(499)

EXTFOR PREPROCESSOR

A Thesis submitted to the Jawaharlal Nehru University
in partial fulfilment of the requirements for
the award of the Degree of
MASTER OF PHILOSOPHY

HARI PARKASH

SCHOOL OF COMPUTER & SYSTEMS SCIENCES
JAWAHARLAL NEHRU UNIVERSITY
NEW DELHI-110067
1982

C E R T I F I C A T E

SCHOOL OF COMPUTER & SYSTEMS SCIENCES
JAWAHARLAL NEHRU UNIVERSITY, NEW DELHI-110 067

The work embodied in this dissertation has been carried out at the School of Computer and Systems Sciences, Jawaharlal Nehru University, New Delhi-110 067.

The work is original and has not been submitted in part or full for any other degree or diploma of any University.

HARI PARKASH
STUDENT

PROF. D.K. BANERJI
DEAN

DR. R.G. GUPTA
SUPERVISOR

A C K N O W L E D G E M E N T S

It is a matter of immense pleasure for me to ove a debt of gratitude to my supervisor, Dr. R.G. Gupta, who has been the guiding spirit and a source of increasing inspiration in carrying out this work. I extend my sincere-most thanks to him, for his valuable guidance and generous help.

I am thankful to Prof. N.P. Mukherjee and Prof. D.K. Banerji, Dean, S.C.S.S., J.N.U., for their extremely helpful and inspiring attitude manifested in providing all the facilities for working on this dissertation.

I am grateful to my friends and colleagues for careful reading, perspective criticism and continuous cheerful support beyond my expectation.

My sincere thanks are due to the management of ORG SYSTEMS, Baroda for providing me the computer facilities during the implementation.

Finally, I would like to thank my better-half for assistance in text formatting, the excellent job of manuscript typing and showing the amazing patience while I was writing this dissertation.

New Delhi HARI PARKASH

C O N T E N T S

INTRODUCTION		1
1.	CONCEPTS AND PROBLEM	4
	1.1 History of FORTRAN	4
	1.2 Structured Programming	5
	1.3 Structured Programming - Advantages	6
	1.4 Statement of the Problem	8
2.	EXTFOR - EXTENDED FORTRAN	12
	2.1 Introduction	12
	2.2 Statement Grouping	13
	2.2.1 The BEGIN Statement	14
	2.3 Decision Structures	15
	2.3.1 The IF Statement	15
	2.4 Loop Structures	18
	2.4.1 The DO Statement	19
	2.4.2 The WHILE Statement	21
	2.4.3 The FOR Statement	22
	2.4.4 The REPEAT-UNTIL Statement	24
	2.5 The BREAK Statement	26
	2.6 The NEXT Statement	28
	2.7 The EXTFOR Grammar	30
3.	EXTFOR PREPROCESSOR	33
	3.1 Preprocessor as a Translator	33
	3.2 An Overview of EXTFOR Preprocessor	34
	3.2.1 Source Program Analysis	37
	3.2.2 Code Generation	38

3.3 EXTFOR Preprocessor Organisation 45
3.3.1 Buffers and Blocks 47
3.3.2 Keyword Table 48
3.3.3 Error Table 48
3.3.4 Context Stack 49
3.3.5 FOR Stack 50
3.3.6 Lexical Analysis 51
3.3.7 Parsing and Code Generation 52

4. EFFICIENCY AND OVERHEADS IN PREPROCESSOR 62
4.1 Efficiency 62
4.2 Execution Time Overhead 63
4.3 Storage Overhead 64

5. EXPERIENCE AND CONCLUSION 65

APPENDICES
A. EXTFOR SOURCE LISTING 68
B. OPERATING SYSTEM AND JOB CONTROL LANGUAGE 129
C. OS-EXTFOR INTERFACE AND OPERATING PROCEDURE 136
D. ERROR MESSAGES 150

REFERENCES 153

INTRODUCTION

A new trend in program writing was introduced by Edsger W. Dijkstra in his monograph Notes on Structured Programming (1972). It takes a 'constructive' approach to the problem of program correctness by controlling the process of program generation to produce a correct program. Initially the term structured programming (sp) was misunderstood by many programmers (e.g. sp is programming without GOTOs, one must use only the conditional and while statements). Although people have different interpretations of sp, it is now widely accepted as an approach to understand the complete programming process so as to carefully control the structure of one's code - its control flow and data organization. The overall effisiency in program development process found to be considerably improved with sp.

Sp methodology tells us not to program in a programming language, but into it. Express the algorithm in a notation that best fits the problem, then refine this algorithm into the programming language. Use any notation that fits the problem and follows clean, understandable approach. If the notation is not with available resources, simulate it in the programming language.

The algorithm expressed in programming language should be easier to read, easier to get right, easier to change, and easier to keep right as they are changed. This is often harder to do with FORTRAN but there is

no harm in simulating the features required to make program writing easy through structured approach.

RATFOR was the first attempt by Dr Kernighan in 1975, which was called a structured language "Rational FORTRAN". Some new features were introduced to FORTRAN and a tool was presented called the preprocessor to translate RATFOR into FORTRAN. The preprocessor proved as a powerful tool to provide with missing features in any programming language. A number of attempts were made by different people on development of an efficient preprocessor and for FORTRAN with better features.

We believe that building on the work of others is the only way to make substantial progress in any field. Therefore we work on the similar guidelines for FORTRAN with the idea of structured programming and use of preprocessor as a tool. Considering FORTRAN as a poor language for structured programming we have extended FORTRAN with better features which we call as EXTFOR (short for Extended FORTRAN). EXTFOR provides modern control flow statements like those in PL/I, ALGOL, or PASCAL, retaining the merits of FORTRAN - a language that is universal and portable while at the same time concealing its worst drawbacks. We are not going to touch the data types of FORTRAN. The new features are easy to read, write, and understand and readily transformable into FORTRAN.

Chapter 1 contains a brief history of development in way of FORTRAN usage. The structured programming is adopted as a trend in programming in FORTRAN.

Chapter 2 introduces the EXTFOR language features with examples and concise summary.

Preprocessing the EXTFOR into FORTRAN is a convenient way to do the job and FORTRAN compiler can take the burden of further processing. Chapter 3 presents the EXTFOR preprocessor.

Finally we discuss the efficiency and overhead in preprocessor, conclusion and experience in Chapters 4 and 5.

In appendices the procedure for interfacing EXTFOR with local OS environment and the EXTFOR operating procedure with examples, error messages list are described.

CONCEPTS AND PROBLEM

1.1 HISTORY OF FORTRAN

Hardly anything associated with computer that is as much as twenty five years old is still in use, unless it is the original Von Neumann "Stored Program" concept itself. Yet in spite of vast changes in hardware architecture and in spite of the reproduction of programming languages since 1957, we are still using FORTRAN in a form - not withstanding a major revision with the introduction of FORTRAN-IV, not much different from the original version announced by John W. Backus of IBM in 1957.

FORTRAN is mostly used for scientific and engineering applications because it is easy to learn, availability of many readymade programs written in FORTRAN and very efficient and optimized compilers, attribute to the popularity of the language.

Trends in computers have resulted in some changes in the FORTRAN language and the way it is used. The changes have occurred in two main areas. First, the FORTRAN users have consolidated to a considerable extent around the ANSI standard version of the language, and the rather arbitrary variations (which were wide spread as late as 1966) have declined in number. Most current compilers contain actual conflicts with or violations of the standard and as was intended, the acceptances of the standard has placed a certain onus on variant versions. This has resulted, of course, in increased interchangeability of programs among various users and is no doubt partly responsible for the combined popularity of the language.

1.2 STRUCTURED PROGRAMMING

Reliability is the sine qua non of software systems. There is no point in engaging in efficiency consideration for unreliable systems. And to achieve reliability we should make our code correct and clear before we try to make it faster or more compact. Structured programming is a technical elaboration of this basic point of view.

The term structured programming has been used with many different meanings since Edsger W. Dijkstra first coined the term. The term is generally misunderstood and lack of a precise definition has allowed, even encouraged, people to use it as they wished, for the sake of fashion, to attribute to structured programming (sp) what they themselves learned from monograph notes on structured programming (Dijkstra). Taken out of context or viewed in the wrong light, some of the resulting definition of sp that have appeared seem stupid (e.g. sp is programming without GOTOs), and it is quite understandable that programmers have looked askance when asked to learn and practice it.

Dijkstra proposed that we take a 'constructive' approach to the problem of program correctness by controlling "the process of generation such as to produce a priori correct program". To facilitate this process we limit the rules for program composition to those which are well understood. We would like to informally establish that our program is correct as we are designing and coding it.

The structured programming provides a systematic approach to the programming. A program can be disintegrated into small components so as to make them smaller programs. These component programs can, in turn, be regarded as systems composed of still smaller programs, and so on,down to some reasonable level where the components level to be consisting of a few statements each. The programs written with this approach are structured.

The important thing here is the interface between the components of the program which are systems themselves. Less is the interaction more it attributes to be called as structured. In the concept of structured programming both a design method and a coding technique are implied. The design aspects often embrace the phases "top-down expansion" and stepwise refinement. The coding features include restriction to a set of single entry/single exit control structures, often with explicit formatting conventions to promote readability. And top-down coding is often considered to be a part of structured programming. The top-down programming is step-by-step process of going from a program functional specifications to the coded program.

1.3 STRUCTURED PROGRAMMING - ADVANTAGES

The general view of the structured programming is that it is the general method by which leading programmers program.

It is the process of minimizing the number of interactions between the components of a program.

It helps with converting arbitrarily large and complete

flow charts into a program represented by interrelating and nesting a small number of basic and standard control logic structures.

It is a manner of organising and coding programs that makes the programs easily understood and modified.

The main advantages of structured programming include ease of writing, understanding and debugging. Writing a program becomes easier because of the top-down approach. The program is written as an interconnection of a few routines with the few interactions between components, they can be designed and tested in relative isolation.

Another important advantage of sp is that the programs so written are easy to understand by a person other than the program writer. Application programs are often modified and run for years by different programmers after the original programmer has gone some where else.

The debugging is the main phase of program development and the time involved in debugging a program is a function of the way (methodology) used in writing the program. Sp facilitates to debug small components in isolation using test data appropriate to the program. When small components have been debugged they can be put together to form the next level of component, which can be debugged in turn and any errors introduced at this level can be attributed with some possibility to the interface between components. The source of the error is narrowed down in this way

and debugging of the large components is simplified. We often use subroutines as component and they are tested independently with variety of test data. The components are interfaced through the parameters.

As far as efficieney is concerned, overall efficiency in a program development is increased considerably.

The current software/hardware ratio of a system is greater than in the early age of computers. Programmers in large software projects typically spend their time as follows:

40-50% is program check out, and one third in program design and less than 20% is actual coding. So the efficieney can be increased if we give more emphasis on design and coding so that the debugging time is minimized. This makes quite clear that sp helps in increasing the overall efficiency.

1.4 STATEMENT OF THE PROBLEM

Some of the main features of a programming language are control structure, data type and input/output. In view of modern trend of programming, FORTRAN has quite weak control structures. In spite of the inadequency of some features in the language, not many extensions have been made of the FORTRAN and whatsoever were made, have continued to improve the usefulness of the language, both for certain advanced application and for the novice users. Some considerable extensions include WATFOR-WATFIV with its exeellent pedagogical features - format free input-output, expressions in output lists, CHARACTER data type, expressions of mixed type, multiple assignments, etc.

As we know that the worst deficiency in using FORTRAN is in the control flow statements - conditional branches and loops which express the logic of the problem. The following examples illustrate:

FORTRAN DO restricts the user to going forward in an arithmetic progression. One cannot use DO with index going from N to M where M< N:

```
      DO  10  I=10,5,-1
        .
        .
10    CONTINUE
      DO  20  R=1.0,3.0,0.5
        .
        .
20    CONTINUE
```

In first case the index is negative arithmatic progression. Second case, the index is real. Both the cases violate the FORTRAN loop index rule. Anyway these deficiencies do not come in the way of developing structured programs using FORTRAN, but the users do feel difficulties with such restrictions.

A statement label is written just after 'DO', which is required to terminate the scope of the DO loop. This label is more serious as far as sp concept is concerned.

The conditional statements in FORTRAN are primitive. The arithmetic IF statement forces the user into at least two statement numbers which in turn give rise to two (implied) GOTOs leading to unintelligible code.

For example:

```
      IF (arithmetic expression) 10,100,200
        .
        .
 10   CONTINUE
        .
100   I=1
        .
200   TEMP=ARRY(I)
```

The above IF statement is a three-way branch. Execution of this statement causes evaluation of the expression following which the statement identified by the label 10, 100 or 200 is executed next as the value of expression is less than zero, zero or greater than zero, respectively.

The logical IF is better:

IF (logical expression) "statement"

Upon the execution of this statement, the logical expression is evaluatéd; if the value is false statement which following IF statement (next statement) is executed, if value is true the "statement" is executed. Unfortunately this statement is restricted from the use of another "IF statement", DO loop as "statement" in above format. Secondly there can be no ELSE part to the IF statement: There is no way to specify an alternative action if the logical expression is false.

The result of these feelings is that FORTRAN programs must be written with numerous labels and branches. The resulting code is particularly difficult to read and understand, and thus hard to debug and modify.

In view of the above discussion we feel that the FORTRAN language requires some modifications or extensions so as to improve the correctness, readability and efficiency in program writing, in other words to write structured programs. The modification of the existing control structures leads to writing a new compiler, which means too much of a burden for a simple problem.

The approach we choose here is to make FORTRAN a better programming language by extension of the control structures and retaining the merits of the FORTRAN.

The primary purpose of this work is to define a new language which is an extension of the FORTRAN. The extensions are only of control structures; the format of the FORTRAN is not changed and the data structures are not extended. This new language we call as "Extended FORTRAN" (EXTFOR). Further, a system program called preprocessor is written to translate the EXTFOR program into the unpleasant FORTRAN.

EXTFOR - EXTENDED FORTRAN

2.1 INTRODUCTION

A program is a static representation of a computation. The execution of a program is a dynamically changing activity. When we use program text to reason about a program execution, we are crucially concerned with the program's flow of control which is determined by the type of constituent control structures. In structured programming we limit the use of three forms of control structures. These forms are simple sequencing or concatenation of statements, selecting the next statement(s) to be executed by the result of a test (often with a true or false answer) and conditional iteration. The restriction to these three means of program compositions is far from arbitrary, rather, it is based directly upon consideration of program correctness, and readability of the program and efficiency.

Each of the forms has the property that it is single entry/single exit. As such they provide alternative ways to structure or to decompose a one-in/ one-out box.

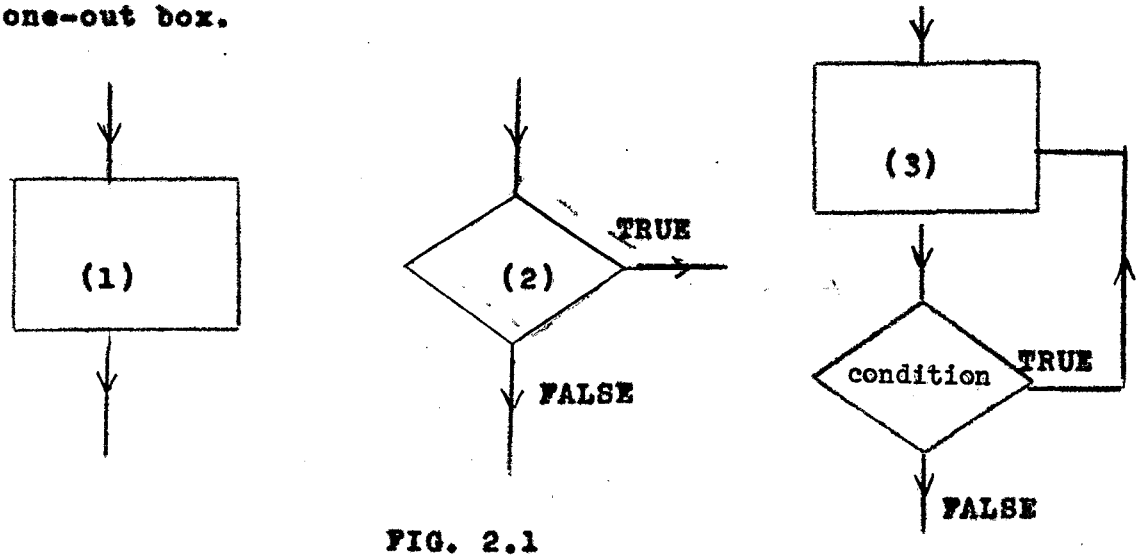


FIG. 2.1

When expanded into code, the forms can be read from top to bottom with no jumping around in the program text, minimizing the interaction among program components.

The control flow structures discussed in this chapter are the extensions to the FORTRAN and selected to provide, keeping in mind the above discussed three forms of control structures for writing structured programs.

2.2 STATEMENT GROUPING

FORTRAN provides no way to group statements together, short of making them into a unit of program. The construction "if a condition is true, do this group of statements", for example:

```
IF (X.GT.100)  Y=X+10
               CALL ERROR (X)
               WRITE (6, 10) Y
               X=0
```

cannot be written directly in FORTRAN. The only way to write is:

```
      IF (X.LE.100) GO TO 100
         Y=X+10
         CALL ERROR (X)
         WRITE (6, 10) Y
   10    FORMAT ("ERROR --", I4)
         X = 0
  100    CONTINUE
```

2.2.1 THE BEGIN STATEMENT

In our EXTFOR a group of statements can be treated as a unit by enclosing them between BEGIN and END, of course this END has different meaning than FORTRAN "END" which terminates the scope of a program segment. The format is:

```
BEGIN
  statement-1
  statement-2
     .
     .
  statement-n
END
```

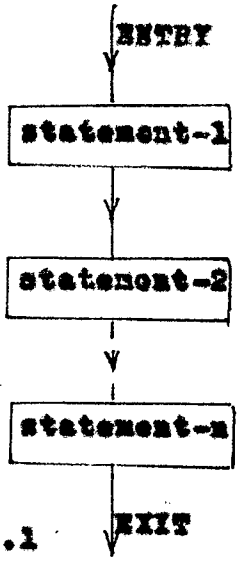


FIG. 2.2.1

BEGIN-END enables us to execute the statements sequentially provided the execution of any of the statements does not transfer control outside the block of statements. This structure is of first type of program components. Using this structure the example in section 2.2 can be written as:

```
IF (X.GT.100) THEN BEGIN
                     Y = X + 10
                     CALL ERROR (X)
                     WRITE (6,10) Y
                     X = 0
                     END
```

This block structure is different from BEGIN-END in PL/I. In PL/I it is often used for new declarations of identifiers or when the meanings of names are to be restricted within a program segment.

2.3 DECISION STRUCTURES

These are structured statements which control the execution of their scope on the basis of a logical expression. These are second type of program components.

2.3.1 THE IF STATEMENT

The format of this construct is:

IF (condition) THEN statement-1
ELSE statement-2

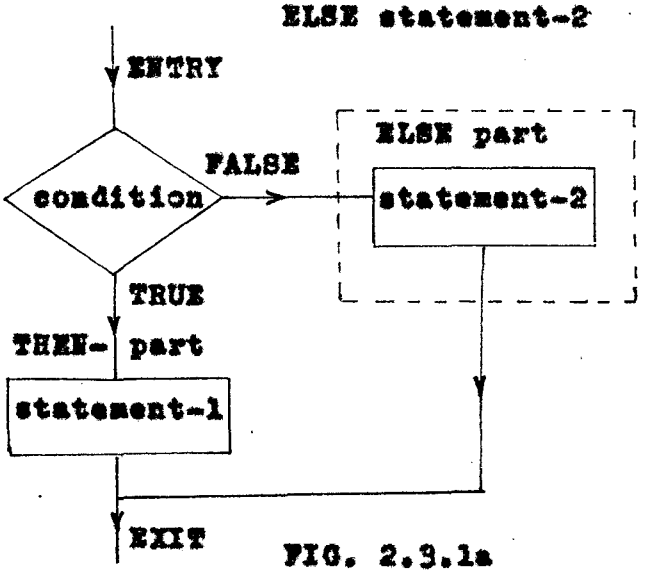


FIG. 2.3.1a

The condition part of the statement is a valid FORTRAN logical expression with value FALSE or TRUE. Statement-1 and statement-2 are any of the EXTFOR statements with some restrictions on using a mixed combination of FORTRAN statements and extended features like FORTRAN DO used as statement in THEN or ELSE unit.

The execution of IF statement first evaluates the condition, if the condition is TRUE then the statement following THEN is executed and ELSE part (if present) is skipped; if the condition results as FALSE the statement following THEN is skipped with execution of the ELSE

part (see FIG. 2.3.1a). So one and only one of the two statements (statement-1 or statement-2) is executed when the IF-THEN-ELSE is encountered. The ELSE part is optional, if not used, the statement is equivalent to FORTRAN logical IF statement. For example:

```
IF (LASTC .GT. MAXL) THEN LASTC = 1
```

is same as

```
IF (LASTC .GT. MAXL) LASTC = 1
```

Both these statements are permitted in EXTFOR.
The recursive use of logical IF statement for example:

```
IF (C.NE.EOF) IF (LINE.LT.60) PAGE=PAGE+1
```

is not allowed in FORTRAN. However, in EXTFOR this nesting of IF statements is permitted. This construct is useful in selecting the next statement execution on subjected to a number of conditions specified. For example:

```
IF (condition-1)
      THEN IF (condition-2)
              THEN statement-1
              ELSE statement-2
      ELSE IF (condition-3)
              THEN statement-3
              ELSE statement-4
```

Execution of this nested construction can be shown graphically as follows:

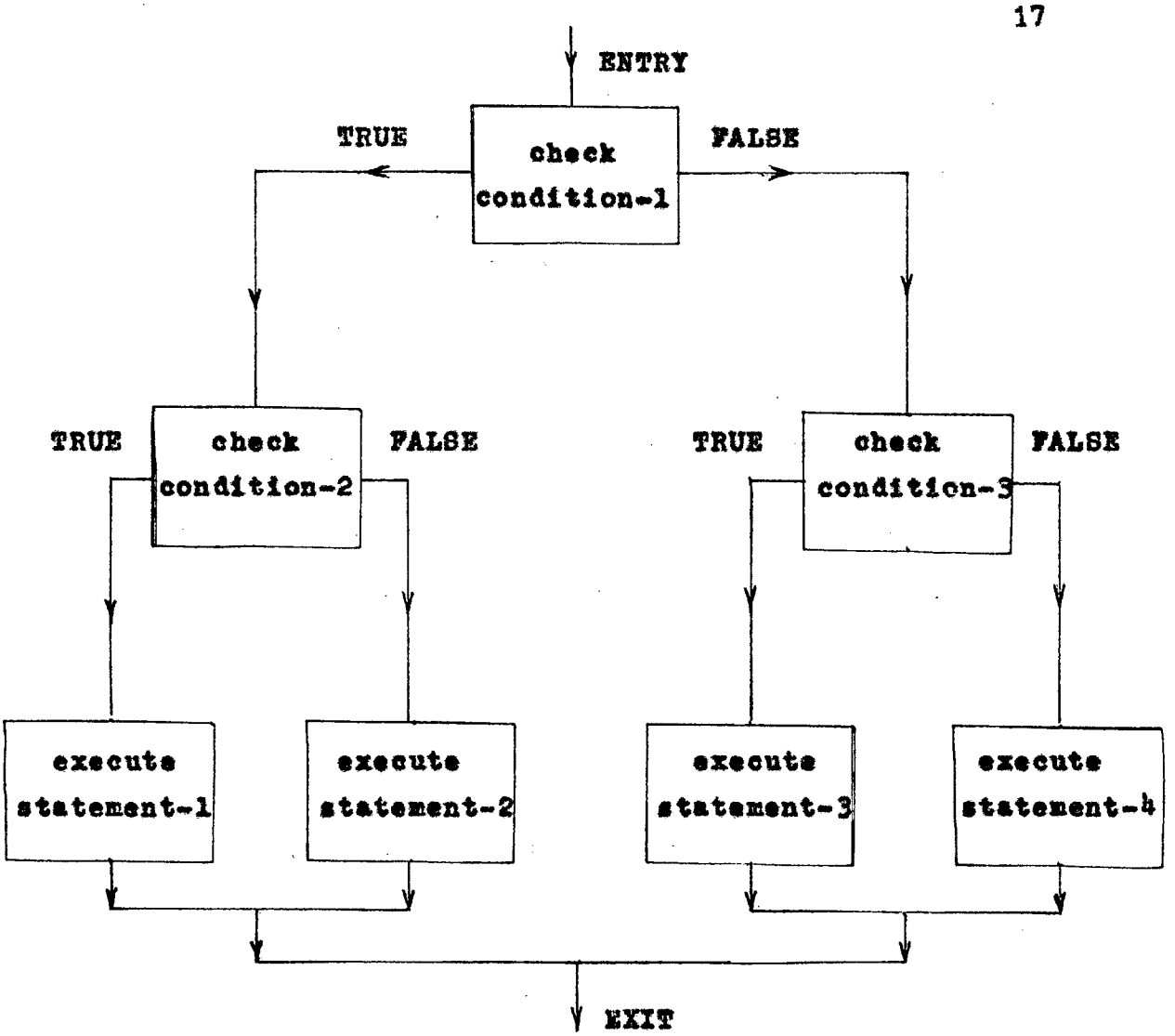


FIG. 2.3.1b

As in most of the languages, the above construction is ambiguous - the ELSE could be associated with either IF. BEGIN-END can be used to disambiguate this as desired. In absence of BEGIN-END, each ELSE goes with the previous un-ELSEd IF, just as in PL/I.

