```
STMT  LEVEL REPLC          SOURCE STATEMENT                                    MAC01  8SEP69

  2     0              %;  John Ahern, SLAC Computation Group, September 1969
  3     0              %;
  4     0              %;      MACROS is a macro processor designed for use with language
  5     0              %;  processors that lack a macro facility.  Its usage can signifi-
  6     0              %;  cantly reduce the amount of clerical work normally required to
  7     0              %;  use such language processors as FORTRAN or SUMX (considering SUMX
  8     0              %;  as a processor of its control language).
  9     0              %;      For one not familiar with macro languages, they can be
 10     0              %;  characterized as providing facilities for describing the generation
 11     0              %;  of a program, in a manner analogous to the way that a program-
 12     0              %;  ming language such as FORTRAN provides facilities for describing
 13     0              %;  a computation.  The "data" of the macro processor consists of
 14     0              %;  statements or fragments of statements of the language used in
 15     0              %;  the generated program.  The language of the generated program
 16     0              %;  is commonly called the "base language."  The macro language
 17     0              %;  uses constructs which are recognized by the "macro processor"
 18     0              %;  (MACROS) to describe operations on the base language--order of
 19     0              %;  generation and modifications of the base language statements.
 20     0              %;  The output of the macro processor is a program in the base
 21     0              %;  language, ready for its processor.
 22     0              %;      MACROS can be used with any base language where statements
 23     0              %;  can be treated as being within specified columns of each record
 24     0              %;  (card or line), although special facilities are provided for
 25     0              %;  to handle FORTRAN continuation conventions.  If the need for
 26     0              %;  recognition of statements split across records can be avoided,
 27     0              %;  (i.e.,the base language continuation rules need not be accepted)
 28     0              %;  then MACROS can be used with other base languages.  It provides
 29     0              %;  definition and expansion of macro-instructions (called "macros")
 30     0              %;  and retrieval of macro definitions from an external library--
 31     0              %;  these facilities control the order of generation of base
 32     0              %;  language statements; and replacement of symbolic names in
 33     0              %;  base language statements in its input--whereby modifications
 34     0              %;  can be obtained.
 35     0              %;
 36     0              %;      This report is organized as a series of examples that
 37     0              %;  gradually introduce features of the MACROS language, and
 38     0              %;  reference sections that summarize the language and information
 39     0              %;  needed to use MACROS (JCL,etc.).  Its contents are:
 40     0              %;
 41     0              %;  I.   Introduction--this section
 42     0              %;  II.  Examples of MACROS usage--
 43     0              %;         1.   Use of macro to include text--introduces macros,
 44     0              %;              usage of macro library for maintaining FORTRAN
 45     0              %;              declaration statements.
 46     0              %;         2.   Use of replacement to generalize program--introduces
 47     0              %;              replacement as a method of setting parameters that are
 48     0              %;              fixed at execution time.
 49     0              %;         3.   Use of replacement to define function type--explains
 50     0              %;              additional replacement facilities in context of
 51     0              %;              writing a FORTRAN function for single and double
 52     0              %;              precision.
 53     0              %;         4.   Macro parameters,continuation statements--introduces
 54     0              %;              macro parameters and FORTRAN continuation statement
 55     0              %;              processing to define a macro that generates debugging
```

```
%;          statements in a FORTRAN program.
%;     5.   Processor variable scope and macro parameters--more
%;          illustrations of the properties of variables used by
%;          MACROS.
%;     6.   Use of MACROS with the SUMX analysis program--illus-
%;          trates usage of MACROS to prepare input for SUMX.
%;  III. MACROS language description--summary of the MACROS language
%;       for reference purposes.
%;  IV.  Using MACROS--options and JCL requirements--information
%;       about options available to the MACROS user, and examples of
%;       JCL for MACROS with FORTRAN and with SUMX.
%;  V.   MACROS output from the examples of section II.
%;
%;  The interested reader should not be discouraged by the length of
%;  this manual--it is not necessary to digest the entire manual to
%;  use MACROS.  For example, a SUMX user can refer to example 6 for
%;  most of the information he needs, going to examples 1 and 4 for
%;  additional details on macros if needed.  The JCL he needs can be
%;  found at the end of section IV.
%;
%;       The report itself is prepared as input to MACROS to
%;  simplify writing the examples.  Sections I to IV are reproductions
%;  of MACROS printed listing, and section V is the output from the
%;  run that produced sections I to IV--it gives the output from the
%;  examples as it would appear to the base language processor.
%;  (To be exact, sections I to II-5, and II-6 to IV are separate
%;  MACROS runs, with and without the special FORTRAN option,respec-
%;  tively.)
%;
%;       The author acknowledges the comments and suggestions of the
%;  MACROS users in Experimental Group D including Dr. Joseph Park
%;  who provided the body of example 6, and John Ehrman of the
%;  Computation Group.  Any suggestions for improving the report
%;  or the macro language will be appreciated.
%;
%;
%;  A note about the designation "statement oriented macro processor":
%;       This term can be used to describe the class of macro
%;  processors that consider their input to be a series of discrete
%;  base language and processor statements; macros are used to
%;  generate another series of statements that replace the processor
%;  statement that called the macro.  This is opposed to what could
%;  be called the "functional macro processor", which considers
%;  its input to be a string of characters, possibly including
%;  statement delimiters; macro calls, written as function references,
%;  produce "side effects" like defining other macros, and return
%;  a string that replaces the macro call.  Thus the macros are used
%;  like functions.
%;       MACROS is a processor of the former type, as are most
%;  assembly language macro processors.  The macro facility in PL/I
%;  is of the latter type, although it has characteristics of the
%;  former also.  MACRO-FORTRAN, written by G. A. Robinson of the
%;  Computation Group to provide macro facilities for FORTRAN users at
%;  Argonne laboratory (somewhat like MACROS), is an example of the
%;  functional macro processor.  (Argonne National Laboratory
```

```
STMT  LEVEL REPLC          SOURCE STATEMENT                                  MAC01  8SEP69
111     0             %;   Report ANL-7309, June 1967.)
```

```
STMT  LEVEL REPLC          SOURCE STATEMENT                                MAC01  8SEP69

113     0              %;        A problem that constantly recurs in the preparation of a
114     0              %;   large FORTRAN program is providing updated copies of declaration
115     0              %;   statements in each subroutine.  What is needed is the ability
116     0              %;   to maintain one copy of each set of declarations and include it
117     0              %;   into the subroutine being compiled immediately prior to com-
118     0              %;   pilation.
119     0              %;        MACROS provides this facility by filing away a sequence of
120     0              %;   text under a name and allowing insertion of this text at
121     0              %;   specified points in the source program.  A sequence of text used
122     0              %;   in this manner is a "macro definition," or more often, a "macro."
123     0              %;   The process of insertion is a "macro expansion."  To define a
124     0              %;   macro and give it a name, special statements which are recog-
125     0              %;   nized by MACROS are used to bracket the model statements (i.e.
126     0              %;   text) of the macro definition.  Another MACROS statement is used
127     0              %;   to invoke the macro expansion at the point it is written.
128     0              %;        In the following example, a main program and a subroutine
129     0              %;   are prepared for compilation using MACROS.  They use the same
130     0              %;   COMMON statement, which will be supplied with a macro.
131     0              %;
132     0      0                          ***EXAMPLE 1-1***
133                    %    MACRO COM1       ;START MACRO DEFINITION
134     E              %;   NEXT IS A MODEL STATEMENT OF THE BASE LANGUAGE (FORTRAN)
135     E                    COMMON   A,B
136     E              %    MEND             ;END MACRO DEFINITION
137     0      0       C
138     0      0       C     MAIN PROGRAM FOLLOWS
139     0              %    CALL COM1       ;THIS INCLUDES THE COMMON STATEMENT
140     1      0             COMMON   A,B
142     0      0       C ... REMAINDER OF MAIN PROGRAM
143     0      0             END
144     0      0             SUBROUTINE SUB
145     0              %    CALL COM1       ;THIS INCLUDES COMMON IN SUBROUTINE
146     1      0             COMMON   A,B
148     0      0       C ... REMAINDER OF SUBROUTINE SUB
149     0      0             RETURN
150     0      0             END
151     0      0                          *****************
152     0              %;
153     0              %;   The first three statements define the macro COM1 whose function
154     0              %;   is to insert the COMMON statement into the source program.
155     0              %;   The expansion of COM1 can be seen following the CALL statements.
156     0              %;
157     0              %;        The final output of MACROS can be determined by use of the
158     0              %;   information appearing on the left of the page.  The first
159     0              %;   column is a statement number assigned by MACROS to all
160     0              %;   statements (both processor and base language) presented to
161     0              %;   it.  Its primary use is for reference purposes in error messages.
162     0              %;   The second column is the macro nest level.  The number in this
163     0              %;   column indicates from what depth MACROS is processing statements.
164     0              %;   Zero is top level, that is not from a macro expansion.  In the
165     0              %;   case above, the expansions of the macro COM1 are at level 1
166     0              %;   since COM1 was invoked from the top level.  If a macro is
167     0              %;   being edited, as in the first four statements of the example,
168     0              %;   then the second column has an 'E' in it, indicating that
```

```
STMT  LEVEL REPLC          SOURCE STATEMENT                                MAC01  8SEP69

169     0              %;  an edited definition is being accepted.  If there is an error
170     0              %;  in the MACRO statement, then the macro definition is skipped;
171     0              %;  this is indicated by an 'S' in the second column.  The
172     0              %;  third column is a replacement count that applies to base
173     0              %;  language statements that become part of the final output.
