PUC

Series: Monographs in Computer Science
and Computer Applications

Nº 6/69

A SOFTWARE WRITING SYSTEM

by

CARLOS JOSÉ PEREIRA DE LUCENA

Computer Science Department - Rio Datacenter

CENTRO TECNICO CIENTIFICO
Pontifícia Universidade Católica do Rio de Janeiro
Rua Marquês de São Vicente, 209 — ZC-20
Rio de Janeiro — Brasil

Simão Sirineo Toscani - 1969

A SOFTWARE WRITING SYSTEM

CARLOS JOSÉ PEREIRA DE LUCENA
COMPUTER SCIENCE DEPARTMENT
PUC - RIO DE JANEIRO

A SOFTWARE WRITING SYSTEM

The purpose of this short paper is to illustrate through an example the main features of the COMASS meta-language. The meta-language we are dealing with can handle a class of languages referred to as z-languages. The example we are going to describe in two versions is a simple z-language which can be handled by COMASS.

This z-language will be composed of a rather classical set of FORTRAN-like statements.

Its general form will be:

```
label: statement;
```

the particular statements that will be treated are:

I/O Statements:

```
READ A1,A2,...;
PRINT A1,A2,...;
```

Declarative Statements:

```
DIMENSION A1(C1),A2(C2)...;
REAL A1,A2...;
INTEGER A1,A2...;
```

Control Statements:

```
   GO TO label;
IF expression comparison expression
THEN statement;
   STOP;
```

Assignment Statement:

to be discussed later.

END of Source Program

```
END;
```

The statements presented have the usual meaning and will be compiled for simplicity into SPECTRE MAP.

Note that the language is stream oriented, with each statement finishing in this specific case by a semi-colon. This feature is imposed by the present version of COMASS.

I. The General Organization of a COMASS Program.

The goal of this description is to show practically and in a simple way how to utilize many of the features of the meta-language, hoping that somebody who wants to get involved in a practical project using COMASS be able to extrapolate more sophisticated applications.

Before going into the details of the programs appended to this text, we are going to show its general organization. The scheme below is going to illustrate a further explanation.

- COMASS PROGRAM
  - Global Statements
  - function 1
    function 2
    .
    .
    .
    function n
  - KEY statements
  - MACRO statements
- DATA
  - SOURCE DECK of the defined Z-language

Although the presented organization is not compulsory it is rather convenient and easy to be understood. Of course, by understanding the example one will be able to propose different organizations.

1. The Global Statements.

The core of this sector of the program is the compilation loop. Observe that in both versious of the example you will find immediately after the declaration and the initialization of some arrays a DO loop saying: while $EOF (which is as function of the system) is equal to zero COMPILE. The COMPILE instruction reads the next syntatic units in the input stream and compares them with all the macros (the syntatic part of it) defined in the COMASS program. If the program manages to find a matching template the instructions contained in the body of the macro will be executed giving the semantic meaning of the recognized syntatic units.

If the program finds no matching template the system will print the message.

NOTE ILLEGAL STATEMENT

and will skip to the next unit.

We will show later on in the second version of our example that a matching template will always be found. This will be obtained through a simple mechanism that was placed in the program to suggest a means for editing more specific error messages.

The reason why most of the declarations and initializations were placed in this sector of the program was the intention of putting the area of constants in common to all macros and functions in the program.

Note that DO loop initiated by the instruction

```
DO (k:=k+1) <= SYM;
```

will only be executed after the reading and processing of all statements in the input stream, that is, when $EOF be equal to 1 indicating an END of FILE. This loop will also be the last part of the program to be executed.

2. The functions

When we are in the process of translating the recognized syntatic units into a target language we find out that many of the actions we must take are common to several macros. For this reason we collect these pieces of coding and put them in the form of a function that can be called from any part of the program.

Besides this reason for constructing functions, we can anticipate that the elaboration of the semantics to certain syntatic definitions will impose the use of a recursive procedure. To cope with that fact the COMASS functions were designed implicity recursive. This avoids the intricated process of stacking several variables to attain the same effect. Functions can be anywhere in the COMASS program except in the body of a macro. To improve readability they were placed immediately after the global statements. Note that the nesting of functions is not allowed. We shall discuss later how the specific functions we use work.

3. Macro and key statements

As we already mentioned elsewhere a MACRO is composed of a syntatic and a semantic part. We called the first a template. The semantic part of the macro is the set of instructions that analyse the template and generate an output based on it. In our case the output is a SPECTRE MAP program. Eventually certain syntatic components are common to more than one macro. In this case it is convenient to factor this component out creating a KEY statement. This is used more than one time in our example but let us repeat here a specific case. The syntatic part of the macros to the statements PRINT and READ in the specified z-language could have the following form

```
    MACRO "PRINT"<$I 0-<",")$I>>";";
and MACRO "READ" <$I 0-<","$I>>";";
```

Nevertheless, if we factor the common part of the definition we would have:

```
    KEY IOLIST=<$I 0-<","$I>>;
    MACRO "PRINT" IOLIST ";";
and MACRO "READ" IOLIST ";";
```

Sometimes the KEY statements are used to improve the readability of some syntatic definitions making the program easily understandable. That is what was done with the definition of the assignement statement in our example.