EXTFOR does not provide a CASE statement, since it may be readily simulated with a series of ELSE IF statements as:

```
IF (condition) THEN
   ELSE IF (condition) THEN
        ---
   ELSE IF (condition) THEN
        ---
   ELSE
        ---
```

2.4 LOOP STRUCTURES

These control statements are used to form loops; for both conditional and unconditional iterations. The flow of control in these control structures is pictured in the following figure.

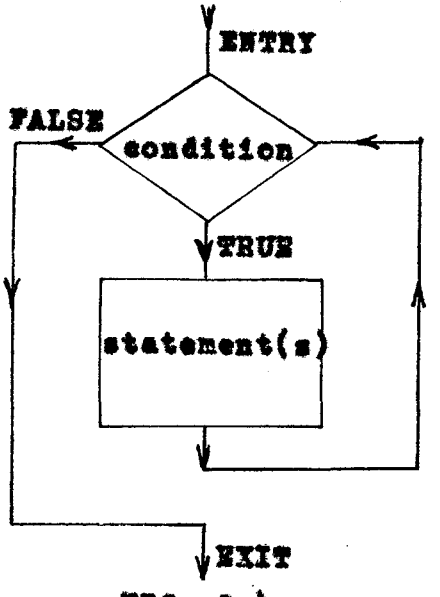


FIG. 2.4

The diagram has a decision symbol following the entry. Depending on the outcome of the test (value TRUE or FALSE), the flow continues either to the processing statement(s), the flow loops back to the entry connection; or exit from the loop. Looping is the objective of this control structure.

The number of times the processing statement(s) called the scope of loop, executed is determined by the specified conditions.

Four loop structures are selected for EXTFOR: DO-statement, WHILE-statement, FOR and REPEAT-UNTIL statements.

2.4.1 THE DO STATEMENT

The format is:

```
DO  index=i1,i2,step
    statements
END
```

The function of this DO is identical to the FORTRAN DO loop. The difference is that it does not use any statement number and the loop scope is limited to the matching END statement. It helps to avoid use of statement label, which has to be remembered to terminate the loop scope.

The scope of the loop is executed with index = initial value (i1); then index is incremented by the step value and loop is executed till the index value exceeds the final value i2, with increments in index after each execution of loop. The diagram shows the flow of control:

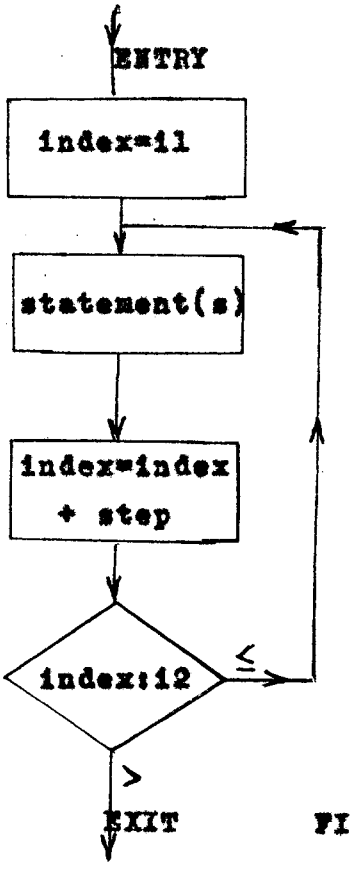


FIG. 2.4.1

Example:

```
      DO I=1,N
         X(I)=0
         Y(I)=0
      END
```

equivalent FORTRAN DO

```
      DO 10 I=1,N
         X(I)=0
         Y(I)=0
   10 CONTINUE
```

Nested DOs are permitted in EXTFOR provided the structures are ended with matching END statements.

Example:

```
      DO I=1,10
      DO J=1,10
         C(I,J)=0
      DO K=1,10
         C(I,J)=C(I,J)+A(J,K)
                *B(K,J)
      END
      END
      END
```

equivalent FORTRAN DO

```
      DO 30  I=1,10
      DO 20  J=1,10
             C(I,J)=0
      DO 10  K=1,10
             C(I,J)=C(I,J)+A(J,K)
                    *B(K,J)
   10 CONTINUE
   20 CONTINUE
   30 CONTINUE
```

Like FORTRAN DO, EXTFOR DO is highly restrictive: the index must typically run from positive, non-zero integer value up to some positive integer limit, and in many versions of FORTRAN the loop is always obeyed at least once regardless of the limits, because the first test is done at the bottom. Jumps into a DO loop are forbidden. The jump out of the loop is by reaching, the statement following the loop after the loop index has executed its maximum value. However, jumps out of the DO loops to other parts of the program are of course permitted, of course this practice is discouraged.

2.4.2 THE WHILE STATEMENT

To overcome the difficulties with DO statement; EXTFOR provides the WHILE statement. The WHILE loop causes its scope to be repeatedly executed while a specified condition is true. As the following diagram shows the condition is checked prior to the first execution of the scope, thus if the condition is initially false the scope will not be executed at all. The format is:

```
WHILE (condition)
      statement-1
      statement-2
          .
          .
END
```

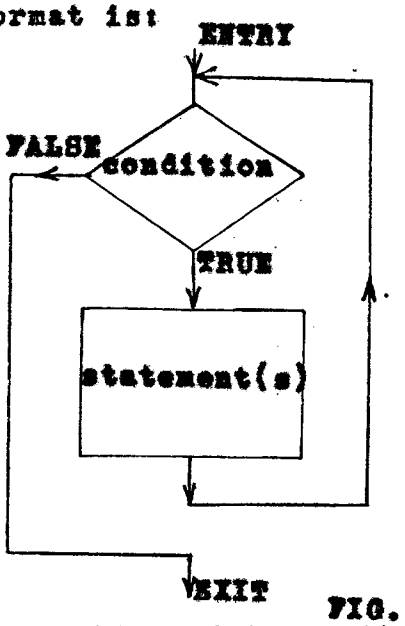


FIG. 2.4.2

The condition part is same as a valid FORTRAN condition. The scope of the loop is terminated by a matching end.

The nested WHILE statements are allowed provided the structures are ended with matching END statements. Example:

```
WHILE (ABS (TERM)  S)
      TERM = - TERM * X
      SN   = SN + TERM
      WHILE (SN  0)
            SN = - SN * 2
            TERM2 = TERM2 + SN
      END
      TERM = TERM2 ** 2
END
```

The innermost END ends the innermost WHILE statement.

2.4.3 THE FOR STATEMENT

This statement compactly summarizes all the loop control code in one line, in other words it separates the scope of the loop from the reason-for-looping. The FOR statement allows explicit initialization and increment steps as part of the statement. The test part of loop is placed at the top of loop instead of the bottom which eliminates a potential boundry error. The format is:

```
FOR (initialize; (condition); reinitialize)
    statement
```

"initialize" is any single FORTRAN statement, which is executed once before the loop begins. "reinitialize" is any single FORTRAN statement, which is executed at the end of each pass through the loop. "condition" is any logical expression.

The diagram illustrates the flow of control.

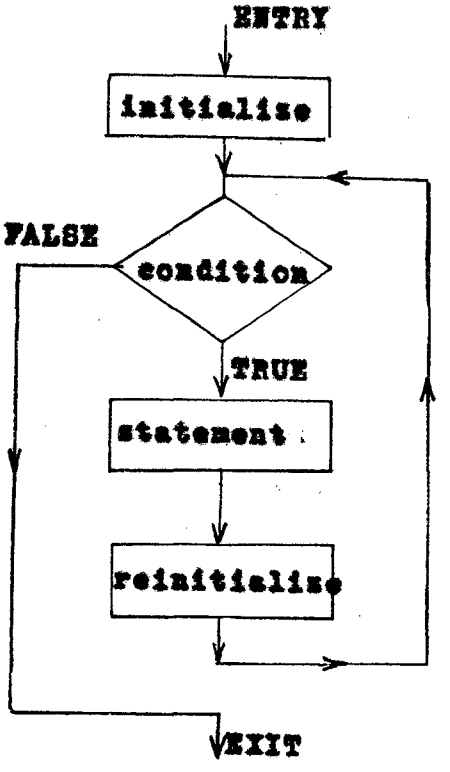


FIG. 2.4.3

The initialize statement is executed only once. Then the condition is checked, if it results a FALSE the statement following the FOR structure is given control. For TRUE value the scope of loop (statement) is executed followed by the execution of reinitialize. The loop continues from the evaluation of condition and further action depending upon the FALSE or TRUE value as discussed above.

The FOR statement is useful for backward loops, chaining along lists, loops that might be performed zero times, and things which are hard to express with a DO statement. For example, here is a 'backward DO loop':

```
FOR (I=80; (I.GE.50); I=I-1)
    IF (STR (I).EQ.BLANK) THEN NBLK=NBLK +1
```

The above code scans the character string STR(80) from last character to 50th character in the right to left direction.

Example:

```
TERM=1.0
FOR (I=2.35; (I.GT.0); I=FUN(I))
    TERM=TERM+VALUE (I)
```

From this example it is clear that the increment in FOR statement need not be an arithmetic progression. FUN and VALUE are functions, FUN generate randum values for I.

In FOR any of initialize, condition and reinitialize may be omitted, although the semicolons must remain. For example

```
(a)     FOR ( ; (condition); reinitialize)
(b)     FOR (initialize; ;reinitialize)
```

(c) FOR (initialize; (condition);)

(d) FOR (; (condition);)

(e) FOR (; ;)

Examples (b) and (e), where the condition is omitted represent indefinite repetition of loop scope execution. Examples (a) and (d) are similar to WHILE loop.

The nested FOR loops are allowed in EXTFOR, for example:

```
FOR (I=2; (BUF(I).NE.BLANK); I=I+1)
    FOR (J=I-1; (BUF(J).NE.EOS); )
        TOKEN (J)=BUF(I)
```

2.4.4 THE REPEAT-UNTIL STATEMENT

The REPEAT-UNTIL is a loop that is repeated one or more times until the condition test at the trailing end of loop results a TRUE. The format is:

```
REPEAT
   statement-1
   statement-2
      .
      .
   statement-n
UNTIL (condition)
```

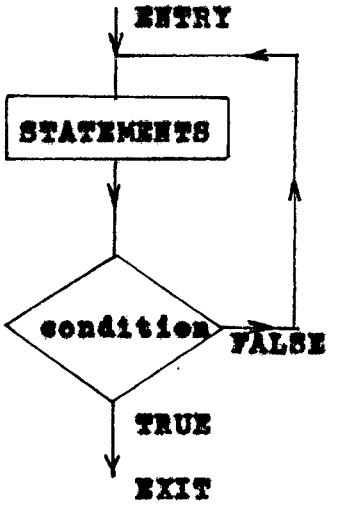


FIG. 2.4.4

The figure illustrates the flow of control within the structure. The scope of the loop (statement(s)) is executed first. The condition is checked and the FALSE value results in a repetition of the loop scope execution, which continues until the condition check results a TRUE value. This loop structure differs from others, because of the reverse condition for looping i.e. for exit out of the loop the condition must become a TRUE value. The loop is repeated atleast once, similar to the DO statement and the condition is examined only at the end of the loop. We shall remember, that the scope of the loop should modify the values of variables involved in condition checked at the end of loop such that after a finite number of repetitions the control comes out of the loop structure. For example the following statements result an indefinite loop.

```
NUMB=1
SUM=100
REPEAT
   SUM=SUM+NUMB
   NUMB=NUMB+1
UNTIL  (SUM.EQ.100)
```

The value of SUM keeps on increasing by successive executions of loop scope and it is never equal to 100.

The condition part of UNTIL is optional; if it is omitted, the result is an infinite loop, unless the loop is broken by some conditional or unconditional branch from the loop scope. Example:

```
I=1
REPEAT
      STR1(I)=BUF(LASTC)
      IF (STR1(I).EQ.STR2(I)) THEN I=I+1
      IF (STR1(I).EQ.EOS) THEN BREAK
      LASTC=LASTC+1
UNTIL
NTOK=NTOK+1
```

The above loop is executed infinitely unless the underlined IF statement is there. When the condition of this IF results a TRUE, the loop is broken by BREAK statement transferring control to the statement NTOK=NTOK+1 which follows the loop.

2.5 THE BREAK STATEMENT:

As a policy we don't want to allow frequent use of GOTO's in EXTFOR programs since it causes a lot of confusion in program reading by creating more interactions among the program units. Therefore some statements are required to be included in our language to provide implied branches. The BREAK statement provides one way to branch out of a loop structure. Whenever this statement is encountered during the program control flow, the control is transferred to the statement, next to the loop structure. The format is:

```
BREAK level
```

where level is a positive integer indicates the depth of the loop nest to be broken. For example

```
WHILE  (NGETC (C).NE.EOF)
       CALL PUTBAK (C)
       REPEAT
         IF((C.NE.BLANK) .AND. (C.NE.TAB))THEN BREAK
         CALL PUTBAK (C)
         UNTIL
         I=1
         WHILE (I.LT.TOKS12)
               GETTOK = TYPE (NGETC (LEXSTR (1)))
               IF (GETTOK.NE.LETTER) THEN BREAK 2
         END
         GETTOK = ALPHA
  END
```

The execution of the first BREAK when the condition results a TRUE causes control to be given to the statement I=1 which just follows the REPEAT loop. Here the level is omitted and only one loop is terminated by default even if the BREAK is contained inside several nested loops. The second BREAK statement terminates two loops i.e. the control comes to the statement next to the outermost END.

In case the level specified is greater than the number of nested loops enclosing it, the execution of BREAK transfers control to the statement next to the outermost loop structure. This statement also provides one way to jump out of the BEGIN-END block. For example

```
----
BEGIN
  SUBST = INJECT (NEW)
  IF (SUBST.EQ.ERR) THEN BREAK
  SUBST = OK
  LASTM = M
END
CALL DELET (LINE, LINE, STATUS)
```

When SUBST has value equal to ERR the BREAK statement causes an immediate exit from the block and control is given to statement CALL DELET.

The BREAK statement should not be used in an undisciplined way, we should always try to express the exit condition, that is an IF controlling a BREAK. The BREAK statement without an enclosing loop is treated as a comment anyway an error message is flagged.

2.6 THE NEXT STATEMENT

This statement causes whatever loop it is contained in to branch to the bottom of the loop; skipping the rest of the loop body, so it causes the next iteration to begin. The format is:

NEXT level

In a WHILE, REPEAT or DO, it goes immediately to the condition part; in a FOR, it goes to the reinitialize step. "level" is a positive integer specifies the level of loop in a nest of several loops for which the next iteration to begin. The other loops inside the loop at "level" are terminated consequently.

Example:

```
DO I = 1, N
   IF  (X (I) .LT.0.0) THEN NEXT
   SUM = SUM + X(I)
END
```

This loop skips over negative values to be added in SUM because whenever the value of X (I) is equal to 0.0 the NEXT causes the SUM = SUM + X (I) statement to be skipped and next iteration begins with next value of I. If I = N and X (I) 0.0 then the execution of NEXT terminates the loop. The "level" can be omitted which has default value as one.

Example:

```
FOR (I=1; (I.LE.10); I=I+1) BEGIN
    A (I) = B(I) **2 + 4*A(I)*C(I)
    DO  J = 1, 10
    N = A (I) + B (J)
    CALL FIGR (N)
    IF (FIGR.GT.100) THEN NEXT 2
    STERM = STERM + A (5) * N
    END
    WRITE (6, 100) STERM
END
```

In this example when FIGR is greater than 100, the execution of NEXT causes, first to terminate DO loop and then the control is transferred to the initialise statement I=I+1 by skipping WRITE statement. In the next iteration DO loop is started with value J=1 as a fresh.

2.7 THE EXTFOR GRAMMAR

The syntax (grammar) is a set of rules by which a legal program in the language is written or recognised. The formal representation of a language by grammar has several advantages rather than describing the language with an informal description in words. The language specifications can be made fairly precise this way.

The language grammar is written as a finite non-empty set of rules, usually written as

U ::= X

where U is a symbol called non-terminal or syntactic entity. X as the right part of the rule is a non-empty finite string of symbols. The symbols appearing on right part may be a mixture of non-terminals and other symbols called terminals.

There are different notations used to describe the language grammar formally, the one used here is the Backus-Naur Form (BNF). In BNF non-terminals are written as symbols enclosed in corner brackets " < "and" > " and sign " ::= " reads "is replaced by". Alternate ways of rewriting a given non-terminal are separated by a vertical bar "|" (read "or"). Symbol $\left\{A\right\}_{a}^{b}$ represents the repeated appearances of A with minimum a times and maximum b times.

The grammar for the specifications of the Extended Fortran language is following one:

```
<program>    ::=   <statement>
                 | <program> <statement>
<statement>  ::=  IF (condition) THEN <statement>
                 | IF (condition) THEN <statement>
                                  ELSE <statement>
                 | BEGIN
                     <statement>
                   END
                 | DO limits
                     <statement>
                   END
                 | WHILE (condition)
                     <statement>
                   END
                 | FOR (initialize;(condition);reinitialize)
                     <statement>
                 | REPEAT
                     <statement>
                   UNTIL (condition)
                 | BREAK  <level>
                 | NEXT   <level>
                 |<label> <statement>
                 | *        END
                 |          OTHER
<level>      ::= [<digit>]2
                 0
<label>      ::= [<digit1>]1 [<digit>]4
                 0          0
<digit>      ::= 0|1|2|3|4|5|6|7|8|9
<digit1>     ::= 1|2|3|4|5|6|7|8
```

The first rule says that a program is a statement, or a program followed by a statement. In other words, a program consists of one or more statements. A statement is one of the constructs represented by the rules other

than first. The vertical bar "|" indicates a choice of alternatives. All the underlined symbols are keywords; keyword is a symbol which has special meaning when used at a specified place in the program. The user is restricted from using these keywords as variable names, particularly the first symbol of the statement. Here EXTFOR grammar differs from FORTRAN since there is no keyword in FORTRAN. A statement consists of, for example keyword WHILE followed by parenthesized condition and a group statement followed by END. The definition of a statement is recursive in EXTFOR which is not allowed in FORTRAN for example, a nest of IF statements.

The '* END' statement is used to indicate the end of subprogram, '*' is put in first column of the source line. The grammatical type OTHER represents all the standard FORTRAN statements. This type is of important simplification, for it frees EXTFOR translator from having to know very much about FORTRAN. Any statement which is not discussed in the preceding sections is recognized as OTHER. Therefore, any statement which does not begin with one of the underlined keywords (leaving the label apart), it must be an OTHER, and no real processing is needed on it.

"Level" is a 2 digit positive integer specifies the loop level to be broken by BREAK/NEXT statement.

"Label" is a standard FORTRAN 5 digit statement number less than 90000. The EXTFOR translator uses label numbers greater than 89999 as generated labels (discussed in chapter 3). Although these labels are relatively rare in EXTFOR programs, the grammar does allow them.

The format of source is same as that of FORTRAN. A 'C' in column 1 indicates a comment line, 1 to 5 columns for statement label, column 6 is continuation column and columns 73 to 80 are for documentation purposes and ignored by EXTFOR.

EXTFOR PREPROCESSOR

3.1 PREPROCESSOR AS A TRANSLATOR

A program written in a high level language, say FORTRAN can not be executed by computer directly. A translator is required to translate the source program into a set of executable instructions (machine language) for the computer as shown in the following figure.

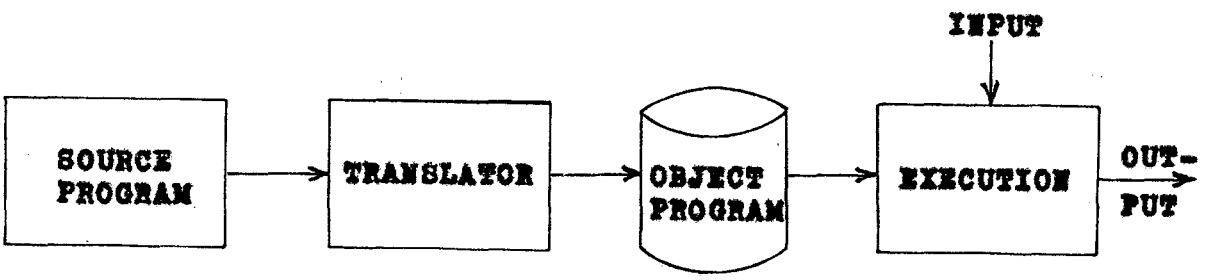


FIG. 3.1

Some translators accept a source program, translate it into some intermediate structure, then execute each operation given in the intermediate structure. Such a translator is called an interpreter. An interpreter is considerably less efficient in execution, but a program can be developed very fast with an interpreter.

A translator which accepts source program and translates it into machine executable code is called a compiler. The advantage of a compiler over an interpreter

is that it generates an efficient machine code. The high level language gives user the flexibility of writing programs independent of computer to be used. A FORTRAN program should yield the same results for a given input regardless of the to which it is translated. These results should be expected from the specification of the FORTRAN language and the algorithm as expressed in FORTRAN.

A program written in EXTFOR language has to be transformed into some code to make it executable or interpretable on a computer. One way to do it is, develop a compiler; which means too much of work involvement for a few extensions to FORTRAN language, which can be translated by a compiler. This approach will treat EXTFOR a new language. In situations when a user wants to avail some features which are not regular in a high level language like FORTRAN, same can be enhanced with a preprocessor. A preprocessor works on the extended features only and translates into the high level language. Thus the second approach is justified here for the translation of EXTFOR programs.

3.2 AN OVERVIEW OF EXTFOR PREPROCESSOR

The EXTFOR preprocessor which will be referred as "EXTFOR" is a program written in FORTRAN, translates programs of EXTFOR language defined by grammar in section 2.7, to programs expressed in FORTRAN as object language without changing the logical flow of the program. The preprocessor functions as an interface between the source of EXTFOR program and the FORTRAN compiler as shown in the figure.

FIG. 3.2a

To translate a source program, the preprocessor must analyze it thoroughly and synthesize an equivalent object program.

During preprocessing that is EXTFOR to FORTRAN translation the process of source program analysis yields a variety of information about source program. This information is preserved by the preprocessor for it to produce an equivalent Fortran program. Various data structures, such as tables, lists, stacks etc. are employed by the preprocessor to preserve this information. The construction of an equivalent Fortran program is directed by the information preserved in these data structures.

The source program to preprocessor may be on any media such as card deck, disk, magnetic tape etc. The outputted object FORTRAN program is written by preprocessor on disk so as to make fast reading by FORTRAN compiler for the compilation into machine language. The complete process of preprocessing involves two steps, source program analysis and code generation.

The following figure (FIG.3.2) illustrates the logical parts of the EXTFOR preprocessor.

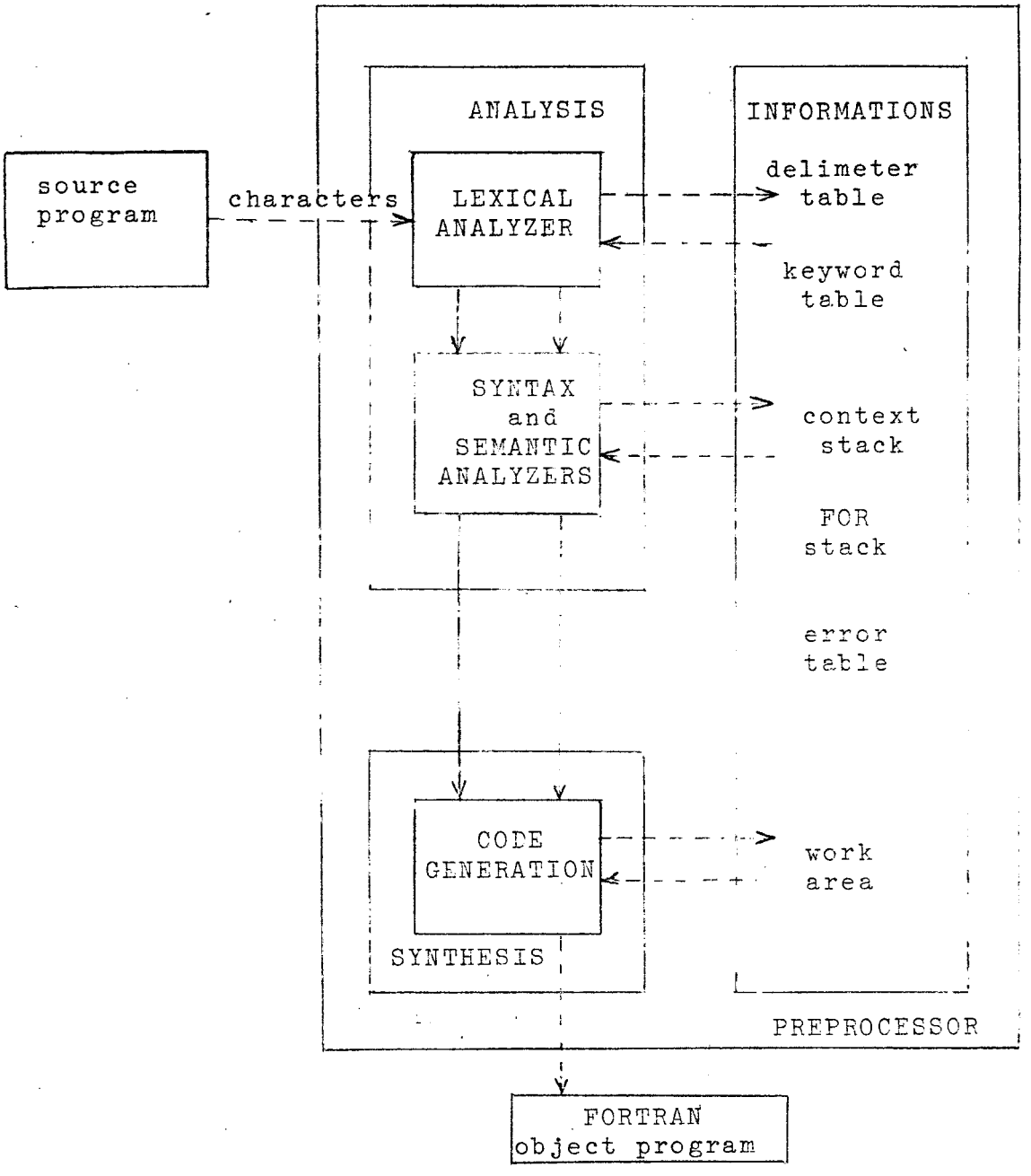


FIG. 3.2 Logical Parts of Preprocessor

3.2.1 SOURCE PROGRAM ANALYSIS

As input to the EXTFOR, the source program is only a string of characters. From this linear representation of the program, the process of analysis should detect the structure and meaning of the program. This is very similar to the actions involved in finding constituent phrases in a sentence in English, in which case we do so by making use of the English grammar. From this analogy, the grammar defined in section 2.7 is used to recognize and formulate the constituents of the source program. The structural analysis of source program proceeds in two logical phases.

