I. MACROS -- STATEMENT ORIENTED MACRO PROCESSOR

John Akerh, SLAC Computer Group, September 1969

MACROS is a macro processor designed for use with language
processors that lack a macro facility. Its usage can signifi-
cantly reduce the amount of clerical work normally required to
use such language processors as FORTRAN or SUX (considering SUX
as a processor of its control language).

For one not familiar with macro languages, they can be
characterized as providing facilities for describing the generation
of a program, in a manner analogous to the way that a program-
ing language such as FORTRAN provides facilities for describing
a computation. The "data" of the macro processor consists of
statements or fragments of statements of the language used in
the generated program. The language of the generated program
is commonly called the "base language." The macro language
uses constructs which are recognized by the "macro processor"
(MACROS) to describe operations on the base language--order of
generation and modifications of the base language statements.
The output of the macro processor is a program in the base
language, ready for its processor.

MACROS can be used with any base language where statements
can be treated as being within specified columns of each record
(card or line), although special facilities are provided for
to handle FORTRAN continuation conventions. If the need for
recognition of statements split across records can be avoided,
i.e., the base language continuation rules need not be accepted.
MACROS can be used with other base languages. It provides
definition and expansion of macro-instructions (called "macros")
and retrieval of macro definitions from an external library--
these facilities control the order of generation of base
language statements; and replacement of symbolic names in
base language statements in its input--whereby modifications
can be obtained.

This report is organized as a series of examples that
gradually introduce features of the MACROS language, and
reference sections that summarize the language and information
needed to use MACROS (JCL, etc.). Its contents are:

1. Introduction--this section

1. Examples of MACROS usage--
   1. Use of macro to include text--introduces macros,
      usage of macro library for maintaining FORTRAN
      declaration statements.
   2. Use of replacement to generalize program--introduces
      replacement as a method of setting parameters that are
      fixed at execution time.
   3. Use of replacement to define function type--explains
      additional replacement facilities in context of
      writing a FORTRAN function for single and double
      precision.
   4. Macro parameters, continuation statements--introduces
      macro parameters and FORTRAN continuation statement
      processing to define a macro that generates debugging
I. MACROS --- STATEMENT ORIENTED MACRO PROCESSOR

MACRO 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---illu-
strates 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
Computations 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 in 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
Computations 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
I. MACROS -- STATEMENT ORIENTED MACRO PROCESSOR

STMT  LEVEL REPLC  SOURCE STATEMENT
111    0           %: Report ANL-7309, June 1967.)
II. Example 1--use of macro to include text

A problem that constantly recurs in the preparation of a large FORTRAN program is providing updated copies of declaration statements in each subroutine. What is needed is the ability to maintain one copy of each set of declarations and include it into the subroutine being compiled immediately prior to compilation.

MACROS provides this facility by filing away a sequence of text under a name and allowing insertion of this text at specified points in the source program. A sequence of text used in this manner is a "macro definition," or more often, a "macro."

The process of insertion is a "macro expansion." To define a macro and give it a name, special statements which are recognized by MACROS are used to bracket the macro statements (i.e., text) of the macro definition. Another MACROS statement is used to invoke the macro expansion at the point it is written.

In the following example, a main program and a subroutine are prepared for compilation using MACROS. They use the same COMMON statement, which will be supplied with a macro.

***EXAMPLE 1-1***

MACRO COM1 ;START MACRO DEFINITION
NEXT IS A MODEL STATEMENT OF THE BASE LANGUAGE (FORTRAN)
COMMON A,B
END MACRO DEFINITION
MAIN PROGRAM FOLLOWS
CALL COM1 ;THIS INCLUDES THE COMMON STATEMENT
COMMON A,B
REMAINDER OF MAIN PROGRAM
END
SUBROUTINE SUB
CALL COM1 ;THIS INCLUDES COMMON IN SUBROUTINE
COMMON A,B
REMAINDER OF SUBROUTINE SUB
RETURN
END

The first three statements define the macro COM1 whose function is to insert the COMMON statement into the source program. The expansion of COM1 can be seen following the CALL statements.

The final output of MACROS can be determined by use of the information appearing on the left of the page. The first column is a statement number assigned by MACROS to all statements (both processor and base language) presented to it. Its primary use is for reference purposes in error messages. The second column is the macro next level. The number in this column indicates from what depth MACROS is processing statements. Zero is top level, that is not from a macro expansion. In the case above, the expansions of the macro COM1 are at level 1 since COM1 was invoked from the top level. If a macro is being edited, as in the first four statements of the example, then the second column has an 'E' in it, indicating that
II. Example 1--use of macro to include text

A SOURCE STATEMENT

MACRO 8SRP69

<table>
<thead>
<tr>
<th>STAT LEVEL REPLC</th>
</tr>
</thead>
<tbody>
<tr>
<td>169 0 4</td>
</tr>
<tr>
<td>170 0 4</td>
</tr>
<tr>
<td>171 0 4</td>
</tr>
<tr>
<td>172 0 4</td>
</tr>
<tr>
<td>173 0 4</td>
</tr>
<tr>
<td>174 0 4</td>
</tr>
<tr>
<td>175 0 4</td>
</tr>
<tr>
<td>176 0 4</td>
</tr>
<tr>
<td>177 0 4</td>
</tr>
<tr>
<td>178 0 4</td>
</tr>
<tr>
<td>179 0 4</td>
</tr>
<tr>
<td>180 0 4</td>
</tr>
<tr>
<td>181 0 4</td>
</tr>
<tr>
<td>182 0 4</td>
</tr>
<tr>
<td>183 0 4</td>
</tr>
<tr>
<td>184 0 4</td>
</tr>
<tr>
<td>185 0 4</td>
</tr>
<tr>
<td>186 0 4</td>
</tr>
<tr>
<td>187 0 4</td>
</tr>
<tr>
<td>188 0 4</td>
</tr>
<tr>
<td>189 0 4</td>
</tr>
<tr>
<td>190 0 4</td>
</tr>
<tr>
<td>191 0 4</td>
</tr>
<tr>
<td>192 0 4</td>
</tr>
<tr>
<td>193 0 4</td>
</tr>
<tr>
<td>194 0 4</td>
</tr>
<tr>
<td>195 0 4</td>
</tr>
<tr>
<td>196 0 4</td>
</tr>
<tr>
<td>197 0 4</td>
</tr>
<tr>
<td>198 0 4</td>
</tr>
<tr>
<td>199 0 4</td>
</tr>
<tr>
<td>200 0 4</td>
</tr>
<tr>
<td>201 0 4</td>
</tr>
<tr>
<td>202 0 4</td>
</tr>
<tr>
<td>203 0 4</td>
</tr>
<tr>
<td>204 0 4</td>
</tr>
<tr>
<td>205 0 4</td>
</tr>
<tr>
<td>206 0 4</td>
</tr>
<tr>
<td>207 0 4</td>
</tr>
<tr>
<td>208 0 4</td>
</tr>
<tr>
<td>209 0 4</td>
</tr>
<tr>
<td>210 0 4</td>
</tr>
<tr>
<td>211 0 4</td>
</tr>
<tr>
<td>212 0 4</td>
</tr>
<tr>
<td>213 0 4</td>
</tr>
<tr>
<td>214 0 4</td>
</tr>
</tbody>
</table>
| 215 0 0 | SUBROUTINE SUB2

***EXAMPLE 1-2***

176 0 0 | MACRO COM
178 0 0 | COMMON ZORCH
180 0 0 | END
182 0 0 | CALL COM ;THIS CALL WILL CAUSE SYSLIB TO BE OPENED
184 0 0 | COMMON ZORCH
186 0 0 | C ... REMAINDER OF SUB2
188 0 0 | RETURN
II. Example 1--use of macro to include text

SOURCEx STATEMENT

275 0 0
276 0 0
277 0 %: Optionally MACROS will print the editing of macro
278 0 %: definitions from SYSLIB as is done here. Editing from SYSLIB
279 0 %: is indicated by a '%%' in the second column on the left.
280 0 %: After all macro definitions on SYSLIB are read, the CALL
281 0 %: statement can be processed. (in this case COM was the only
282 0 %: macro in SYSLIB.)
283 0 %: The only specific requirement about the order of appearance
284 0 %: of macro definitions is that they appear before they are used.
285 0 %: This can be satisfied by direct inclusion in the MACROS input
286 0 %: anywhere before the first CALL statement referencing them, or
287 0 %: by placing the definitions in a separate library prior to a
288 0 %: MACROS run using that library.
Example 2—use of replacement to generalize program

This and subsequent examples will utilize the second
facility of macro processors—modification of base language
statements. MACROS provides a "replacement" or "substitution"
facility where occurrences of symbolic names (processor vari-
ables) in base language statements can be replaced by the
(character string) value associated with the processor variable.
The following demonstrates a simple usage of replacement
to set array bounds and describe computations depending on
the array bounds. This enables "tailoring" the program to a
specific application immediately prior to compilation by
use of processor statements to set the values of processor
values:

```plaintext
***EXAMPLE 2-1***
```

```plaintext
ADIM=10  ; bound of array A
DIMENSION A(ADIM)
READ (5,1)N
C CHECK IF ARRAY A LARGE ENOUGH
  IF(N.IE.10) GO TO 3
  IF(N.IE.30) GO TO 3
C ... DO SOMETHING ABOUT ERROR CONDITION
C ASSUME WE USE ONLY ADIM ITEMS THEN
C ASSUME WE USE ONLY 10 ITEMS THEN
N=ADIM
N=10
=N=10
N=10
C ... PROCESS DATA
```

The first statement is a processor assignment statement which
sets the value of processor variable ADIM to 10. The statement
also declares ADIM as a processor variable since this
is the first use of ADIM. The effect of the statement is to
make occurrences of the name ADIM preceded by the "escape
character" ' ' in base language statements eligible for
replacement. The combination of the escape character and the
processor variable identifier is called the "escape name" in
this discussion. "Replacement" replaces the escape name in the
text by the character string value of the processor variable.
This occurs in the four FORTRAN statements that use the escape
name 'ADIM' to obtain the bounds of array A.