II. Some basic ideas in using the language

The semantic meta-language of COMASS is a PL/I like language. This greatly facilities the task of programming in COMASS. Nevertheless it is ve very important to call the user's attention to certain basic features of the language on which most of the process of programming in COMASS is based. These features are mainly related to the process of qualification and retrieval of the components of a syntatic unit that had just been recognized.

When a syntatic unit is read from the input stream and satisfactorily matched against a macro template the programmer must know which is the particular format of the instruction he has in his hands. Suppose the following example. As the macro for the DIMENSION statement was specified as being.

```
    KEY DIMLIST=<$I"("$C")"0-<","$I "("$C")">>;
    MACRO "DIMENSION" DIMLIST ";";
```

both of the following statements will match with the template:

```
    DIMENSION A(5);     DIMENSION B(6), C(7);
```

We will now want to refer to the element being dimensioned so that we can store its name and dimension in a symbol table. If we do this we will be able in the future to reserve an area to these variables in the object program.

In our case this was done in the following way:

```
    DO DIMLIST.$I(I:=I+1)  =";
    SYM:=SYM+1;
    SYMBOL (SYM):=DIMLIST.$I(I);
    SIZE (SYM):=DIMLIST.$C(I);
    .
    .
    .
```

The notation used in this piece of coding has the following meaning:

DIMLIST.$I(I:=I+1) =" → The do loop will proceed till the ith identifier in the KEY DIMLIST is null. By the null we mean that no ith identifier was present in a specific case. Note that in COMASS all variables are initialized with zero and so the first item to be referenced in the loop is DIMLIST.$I(1).

This way of scanning the statement imposes no restriction to the number of specifications in the list.

Another rather frequent problem is the one of determining which of the alternatives of a syntatic definition has actually occurred in a certain case. The task of solving this problem is made a rather trivial one by using the system function $ALT.

For instance, when we wrote in the first version of the example the statement IF $ALT (AEXP.TERM(I).MOP(J-1)=1 THEN DO; we meant the following:

If in the ith term of the arithmetic expression (AEX.TERM(I)) the J-1th sign indicates a multiplication the get into the DO block.

Note that in the KEY MOP= <"*"|"/">, the multiplication sign occupies the first alternative.

In the process of "walking" inside a syntatic definition the use of functions prove to be a rather useful tool. If we look to the second version of the example we see the above statement took the form: A function TERMO is called TERMO (AEXP); and inside it we have

```
FUNCTION TERMO (Y);
.
.
.
IF $ALT(Y.TERM(I).MOP(J-1))=1 THEN DO;
.
.
.
```

When we have to face recursive syntatic definitions (see Version two), the use of functions is practically the only solution to the problem.

III. Description of the example.

The first fact to stress when applying the COMASS language to design things as compilers is that the most important problem that will certainly appear is the semantic description of the assignment statement.
This problem is enlarged when we consider a so simplified version of a compiler as we did in our example.

A general scheme of the appended example could be presented in the following way:

1. A set of global statements initialize some declared arrays. The constants we feed in to the arrays are operation codes in SPECTRE MAP. As we mentioned before the DO $EOF loop provokes the sequential comparison of the syntatic units with the defined macros.
2. The macros referring to the specifications DIMENSION, REAL and INTEGER, when executed, will simply add to the symbol table (SYMBOL, SIZE, MODE) the dimension and mode of the arrays and variables defined in those instructions.
3. The compilation of the statements PRINT, READ, STOP (and LABEL, that has to be considered an independent syntatic unit for obvious reasons) is also straight forward.
4. The if statement is almost entirely based in the assignement state-

ment. For this reason we will come back to it later.

5. The assignment statement is basically the only problem that has to be faced. We shall now describe it with detail.

A relatively simple assignment statement could be defined in COMASS in the following way:

```
          MACRO $I"=" AEXP;";
      KEY AEXP=  <TERM 0- <AOP TERM>>;
      KEY TERM=  <FACTOR 0-<MOP FACTOR>>;
      KEY FACTOR=<PRIMARY 0- <"**" PRIMARY>>;
      KEY PRIMARY= <$I $C  $I"("AEXP")"|"("AEXP")">;
      KEY AOP= <"+'  "-" >;
      KEY MOP= <"*"| "/" >;
```

The definition above was the primary goal of the example. Nevertheless we noticed while programming that some of the features defined above were not adding anything new to the program but, on the contrary they were confusing the example. So we decided to keep in the program only the unique features of the definition and to mention here in the text how easily the others could be added to it.

The problem was approached in two versions. In the first version we show how to "walk" through a syntatic definition by fully using the qualification feature. In the second version we show how to cope with recursive syntatic definitions. The definition we use the assignment statement is actually the following:

```
          MACRO $I"="AEXP";";
      KEY AEXP=<<TERM 0- <AOP TERM>>;
      KEY TERM=<FACTOR 0-<MOP FACTOR>>;
      KEY FACTOR=<$I|$C|"("AEXP")">;
      KEY AOP=<"+"|"-" >;
      KEY MOP=<"*"|"/" >;
```

Note that elimination of the exponentiation means only to reduce one level of qualification. Note also that the treatment of the component $I"(AEXP")" is exactly the same as the one we give to "(AEXP")".