174     0              %;  This term is explained in subsequent examples.
175     0              %;      In summary,the final output of MACROS consists of those base
176     0              %;  language statements that have a numeric entry in the third column.
177     0              %;  The text that you are reading now is processor comments,
178     0              %;  indicated by a '%' in column 1 followed by a ';'--it is
179     0              %;  completely ignored by MACROS.
180     0              %;
181     0              %;      Returning to the example, we can now see that MACROS
182     0              %;  provided us with a means where only one copy of the COMMON
183     0              %;  statement needs to be maintained.  If it needs to be changed
184     0              %;  only the macro definition need be changed, rather than all
185     0              %;  COMMON declarations (usually more than two) in the source deck.
186     0              %;
187     0              %;      We can further improve the situation by using
188     0              %;  another facility of MACROS.  A normal occurence with large
189     0              %;  programs is the need to recompile only selected subroutines
190     0              %;  where changes are being made.  If declaration statements are
191     0              %;  provided using macro definitions, they still need to be
192     0              %;  included in each MACROS run--which can easily lead to duplicate
193     0              %;  macro definitions in each update deck, particularly if more than
194     0              %;  one person is making modifications.  The solution to this
195     0              %;  problem, which is available in the assembler or PL/I languages
196     0              %;  is to provide a means for retrieving text or macro definitions
197     0              %;  from an external library.
198     0              %;      This facility is provided in MACROS in a limited fashion.
199     0              %;  To use it, the macro definitions are put in a sequential dataset
200     0              %;  (SDS) and made available to MACROS under the ddname SYSLIB (this
201     0              %;  may be the concatenation of SDSs or partitioned dataset (PDS)
202     0              %;  members, however).
203     0              %;      When MACROS encounters a macro call for a macro that is
204     0              %;  not defined in the MACROS input, it will open SYSLIB and edit
205     0              %;  all macro definitions that appear there.  Presumably, the macro
206     0              %;  needed was there, so the CALL statement can be processed.
207     0              %;
208     0              %;      For our example, we could put the definition for COM1
209     0              %;  (and other needed macros) in a SDS, and have it retrieved
210     0              %;  from there when the CALL statement was encountered.
211     0              %;      The following input to MACROS could be used to compile
212     0              %;  subroutine SUB2 which uses a COMMON statement defined by a
213     0              %;  macro in SYSLIB called COM.
214     0              %;
215     0      0                        ***EXAMPLE 1-2***
216     0      0            SUBROUTINE SUB2
217    *               %   MACRO  COM
218    *E                    COMMON  ZORCH
219    *E              %   MEND
220     0              %    CALL COM   ;THIS CALL WILL CAUSE SYSLIB TO BE OPENED
221     1      0             COMMON  ZORCH
223     0      0       C ... REMAINDER OF SUB2
224     0      0             RETURN
```

```
STMT  LEVEL REPLC        SOURCE STATEMENT                                MAC01  8SEP69

225     0     0           END
226     0     0                        *****************
227     0             %;
228     0             %;       Optionally MACROS will print the editing of macro
229     0             %;  definitions from SYSLIB as is done here.  Editing from SYSLIB
230     0             %;  is indicated by a '*E' in the second column on the left.
231     0             %;  After all macro definitions on SYSLIB are read,the CALL
232     0             %;  statement can be processed.  (in this case COM was the only
233     0             %;  macro in SYSLIB.)
234     0             %;
235     0             %;       The only specific requirement about the order of appearance
236     0             %;  of macro definitions is that they appear before they are used.
237     0             %;  This can be satisfied by direct inclusion in the MACROS input
238     0             %;  anywhere before the first CALL statement referencing them, or
239     0             %;  by placing the definitions in a separate library prior to a
240     0             %;  MACROS run using that library.
```

```
STMT  LEVEL REPLC        SOURCE STATEMENT                                MAC01  8SEP69

242     0            %;      This and subsequent examples will utilize the second
243     0            %;  facility of macro processors--modification of base language
244     0            %;  statements.  MACROS provides a "replacement" or "substitution"
245     0            %;  facility where occurences of symbolic names (processor vari-
246     0            %;  ables) in base language statements can be replaced by the
247     0            %;  (character string) value  associated with the processor variable.
248     0            %;
249     0            %;      The following demonstrates a simple usage of replacement
250     0            %;  to set array bounds and describe computations depending on
251     0            %;  the array bounds.  This enables "tailoring" the program to a
252     0            %;  specific  application immediately prior to compilation by
253     0            %;  use of processor statements to set the values of processor
254     0            %;  values.
255     0            %;
256     0      0                      ***EXAMPLE 2-1***
257     0            %    ADIM=10    ;SET BOUND OF ARRAY A
258     0     BR            DIMENSION A(&ADIM)
258     0      1            DIMENSION A(10)
259     0      0       C  READ NUMBER OF DATA ITEMS
260     0      0            READ (5,1)N
261     0      0       C  CHECK IF ARRAY A LARGE ENOUGH
262     0     BR            IF(N.LE.&ADIM) GO TO 3
262     0      1            IF(N.LE.10) GO TO 3
263     0      0       C  ... DO SOMETHING ABOUT ERROR CONDITION
264     0     BR       C  ASSUME WE USE ONLY &ADIM ITEMS THEN
264     0      1       C  ASSUME WE USE ONLY 10 ITEMS THEN
265     0     BR            N=&ADIM
265     0      1            N=10
266     0      0       3    CONTINUE
267     0      0       C  ... PROCESS DATA
268     0      0       C
269     0      0            END
270     0      0                      *****************
271     0            %;
272     0            %;  The first statement is a processor assignment statement which
273     0            %;  sets the value of processor variable ADIM to 10.  The state-
274     0            %;  ment also declares ADIM as a processor variable since this
275     0            %;  is the first use of ADIM.  The effect of the statement is to
276     0            %;  make occurences of the name ADIM preceded by the "escape
277     0            %;  character" '&' in base language statements eligible for
278     0            %;  replacement.  The combination of the escape character and the
279     0            %;  processor variable identifier is called the "escape name" in
280     0            %;  this discussion.  "Replacement" replaces the escape name in the
281     0            %;  text by the character string value of the processor variable.
282     0            %;  This occurs in the four FORTRAN statements that use the escape
283     0            %;  name '&ADIM' to obtain the bound of array A.
284     0            %;      Now we can see the use of the third column on the left
285     0            %;  of the page (the REPLacement Count).  The first printing
286     0            %;  of the statements using ADIM shows the statement before
287     0            %;  replacement (hence the 'BR' in the REPLC column).  The
288     0            %;  second printing shows the statement as it will actually
289     0            %;  appear in the output of MACROS--the occurence(s) of
290     0            %;  ADIM and the escape character have been replaced by the value
291     0            %;  of ADIM (the string representation of 10).  The number in
```

```
STMT  LEVEL REPLC          SOURCE STATEMENT                                MAC01  8SEP69

292     0            %;   the replacement column (1) indicates that one replacement
293     0            %;   has occured.  (Note--the printing before replacement is
294     0            %;   optional and is normally suppressed.)
295     0            %;        What we have gained by replacement is the ability to
296     0            %;   set parameters in the program which would either be impos-
297     0            %;   ible to set at execution time or would cost execution time
298     0            %;   efficiency.  The next example indicates a use of replacement
299     0            %;   to solve an impossible case for normal FORTRAN.
```

```
STMT  LEVEL REPLC          SOURCE STATEMENT                                  MAC01  8SEP69

301     0             %;        Replacement will be used to define a function in either
302     0             %;  single or double precision.  The body of the function
303     0             %;  will be generated with a macro.  This is done here for
304     0             %;  convenience since the document you are reading is input
305     0             %;  to MACROS (this avoids rewriting the function).
306     0             %;        First the function will be provided as a macro definition:
307     0             %;
308                   %   MACRO MDISCRM
309     E                  REAL FUNCTION DISCRM*&RLEN(A,B,C)
310     E                  REAL*&RLEN  B,A,C
311     E                  DISCRM=&FPRE.SQRT(B*B-4*A*C)
312     E                  RETURN
313     E                  END
314     E             %   MEND
315     0             %;
316     0             %;  RLEN is the real datum length (4 or 8).  FPRE is the prefix
317     0             %;  required for double precision ('D').  To get the double
318     0             %;  precision version,use the following:
319     0             %;
320     0      0                         ***EXAMPLE 3-1***
321     0             %   RLEN=8
322     0             %   FPRE='D'
323     0             %   CALL MDISCRM
324     1     BR           REAL FUNCTION DISCRM*&RLEN(A,B,C)
324     1      1           REAL FUNCTION DISCRM*8(A,B,C)
325     1     BR           REAL*&RLEN  B,A,C
325     1      1           REAL*8  B,A,C
326     1     BR           DISCRM=&FPRE.SQRT(B*B-4*A*C)
326     1      1           DISCRM=DSQRT(B*B-4*A*C)
327     1      0           RETURN
328     1      0           END
330     0      0                         *****************
331     0             %;
332     0             %;  The two assignment statements before the CALL statement set the
333     0             %;  values of RLEN and FPRE.  The replacement for RLEN is
334     0             %;  similiar to that used in the previous example.  The use
335     0             %;  of FPRE illustrates the facility used to recognize the end
336     0             %;  of an occurence of a processor variable when it is to be
337     0             %;  followed by a letter or number.  The period following FPRE is
338     0             %;  also removed from the text when FPRE's string value ('D') is
339     0             %;  substituted.  The assignment statement for FPRE shows that
340     0             %;  the type of processor variable is determined dynamically,
341     0             %;  and may change in different uses.  In theory, RLEN has
342     0             %;  integer type, while FPRE has string type;  however in this
343     0             %;  version of MACROS all the internal values are held as strings.