FIRST PHASE - LEXICAL ANALYSIS

The first phase consists of the analysis of the string of characters in the source program so as to form meaningful primitives (analogous to words and punctuation marks in an English sentence). Examples of such meaningful primitives are identifiers, operators like '+', '-', '*' etc. and separators or delimiters like ",", blank 'b', etc. The task of first phase is usually termed lexical analysis. Subsequent to this phase of analysis, a source program may be viewed as a sequence of meaningful primitive items referred to as lexemes or tokens.

SECOND PHASE - SYNTAX & SEMANTIC ANALYSIS

Following lexical analysis is the task of recognising grammatical phrases in a source program. This is a more complicated task; sequences of tokens are grouped together to form simple structures of the source language;

these simple structures are used to form more complex ones and, ultimately, source programs. All these actions are the substance of syntax analysis also called parsing. The end result of this analysis is that we have discerned the grammatical structure of the source program. The routine called parser, analyses the grammatical structure of input, recognizes the statement of EXTFOR by seeing the first token of the statement. For instance, when an IF is seen the parser calls a routine which handles IF statements, which does the further semantic checking of the statement. When the parser recognizes a source language construct it calls a so called semantic procedure which takes the construct, checks it for semantic correctness that is the meaning part of the construct, and stores necessary information about it into context stack. This stack contains information about the scope, recursion and nesting of structures. The semantic procedure of the preprocessor is very simple since the most of the semantic checks are performed by the FORTRAN compiler at compilation level. After the syntax and semantic checks the next step is the code generation.

3.2.2 CODE GENERATION

This is the actual translation of the EXTFOR source program into FORTRAN language. In this section we will discuss the FORTRAN codes for the structures selected for EXTFOR. The preprocessor while generating codes generates statement labels, therefore a label slot 90000 to 99999 is reserved and the users are restricted to use them. In this section anything enclosed in square brackets [and] is optional.

BEGIN statement

If a BEGIN is encountered with a label then we replace BEGIN by

```
Label  CONTINUE
```

No code is generated for unlabelled BEGIN. A label L is reserved and when an END terminates the BEGIN block, it is replaced by

```
L      CONTINUE
```

If a BREAK or NEXT statement is enclosed by a BEGIN and the level of loop/structure corresponds to BEGIN then the statement BREAK or NEXT is replaced by

```
GO TO L
```

IF statement

The translation of

```
IF (condition) THEN statement
```

is something like

```
IF (condition is not true) go around statement
```

Thus when an IF is encountered, we

isolate the condition part

generate and save some unique label L

output "IF (.NOT.(condition)) GO TO L"

The .NOT. inverts the value of the condition. The statement label with IF (if any) is put as it is. When we get to the end of the statement that follows THEN, there are two possibilities:

If there is no ELSE following, we need output only

```
L      CONTINUE
```

If an ELSE follows, however, we must generate another label and output

```
L1     GO TO  L2
L      CONTINUE
```

to branch around the ELSE part, and then, after whatever

statement follows the ELSE, we output

```
          L2      CONTINUE
```

to terminate the IF-THEN-ELSE construction, the extra label L1 is required because there could be a statement like BREAK, NEXT or GO TO as the THEN part, then FORTRAN compiler flags an obvious error since any statement following unconditional branch has to be labelled. In summary, the code generation for

```
          [label]IF  (condition) THEN statement
```

is

```
          [label] IF (.NOT.(condition)) GO TO L
                  statement
          L       CONTINUE
```

and for

```
          [label] IF (condition) THEN statement1
                                 ELSE statement2
```

is

```
          [label] IF (.NOT.(condition)) GO TO L
                  statement1
          L1      GO TO L2
          L       CONTINUE
                  statement2
          L2      CONTINUE
```

EXTFOR generates three consecutive labels when an IF is seen: L1 is L+1 and L2 is L+2. If one of the labels is not used because there is no ELSE, it doesn't cost anything. As we know that the labels are always L, L+1 and L+2, only one of them need be remembered; the others are deduced by adding 1 and 2 respectively.

DO statement

The EXTFOR DO is a FORTRAN DO without a label. When a DO is encountered, we

isolate the "limits"
generate a label L
output "DO L limits"

No check is performed on "limits" by EXTFOR and left for FORTRAN. Then at the end of DO scope indicated by an END, we output

```
L       CONTINUE
```

Thus the EXTFOR

```
[label] DO "limits"
        statements
        END
```

is translated into

```
[label] DO L "limits"
        statements
L       CONTINUE
L+1     CONTINUE
```

The second CONTINUE has to be produced in case the loop contains a BREAK statement. In this case we generate the second CONTINUE regardless of whether or not there is a BREAK; it is simply too much effort to check. To make out code generation task easier, we take advantage of the fact that FORTRAN compilers are usually quite clever about dealing with unreferenced CONTINUEs.

The enclosed BREAK statement is replaced by

```
        GO TO L+1
```

and the NEXT by

```
        GO TO L
```

which causes next loop iteration to start.

WHILE statement

When a WHILE is encountered, we
isolate the condition part
generate and save 3 unique labels L, L+1 and L+2
output"L+1 "IF (.NOT.(condition)) GO TO L+2"

when we get an END terminating the WHILE, we output

```
L       GO TO L+1
L+2     CONTINUE
```

Thus the EXTFOR

```
[label] WHILE (condition)
        statements
        END
```

is translated into

```
[label] CONTINUE]
L+1     IF (.NOT.(condition)) GO TO L+2
        statements
L       GO TO L+1
L+2     CONTINUE
```

the CONTINUE before 'IF' is for the labelled WHILE as in:

```
10      WHILE (LASTC.LT.73)
        ...
```

The BREAK statement to break the WHILE loop is translated into

```
GO TO L+2
```

L+2 is the exit point of the loop. The NEXT statement terminating the current iteration and starting the next is translated as

```
GO TO L+1
```

FOR statement

When a FOR statement

```
[label] FOR (initialize; (condition); reinitialize)
                statement
```

is encountered, we

output [label] initialize
isolate the condition part
preserve the reinitialize part
output "L IF (.NOT.(condition)) GO TO L+2"

Then at the end of the statement associated with FOR, we output

```
L+1     reinitialize
        GO TO L
L+2     CONTINUE
```

Thus the code generation for FOR statement is

```
[label] initialize
L       IF (.NOT.(condition)) GO TO L+2
        statement
L+1     reinitialize
        GO TO L
L+2     CONTINUE
```

Three labels L, L+1 and L+2 are reserved for a FOR statement. Here we note that reinitialize can not be a branch statement since GO TO L is unlabelled.

If "statement" in FOR is a BREAK which causes exit from FOR structure, it is translated into

```
GO TO L+2
```

similarly the NEXT statement which transfers control to the reinitialize part is translated into

```
GO TO L+1
```

REPEAT statement

When a REPEAT is encountered, we
output L CONTINUE

Then at the end of the loop which is identified by UNTIL (condition), we
isolate the condition part, if present
output "L+1 IF (condition) GO TO L
L+2 CONTINUE"

Thus the

```
[label] REPEAT
        statements
        UNTIL [(condition)]
```

is translated into

```
[label  CONTINUE]
L       CONTINUE
        statements
L+1     [IF (condition)] GO TO L
L+2     CONTINUE
```

The first CONTINUE is generated only if REPEAT is labelled. In case the loop is infinite that is condition part is omitted "L+1 IF (condition) GO TO L" is replaced by "L+1 GO TO L". Three generated labels are reserved for a REPEAT statement. The BREAK statement encountered in the REPEAT loop to cause the exit from loop, is translated into

```
GO TO L+2
```

similary NEXT statement causing control transfer to the UNTIL part,is translated into

```
GO TO L+1
```

Labels and Others

The statement label given by the user in EXTFOR source program is output at column 1 and followed by enough blanks that the next character will come in column 7. Input statement of lexical type OTHERS is copied from input to output without any change.

The error detecting abilities of EXTFOR are not as good as they might be with a more comprehensive grammar. This is not a serious drawback, however, since we are translating into FORTRAN, and FORTRAN compilers are perfectly capable of detecting any syntax errors that escape the preprocessor. The errors detected by EXTFOR are output as comments in the listing of program.

3.3 EXTFOR PREPROCESSOR ORGANISATION

The organisation of EXTFOR preprocessor is shown in FIG.3.3. The top level routine PARSER controlsby analyzing the grammatical structure of EXTFOR source program as input it sees. It employs other routines of lower level to help in analyzing, storing informations and finally outputting the FORTRAN code.

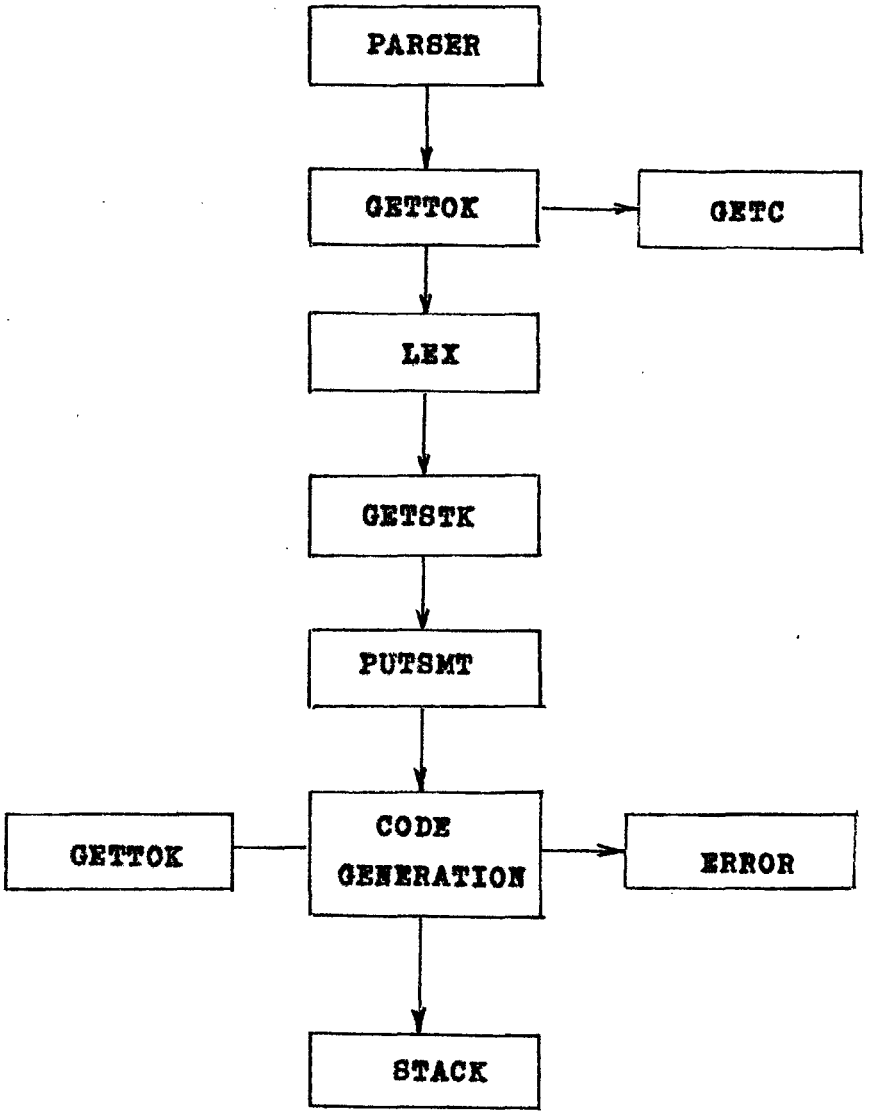


FIG. 3.3

The PARSER calls GETTOK to get the next token of the statement in the input buffer. GETC is the lowest level routine which gets a character from the input buffer to GETTOK. GETTOK constructs a token; LEX is called by PARSER to work on the first token of the statement and the token is identified as one of the keywords. This way PARSER determines the statement type. GETSTK routine tells about the context of the current statement which helps in analysis and proper code generation. For instance, when an IF is seen PARSER calls a routine which handles IF statements. That routine in turn isolates the condition part and makes an entry in the context stack with help of STACK. Necessary codes are generated. When the end of the "statement" part of THEN unit is reached, the correct terminating code for an IF can be produced by calling GETSTK to know about the IF and the generated labels reserved for it. This also include dealing with an ELSE if it is present.

PUTSMT outputs the text of the statement presently being processed as a comment for the user's reference. The FORTRAN codes are output just after the text of source, with "++" in columns 73 and 74 of the object line.

There are number of code generation routines called by PARSER; simple codes are generated by PARSER itself. Some code generation routines also use GETTOK to read further parts of the statement being processed. ERROR outputs the appropriate error message detected during the analysis of the current statement. The calling routine passes the error code to ERROR to output the error message from Error Table.

Now, we describe the various buffers, tables and stacks used to store the informations, lexical analysis, parsing, code generation and general purpose routines.

The following discussion is based on the source listing of EXTFOR preprocessor (Appendix A).

3.3.1 BUFFERS AND BLOCKS

A number of buffers are defined in the work area of EXTFOR, used to keep the parts of source program as well as object program for the current processing. The work area is a set of FORTRAN COMMON blocks.

INBLK Input buffer, COMMON block of 80 bytes. It holds one line of source statement at a time. When the line is processed, next line is read into this block.

INPTR A block which contains the pointer to the last character referred in INBLK.

TKN Holds the current token under processing. The token is constructed by GETTOK.

TKSIZE It contains the number of characters in the token TKN.

INLBL The common block for the statement label of source statement under processing.

CONBLK The condition buffer to contain the condition part of control structures like IF, WHILE, UNTIL and FOR. The condition part is separated out from the statement and put into this buffer till the code "IF (.NOT.(condition))" is generated.CONBLK can contain as long as 17 lines of condition.

CONPTR A pointer used to scan the condition CONBLK.

ERRNO The serial number of error in a statement, detected by EXTFOR.

NOERR It contains the total number of errors encountered in the source program at preprocessor level.

RTFLAB The next generated label available to EXTFOR for code generation is put in this block by function LABGEN which reserves next N labels by incrementing label in

RTFLAB by N and returning old value in LABGEN. It is referred as LABGEN(N).

3.3.2 KEYWORD TABLE

A table is an ordered set of elements of the same type. The ordering of elements in table is used to access the elements, by means of one or more subscripts. The following keywords are stored in form of a table. Each keyword is associated with a unique number which is returned by function LEX and called the lexical type. The lexical types are given negative values to avoid any confusion for LEX, with a printable character.

Keyword	Lexical Type
IF	-30
DO	-31
WHILE	-32
BEGIN	-33
FOR	-34
REPEAT	-35
ELSE	-36
END	-37
NEXT	-38
BREAK	-39
UNTIL	-40
OTHERS	-41

The OTHERS is not a keyword, it corresponds to a token which is not identified as any of the preceding types. This category actually encompasses most of the FORTRAN statements. The function LEX returns -30 as lexical type for IF when referred as LEX(IF).

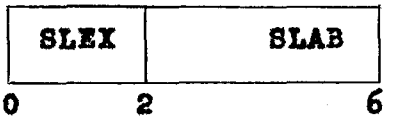
3.3.3 ERROR TABLE

This table contains the text of all the error messages with their codes. The routine ERROR uses this table to output

the error message .

3.3.4 CONTEXT STACK

A stack is a data structure in which elements are arranged in an order having only one end open for addition and deletion of data elements. Stacks are widely used in systems programming for housekeeping functions needed for recursive programs or structures, block structures, and bracketed expressions, etc. The context stack is used for these purposes except the expression processing since this work we leave for FORTRAN compiler. Each entry in context stack is of 6 bytes.



SLEX is the lexical type of the keyword and SLAB is the generated label preserved in the context stack.

Whenever either of the statements IF, ELSE, BEGIN, DO, WHILE, FOR and REPEAT is encountered, the lexical type and the generated label reserved for the structure are pushed on the stack. Whenever a statement ends the context stack is checked for the structure to generate codes for the exit point. For example, when a BEGIN is encountered the lexical type -33 for BEGIN and the generated label L is pushed down in the stack. Later on seeing the matching END statement the context stack is checked; if the stack does not contain BEGIN or anything, an error is flagged for unexpected appearance of END in the source unless it is the last statement of a subprogram. For BEGIN the entry from stack is removed and code "L CONTINUE" as the end point of the block structure is output.

A pointer called stack pointer (sp) is used to point

at the top entry on the stack. To delete an entry the pointer is decreased by 1 and to add sp is incremented by 1 with proper contents on the stack entry. The common block STKBLK defines the area for context stack with a maximum of 100 entries and STKPTR is the stack pointer(sp). The routine STACK pushes down an entry in the stack. The lexical type and the generated label are passed as parameters to STACK. GETSTK gets the top most entry and puts the contents in the arguments SLEX and SLAB.

3.3.5 FOR STACK

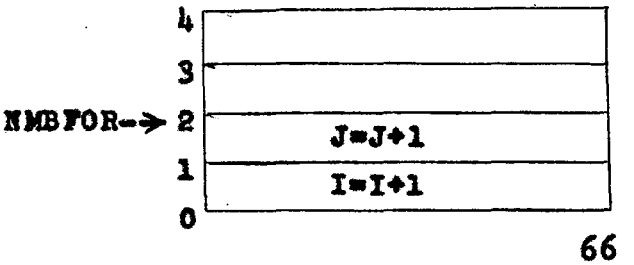
This stack is defined as the COMMON block RIBBLK and the stack pointer NMBFOR. Whenever a FOR statement is encountered the reinitialise part is preserved in this stack, which is used to generate code for end of FOR statement.

The reinitialise part is limited to a length of 66 character i.e. one line of source. The size of this stack is four entries; thus limits the maximum number of nested FOR to 4.

Example:

```
FOR (I=1; (I.LT.100); I=I+1)
    FOR (J=1; (J.LT.10); J=J+1) ....
                                $
```

when the input buffer pointer points at $ the FOR stack looks like



3.3.6 LEXICAL ANALYSIS

A token in EXTFOR is a character string constituated by the elements of FORTRAN character set except the (,), comma and blank. These four characters are significant delimiters for EXTFOR and separate the tokens in the source line. To construct a token GETTOK calls GETC to read the source line; GETC supplies characters one by one to GETTOK. GETTOK checks the type of the characters returned by GETC, if it is a delimiter then the character string formed by GETTOK in TKN is the token. Otherwise, the GETTOK keeps on adding the characters to TKN unless a delimiter or the end of source is encountered. At the end of source, EOF with value -1 is returned by GETC as character to GETTOK; also a flag EOSRC is set, used by PARSER to stop the processing.

Since GETTOK uses blanks to separate tokens, blanks are significant in EXTFOR which is not there in FORTRAN. The keywords like WHILE must not contain blanks, otherwise they won't be recognized.

GETC while reading the source preserves the statement label which is any thing appears before column 6 of the source line, unless it is a continuation line, in COMMON block INLBL. This label is output later on in the translated code. No check is made on the validity of statement label and left for FORTRAN compiler to do the check.

GETTOK also finds out whether the token is the first token of a statement. The function LEX takes this token and determines the lexical type of token by comparing it with all the keywords. This lexical type is used by PARSER in calling the appropriate routine to generate the code.

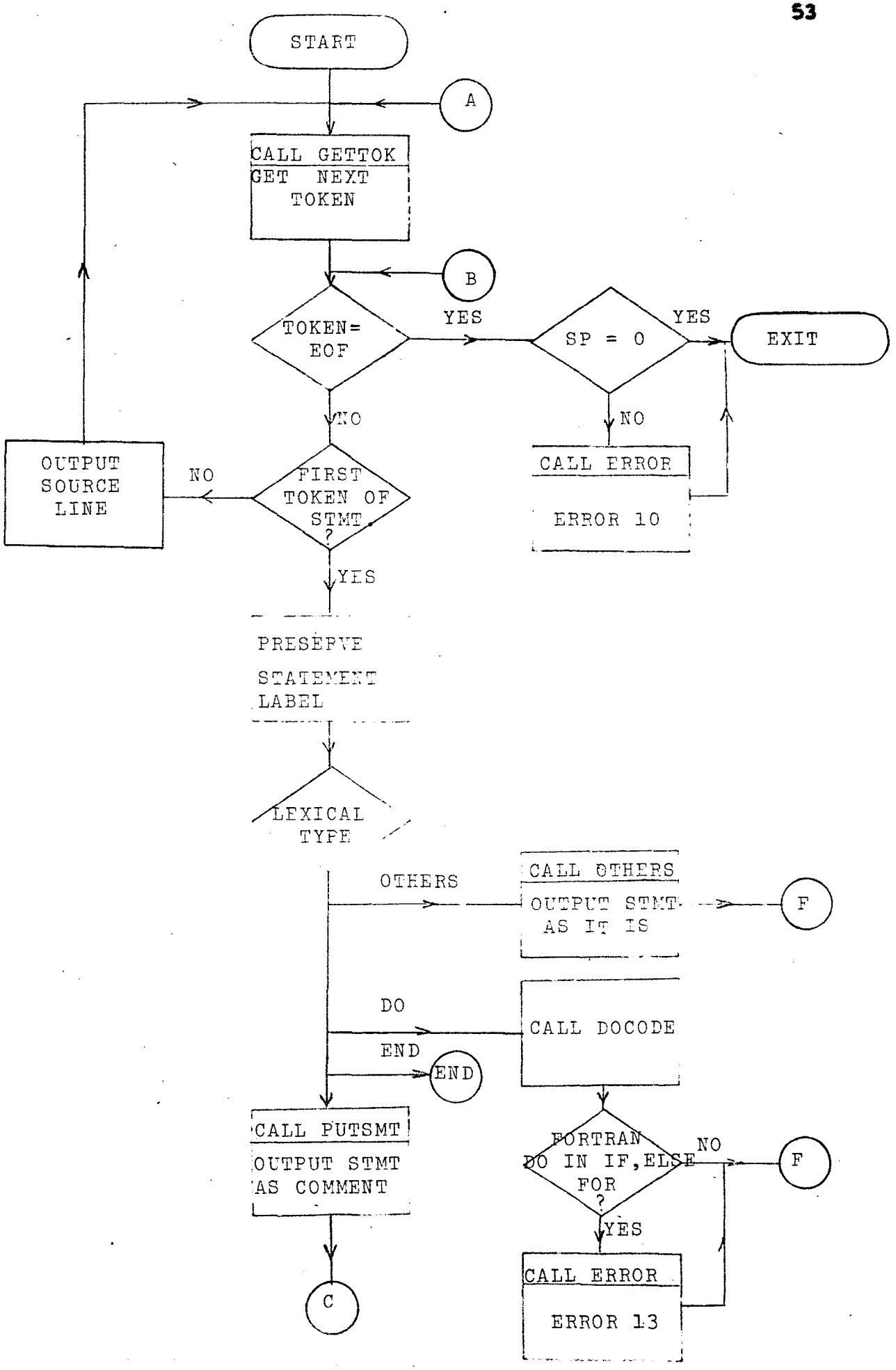
3.3.7 PARSING AND CODE GENERATION

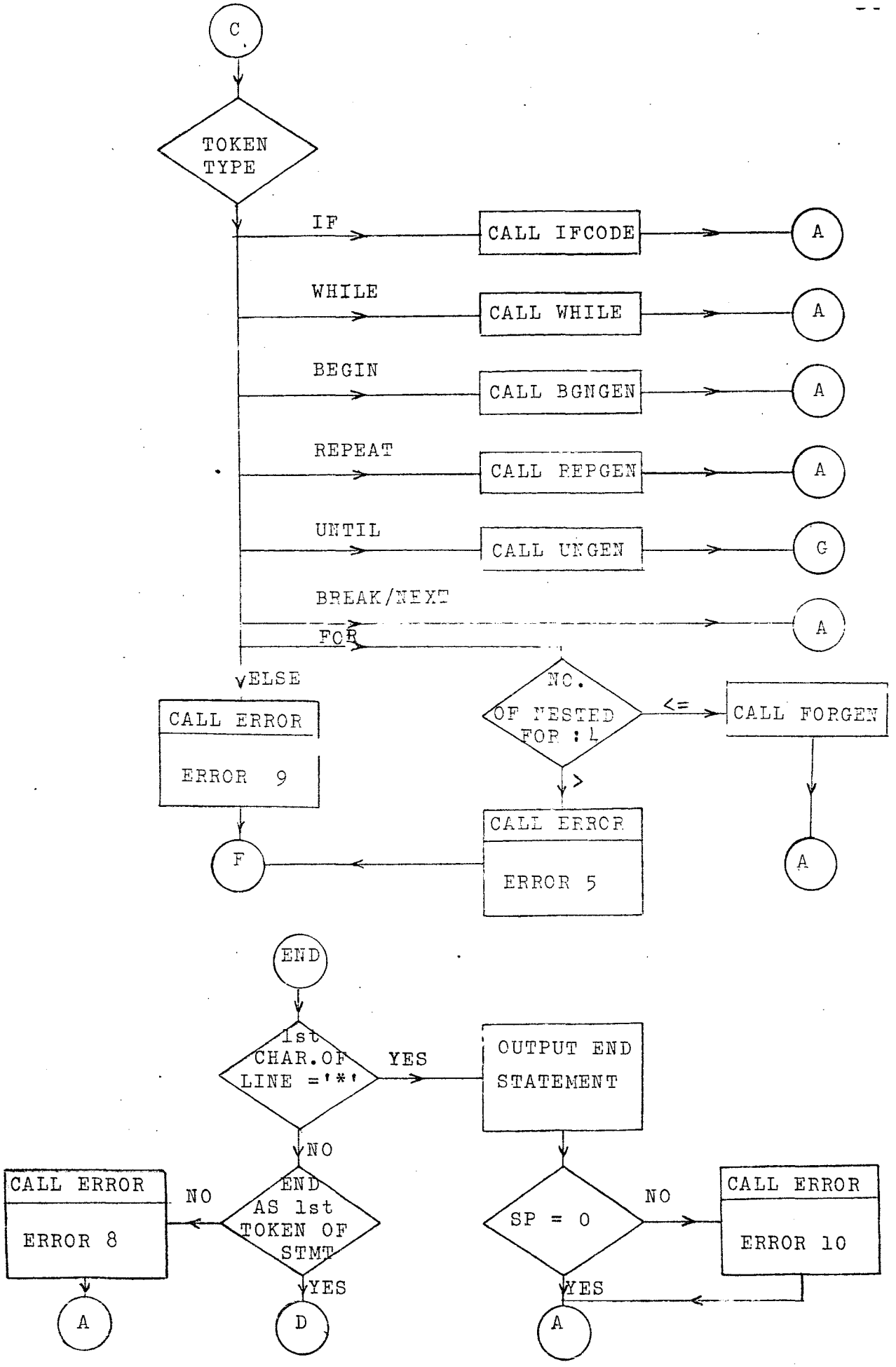
When the beginning of a statement is encountered and recognized as one of the IF, ELSE, BEGIN, DO, WHILE, FOR and REPEAT, the corresponding lexical type and generated label are pushed on the context stack and code generation routine for that type is called after outputting the text of the source statement as a comment. At the end of the statement indicated by type END or BREAK, NEXT, or OTHERS, an appropriate code generation routine is called. The PARSER also checks the stack entry at the top and may be able to pop one or more things of the stack depending upon the lexical type on the stack. For example in