Now we can see the use of the third column on the left
of the page (the REPLACEMENT COLUMN). The first printing
of the statements using ADIM shows the statement before
replacement (hence the 'BR' in the REPL column). The
second printing shows the statement as it will actually
appear in the output of MACROS—the occurrence(s) of
ADIM and the escape character have been replaced by the value
of ADIM (the string representation of 10). The number in
II. Example 2--use of replacement to generalize program

<table>
<thead>
<tr>
<th>STAT</th>
<th>LEVEL</th>
<th>REPLC</th>
<th>SOURCE STATEMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>292</td>
<td>0</td>
<td></td>
<td>%: the replacement column (1) indicates that one replacement</td>
</tr>
<tr>
<td>293</td>
<td>0</td>
<td></td>
<td>%: has occurred. (Note—the printing before replacement is</td>
</tr>
<tr>
<td>294</td>
<td>0</td>
<td></td>
<td>%: optional and is normally suppressed.)</td>
</tr>
<tr>
<td>295</td>
<td>0</td>
<td></td>
<td>%: What we have gained by replacement is the ability to</td>
</tr>
<tr>
<td>296</td>
<td>0</td>
<td></td>
<td>%: set parameters in the program which would either be impos-</td>
</tr>
<tr>
<td>297</td>
<td>0</td>
<td></td>
<td>%: sible to set at execution time or would cost execution time</td>
</tr>
<tr>
<td>298</td>
<td>0</td>
<td></td>
<td>%: efficiency. The next example indicates a use of replacement</td>
</tr>
<tr>
<td>299</td>
<td>0</td>
<td></td>
<td>%: to solve an impossible case for normal FORTRAN.</td>
</tr>
</tbody>
</table>
II. Example 3—use of replacement to define function type

SOURCE STATEMENT

MACRO MDISCR

REAL FUNCTION DISCRM*8&LEN(A,B,C)
REAL*LEN B,A,C
DISCRM=FPRE.SQRT(B*B-4*A*C)
RETURN
END
MEND

LEN in the real data length [4 or 8]. PPRE is the prefix
required for double precision ("D"). To get the double
precision version, use the following:

***EXAMPLE 3-1***

LEN=8
FPRE="D"

CALL MDISCR

REAL FUNCTION DISCRM*8&LEN(A,B,C)
REAL*LEN B,A,C
DISCRM=FPRE.SQRT(B*B-4*A*C)
DISCRM=DQRT(B*B-4*A*C)
RETURN
END

The two assignment statements before the CALL statement set the
values of LEN and PPRE. The replacement for LEN is
similar to that used in the previous example. The use
of PPRE illustrates the facility used to recognize the end
of an occurrence of a processor variable when it is to be
followed by a letter or number. The period following PPRE is
also removed from the text when PPRE's string value ("D") is
substituted. The assignment statement for PPRE shows that
the type of processor variable is determined dynamically,
and may change in different uses. In theory, LEN has
integer type, while PPRE has string type; however in this
version of MACROS all the internal values are held as strings.

Also, the writing of string constants is shown—text
between quotes. A quote in the string is represented by
2 quotes, e.g. "FINNEGAR'S WAKE". There is also the
string constant of zero length, or the "null string," repre-
sented by "". This is used when an occurrence of a processor
variable is to be completely removed from the text, as will
be done for the single precision version of MDISCR.

For a single precision DISCRM, use the following:
## Example J — use of replacement to define function type

<table>
<thead>
<tr>
<th>STMT</th>
<th>LEVEL</th>
<th>REPLC</th>
<th>SOURCE STATEMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>353</td>
<td>0</td>
<td>0</td>
<td>% RLEN=4</td>
</tr>
<tr>
<td>354</td>
<td>0</td>
<td></td>
<td>% FPRE=''</td>
</tr>
<tr>
<td>355</td>
<td>0</td>
<td></td>
<td>CALL RDISCN</td>
</tr>
<tr>
<td>356</td>
<td>1</td>
<td>1</td>
<td>REAL FUNCTION DISCRN=6RLEN(A,B,C)</td>
</tr>
<tr>
<td>357</td>
<td>1</td>
<td>1</td>
<td>REAL FUNCTION DISCRN=A,B,C</td>
</tr>
<tr>
<td>358</td>
<td>1</td>
<td>1</td>
<td>REAL* RLEN B,A,C</td>
</tr>
<tr>
<td>359</td>
<td>1</td>
<td>1</td>
<td>DISCRN=DPRE-SQRT(B<em>B-4</em>A*C)</td>
</tr>
<tr>
<td>360</td>
<td>1</td>
<td>0</td>
<td>RETURN</td>
</tr>
<tr>
<td>361</td>
<td>1</td>
<td>0</td>
<td>END</td>
</tr>
</tbody>
</table>

doctest: >>>

---

MAC01 8SEP69
II. Example 4--macro parameters, continuation statements

A macro will be used to generate debugging statements. Using a macro will reduce the amount of writing and make it possible to completely remove the statements from the source program by simply defining the macro with no macro statements. Our debug macro will generate a NAMELIST statement and a WRITE statement:

```
MACRO DEBUG(MLEN, DBLIST)
  NAMELIST /ENLN/  &DBLIST
  ENDO
```

The MACRO statement declares two "formal parameters" which represent the NAMELIST name (MLEN) and a list of identifiers (DBLIST). The formal parameters are assigned values when the macro is expanded by using an "actual parameter list" with the CALL statement. (Terminology borrowed from ALGOL 60.) To use DEBUG, one could write:

```
***EXAMPLE 4-1****
CALL DEBUG('DB', 'A,B,C')
```

Here, the actual parameters are the two strings 'DB' (NAMELIST name), and 'A,B,C' (the identifier list).

If this macro is rewritten to contain only a MACRO and a MENU statement, the effect will be to remove the debugging statements from the program. We cannot do that here, since that would be a duplicate macro definition, which MACROS does not allow.

Another feature of MACROS that is useful when working with FORTRAN as the base language is the automatic continuation statement processing. This is available if the 'FORTN=1' option is used. (Options are detailed in section IV.) MACROS will automatically generate continuation statements if the generated statement exceeds one card, as occurs here:

```
***EXAMPLE 4-2****
CALL DEBUG('DB', 'IDEN1, IDEN2, IDEN3, IDEN4, IDEN5, IDEN6, IDEN7, FOOBAZ')
```

If the FORTN=1 option is used, the total number of characters in any FORTRAN statement in the MACROS input (before and after replacement, if any) is limited to 270 (minus trailing blanks and
Example 4—Macro parameters, continuation statements

SOURCE STATEMENT

```plaintext
REGION
1F(I)
= +
5F(I-1) +F(I-2) +F(I-1)
```

MACRO FORMAT:

```plaintext
'THIS IS A HOLLERITH LITERAL IN QUOTES'
1 'NOTICE THAT ALL BLANKS ARE RETAINED'
```

MACRO FORMAT:

```plaintext
'THIS IS A HOLLERITH LITERAL IN QUOTES'
1 'NOTICE THAT ALL BLANKS ARE RETAINED'
```

In the first example, it can be seen that all trailing blanks were removed, but leading blanks (preceding the *) were retained. In the second example, the Hollerith literals caused all blanks to be retained—this is done to avoid truncation of literals extending across card boundaries.
II. Example 5—processor variable scope & macro parameters

The previous examples used processor variables that were simple variables (i.e. not formal parameters) global to a macro, or formal parameters. In the following, more about the scope of processor variables will be illustrated using the macro MDISCRM of example 3. The term "scope" refers to the way that MACROS determines what declaration of an identifier is to be associated with a usage of that identifier—possibilities are: simple variable declared by assignment, formal parameter local to the macro being expanded, or formal parameter local to some macro that called the one currently being expanded. The first two cases have been illustrated; the last case can be shown with the following macro:

MACRO DISCPR(BLEN,FPRE)
CALL MDISCRM
MEND

We can use it to set BLEN and FPRE and call MDISCRM for a double precision DISCRM:

***EXAMPLE 5-1***

CALL DISCPR(0,'D')
REAL FUNCTION DISCRM&6BLEN(A,B,C)
REAL FUNCTION DISCRM&6BLEN(A,B,C)
REAL&BLEN B,A,C
REAL&8 B,A,C
DISCRM=6FPRE,SQRT(B+B-4*A*C)
DISCRM=6FPRE,SQRT(B+B-4*A*C)
RETURN
END

***************

When the level 2 macro MDISCRM was expanded, the current values associated with the closest occurrence of the names BLEN and FPRE was used. These were the values supplied to the formal parameters in DISCPR by the CALL statement. The macro expansion consists of the following steps:

(1) Save any values associated with the formal parameter identifiers (BLEN and FPRE).
(2) Associate (assign) the values of the corresponding actual parameters with the formal parameter identifiers.
(3) Expand the macro body in the usual fashion (process the CALL statement).
(4) When returning from this macro (DISCPR), free the values associated with the focal parameters on step 2 and restore those previously associated with the identifiers, if any.