344     0             %;        Also, the writing of string constants is shown--text
345     0             %;  between quotes.  A quote in the string is represented by
346     0             %;  2 quotes, e.g. 'FINNEGAN''S WAKE'.  There is also the
347     0             %;  string constant of zero length, or the "null string," repre-
348     0             %;  sented by ''.  This is used when an occurence of a processor
349     0             %;  variable is to be completely removed from the text,as will
350     0             %;  be done for the single precision version of MDISCRM.
351     0             %;        For a single precision DISCRM, use the following:
352     0             %;
```

```
STMT  LEVEL REPLC          SOURCE STATEMENT                                    MAC01  8SEP69

353     0     0                          ***EXAMPLE 3-2***
354     0             %    RLEN=4
355     0             %    FPRE=''
356     0             %    CALL MDISCRM
357     1    BR             REAL FUNCTION DISCRM*&RLEN(A,B,C)
357     1     1             REAL FUNCTION DISCRM*4(A,B,C)
358     1    BR             REAL*&RLEN  B,A,C
358     1     1             REAL*4  B,A,C
359     1    BR             DISCRM=&FPRE.SQRT(B*B-4*A*C)
359     1     1             DISCRM=SQRT(B*B-4*A*C)
360     1     0             RETURN
361     1     0             END
363     0     0                          ****************
```

```
STMT  LEVEL REPLC        SOURCE STATEMENT                                    MAC01  8SEP69

365     0            %;  A macro will be used to generate debugging statements.
366     0            %;  Using a macro will reduce the amount of writing and make it
367     0            %;  possible to completely remove the statements from the source
368     0            %;  program by simply defining the macro with no model statements.
369     0            %;       Our debug macro will generate a NAMELIST statement and a
370     0            %;  WRITE statement:
371     0            %;
372                  %   MACRO  DEBUG(NLN,DBLIST)
373     E                 NAMELIST /&NLN/       &DBLIST
374     E                 WRITE (6,&NLN)
375     E            %   MEND
376     0            %;
377     0            %;  The MACRO statement declares two "formal parameters" which
378     0            %;  represent the NAMELIST name (NLN) and a list of identifiers
379     0            %;  (DBLIST).  The formal parameters are assigned values when the
380     0            %;  macro is expanded by using an "actual parameter list" with
381     0            %;  the CALL statement.  (Terminology borrowed from ALGOL 60.)  To
382     0            %;  use DEBUG, one could write:
383     0            %;
384     0      0                          ***EXAMPLE 4-1***
385     0            %   CALL DEBUG('DB','A,B,C')
386     1     BR            NAMELIST /&NLN/       &DBLIST
386     1      2            NAMELIST /DB/       A,B,C
387     1     BR            WRITE (6,&NLN)
387     1      1            WRITE (6,DB)
389     0      0                          *****************
390     0            %;
391     0            %;  Here, the actual parameters are the two strings 'DB' (NAMELIST
392     0            %;  name), and 'A,B,C' (the identifier list).
393     0            %;
394     0            %;  IF this macro is rewritten to contain only a MACRO and a MEND
395     0            %;  statement, the effect will be to remove the debugging statements
396     0            %;  from the program.  We cannot do that here, since that would be a
397     0            %;  duplicate macro definition, which MACROS does not allow.
398     0            %;
399     0            %;       Another feature of MACROS that is useful when working
400     0            %;  with FORTRAN as the base language is the automatic continuation
401     0            %;  statement processing.  This is available if the 'FORT=1' option
402     0            %;  is used.  (Options are detailed in section IV.)
403     0            %;       MACROS will automatically generate continuation statements
404     0            %;  if the generated statement exceeds one card,as occurs here:
405     0            %;
406     0      0                          ***EXAMPLE 4-2***
407     0            % CALL DEBUG('DB','IDEN1,IDEN2,IDEN3,IDEN4,IDEN5,IDEN6,IDEN7,FOOBAZ')
408     1     BR            NAMELIST /&NLN/       &DBLIST
408     1      2            NAMELIST /DB/       IDEN1,IDEN2,IDEN3,IDEN4,IDEN5,IDEN6,IDEN7,FOOBA
                            1Z
409     1     BR            WRITE (6,&NLN)
409     1      1            WRITE (6,DB)
411     0      0                          *****************
412     0            %;
413     0            %;  If the FORT=1 option is used, the total number of characters
414     0            %;  in any FORTRAN statement in the MACROS input (before and after
415     0            %;  replacement,if any) is limited to 270 (minus trailing blanks and
```

```
STMT  LEVEL REPLC          SOURCE STATEMENT                                    MAC01  8SEP69

416     0              %;  columns 1-6 of the continuation statements).  Hollerith literals
417     0              %;  in quotes cause all blanks to be counted, so the lower limit on
418     0              %;  continuation cards is 3.  Since it is possible to encounter
419     0              %;  longer FORTRAN declaration statements, they must be rewritten
420     0              %;  into smaller statements so that the continuation limits are not
421     0              %;  exceeded.  This presents no problems, since COMMON statements
422     0              %;  are cumulative--two declarations for the same COMMON block are
423     0              %;  treated by the compiler as if the second were an extension of
424     0              %;  the first.
425     0              %;       To illustrate MACROS handling of continuation statements,
426     0              %;  several arbitrary FORTRAN statements will be shown before and
427     0              %;  after MACROS editing.  Printing before editing, if selected by
428     0              %;  the user, is done whenever an escape character appears in the
429     0              %;  statement.  The pre-editing printing will be forced by putting
430     0              %;  an escape name in the statements(&DUM).  MACROS will leave
431     0              %;  occurences of an escape name using an inactive processor
432     0              %;  variable unchanged in the output text:
433     0              %;
434     0     0                          ***EXAMPLE 4-3***
435     0    BR        &DUM
                            1F(I)
                            2=
                            3F(I-2)
                            4  +
                            5F(I-1)
435     0     0        &DUM  F(I)=F(I-2)   +F(I-1)
436     0     0
437     0    BR        &DUM  FORMAT(   'THIS IS A HOLLERITH LITERAL IN QUOTES'        ,
                            1   'NOTICE THAT ALL BLANKS ARE RETAINED'   )
437     0     0        &DUM  FORMAT(   'THIS IS A HOLLERITH LITERAL IN QUOTES'        ,
                            1   'NOTICE THAT ALL BLANKS ARE RETAINED'   )
438     0     0                          *****************
439     0              %;
440     0              %;  In the first example, it can be seen that all trailing blanks
441     0              %;  were removed,but leading blanks (preceding the +) were
442     0              %;  retained.  In the second example, the Hollerith literals
443     0              %;  caused all blanks to be retained--this is done to avoid trun-
444     0              %;  cation of literals extending across card boundaries.
```

```
STMT  LEVEL REPLC          SOURCE STATEMENT                                MAC01  8SEP69

446     0              %;      The previous examples used processor variables that were
447     0              %;  simple variables (i.e. not formal parameters) global to a macro,
448     0              %;  or formal parameters.  In the following, more about the scope
449     0              %;  of processor variables will be illustrated using the macro
450     0              %;  MDISCRM of example 3.  The term "scope" refers to the way that
451     0              %;  MACROS determines what declaration of an identifier is to be
452     0              %;  associated with a usage of that identifier--possibilities are:
453     0              %;  simple variable declared by assignment, formal parameter local
454     0              %;  to the macro being expanded, or formal parameter local to some
455     0              %;  macro that called the one currently being expanded.
456     0              %;      The first two cases have been illustrated; the last case
457     0              %;  can be shown with the following macro:
458     0              %;
459                    %   MACRO DISCP(RLEN,FPRE)
460     E              %   CALL MDISCRM
461     E              %   MEND
462     0              %;
463     0              %;  We can use it to set RLEN and FPRE and call MDISCRM for a
464     0              %;  double precision DISCRM:
465     0              %;
466     0      0                        ***EXAMPLE 5-1***
467     0              %   CALL DISCP(8,'D')
469     2     BR            REAL FUNCTION DISCRM*&RLEN(A,B,C)
469     2      1            REAL FUNCTION DISCRM*8(A,B,C)
470     2     BR            REAL*&RLEN  B,A,C
470     2      1            REAL*8  B,A,C
471     2     BR            DISCRM=&FPRE.SQRT(B*B-4*A*C)
471     2      1            DISCRM=DSQRT(B*B-4*A*C)
472     2      0            RETURN
473     2      0            END
476     0      0                        *****************
477     0              %;
478     0              %;  When the level 2 macro MDISCRM was expanded, the current values
479     0              %;  associated with the closest occurence of the names RLEN and
480     0              %;  FPRE was used.  These were the values supplied to the formal
481     0              %;  parameters in DISCP by the CALL statement.  The macro expansion
482     0              %;  consists of the following steps:
483     0              %;
484     0              %;  (1)  Save any values associated with the formal parameter
485     0              %;       identifiers (RLEN and FPRE).
486     0              %;  (2)  Associate (assign) the values of the corresponding actual
487     0              %;       parameters with the formal parameter identifiers.