The implicit mode of the variable will be known by checking the first letter I,J,K,L,M,N for integers. The only kind of constants accepted will be integer constants. Mixed mode is not allowed, but the forms:

Real v. = Integer exp   and
Integer V. = Real exp

will be accepted and conveniently compiled.
In the first version of the example, all the semantics referring to the assignment statement will be contained in the body of the macro $I"="AEXP";".
This is made possible by the elimination of the alternative "("AEXP")" from the definition of factor. This simplification reduces the organization of the semantics for this macro to the following scheme:

```
DO AEXP.TERM(I:=I+1)  =";
J:=0;
DO AEXP.TERM(I).FACTOR(J:=J+1)  =";
.
.
.
TF:END;
          Store in a temporary storage
          the value of one factor
AT:END;
          Add(subtract) the various
          factors.
MCR:END;
```

This has the following meaning:
To each term consider one factor at a time outputting for it the corresponding object code. In a second phase operate over the set of factors.
The exixtence of variables and constants and of the integer and real modes will originate the following typical output statement:

```
                 1
'  ' LOAD($ALT(AEXP.TERM(I).MOP(J)))
                 2
     VORC($ALT(AEXP.TERM(I).FACTOR(J)))
                 3
     AEXP.TERM(I).FACTOR(J) '  ' $STAT;
```

Its meaning is the following:

1. Output CLA or LDQ deperding on the sign (MOP) proceeding the next element in the factor.
2. Output an '=' if the element to be printed is a constant. Otherwise don't.
3. Output the element itself.

The system function $STAT prints the statement that is being processed. This output statement could produce something as

```
LDQ=125 comment
```

The only function used in this version of the program is the SETMODE function which indicates the mode of a variable. The variable MODEIN keeps track of the current mode of an expression and the variable CHECK indicates the mode of each variable at a time.

When a mixed mode expression is detected an error message is printed and the production of the object code to the current expression is interrupted.

In the second version of the example we introduced the alternative "("AEXP")" in the definition of FACTOR. This was sufficient to change the whole structure of the program. Besides that we also added to the example the IF, the GO TO statements and simple feature that intends to suggest how formal errors could be handled by the language.

The structure of the assignment statement in the second version now has the following form:

```
(1) FUNCTION TERMO(Y)
          .
          .
          .
```

```
MACRO $I"="AEXP";";                  DO Y.TERM(N:=N+1)¬ =";
  TERMO (AEXP): → (1)          (2) ← CHECK:=FATOR(Y.TERM(I),PROV(N));
  .
  .
  .
  END;
                                    {Operations involving the
                                    {factors (Σ)
                                     END;
                               (2)     FUNCTION FATOR(X,PROVN);
                                         .
                                         .
                                         .
                                       DO X.FACTOR(J:=J+1)¬ =";
                                         .
                                         .
                                         .
                                       GO TO (VARI,CONST,PAREX)$ALT(X.FACTOR(J))
                                         .
                                         .
                                         .
                                   VARI:CONST:
                                         .
                                         .
                                         .
                                  PAREX:
                               (1) ←    TERMO(X.FACTOR(J).AEXP);
                                         .
                                         .
                                         .
                                     END;
```

Note that to introduce indexed variables and fucntions the only thing to do would be to create a new entry in the computed go to with an identical recursive call as in PAREX. Of course the only difference would be in the few output instructions that would generate object code at the end of the recursion.

Note that in this version practically all working areas were declared as global areas.

Although we had no intention of optimizing the object code produced we suggest one way of doing this in the semantics of one single factor. Note that we keep track of the register being used and that we shift the information back and forth accordingly. This could be with a few changes extended to the whole program.

Note the way the macro (were the IF statement defined)refers to the arithmetic statements involved in the instruction e.g. TERMO(AEXP(1));

In the if statement there is the necessary check for mixed mode across the comparison with the corresponding error message.

The macro we call 'ERROR TRAP' recognizes by default any set of syntatic units finishing by a semi-colon:

```
                MACRO PT= <";"  | $SU  PT >;
```

The objective of the utilization of this feature is to show how a programmer could enlarge the number of error messages edited.by his COMASS compiler.

Suppose that one believes that the omission of commas in the list of specification of a Dimension statement is a rather common error. If he wants to issue an error specially for this very specific case, he could write down the following:

```
                MACRO "DIMENSION" $I"("$C")"0-< 0","$I"("$C")' >>;
                NOTE  ' COLON MISSING IN THE FOLLOWING DIMENSION
                           STATEMENT '   $STAT;
                END;
```

The 0"," means that the comma may or may not be present before the identifier in the list. Again this technique could be extended and applied wisely.

Conclusion

The COMASS meta-language whose utilization one can master in a very short period of time is potentially an excellent tool to be applied in software development.

Although we were not able to experiment with it in other fields rather than compiler, we think that intuitively the following other applications could be listed:

1) Design of special purpose languages.

One numerical analyst could want to interact with the computer by writing, for instances:

```
INTEGRATE  function Y=x**2
    FROM        0.5  TO  1
                using ROMBERG;
```

Also one statistician could be willing to tell the computer:

```
CORRELATE  Age WITH Income
  testing STANDARD DEVIATION
  with a T DISTRIBUTION;
```

2) Design of parts of operating systems.