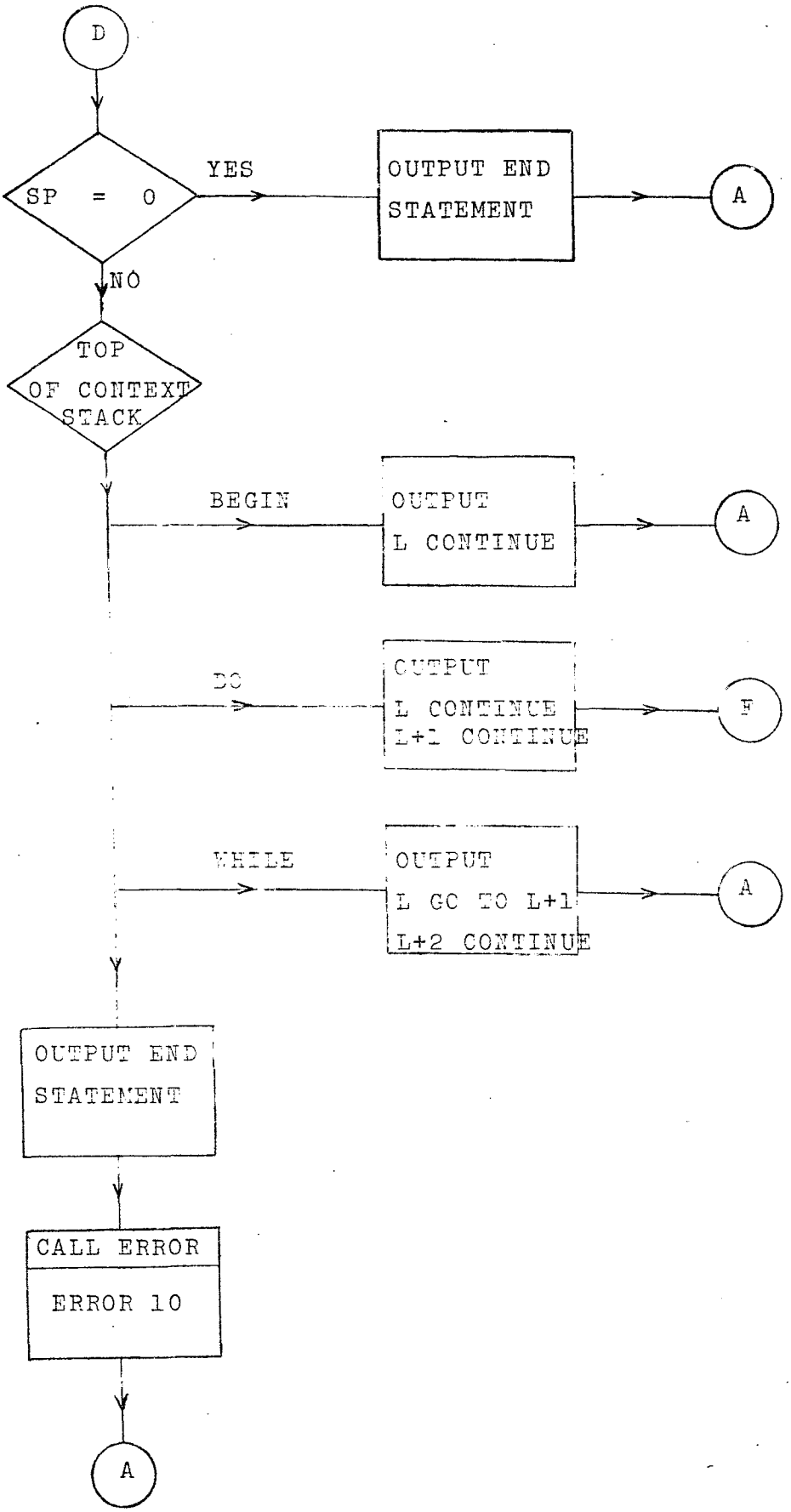
```
IF (cond-1) THEN
            IF (cond-2) THEN M=M+1
                        ELSE M=M+2
LASTC=LASTC+1
```

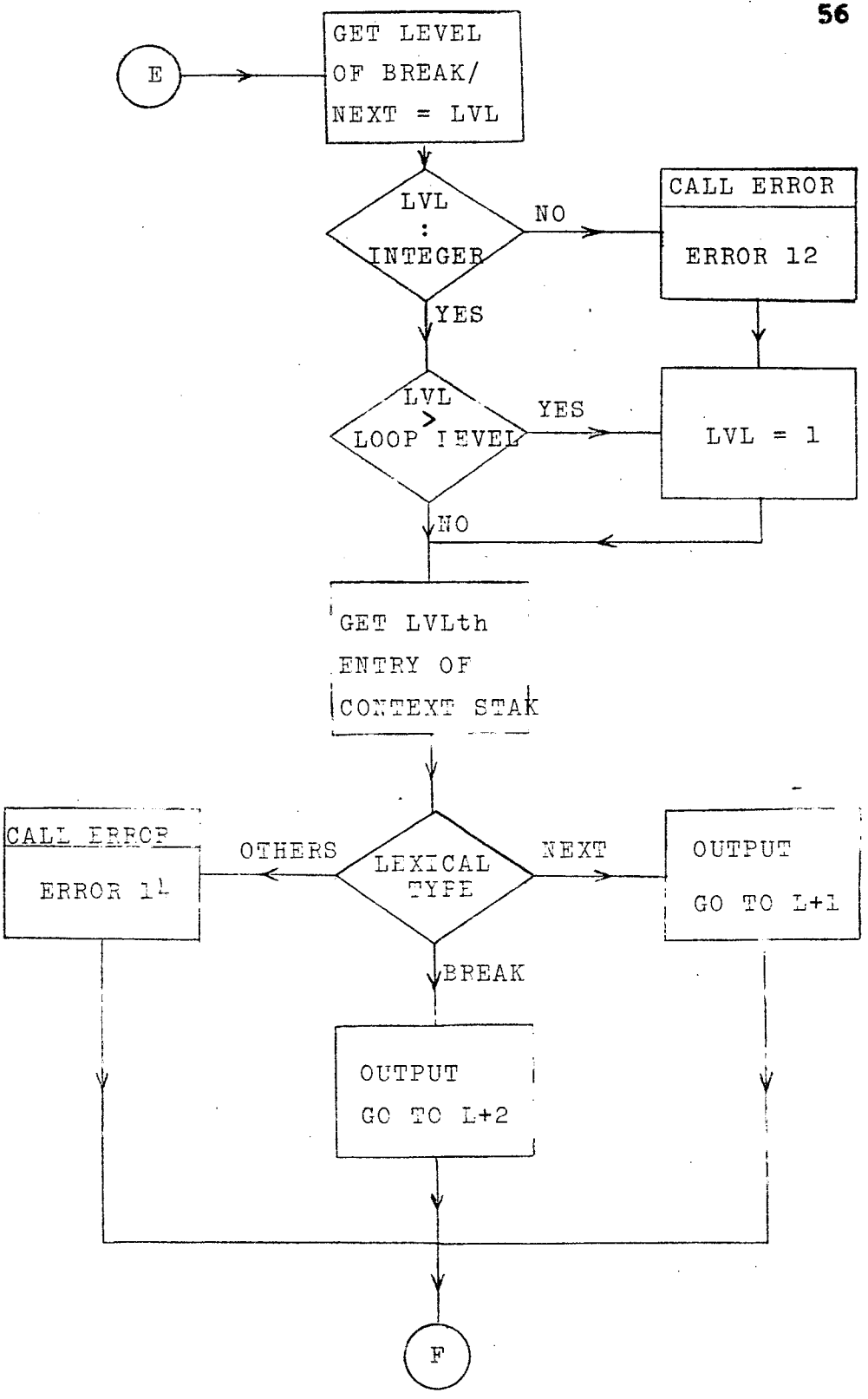
when LASTC=LASTC+1 is seen (lexical type OTHERS), there are entries for ELSE and both IFs on the stack. PARSER pops ELSE and both IFs; because there is no following ELSE matching with first IF, both IF statements are finished. The following ELSE is checked by looking ahead one token.

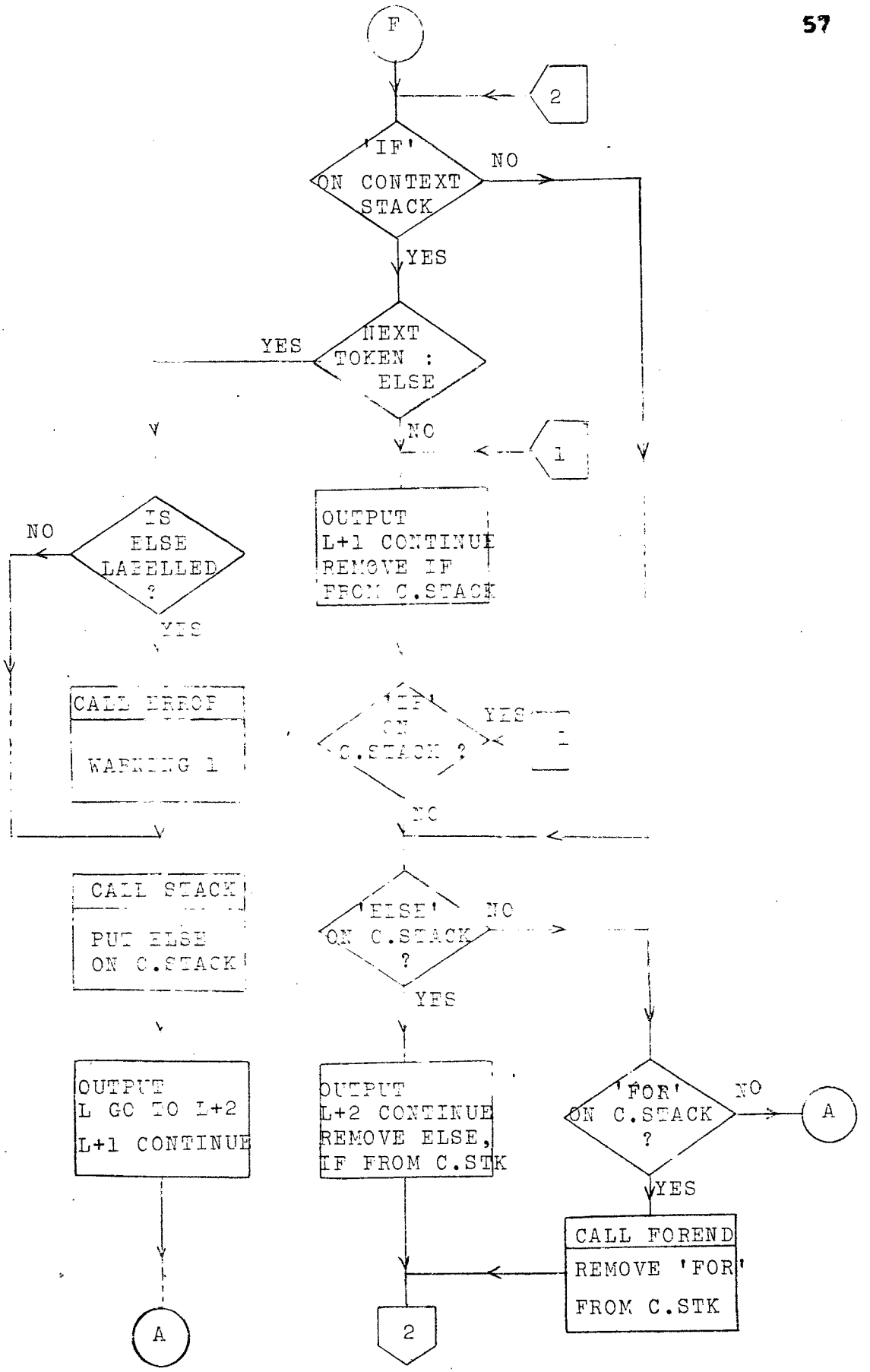
The following flow-chart of PARSER and descriptions of the routines give a close idea how EXTFOR works.


START
A
CALL GETTOK
GET NEXT TOKEN
B
TOKEN= EOF
YES
SP = 0
YES
EXIT
NO
NO
CALL ERROR
ERROR 10
OUTPUT SOURCE LINE
NO
FIRST TOKEN OF STMT. ?
YES
PRESERVE STATEMENT LABEL
LEXICAL TYPE
OTHERS
CALL OTHERS
OUTPUT STMT AS IT IS
F
DO
CALL DOCODE
END
END
CALL PUTSMT
OUTPUT STMT AS COMMENT
FORTRAN DO IN IF,ELSE FOR ?
NO
F
YES
CALL ERROR
ERROR 13
C


C
TOKEN TYPE
IF
CALL IFCODE
A
WHILE
CALL WHILE
A
BEGIN
CALL BGNGEN
A
REPEAT
CALL REPGEN
A
UNTIL
CALL UNGEN
G
BREAK/NEXT
FOR
A
ELSE
CALL ERROR
ERROR 9
F
NO. OF NESTED FOR : L
<=
CALL FORGEN
A
>
CALL ERROR
ERROR 5
END
1st CHAR.OF LINE ='*'
YES
OUTPUT END STATEMENT
NO
CALL ERROR
ERROR 8
NO
A
END AS 1st TOKEN OF STMT
YES
D
SP = 0
NO
CALL ERROR
ERROR 10
YES
A


D
SP = 0
YES
OUTPUT END STATEMENT
A
NO
TOP OF CONTEXT STACK
BEGIN
OUTPUT L CONTINUE
A
DO
OUTPUT L CONTINUE L+1 CONTINUE
F
WHILE
OUTPUT L GO TO L+1 L+2 CONTINUE
A
OUTPUT END STATEMENT
CALL ERROR
ERROR 10
A


E
GET LEVEL OF BREAK/ NEXT = LVL
LVL : INTEGER
NO
CALL ERROR
ERROR 12
YES
LVL > LOOP LEVEL
YES
LVL = 1
NO
GET LVLth ENTRY OF CONTEXT STAK
CALL ERROR
ERROR 14
OTHERS
LEXICAL TYPE
NEXT
OUTPUT
GO TO L+1
BREAK
OUTPUT
GO TO L+2
F


F
2
'IF' ON CONTEXT STACK
NO
YES
NEXT TOKEN : ELSE
YES
NO
1
OUTPUT L+1 CONTINUE REMOVE IF FROM C.STACK
IS ELSE LABELLED ?
NO
YES
CALL ERROR
WARNING 1
'IF' ON C.STACK ?
YES
NO
CALL STACK
PUT ELSE ON C.STACK
'ELSE' ON C.STACK ?
NO
YES
OUTPUT L GO TO L+2
L+1 CONTINUE
OUTPUT L+2 CONTINUE REMOVE ELSE, IF FROM C.STK
'FOR' ON C.STACK ?
NO
YES
A
CALL FOREND
REMOVE 'FOR' FROM C.STK
A
2

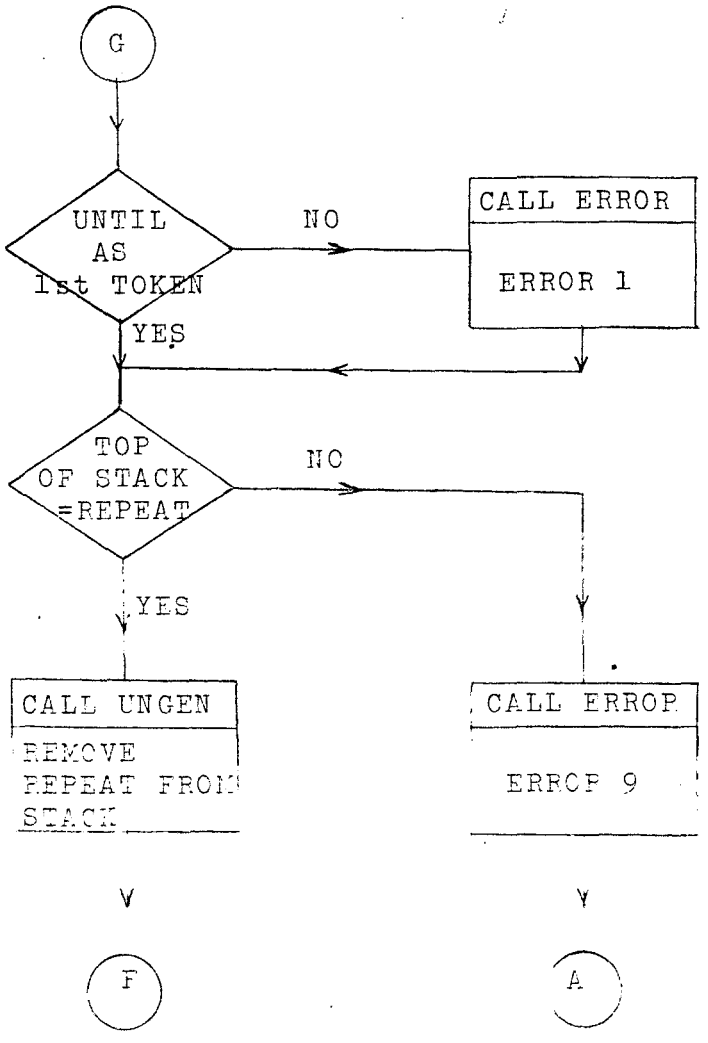


FIG:3.3.6 Flow Chart of PARSEP

ROUTINE DESCRIPTIONS

ERROR This routine outputs the error messages (Appendix D) when the error code is supplied as a parameter.

OTHERS This routine outputs the statement which is not any of the extended features.

DOCODE This routine processes DO statements. The token following 'DO' is checked, if it is an integer number then the DO is a FORTRAN DO and it is output as such. Otherwise, the 'limits' which follows DO is collected, two labels L and L+1 are generated and DOCODE outputs

DO L 'limits'

BGNGEN This routine generates one label L for the BEGIN. If BEGIN statement is labelled "Label CONTINUE" is output. The lexical type of BEGIN is put on the context stack.

BALPAR This routine collects the condition part of the control structure, which is a string enclosed in balanced parentheses. If this is spreaded over several lines BALPAR handles continuations upto 17 lines and puts in COMMON block CONBLK with the number of characters in condition, in COMMON block CONPTR.

PUTIF This routine is referred as CALL PUTIF (TYPE, LAB). In case value of TYPE is passed as 1 PUTIF outputs

IF (.NOT.(condition)) GO TO LAB

when TYPE=0, PUTIF outputs

IF (condition) GO TO LAB

IFGO The routine processes the condition part of IF, WHILE, UNTIL and FOR statements. If IFGO is called with option CNTRL=1, it checks for FORTRAN logical and arithmatic IF statements and outputs without any change. With option CNTRL=0, it calls BALPAR and PUTCND to put the condition part in condition buffer CONBLK and,then calls PUTIF to generate the code

[label] IF (.NOT.(condition)) GO TO L

IFCODE Routine is called to process the IF statement. Three generated labels are reserved and IFGO is called. If the statement is found to be anyone of the FORTRAN IF statements control is returned to PARSER after outputting the statement as it is. For 'IF' statement the code

IF (.NOT. (condition)) GO TO L

is output and lexical type IF and the label L are put on the context stack.

WHILE Routine generates FORTRAN code for WHILE statement. Three generated labels are reserved for each appearance of WHILE. Routine outputs

```
label  CONTINUE
```

if statement is labelled and calls IFGO to output

```
L+1    IF (.NOT.(condition)) GO TO L+2
```

The lexical type WHILE and label L are put on the context stack.

FORGEN This routine processes the statement:

FOR (initialize; (condition); reinitialize) statement

The procedure followed is:

Call PUTCND and BALPAR to collect source between outermost parentheses, in condition buffer CONBLK.

Scan CONBLK to find the positions of semicolons as PTR1 and PRT2; if number of semicolons is not equal to 2 then, call ERROR(6) and output the statement; return control to PARSER.

Output the initialize part which appears before first semicolon as [label] initialize

Push down the reinitialize part which appears after second semicolon, on FOR stack.

Shift the condition part which appears between two semicolons i.e. PTR1 and PTR2 positions, to the first character position of CONBLK and set pointer CONPTR=PTR2-PTR1.

Call PUTIF to output

```
L      IF (.NOT.(condition)) GO TO L+2
```

Stack the lexical type FOR and generated label L.
RNturn control to PARSER.

FOREND This routine generates FORTRAN code for end of FOR structure. The code output are

```
L+1     reinitialize
        GO TO L
L+2     CONTINUE
```

The reinitialize part is taken from FOR stack and stack pointer is decremented by 1. The FOR entry from context is also removed.

REPGEN The routine reserves three generated labels for each appearance of REPEAT and outputs

```
Label   CONTINUE   (if REPEAT is labelled)
L       CONTINUE
```

The lexical type REPEAT and generated label L are stacked.

UNGEN Processes UNTIL part of REPEAT structure. In case UNTIL is preceded by label ERROR(1) is flagged. If there is no token which follows UNTIL, the loop is indefinite and UNGEN outputs

```
L+1     GO TO L
L+2     CONTINUE
```

returns the control to PARSER.
For finite loop UNGEN calls BALPAR and PUTCND to collect the condition part of UNTIL. Finally PUTIF is called to output

```
        IF (condition) GO TO L+2
L+1     GO TO L
L+2     CONTINUE
```

The entry for REPEAT from context stack is removed and control is returned to PARSER.

There are some small routines called by PARSER, their functions and algorithms are quite simple, one can refer EXTFOR source listing (Appendix A).

EFFICIENCY AND OVERHEADS IN PREPROCESSOR

4.1 EFFICIENCY

The efficiency increases considerably while writing programs in EXTFOR but the additional translation step from EXTFOR to FORTRAN increases the cost of producing an executable version of the program.

One of the major factors affecting the efficiency of preprocessor is the features selected, to be translated by preprocessor. More sophisticated features and lot of improvements attribute to the inefficiency of preprocessor. The best way is to select a small set of improvements, depending on the user's requirements. A better design and implementation also make the preprocessor an efficient tool.

The first FORTRAN preprocessor RATFOR developed by Dr Kernighan is found to be extremely inefficient. The cost ratios of a RATFOR and FORTRAN programs in terms of CPU times, are from 10:1 for small programs to 19:1 for RATFOR compiling itself. During the final testing of EXTFOR and the test programs attached in Appendix C, it is found that same ratio for EXTFOR is 1.1:1 to 2:1; a substantial improvement in efficiency of preprocessor.