We can see this effect by checking that the values associated with the top level occurrences of BLEN and FPRE (4 and "", respectively) have been restored by substituting them in this arbitrary statement:
II. Example 5--processor variable scope & macro parameters

<table>
<thead>
<tr>
<th>STMT</th>
<th>LEVEL</th>
<th>REPLC</th>
<th>SOURCE STATEMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>500</td>
<td>0</td>
<td>0</td>
<td>THE TOP LEVEL VALUES OF BLEN=6, LEBL=E, AND FPRE=6FPRE.</td>
</tr>
<tr>
<td>501</td>
<td>0</td>
<td>B8</td>
<td>THE TOP LEVEL VALUES OF BLEN=4, AND FPRE=.</td>
</tr>
<tr>
<td>502</td>
<td>0</td>
<td>0</td>
<td>******************************************</td>
</tr>
<tr>
<td>503</td>
<td>0</td>
<td>0</td>
<td>%: As expected, MACROS restored the top level values as claimed.</td>
</tr>
<tr>
<td>504</td>
<td>0</td>
<td>0</td>
<td>%: also, the two periods following 6FPRE show how one obtains a period</td>
</tr>
<tr>
<td>505</td>
<td>0</td>
<td>0</td>
<td>%: following an escape name. The first period is included in the</td>
</tr>
<tr>
<td>506</td>
<td>0</td>
<td>0</td>
<td>%: escape name, while the second remains in the text.</td>
</tr>
<tr>
<td>507</td>
<td>0</td>
<td>0</td>
<td>%:</td>
</tr>
<tr>
<td>508</td>
<td>0</td>
<td>0</td>
<td>%: This scope mechanism differs from that of ALGOL or PL/I</td>
</tr>
<tr>
<td>509</td>
<td>0</td>
<td>0</td>
<td>%: in that it is dynamically determined by the sequence of macro</td>
</tr>
<tr>
<td>510</td>
<td>0</td>
<td>0</td>
<td>%: calls rather than from explicit nesting when the program is</td>
</tr>
<tr>
<td>511</td>
<td>0</td>
<td>0</td>
<td>%: written (lexical scope). (It is similar to that used in</td>
</tr>
<tr>
<td>512</td>
<td>0</td>
<td>0</td>
<td>%: Iverson language.)</td>
</tr>
</tbody>
</table>

MAC01 8SEP69
II. Example 6 -- use of MACROS with the SUMX analysis program

This example will illustrate the usage of MACROS with a base language other than FORTRAN. MACROS can be used to simplify preparation of a set of control statements to the SUMX data analysis program. The features of MACROS illustrated separately in the previous examples will be combined here. In addition, another mode of replacement will be introduced that is useful for generating statements that have fixed format requirements, as do SUMX control statements.

For those not familiar with SUMX, it can be briefly described as a general processor providing facilities for a user to perform statistical analysis of large quantities of data, such as that encountered in high energy physics. By means of an elaborate set of control statements, the user defines "tests" which can be referenced in the data collection and display sections of SUMX to determine whether or not a data sample is to be included in the display (histogram, scatter plot, etc.). The tests are Boolean expressions that describe some relation on an input record or on additional quantities computed by the user from data in the input record. If the test(s) are satisfied (or not satisfied, as desired), then SUMX performs some action such as including data from the input record in the display.

In the example at hand, control statements to direct SUMX to generate a set of histograms that display characteristics of a class of inter-particle reaction (three-prong event -- photon + proton -> pi+pi-proton). Control statements will be needed to describe format of the data records, criteria for including a data record in a display (the input contains events other than the type of interest), actions performed by a user-supplied routine that computes additional quantities from the input ("CHARM" routines), and parameters used for making the histograms (number of bins, bin width, etc).

The amount of writing, thus margin for error, is reduced by using macro to generate sequences of control statements. By maintaining a macro library of commonly used sequences as macros, the user can reduce the amount of card manipulation for equivalent using a text editor. Most powerful is the replacement facility which allows writing of general control statement sequences which can be adapted to an application by setting global variables and macro parameters.

This example will be presented without great concern for the details of the control statements or the physics involved; emphasis will be on the usage of MACROS. First, several macros will be defined. These would probably be kept in a macro library for general use. Then the MACROS input will be given, interspersed with commentary describing it.

This macro will define some tests needed for determining the class of events:

MACRO T33

TEST 2 -4 EQU 30001
II. Example 6—use of MACROS with the SUMX analysis program

SOURCE STATEMENT

MACRO SYMBOL

MACRO INTEGRAL, MASS, LOC, WGT

MACRO MASS(SYS, ML, LOC, WGT)

MACRO INTEGRAL MASS OF SYS

The assignment statement for DM uses a fixed point number, which is accepted by MACROS as if it were enclosed in quotes—syntactically, it is a string. This is done for convenience in writing fixed point numbers, which is a common occurrence for SUMX. The next macro is used to generate the sequence of control statements for a histogram:

END

END

END

END

END

END

END

END

END

END

END

END

END

END

END

END

END

END
II. Example 6--use of MACROS with the SUMX analysis program

<table>
<thead>
<tr>
<th>STATEMENT</th>
<th>LEVEL</th>
<th>REPLC</th>
<th>SOURCE STATEMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>111</td>
<td>0</td>
<td>0</td>
<td>*DISCARD 0</td>
</tr>
<tr>
<td>112</td>
<td>0</td>
<td>0</td>
<td>*LIST 8,9</td>
</tr>
<tr>
<td>113</td>
<td>0</td>
<td>0</td>
<td>%</td>
</tr>
<tr>
<td>114</td>
<td>0</td>
<td>0</td>
<td>% These supply page headings and indicate general options.</td>
</tr>
<tr>
<td>115</td>
<td>0</td>
<td>0</td>
<td>%</td>
</tr>
<tr>
<td>116</td>
<td>0</td>
<td>0</td>
<td>% New some tests will be defined:</td>
</tr>
<tr>
<td>117</td>
<td>0</td>
<td>0</td>
<td>%</td>
</tr>
<tr>
<td>118</td>
<td>0</td>
<td>0</td>
<td>% SELECT</td>
</tr>
<tr>
<td>119</td>
<td>1</td>
<td>0</td>
<td>% CALL T33 ; BRING IN THE STANDARD TESTS</td>
</tr>
<tr>
<td>120</td>
<td>1</td>
<td>0</td>
<td>TEST 2</td>
</tr>
<tr>
<td>121</td>
<td>1</td>
<td>0</td>
<td>-4 EQU 30001</td>
</tr>
<tr>
<td>122</td>
<td>1</td>
<td>0</td>
<td>TEST 3</td>
</tr>
<tr>
<td>123</td>
<td>1</td>
<td>0</td>
<td>-4 EQU -30001</td>
</tr>
<tr>
<td>124</td>
<td>1</td>
<td>0</td>
<td>TEST 4</td>
</tr>
<tr>
<td>125</td>
<td>1</td>
<td>0</td>
<td>2 TRUE</td>
</tr>
<tr>
<td>126</td>
<td>1</td>
<td>0</td>
<td>3 TRUE</td>
</tr>
<tr>
<td>127</td>
<td>0</td>
<td>0</td>
<td>% ADD A SPECIAL TEST FOR THIS RUN TO SELECT 3-PROM EVENTS</td>
</tr>
<tr>
<td>128</td>
<td>0</td>
<td>0</td>
<td>%</td>
</tr>
<tr>
<td>129</td>
<td>1</td>
<td>0</td>
<td>TEST 1</td>
</tr>
<tr>
<td>130</td>
<td>1</td>
<td>0</td>
<td>-2 EQU 300</td>
</tr>
<tr>
<td>131</td>
<td>0</td>
<td>0</td>
<td>%</td>
</tr>
<tr>
<td>132</td>
<td>0</td>
<td>0</td>
<td>%</td>
</tr>
<tr>
<td>133</td>
<td>0</td>
<td>0</td>
<td>%</td>
</tr>
<tr>
<td>134</td>
<td>0</td>
<td>0</td>
<td>% Next, information for the user routines which compute additional</td>
</tr>
<tr>
<td>135</td>
<td>0</td>
<td>0</td>
<td>% quantities from the input is given. This set of assignment</td>
</tr>
<tr>
<td>136</td>
<td>0</td>
<td>0</td>
<td>% statements define processor variables that specify the locations</td>
</tr>
<tr>
<td>137</td>
<td>0</td>
<td>0</td>
<td>% of data used by the user routines:</td>
</tr>
<tr>
<td>138</td>
<td>0</td>
<td>0</td>
<td>%</td>
</tr>
<tr>
<td>139</td>
<td>0</td>
<td>0</td>
<td>% FVB=701</td>
</tr>
<tr>
<td>140</td>
<td>0</td>
<td>0</td>
<td>% M3=721</td>
</tr>
<tr>
<td>141</td>
<td>0</td>
<td>0</td>
<td>% M35=722</td>
</tr>
<tr>
<td>142</td>
<td>0</td>
<td>0</td>
<td>% M35L=1.08</td>
</tr>
<tr>
<td>143</td>
<td>0</td>
<td>0</td>
<td>% M5=723</td>
</tr>
<tr>
<td>144</td>
<td>0</td>
<td>0</td>
<td>% M5L=2.26</td>
</tr>
<tr>
<td>145</td>
<td>0</td>
<td>0</td>
<td>% C34=724</td>
</tr>
<tr>
<td>146</td>
<td>0</td>
<td>0</td>
<td>% C35=727</td>
</tr>
<tr>
<td>147</td>
<td>0</td>
<td>0</td>
<td>% C45=730</td>
</tr>
<tr>
<td>148</td>
<td>0</td>
<td>0</td>
<td>%</td>
</tr>
<tr>
<td>149</td>
<td>0</td>
<td>0</td>
<td>% These variables will be used to generate parameter cards for the</td>
</tr>
<tr>
<td>150</td>
<td>0</td>
<td>0</td>
<td>% user routines. CHARM 8 is used to set up the bank of four</td>
</tr>
<tr>
<td>151</td>
<td>0</td>
<td>0</td>
<td>% vectors (starting from symbolic Constant location 'FVB'), from which</td>
</tr>
<tr>
<td>152</td>
<td>0</td>
<td>0</td>
<td>% all quantities of interest are computed: CHARM 2 for invariant</td>
</tr>
<tr>
<td>153</td>
<td>0</td>
<td>0</td>
<td>% mass of the system, CHARM 3 for production cosines.</td>
</tr>
<tr>
<td>154</td>
<td>0</td>
<td>0</td>
<td>%</td>
</tr>
<tr>
<td>155</td>
<td>0</td>
<td>0</td>
<td>% CHARM</td>
</tr>
<tr>
<td>156</td>
<td>0</td>
<td>0</td>
<td>BR 3F3C 8 66FVB</td>
</tr>
<tr>
<td>157</td>
<td>0</td>
<td>0</td>
<td>BR M35 2 66F35  2 35 0 66FVB</td>
</tr>
<tr>
<td>158</td>
<td>0</td>
<td>0</td>
<td>BR M45 2 66F45  2 45 0 66FVB</td>
</tr>
<tr>
<td>159</td>
<td>0</td>
<td>0</td>
<td>BR C35 3 66C35  1 3 2 66FVB</td>
</tr>
<tr>
<td>160</td>
<td>0</td>
<td>0</td>
<td>%</td>
</tr>
<tr>
<td>161</td>
<td>0</td>
<td>0</td>
<td>% A macro that generates test statements to describe energy</td>
</tr>
<tr>
<td>162</td>
<td>0</td>
<td>0</td>
<td>% intervals is defined:</td>
</tr>
</tbody>
</table>
II. Example 6—use of MACROS with the SUX analysis program