488     0              %;  (3)  Expand the macro body in the usual fashion (process the
489     0              %;       CALL statement).
490     0              %;  (4)  When returning from this macro (DISCP), free the values
491     0              %;       associated with the formal parameters on step 2 and
492     0              %;       restore those previously associated with the identifiers,
493     0              %;       if any.
494     0              %;
495     0              %;      We can see this effect by checking that the values assoc-
496     0              %;  iated with the top level occurences of RLEN and FPRE (4 and '',
497     0              %;  respectively) have been restored by substituting them in this
498     0              %;  arbitrary statement:
499     0              %;
```

```
STMT  LEVEL REPLC          SOURCE STATEMENT                                    MAC01  8SEP69

500     0     0                          ***EXAMPLE 5-2***
501     0    BR       THE TOP LEVEL VALUES OF RLEN=&RLEN, AND FPRE=&FPRE..
501     0     2       THE TOP LEVEL VALUES OF RLEN=4, AND FPRE=.
502     0     0                          *****************
503     0              %;
504     0              %;   As expected, MACROS restored the top level values as claimed.
505     0              %;   Also, the two periods following &FPRE show how one obtains a period
506     0              %;   following an escape name.  The first period is included in the
507     0              %;   escape name, while the second remains in the text.
508     0              %;
509     0              %;        This scope mechanism differs from that of ALGOL or PL/I
510     0              %;   in that it is dynamically determined by the sequence of macro
511     0              %;   calls rather than from explicit nesting when the program is
512     0              %;   written (lexic scope).  (It is similiar to that used in
513     0              %;   Iverson language.)
```

```
STMT  LEVEL REPLC          SOURCE STATEMENT                                    MAC01  8SEP69

  2     0                %;      This example will illustrate the usage of MACROS with
  3     0                %;  a base language other than FORTRAN.  MACROS will be used
  4     0                %;  to simplify preparation of a set of control statements to the
  5     0                %;  SUMX data analysis program.  The features of MACROS illustrated
  6     0                %;  separately in the previous examples will be combined here.  In
  7     0                %;  addition, another mode of replacement will be introduced that is
  8     0                %;  useful for generating statements that have fixed format require-
  9     0                %;  ments, as do SUMX control statements.
 10     0                %;      For those not familiar with SUMX, it can be briefly
 11     0                %;  described as a general processor providing facilities for
 12     0                %;  a user to perform statistical analysis of large quantities of
 13     0                %;  data, such as that encountered in high energy physics.
 14     0                %;  By means of an elaborate set of control statements, the user
 15     0                %;  defines "TESTs" which can be referenced in the data collection and
 16     0                %;  display sections of SUMX to determine whether or not a data sample
 17     0                %;  is to be included in the display (histogram, scatter plot,etc.).
 18     0                %;  The tests are Boolean expressions that describe some relation
 19     0                %;  on an input record or on additional quantities computed by
 20     0                %;  the user from data in the input record.  If the test(s) are
 21     0                %;  satisfied (or not satisfied, as desired), then SUMX performs
 22     0                %;  some action such as including data from the input record in
 23     0                %;  the display.
 24     0                %;      In the example at hand, control statements to direct
 25     0                %;  SUMX to generate a set of histograms that display character-
 26     0                %;  istics of a class of inter-particle reaction(three-prong
 27     0                %;  event--photon+proton...pi+pi-proton).  Control statements
 28     0                %;  will be needed to describe format of the data records,
 29     0                %;  criteria for including a data record in a display(the input
 30     0                %;  contains events other than the type of interest),actions
 31     0                %;  performed by a user-supplied routine that computes additional
 32     0                %;  quantities from the input ("CHARM" routines), and parameters
 33     0                %;  used for making the histograms (number or bins,bin width,etc).
 34     0                %;
 35     0                %;      The amount or writing, thus margin for error, is reduced
 36     0                %;  by using macros to generate sequences of control statements.
 37     0                %;  By maintaining a macro library of commonly used sequences as
 38     0                %;  macros, the user can reduce the amount of card manipulation
 39     0                %;  (or equivalent using a text editor).  Most powerful is the
 40     0                %;  replacement facility which allows the writing of general
 41     0                %;  control statement sequences which can be adapted to an appli-
 42     0                %;  cation by setting global variables and macro parameters.
 43     0                %;      The example will be presented without great concern for
 44     0                %;  the details of the control statements or the physics involved;
 45     0                %;  emphasis will be on the usage of MACROS.  First, several
 46     0                %;  macros will be defined.  These would probably be kept in a
 47     0                %;  macro library for general use.  Then the MACROS input will
 48     0                %;  be given, interspersed with commentary describing it.
 49     0                %;
 50     0                %;  This macro will define some tests needed for determining the
 51     0                %;  class of events:
 52     0                %;
 53                      %   MACRO T33
 54     E                TEST      2
 55     E                          -4        EQU       30001
```

```
STMT  LEVEL REPLC          SOURCE STATEMENT                                    MAC01  8SEP69

 56     E                      -4        EQU       30002
 57     E              TEST    3
 58     E                      -4        EQU       -30001
 59     E                      -4        EQU       -30002
 60     E              TEST    4
 61     E                      2         TRUE
 62     E                      3         TRUE
 63     E              %     MEND
 64     0              %;
 65     0              %;    In the actual application, there are more tests, but this
 66     0              %;    will suffice to indicate the point.  The next macro will
 67     0              %;    define default values of processor variables used globally:
 68     0              %;
 69                    %     MACRO SYMBOL
 70     E              %     NM=80
 71     E              %     DM=.04
 72     E              %     NPT=''
 73     E              %     MFAC=''
 74     E              %     LOG=''
 75     E              %     NT=''
 76     E              %     SGM=''
 77     E              %     MEND
 78     0              %;
 79     0              %;    The assignment statement for DM uses a fixed point number,
 80     0              %;    which is accepted by MACROS as if it were enclosed in quotes--
 81     0              %;    syntactically, it is a string.  This is done for convenience
 82     0              %;    in writing fixed point numbers, which is a common occurence
 83     0              %;    for SUMX.  The next macro is used to generate the sequence of
 84     0              %;    control statements for a histogram:
 85     0              %;
 86                    %     MACRO MASS(SYS,ML,LOC,WGT)
 87     E                  INVARIANT MASS OF &SYS
 88     E              &&NM      &&DM      &&ML      &&NPT     &&MFAC                    &&LOG
 89     E              &&LOC     &&WGT     &&NT      &&SGM
 90     E              %     MEND
 91     0              %;
 92     0              %;    The second and third model statements use the "non-packed" replace-
 93     0              %;    ment facility.  In all previous examples, "packed" replacement
 94     0              %;    has been used.  The packed replacement causes shifting of the
 95     0              %;    part of the statement to the right of the escape name (escape
 96     0              %;    character and identifier) to accommodate the value of the
 97     0              %;    processor variable.  However, SUMX control statements require
 98     0              %;    operands to appear in fixed columns.  The non-packed replacement
 99     0              %;    provides the needed facility--only positions occupied by the
100     0              %;    escape name and any blank positions following are used to
101     0              %;    accommodate the value.  There is no shifting of the right part
102     0              %;    of the statement.  This will be seen when the macro is used.
103     0              %;
104     0              %;    Now the MACROS input will be given.  First, some control
105     0              %;    statements for SUMX:
106     0              %;
107     0      0       *NEW PASS 1000,200
108     0      0        PHOTON + PROTON ... PI+ PI- PROTON
109     0      0       *TAPE
110     0      0        10
```

```
STMT  LEVEL REPLC        SOURCE STATEMENT                                    MAC01  8SEP69

111     0     0      *DISCARD  0
112     0     0      *LIST     8,9
113     0            %;
114     0            %;   These supply page headings and indicate general options.
115     0            %;   Now some tests will be defined:
116     0            %;
117     0     0      *SELECT
118     0            %    CALL   T33    ;BRING IN THE STANDARD TESTS
119     1     0      TEST      2
120     1     0                -4        EQU       30001
121     1     0                -4        EQU       30002
122     1     0      TEST      3
123     1     0                -4        EQU       -30001
124     1     0                -4        EQU       -30002
125     1     0      TEST      4
126     1     0                2         TRUE
127     1     0                3         TRUE
129     0            %;   ADD A SPECIAL TEST FOR THIS RUN TO SELECT 3-PRONG EVENTS
130     0     0      TEST      1
131     0     0                -2        EQU       300
132     0     0      AND       4         TRUE
133     0            %;
134     0            %;   Next, information for the user routines which compute additional
135     0            %;   quantities from the input is given.  This set of assignment
136     0            %;   statements define processor variables that specify the locations
137     0            %;   of data used by the user routines:
138     0            %;
139     0            %    PVB=701
140     0            %    M34=721
141     0            %    M35=722
142     0            %    M35L=1.08
143     0            %    M45=723
144     0            %    M45L=-.26
145     0            %    C34=724
146     0            %    C35=727
147     0            %    C45=730
148     0            %;
149     0            %;   These variables will be used to generate parameter cards for the
150     0            %;   user routines.  CHARM 8 is used to set up the bank of four-
151     0            %;   vectors (starting from symbolic BOUT location 'PVB'), from which
152     0            %;   all quantities of interest are computed:  CHARM 2 for invariant
153     0            %;   mass of the system, CHARM 3 for production cosines.