On interesting application would be to teach students how to design a simple operating system using COMASS.

References

1 Robert Zarnke - "A Compiler and Software Writing System" University of Waterloo - 1969.

2 E.J. Desautels and D.K. Smith - "An Introduction to the String Manipulation Language SNOBOL" in "Programming Systems & Languages" - 1967.

3 T.E. Cheatham and K. Sattley - "Syntax Directed Compiling" in "Programming Systems & Languages" - 1967.

VERSION 1

```
  DCL    SYMBOL(2 ) CHAR(6),MODE(2 ) FIXED,
SYM FIXED,ITYP FIXED,NMODE(6) CHAR(1);
SYM:=0;
NMODE:='I','J','K','L','M','N';
DO $EOF=0;
COMPILE;
END;
'  SYMBOL TABLE';
DO (K:=K+1)<=SYM;
'' K ' ' SYMBOL(K) ' ' MODE(K);
END;
KEY  IOLIST=<$I 0-< "," $I>>;
KEY  DIMLIST=<$I "(" $C ")" 0-< "," $I "(" $C ")" >>;
KEY  AEXP=< TERM 0-<AOP TERM>>;
KEY  TERM=< FACTOR 0-< MOP FACTOR >>;
KEY  FACTOR=< $I| $C | $I "(" AEXP ")" | "(" AEXP ")" >;
KEY  MOP=<"*"|"/">;
KEY  AOP=< "+" | "-">;

     MODE VERIFICATION

FUNCTION SETMODE(X);
IK:=$SEARCH(X,SYMBOL);
IF IK=0 THEN DO;
IF $SEARCHS(SUBSTR(X,1,1),NMODE)=0 THEN
RETURN 2;
ELSE RETURN 1;
END;
ELSE RETURN MODE(IK);
END;

     DIMENSION

MACRO  "DIMENSION" DIMLIST ";" ;
DO DIMLIST.$I(I:=I+1)¬=' ';
IF I=1 THEN
'' DIMLIST.$I(I) ' RES ' DIMLIST.$C(I) '        '$STAT;
ELSE
'' DIMLIST.$I(I) '  RES ' DIMLIST.$C(I);
SYM:=SYM+1;
SYMBOL(SYM):=DIMLIST.$I(I);
IF $SEARCHS(SUBSTR(DIMLIST.$I(I),1,1),NMODE)=0
THEN MODE(SYM):=2;
ELSE MODE(SYM):=1;
END;
END;

     REAL

MACRO  "REAL" IOLIST ";";
DO IOLIST.$I(I:=I+1)¬= '';
SYM:=SYM+1;
SYMBOL(SYM):=IOLIST.$I(I);
MODE(SYM):=2;
END;
END;

     INTEGER
```

```
END;
END;
```

READ

```
MACRO  "READ" IOLIST ";";
DO IOLIST.$I(I:=I+1)¬='';
IF I=1 THEN
'   INP ' IOLIST.$I(I) '          '$STAT;
ELSE
'          INP ' IOLIST.$I(I);
END;
END;
```

PRINT

```
MACRO "PRINT" IOLIST ";";
DO IOLIST.$I(I:=I+1)¬='';
IF I=1 THEN
'   OUT ' IOLIST.$I(I) '          ' $STAT;
ELSE
'          OUT ' IOLIST.$I(I);
END;
END;
```