MACRO EINTY(RH,ELOW, EHIGH)
TEST ELEN
GEVF B ELELW ELEHIG
% END

%SELECT
% CALL EINTY(20,2.5,3.5)
TEST ELEN
TEST 20
GEVF BET ELLOW ELEHIG
701 BET 2.5 3.5
% CALL EINTY(21,2.5,4.5)
TEST ELEN
TEST 21
GEVF B ELELW ELEHIG
701 BET 3.5 4.5
% THIS TEST IS USED FOR CONDITIONAL GENERATION OF HISTOGRAMS
% TEST 30
% BIG .9
% 300 BIG .9
% Now the control statements for the histograms themselves will be
given. First, the macro SYMBOL will be called to define
% default values of processor variables:
% SYMBOL
% Then a macro that defines a group of three histograms:
% MACRO HISTGROUP
% CALL MASS('M35','M35L,M35')
% CALL MASS('M45','M45L,M45')
% CALL MASS('M45 WEIGHTED','M45L,M45,10')
% END
% This macro makes use of the default value of "\" assigned to
% omitted macro parameters in the CALL MASS statements. Finally,
% a macro that uses HISTGROUP is defined such that histograms
% for various energy intervals may be generated:
% MACRO EINTTEST(EINTTEST)
% IF TEST EINTTEST SATISFIED, DO UP TO THE 'FINISH 1'
% CALL HISTGROUP
% IF TEST 40 SATISFIED, DO UP TO NEXT 'FINISH 2'
% CALL HISTGROUP
% GENERAL TEST #1 IS THE COMPLEMENT OF TEST N
% CALL HISTGROUP
% FINISH 2
% CALL HISTGROUP
% FINISH 2
II. Example 6--use of MACROS with the SUMX analysis program

<table>
<thead>
<tr>
<th>STMT</th>
<th>LEVEL</th>
<th>REPLC</th>
<th>SOURCE STATEMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>223</td>
<td>3</td>
<td></td>
<td>FINISH 1</td>
</tr>
<tr>
<td>224</td>
<td>3</td>
<td></td>
<td>% REND</td>
</tr>
<tr>
<td>225</td>
<td>0</td>
<td></td>
<td>%</td>
</tr>
<tr>
<td>226</td>
<td>0</td>
<td></td>
<td>%; Now, by writing 2 MACROS statements, we can generate SUMX</td>
</tr>
<tr>
<td>227</td>
<td>0</td>
<td></td>
<td>%; statements for 18 histograms, or a total of 54 SUMX control</td>
</tr>
<tr>
<td>228</td>
<td>0</td>
<td></td>
<td>%;</td>
</tr>
<tr>
<td>229</td>
<td>0</td>
<td></td>
<td>%</td>
</tr>
<tr>
<td>230</td>
<td>0</td>
<td></td>
<td>%</td>
</tr>
<tr>
<td>231</td>
<td>0</td>
<td>0</td>
<td>EBLOCK6</td>
</tr>
<tr>
<td>232</td>
<td>0</td>
<td></td>
<td>%; *****<em><strong><strong><strong><strong><strong><strong>HISTOGRAMS FOR THE FIRST ENERGY INTERVAL</strong></strong></strong></strong></strong></strong></em></td>
</tr>
<tr>
<td>233</td>
<td>0</td>
<td></td>
<td>%; CALL EINTHIST(20)</td>
</tr>
<tr>
<td>234</td>
<td>1</td>
<td>BR</td>
<td>EVA E6EINTTEST</td>
</tr>
<tr>
<td>235</td>
<td>1</td>
<td>1</td>
<td>EVA 20</td>
</tr>
<tr>
<td>236</td>
<td>3</td>
<td>BB</td>
<td>INVARINT MASS OF E6YS</td>
</tr>
<tr>
<td>237</td>
<td>3</td>
<td>1</td>
<td>INVARINT MASS OF M35</td>
</tr>
<tr>
<td>238</td>
<td>3</td>
<td>BB</td>
<td>E6MN E6DM E6ML E6NPT E6EPAC</td>
</tr>
<tr>
<td>239</td>
<td>3</td>
<td>6</td>
<td>80 .04 1.00</td>
</tr>
<tr>
<td>240</td>
<td>3</td>
<td>BB</td>
<td>E6LOC E6GWGT E6MT E6SGM</td>
</tr>
<tr>
<td>241</td>
<td>3</td>
<td>4</td>
<td>723</td>
</tr>
<tr>
<td>242</td>
<td>3</td>
<td>BB</td>
<td>INVARINT MASS OF E6YS</td>
</tr>
<tr>
<td>243</td>
<td>3</td>
<td>1</td>
<td>INVARINT MASS OF M45</td>
</tr>
<tr>
<td>244</td>
<td>3</td>
<td>BB</td>
<td>E6NM E6DM E6ML E6NPT E6EPAC</td>
</tr>
<tr>
<td>245</td>
<td>3</td>
<td>6</td>
<td>80 .04 .26</td>
</tr>
<tr>
<td>246</td>
<td>3</td>
<td>BR</td>
<td>E6LOC E6GWGT E6MT E6SGM</td>
</tr>
<tr>
<td>247</td>
<td>3</td>
<td>4</td>
<td>723</td>
</tr>
<tr>
<td>248</td>
<td>3</td>
<td>BB</td>
<td>INVARINT MASS OF E6YS</td>
</tr>
<tr>
<td>249</td>
<td>3</td>
<td>1</td>
<td>INVARINT MASS OF M45 WEIGHTED</td>
</tr>
<tr>
<td>250</td>
<td>3</td>
<td>BB</td>
<td>E6NM E6DM E6ML E6NPT E6EPAC</td>
</tr>
<tr>
<td>251</td>
<td>3</td>
<td>6</td>
<td>80 .04 1.00</td>
</tr>
<tr>
<td>252</td>
<td>3</td>
<td>BR</td>
<td>E6LOC E6GWGT E6MT E6SGM</td>
</tr>
<tr>
<td>253</td>
<td>3</td>
<td>4</td>
<td>723</td>
</tr>
<tr>
<td>254</td>
<td>1</td>
<td>0</td>
<td>EVA 40</td>
</tr>
<tr>
<td>255</td>
<td>3</td>
<td>BB</td>
<td>INVARINT MASS OF E6YS</td>
</tr>
<tr>
<td>256</td>
<td>3</td>
<td>1</td>
<td>INVARINT MASS OF M35</td>
</tr>
<tr>
<td>257</td>
<td>3</td>
<td>BB</td>
<td>E6NM E6DM E6ML E6NPT E6EPAC</td>
</tr>
<tr>
<td>258</td>
<td>3</td>
<td>6</td>
<td>80 .04 1.00</td>
</tr>
<tr>
<td>259</td>
<td>3</td>
<td>BB</td>
<td>E6LOC E6GWGT E6MT E6SGM</td>
</tr>
<tr>
<td>260</td>
<td>3</td>
<td>4</td>
<td>723</td>
</tr>
<tr>
<td>261</td>
<td>3</td>
<td>BB</td>
<td>INVARINT MASS OF E6YS</td>
</tr>
<tr>
<td>262</td>
<td>3</td>
<td>1</td>
<td>INVARINT MASS OF M45</td>
</tr>
<tr>
<td>263</td>
<td>3</td>
<td>BB</td>
<td>E6NM E6DM E6ML E6NPT E6EPAC</td>
</tr>
<tr>
<td>264</td>
<td>3</td>
<td>6</td>
<td>80 .04 .26</td>
</tr>
<tr>
<td>265</td>
<td>3</td>
<td>BB</td>
<td>INVARINT MASS OF E6YS</td>
</tr>
<tr>
<td>266</td>
<td>3</td>
<td>1</td>
<td>INVARINT MASS OF M45 WEIGHTED</td>
</tr>
<tr>
<td>267</td>
<td>3</td>
<td>BB</td>
<td>E6NM E6DM E6ML E6NPT E6EPAC</td>
</tr>
<tr>
<td>268</td>
<td>3</td>
<td>6</td>
<td>80 .04 .26</td>
</tr>
<tr>
<td>269</td>
<td>3</td>
<td>BB</td>
<td>E6LOC E6GWGT E6MT E6SGM</td>
</tr>
<tr>
<td>270</td>
<td>1</td>
<td>0</td>
<td>FINISH 2</td>
</tr>
<tr>
<td>271</td>
<td>1</td>
<td>0</td>
<td>EVA 41</td>
</tr>
<tr>
<td>272</td>
<td>3</td>
<td>BB</td>
<td>INVARINT MASS OF E6YS</td>
</tr>
<tr>
<td>273</td>
<td>3</td>
<td>1</td>
<td>INVARINT MASS OF M35</td>
</tr>
<tr>
<td>274</td>
<td>3</td>
<td>BB</td>
<td>E6NM E6DM E6ML E6NPT E6EPAC</td>
</tr>
</tbody>
</table>