154     0            %;
155     0     0      *CHARM
156     0    BR      3P3C                8         &&PVB
156     0     1      3P3C                8         701
157     0    BR      M35                 2         &&M35     2         35        0         &&PVB
157     0     2      M35                 2         722       2         35        0         701
158     0    BR      M45                 2         &&M45     2         45        0         &&PVB
158     0     2      M45                 2         723       2         45        0         701
159     0    BR      C45                 3         &&C45     1         3         2         &&PVB
159     0     2      C45                 3         730       1         3         2         701
160     0            %;
161     0            %;   A macro that generates test statements to describe energy
162     0            %;   intervals is defined:
```

```
STMT  LEVEL REPLC        SOURCE STATEMENT                                    MAC01  8SEP69

163     0            %;
164                  %    MACRO EINTV(EN,ELOW,EHIGH)
165     E            TEST      &&EN
166     E                      &&FVB     BET       &&ELOW    &&EHIGH
167     E            %    MEND
168     0            %;
169     0            %;   Now we use it to define the test intervals:
170     0            %;
171     0     0      *SELECT
172     0            %    CALL EINTV(20,2.5,3.5)
173     1    BR      TEST      &&EN
173     1     1      TEST      20
174     1    BR                &&FVB     BET       &&ELOW    &&EHIGH
174     1     3                701       BET       2.5       3.5
176     0            %    CALL EINTV(21,3.5,4.5)
177     1    BR      TEST      &&EN
177     1     1      TEST      21
178     1    BR                &&FVB     BET       &&ELOW    &&EHIGH
178     1     3                701       BET       3.5       4.5
180     0            %;   THIS TEST IS USED FOR CONDITIONAL GENERATION OF HISTOGRAMS
181     0     0      TEST      40
182     0    BR                &&C45     BIG                 .9
182     0     1                730       BIG                 .9
183     0            %;
184     0            %;   Now the control statements for the histograms themselves will be
185     0            %;   given.  First, the macro SYMBOL will be called to define
186     0            %;   default values of processor variables:
187     0            %;
188     0            %    CALL SYMBOL
197     0            %;
198     0            %;   Then a macro that defines a group of three histograms:
199     0            %;
200                  %    MACRO  HISTGROUP
201     E            %    CALL MASS('M35',M35L,M35)
202     E            %    CALL MASS('M45',M45L,M45)
203     E            %    CALL MASS('M45 WEIGHTED',M45L,M45,10)
204     E            %    MEND
205     0            %;
206     0            %;   This macro makes use of the default value of '' assigned to
207     0            %;   omitted macro parameters in the CALL MASS statements. Finally,
208     0            %;   a macro that uses HISTGROUP is defined such that histograms
209     0            %;   for various energy intervals may be generated:
210     0            %;
211                  %    MACRO EINTHIST(EINTEST)
212     E            %;  IF TEST EINTEST SATISFIED, DO UP TO THE 'FINISH 1'
213     E            EVA       &&EINTEST
214     E            %    CALL HISTGROUP
215     E            %;  IF TEST 40 SATISFIED, DO UP TO NEXT 'FINISH 2'
216     E            EVA       40
217     E            %    CALL HISTGROUP
218     E            FINISH    2
219     E            %;   GENERALLY TEST N+1 IS THE COMPLEMENT OF TEST N
220     E            EVA       41
221     E            %    CALL HISTGROUP
222     E            FINISH    2
```

```
STMT  LEVEL REPLC        SOURCE STATEMENT                                          MAC01  8SEP69

223     E             FINISH    1
224     E             %    MEND
225     0             %;
226     0             %;   Now, by writing 2 MACROS statements, we can generate SUMX
227     0             %;   statements for 18 histograms, or a total of 54 SUMX control
228     0             %;   statements:
229     0             %;
230     0             %;
231     0      0      *BLOCK6
232     0             %;   **********HISTOGRAMS FOR THE FIRST ENERGY INTERVAL***********
233     0             %    CALL EINTHIST(20)
234     1     BR      EVA       &&EINTEST
234     1      1      EVA       20
237     3     BR        INVARIANT MASS OF &SYS
237     3      1        INVARIANT MASS OF M35
238     3     BR      &&NM      &&DM      &&ML      &&NPT     &&MFAC                    &&LOG
238     3      6      80        .04       1.08
239     3     BR      &&LOC     &&WGT     &&NT      &&SGM
239     3      4      722
242     3     BR        INVARIANT MASS OF &SYS
242     3      1        INVARIANT MASS OF M45
243     3     BR      &&NM      &&DM      &&ML      &&NPT     &&MFAC                    &&LOG
243     3      6      80        .04       .26
244     3     BR      &&LOC     &&WGT     &&NT      &&SGM
244     3      4      723
247     3     BR        INVARIANT MASS OF &SYS
247     3      1        INVARIANT MASS OF M45 WEIGHTED
248     3     BR      &&NM      &&DM      &&ML      &&NPT     &&MFAC                    &&LOG
248     3      6      80        .04       .26
249     3     BR      &&LOC     &&WGT     &&NT      &&SGM
249     3      4      723       10
252     1      0      EVA       40
255     3     BR        INVARIANT MASS OF &SYS
255     3      1        INVARIANT MASS OF M35
256     3     BR      &&NM      &&DM      &&ML      &&NPT     &&MFAC                    &&LOG
256     3      6      80        .04       1.08
257     3     BR      &&LOC     &&WGT     &&NT      &&SGM
257     3      4      722
260     3     BR        INVARIANT MASS OF &SYS
260     3      1        INVARIANT MASS OF M45
261     3     BR      &&NM      &&DM      &&ML      &&NPT     &&MFAC                    &&LOG
261     3      6      80        .04       .26
262     3     BR      &&LOC     &&WGT     &&NT      &&SGM
262     3      4      723
265     3     BR        INVARIANT MASS OF &SYS
265     3      1        INVARIANT MASS OF M45 WEIGHTED
266     3     BR      &&NM      &&DM      &&ML      &&NPT     &&MFAC                    &&LOG
266     3      6      80        .04       .26
267     3     BR      &&LOC     &&WGT     &&NT      &&SGM
267     3      4      723       10
270     1      0      FINISH    2
271     1      0      EVA       41
274     3     BR        INVARIANT MASS OF &SYS
274     3      1        INVARIANT MASS OF M35
275     3     BR      &&NM      &&DM      &&ML      &&NPT     &&MFAC                    &&LOG
```

```
STMT  LEVEL REPLC        SOURCE STATEMENT                                          MAC01  8SEP69

275     3     6        80          .04         1.08
276     3    BR        &&LOC       &&WGT       &&NT        &&SGM
276     3     4        722
279     3    BR          INVARIANT MASS OF &SYS
279     3     1          INVARIANT MASS OF M45
280     3    BR        &&NM        &&DM        &&ML        &&NPT       &&MFAC                          &&LOG
280     3     6        80          .04         .26
281     3    BR        &&LOC       &&WGT       &&NT        &&SGM
281     3     4        723
284     3    BR          INVARIANT MASS OF &SYS
284     3     1          INVARIANT MASS OF M45 WEIGHTED
285     3    BR        &&NM        &&DM        &&ML        &&NPT       &&MFAC                          &&LOG
285     3     6        80          .04         .26
286     3    BR        &&LOC       &&WGT       &&NT        &&SGM
286     3     4        723         10
289     1     0        FINISH      2
290     1     0        FINISH      1
292     0              %;
293     0              %;     **********HISTOGRAMS FOR THE SECOND ENERGY INTERVAL**********
294     0              %      CALL EINTHIST(21)
295     1    BR        EVA         &&EINTEST
295     1     1        EVA         21
298     3    BR          INVARIANT MASS OF &SYS
298     3     1          INVARIANT MASS OF M35
299     3    BR        &&NM        &&DM        &&ML        &&NPT       &&MFAC                          &&LOG
299     3     6        80          .04         1.08
300     3    BR        &&LOC       &&WGT       &&NT        &&SGM
300     3     4        722
303     3    BR          INVARIANT MASS OF &SYS
303     3     1          INVARIANT MASS OF M45
304     3    BR        &&NM        &&DM        &&ML        &&NPT       &&MFAC                          &&LOG
304     3     6        80          .04         .26
305     3    BR        &&LOC       &&WGT       &&NT        &&SGM
305     3     4        723
308     3    BR          INVARIANT MASS OF &SYS
308     3     1          INVARIANT MASS OF M45 WEIGHTED
309     3    BR        &&NM        &&DM        &&ML        &&NPT       &&MFAC                          &&LOG
309     3     6        80          .04         .26
310     3    BR        &&LOC       &&WGT       &&NT        &&SGM
310     3     4        723         10
313     1     0        EVA         40
316     3    BR          INVARIANT MASS OF &SYS
316     3     1          INVARIANT MASS OF M35
317     3    BR        &&NM        &&DM        &&ML        &&NPT       &&MFAC                          &&LOG
317     3     6        80          .04         1.08
318     3    BR        &&LOC       &&WGT       &&NT        &&SGM
318     3     4        722
321     3    BR          INVARIANT MASS OF &SYS
321     3     1          INVARIANT MASS OF M45
322     3    BR        &&NM        &&DM        &&ML        &&NPT       &&MFAC                          &&LOG
322     3     6        80          .04         .26
323     3    BR        &&LOC       &&WGT       &&NT        &&SGM
323     3     4        723
326     3    BR          INVARIANT MASS OF &SYS
326     3     1          INVARIANT MASS OF M45 WEIGHTED
```

```
STMT  LEVEL REPLC        SOURCE STATEMENT
327    3    BR      &&NM      &&DM      &&ML      &&NPT     &&MFAC                    &&LOG
327    3     6      80        .04       .26
328    3    BR      &&LOC     &&WGT     &&NT      &&SGM
328    3     4      723       10
331    1     0      FINISH    2
332    1     0      EVA       41
335    3    BR        INVARIANT MASS OF &SYS
335    3     1        INVARIANT MASS OF M35
336    3    BR      &&NM      &&DM      &&ML      &&NPT     &&MFAC                    &&LOG
336    3     6      80        .04       1.08
337    3    BR      &&LOC     &&WGT     &&NT      &&SGM
337    3     4      722
340    3    BR        INVARIANT MASS OF &SYS
340    3     1        INVARIANT MASS OF M45
341    3    BR      &&NM      &&DM      &&ML      &&NPT     &&MFAC                    &&LOG
341    3     6      80        .04       .26
342    3    BR      &&LOC     &&WGT     &&NT      &&SGM
342    3     4      723
345    3    BR        INVARIANT MASS OF &SYS
345    3     1        INVARIANT MASS OF M45 WEIGHTED
346    3    BR      &&NM      &&DM      &&ML      &&NPT     &&MFAC                    &&LOG
346    3     6      80        .04       .26
347    3    BR      &&LOC     &&WGT     &&NT      &&SGM
347    3     4      723       10
350    1     0      FINISH    2
351    1     0      FINISH    1
353    0            %;
354    0            %;   And lastly, we need two more SUMX statements:
355    0            %;
356    0     0      FINISH
357    0     0      *ALL DONE
358    0            %;
359    0            %;   Now, if one looks at the MACROS output (section V), the entire
360    0            %;   SUMX input can be seen.