LABEL

```
MACRO  $I ":" ;
'' $I ' RES         C' '        ' $STAT;
END;
```

STOP

```
MACRO "STOP ;";
'  STP ' '       ' $STAT;
END;
```

END

```
MACRO " END ;" ;
'  END       ' $STAT;
END;
```

ASSIGNMENT

```
MACRO $I "=" AEXP ";";
  DCL    PROV(15) CHAR(6),MY(2) CHAR(3),DV(2) CHAR(3),
  AD(2) CHAR(3),SU(2) CHAR(3),LOAD(2) CHAR(3),CHECK FIXED(1),
  VORC(2) CHAR(2),CONVERT(2) CHAR(5);
MY:='MPY','FMY';
DV:='DIV','FDV';
AD:='ADD','FAD';
SU:='SUB','FSU';
LOAD:='LDQ','CLA';
VORC:='  ',' =';
CONVERT:='IFIX ','FLOAT';
DO AEXP.TERM(I:=I+1)¬='';
J:= ;
DO AEXP.TERM(I).FACTOR(J:=J+1)¬='';
IF J=1 THEN DO;
IF AEXP.TERM(I).MOP(J)='' THEN DO;
```

```
      IF MODEIN=2 THEN DO;
      '  LDQ ' VORC($ALT(AEXP.TERM(I).FACTOR(J)))
      AEXP,TERM(I) FACTOR(J) '        ' $STAT;
      GO TO TF;
      END;
      '  ' LOAD($ALT(AEXP,TERM(I).MOP(J)))
      VORC($ALT(AEXP,TERM(I) FACTOR(J))).
      AEXP TERM(I) FACTOR(J) '        ' $STAT;
      END;
      GO TO TF;
      END;
      CHECK:=SETMODE(AEXP TERM(I) FACTOR(J));
      IF CHECK¬= MODEIN THEN DO;
      '  ATTEMPT TO USE A MIXED MODE EXPRESSION';
      GO TO MCR;
      END;
      IF $ALT(AEXP TERM(I) MOP(J-1))=1 THEN DO;
      IF CHECK=2 & J¬=2 THEN
      '  LRS        10';
      '          'MY(CHECK) VORC($ALT(AEXP TERM(I) FACTOR(J)))
      AEXP,TERM(I),FACTOR(J).;
      END;
      ELSE DO;
      IF CHECK=2 & J¬=2 THEN
      ' LRS        10';
      IF J=2 | CHECK=2 THEN GO TO SKIP;
      '          LLS        10';
SKIP: '          'DV(CHECK) VORC($ALT(AEXP TERM(I) FACTOR(J)))
      AEXP,TERM(I),FACTOR(J);
      END;
TF:END;
TEMP: ITEMP:=ITEMP+1;
      PROV(I):='T'||ITEMP;
      IF J=1 | CHECK=2 THEN
      '  STO ' PROV(I);
      ELSE '  STQ ' PROV(I);
AT:END;
      '          CLA 'PROV(1);
      L:=1;
      IF I=1 THEN GO TO FINAL;
      DO (L:=L+1)<=I-1;
      IF $ALT(AEXP AOP(L-1))=1 THEN
      '          'AD(CHECK)' ' PROV(L);
      ELSE
      '          'SU(CHECK) ' ' PRUV(L);
      END;
FINAL: MODEIN:=SETMODE($I);
      IF MODEIN¬=CHECK THEN DO;
      '          CALL ' CONVERT(MODEIN);
      '          STO ' $I '      END OF ASSIGNMENT';
      END;
      ELSE '          STQ ' $I '      END OF ASSIGNMENT';
MCR:END;
      END;
      DIMENSION A(5),G(10);
      INTEGER C;
      REAL MIN,MAX;
      READ A,B,C,MIN,MAX;
      D=B*C/37+MIN*MAX-MAX;
      C=MIN*MAX;
      D=B*E/97+MIN*MAX-MAX;
```

```
              END ;
DATA SET UTILITY - GENERATE


PROCESSING ENDED AT EOD
```

```
RES    5      DIMENSION A(5),G(10);                         000
RES    1
INP    A      READ A,B,C,MIN,MAX;
INP    B
INP    C
INP    MIN
INP    MAX
LDQ    B      D=B*C/37+MIN*MAX-MAX;
ATTEMPT TO USE A MIXED MODE EXPRESSION
LDQ    MIN        C=MIN*MAX;
FMY    MAX
STO    T1
CLA    T1
CALL   IFIX
STO    C     END OF ASSIGNMENT
LDQ    B      D=B*E/97+MIN*MAX-MAX;
FMY    E
LRS    10
FDV    =97
STO    T2
LCQ    MIN        D=B*E/97+MIN*MAX-MAX;
FMY    MAX
STO    T3
CLA    MAX        D=B*E/97+MIN*MAX-MAX;
STO    T4
CLA    T2
FAD    T3
```

```
          DIV    N
          STQ    T5
          CLA    K        D=I*C/N+K/MN-LER;
          DIV    MN
          STQ    T6
          CLA    LER         D=I*C/N+K/MN-LER;
          STO    T7
          CLA    T5
          ADD    T6
          SUB    T7
          CALL   FLOAT
          STO    D       END OF ASSIGNMENT
          LDQ    AB         K=AB/D*END-AC*UK*PQ;
          FCV    D
          LRS    10
          FMY    END
          STO    T8
          LDQ    AC         K=AB/D*END-AC*UK*PQ;
          FMY    UK
          LRS    10
          FMY    PQ
          STO    T9
          CLA    T8
          FSU    T9
          CALL   IFIX
          STO    K       END OF ASSIGNMENT
          OUT    D         PRINT D,C;
          OUT    C
          STP    STOP;
          END    END;
          SYMBOL TABLE
1         A      2
2         G      2
3         C      1
4         MIN    2
5         MAX    2
```

VERSION 2

```
DCL SYMBCL(50) CHAR(6), MODE(50) FIXED(1),
SIZE(50) FIXED, NBMODE(10) CHAR(1),
SYM FIXED,ITEMP FIXED,NMODE(6) CHAR(1);
DCL MY(2) CHAR(3), DV(2) CHAR(3),
  AD(2) CHAR(3),SU(2) CHAR(3),LOAD(2) CHAR(3),CHECK FIXED(1),
VORC(2) CHAR(2), CONVERT(2) CHAR(5);
MY:='MPY','FMY';
DV:='DIV','FDV';
AD:='ADD','FAD';
SU:='SUB','FSU';
LOAD:='LDQ','CLA';
VORC:='  ',' =';
CONVERT:='IFIX ','FLOAT';
SYM:=0;
NMODE:='I','J','K','L','M','N';
NBMODE:='0','1','2','3','4','5','6','7','8','9';
DO $EOF=0;
COMPILE;
END;
DO (K:=K+1)<=SYM;
'' SYMBOL(K) ' RES ' SIZE(K);
END;
' END ';
KEY  IOLIST=<$I 0-< "," $I>>;
KEY  DIMLIST=<$I "(" $C ")" 0-< "," $I "(" $C ")" >>;
KEY  AEXP=< TERM 0-<AOP TERM>>;
KEY  TERM=< FACTOR 0-< MOP FACTOR >>;
KEY  FACTOR=< $I | $C | "(" AEXP ")" >;
KEY  MOP=<"*"|"/">;
KEY  AOP=< "+" | "-">;
```