PAGE  19 MAC01 8SEP69
## II. Example 6: Use of MACROS with the SUMX Analysis Program

<table>
<thead>
<tr>
<th>STMT</th>
<th>LEVEL</th>
<th>REPLC</th>
<th>SOURCE STATEMENT</th>
<th>MACRO</th>
<th>DATE</th>
</tr>
</thead>
<tbody>
<tr>
<td>275</td>
<td>3</td>
<td>6</td>
<td>80 ELOC E6WGT E6NT E6SM</td>
<td></td>
<td></td>
</tr>
<tr>
<td>276</td>
<td>3</td>
<td>BR</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>277</td>
<td>3</td>
<td>4</td>
<td>722</td>
<td></td>
<td></td>
</tr>
<tr>
<td>278</td>
<td>3</td>
<td>BR</td>
<td>INARIANT MASS OF E6YS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>279</td>
<td>3</td>
<td>1</td>
<td>INARIANT MASS OF M45</td>
<td></td>
<td></td>
</tr>
<tr>
<td>280</td>
<td>3</td>
<td>BR</td>
<td>E6NM E6DM E6ML E6NPT E6MFAC</td>
<td></td>
<td>E6LOG</td>
</tr>
<tr>
<td>281</td>
<td>3</td>
<td>6</td>
<td>80 .04 .26</td>
<td></td>
<td></td>
</tr>
<tr>
<td>282</td>
<td>3</td>
<td>BR</td>
<td>E6LOC E6WGT E6NT E6SM</td>
<td></td>
<td></td>
</tr>
<tr>
<td>283</td>
<td>3</td>
<td>4</td>
<td>723</td>
<td></td>
<td></td>
</tr>
<tr>
<td>284</td>
<td>3</td>
<td>BR</td>
<td>INARIANT MASS OF M45 WEIGHTED</td>
<td></td>
<td></td>
</tr>
<tr>
<td>285</td>
<td>3</td>
<td>BR</td>
<td>E6NM E6DM E6ML E6NPT E6MFAC</td>
<td></td>
<td>E6LOG</td>
</tr>
<tr>
<td>286</td>
<td>3</td>
<td>6</td>
<td>80 .04 .26</td>
<td></td>
<td></td>
</tr>
<tr>
<td>287</td>
<td>3</td>
<td>BR</td>
<td>E6LOC E6WGT E6NT E6SM</td>
<td></td>
<td></td>
</tr>
<tr>
<td>288</td>
<td>3</td>
<td>4</td>
<td>723</td>
<td></td>
<td>10</td>
</tr>
<tr>
<td>289</td>
<td>1</td>
<td>0</td>
<td>FINISH 2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>290</td>
<td>1</td>
<td>0</td>
<td>FINISH 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>291</td>
<td>0</td>
<td>%</td>
<td>$<strong><strong><strong><strong><strong>HISTOGRAMS FOR THE SECOND ENERGY INTERVAL</strong></strong></strong></strong></strong>$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>292</td>
<td>0</td>
<td>%</td>
<td>CALL EINTHIST(21)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>293</td>
<td>1</td>
<td>BR</td>
<td>EYA E6EINTTST</td>
<td></td>
<td></td>
</tr>
<tr>
<td>294</td>
<td>1</td>
<td>1</td>
<td>EYA 21</td>
<td></td>
<td></td>
</tr>
<tr>
<td>295</td>
<td>3</td>
<td>BR</td>
<td>INARIANT MASS OF E6YS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>296</td>
<td>3</td>
<td>1</td>
<td>INARIANT MASS OF M35</td>
<td></td>
<td></td>
</tr>
<tr>
<td>297</td>
<td>3</td>
<td>BR</td>
<td>E6NM E6DM E6ML E6NPT E6MFAC</td>
<td></td>
<td>E6LOG</td>
</tr>
<tr>
<td>298</td>
<td>3</td>
<td>6</td>
<td>80 .04 .26</td>
<td></td>
<td></td>
</tr>
<tr>
<td>300</td>
<td>3</td>
<td>BR</td>
<td>E6LOC E6WGT E6NT E6SM</td>
<td></td>
<td></td>
</tr>
<tr>
<td>301</td>
<td>3</td>
<td>4</td>
<td>723</td>
<td></td>
<td></td>
</tr>
<tr>
<td>302</td>
<td>3</td>
<td>BR</td>
<td>INARIANT MASS OF E6YS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>303</td>
<td>3</td>
<td>1</td>
<td>INARIANT MASS OF M45</td>
<td></td>
<td></td>
</tr>
<tr>
<td>304</td>
<td>3</td>
<td>BR</td>
<td>E6NM E6DM E6ML E6NPT E6MFAC</td>
<td></td>
<td>E6LOG</td>
</tr>
<tr>
<td>305</td>
<td>3</td>
<td>6</td>
<td>80 .04 .26</td>
<td></td>
<td></td>
</tr>
<tr>
<td>306</td>
<td>3</td>
<td>BR</td>
<td>E6LOC E6WGT E6NT E6SM</td>
<td></td>
<td></td>
</tr>
<tr>
<td>307</td>
<td>3</td>
<td>4</td>
<td>723</td>
<td></td>
<td></td>
</tr>
<tr>
<td>308</td>
<td>3</td>
<td>BR</td>
<td>INARIANT MASS OF E6YS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>309</td>
<td>3</td>
<td>1</td>
<td>INARIANT MASS OF M45 WEIGHTED</td>
<td></td>
<td>E6LOG</td>
</tr>
<tr>
<td>310</td>
<td>3</td>
<td>BR</td>
<td>E6NM E6DM E6ML E6NPT E6MFAC</td>
<td></td>
<td>E6LOG</td>
</tr>
<tr>
<td>311</td>
<td>3</td>
<td>6</td>
<td>80 .04 .26</td>
<td></td>
<td></td>
</tr>
<tr>
<td>312</td>
<td>3</td>
<td>BR</td>
<td>E6LOC E6WGT E6NT E6SM</td>
<td></td>
<td></td>
</tr>
<tr>
<td>313</td>
<td>3</td>
<td>4</td>
<td>723</td>
<td></td>
<td></td>
</tr>
<tr>
<td>314</td>
<td>3</td>
<td>BR</td>
<td>INARIANT MASS OF E6YS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>315</td>
<td>3</td>
<td>1</td>
<td>INARIANT MASS OF M45</td>
<td></td>
<td></td>
</tr>
<tr>
<td>316</td>
<td>3</td>
<td>BR</td>
<td>E6NM E6DM E6ML E6NPT E6MFAC</td>
<td></td>
<td>E6LOG</td>
</tr>
<tr>
<td>317</td>
<td>3</td>
<td>6</td>
<td>80 .04 .26</td>
<td></td>
<td></td>
</tr>
<tr>
<td>318</td>
<td>3</td>
<td>BR</td>
<td>E6LOC E6WGT E6NT E6SM</td>
<td></td>
<td></td>
</tr>
<tr>
<td>319</td>
<td>3</td>
<td>4</td>
<td>723</td>
<td></td>
<td></td>
</tr>
<tr>
<td>320</td>
<td>3</td>
<td>BR</td>
<td>INARIANT MASS OF E6YS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>321</td>
<td>3</td>
<td>1</td>
<td>INARIANT MASS OF M45</td>
<td></td>
<td></td>
</tr>
<tr>
<td>322</td>
<td>3</td>
<td>BR</td>
<td>E6NM E6DM E6ML E6NPT E6MFAC</td>
<td></td>
<td>E6LOG</td>
</tr>
<tr>
<td>323</td>
<td>3</td>
<td>6</td>
<td>80 .04 .26</td>
<td></td>
<td></td>
</tr>
<tr>
<td>324</td>
<td>3</td>
<td>BR</td>
<td>E6LOC E6WGT E6NT E6SM</td>
<td></td>
<td></td>
</tr>
<tr>
<td>325</td>
<td>3</td>
<td>4</td>
<td>723</td>
<td></td>
<td></td>
</tr>
<tr>
<td>326</td>
<td>3</td>
<td>BR</td>
<td>INARIANT MASS OF M45 WEIGHTED</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
**II. Example 6—use of MACROS with the SUMX analysis program**