361    0            %;
362    0            %;   Reference for SUMX control statement syntax:
363    0            %;     CERN SUMX Manual.  T.C. Program Library, November 1967
```

```
365   0   %;      This section will describe the elements of the MACROS
366   0   %;  language in detail.  The examples of preceding sections were
367   0   %;  of an introductory nature, while this section is designed for
368   0   %;  reference purposes.
369   0   %;      The syntax notation uses names of items enclosed in < >
370   0   %;  symbols to denote syntactic entities.  Definitions of them are
371   0   %;  either given in words or in terms of other entities.  This is
372   0   %;  indicated by an <item> followed by ::= and definitions.  The
373   0   %;  symbol | is used to indicate alternates.
374   0   %;
375   0   %;  I.  ELEMENTS
376   0   %;
377   0   %;  <identifier> -- 1-16 alphanumeric characters, the first of
378   0   %;       which must be alphabetic.  Letters are A-Z,$,#,@; digits
379   0   %;       are 0-9.  Identifiers are used to denote processor
380   0   %;       variables and macros.
381   0   %;
382   0   %;  <integer> -- 1-9 digits.  Its usage in MACROS is identical to
383   0   %;       strings, since this version has no macro-time arithmetic.
384   0   %;  <string> -- there are two forms:
385   0   %;       (1) Text string-- 0-80 characters enclosed in quotes.
386   0   %;           A quote within a string is represented by 2 consec-
387   0   %;           utive quotes.  The string of no length,or null string
388   0   %;           is represented by ''.
389   0   %;       (2) Fixed point number--string of digits preceded by, or
390   0   %;           followed by, or containing a decimal point.  This
391   0   %;           construct is included for convenience in writing
392   0   %;           MACROS statements; but it is not a number and does
393   0   %;           not possess a numeric value.  It should be thought
394   0   %;           of as if it were enclosed in quotes.
395   0   %;
396   0   %;  <constant> ::= <integer> | <string>
397   0   %;
398   0   %;  <formal parameter> ::= <identifier>
399   0   %;       The formal parameter is declared by appearance in a
400   0   %;       MACRO statement.  Its value is defined when the macro
401   0   %;       is expanded--rules are given below.
402   0   %;
403   0   %;  <simple variable> ::= <identifier>
404   0   %;       Simple variables are declared and assigned values
405   0   %;       by appearance on the left side of an assignment statement.
406   0   %;       They assume the type of the right side of the statement--
407   0   %;       in this version of MACROS it can always be considered
408   0   %;       string type since no macro-time arithmetic is permitted.
409   0   %;
410   0   %;  <processor variable> ::= <simple variable> | <formal parameter>
411   0   %;
412   0   %;  <processor expression> ::= <constant> | <processor variable>
413   0   %;       The processor expression is used on the right side of
414   0   %;       assignment statements or in actual parameters of CALL
415   0   %;       statements.  Its value is either the string value of
416   0   %;       the constant or the value currently associated with
417   0   %;       the processor variable.  If the variable has no value
418   0   %;       associated with it, then the expression is erroneous,
```

```
419   0          %;      and its value is taken to be the null string.
420   0          %;
421   0          %;  Examples:
422   0          %;      <identifier> --  ABCD   $$$   OTHER1
423   0          %;      <integer> --  0  123  123456789
424   0          %;      <string> --  'TEXT STRING'  'PEOPLE''S PARK'  ''
425   0          %;                  .123  12.3  456.   (Fixed point number)
426   0          %;      <processor expression> --   ABCD  123   .9   'STRING'
427   0          %;
428   0          %;
429   0          %;  II.  PROCESSOR STATEMENTS
430   0          %;
431   0          %;  General:
432   0          %;      Processor statements always have the character '%' in
433   0          %;  column 1.  The body of the statement must be in columns 2-72.
434   0          %;  No continuation statements are allowed.  With the exception of
435   0          %;  assignment and null statements, they begin with a reserved word
436   0          %;  followed by an operand field.  Comments may follow the
437   0          %;  logical end of the statement when separated by a semi-colon
438   0          %;  (;).  Blanks may be used freely between elements and keywords,
439   0          %;  they are only required to separate alphanumeric items.
440   0          %;      Processor statements are not subject to replacement (section
441   0          %;  IV) or may they be generated by replacement.  Thus if a '%' is
442   0          %;  generated by replacement in column 1, MACROS will not recognize
443   0          %;  the statement as a processor statement.
444   0          %;      Processor statements are always printed at top level and
445   0          %;  while a macro is being edited.  They are never printed at lower
446   0          %;  levels (i.e. within a macro expansion).
447   0          %;
448   0          %; Null or comment statement--
449   0          %;  Form:     % ;  comments
450   0          %;      The null statement is ignored by MACROS.  When in a macro,
451   0          %;  it is not encoded, so no space is required for it in the
452   0          %;  edited macro.
453   0          %;  Example:
454   0          %;      %;The report you are reading is mostly null statements
455   0          %;
456   0          %; MACRO statement
457   0          %;  Form:     %  MACRO  <macroname> (<formal parameter list>)
458   0          %;      or   %  MACRO  <macroname>
459   0          %;              <macroname> ::= <identifier>
460   0          %;              <formal parameter list> ::=  <formal parameter> |
461   0          %;                   <formal parameter list> , <formal parameter>
462   0          %;      The MACRO statement heads a macro definition.  The macro-
463   0          %;  name is used in CALL statements to reference the macro.  It may
464   0          %;  be the same as a processor variable identifier.  No two macros
465   0          %;  may have the same name.
466   0          %;      The formal parameter list, if supplied declares the
467   0          %;  identifiers as formal parameters.  A maximum of ten may be given.
468   0          %;  Details of their interpretation are in the next section.
469   0          %;      Macros may not be nested.  The macro definition ends at
470   0          %;  a MEND statement, the next MACRO statement, or end-of-file.
471   0          %;  Examples:
472   0          %;      %  MACRO  FIBONACCI(I)
473   0          %;      %  MACRO  $DECLARATIONS
```

```
STMT  LEVEL REPLC          SOURCE STATEMENT                                MAC01  8SEP69

474     0              %;        %  MACRO  FUNCTION   ( TYP,ARG )
475     0              %;
476     0              %;  MEND statement
477     0              %;   Form:     %  MEND
478     0              %;        The MEND statement terminates a macro definition that
479     0              %;   has not been otherwise ended.
480     0              %;
481     0              %;   CALL Statement
482     0              %;   Form:     %  CALL  <macroname> (<actual parameter list>)
483     0              %;        or   %  CALL  <macroname>
484     0              %;             <actual parameter list> ::=  <actual parameter>  |
485     0              %;                  <actual parameter list> , <actual parameter>
486     0              %;             <actual parameter> ::= <processor expression>
487     0              %;        The CALL statement invokes the macro <macroname>,
488     0              %;   which must have been previously defined or be available on file
489     0              %;   SYSLIB.
490     0              %;        The actual parameter list if present is used to assign
491     0              %;   values to the formal parameters.  Details are given in the next
492     0              %;   section.
493     0              %;   Examples:
494     0              %;        %   CALL  FIBONNACI ( 1 )
495     0              %;        %   CALL  $DECLARATIONS
496     0              %;        %   CALL  FUNCTION ( 'SIN' , 'X+Y' )
497     0              %;
498     0              %;  TITLE statement
499     0              %;   Form:     %  TITLE   <string>
500     0              %;         or  %   TITLE
501     0              %;        The TITLE statement sets the page title to <string> if
502     0              %;   present and causes a page eject.  Omitted operand leaves the
503     0              %;   title unchanged and causes a page eject only.  The statement
504     0              %;   is not processed within a macro definition--only when it is
505     0              %;   expanded.  The TITLE statement is never printed.
506     0              %;   Example:
507     0              %;        %   TITLE  'III. MACROS Language Description'
508     0              %;        The above was used to get the current page title.