The efficiency can be improved further by identifying some of the routines as 'high spot' in preprocessor and record them to impsove efficiency. Dr Kernighan measured that about 60% time is spent doing input and output at the lowest level. A major part of remaining 40% time is shared by GETTOK. We might impsove the efficiency by concentrating on the better design of these

'high spot' routines. User also has to pay penalty in terms of run-time efficiency of the code produced by the preprocessor and the storage requirements.

The straightforward translation process from EXTFOR to FORTRAN makes little attempt to produce optimal FORTRAN code. Sometimes unnecessary GOTOs and CONTINUEs are produced and left for FORTRAN compiler to optimize and avoid the redundancy in FORTRAN code. In the early stages of the idea of FORTRAN preprocessor, it was thought that preprocessed and hand coded versions of a program would not display significantly different time and space characterstics if both were compiled using a good optimising compiler even if the program involved a moderate amount of input/output. Later on, a number of experiments were done on several preprocessors, compilers, and machines and following facts were concluded.

Two factors would affect overall time and space characteristics - the quality of optimization performed by the FORTRAN compiler and the amount of input/output performed.

4.2 EXECUTION TIME OVERHEAD

With moderate input/output activities, the run-time overhead incurred by using a preprocessor is insignificant. With output removed, the results change considerably. The execution time overhead ranges from 5 to 30 percent. When a compiler's optimisation is turned on interesting things were observed. For programs with a moderate amount of output, there is an insignificant change in execution time overhead with improved optimization. Surprisingly, in several cases the overhead

is increased by optimising the code. This behaviour is found to be different for different compilers. With output removed the run-time overhead increases with greater optimisation! Except for a few preprocessors and compilers.

4.3 STORAGE OVERHEAD

The storage penalty incurred by using a preprocessor seems highly dependent on the style of the generated FORTRAN code and on the sophistication of the preprocessor features. A preprocessor with more sophisticated language features has more storage overheads than one with simple features. The presence of large arrays in a program would decrease the storage overhead. With input/output the storage overhead are less than the same for programs without input/output. The compiler's optimization has mixed affects on storage overhead.

The overhead incurred seems highly dependent on both the FORTRAN compilers and the particular characteristics of the generated code. Anyway the use of preprocessors is not discouraged. They offer many advantages in coding over working with FORTRAN code directly. However, it is important for a user to recognise all of the costs of using the preprocessor, as well as all of the benefits received.

CHAPTER 5

EXPERIENCE AND CONCLUSION

The preprocessor is a good software tool for enhancing the facilities in a programming language. It is quite obvious that many things can be done with a preprocessor once it is available. For example, the data types of FORTRAN can be improved by adding the CHARACTER type, modifications in EXTFOR to translate the operators =, <, >, ¬, |, >=, <=, ¬=, and & as .EQ., .LT., .GT., .NOT., .OR., .GE., .LE., .NE., and .AND..

When designing a language or enhancing it, it is important to concentrate on the essential requirement of providing the user with the best language possible for a given effort. One must avoid throwing features - things which the user may trivially construct within the existing framework. For example, the REPEAT statement which is a loop with its test at the bottom. This statement encourages programs which fail at their boundries. In the few cases where it is needed, it can be easily simulated with an infinite loop and a test and break at the bottom.

The main problem faced while using preprocessor is the fact that errors detected by the compiler are reported with respect to the preprocessor generated program, not the program written by the user. However, the readability of the program, easier maintenance and modification, and the reduction in programming time by structured programming are not disputable with difficulty faced because of bad error detection. While designing EXTFOR we concentrated on this aspect.

The PARSER is made better to recover from invalid syntax like ELSEs without IFs, UNTILs without REPEAT, etc. The situation where a matching END is not encountered to terminate a BEGIN,WHILE or DO loop, becomes embarassing because the END which is also used as the last statement of a subprogram, is assumed to be associated with the loop or block and the next subprogram becomes the part of the first. This unrecovered error keeps on communicating; to avoid this EXTFOR provides the user to specify '*' in the first column of the line with END statement, used as the last statement of subprogram; seeing it, EXTFOR terminates the scope of current subprogram. If the context stack is not empty it is made so and an error is reported; processing of next subprogram starts as a fresh with no interaction with errors in previous subprogram.

The error detection capabilities of EXTFOR are improved by tagging each output line of object code with the source line of extended feature, that created the code. The source lines of an extended feature are listed as comment followed by equivalent FORTRAN code generated by EXTFOR with '++' in columns 73 and 74 of each object line. This way user can distinguish between the regular FORTRAN features in his program and the EXTFOR generated code. The errors which come from the FORTRAN compiler in generated codes can be readily associated with their source of extended features. The errors reported by preprocessor are input with the source of EXTFOR appearing as comments in the listing.

The choice of selecting a programming language to write the preprocessor is opened to any language but,

one with better character handling facilities is the ideal choice; preferably PL/I or PASCAL could be selected to improve the efficiency of the preprocessor. Our choice for EXTFOR writing is the FORTRAN because we already have FORTRAN compiler to compile the EXTFOR generated code. The EXTFOR is portable to all computer installations where FORTRAN users want to avail the EXTFOR features.

EXTFOR is written using the modular approach. The program is divided into small subroutines and functions. Subroutines and functions rarely spread over more than two pages; most are much shorter. As a result the code is readable. It is easy to convince that EXTFOR module is probably correct because it is broken up into pieces that one can grasp one at a time. It is tried to make the routines easy to modify by keeping the routines unknown about information that not needed by them. The informations, commonly used by more than two routines are defined in COMMON blocks referred as EXTFOR work area. The less frequently used information is communicated between two routines through parameters.

It is well known that no program comes out to a perfect work of art on its first draft, regardless of the technique one uses to write it. Most of the routines were written, modified and tested several times, yet we still would not claim that any one is flawless. The EXTFOR module is tested thoroughly particularly at boundries.

APPENDIX A

EXTFOR SOURCE LISTING

```
C*************************************************************************
C                                                                        ********
C                    A  FORTRAN  PREPROCESSOR                            ********
C                                                                        ********
C------------------------------------------------------------------------********
C                                                                        ********
C       THIS PROGRAM TRANSLATES 'FORTRAN' WITH EXTENDED FEATURES,        ********
C       INTO 'FORTRAN', THE LANGUAGE IS SPECIFIED BY FOLLOWING GRAMMAR   ********
C                                                                        ********
C       PROGRAM   :  STATEMENT                                           ********
C                    PROGRAM STATEMENT .                                 ********
C       STATEMENT I  IF (CONDITION) THEN STATEMENT                       ********
C                 I  IF (CONDITION) THEN STATEMENT                       ********
C                    ELSE STATEMENT                                      ********
C                 I  BEGIN                                               ********
C                    STATEMENT                                           ********
C                    END                                                 ********
C                 I  DO LIMITS                                           ********
C                    STATEMENT                                           ********
C                    END                                                 ********
C                 I  WHILE (CONDITION) STATEMENT                         ********
C                    END                                                 ********
C                 I  FOR (INITIALIZE;(CONDITION);REINITIALIZE)           ********
C                    STATEMENT                                           ********
C                 I  REPEAT STATEMENT                                    ********
C                    UNTIL < (CONDITION) >                               ********
C                 I  LABEL STATEMENT                                     ********
C                 I  BREAK <LEVEL>                                       ********
C                 I  NEXT  <LEVEL>                                       ********
C                 I  <PROGRAM>                                           ********
```

```
                        *** PARSER ***

C                 I  OTHER                                                  ********
C                                                                           ********
C          THE VERTICAL BAR I INDICATES A CHOICE OF ALTERNATIVES            ********
C          < > INDICATES TO BE OPTIONAL,'OTHER' IS ANYTHING THAT            ********
C          WAS NOT RECOGNIZED AS ANY OF THE PRECEDING TYPES.                ********
C             TO TRANSLATE THE EXTENDED CONTROL STRUCTURCE, AT THE           ********
C          BEGINNING OF EACH STATEMENT THIS PROGRAM CALLS GETTOK TO          ********
C          GET THE FIRST TOKEN OF THE STATEMENT WHICH DETERMINES             ********
C          THE STATEMENT TYPE. AT A TIME 80 CHARACTERS OF A LINE             ********
C          ARE READ INTO IN-BUFFER WITH A SOURCE POINTER LASTC USED          ********
C          TO SCAN THE SOURCE LINE. WHEN THE STATEMENT IS CLASSIFIED,        ********
C          THIS ROUTINE CALLS THE APPROPRIATE CODE GENERATION ROUTINE        ********
C          AND CORRESPONDING TYPE WITH A UNIQUE GENERATED LABEL ARE          ********
C          PUT ON THE CONTEXT STACK IN CASE OF 'IF','ELSE','BEGIN',          ********
C          'DO','WHILE','REPEAT', AND 'FOR'. WHEN THE END OF STATEMENT       ********
C          OR MATCHING 'END' FOR 'DO','BEGIN', AND 'WHILE' IS ENCOUN-        ********
C          TERED APPROPRIATE CODES ARE GENERATED ALSO, ENTRY FROM            ********
C          CONTEXT STACK IS REMOVED. WHEN A LINE IS PROCESSED, NEXT          ********
C          LINE IS READ FOR THE CONTINUATION OF THE STATEMENT OR             ********
C          THE BEGINNING OF NEXT STATEMENT.                                  ********
C          THE EXTENDED CONTROL STATEMENT ARE PRINT AS SUCH STARTING         ********
C          WITH 'C+'  AND GENERATE CODES ENDING WITH '++' IN 72-73           ********
C          COLUMN , APPROPRIATE ERROR MESSAGES ARE PRINTED WHEN AN           ********
C          ERROR IS DETECTED , THE CONDITION PART OF SOME OF THE             ********
C          STATEMENT IS LEFT TO FORTRAN COMPILER TO CHECK IT.                ********
C          THE FIRST GENERATED LABEL IS 900000.                              ********
C                                                                            ********
C                                                                            ********
C                                                                            ********
C*****************************************************************************
```

*** PARSER ***

```
C*************************************************************************
C                                                                        ********
C       PARSER                                                           ********
C                                                                        ********
C........................................................................ ********
      INTEGER ERRNOB,J,LAB,LABS,LABST,LASTC,LEVEL                        PARS0001
      INTEGER NLASTC,NFOR,NERR,OUTP,PTR,SP                               PARS0002
      INTEGER*2 ZERO/0/                                                  PARS0003
      INTEGER*2 STAR/'*'/,BLANK/' '/,ZERO/'0'/,EOS/-2/,EOF/-1/           PARS0004
      INTEGER*2 LEXIF/-30/,LEXDO/-31/,LEXWHL/-32/,LEXBEG/-33/            PARS0005
      INTEGER*2 LEXFOR/-34/,LEXREP/-35/,LEXELS/-36/,LEXEND/-37/          PARS0006
      INTEGER*2 LEXNXT/-38/,LEXBRK/-39/,LEXUNT/-40/,LEXOTR/-41/          PARS0007
      INTEGER*2 LEX,LEXS,LEXST,LEXTKN,NSTTOK,RRNFLG,TKNSWH,IFLAG,I       PARS0008
      INTEGER*2 BUF(80),CLVEL(2),INLAB(5),TOKEN(80)                      PARS0009
C                                                                        ********
C     COMMON BLOCKS                                                      ********
      COMMON /CONPTR/ PTR                                                PARS0010
      COMMON /ERRNO / ERRNOB                                             PARS0011
      COMMON /EOSRC / TEOSRC                                             PARS0012
      COMMON /INPTR / LASTC                                              PARS0013
      COMMON /INBLK / BUF                                                PARS0014
      COMMON /INLBL / INLAB                                              PARS0015
      COMMON /NSTFLG/ NSTTOK                                             PARS0016
      COMMON /NMBFOR/ NFOR                                               PARS0017
      COMMON /NOERR / NERR                                               PARS0018
      COMMON /NLST  / NLASTC                                             PARS0019
      COMMON /OUTPTR/ OUTP                                               PARS0020
      COMMON /RNFLG / RRNFLG                                             PARS0021
      COMMON /RTFLAB/ LAB                                                PARS0022
      COMMON /STKPTR/ SP                                                 PARS0023
```

*** PARSER ***

```
      COMMON /SYSIN / ISYSIN                                            PARS0024
      COMMON /SYSOUT/ ISYOUT                                            PARS0025
      COMMON /TKN   / TOKEN                                             PARS0026
C                                                                       ********
C     NERR IS NUMBER OF DIAGNOSTICS GENERATED.                          ********
C     ERRNOB IS NUMBER OF ERROR IN A STATEMENT.                         ********
C     IEOSRC IS END OF SOURCE FLAG.                                     ********
C     SP IS STACK POINTER.                                              ********
C     LASTC IS SOURCE BUFFER POINTER.                                   ********
C     PTR   IS CONDITION BUFFER POINTER.                                ********
C     NFOR  IS NO. OF NESTED FOR.                                       ********
C     LAB   IS GENERATED LABEL.                                         ********
C     RRNFLG IS RERUN FLAG.                                             ********
C     TKNSWH IS TOKEN SWITCH.                                           ********
C     NSTTOK IS NEW STATEMENT TOKEN.                                    ********
C                                                                       ********
      ISYSIN=1                                                          PARS0027
      ISYOUT=3                                                          PARS0028
      NERR=0                                                            PARS0029
      ERRNOB=0                                                          PARS0030
      IEOSRC=0                                                          PARS0031
      SP=0                                                              PARS0032
      LASTC=72                                                          PARS0033
      OUTP=6                                                            PARS0034
      PTR=14                                                            PARS0035
      NFOR=0                                                            PARS0036
      LAB=90000                                                         PARS0037
      RRNFLG=0                                                          PARS0038
      TKNSWH=0                                                          PARS0039
      GO TO 50                                                          PARS0040
```

*** PARSER ***

```
5       NLASTC=LASTC+1                                              PARS0041
C       GET THE NEXT TOKEN AND CHECK FOR A NEW STATEMENT            ********
        CALL GETTOK                                                 PARS0042
        IF(TOKEN(1).EQ.EOF) GO TO 3000                              PARS0043
        IF(NSTTOK.EQ.0) GO TO 158                                   PARS0044
        LASTC=6                                                     PARS0045
        GO TO 2001                                                  PARS0046
10      RRNFLG=0                                                    PARS0047
20      IF (LASTC.EQ.6) GO TO 50                                    PARS0048
40      LASTC=72                                                    PARS0049
50      NSTTOK=0                                                    PARS0050
        NLASTC=1                                                    PARS0051
100     CALL GETTOK                                                 PARS0052
        NLASTC=1                                                    PARS0053
        IF (TOKEN(1).EQ.EOF) GO TO 3000                             PARS0054
C       CHECK FOR THE CONTTNUATION OF THE STATEMENT AND             ********
C       PRINT IF SO.                                                ********
        IF (NSTTOK.NE.0) GO TO 150                                  PARS0055
        CALL PUTLIN(BUF)                                            PARS0056
        GO TO 40                                                    PARS0057
C       STATEMENT LABEL.                                            ********
150     DO 155 I=1,5                                                PARS0058
        INLAB(I)=BUF(I)                                             PARS0059
155     CONTINUE                                                    PARS0060
158     BUF(6)=BLANK                                                PARS0061
        CALL GETSTK (LEXS,LABS)                                     PARS0062
160     LEXTKN=LEX(TOKEN)                                           PARS0063
170     IF (TOKEN(1).EQ.EOF) GO TO 3000                             PARS0064
        IF (LEXTKN.EQ.LEXOTR) GO TO 1200                            PARS0065
        IF (LEXTKN.EQ.LEXDO ) GO TO 300                             PARS0066
```

*** PARSER ***

```
      IF (LEXTKN.EQ.LEXEND) GO TO 900                                   PARS0067
C     PRINT OUT THE STATEMENT AS A COMMENT.                             ********
      CALL PUTSMT(NLASTC)                                               PARS0068
      I=-(LEXTKN)-29                                                    PARS0069
C     BRANCH TO PROCESS THE STATEMENT DEPENDING                         ********
C     UPON THE THE FIRST TOKEN OF THE STATEMENT.                        ********
      GO TO(200,300,400,500,600,700,800,900,1000,1000,1100,1200),I      PARS0070
C                                                                       ********
C*    IF STATEMENT                                                      ********
200   CALL IFCODF                                                       PARS0071
      GO TO 2000                                                        PARS0072
C                                                                       ********
C*    DO STATEMENT                                                      ********
300   CALL DOCODF (I,NLASTC)                                            PARS0073
      IF ((I.EQ.ZERO).OR.((LEXS.NE.LEXIF)                               PARS0074
     *  .AND.(LEXS.NE.LEXELS).AND.                                       PARS0075
     *  (LEXS.NE.LEXFOR))) GO TO 10                                      PARS0076
C     ERROR--'FORTRAN DO STATEMENT ILLEGAL HERE'                        ********
      CALL ERROR (13)                                                   PARS0077
      GO TO 2001                                                        PARS0078
C                                                                       ********
C*    WHILE STATEMENT                                                   ********
400   CONTINUE                                                          PARS0079
      CALL WHILE                                                        PARS0080
      BRNFLG=1                                                          PARS0081
      GO TO 5                                                           PARS0082
C                                                                       ********
C*    BEGIN STATEMENT                                                   ********
500   CONTINUE                                                          PARS0083
      CALL CHKBLK                                                       PARS0084
```

*** PARSER ***

```
      CALL BGNGEN                                                    PARS0085
      GO TO 10                                                       PARS0086
C                                                                    ********
C*    FOR STATEMENT                                                  ********
600   CONTINUE                                                       PARS0087
      IF(NFOR.LT.4) GO TO 650                                        PARS0088
C     ' TOO MANY NESTED 'FOR' STATEMENTS'                            ********
      CALL ERROR (5)                                                 PARS0089
      GO TO 2001                                                     PARS0090
650   CONTINUE                                                       PARS0091
      CALL FORGEN                                                    PARS0092
      GO TO 2000                                                     PARS0093
C                                                                    ********
C*    REPEAT STATEMENT                                               ********
700   CONTINUE                                                       PARS0094
      CALL REPGEN                                                    PARS0095
      RRNFLG=1                                                       PARS0096
      GO TO 5                                                        PARS0097
C                                                                    ********
C*    ELSE STATEMENT                                                 ********
800   CONTINUE                                                       PARS0098
C     ERROR= 'UNM
      CALL ERROR (9)                                                 PARS0099
      GO TO 2001                                                     PARS0100
C                                                                    ********
C*    END STATEMENT                                                  ********
900   CONTINUE                                                       PARS0101
      IF (RRNFLG.NE.ZERO) GO TO 910                                  PARS0102
      IF (BUF(1).NE.STAR) GO TO 950                                  PARS0103
901   WRITE(ISYOUT,902) (BUF(I),I=2,80)                              PARS0104
```

*** PARSER ***

```
902     FORMAT (1X,79A1)                                                PARS0105
        IF(SP.EQ.0) GO TO 10                                            PARS0106
C       ERROR--'UNEXPECTED END OF SUBPROGRAM'                           ********
        CALL ERROR (10)                                                 PARS0107
        SP=0                                                            PARS0108
        GO TO 10                                                        PARS0109
905     IF ((LEXS.NE.LEXIF).AND.(LEXS.NE.LEXREP)                        PARS0110
     1     .AND.(LEXS.NE.LEXFOR)) GO TO 950                             PARS0111
910     CALL PUTSMT (NLASTC)                                            PARS0112
C       ERROR--'CONDITIONAL END STATEMENT'                              ********
915     CALL ERROR (8)                                                  PARS0113
        GO TO 2001                                                      PARS0114
950     IF (SP.NE.0) GO TO 952                                          PARS0115
        CALL PUTLIN(BUF)                                                PARS0116
        GO TO 40                                                        PARS0117
952     CALL PUTSMT(NLASTC)                                             PARS0118
        CALL CHKBLK                                                     PARS0119
        IF ((LEXS.EQ.LEXFOR).OR.                                        PARS0120
     1     (LEXS.EQ.LEXREP)) GO TO 915                                  PARS0121
C       REMOVE ENTRY FROM STACK                                         ********
        SP=SP-1                                                         PARS0122
        I=-(LEXS)-30                                                    PARS0123
        GO TO (960,970,965),I                                           PARS0124
C                                                                       ********
C*      END OF BEGIN/DO UNIT                                            ********
C       GENERATE 'LABS+1  CONTINUE'                                     ********
C                'LABS    CONTINUE'                                     ********
960     CALL OUTCON (LABS+1)                                            PARS0125
965     CALL OUTCON(LABS+2)                                             PARS0126
        GO TO 2001                                                      PARS0127
```

*** PARSER ***

```
C                                                                       ********
C*      END OF WHILE UNIT                                               ********
C       GENERATE 'LABS    GO TO LABS+1'                                 ********
C                'LABS+2  CONTINUE    '                                 ********
970     CALL OUTGON (LABS,LABS+1)                                       PARS0128
        CALL OUTCON(LABS+2)                                             PARS0129
        GO TO 2001                                                      PARS0130
995     CONTINUE                                                        PARS0131
C                                                                       ********
C*      BREAK/NEXT STATEMENT                                            ********
C       CHECK THE LEVEL OF THE LOOP TO BE BROKEN                        ********
C       IF LEVEL IS NOT SPECIFIED '1' IS DEFAULT                        ********
1000    CONTINUE                                                        PARS0132
        CALL GETTOK                                                     PARS0133
        IF(NSTTOK.NE.1) GO TO 1008                                      PARS0134
        TKNSWH=1                                                        PARS0135
1005    LEVEL=1                                                         PARS0136
        GO TO 1030                                                      PARS0137
1008    CONTINUE                                                        PARS0138
        DO 1010 I=1,80                                                  PARS0139
        IF(TOKEN(I).EQ.EOS) GO TO 1015                                  PARS0140
1010    CONTINUE                                                        PARS0141
1015    IF(I.EQ.1) GO TO 1005                                           PARS0142
        CLVEL(1)=ZERO                                                   PARS0143
        IF(I.EQ.2) GO TO 1020                                           PARS0144
        CLVEL(1)=TOKEN(1)                                               PARS0145
        CLVEL(2)=TOKEN(2)                                               PARS0146
        GO TO 1023                                                      PARS0147
1020    CLVEL(2)=TOKEN(1)                                               PARS0148
1023    CALL CHKBLK                                                     PARS0149
```

*** PARSER ***

```
1025    LEVEL=10*((CLVFL(1)-ZERO)/256)                                  PARS0150
     *  +((CLVFL(2)-ZERO)/256)                                          PARS0151
1030    IF(LEVEL.EQ.0) GO TO 2001                                       PARS0152
        IF(LEVEL.LT.0) CALL ERROR (12)                                  PARS0153
C       ERROR-- 'INVALID LEVEL SPECIFIED'                               ********
        J=SP                                                            PARS0154
        LEXST=LEXS                                                      PARS0155
        LABST=LABS                                                      PARS0156
        IFLAG=ZERO                                                      PARS0157
        DO 1050 I=1,LEVEL                                               PARS0158
        IF ((LEXS.LE. LEXDO).AND.(LEXS.GE.LEXFOR))                      PARS0159
     1     GO TO 1035                                                  PARS0160
        I=I-1                                                           PARS0161
        IF(SP.GT.0) GO TO 1040                                          PARS0162
        IF (IFLAG.NE.ZERO) GO TO 1060                                   PARS0163
        CALL ERROR (14)                                                 PARS0164
C       ERROR--'BREAK/NEXT ILLEGAL HERE'                                ********
        SP=J                                                            PARS0165
        GO TO 2001                                                      PARS0166
1035    IFLAG=1                                                         PARS0167
        LEXST=LEXS                                                      PARS0168
        LABST=LABS                                                      PARS0169
1040    SP=SP-1                                                         PARS0170
        CALL GETSTK(LEXS,LABS)                                          PARS0171
1050    CONTINUE                                                        PARS0172
1060    CALL BRKNXT(LEXTKN,LEXST,LABST)                                 PARS0173
        SP=J                                                            PARS0174
        GO TO 2001                                                      PARS0175
C                                                                       ********
C*      UNTIL UNIT                                                      ********
```

*** PARSER ***

```
1100    CONTINUE                                                        PARS0176
        IF ( RRNFLG.EQ.ZERO )  GO TO 1120                               PARS0177
C       ERROR--'CONDITIONAL UNTIL'                                      ********
        CALL ERROR (8)                                                  PARS0178
        GO TO 2001                                                      PARS0179
1120    IF (LEXS.EQ.LEXREP) GO TO 1130                                  PARS0180
C       ERROR--'UNMATCHING UNTIL'                                       ********
        CALL ERROR (9)                                                  PARS0181
        GO TO 40                                                        PARS0182
1130    CONTINUE                                                        PARS0183
        CALL UNGEN (LABS)                                               PARS0184
        SP=SP-1                                                         PARS0185
        GO TO 2001                                                      PARS0186
C                                                                       ********
C*      FORTRAN STATEMENTS                                              ********
C       IF THE STATEMENT IS NOT RECOGNIZED AS AN EXTENDED               ********
C       STATEMENT, THE TRANSLATER ASSUMES IT MUST BE A                  ********
C       FORTRAN STATEMENT AND PASSES IT UNALTERED.                      ********
1200    CALL OTHERS(NLASTC)                                             PARS0187
        GO TO 2001                                                      PARS0188
C       NON-ZERO RRRFLG INDICATES THE STATEMENT IS IF, WHILE,           ********
C       FOR STATEMENTS.IN CASE THERE IS AN ENTRY FOR 'IF' ON            ********
C       THE TOP OF STACK, NEXT STATEMENT IS CHECKED WHETHER             ********
C       IT IS 'ELSE',OTHERWISE CODE IS GENERATED FOR THE END            ********
C       OF IF STATEMENT, FOR ELSE AN ENTRY IS MADE ON STACK             ********
C       AND CODES ARE GENERATED FOR BEGIN OF ELSE-UNIT.                 ********
2000    IF ( RRNFLG.NE.ZERO ) GO TO 5                                   PARS0189
        ERRNOB=0                                                        PARS0190
2001    CALL GETSTK (LEXS,LABS)                                         PARS0191
        RRNFLG=ZERO                                                     PARS0192
```

