, 0 <= n < 256 */
538     2                   f = float(n,8)/float(256,8);        /* map n into a floating point number f, 0 <= f < 1 */
539     2                   letter_index = fixed(trunc(f*26),8);/* map f into an integer of the correct range: 0-25 */
540     2                   char = letters(letter_index);       /* get a letter from the table */
541     2                   chars = chars || char;              /* add letter to end of string */
542     2                   byte_count = byte_count + 1;        /* 1 byte of seed string has been used */
543     2                   end;
544     1
545     1
546     1       end random_chars_;
547
548
```

```
549 ;           /* ROUTINE get_random_bits.pli
550 ;            *
551 ;            * FUNCTIONAL DESCRIPTION:
552 ;            *
553 ;            * Generate a random pattern of 64 bits using the MTH$RANDOM algorithm.
554 ;            *
555 ;            * INPUT PARAMETERS:
556 ;            *       NONE
557 ;            *
558 ;            * OUTPUT PARAMETERS:
559 ;            *       s -     8 byte random seed.
560 ;            *
561 ;            * ROUTINE VALUE:
562 ;            *       NONE
563 ;            *
564 ;            * SIDE EFFECTS:
565 ;            *       NONE
566 ;            *
567 ;            */
568
569             get_random_bits: proc (s);
570     1
571     1       dcl s bit(64) aligned;  /* PARAMETER: 8-byte random seed                                    */
572 :   1                               /*            Also used as random bit pattern output from MTH$RANDOM */
573     1       dcl s_long(0:1) fixed bin(31) based(addr(s));
574     1       dcl s_byte(0:7) fixed bin(7)  based(addr(s));
575     1       dcl t fixed bin(31);
576     1       dcl t_byte(0:3) fixed bin(7) based(addr(t));
577     1
578     1       dcl  mth$random entry (fixed binary (31));
579     1
580 ;   1       /* get a random bit pattern by using the random number seed, s, as a key.       */
581 ;   1       /* Two calls to the random number generator are necessary because it will only  */
582 ;   1       /* return 32 bits at a time, and we want a 64 bit sequence.                     */
583     1
584     1       call mth$random (s_long(0));
585     1       call mth$random (s_long(1));
586     1       call mth$random (t);
587     1
588     1       s_byte(0) = t_byte(2);
589     1       s_byte(4) = t_byte(3);
590     1
591     1       end get_random_bits;
```

COMMAND LINE

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

N 10

PRONOUNCE
LIS

QUEMAN
LIS

MATCHKEY
LIS

PUTCLIMSG
LIS

QUEMANMSG
LIS

PASSWORDS
LIS

QUEMANSHO
LIS