509     0              %;
510     0              %;  Assignment statement
511     0              %;   Form:     %  <leftside> =  <processor expression>
512     0              %;             <leftside> ::=  <processor variable>
513     0              %;        The assignment statement is used to set the value of the
514     0              %;   leftside to the value of the processor expression.  If the
515     0              %;   leftside is a processor variable, then the statement also
516     0              %;   declares the identifier as such if it has not been used
517     0              %;   before.  If it is a formal parameter, then the assignment
518     0              %;   is only retained during this expansion of the macro.  The
519     0              %;   actual parameter, if it was a processor variable, will not
520     0              %;   be changed by the assignment.
521     0              %;
522     0              %;
523     0              %;   III. MACRO EXPANSIONS
524     0              %;
525     0              %;        When a macro is expanded using the CALL statement, the
526     0              %;   following actions take place:
527     0              %;   (1)  If any formal parameters were declared, any values currently
528     0              %;        assigned to the formal parameter identifiers are saved on a
```

```
0          %;      pushdown stack.
0          %;  (2) The values of the actual parameters supplied in the
0          %;      CALL statement are assigned to the formal parameter
0          %;      identifiers.  If any or all of the actual parameters are
0          %;      omitted, then the corresponding formal parameters are
0          %;      set to the null string.  Excess actual parameters are
0          %;      ignored.  Thus a call statement of the form
0          %;           % CALL FUNCTION(,3)
0          %;      for the macro FUNCTION above, will set TYP='' and ARG='3'.
0          %;  (3) MACROS begins fetching input from the body of the macro
0          %;      definition.  The processor variables that are currently
0          %;      active will be used during processing--this includes
0          %;      any variables set by macros that called this one.
0          %;  (4) When the end of the macro definition is reached, the
0          %;      values previously associated with the formal parameter
0          %;      identifiers are restored with the values saved on the
0          %;      pushdown stack, if any.  MACROS then resumes fetching
0          %;      statements in the environment of the invoking CALL
0          %;      statement; but changes made to variables global to the
0          %;      called macro are retained.
0          %;  The maximum depth of macro calls is 10 levels.
0          %;
0          %;
0          %;  IV.  REPLACEMENT IN BASE LANGUAGE STATEMENTS
0          %;
0          %;       Every base language statement (i.e. no '%' in col 1)
0          %;  outside of a macro definition is scanned by MACROS for the
0          %;  escape character '&', which signals a potential replacement.
0          %;  If the '&' is followed by an identifier or by another '&' and
0          %;  an identifier, and if the identifier is that of a processor
0          %;  variable that is currently active, then a replacement is per-
0          %;  formed.  The single ampersand is used to denote "packed"
0          %;  replacement, for free format applications; and the double
0          %;  ampersand for "non-packed" replacement, for use where column
0          %;  positions are critical.
0          %;       The term "escape name" is used to refer to the escape
0          %;  character(s) and the immediately following identifier.  For one
0          %;  case of packed replacement, a period delimiter is included in
0          %;  the escape name.
0          %;       Scanning for an escape character proceeds from left to
0          %;  right.  The scanning field is the entire Fortran statement,
0          %;  including continuation lines, if the FORT=1 option is used;
0          %;  else it is the columns between the begin and end columns (inclu-
0          %;  sive) of the 80 character record.  The default scanning columns
0          %;  are 1 thru 80; they may be changed by the user if needed.
0          %;       When an escape name if found, and replacement is done (the
0          %;  identifier has a value), then the scan is restarted from the begin
0          %;  column.  If the identifier does not have a value, then the scan
0          %;  proceeds to the right.  Rescanning continues until no
0          %;  more escape characters remain, or no more replacements can be
0          %;  done.  The rescanning mechanism allows construction of escape
0          %;  names by replacement.  The maximum number of replacements allowed
0          %;  in a base language statement is 30.
0          %;       Specific rules for the replacement modes are as follows:
0          %;
```

```
584     0          %;   (1)  Packed replacement--This is indicated by a single '&'
585     0          %;        followed by an identifier, delimited by any special
586     0          %;        character.  If it is delimited by '.', then the period
587     0          %;        will also be removed from the text--this allows concat-
588     0          %;        enation of a replacement value with a letter or digit.
589     0          %;        Replacement is done by removing the escape character
590     0          %;        and the identifier (and period, if necessary) from the
591     0          %;        text, and inserting the value associated with the iden-
592     0          %;        tifier in its place.  The remainder of the statement to
593     0          %;        the right of the escape name , extending to the end of the
594     0          %;        scanning field, is shifted right or left as needed to
595     0          %;        accommodate the replacement value.  Blanks will be added
596     0          %;        at the end of the statement if the value is shorter than
597     0          %;        the escape name, or truncation will be preformed if the
598     0          %;        value is longer.  In the latter case, loss of non-blank
599     0          %;        characters will be noted by MACROS.
600     0          %;
601     0          %;   (2)  Non-packed replacement--This is denoted by a double
602     0          %;        escape character '&&' followed by an identifier.  The
603     0          %;        identifier is delimited by any special character (no
604     0          %;        special rule for period applies).  The replacement
605     0          %;        of the escape name is done without any shifting of the
606     0          %;        part of the statement to the right of the escape name.
607     0          %;        If the replacement value is shorter than the escape
608     0          %;        name, then blanks are added to it.  If the value is longer,
609     0          %;        then blank positions following the escape name are used
610     0          %;        to hold the trailing non-blank characters of the value.
611     0          %;        If there is still not enough room, even using the blank
612     0          %;        positions, and discounting trailing blanks of the value,
613     0          %;        then MACROS notes the truncation of the value.
```

```
STMT  LEVEL REPLC          SOURCE STATEMENT                                  MAC01  8SEP69

615     0              %;  MACROS is a load module named MACRO01 which can be used
616     0              %;  from the library PUB.JEA.LOADMODS.  It uses the following
617     0              %;  ddnames:
618     0              %;
619     0              %;  SYSPRINT -- output listing. DCB information is supplied by the
620     0              %;       program--RECFM=VB,LRECL=125,BLKSIZE=3425.  The PGEN option
621     0              %;       may be set to suppress the output listing in whole or part.
622     0              %;  SYSIN -- primary input file. May be any sequential dataset
623     0              %;       or partitioned dataset member (or concatenation) with
624     0              %;       logical record length 80 bytes.
625     0              %;  SYSOUT -- MACROS processed output.  DCBs must be supplied by
626     0              %;       user such that LRECL=80 is used.  Generally, this file
627     0              %;       is associated with a temporary dataset that is used later
628     0              %;       in the job.  An option may be given to MACROS to suppress
629     0              %;       writing on SYSOUT.
630     0              %;  SYSLIB -- secondary input file for macro definitions.  This
631     0              %;       is used to make a library of macro definitions available
632     0              %;       to MACROS.  If not required, then the DD statement can be
633     0              %;       omitted.  If used, the DD statement must define a sequential
634     0              %;       dataset or partitioned dataset member.  Concatenations of
635     0              %;       these two items are acceptable if the DCBs are the same.
636     0              %;       The logical record length must be 80 bytes.
637     0              %;
638     0              %;       Options are passed to MACROS using the PARM field on the
639     0              %;  EXEC statement.  The format is a list of keywords followed by
640     0              %;  an equal sign and an integer, separated by commas.  The options
641     0              %;  and their default values are:
642     0              %;
643     0              %;  BEGC (1)        Begin column for scanning of base language state-
644     0              %;                   ments.  This is ignored if the FORT=1 option is
645     0              %;                   used.
646     0              %;  ENDC (80)       End column for scanning of base lanugage state-
647     0              %;                   ments.  Also ignored if FORT=1.
648     0              %;  MOUT (1)        If not zero, then the MACROS output is written
649     0              %;                   on SYSOUT.
650     0              %;  PGEN (2)        If zero, then no output listing is produced.
651     0              %;                   If one, then top level statements only are listed.
652     0              %;                   If two, then base language statements from macro
653     0              %;                   expansions are listed in addition to top level
654     0              %;                   statements.
655     0              %;  PLIB (0)        If not zero, then the macro definitions edited
656     0              %;                   from SYSLIB are listed when it is opened.
657     0              %;  FORT (0)        If not zero, then the FORTRAN statement conven-
658     0              %;                   tions are used--columns 1-72 of the first card of
659     0              %;                   a statement and columns 7-72 of any continuation
660     0              %;                   cards are treated as a single base language
661     0              %;                   statement for replacement scanning.  If there are
662     0              %;                   no Hollerith literals in quotes in the statement,
663     0              %;                   then trailing blanks at the end of each line are
664     0              %;                   removed.  There is a limit of 270 characters that
665     0              %;                   may be retained, discounting trailing blanks.
666     0              %;                   Thus, in the worst case, 3 continuation cards are
667     0              %;                   allowed.
668     0              %;  PRBS (0)        If not zero, then 'PRint Before Substitution',
```

```
STMT  LEVEL REPLC          SOURCE STATEMENT                                    MAC01  8SEP69

669     0            %;                 which causes base language statements containing
670     0            %;                 escape characters to be listed before replacement
671     0            %;                 is done.  Normally, they are listed after replace-
672     0            %;                 ment only.
673     0            %;  SIZS (10000)   The amount of space (in bytes) to be allocated to
674     0            %;                 storage of the values of variables.  Should one
675     0            %;                 find that he has run out (error message), he can
676     0            %;                 rerun with a larger value (up to 32767).  Space
677     0            %;                 could run out if a large number of variables are
678     0            %;                 all active at the same time.