<table>
<thead>
<tr>
<th>STMT</th>
<th>LEVEL</th>
<th>REPLC</th>
<th>SOURCE STATEMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>327</td>
<td>3</td>
<td>BR</td>
<td>$\text{EWM}$ $\text{EDM}$ $\text{EML}$ $\text{ENPT}$ $\text{ENFAC}$ $\text{EFFOC}$</td>
</tr>
<tr>
<td>327</td>
<td>3</td>
<td>6</td>
<td>80 $\cdot08$ $\cdot26$</td>
</tr>
<tr>
<td>328</td>
<td>3</td>
<td>BR</td>
<td>$\text{ELOC}$ $\text{ENWGT}$ $\text{ENWT}$ $\text{ESGM}$</td>
</tr>
<tr>
<td>328</td>
<td>3</td>
<td>8</td>
<td>$\cdot723$ $\cdot10$</td>
</tr>
<tr>
<td>331</td>
<td>1</td>
<td>0</td>
<td>$\text{FINISH}$ $\cdot2$</td>
</tr>
<tr>
<td>332</td>
<td>1</td>
<td>0</td>
<td>$\text{EVA}$ $\cdot41$</td>
</tr>
<tr>
<td>335</td>
<td>3</td>
<td>BR</td>
<td>$\text{INVARIANT MASS OF ETS}$</td>
</tr>
<tr>
<td>336</td>
<td>3</td>
<td>1</td>
<td>$\text{INVARIANT MASS OF MTS}$</td>
</tr>
<tr>
<td>336</td>
<td>3</td>
<td>BR</td>
<td>$\text{ECM}$ $\text{ECDM}$ $\text{ECLR}$ $\text{ENPT}$ $\text{ENFAC}$ $\text{EFFOC}$</td>
</tr>
<tr>
<td>336</td>
<td>3</td>
<td>6</td>
<td>80 $\cdot08$ $\cdot108$</td>
</tr>
<tr>
<td>337</td>
<td>3</td>
<td>BR</td>
<td>$\text{ELOC}$ $\text{ENWGT}$ $\text{ENWT}$ $\text{ESGM}$</td>
</tr>
<tr>
<td>337</td>
<td>3</td>
<td>4</td>
<td>$\cdot722$</td>
</tr>
<tr>
<td>340</td>
<td>3</td>
<td>BR</td>
<td>$\text{INVARIANT MASS OF ETS}$</td>
</tr>
<tr>
<td>340</td>
<td>3</td>
<td>1</td>
<td>$\text{INVARIANT MASS OF MTS}$</td>
</tr>
<tr>
<td>341</td>
<td>3</td>
<td>BR</td>
<td>$\text{ECM}$ $\text{ECDM}$ $\text{ECLR}$ $\text{ENPT}$ $\text{ENFAC}$ $\text{EFFOC}$</td>
</tr>
<tr>
<td>341</td>
<td>3</td>
<td>6</td>
<td>80 $\cdot08$ $\cdot26$</td>
</tr>
<tr>
<td>342</td>
<td>3</td>
<td>BR</td>
<td>$\text{ELOC}$ $\text{ENWGT}$ $\text{ENWT}$ $\text{ESGM}$</td>
</tr>
<tr>
<td>342</td>
<td>3</td>
<td>4</td>
<td>$\cdot723$</td>
</tr>
<tr>
<td>345</td>
<td>3</td>
<td>BR</td>
<td>$\text{INVARIANT MASS OF ETS}$</td>
</tr>
<tr>
<td>345</td>
<td>3</td>
<td>1</td>
<td>$\text{INVARIANT MASS OF MTS WEIGHTED}$ $\text{ENPT}$ $\text{ENFAC}$ $\text{EFFOC}$</td>
</tr>
<tr>
<td>346</td>
<td>3</td>
<td>BR</td>
<td>$\text{ECM}$ $\text{ECDM}$ $\text{ECLR}$ $\text{ENPT}$ $\text{ENFAC}$ $\text{EFFOC}$</td>
</tr>
<tr>
<td>346</td>
<td>3</td>
<td>6</td>
<td>80 $\cdot08$ $\cdot26$</td>
</tr>
<tr>
<td>347</td>
<td>3</td>
<td>BR</td>
<td>$\text{ELOC}$ $\text{ENWGT}$ $\text{ENWT}$ $\text{ESGM}$</td>
</tr>
<tr>
<td>347</td>
<td>3</td>
<td>4</td>
<td>$\cdot723$ $\cdot10$</td>
</tr>
<tr>
<td>350</td>
<td>1</td>
<td>0</td>
<td>$\text{FINISH}$ $\cdot2$</td>
</tr>
<tr>
<td>351</td>
<td>1</td>
<td>0</td>
<td>$\text{FINISH}$ $\cdot1$</td>
</tr>
<tr>
<td>353</td>
<td>0</td>
<td>0</td>
<td>$%$</td>
</tr>
<tr>
<td>354</td>
<td>0</td>
<td>0</td>
<td>$%$</td>
</tr>
</tbody>
</table>
| 355  | 0     | 0     | $\%$ | And lastly, we need two more SUMX statements:
| 356  | 0     | 0     | $\%$ | $\%$ |
| 357  | 0     | 0     | $\%$ | $\%$ | $\%$ | $\%$ | $\%$ | $\%$ | $\%$ | $\%$ | $\%$ | $\%$ | $\%$ | $\%$ |
| 358  | 0     | 0     | $\%$ | $\%$ | $\%$ | $\%$ | $\%$ | $\%$ | $\%$ | $\%$ | $\%$ | $\%$ | $\%$ | $\%$ | $\%$ | $\%$ |
| 359  | 0     | 0     | $\%$ | $\%$ | $\%$ | $\%$ | $\%$ | $\%$ | $\%$ | $\%$ | $\%$ | $\%$ | $\%$ | $\%$ | $\%$ | $\%$ |
| 360  | 0     | 0     | $\%$ | $\%$ | $\%$ | $\%$ | $\%$ | $\%$ | $\%$ | $\%$ | $\%$ | $\%$ | $\%$ | $\%$ | $\%$ | $\%$ |
| 361  | 0     | 0     | $\%$ | $\%$ | $\%$ | $\%$ | $\%$ | $\%$ | $\%$ | $\%$ | $\%$ | $\%$ | $\%$ | $\%$ | $\%$ | $\%$ |
| 362  | 0     | 0     | $\%$ | $\%$ | $\%$ | $\%$ | $\%$ | $\%$ | $\%$ | $\%$ | $\%$ | $\%$ | $\%$ | $\%$ | $\%$ | $\%$ |
| 363  | 0     | 0     | $\%$ | $\%$ | $\%$ | $\%$ | $\%$ | $\%$ | $\%$ | $\%$ | $\%$ | $\%$ | $\%$ | $\%$ | $\%$ | $\%$ |

**SUMX Control Statement Syntax:**

III. MACHOS Language Description

SOURCE STATEMENT

This section will describe the elements of the MACHOS language in detail. The examples of preceding sections were of an introductory nature, while this section is designed for reference purposes.

The syntax notation uses names of items enclosed in < > symbols to denote syntactic entities. Definitions of these are either given in words or in terms of other entities. This is indicated by an <item> followed by ::= and definitions. The symbol | is used to indicate alternates.

1. ELEMENTS

<identifier> ::= 1-16 alphanumeric characters, the first of which must be alphabetic. Letters are A-Z, a-z; digits are 0-9. Identifiers are used to denote processor variables and macros.

<integer> ::= 1-9 digits. Its usage in MACHOS is identical to strings, since this version has no macro-time arithmetic.

<string> ::= there are two forms:

1) Text string-- 0-80 characters enclosed in quotes.

2) Fixed point number--string of digits preceded by, or followed by, or containing a decimal point. This construct is included for convenience in writing MACROS statements; but it is not a number and does not possess a numeric value. It should be thought of as if it were enclosed in quotes.

<constant> ::= <integer> | <string>

<formal parameter> ::= <identifier>

The formal parameter is declared by appearance in a MACRO statement. Its value is defined when the macro is expanded--rules are given below.

<simple variable> ::= <identifier>

Simple variables are declared and assigned values by appearance on the left side of an assignment statement. They assume the type of the right side of the statement--in this version of MACHOS it can always be considered string type since no macro-time arithmetic is permitted.