*** PARSER ***

```
2002    IF (LEXS.NE.LEXIF) GO TO 2020                                  PARS0193
        IF (TKNSWH.EQ.1) GO TO 2006                                    PARS0194
        IF (LASTC.EQ.6) GO TO 2005                                     PARS0195
2004    LASTC=72                                                       PARS0196
2005    NSTTOK=ZERO                                                    PARS0197
        CALL GETTOK                                                    PARS0198
        TKNSWH=1                                                       PARS0199
2006    NLASTC=1                                                       PARS0200
        IF ( NSTTOK.NE.ZERO ) GO TO 2008                               PARS0201
        CALL PUTLIN(BUF)                                               PARS0202
        GO TO 2004                                                     PARS0203
2008    LEXTKN=LEX(TOKEN)                                              PARS0204
        DO 2009 I=1,5                                                  PARS0205
        INLAB(I)=BUF(I)                                                PARS0206
2009    CONTINUE                                                       PARS0207
        IF(LEXTKN.NE.LEXELS) GO TO 2015                                PARS0208
C                                                                      ********
C*      ELSE STATEMENT                                                 ********
2010    CALL PUTSMT (1)                                                PARS0209
        DO 2011 I=1,5                                                  PARS0210
        IF(BUF(I).NE.BLANK) GO TO 2012                                 PARS0211
2011    CONTINUE                                                       PARS0212
        GO TO 2013                                                     PARS0213
2012    CALL ERROR (1)                                                 PARS0214
C       LABEL ILLEGAL HERE-WARNING                                     ********
2013    CONTINUE                                                       PARS0215
C       STACK 'ELSE'                                                   ********
        CALL STACK(LEXFLS,LABS)                                        PARS0216
        K=LABS+2                                                       PARS0217
        L=LABS+1                                                       PARS0218
```

*** PARSER ***

```
C       GENERATE  'LABS   GO TO LABS+2'                                  ********
C                 'LABS+1 CONTINUE    '                                  ********
        CALL OUTGON (LABS,K)                                             PARS0219
        CALL OUTCON (L)                                                  PARS0220
      - RRNFLG=1                                                         PARS0221
        TKNSWH=ZERO                                                      PARS0222
        GO TO 5                                                          PARS0223
C*      CODE FOR END OF IF UNIT                                          ********
C       'LABS+1 CONTINUE'                                                ********
2015    CALL OUTCON (LABS+1)                                             PARS0224
C       POP OFF 'IF' FROM CONTEXT STACK                                  ********
        SP=SP-1                                                          PARS0225
        CALL GETSTK(LEXS,LABS)                                           PARS0226
        IF(LEXS.EQ.LEXIF) GO TO 2015                                     PARS0227
2020    IF(LEXS.NE.LEXELS) GO TO 2100                                    PARS0228
C*      CODE FOR END OF ELSE UNIT                                        ********
C       'LABS+2 CONTINUE'                                                ********
        CALL OUTCON (LABS+2)                                             PARS0229
C       POP OFF 'ELSE' AND 'IF' FROM CONTEXT STACK                       ********
        SP=SP-2                                                          PARS0230
        GO TO 2001                                                       PARS0231
C       THE TOP MOST ENTRY AT STACK IS CHECKED,                          ********
C       IF 'FOR' THEN CODE FOR END OF FOR UNIT                           ********
C       ARE GENERATED AND ENTRY IS REMOVED.                              ********
2100    IF (LEXS.NE.LEXFOR) GO TO 2200                                   PARS0232
        CALL FOREND (LABS)                                               PARS0233
        SP=SP-1                                                          PARS0234
        GO TO 2001                                                       PARS0235
2200    CONTINUE                                                         PARS0236
        IF(TKNSWH.NE.1) GO TO 10                                         PARS0237
```

*** PARSER ***

```
      TKNSWH=ZERO                                                       PARS0238
      GO TO 170                                                         PARS0239
C     WHEN END OF SOURCE IS ENCOUNTERED THE STACK IS CHECKED,           ********
C     WHICH MUST BE EMPTY FOR A COMPLETE PROGRAM.                       ********
3000  CONTINUE                                                          PARS0240
      IF ( SP.NE.0 ) CALL ERROR (10)                                    PARS0241
C     ERROR--'UNEXPECTED END OF SUBPROGRAM'                             ********
      WRITE  (6,3010) NERR                                              PARS0242
3010  FORMAT (1H1,1X/1H ,10 X,'**  STATISTICS AT PREPROCESSOR LEVEL    *PARS0243
     1*'/1H ,1X/1H ,10X,I3,' DIAGNOSTICS GENERATED')                    PARS0244
      STOP                                                              PARS0245
      END                                                               PARS0246
```

*** LEX ***

```
      INTEGER FUNCTION LEX*2(TOKEN)                                          LEX 0000
C*************************************************************************
C                                                                            ********
C     RETURNS LEXICAL TYPE OF TOKEN                                          ********
C                                                                            ********
C*************************************************************************
C                                                                            ********
      INTEGER*2 TOKEN(80),STRING(7),I,J,COMPAR                               LEX 0001
      INTEGER*2 LEXC(11)/-30,-31,-32,-33,-34,-35,-36,-37,-38,-39,-40/        LEX 0002
      INTEGER*2 KEYWRD(7,11) /'I','F',-2,4*' ','D','O',-2,4*' ',            LEX 0003
     *           'W','H','I','L','E',-2,' ','B','E','G','I','N',-2,' ',LEX 0004
     *           'F','O','R',-2,3*' ','R','E','P','E','A','T',-2,          LEX 0005
     *           'E','L','S','E',-2,2*' ', 'E','N','D',-2,3*' ',           LEX 0006
     *           'N','E','X','T',-2,2*' ', 'B','R','E','A','K',-2,' ', LEX 0007
     *           'U','N','T','I','L',-2,' '/                               LEX 0008
C     IF, DO,WHILE,BEGIN,FOR,REPEAT,ELSE,END,NEXT,BREAK,UNTIL                ********
C                                                                            ********
      DO 200 I=1,11                                                          LEX 0009
      IF ((TOKEN(1).EQ.KEYWRD(1,I)).AND.(TOKEN(2).EQ.KEYWRD(2,I)))           LEX 0010
     *  GO TO 300                                                           LEX 0011
200   CONTINUE                                                               LEX 0012
250   LEX=-41                                                                LEX 0013
      RETURN                                                                 LEX 0014
300   DO 400 J=1,7                                                           LEX 0015
      STRING(J)=KEYWRD(J,I)                                                  LEX 0016
400   CONTINUE                                                               LEX 0017
      IF (COMPAR(STRING,TOKEN).NE.1) GO TO 250                               LEX 0018
      LEX=LEXC(I)                                                            LEX 0019
      RETURN                                                                 LEX 0020
      END                                                                    LEX 0021
```

*** COMPAR ***

```
      INTEGER FUNCTION COMPAR*2(STR1,STR2)                                  COMP0000
C*********************************************************************************
C                                                                             ********
C     THIS FUNCTION COMPARES STRING(STR1) AND STRING (STR2),                  ********
C     RETURNS COMPAR=1,IF EQUAL OTHERWISE;0.                                  ********
C                                                                             ********
C*********************************************************************************
C                                                                             ********
      DIMENSION STR1(80),STR2(80)                                             COMP0001
      INTEGER*2 STR1,STR2                                                     COMP0002
      INTEGER*2 EOS/-2/,I                                                     COMP0003
C                                                                             ********
      I=1                                                                     COMP0004
10    IF(STR1(I).NE.STR2(I)) GO TO 100                                        COMP0005
      IF(STR2(I).EQ.EOS) GO TO 50                                             COMP0006
      I=I+1                                                                   COMP0007
      GO TO 10                                                                COMP0008
50    IF(STR1(I).NE.EOS) GO TO 100                                            COMP0009
      COMPAR=1                                                                COMP0010
      RETURN                                                                  COMP0011
100   COMPAR=0                                                                COMP0012
      RETURN                                                                  COMP0013
      END                                                                     COMP0014
```

*** GETTOK ***

```
      SUBROUTINE GETTOK                                                 GETT0000
C*****************************************************************************
C                                                                       ********
C     THIS ROUTINE GETS TOKEN AND PUT IN COMMON BLOCK TKN,              ********
C     FLAG NSTTOK IS SET TO 1,IF TOKEN IS THE FIRST TOKEN OF THE        ********
C     NEW STATEMENT, TO BE USED TO CLASSFY THE STATEMENT.               ********
C     THE END OF TOKEN IS INDICATED BY EOS.                             ********
C                                                                       ********
C*****************************************************************************
C                                                                       ********
      INTEGER   LASTC,LENTH1,TKLNTH                                     GETT0001
      INTEGER*2 C,INBUF(80),LEXSTR(80),NSTTOK,STMCON,TYPE               GETT0002
      INTEGER*2 BLANK/' '/,DLIMIT/-10/,EOF/-1/,EOS/-2/,ZERO/'0'/        GETT0003
C     COMMON BLOCKS                                                     ********
      COMMON /INBLK / INBUF                                             GETT0004
      COMMON /INPTR / LASTC                                             GETT0005
      COMMON /NSTFLG/ NSTTOK                                            GETT0006
      COMMON /PLENTH/ LENTH1                                            GETT0007
      COMMON /TKN   / LEXSTR                                            GETT0008
      COMMON /TKSIZE/ TKLNTH                                            GETT0009
C                                                                       ********
C     TKLNTH IS THE NO. OF CHARS IN THE TOKEN.                          ********
C     IF TOKEN IS CONTINUED ON THE NEXT LINE THEN LENTH1                ********
C     INDICATES THE NO. OF CHARS OF TOKEN ON THE FIRST LINE.            ********
      NSTTOK=0                                                          GETT0010
      TKLNTH=0                                                          GETT0011
      I=1                                                               GETT0012
      LENTH1=0                                                          GETT0013
   10 CALL GETC(C)                                                      GETT0014
      IF (LASTC.NE.7) GO TO 25                                          GETT0015
```

*** GETTOK ***

```
C       STMCON IS STATEMENT CONTINUATION CHARACTER IN THE 6TH COL.      ********
        STMCON=INBUF(6)                                                 GETT0016
        IF ((STMCON.EQ.BLANK).OR.(STMCON.EQ.ZERO)) NSTTOK=1             GETT0017
   20   IF(C.EQ.EOF) GO TO 60                                           GETT0018
   25   IF(C.NE.BLANK) GO TO 30                                         GETT0019
        CALL GETC(C)                                                    GETT0020
        IF (LASTC.NE.7) GO TO 25                                        GETT0021
        STMCON=INBUF(6)                                                 GETT0022
        IF ((STMCON.EQ.BLANK).OR.(STMCON.EQ.ZERO)) GO TO 200            GETT0023
        GO TO 20                                                        GETT0024
C       THE SIGNIFICANT DELIMITER FOR PREPROCESSOR ARE ')','(',',' ********
C       AND BLANK                                                       ********
   30   IF(TYPE(C).EQ.DLIMIT) GO TO 60                                  GETT0025
        LEXSTR(I)=C                                                     GETT0026
        I=I+1                                                           GETT0027
   40   CALL GETC(C)                                                    GETT0028
        IF(LASTC.NE.7) GO TO 30                                         GETT0029
        STMCON=INBUF(6)                                                 GETT0030
   50   IF((STMCON.EQ.BLANK).OR.(STMCON.EQ.ZERO)) GO TO 200             GETT0031
        LENTH1=I-1                                                      GETT0032
        GO TO 30                                                        GETT0033
   60   IF(I.GT.1) GO TO 100                                            GETT0034
C       THE TOKEN IS A NON-BLANK DELIMITER                              ********
   70   LEXSTR(1)=C                                                     GETT0035
        LEXSTR(2)=EOS                                                   GETT0036
        RETURN                                                          GETT0037
  100   LASTC=LASTC-1                                                   GETT0038
        LEXSTR(I)=EOS                                                   GETT0039
        TKLNTH=I-1                                                      GETT0040
        RETURN                                                          GETT0041
```

*** GETTOK ***

```
200     LASTC=6                                                 GETT0042
250     LEXSTR (I)=EOS                                          GETT0043
        TKLNTH=I-1                                              GETT0044
        RETURN                                                  GETT0045
        END                                                     GETT0046
```

*** GETC ***

```
      SUBROUTINE GETC(C)                                                GETC0000
C*************************************************************************
C                                                                       ********
C     THIS ROUTINE READS A SOURCE LINE IN BUF WITH SOURCE POINTER       ********
C     LASTC, IF THE SOURCE LINE IS A COMMENT IT IS PRINTED AS SUCH,     ********
C     PUTS A CHARACTER IN C,WHICH IS NEXT TO ONE POINTED BY LASTC       ********
C     AND LASTC IS INCREMENTED BY ONE. IN CASE END OF SOURCE IS         ********
C     ENCOUNTERED IEOSRC FLAG IS SET TO 1. THE LABEL OF STATEMENT IS    ********
C     PUT IN LABEL                                                      ********
C                                                                       ********
C*****************************************************************************
C                                                                       ********
      INTEGER IEOSRC,LASTC,ISYSIN,ISYOUT                                GETC0001
      INTEGER*2 BUF (80),C,I,LABEL(5)                                   GETC0002
      INTEGER*2  BLANK/' '/,COMT/'C'/,EOF/-1/,MAXCRD/80/                GETC0003
C     COMMON BLOCKS                                                     ********
      COMMON /INBLK/BUF                                                 GETC0004
      COMMON /INPTR/LASTC                                               GETC0005
      COMMON /INLBL/LABEL                                               GETC0006
      COMMON /EOSRC/IEOSRC                                              GETC0007
      COMMON /SYSIN/ISYSIN                                              GETC0008
      COMMON /SYSOUT/ISYOUT                                             GETC0009
C                                                                       ********
      IF (IEOSRC.EQ.1) GO TO 250                                        GETC0010
C     INCREMENT THE SOURCE POINTER                                      ********
      LASTC=LASTC+1                                                     GETC0011
      IF(LASTC.LE.6) LASTC=7                                            GETC0012
C     IN CASE THE LINE IS SCANNED FULLY GO TO GET NEXT LINE.            ********
      IF(LASTC.GT.72) GO TO 5                                           GETC0013
      GO TO 100                                                         GETC0014
```

*** GETC ***

```
5       READ (ISYSIN,10,END=200) (BUF(I),I=1,MAXCRD)                    GETC0015
10      FORMAT(80A1)                                                    GETC0016
C       COMMENT STATEMENT IS PRINTED AS IT IS.                          ********
        IF (BUF(1).NE.COMT) GO TO 50                                    GETC0017
        WRITE  (ISYOUT,10) (BUF(I),I=1,MAXCRD)                          GETC0018
C       GO TO READ THE NEXT LINE                                        ********
        GOTO 5                                                          GETC0019
50      LASTC=7                                                         GETC0020
C       STORE THE STATEMENT LABEL                                       ********
        DO 60 I=1,5                                                     GETC0021
        LABEL(I)=BUF(I)                                                 GETC0022
60      CONTINUE                                                        GETC0023
100     C=BUF(LASTC)                                                    GETC0024
        RETURN                                                          GETC0025
200     IEOSRC=1                                                        GETC0026
C       SET END OF SOURCE FLAG                                          ********
250     C=EOF                                                           GETC0027
        LASTC=7                                                         GETC0028
        BUF(6)=BLANK                                                    GETC0029
        BUF(7)=BLANK                                                    GETC0030
        RETURN                                                          GETC0031
        END                                                             GETC0032
```

*** TYPE ***

```
      INTEGER FUNCTION TYPE*2 (CHAR)                                    TYPE0000
C*****************************************************************************
C                                                                       ********
C     THE FUNCTION TYPE DETERMINES WHETHER A PARTICULAR CHARACTER       ********
C     CHAR IS A SIGNIFICANT DELIMITER FOR PREPROCESSOR,OR SOMETHING     ********
C     ELSE; IT RETURNS DLIMIT OR THE CHARACTER ITSELF.                  ********
C                                                                       ********
C*****************************************************************************
C                                                                       ********
      INTEGER*2 BLANK/' '/,CHAR,DLIMIT/-10/,COMMA/','/,                 TYPE0001
     1          LPAREN/'('/,RPAREN/')'/                                 TYPE0002
C                                                                       ********
      IF ((CHAR.EQ.BLANK).OR.                                           TYPE0003
     1    (CHAR.EQ.LPAREN).OR.                                          TYPE0004
     2    ( CHAR.EQ.RPAREN ).OR.                                        TYPE0005
     3    (CHAR.EQ.COMMA)) GO TO 100                                    TYPE0006
50    TYPE=CHAR                                                         TYPE0007
      RETURN                                                            TYPE0008
100   TYPE=DLIMIT                                                       TYPE0009
      RETURN                                                            TYPE0010
      END                                                               TYPE0011
```

*** CHKBLK ***

```
      SUBROUTINE CHKBLK                                                 CHKB0000
C*****************************************************************************
C                                                                       ********
C     THIS ROUTINE CHECKS FOR A NON-BLANK CHARACTER IN THE REMAINING    ********
C     PART OF THE BUFFER AND ISSUES AN ERROR IF ENCOUNTERED.            ********
C                                                                       ********
C*****************************************************************************
C                                                                       ********
      INTEGER*2 BUF(80),BLANK/' '/                                      CHKB0001
      INTEGER LASTC                                                     CHKB0002
      COMMON/INBLK/BUF                                                  CHKB0003
      COMMON/INPTR/LASTC                                                CHKB0004
C                                                                       ********
5     J=LASTC+1                                                         CHKB0005
      IF(J.GE.73) RETURN                                                CHKB0006
      DO 10 I=J,72                                                      CHKB0007
      IF(BUF(I).NE.BLANK) GO TO 100                                     CHKB0008
10    CONTINUE                                                          CHKB0009
      RETURN                                                            CHKB0010
C     ERROR--'SYNTAX ERROR'                                             ********
100   CALL ERROR (15)                                                   CHKB0011
      RETURN                                                            CHKB0012
      END                                                               CHKB0013
```

*** STACK ***

```
      SUBROUTINE STACK (SLEX,SLAB)                                      STAC0000
C*****************************************************************************
C                                                                       ********
C     THIS ROUTINE POPS IN THE CONTEXT STACK WITH SLEX-SLAB             ********
C     MAKING ONE ENTRY.                                                 ********
C                                                                       ********
C*****************************************************************************
C                                                                       ********
      DIMENSION STKLEX(100),STKLAB(100)                                 STAC0001
      INTEGER STKLAB,SLAB,SP                                            STAC0002
      INTEGER*2 STKLEX,SLEX                                             STAC0003
      COMMON/STKPTR/SP                                                  STAC0004
      COMMON/STKBLK/STKLEX,STKLAB                                       STAC0005
C                                                                       ********
C     INCREMENT THE STACK POINTER.                                      ********
      SP=SP+1                                                           STAC0006
      IF(SP.GT.100) GO TO 10                                            STAC0007
      STKLEX(SP)=SLEX                                                   STAC0008
      STKLAB(SP)=SLAB                                                   STAC0009
      RETURN                                                            STAC0010
10    CONTINUE                                                          STAC0011
      RETURN                                                            STAC0012
      END                                                               STAC0013
```

*** GETSTK ***

```
      SUBROUTINE GETSTK (LEXS,LABS)                                    GETS0000
C*****************************************************************************
C                                                                      ********
C       THIS ROUTINE POPS THE STACK AND PUTS THE FIRST ENTRY IN        ********
C     LEXS & LABS WITHOUT DECREMENTING STACK POINTER.                  ********
C                                                                      ********
C*****************************************************************************
C                                                                      ********
      INTEGER   LABS,SP,STKLAB(100)                                    GETS0001
      INTEGER*2 BLANK/' '/,LEXS,STKLEX(100)                            GETS0002
      COMMON /STKPTR/ SP                                               GETS0003
      COMMON /STKBLK/ STKLEX,STKLAB                                    GETS0004
C                                                                      ********
      IF (SP.EQ.0) GO TO 100                                           GETS0005
      LEXS=STKLEX(SP)                                                  GETS0006
      LABS=STKLAB(SP)                                                  GETS0007
      RETURN                                                           GETS0008
  100 LEXS=BLANK                                                       GETS0009
      LABS=0                                                           GETS0010
      RETURN                                                           GETS0011
      END                                                              GETS0012
```

*** LABGEN ***

```
      INTEGER FUNCTION LABGEN(N)                                        LABG0000
C*************************************************************************
C                                                                       ********
C       THIS FUNCTION PLACES CURRENT VALVE OF GENERATED LABEL           ********
C       IN LABGEN AND RESERVES NEXT N LABELS.                           ********
C                                                                       ********
C*************************************************************************
C                                                                       ********
      INTEGER LABEL,N                                                   LABG0001
      COMMON/RTFLAB/LABEL                                               LABG0002
C                                                                       ********
      LABGEN=LABEL                                                      LABG0003
      LABEL=LABEL+N                                                     LABG0004
      RETURN                                                            LABG0005
      END                                                               LABG0006
```

*** ERROR ***

```
      SUBROUTINE ERROR (ERCODE)                                          ERR00000
C*************************************************************************
C                                                                        ********
C     PRINTS ERROR MESSAGE                                               ********
C                                                                        ********
C*************************************************************************
C                                                                        ********
      INTEGER ERCODE,LASTC,NERR,ERRNOB,ERCOD2                            ERR00001
      INTEGER*2 ISNTX(66)/66*' '/,DOLLER/'$'/,BLANK/' '/                 ERR00002
      INTEGER*2 ERCD,ZERO/'0'/,ONE/'1'/                                  ERR00003
      INTEGER ERRORM(8,14)/'WARN','ING-','-LAB','EL I','LLEG','AL H',    ERR00004
     *  'ERE ','    ',                                                   ERR00005
     *  ''''('''-','-MIS','STNG','    ','    ','    ','    ','    ',    ERR00006
     *  'UNMA','TCHI','NG ''','('''S ','    ','    ','    ','    ',    ERR00007
     *  'UNRE','COGN','IZAB','LE S','TATE','MENT','    ','    ',       ERR00008
     *  'TOO ','MANY',' NES','TED ','''FOR','''' ST','ATEM','ENTS',     ERR00009
     *  'SYNT','AX E','RROR',' IN ','''FOR','''' ST','ATEM','ENT ',     ERR00010
     *  'TOO ','LARG','E A ','COND','ITIO','N/ST','ATEM','ENT ',        ERR00011
     *  'COND','ITIO','NAL ','END/','ELSE','/UNT','IL  ','    ',       ERR00012
     *  'UNMA','TCHI','NG E','LSE/','UNTI','L   ','    ','    ',       ERR00013
     *  'UNEX','PECT','ED E','ND O','F SU','BPRO','GRAM','    ',       ERR00014
     *  'NO L','OOP ','EXIS','TS  ','    ','    ','    ','    ',       ERR00015
     *  'ILLE','GAL ','LEVE','L SP','ECIF','IED ','    ','    ',       ERR00016
     *  'FORT','RAN ','DO S','TMT ','ILLE','GAL ','HERE','    ',       ERR00017
     *  'BREA','K/NE','XT I','LLEG','AL H','ERE ','    ','    '/       ERR00018
C     COMMON BLOCKS                                                      ********
      COMMON /ERRNO / ERRNOB                                             ERR00019
      COMMON /INPTR / LASTC                                              ERR00020
      COMMON /NOERR / NERR                                               ERR00021
      COMMON /SYSOUT/ ISYOUT                                             ERR00022
```

*** ERROR ***

```
C                                                                       ********
C       NERR IS TOTAL NUMBER OF ERROR S IN A PROGRAM                    ********
        NERR=NERR+1                                                     ERR00023
C       ERRNOB IS SERIAL NO. OF ERROR IN A STATEMENT                    ********
        ERRNOB=ERRNOB+1                                                 ERR00024
        IF(ERCODE.GE.10) GO TO 20                                       ERR00025
        ERCOD2=ERCODE                                                   ERR00026
        ERCD=ZERO                                                       ERR00027
        GO TO 30                                                        ERR00028
20      ERCD=ONE                                                        ERR00029
        ERCOD2=ERCODE-10                                                ERR00030
        IF (ERCODE.GT.13) GO TO 100                                     ERR00031
30      WRITE  (ISYOUT,50) ERRNOB,ERCD,ERCOD2,(ERRORM(I,ERCODE),I=1,8)  ERR00032
50      FORMAT ('C****0',I1,')  JFP0',A1,I1,'I  ',8A4,29X)              ERR00033
C       THE FORMAT OF ERRORE MESSAGE IS                                 ********
C       C****01)  JFP002I TEXT OF MESSAGE                               ********
        RETURN                                                          ERR00034
100     I=LASTC-6                                                       ERR00035
        ISNTX(I)=DOLLER                                                 ERR00036
        WRITE  (ISYOUT,150) ISNTX,ERRNOB,ERCD,ERCOD2                    ERR00037
150     FORMAT ('C     ',66A1,8X/                                       ERR00038
     1  'C****0',I1,')  JFP0',A1,I1,'I  SYNTAX',55X)                    ERR00039
        ISNTX(I)=BLANK                                                  ERR00040
        RETURN                                                          ERR00041
        END                                                             ERR00042
```