SEMANTICS OF THE SYNTATIC UNIT <TERM>

```
FUNCTION TERMO(Y);
DCL PROV(15) CHAR(6);
N:=0;
DO Y.TERM(N:=N+1)¬='';
CHECK:= FATOR(Y.TERM(N),PROV(N));
IF CHECK=0 THEN RETURN ;
END;
'  CLA ' PROV(1);
L:=1;
IF N=1 THEN RETURN ;
DO (L:=L+1)<=N-1;
IF $ALT(Y.AOP(L-1))=1 THEN
'  ' AD(CHECK) ' ' PROV(L);
ELSE '  ' SU(CHECK) ' ' PROV(L);
END;
RETURN ;
END;
```

SEMANTICS OF THE SYNTATIC UNIT <FACTOR>

```
FUNCTION FATOR (X,PROVN);
DCL T CHAR(6),Q CHAR(6);
J:=0;
DO X.FACTOR(J:=J+1) ¬='';
KEYON:=0;
GO TO (VARI,CONST,PAREX) $ALT(X.FACTOR(J));
VARI:CONST;
```

```
     X.FACTOR(J) '        ' $STAT;
     GO TO TEMP;
     END;
     KEEPOP:=$ALT(X.MOP(J));
     IF MODEIN=2 THEN DO;
     ' LDQ ' VORC($ALT(X,FACTOR(J))) X,FACTOR(J) '        '$STAT;
     GO TO RECURS;
     END;
     '  'LOAD($ALT(X,MOP(J))) VORC($ALT(X.FACTOR(J)))
     X.FACTOR(J) '        ' $STAT;
RECURS: IF $ALT(X,FACTOR(J+1))>2 THEN DO;
     ITEMP:=ITEMP+1;
     T:='T'||ITEMP;
     IF $ALT(X,MOP(J))=2 & CHECK=1 THEN
     '  STO ' T;
     ELSE '  STQ ' T;
     GO TO PAREXP;
     END;
     GO TO LOOP;
     END;
     CHECK:= SETMODE(X,FACTOR(J));
     IF CHECK ¬= MODEIN THEN DO;
     ' ATTEMPT TO USE A MIXED MODE EXPRESSION';
     RETURN CHECK=0;
     END;
     IF $ALT(X,MOP(J-1))=1 THEN DO;
     IF CHECK=2 & J¬=2 THEN
     ' LRS         10';
     '  ' MY(CHECK) VORC($ALT(X.FACTOR(J))) X,FACTOR(J);
     GO TO PAREN;
     END;
     ELSE DO;
     IF CHECK=2 & J¬=2 THEN
     ' LRS         10';
     IF J=2 | CHECK=2 THEN GO TO SKIP;
     ' LLS        10';
SKIP: '  'DV(CHECK) VORC($ALT(X,FACTOR(J))) X,FACTOR(J);
     END;
PAREN:
     IF $ALT(X,FACTOR(J+1))>2 THEN DO;
     KEEPOP:=$ALT(X,MOP(J));
     ITEMP:=ITEMP+1;
     T:='T'||ITEMP;
     IF CHECK=1 THEN ' STQ ' T;
     ELSE ' STO ' T;
     END;
     ELSE GO TO LOOP;
PAREXP: J:=J+1;
     KEYON:=1;
PAREX: TERMO(X,FACTOR(J),ABXP);
     IF KEYON=0 THEN GO TO LOOP;
     ITEMP:=ITEMP+1;
     Q:='T'||ITEMP;
     ' STO ' Q;
     IF CHECK=2 THEN
     ' LDQ ' T;
     ELSE ' ' LOAD(KEEPOP) ' ' T;
     IF KEEPOP=1 THEN
     ' ' MY(CHECK) ' ' Q;
     ELSE ' ' DV(CHECK) ' ' Q;
LOOP:END;
```

[redacted]

```
END;
```

MODE VERIFICATION

```
FUNCTION SETMODE(X);
IK:=$SEARCH(X,SYMBOL);
IF IK=0 THEN DO;
IF $SEARCH(SUBSTR(X,1,1),NBMODE)¬=0
THEN RETURN 1;
SYM:= SYM+1;
SYMBOL(SYM):=X;
SIZE(SYM):=1;
IF $SEARCHS(SUBSTR(X,1,1),NMODE)=0 THEN
RETURN MODE(SYM):=2;
ELSE RETURN MODE(SYM):=1;
END;
ELSE RETURN MODE(IK);
END;
```