<processor variable> ::= <simple variable> | <formal parameter>

<processor expression> ::= <constant> | <processor variable>

The processor expression is used on the right side of assignment statements or in actual parameters of CALL statements. Its value is either the string value of the constant or the value currently associated with the processor variable. If the variable has no value associated with it, then the expression is erroneous.
III. MACROs Language Description

SOURCE STATEMENT

and its value is taken to be the null string.

Examples:

<Identifier> -- ABCD $2 OTHER1

<String> -- 'TEXT STRING' 'PEOPLE'S PARK'

.123 12.3 456. (Fixed point number)

<processor expression> -- ABCD 123 .9 'STRING'

II. PROCESSOR STATEMENTS

General:

Processor statements always have the character '*' in column 1. The body of the statement must be in columns 2-72.

No continuation statements are allowed. With the exception of
assignment and null statements, they begin with a reserved word
followed by an operand field. Comments may follow the
logical end of the statement when separated by a semi-colon
(\%). Blanks may be used freely between elements and keywords,
they are only required to separate alphanumeric items.

Processor statements are not subject to replacement (section
IV) or may they be generated by replacement. Thus if a '*' is
generated by replacement in column 1, MACROS will not recognize
the statement as a processor statement.

Processor statements are always printed at top level and
while a macro is being edited. They are never printed at lower
levels (i.e., within a macro expansion).

Null or comment statement--

Form: *: comments

The null statement is ignored by MACROS. When in a macro,
it is not encoded, so no space is required for it in the
edited macro.

Example:

*: The report you are reading is mostly null statements

MACRO statement

Form: * MACRO <macroname> <(formal parameter list)> or

* MACRO <macroname>

* <macroname> := <Identifier>

* <(formal parameter list) := <(formal parameter)> | <(formal parameter list), <(formal parameter)>

The MACRO statement heads a macro definition. The macroname
is used in CALL statements to reference the macro. It may
be the same as a processor variable identifier. No two macros
may have the same name.

The formal parameter list, if supplied declares the
identifiers as formal parameters. A maximum of ten may be given.
Details of their interpretation are in the next section.

Macros may not be nested. The macro definition ends at

MACRO statement, the next MACRO statement, or end-of-file.

Examples:

* MACRO FIBONACCI(I)

* MACRO $DECLARATIONS
III. MACROS Language Description

SOURCE STATEMENT

MACRO FUNCTION (TYP, ARG)

REND statement

Form: % REND

The REND statement terminates a macro definition that
has not been otherwise ended.

CALL Statement

Form: % CALL <macroname> (<actual parameter list>)

or % CALL <macroname>

<actual parameter list> ::= <actual parameter> | <actual parameter list> , <actual parameter>

<actual parameter> ::= <processor expression>

The CALL statement invokes the macro <macroname>,
which must have been previously defined or be available on file
SYSLIB.

The actual parameter list if present is used to assign
values to the formal parameters. Details are given in the next
section.

Examples:

% CALL PIROMAGI (1)
% CALL DECLARATIONS
% CALL FUNCTION ( 'SIN', 'X+Y' )

TITLE statement

Form: % TITLE <string>

or % TITLE

The TITLE statement sets the page title to <string> if
present and causes a page eject. Omitted operand leaves the
title unchanged and causes a page eject only. The statement
is not processed within a macro definition—only when it is
expanded. The TITLE statement is never printed.

Example:

% TITLE 'III. MACROS Language Description'

The above was used to get the current page title.

Assignment statement

Form: % <leftside> = <processor expression>

or % <leftside> ::= <processor variable>

The assignment statement is used to set the value of the
leftside to the value of the processor expression. If the
leftside is a processor variable, then the statement also
declares the identifier as such if it has not been used
before. If it is a formal parameter, then the assignment
is only retained during this expansion of the macro. The
actual parameter, if it was a processor variable, will not
be changed by the assignment.

III. MACRO EXPANSIONS

When a macro is expanded using the CALL statement, the
following actions take place:

(1) If any formal parameters were declared, any values currently
    assigned to the formal parameter identifiers are saved on a
III. MACROS Language Description

SOURCE STATEMENT

MAC01 8SEP69

529 0 5:
pushdown stack.

530 0 5:
(2) The values of the actual parameters supplied in the
531 0 5:
CALL statement are assigned to the formal parameter
532 0 5:
identifiers. If any or all of the actual parameters are
533 0 5:
omitted, then the corresponding formal parameters are
534 0 5:
not to the null string. Excess actual parameters are
535 0 5:
ignored. Thus a call statement of the form
536 0 5:
% CALL FUNCTION (.3)
537 0 5:
for the macro FUNCTION above, will set TFP="1" and ARG="3".
538 0 5:
(3) MACROS begins fetching input from the body of the macro
539 0 5:
definition. The processor variables that are currently
540 0 5:
active will be used during processing—this includes
541 0 5:
any variables set by macros that called this one.
542 0 5:
(4) When the end of the macro definition is reached, the
543 0 5:
values previously associated with the formal parameter
544 0 5:
identifiers are restored with the values saved on the
545 0 5:
pushdown stack, if any. MACROS then resumes fetching
546 0 5:
statements in the environment of the invoking CALL
547 0 5:
statement; but changes made to variables global to the
548 0 5:
called macro are retained.
549 0 5:
The maximum depth of macro calls is 10 levels.
550 0 5:

IV. REPLACEMENT IN BASE LANGUAGE STATEMENTS

551 0 5:
Every base language statement (i.e. no '*' in col 1)
552 0 5:
outside of a macro definition is scanned by MACROS for the
553 0 5:
escape character '6', which signals a potential replacement.
554 0 5:
If the '6' is followed by an identifier or by another '6' and
555 0 5:
an identifier, and if the identifier is that of a processor
556 0 5:
variable that is currently active, then a replacement is per-
557 0 5:
fomed. The single apostrophe is used to denote "packed"
558 0 5:
replacement, for free format applications; and the double
559 0 5:
apostrophes for "non-packed" replacement, for use where column
560 0 5:
positions are critical.
561 0 5:
The term "escape name" is used to refer to the escape
562 0 5:
case of packed replacement, a period delimiter is included in
563 0 5:
the escape name.
564 0 5:
Scanning for an escape character proceeds from least to
565 0 5:
right. The scanning field is the entire Fortran statement,
566 0 5:
including continuation lines, if the FORT=1 option is used;
567 0 5:
else it is the columns between the begin and end columns (in-
568 0 5:
sive) of the 80 character record. The default scanning columns
569 0 5:
are 1 thru 80; they may be changed by the user if needed.
570 0 5:
When an escape name is found, and replacement is done (the
571 0 5:
identifier has a value), then the scan is restarted from the begin
572 0 5:
column. If the identifier does not have a value, then the scan
573 0 5:
proceeds to the right. Rescanning continues until no
574 0 5:
more escape characters remain, or no more replacements can be
575 0 5:
done. The rescanning mechanism allows construction of escape
576 0 5:
names by replacement. The maximum number of replacements allowed
577 0 5:
in a base language statement is 30.
578 0 5:
Specific rules for the replacement modes are as follows:
III. MACROS Language Description

584 0 $:
585 0 $:
586 0 $:
587 0 $:
588 0 $:
589 0 $:
590 0 $:
591 0 $:
592 0 $:
593 0 $:
594 0 $:
595 0 $:
596 0 $:
597 0 $:
598 0 $:
599 0 $:
600 0 $:
601 0 $:
602 0 $:
603 0 $:
604 0 $:
605 0 $:
606 0 $:
607 0 $:
608 0 $:
609 0 $:
610 0 $:
611 0 $:
612 0 $:
613 0 $:

(1) Packed replacement—This is indicated by a single \"&\" followed by an identifier, delimited by any special character. If it is delimited by \".\", then the period will also be removed from the text—this allows concatenation of a replacement value with a letter or digit. Replacement is done by removing the escape character and the identifier (and period, if necessary) from the text, and inserting the value associated with the identifier in its place. The remainder of the statement to the right of the escape name, extending to the end of the scanning field, is shifted right or left as needed to accommodate the replacement value. Blanks will be added at the end of the statement if the value is shorter than the escape name, or truncation will be performed if the value is longer. In the latter case, loss of non-blank characters will be noted by MACROS.