*** BGNGEN ***

```
      SUBROUTINE BGNGEN                                                 BGNG0000
C*****************************************************************************
C                                                                       ********
C     THIS ROUTINE GENERATE CODE FOR BEGIN STATEMENT,                   ********
C     NO CODE IS GENERATED IF STATEMENT IS UNLABELED,                   ********
C     AN ENTRY IS MADE ON CONTEXT STACK                                 ********
C                                                                       ********
C*****************************************************************************
C                                                                       ********
      INTEGER LAB,LABGEN                                                BGNG0001
      INTEGER*2 LABEL(5),LEXBGN/-33/,BLANK/' '/,I                       BGNG0002
      COMMON/INLBL/LABEL                                                BGNG0003
      COMMON /SYSOUT/ISYOUT                                             BGNG0004
C                                                                       ********
      LAB=LABGEN (1)                                                    BGNG0005
C     1 GENERATED LABEL IS RESERVED                                     ********
      DO 10 I=1,5                                                       BGNG0006
      IF (LABEL(I).NE.BLANK) GO TO 20                                   BGNG0007
10    CONTINUE                                                          BGNG0008
      GO TO 50                                                          BGNG0009
C     FOR LABELED 'BEGIN' CODE 'LABEL CONTINUE' IS GENERATED            ********
20    WRITE (ISYOUT,30) LABEL                                           BGNG0010
30    FORMAT(5A1,' CONTINUE',58X,'++',6X)                               BGNG0011
C     POP IN LEXBGN AND GENERATED LABEL                                 ********
50    CALL STACK(LEXBGN,LAB-2)                                          BGNG0012
      RETURN                                                            BGNG0013
      END                                                               BGNG0014
```

*** BALPAR ***

```
      SUBROUTINE BALPAR                                                 BALP0000
C*****************************************************************************
C                                                                       ********
C     BALPAR COLLECTS AND PUT THE CONDITION PART IN CONBUF,             ********
C     WHICH IS A STRING ENCLOSED IN BALANCED PARENTHESES.               ********
C                                                                       ********
C*****************************************************************************
C                                                                       ********
      INTEGER*2 ZERO/1H0/,BLANK/1H /,EOF/-1/,LBRK/1H(/,RBRK/1H)/        BALP0001
      INTEGER*2   BUF(80),C,CONT,NL                                     BALP0002
      INTEGER LASTC                                                     BALP0003
      COMMON/INBLK/  BUF                                                BALP0004
      COMMON/INPTR/LASTC                                                BALP0005
C                                                                       ********
C     NL IS NO. OF '(' ENCOUNTERED                                      ********
      NL=1                                                              BALP0006
      GO TO 17                                                          BALP0007
10    NL=NL+1                                                           BALP0008
15    CALL PUTCND(C)                                                    BALP0009
17    CALL GETC(C)                                                      BALP0010
      IF(LASTC.NE.7) GO TO 20                                           BALP0011
      CONT=BUF(6)                                                       BALP0012
C     CONTINUATION OF CONDITION ON NEXT LINE                            ********
      IF ((CONT.EQ.ZERO).OR.(CONT.EQ.BLANK)) GO TO 50                   BALP0013
20    IF(C.EQ.LBRK) GO TO 10                                            BALP0014
      IF(C.NE.RBRK) GO TO 15                                            BALP0015
      NL=NL-1                                                           BALP0016
      IF(NL.EQ.0) GOTO 100                                              BALP0017
      GO TO 15                                                          BALP0018
50    LASTC=6                                                           BALP0019
```

*** BALPAR ***

```
      IF(NL.EQ.0) RETURN                                        BALP0020
C     ERROR--'UNMATCHING ')''                                   ********
      CALL ERROR (3)                                            BALP0021
      RETURN                                                    BALP0022
100   CALL PUTCND(C)                                            BALP0023
      RETURN                                                    BALP0024
      END                                                       BALP0025
```

*** PUTCND ***

```
      SUBROUTINE PUTCND(CHAR)                                          PUTC0000
C*****************************************************************************
C                                                                      ********
C     PUTS A CHARACTER (CHAR) IN CONDITION BUFFER AND                  ********
C     THE CONDITION BUFFER POINTER (PTR) IS INCREMENTED BY 1.          ********
C                                                                      ********
C*****************************************************************************
C                                                                      ********
      INTEGER*2 CONBUF(1254),CHAR                                      PUTC0001
      INTEGER   PTR1                                                   PUTC0002
      COMMON/CONBLK/CONBUF                                             PUTC0003
      COMMON/CONPTR/PTR1                                               PUTC0004
C                                                                      ********
      PTR1=PTR1+1                                                      PUTC0005
      IF (PTR1.LE.1254) GO TO 10                                       PUTC0006
C     ERROR-- 'TOO LARGE A CONDITION/STATEMENT'                        ********
      CALL ERROR (7)                                                   PUTC0007
      RETURN                                                           PUTC0008
10    CONBUF(PTR1)=CHAR                                                PUTC0009
      RETURN                                                           PUTC0010
      END                                                              PUTC0011
```

*** IFCODE ***

```
      SUBROUTINE IFCODE                                                 IFC00000
C*****************************************************************************
C                                                                       ********
C     PROCESS IF STATEMENT, FORTRAN IF STATEMENTS ARE BYPASSED          ********
C                                                                       ********
C*****************************************************************************
C                                                                       ********
      INTEGER LAB,LABGEN,KEY                                            IFC00001
      INTEGER*2 LEXIF/-30/,RRNFLG                                       IFC00002
      COMMON/RNFLG/RRNFLG                                               IFC00003
C                                                                       ********
      RRNFLG=0                                                          IFC00004
C     3 GENERATED LABELS ARE RESERVED FOR IF STATEMENT                  ********
      LAB=LABGEN(3)                                                     IFC00005
      CALL IFGO(1,LAB+1,KEY)                                            IFC00006
C     KEY VALUES 1 WHEN STATEMENT IS FORTRAN IF STATEMENT               ********
      IF(KEY.EQ.1) RETURN                                               IFC00007
C     AN ENTRY FOR 'IF' ON CONTEXT STACK.                               ********
      CALL STACK (LEXIF,LAB)                                            IFC00008
      RRNFLG=1                                                          IFC00009
      RETURN                                                            IFC00010
      END                                                               IFC00011
```

*** IFGO ***

```
      SUBROUTINE IFGO(CNTRL,LAB,TYPE)                                    IFG00000
C*************************************************************************
C                                                                         ********
C       THIS ROUTINE IS CALLED TO PROCESS THE CODITION PART OF            ********
C       IF, WHILE, UNTIL, AND FOR STATEMENT. IF CALLED WITH               ********
C       CNTRL=1 IT CHECKS FOR FORTRAN IF STATEMENTS,IF THE STMT           ********
C       IS NOT FORTRAN STMT KEY IS SET TO 0 OTHERWISE 1.                  ********
C                                                                         ********
C*************************************************************************
C                                                                         ********
      INTEGER   CNTRL,LAB,LASTC,TYPE                                      IFG00001
      INTEGER*2 THEN(5)/'T','H','E','N',-2/,LBRK/'('/,EOS/-2/             IFG00002
      INTEGER*2 C,COMPAR,NSTTOK,TFLG,TOKEN(80)                            IFG00003
C     COMMON BLOCKS                                                       ********
      COMMON /INPTR /LASTC                                                IFG00004
      COMMON /TKNFLG/TFLG                                                 IFG00005
      COMMON /TKN   /TOKEN                                                IFG00006
      COMMON /NSTFLG/NSTTOK                                               IFG00007
C                                                                         ********
      TFLG=0                                                              IFG00008
C     CHECK FOR THE CONDITION PART                                        ********
      CALL GETTOK                                                         IFG00009
      IF (NSTTOK.EQ.1) GO TO 300                                          IFG00010
      IF (TOKEN(1).EQ.LBRK) GO TO 20                                      IFG00011
C     ERROR--'(' MISSING                                                  ********
      CALL ERROR (2)                                                      IFG00012
C     THE CONDITION PART IS STORED IN CODITION BUFFER                     ********
20    DO 30 I=1,80                                                        IFG00013
      C=TOKEN(I)                                                          IFG00014
      IF (C.EQ.EOS) GO TO 35                                              IFG00015
```

*** IFGO ***

```
      CALL PUTCND (C)                                                  IFG00016
30    CONTINUE                                                         IFG00017
35    CONTINUE                                                         IFG00018
C     BALPAR TO BALANCE THE PARENTHISIS                                ********
      CALL BALPAR                                                      IFG00019
C     WHEN CNTRL IS 1 STATEMENT IS IF-STATEMENT                        ********
      IF (CNTRL.EQ.0) GO TO 100                                        IFG00020
      IF (LASTC.EQ.6) GO TO 50                                         IFG00021
      CALL GETTOK                                                      IFG00022
      IF (NSTTOK.NE.1) GO TO 40                                        IFG00023
      LASTC=6                                                          IFG00024
      GO TO 50                                                         IFG00025
40    IF (COMPAR(TOKEN,THEN).EQ.1) GO TO 100                           IFG00026
      IFLG=1                                                           IFG00027
50    CALL PUTIF(0,LAB)                                                IFG00028
C     THE FORTRAN IF STATEMENTS ARE PRINTED AS SUCH AND                ********
C     TYPE IS SET TO 1                                                 ********
      TYPE=1                                                           IFG00029
      RETURN                                                           IFG00030
100   CALL PUTIF(1,LAB)                                                IFG00031
C     THE CODE IF(.NOT.(CONDITION))------- IS GENERATED                ********
      TYPE=0                                                           IFG00032
      RETURN                                                           IFG00033
300   LASTC=6                                                          IFG00034
      TYPE=0                                                           IFG00035
      RETURN                                                           IFG00036
      END                                                              IFG00037
```

*** PUTIF ***

```
      SUBROUTINE PUTIF (TYPE,LAB)                                        PUTI0000
C*************************************************************************
C                                                                        ********
C     THIS ROUTINE GENERATES CODE 'IF(.NOT.(CONDITION)) ...'             ********
C     OR 'IF (CONDITION) ....' WHEN TYPE=1 OR 0; THE CONDITION           ********
C     IS TAKEN FROM THE CONDITION BUFFER.                                ********
C                                                                        ********
C*************************************************************************
C                                                                        ********
      DIMENSION IFNOT(8)                                                 PUTI0001
      INTEGER*2 CONBUF(1254),TOKEN(80),INBUF(80),INLAB(5),C,CNT,TFLG     PUTI0002
      INTEGER*2 IF(2)/'I','F'/,IFNOT/'T','F','(','.','N','O','T','.'/,  PUTI0003
     *          BLANK/' '/,ZERO/'0'/,EOF/-1/,EOS/-2/,                    PUTI0004
     *          GOTO(8)/')',' ','G','O',' ','T','O',' '/                 PUTI0005
      INTEGER I,I1,I2,J,LASTC,PTR,TYPE                                   PUTI0006
C     COMMON BLOCKS                                                      ********
      COMMON /CONBLK/ CONBUF                                             PUTI0007
      COMMON /CONPTR/ PTR                                                PUTI0008
      COMMON /INBLK / INBUF                                              PUTI0009
      COMMON /INPTR / LASTC                                              PUTI0010
      COMMON /INLBL / INLAB                                              PUTI0011
      COMMON /SYSOUT/ ISYOUT                                             PUTI0012
      COMMON /TKNFLG/ TFLG                                               PUTI0013
      COMMON /TKN   / TOKEN                                              PUTI0014
C                                                                        ********
      DO 10 I=6,14                                                       PUTI0015
      CONBUF(I)=BLANK                                                    PUTI0016
10    CONTINUE                                                           PUTI0017
C     MOVE STATEMENT LABEL TO CONBUF                                     ********
      DO 20 I=1,5                                                        PUTI0018
```

*** PUTIF ***

```
      CONBUF(I)=INLAB(I)                                                PUTI0019
20    CONTINUE                                                          PUTI0020
C     CLEAR INLAB                                                       ********
      CALL CLRFLD (INLAB,1,5)                                           PUTI0021
25    CONTINUE                                                          PUTI0022
30    IF(TYPE.EQ.1) GO TO 1000                                          PUTI0023
C     WHEN TYPE=0 'IF' IS MOVED TO 14TH COLUMN OF CONBUF                ********
      CONBUF(13)=IF(1)                                                  PUTI0024
      CONBUF(14)=IF(2)                                                  PUTI0025
      I1=13                                                             PUTI0026
      IF(TFLG.EQ.0) GO TO 100                                           PUTI0027
C     UNPROCESSED TOKEN IS PUT IN CONBUF                                ********
      DO 50 I=1,80                                                      PUTI0028
      C=TOKEN(I)                                                        PUTI0029
      IF (C.EQ.EOS) GO TO 60                                            PUTI0030
      CALL PUTCND(C)                                                    PUTI0031
50    CONTINUE                                                          PUTI0032
60    TFLG=0                                                            PUTI0033
C     THE UNCOVERED PART OF THE STATEMENT IS PUT IN CONBUF              ********
100   CALL GETC(C)                                                      PUTI0034
      IF(LASTC.EQ.7) GO TO 200                                          PUTI0035
150   CALL PUTCND(C)                                                    PUTI0036
      GO TO 100                                                         PUTI0037
200   CNT=INBUF(6)                                                      PUTI0038
      IF(.NOT.((CNT.EQ.BLANK).OR.(CNT.EQ.ZERO))) GO TO 150              PUTI0039
      LASTC=6                                                           PUTI0040
      I2=78                                                             PUTI0041
      GO TO 2000                                                        PUTI0042
C     THE FOLLOWING PART GENERATES THE CODE                             ********
C     'IF (.NOT.(CONDITION)) GO TO LAB'                                 ********
```

*** PUTTF ***

```
1000    DO 1100 I=1,8                                                   PUTI0043
        J=I+6                                                           PUTI0044
C       MOVE 'IF (.NOT.' IN CONBUF LEFT TO CONDITION                    ********
        CONBUF(J)=IFNOT(I)                                              PUTI0045
1100    CONTINUE                                                        PUTI0046
C       MOVE 'GO TO' IN CONBUF RIGHT TO THE CONDITION                   ********
        DO 1200 I=1,8                                                   PUTI0047
        C=GOTO(I)                                                       PUTI0048
        CALL PUTCND(C)                                                  PUTI0049
1200    CONTINUE                                                        PUTI0050
        I=LAB                                                           PUTI0051
C       CONVERT LAB INTO CHARACTER FORM AND MOVE TO CONBUF              ********
        CALL DGTCHR(I,INLAB)                                            PUTI0052
        DO 1300 I=1,5                                                   PUTI0053
        C=INLAB(I)                                                      PUTI0054
        CALL PUTCND(C)                                                  PUTI0055
1300    CONTINUE                                                        PUTI0056
        CALL CLRFLD (INLAB,1,5)                                         PUTI0057
        I1=7                                                            PUTI0058
        I2=72                                                           PUTI0059
C       FOLLOWING PART PRINTS THE CONTENTS OF CONBUF                    ********
2000    IF (PTR.EQ.I2) GO TO 2400                                       PUTI0060
        J=PTR+1                                                         PUTI0061
        IF (PTR.GT.I2) GO TO 2100                                       PUTI0062
        K=I2                                                            PUTI0063
        GO TO 2200                                                      PUTI0064
2100    IQ=(PTR-I2)/66                                                  PUTI0065
        I=(IQ*66)+I2                                                    PUTI0066
        IF(I.NE.PTR) GO TO 2150                                         PUTI0067
        K=PTR                                                           PUTI0068
```

*** PUTTF ***

```
      GO TO 2400                                                          PUTI0069
2150  K=((IQ+1)*66)+I2                                                    PUTI0070
2200  DO 2300 I=J,K                                                       PUTI0071
      CONBUF(I)=BLANK                                                     PUTI0072
2300  CONTINUE                                                            PUTI0073
2400  WRITE (ISYOUT,2500) (CONBUF(I),I=1,5),(CONBUF(I),I=I1,I2)           PUTI0074
2500  FORMAT (5A1,' ',66A1,'++',6X)                                       PUTI0075
      IF(PTR.LE.I2) GO TO 3000                                            PUTI0076
      I2=I2+1                                                             PUTI0077
      WRITE  (ISYOUT,2600) (CONBUF(I),I=I2,K)                             PUTI0078
2600  FORMAT ('     *',66A1,'++',6X)                                      PUTI0079
3000  PTR=14                                                              PUTI0080
      RETURN                                                              PUTI0081
      END                                                                 PUTI0082
```

*** DOCODE ***

```
      SUBROUTINE DOCODE (ID,NLASTC)                                      DOC00000
C*************************************************************************
C                                                                        ********
C       THIS ROUTINE PROCESS 'DO' STATEMENT, IN CASE THE STATEMENT       ********
C       IS FORTRAN 'DO' VALUE OF (ID) IS RETURNED AS 1, OTHERWISE 0      ********
C       A GENERATED LABEL IS INSERTED BETWEEN 'DO' AND INDEX, WITH       ********
C       AN ENTRY ON CONTEXT STACK                                        ********
C                                                                        ********
C*************************************************************************
C                                                                        ********
      INTEGER LASTC,NLASTC,LAB,LABGEN                                    DOC00001
      INTEGER*2 DIGITS(10)/'1','2','3','4','5','6','7','8','9','0'/,    DOC00002
     * DO(2)/'D','O'/,BUF(80),TOKEN(80),INLAB(5),OBUF(80)/80*' '/,ID     DOC00003
      INTEGER*2 BLANK/' '/,LEXDO/-31/,EOS/-2/,STAR/'*'/                  DOC00004
C     COMMON BLOCKS                                                      ********
      COMMON/TKN/TOKEN                                                   DOC00005
      COMMON /INBLK / BUF                                                DOC00006
      COMMON /INPTR / LASTC                                              DOC00007
      COMMON /INLBL / INLAB                                              DOC00008
      COMMON /SYSOUT/ ISYOUT                                             DOC00009
C                                                                        ********
      IF(NLASTC.EQ.1) NLASTC=7                                           DOC00010
      N=LASTC                                                            DOC00011
      CALL GETTOK                                                        DOC00012
C     WHEN THE TOKEN AFTER 'DO' IS AN INTEGER (DIGITS), THE              ********
C     STATEMENT IS RECOGNIZED AS FORTRAN 'DO'                            ********
      DO 10 I=1,10                                                       DOC00013
      IF (TOKEN(1).EQ.DIGITS(I)) GO TO 500                               DOC00014
10    CONTINUE                                                           DOC00015
      LAB=LABGEN(2)                                                      DOC00016
```

*** DOCODE ***

```
      CALL STACK (LEXDO,LAB-1)                                    DOC00017
      DO 30 I=1,5                                                 DOC00018
      OBUF(I)=INLAB(I)                                            DOC00019
30    CONTINUE                                                    DOC00020
      CALL CLRFLD (OBUF,6,80)                                     DOC00021
      OBUF(7)=DO(1)                                               DOC00022
      OBUF(8)=DO(2)                                               DOC00023
C     CONVERT LAB INTO CHARACTER FORM                             ********
      CALL DGTCHR (LAB,INLAB)                                     DOC00024
      J=0                                                         DOC00025
      DO 40 I=10,14                                               DOC00026
      J=J+1                                                       DOC00027
      OBUF(I)=INLAB(J)                                            DOC00028
40    CONTINUE                                                    DOC00029
      I=15                                                        DOC00030
      DO 50 J=1,80                                                DOC00031
      I=I+1                                                       DOC00032
      IF (TOKEN(J).EQ.FOS) GO TO 60                               DOC00033
      OBUF(I)=TOKEN(J)                                            DOC00034
50    CONTINUE                                                    DOC00035
60    I=I-1                                                       DOC00036
65    IF  (LASTC.EQ.72) GO TO 200                                 DOC00037
      LASTC=LASTC+1                                               DOC00038
      IF(I.LT.72) GO TO 80                                        DOC00039
      WRITE (ISYOUT,70) (OBUF(IJ),IJ=1,72)                        DOC00040
70    FORMAT (72A1,'++',6X)                                       DOC00041
      OBUF(6)=STAR                                                DOC00042
      I=6                                                         DOC00043
80    IF  (BUF(LASTC).EQ.BLANK) GO TO 100                         DOC00044
      I=I+1                                                       DOC00045
```

*** DOCODE ***

```
      OBUF(I)=BUF(LASTC)                                              DOC00046
100   GO TO 65                                                        DOC00047
200   WRITE  (ISYOUT,70) (OBUF(IJ),IJ=1,72)                           DOC00048
      TD=0                                                            DOC00049
      RETURN                                                          DOC00050
500   IF(LASTC.LT.N) LASTC=6                                          DOC00051
      WRITE  (ISYOUT,600) INLAB,(BUF(I),I=NLASTC,72)                  DOC00052
600   FORMAT (5A1,' ',66A1,8X)                                        DOC00053
      TD=1                                                            DOC00054
      RETURN                                                          DOC00055
      END                                                             DOC00056
```

*** DGTCHR ***

```
      SUBROUTINE DGTCHR (DIGIT,CHAR)                                    DGTC0000
C*********************************************************************
C                                                                       ********
C     A 5 FIGURE NO. DIGIT IS CONVERTED INTO CHARACTER STRING           ********
C                                                                       ********
C*********************************************************************
C                                                                       ********
      INTEGER DIGIT,I,NI,IQ                                             DGTC0001
      INTEGER*2  CHAR(5)                                                DGTC0002
C                                                                       ********
      DO 10 I=1,5                                                       DGTC0003
      IQ=DIGIT/10000                                                    DGTC0004
      CHAR(I)=((IQ*256)+(15*(16**3))+64)                                DGTC0005
      NI=IQ*10000                                                       DGTC0006
      DIGIT=(DIGIT-NI)*10                                               DGTC0007
10    CONTINUE                                                          DGTC0008
      RETURN                                                            DGTC0009
      END                                                               DGTC0010
```

*** WHILE ***

```
      SUBROUTINE WHILE                                                  WHIL0000
C*****************************************************************************
C                                                                       ********
C     ROUTINE TO PROCESS 'WHILE' STATEMENT                              ********
C                                                                       ********
C*****************************************************************************
C                                                                       ********
      INTEGER LAB,LABGEN,I                                              WHIL0001
      INTEGER*2 LEXWHL/-32/,BLANK/' '/,INLAB(5)                         WHIL0002
      COMMON/INLBL/INLAB                                                WHIL0003
C                                                                       ********
C     3 GENERATED LABEL RESERVED                                        ********
      LAB=LABGEN(3)                                                     WHIL0004
      DO 10 I=1,5                                                       WHIL0005
      IF(INLAB(I).NE.BLANK) GO TO 100                                   WHIL0006
10    CONTINUE                                                          WHIL0007
      GO TO 200                                                         WHIL0008
C     FOR LABELED STATEMENT THE CODE 'LABEL CONTINUE' IS GENERATED      ********
100   CALL LABCON(INLAB)                                                WHIL0009
200   I=LAB+1                                                           WHIL0010
      CALL DGTCHR(I,INLAB)                                              WHIL0011
C     CODE 'IF (.NOT.(CONDITION)) GO TO LAB+2' IS GENERATED AND         ********
C     ENTRY IS MADE ON CONTEXT STACK.                                   ********
      CALL IFGO(0,LAB+2,I)                                              WHIL0012
      CALL STACK (LEXWHL,LAB)                                           WHIL0013
      RETURN                                                            WHIL0014
      END                                                               WHIL0015
```

*** REPGEN ***

```
      SUBROUTINE REPGEN                                                 REPG0000
C*****************************************************************************
C                                                                       ********
C     PROCESSES THE 'REPEAT' STATEMENT AND MAKES AN ENTRY IN            ********
C     CONTEXT STACK.                                                    ********
C                                                                       ********
C*****************************************************************************
C                                                                       ********
      INTEGER*2 INLAB(5),BLANK/' '/,LEXREP/-35/                         REPG0001
      INTEGER LAB,LABGEN                                                REPG0002
      COMMON/INLBL/INLAB                                                REPG0003
C                                                                       ********
      LAB=LABGEN(3)                                                     REPG0004
C     3 GENERATED LABELS ARE RESERVED                                   ********
C     STACK  'REPEAT'                                                   ********
      CALL STACK (LEXREP,LAB)                                           REPG0005
      DO 10 I=1,5                                                       REPG0006
      IF(INLAB(I).NE.BLANK) GO TO 50                                    REPG0007
10    CONTINUE                                                          REPG0008
      GO TO 100                                                         REPG0009
C     CODES GENERATED  ' INLAB  CONTINUE' FOR LABELED STATEMENT         ********
C                      ' LAB    CONTINUE'                             ********
50    CALL LABCON(INLAB)                                                REPG0010
100   CALL OUTCON(LAB)                                                  REPG0011
      RETURN                                                            REPG0012
      END                                                               REPG0013
```