679     0            %;
680     0            %;      When MACROS is run, the first page of the listing gives the
681     0            %;  options used, and a number 'SIZW', which is the amount of free
682     0            %;  space in bytes that MACROS thinks it has for storage of strings,
683     0            %;  macro definitions and control information.  MACROS will use all
684     0            %;  the free space in the region in which it is run.  Minimum region
685     0            %;  size is about 100k bytes.
686     0            %;
687     0            %;      Errors in MACROS input are flagged with a text message
688     0            %;  indicating the type of error,disposition, and location.
689     0            %;  Severities are associated with the errors ranging from 4-16.
690     0            %;  The highest severity is passed back to OS as a return code
691     0            %;  that can be tested in JCL to suppress subsequent job steps.
692     0            %;
693     0            %;  JOB CONTROL LANGUAGE EXAMPLES
694     0            %;
695     0            %;  To use MACROS for preparation of a FORTRAN source program,
696     0            %;  one might use:
697     0            %;
698     0            %;  //jobname  JOB   etc
699     0            %;  //JOBLIB DD DSN=PUB.JEA.LOADMODS,DISP=(SHR,PASS)
700     0            %;  //MACROS  EXEC  PGM=MACRO01,PARM='FORT=1'
701     0            %;  //SYSPRINT  DD  SYSOUT=A
702     0            %;  //SYSLIB  DD   DSN=PUB.xxx.macrolibrary,DISP=SHR
703     0            %;  //SYSOUT  DD  DSN=&MACOUT,DISP=(NEW,PASS),UNIT=SYSDA,
704     0            %;  //    SPACE=(CYL,(2,1)),DCB=(RECFM=FB,LRECL=80,BLKSIZE=3200)
705     0            %;  //SYSIN  DD  *
706     0            %;     MACROS input
707     0            %;  /*
708     0            %;  //FORTRAN  EXEC  FORTHCLG,COND=(5,LT)
709     0            %;  //FORT.SYSIN  DD  DSN=&MACOUT,DISP=(OLD,DELETE)
710     0            %;  //LKED. cards if needed
711     0            %;  //GO.  cards
712     0            %;
713     0            %;  The SYSLIB DD statement is optional; if used, the user supplies
714     0            %;  the appropriate parameters to specify his library.  For testing
715     0            %;  purposes, it may be given as //SYSLIB DD *, followed by an
716     0            %;  ordinary input stream dataset.
717     0            %;  The COND parameter on the EXEC FORTHCLG statement will
718     0            %;  suppress remaining steps in the procedure if any preceding
719     0            %;  (MACROS,FORTRAN.FORT,FORTRAN.LKED) steps had a return code
720     0            %;  6 or higher(i.e., stop on error level higher than warning).
721     0            %;
722     0            %;  To use MACROS with SUMX, this JCL is representative:
723     0            %;
```

```
STMT  LEVEL REPLC          SOURCE STATEMENT                                    MAC01  8SEP69

724     0            %;   //jobname  JOB  etc
725     0            %;   //JOBLIB  DD  DSN=PUB.JEA.LOADMODS,DISP=(SHR,PASS)
726     0            %;   //MACROS  EXEC  PGM=MACRO01
727     0            %;   //SYSPRINT  DD  SYSOUT=A
728     0            %;   //SYSLIB  DD   DSN=PUB.xxx.macrolibrary,DISP=SHR
729     0            %;   //SYSOUT  DD  DSN=&MACOUT,DISP=(NEW,PASS),UNIT=SYSDA,
730     0            %;   //    SPACE=(CYL,(2,1)),DCB=(RECFM=FB,LRECL=80,BLKSIZE=3200)
731     0            %;   //SYSIN  DD  *
732     0            %;     MACROS input
733     0            %;   /*
734     0            %;   //SUMX  EXEC  FORTHCLG,COND=(5,LT)
735     0            %;   //FORT.SYSIN  DD  *
736     0            %;        READST, CHARMS etc
737     0            %;   /*
738     0            %;   //LKED.LIB  DD  Info for SUMX load module library
739     0            %;   //LKED.SYSIN  DD  *
740     0            %;        INCLUDE  LIB(SUMXloadmodulename)
741     0            %;        ENTRY  MAIN
742     0            %;   /*
743     0            %;   //GO. cards for tapes,etc
744     0            %;   //GO.SYSIN  DD  DSN=&MACOUT,DISP=(OLD,DELETE)

          ---M A C R O S PROCESSING COMPLETED,HIGHEST SEVERITY= 0
```

V. MACROS O/P FROM EXAMPLES OF SECT II

```
                    ***EXAMPLE 1-1***
C
C     MAIN PROGRAM FOLLOWS
      COMMON   A,B
C ... REMAINDER OF MAIN PROGRAM
      END
      SUBROUTINE SUB
      COMMON   A,B
C ... REMAINDER OF SUBROUTINE SUB
      RETURN
      END
                    *****************
                    ***EXAMPLE 1-2***
      SUBROUTINE SUB2
      COMMON  ZORCH
C ... REMAINDER OF SUB2
      RETURN
      END
                    *****************
                    ***EXAMPLE 2-1***
      DIMENSION A(10)
C  READ NUMBER OF DATA ITEMS
      READ (5,1)N
C  CHECK IF ARRAY A LARGE ENOUGH
      IF(N.LE.10) GO TO 3
C  ... DO SOMETHING ABOUT ERROR CONDITION
C  ASSUME WE USE ONLY 10 ITEMS THEN
      N=10
3     CONTINUE
C  ... PROCESS DATA
C
      END
                    *****************
                    ***EXAMPLE 3-1***
      REAL FUNCTION DISCRM*8(A,B,C)
      REAL*8  B,A,C
      DISCRM=DSQRT(B*B-4*A*C)
      RETURN
      END
                    *****************
                    ***EXAMPLE 3-2***
      REAL FUNCTION DISCRM*4(A,B,C)
      REAL*4  B,A,C
      DISCRM=SQRT(B*B-4*A*C)
      RETURN
      END
                    *****************
                    ***EXAMPLE 4-1***
      NAMELIST /DB/      A,B,C
      WRITE (6,DB)
                    *****************
                    ***EXAMPLE 4-2***
      NAMELIST /DB/      IDEN1,IDEN2,IDEN3,IDEN4,IDEN5,IDEN6,IDEN7,FOOBA
     1Z
      WRITE (6,DB)
                    *****************
                    ***EXAMPLE 4-3***
&DUM  F(I)=F(I-2)  +F(I-1)
```

V. MACROS O/P FROM EXAMPLES OF SECT II

```
&DUM  FORMAT(   'THIS IS A HOLLERITH LITERAL IN QUOTES'
     1   'NOTICE THAT ALL BLANKS ARE RETAINED'   )
                        *****************
                        ***EXAMPLE 5-1***
      REAL FUNCTION DISCRM*8(A,B,C)
      REAL*8  B,A,C
      DISCRM=DSQRT(B*B-4*A*C)
      RETURN
      END
                        *****************
                        ***EXAMPLE 5-2***
THE TOP LEVEL VALUES OF RLEN=4, AND FPRE=.
                        *****************
*NEW PASS 1000,200
 PHOTON + PROTON ... PI+ PI- PROTON
*TAPE
 10
*DISCARD  0
*LIST     8,9
*SELECT
TEST      2
          -4        EQU       30001
          -4        EQU       30002
TEST      3
          -4        EQU       -30001
          -4        EQU       -30002
TEST      4
          2         TRUE
          3         TRUE
TEST      1
          -2        EQU       300
AND       4         TRUE
*CHARM
3P3C                8         701
M35                 2         722       2         35        0         701
M45                 2         723       2         45        0         701
C45                 3         730       1         3         2         701
*SELECT
TEST      20
          701       BET       2.5       3.5
TEST      21
          701       BET       3.5       4.5
TEST      40
          730       BIG                 .9
*BLOCK6
EVA       20
  INVARIANT MASS OF M35
80        .04       1.08
722
  INVARIANT MASS OF M45
80        .04       .26
723
  INVARIANT MASS OF M45 WEIGHTED
80        .04       .26
723       10
EVA       40
  INVARIANT MASS OF M35
```

```
80          .04         1.08
722
  INVARIANT MASS OF M45
80          .04         .26
723
  INVARIANT MASS OF M45 WEIGHTED
80          .04         .26
723         10
FINISH      2
EVA         41
  INVARIANT MASS OF M35
80          .04         1.08
722
  INVARIANT MASS OF M45
80          .04         .26
723
  INVARIANT MASS OF M45 WEIGHTED
80          .04         .26
723         10
FINISH      2
FINISH      1
EVA         21
  INVARIANT MASS OF M35
80          .04         1.08
722
  INVARIANT MASS OF M45
80          .04         .26
723
  INVARIANT MASS OF M45 WEIGHTED
80          .04         .26
723         10
EVA         40
  INVARIANT MASS OF M35
80          .04         1.08
722
  INVARIANT MASS OF M45
80          .04         .26
723
  INVARIANT MASS OF M45 WEIGHTED
80          .04         .26
723         10
FINISH      2
EVA         41
  INVARIANT MASS OF M35
80          .04         1.08
722
  INVARIANT MASS OF M45
80          .04         .26
723
  INVARIANT MASS OF M45 WEIGHTED
80          .04         .26
723         10
FINISH      2
FINISH      1
FINISH
*ALL DONE
```