GO TO

```
MACRO < "GOTO" | "GO TO"> $I ";";
'  TRA '  $I '       ' $STAT;
END;
```

DIMENSION

```
MACRO  "DIMENSION" DIMLIST ";" ;
DO DIMLIST,$I(I:=I+1)¬='';
SYM:=SYM+1;
SYMBOL(SYM):=DIMLIST.$I(I);
SIZE(SYM):=DIMLIST,$C(I);
IF $SEARCHS(SUBSTR(DIMLIST.$I(I),1,1),NMODE)=0
THEN MODE(SYM):=2;
ELSE MODE(SYM):=1;
END;
END;
```

REAL

```
MACRO  "REAL" IOLIST ";";
DO IOLIST,$I(I:=I+1)¬= '';
SYM:=SYM+1;
SYMBOL(SYM):=IOLIST.$I(I);
SIZE(SYM):=1;
MODE(SYM):=2;
END;
END;
```

INTEGER

```
MACRO  "INTEGER" IOLIST ";";
DO IOLIST,$I(I:=I+1)¬='';
SYM:=SYM+1;
SYMBOL(SYM):=IOLIST.$I(I);
SIZE(SYM):=1;
MODE(SYM):=1;
END;
END;
```

READ

```
    '  INP ' IOLIST.$I(I) '          '$STAT;
    ELSE
    '        INP ' IOLIST,$I(I);
    END;
    END;

         PRINT

    MACRO "PRINT" IOLIST ";";
    DO IOLIST,$I(I:=I+1)¬='';
    IF I=1 THEN
    '  OUT ' IOLIST,$I(I) '          ' $STAT;
    ELSE
    '        OUT ' IOLIST.$I(I);
    END;
    END;

         LABEL

    MACRO  $I ":" ;
    '' $I ' RES        0' '         ' $STAT;
    END;

         STOP

    MACRO "STOP ;";
    '  STP ' '         ' $STAT;
    END;

         ASSIGNMENT

    MACRO $I "=" AEXP ";" ;
    TERMO(AEXP);
    IF CHECK=0 THEN GO TO MCR;
    MODEIN:=SETMODE($I);
    IF MODEIN¬=CHECK THEN DO;
    '  CALL ' CCNVERT(MODEIN);
    ' STO ' $I '       END CF ASSIGNMENT';
    END;
    ELSE ' STO ' $I '       END CF ASSIGNMENT';
MCR:END;

         IF STATEMENT

    MACRO  "IF" AEXP COP=<"="|"¬="|">"|">="|"<"|"<="> AEXP
    "THEN";
    DCL LABEL1 CHAR(6),LABEL2 CHAR(6),T(2) CHAR(6);
    LABEL1:='X'||$ILINE;
    LABEL2:='X'||$ILINE+1;
    TERMO(AEXP(1));
    MODEIN:=CHECK;
    ITEMP:=ITEMP+1;
    T(1):='T'||ITEMP;
    ' STO ' T(1);
    TERMO(AEXP(2));
    IF MODEIN ¬= CHECK THEN DO;
    '  MIXED MODE IN THE IF STATEMENT ';
    GO TO MCR;
    END;
    ITEMP:=ITEMP+1;
    T(2):='T'||ITEMP;
```

```
         GO TO KEYON;
   NE:   ' TZE ' LABEL2;
         ' TRA ' LABEL1;
         GO TO KEYON;
   GE:   ' TZE ' LABEL1;
   GT:   ' TPL ' LABEL1;
         ' TRA ' LABEL2;
         GO TO KEYON;
   LE:   ' TZE ' LABEL1;
   LT:   ' TMI ' LABEL1;
         ' TRA ' LABEL2;
   KEYON: KKY:=1;
         '' LABEL1 ' RES        0';
         COMPILE;
         '' LABEL2 ' RES        0';
   MCR:END;

             ERROR TRAP

         MACRO PT=< ";" | $SU PT >;
         NOTE  ' UNABLE TO IDENTIFY THE STATEMENT  ' $STAT;
         END;
         END;

         DIMENSION A(5),G(10);
         INTEGER C;
         REAL MIN,MAX;
         READ A,B,C,MIN,MAX;
         D=B*C/37+MIN*MAX-MAX;
         C=MIN*MAX;
         D=B*E/RT+MIN*MAX-MAX;
         D=I*C/N+K/MN-123;
         K=AB/D*END-AC*UK*PQ;
         PRINT D,C;
         A=(A*(D-E))+G*(E*F-G);
         I = J/ (K-L) - MN;
         IF I/(J-K)¬=I-K*L THEN B=G*D+R;
         IF K+5 >= I*7-J THEN PRINT K,I;
         GO TO 17;
         STOP ;
NOTE  UNABLE TO IDENTIFY THE STATEMENT  GO TO 17;
```

```
INP     A        READ A,B,C,MIN,MAX;
INP     B
INP     C
INP     MIN
INP     MAX
LDQ     B        D=B*C/37+MIN*MAX-MAX;
ATTEMPT TO USE A MIXED MODE EXPRESSION
LDQ     MIN         C=MIN*MAX;
FMY     MAX
STO     T1
CLA     T1
CALL    IFIX
STO     C        END OF ASSIGNMENT
LDQ     B        D=B*E/RT+MIN*MAX-MAX;
FMY     E
LRS     10
FDV     RT
STO     T2
LDQ     MIN         D=B*E/RT+MIN*MAX-MAX;
FMY     MAX
STO     T3
CLA     MAX         D=B*E/RT+MIN*MAX-MAX;
STO     T4
CLA     T2
FAD     T3
FSU     T4
STO     D        END OF ASSIGNMENT
LDQ     I        D=I*C/N+K/MN-123;
MPY     C
LLS     10
DIV     N
STQ     T5
CLA     K        D=I*C/N+K/MN-123;
DIV     MN
STQ     T6
CLA     =123        D=I*C/N+K/MN-123;
STO     T7
CLA     T5
ADD     T6
SUB     T7
CALL    FLOAT
STO     D        END OF ASSIGNMENT
LDQ     AB        K=AB/D*END-AC*UK*PQ;
FDV     D
LRS     10
FMY     END
STO     T8
LDQ     AC        K=AB/D*END-AC*UK*PQ;
FMY     UK
LRS     10
FMY     PQ
STO     T9
CLA     T8
FSU     T9
CALL    IFIX
STO     K        END OF ASSIGNMENT
OUT     D        PRINT D,C;
OUT     C
LDQ     A        A=(A*(D-E))+G*(E*F-G);
STQ     T10
CLA     D         A=(A*(D-E))+G*(E*F-G);
STO     T11
```

```
CLA    T11
FSU    T12
STO    T13
LDQ    T10
FMY    T13
STO    T14
CLA    T14
STO    T15
LDQ    G        A=(A*(D-E))+G*(E*F-G);
STQ    T16
LDQ    E        A=(A*(D-E))+G*(E*F-G);
FMY    F
STO    T17
CLA    G        A=(A*(D-E))+G*(E*F-G);
STO    T18
CLA    T17
FSU    T18
STO    T19
LDQ    T16
FMY    T19
STO    T20
CLA    T15
FAD    T20
STO    A        END OF ASSIGNMENT
CLA    J        I=J/(K-L)-MN;
STO    T21
CLA    K        I=J/(K-L)-MN;
STO    T22
CLA    L        I=J/(K-L)-MN;
STO    T23
CLA    T22
SUB    T23
STO    T24
CLA    T21
DIV    T24
STQ    T25
CLA    MN        I=J/(K-L)-MN;
STO    T26
CLA    T25
SUB    T26
STO    I        END OF ASSIGNMENT
CLA    I        IF I/(J-K)¬=I-K*L THEN