(2) Non-packed replacement—This is denoted by a double escape character \"&\" followed by an identifier. The identifier is delimited by any special character (no special rule for period applies). The replacement of the escape name is done without any shifting of the part of the statement to the right of the escape name. If the replacement value is shorter than the escape name, then blanks are added to it. If the value is longer, then blank positions following the escape name are used to hold the trailing non-blank characters of the value. If there is still not enough room, even using the blank positions, and discounting trailing blanks of the value, then MACROS notes the truncation of the value.
IV. Using MACROS--options and JCL requirements

```
MACROS is a load module named MACRO01 which can be used
from the library PUB.JEB.LOADMOS. It uses the following
defines:

SYSPRINT -- output listing. DCB information is supplied by the
program--RECFILE=VA, LRECL=125, BLKSIZE=3825. The PGEN option
may be set to suppress the output listing in whole or part.
SYSTMP -- primary input file. May be any sequential dataset
or partitioned dataset member (or concatenation) with
logical record length 80 bytes.
SYSOUT -- MACROS processed output. DCBs must be supplied by
user such that LRECL=80 is used. Generally, this file
is associated with a temporary dataset that is used later
in the job. An option may be given to MACROS to suppress
writing on SYSOUT.
STSLIB -- secondary input file for macro definitions. This
is used to make a library of macro definitions available
to MACROS. If not required, then the DD statement can be
omitted. If used, the DD statement must define a sequential
dataset or partitioned dataset member. Concatenations of
these two items are acceptable if the DCBs are the same.
The logical record length must be 80 bytes.

Options are passed to MACROS using the PARM field on the
EXEC statement. The format is a list of keywords followed by
an equal sign and an integer, separated by commas. The options
and their default values are:

BEGC (1) Begin column for scanning of base language state-
ments. This is ignored if the FORT=1 option is used.
ENDC (80) End column for scanning of base language state-
ments. Also ignored if FORT=1.
MOUL (1) If not zero, then the MACROS output is written
on SYSOUT.
PGEN (2) If zero, then no output listing is produced.
If one, then top level statements only are listed.
If two, then base language statements from macro
expansions are listed in addition to top level
statements.
PLIB (0) If not zero, then the macro definitions edited
from STSLIB are listed when it is opened.
FORT (0) If not zero, then the FORTRAN statement conven-
ations are used--columns 1-72 of the first card of
a statement and columns 7-72 of any continuation
cards are treated as a single base language
statement for replacement scanning. If there are
no Hollerith literals in quotes in the statement,
then trailing blanks at the end of each line are
removed. There is a limit of 270 characters that
may be retained, discounting trailing blanks.
Thus, in the worst case, 3 continuation cards are
allowed.
PRBS (0) If not zero, then 'PRINT Before Substitution',
```
IV. Using MACROS--options and JCL requirements

SOURCE STATEMENT

MACRO 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 %: SIZE (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 'SIZE', 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 %: Severity is 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 %: */
699 0 %: //JOBNAME JOB etc
700 0 %: //JOBLIB DD DSN=PUB.JEA.LOADMDS,DISP=(SHR,PASS)
701 0 %: //MACROS EXEC PGM=MACROOT,PARM='PORT=1'
702 0 %: //SYSPRT DD SYSOUT=A
703 0 %: //SYSLIB DD DSN=PUB.xxx.macrolibrary,DISP=SHR
704 0 %: //SYSLIB DD DSN=MACOUT,DISP=(NEW,PASS),UNIT=SYSDA,
705 0 %: // SPACE=(CYL,2,1),DCB=(RECFM=FB,LBLCL=80,BLKLST=3200)
706 0 %: //SYSIN DD *
707 0 %: MACROS input
708 0 %: */
709 0 %: //FORTRAN EXEC FORTHCLG,COND=(5,LT)
710 0 %: //FORXT.SYSIN DD DSN=MACOUT,DISP=(OLD,DELETE)
711 0 %: //LKD.. cards if needed
712 0 %: //GO. cards
713 0 %: */
714 0 %: The SYSLIB DD statement is optional; if used, the user supplies
715 0 %: the appropriate parameters to specify his library. For testing
716 0 %: purposes, it may be given as //SYSLIB DD *, followed by an
717 0 %: ordinary input stream dataset.
718 0 %: The COND parameter on the EXEC FORTHCLG statement will
719 0 %: suppress remaining steps in the procedure if any preceding
720 0 %: (MACROS,PORTAN,PORT,PORTAN.LKED) steps had a return code
721 0 %: 6 or higher (i.e., stop on error level higher than warning).
722 0 %:
723 0 %: To use MACROS with SOUN, this JCL is representative:
```
### IV. Using MACROS---options and JCL requirements

<table>
<thead>
<tr>
<th>STMT</th>
<th>LEVEL</th>
<th>REPLC</th>
<th>SOURCE STATEMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>724</td>
<td>0</td>
<td>$</td>
<td>//jobname JOB etc</td>
</tr>
<tr>
<td>725</td>
<td>0</td>
<td>$</td>
<td>//JOBLIB DD DSN=PUBLIC.LOADMODS,DISP=(SHR,PASS)</td>
</tr>
<tr>
<td>726</td>
<td>0</td>
<td>$</td>
<td>//MACROS EXEC PGM=MACRO1</td>
</tr>
<tr>
<td>727</td>
<td>0</td>
<td>$</td>
<td>//SYSPRNT DD SYSOUT=A</td>
</tr>
<tr>
<td>728</td>
<td>0</td>
<td>$</td>
<td>//SYSLIB DD DSN=PUBLIC.macrolibrary,DISP=SHR</td>
</tr>
<tr>
<td>729</td>
<td>0</td>
<td>$</td>
<td>//SYSOUT DD DSN=ERACOUT,DISP=(NEW,PASS),UNIT=SYSDA,</td>
</tr>
<tr>
<td>730</td>
<td>0</td>
<td>$</td>
<td>// SPACE={CYL,(2,1)},DCB={RECFM=FB,LRQCN=80,BSKSIZE=3200}</td>
</tr>
<tr>
<td>731</td>
<td>0</td>
<td>$</td>
<td>//SYSIN DD *</td>
</tr>
<tr>
<td>732</td>
<td>0</td>
<td>$</td>
<td>// MACROS input</td>
</tr>
<tr>
<td>733</td>
<td>0</td>
<td>$</td>
<td>// SUMX EXEC FORTHCLG,COND={S,LT}</td>
</tr>
<tr>
<td>735</td>
<td>0</td>
<td>$</td>
<td>//FORT,SYSIN DD *</td>
</tr>
<tr>
<td>736</td>
<td>0</td>
<td>$</td>
<td>// READST, CHARMS etc</td>
</tr>
<tr>
<td>737</td>
<td>0</td>
<td>$</td>
<td>// LKED.LIB DD Info for SUMX load module library</td>
</tr>
<tr>
<td>739</td>
<td>0</td>
<td>$</td>
<td>//LKED.SYSIN DD *</td>
</tr>
</tbody>
</table>
| 740  | 0     | $      | INCLUDE LIB(SUMXloadmodule)
| 741  | 0     | $      | ENTRY MAIN |
| 742  | 0     | $      | //GO. cards for tapes,etc |
| 744  | 0     | $      | //GO.SYSIN DD DSN=ERACOUT,DISP=(OLD,DELETE) |

--- MACROS 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 ...
C REMAINDER OF MAIN PROGRAM
END
SUBROUTINE SUB
COMMON A,B
C ...
C REMAINDER OF SUBROUTINE SUB
RETURN
END

SUBROUTINE SUB2
COMMON ZURCH
C ...
C REMAINDER OF SUB2
RETURN
END

DIMENSION A(10)
C READ NUMBER OF DATA ITEMS
READ (5,1) N
C CHECK IF ARRAY A LARGE ENOUGH
IF(N.LT.10) GO TO 3
C ...
C DO SOMETHING ABOUT ERROR CONDITION
C ASSUM we USE ONLY 10 ITEMS THEN
N=10
3 CONTINUE
C ...
C PROCESS DATA
C END

***EXAMPLE 2-1***

REAL FUNCTION DISCRM*8(A,B,C)
REAL*8 B,A,C
DISCRM=D5QRT(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
WRITE (6, DB)

***EXAMPLE 4-3***

END

F(I)=F(I-2)+F(I-1)
V. MACROS O/P FROM EXAMPLES OF SECT II

EDUX FORMAT ('THIS IS A NULLARY LITERAL IN QUOTES'
1 'NOTICE THAT ALL BLANKS ARE RETAINED')

***************

***EXAMPLE 5-1***

REAL FUNCTION DISCRMN(A,B,C)
REAL B,A,C
DISCRMN=DQRT(D*B*B-4*A*C)
RETURN
END

***************

***EXAMPLE 5-2***

THE TOP LEVEL VALUES OF RLEN=4, AND PPRE=-

***************

*NEW PASS 1000,200
PHOTON + PROTON ... Pi+ Pi- PROTON

*TAPE
10
*DISCARD 0
*LIST 8,9
*SELECT
TEST 2
\-4 EQU 30001
\-4 EQU 3002

TEST 3
\-4 EQU \-30001
\-4 EQU \-30002

TEST 4
2 TRUE
3 TRUE

TEST 1
\-2 EQU 300

AND 4
TRUE

*CHAR
3P3C 8 701
M35 2 722 2 35 0 701
M45 2 723 2 65 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 80
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
V. MACROS O/P FROM EXAMPLES OF SECT II

80  .04  1.08
722

IN Variant MASS OF M45
80  .04  .26
723

IN Variant MASS OF M45 WEIGHTED
80  .04  .26
723  10
FINISH 2
EVA 41

IN Variant MASS OF M35
80  .04  1.08
722

IN Variant MASS OF M45
80  .04  .26
723

IN Variant MASS OF M45 WEIGHTED
80  .04  .26
723  10
FINISH 2
FINISH 1
EVA 21

IN Variant MASS OF M35
80  .04  1.08
722

IN Variant MASS OF M45
80  .04  .26
723

IN Variant MASS OF M45 WEIGHTED
80  .04  .26
723  10
EVA 40

IN Variant MASS OF M35
80  .04  1.08
722

IN Variant MASS OF M45
80  .04  .26
723

IN Variant MASS OF M45 WEIGHTED
80  .04  .26
723  10
FINISH 2
EVA 41

IN Variant MASS OF M35
80  .04  1.08
722

IN Variant MASS OF M45
80  .04  .26
723

IN Variant MASS OF M45 WEIGHTED
80  .04  .26
723  10
FINISH 2
FINISH 1
FINISH

*ALL DONE