STO    T27
CLA    J        IF I/(J-K)¬=I-K*L THEN
STO    T28
CLA    K        IF I/(J-K)¬=I-K*L THEN
STO    T29
CLA    T28
SUB    T29
STO    T30
CLA    T27
DIV    T30
STQ    T31
CLA    T31
STO    T32
CLA    I        IF I/(J-K)¬=I-K*L THEN
STO    T33
LDQ    K        IF I/(J-K)¬=I-K*L THEN
MPY    L
STQ    T34
CLA    T33
```

```
         TRA    X15                                                    00001
X13      RES    0                                                      00001
         LDQ    G         B=G*D+R;                                     00001
         FMY    D                                                      00001
         STO    T36                                                    00001
         CLA    R         B=G*D+R;                                     00001
         STO    T37                                                    00001
         CLA    T36                                                    00001
         FAD    T37                                                    00001
         STO    B         END OF ASSIGNMENT                            00001
X14      RES    0                                                      00001
         CLA    K         IF K+5>=I*7-J THEN                           00001
         STO    T38                                                    00001
         CLA    =5         IF K+5>=I*7-J THEN                          00001
         STO    T39                                                    00001
         CLA    T38                                                    00001
         ADD    T39                                                    00001
         STO    T40                                                    00001
         LDQ    I         IF K+5>=I*7-J THEN                           00001
         MPY    =7                                                     00001
         STQ    T41        IF K+5>=I*7-J THEN                          00001
         CLA    J                                                      0000
         STO    T42                                                    0000
         CLA    T41                                                    0000
         SUB    T42                                                    0000
         STO    T43                                                    0000
         CLA    T40                                                    0000
         SUB    T43                                                    0000
         TZE    X15                                                    0000
         TPL    X15                                                    0000
         TRA    X16                                                    0000
X15      RES    0                                                      0000
         OUT    K         PRINT K,I;                                   0000
         OUT    I                                                      0000
X16      RES    0                                                      0000
         STP    STOP;                                                  0000
A        RES    5                                                      0000
G        RES    10                                                     0000
C        RES    1                                                      0000
MIN      RES    1                                                      0000
MAX      RES    1                                                      0000
B        RES    1                                                      0000
E        RES    1                                                      0000
RT       RES    1                                                      0000
D        RES    1                                                      0000
I        RES    1                                                      0000
N        RES    1                                                      0000
K        RES    1                                                      0000
MN       RES    1                                                      0000
AB       RES    1                                                      0000
END      RFS    1                                                      0000
AC       RES    1                                                      0000
UK       RES    1                                                      0000
PQ       RES    1                                                      0000
F        RES    1                                                      0000
J        RES    1                                                      0000
L        RES    1                                                      0000
         RES    1                                                      0000
```