*** UNGEN ***

```
      SUBROUTINE UNGEN(LBL)                                              UNGE0000
C*************************************************************************
C                                                                        ********
C     GENERATE CODE FOR 'UNTIL' UNIT                                     ********
C                                                                        ********
C*************************************************************************
C                                                                        ********
      INTEGER LBL                                                        UNGE0001
      INTEGER*2 TOKEN(80),INLAB(5),BLANK/' '/,TFLG,EOS/-2/               UNGE0002
C     COMMON BLOCKS                                                      ********
      COMMON /INLBL / INLAB                                              UNGE0003
      COMMON /TKN   / TOKEN                                              UNGE0004
      COMMON /TKNFLG/ TFLG                                               UNGE0005
      COMMON /SYSOUT/ ISYOUT                                             UNGE0006
C                                                                        ********
      CALL GETTOK                                                        UNGE0007
      TFLG=0                                                             UNGE0008
C     LABEL IS NOT ALLOWED WITH  UNTIL                                   ********
      DO 10 I=1,5                                                        UNGE0009
      IF (INLAB(I).NE.BLANK) GO TO 50                                    UNGE0010
10    CONTINUE                                                           UNGE0011
      GO TO 60                                                           UNGE0012
C     ERROR--LABEL ILLEGAL HERE                                          ********
50    CALL ERROR (1)                                                     UNGE0013
60    CONTINUE                                                           UNGE0014
C     CHECK FOR INFINITE REPEAT LOOP  (UNCONDITIONAL REPEAT)             ********
      IF (TOKEN(1).EQ.EOS) GO TO 200                                     UNGE0015
C     FOLLOWING PART GENERATES CODE FOR CONDITIONAL REPEAT               ********
      I=LBL+1                                                            UNGE0016
      CALL DGTCHR(I,INLAB)                                               UNGE0017
```

*** UNGEN ***

```
      DO 100 I=1,80                                                    UNGE0018
      IF (TOKEN(I).EQ.EOS) GO TO 150                                   UNGE0019
      CALL PUTCND (TOKEN(I))                                           UNGE0020
C     CONDITION IS BEING PUT IN CONBUF                                 ********
100   CONTINUE                                                         UNGE0021
150   CALL BALPAR                                                      UNGE0022
C     CODE 'IF (CONDITION) GO TO LBL' IS PRINTED                       ********
      CALL PUTIF(0,LBL)                                                UNGE0023
      WRITE  (ISYOUT,160) LBL                                          UNGE0024
160   FORMAT ('     *GO TO ',I5,55X,'++',6X)                           UNGE0025
      GO TO 300                                                        UNGE0026
C     CODE  ' LBL+1 GO TO LBL '                                        ********
200   CALL OUTGON(LBL+1,LBL)                                           UNGE0027
C     CODE  ' LBL+2 CONTINUE  '                                        ********
300   CALL OUTCON(LBL+2)                                               UNGE0028
      RETURN                                                           UNGE0029
      END                                                              UNGE0030
```

*** FORGEN ***

```
      SUBROUTINE FORGEN                                                 FORG0000
C*****************************************************************************
C                                                                       ********
C     PROCESS THE 'FOR' STATEMENT.                                      ********
C     INITIALIZATION, CONDITION AND REINITIALIZATION PARTS ARE          ********
C     SEPARATED BY ';', RE-INITIALIZATION PART IS PUT IN AREA           ********
C     RETEXP.                                                           ********
C                                                                       ********
C*****************************************************************************
C                                                                       ********
      INTEGER   I,J,K,LAB,LABGEN,NFOR,PTR,PTR1,PTR2                     FORG0001
      INTEGER*2 CONBUF(1254),TOKEN(80),INLAB(5),RETEXP(4,66)            FORG0002
      INTEGER*2 LEXFOR/-34/,LBRK/'('/,RBRK/')'/,SCOLN/';'/,EOS/-2/,     FORG0003
     *  BLANK/' '/,NSMCLN,NSTTOK,RRNFLG                                 FORG0004
C     COMMON BLOCKS                                                     ********
      COMMON /CONBLK/ CONBUF                                            FORG0005
      COMMON /CONPTR/ PTR                                               FORG0006
      COMMON /INLBL / INLAB                                             FORG0007
      COMMON /NMBFOR/ NFOR                                              FORG0008
      COMMON /NSTFLG/ NSTTOK                                            FORG0009
      COMMON /RINBLK/ RETEXP                                            FORG0010
      COMMON /RNFLG / RRNFLG                                            FORG0011
      COMMON /SYSOUT/ ISYOUT                                            FORG0012
      COMMON /TKN   / TOKEN                                             FORG0013
C                                                                       ********
      RRNFLG=0                                                          FORG0014
      CALL GETTOK                                                       FORG0015
      IF(NSTTOK.EQ.1) GO TO 1100                                        FORG0016
      IF(TOKEN(1).EQ.LBRK) GO TO 100                                    FORG0017
C     '(' MISSING                                                       ********
```

*** FORGEN ***

```
          CALL ERROR (2)                                                        FORG0018
          PTR=14                                                                FORG0019
C         LBRK MISSNG                                                           ********
          DO 10 I=1,80                                                          FORG0020
          IF(TOKEN(I).EQ.EOS) GO TO 100                                         FORG0021
          CALL PUTCND (TOKEN(I))                                                FORG0022
10        CONTINUE                                                              FORG0023
100       CALL BALPAR                                                           FORG0024
          NSMCLN=0                                                              FORG0025
          DO 200 I=15,PTR                                                       FORG0026
          IF(CONBUF(I).NE.SCOLN) GO TO 200                                      FORG0027
          NSMCLN=NSMCLN+1                                                       FORG0028
          IF(NSMCLN.EQ.1)  PTR1=I                                               FORG0029
          IF(NSMCLN.EQ.2) PTR2=I                                                FORG0030
200       CONTINUE                                                              FORG0031
          IF (NSMCLN.EQ.2) GO TO 300                                            FORG0032
C         ERROR--'SYNTAX ERROR IN 'FOR' STATEMENT'                              ********
          CALL ERROR (6)                                                        FORG0033
          PTR=14                                                                FORG0034
          RETURN                                                                FORG0035
C         3 GENERATED LABELS                                                    ********
300       LAB=LABGEN(3)                                                         FORG0036
          IF(CONBUF(PTR).EQ.PBRK) PTR=PTR-1                                     FORG0037
C         FOLLOWING PART PUTS THE RE-INITIALISTING PART IN AREA REIEXP          ********
          NFOR=NFOR+1                                                           FORG0038
          DO 350 I=1,66                                                         FORG0039
          REIEXP(NFOR,I)=BLANK                                                  FORG0040
350       CONTINUE                                                              FORG0041
          J=PTR2+1                                                              FORG0042
          K=1                                                                   FORG0043
```

*** FORGEN ***

```
      DO 400 I=J,PTR                                                    FORG0044
      RETEXP(NFOR,K)=CONBUF(I)                                          FORG0045
      IF(K.GE.66) GO TO 500                                             FORG0046
      K=K+1                                                             FORG0047
400   CONTINUE                                                          FORG0048
500   I=PTR1-14                                                         FORG0049
      K=15                                                              FORG0050
      DO 600 J=1,I                                                      FORG0051
      TOKEN(J)=CONBUF(K)                                                FORG0052
      IF(J.EQ.66) GO TO 700                                             FORG0053
      K=K+1                                                             FORG0054
600   CONTINUE                                                          FORG0055
      CALL CLRFLD(TOKEN,J,66)                                           FORG0056
700   WRITE  (ISYOUT,750) INLAB,(TOKEN(I),I=1,66)                       FORG0057
750   FORMAT (5A1,' ',66A1,'++',6X)                                     FORG0058
C     ABOVE IS TO PRINT INITIALISING STATEMENT                          ********
      I=LAB                                                             FORG0059
      CALL DGTCHR(I,INLAB)                                              FORG0060
      J=PTR2-PTR1-1                                                     FORG0061
      K=15                                                              FORG0062
      PTR1=PTR1+1                                                       FORG0063
      DO 1000 I=1,J                                                     FORG0064
      CONBUF(K)=CONBUF(PTR1)                                            FORG0065
      K=K+1                                                             FORG0066
      PTR1=PTR1+1                                                       FORG0067
1000  CONTINUE                                                          FORG0068
      PTR=J+14                                                          FORG0069
C     CODE  'IF ('NOT.(CONDITION)) GO TO LAB+2 '                        ********
      CALL PUTIF(1,LAB+2)                                               FORG0070
C     SET RE-RUN FLAG ON AND PUT THE 'FOR' ON STACK                     ********
```

*** FORGEN ***

```
      RRNFLG=1                                                  FORG0071
      CALL STACK (LEXFOR,LAB)                                   FORG0072
      RETURN                                                    FORG0073
1100  CALL ERROR (6)                                            FORG0074
      RETURN                                                    FORG0075
      END                                                       FORG0076
```

*** FOREND ***

```
      SUBROUTINE FOREND (LABS)                                          FORE0000
C*****************************************************************************
C                                                                       ********
C     GENERATE CODE FOR END OF FOR, REMOVING ENTRY FROM STACK           ********
C                                                                       ********
C*****************************************************************************
C                                                                       ********
      INTEGER   I,J,K,LABS,NFOR,ISYOUT                                  FORE0001
      INTEGER*2 REIEXP(4,66)                                            FORE0002
      COMMON/NMBFOR/NFOR                                                FORE0003
      COMMON/RINBLK/REIEXP                                              FORE0004
      COMMON/SYSOUT/ ISYOUT                                             FORE0005
C                                                                       ********
      I=LABS+1                                                          FORE0006
      J=LABS+2                                                          FORE0007
C     CODES TO BE GENERATED ARE                                         ********
C     'LABS+1 REINITIALIZATION'                                         ********
C     '       GO TO LABS      '                                         ********
C     'LABS+2 CONTINUE        '                                         ********
      WRITE  (ISYOUT,10) I,(REIEXP(NFOR,K),K=1,66),LABS,J               FORE0008
10    FORMAT (I5,' ',66A1,'++',6X/                                      FORE0009
     1         5X,' GO TO ',I5,55X,'++',6X/                             FORE0010
     2       I5,' CONTINUE',58X,'++',6X)                                FORE0011
      NFOR=NFOR-1                                                       FORE0012
      RETURN                                                            FORE0013
      END                                                               FORE0014
```

*** BRKNXT ***

```
      SUBROUTINE BRKNXT (LEXTKN,LEXS,LABS)                                BRKN0000
C*************************************************************************
C                                                                         ********
C     GENERATES CODE FOR BREAK ' GO TO LABS+1'                            ********
C                        NEXT  ' GO TO LABS  '                            ********
C                                                                         ********
C*************************************************************************
C                                                                         ********
      INTEGER LABS                                                        BRKN0001
      INTEGER*2 LEXTKN,LEXS,INLAB(5)                                      BRKN0002
      COMMON/INLBL/INLAB                                                  BRKN0003
      COMMON/SYSOUT/ISYOUT                                                BRKN0004
C                                                                         ********
      I=LABS                                                              BRKN0005
      IF((LEXTKN.EQ.-38).AND.(LEXS.NE.-33)) GO TO 50                      BRKN0006
C     BREAK/NEXT(BEGIN)                                                   ********
      I=I+2                                                               BRKN0007
      GO TO 100                                                           BRKN0008
50    I=I+1                                                               BRKN0009
100   WRITE (ISYOUT,150) INLAB,I                                          BRKN0010
150   FORMAT(5A1,' GO TO ',I5,55X,'++',6X)                                BRKN0011
      RETURN                                                              BRKN0012
      END                                                                 BRKN0013
```

*** OTHERS ***

```
      SUBROUTINE OTHERS (NLASTC)                                        OTHE0000
C*****************************************************************************
C                                                                       ********
C     THIS ROUTINE OUTPUTS A STATEMENT WHICH IS NOT CLASSIFIED          ********
C     AS ANY OF THE EXTENDED STATEMENTS, WITHOUT ANY CHANGE.            ********
C                                                                       ********
C*****************************************************************************
C                                                                       ********
      INTEGER   NLASTC,LASTC,LNTH,TKLNTH,ISYOUT                         OTHE0001
      INTEGER*2 BUF(80),TOKEN(80),RRNFLG,C                              OTHE0002
      INTEGER*2 BLANK/' '/,EOS/-2/,STAR/'*'/,ZERO/'0'/                  OTHE0003
C     COMMON BLOCKS                                                     ********
      COMMON /INBLK / BUF                                               OTHE0004
      COMMON /INPTR / LASTC                                             OTHE0005
      COMMON /PLENTH/ LNTH                                              OTHE0006
      COMMON /RNFLG / RRNFLG                                            OTHE0007
      COMMON /SYSOUT/ ISYOUT                                            OTHE0008
      COMMON /TKN   / TOKEN                                             OTHE0009
      COMMON /TKSIZE/ TKLNTH                                            OTHE0010
C                                                                       ********
      IF (TOKEN(1).EQ.EOS) RETURN                                       OTHE0011
      IF(RRNFLG.EQ.0) GO TO 8                                           OTHE0012
      IF (LNTH.EQ.0) GO TO 6                                            OTHE0013
      WRITE (ISYOUT,10) (TOKEN(I),I=1,LNTH)                             OTHE0014
      BUF(6)=STAR                                                       OTHE0015
      GO TO 25                                                          OTHE0016
6     IF (LASTC.NE.6) GO TO 7                                           OTHE0017
      WRITE (ISYOUT,10) (TOKEN(I),I=1,TKLNTH)                           OTHE0018
      GO TO 22                                                          OTHE0019
7     J=LASTC+1                                                         OTHE0020
```

*** OTHERS ***

```
      WRITE (ISYOUT,110) (TOKEN(I),I=1,TKLNTH),(BUF(I),I=J,72)          OTHE0021
      GO TO 20                                                          OTHE0022
8     IF (NLASTC.EQ .1) GO TO 25                                        OTHE0023
      WRITE (ISYOUT,10) (BUF(I),I=NLASTC,72)                            OTHE0024
10    FORMAT (6X,74A1)                                                  OTHE0025
20    LASTC=72                                                          OTHE0026
      CALL GETC(C)                                                      OTHE0027
22    C=BUF(6)                                                          OTHE0028
      IF((C.EQ.BLANK).OR.(C.EQ.ZERO)) GO TO 100                         OTHE0029
25    CALL PUTLIN(BUF)                                                  OTHE0030
      GO TO 20                                                          OTHE0031
100   LASTC=6                                                           OTHE0032
      RETURN                                                            OTHE0033
110   FORMAT (6X,66A1,8X/5X,'*',66A1,8X)                                OTHE0034
      END                                                               OTHE0035
```

*** CLRFLD ***

```
      SUBROUTINE CLRFLD (FIELD,N1,N2)                                   CLRF0000
C*****************************************************************************
C                                                                       ********
C     THIS ROUTINE BLANKS THE FIELD FROM POSITION N1 TO (2              ********
C                                                                       ********
C*****************************************************************************
C                                                                       ********
      INTEGER N1,N2,I                                                   CLRF0001
      INTEGER*2 BLANK/' '/,FIELD(80)                                    CLRF0002
C                                                                       ********
      DO 10 I=N1,N2                                                     CLRF0003
      FIELD(I)=BLANK                                                    CLRF0004
10    CONTINUE                                                          CLRF0005
      RETURN                                                            CLRF0006
      END                                                               CLRF0007
```

*** PUTSMT ***

```
      SUBROUTINE PUTSMT(N)                                                PUTS0000
C*************************************************************************
C                                                                         ********
C     OUTPUTS THE PART OF IN-BUFFER RIGHT TO THE POINTER 'N'              ********
C     AS A COMMENT.                                                       ********
C                                                                         ********
C*************************************************************************
C                                                                         ********
      INTEGER*2 BUF(80)                                                   PUTS0001
      COMMON/INBLK/BUF                                                    PUTS0002
      COMMON /SYSOUT/ ISYOUT                                              PUTS0003
C                                                                         ********
      IF (N.LE.6) N=6                                                     PUTS0004
      WRITE (ISYOUT,10) (BUF(I),I=N,80)                                   PUTS0005
10    FORMAT ('C+    ',75A1)                                              PUTS0006
      RETURN                                                              PUTS0007
      END                                                                 PUTS0008
```

*** PUTLIN ***

```
      SUBROUTINE PUTLIN (STRING)                                        PUTL0000
C*****************************************************************************
C                                                                       ********
C     THIS ROUTINE PRINTS STRING AS SUCH.                               ********
C                                                                       ********
C*****************************************************************************
C                                                                       ********
      INTEGER*2 BLANK /' '/,  STRING(80)                                PUTL0001
      COMMON /SYSOUT/ ISYOUT                                            PUTL0002
C                                                                       ********
      WRITE  (ISYOUT,10) STRING                                         PUTL0003
10    FORMAT (80A1)                                                     PUTL0004
      DO 20 I=1,80                                                      PUTL0005
      STRING(I)=BLANK                                                   PUTL0006
20    CONTINUE                                                          PUTL0007
      RETURN                                                            PUTL0008
      END                                                               PUTL0009
```

*** LABCON ***

```
      SUBROUTINE LABCON (LABL)                                          LABC0000
C*****************************************************************************
C                                                                       ********
C     GENERATE  CODE 'LABL CONTINUE'                                    ********
C                                                                       ********
C*****************************************************************************
C                                                                       ********
      INTEGER*2 LABL(5)                                                 LABC0001
      COMMON /SYSOUT/ ISYOUT                                            LABC0002
C                                                                       ********
      WRITE  (ISYOUT,10) LABL                                           LABC0003
10    FORMAT (5A1,' CONTINUE',58X,'++',6X)                              LABC0004
      RETURN                                                            LABC0005
      END                                                               LABC0006
```

*** OUTGON ***

```
      SUBROUTTINE OUTGON(N1,N2)                                          OUTG0000
C*************************************************************************
C                                                                        ********
C     THIS ROUTINE GENERATES THE CODE ' N1 GO TO N2'                     ********
C                                                                        ********
C*************************************************************************
C                                                                        ********
      INTEGER N1,N2                                                      OUTG0001
      COMMON /SYSOUT/ISYOUT                                              OUTG0002
C                                                                        ********
      WRITE (ISYOUT,10) N1,N2                                            OUTG0003
10    FORMAT (I5,' GO TO ',I5,55X,'++',6X)                               OUTG0004
      RETURN                                                             OUTG0005
      END                                                                OUTG0006
C                                                                        ********
      SUBROUTTINE OUTCON (N)                                             OUTG0007
C*************************************************************************
C                                                                        ********
C     THIS ROUTINE PRINTS THE CODE ' N CONTINUE'                         ********
C                                                                        ********
C*************************************************************************
C                                                                        ********
      INTEGER N                                                          OUTG0008
      COMMON /SYSOUT/ ISYOUT                                             OUTG0009
C                                                                        ********
      WRITE (ISYOUT,10) N                                                OUTG0010
10    FORMAT(I5,' CONTINUE',58X,'++',6X)                                 OUTG0011
      RETURN                                                             OUTG0012
      END                                                                OUTG0013
```

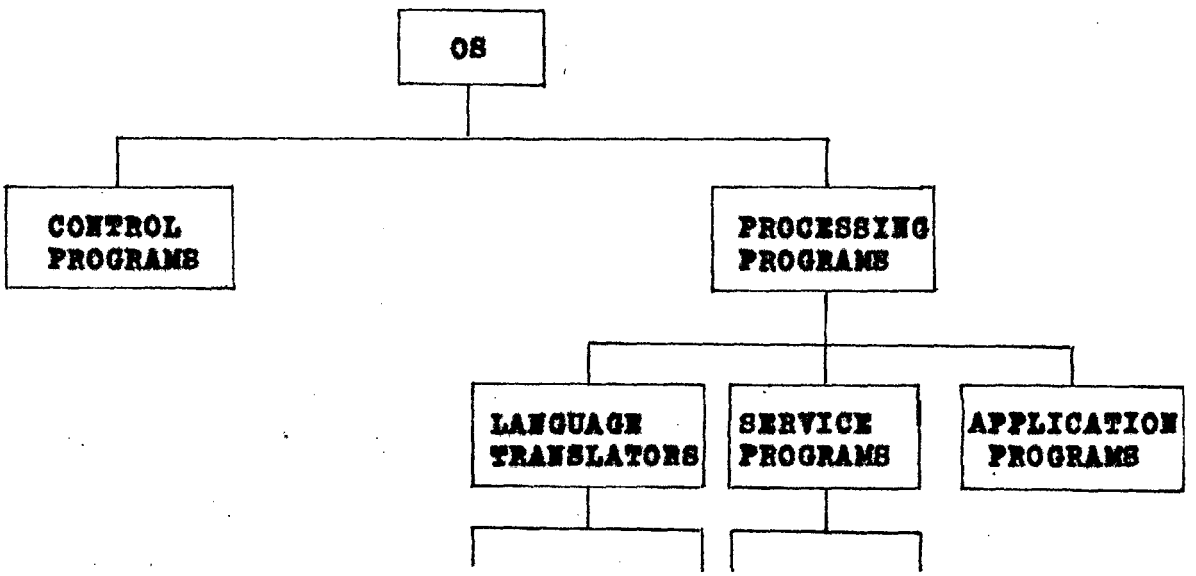
APPENDIX B

OPERATING SYSTEM AND JOB CONTROL LANGUAGE

OPERATING SYSTEM (OS)

The operating system (OS) introduces programs to the computing system, initiates their execution, and provides all the resources and services they require.

The programs and routines that compose the operating system are classified as control programs and processing programs as shown in the following figure.



The three main functions of the control programs are to accept and schedule jobs in a continuous flow (job management), to supervise each unit of work to be done (task management), and to simplify retrieval of all data, regardless of the way it is organised and stored (data management). The processing programs consist of language

translators (such as FORTRAN, PL/I, COBOL compilers), service programs (such as the linkage editor, loader), and application programs (such as user programs).

The operating system with which EXTFOR has been interfaced is the OS-MFT (Multiprogramming with a Fixed number of Tasks) variation of OS. Several jobs can run concurrently under MFT.

OS-JCL (JOB CONTROL LANGUAGE)

JCL serves as a means of communication between the user's program and the system. JCL statements tell the system how to execute a job, request needed system facilities and describe required input/output(I/O) devices.

The normal JCL statements (or cards) are:

1. The JOB statement. This is the first control card; it marks the beginning of a job.

2. The EXEC (Execute) statement. This card follows the JOB card and names the program or procedure to execute.

3. The DD (Data Definition) statement. This card describes each data set (a file on tape or direct access device, or a deck of cards), and requests the allocation of I/O devices.

4. The Delimiter (/*) statement.

   This is the "end-of-file" card for marking the end of a card deck.

Other JCL statements less often used are:

5. The Null (//) statement. The null may be used to mark the end of a job.

6. The comment (//*) statement. Comments may be coded in rest of the columns as an aid in documenting JCL.

7. The PROC statement. This card assigns default values for symbolic parameters in cataloged procedures.

8. The PEND statement. This card terminates the definition of a procedure.

9. The Command Statement. This card may be used by operators to enter operator commands from the input stream.

All the JCL statements (except the /* card) begin with a // in columns 1 and 2, followed by a name field, an operation field, and an operand field. The name field begins immediately after the second slash, while the name, operation, and operand fields are separated from each other by one or more blank spaces.

//name operation operand.

The name field identifies the control card so that other cards or system control blocks can refer to it. It can range from 1 to 8 characters in length and can contain any alphanumeric or national (@$#) characters. However, the first character of the name must be alphabetic and in column 3.

The operation field specifies the type of control card: JOB, EXEC, DD, PROC, PEND, or an operator command.

The operand field contains parameters separated by commas.

The JOB, EXEC, and DD cards enable a complete job to be executed. As an example, suppose that some FORTRAN source language statements are to be compiled. Assume the FORTRAN compiler is a system program named IEYFORT which requires three data sets: an input data set consisting of source language statements to compile, an output data set to print the compilation listing, and an output data set to contain the object module. Such a program might be executed by the following JCLs.

```
//TEST        JOB
//FORT        EXEC  PGM=IEYFORT,PARM=(SOURCE,NOMAP)
//SYSPRINT    DD    SYSOUT=A
//SYSLIN      DD    DSNAME=&LOADST,DISP=(NEW,PASS),
//                  UNIT=2311,SPACE=(80,(200,100),RLSE),
//                  DCB=BLKSIZE=80,VOL=SER=LIB400
//SYSIN       DD  *
                  (source language statements to compile)
/*
//
```

The job is named TEST; the FORTRAN compiler named as IEYFORT is executed in a job step named FORT. PARM specifies the parameters SOURCE and NOMAP to be passed to compiler.

```
//SYSPRINT    DD    SYSOUT=A
```

SYSOUT=A defines a print data set. The SYSOUT keyword instructs the system to queue the output on direct excess storage. A is the class of output devices defined by an installation, traditionally, SYSOUT=A is a printer.

The SYSLIN DD card defines a data set on direct access device 2311 volume name as LIB400, DSNAME specifies the name of the data set &LOADST with disposition NEW and

PASS which says that after the job step FORT the data set &LOADST to be kept for the use of next job step (say linkage). The data sets with first character as "&" are temporary and deleted by the system as soon as the job is over. The SPACE parameter tells about the space to be allocated to the data set.

SPACE=(80,(200,100)RLSE) requests 200 blocks of 80 bytes each with an increment of 100 blocks for the secondary allocation in case the primary allocated space is full, maximum 15 secondary allocations can be given to a data set. RLSE releases all unused space when the data set is closed.

The DCB(Data Control Block) gives parameter for the data control block which is a table of data in core that describes each data set used by the program. The DCB parameters are LRECL (logical record length), BLKSIZE (blocksize), BUFNO (Number of buffers) and, RECFM (record format).

//SYSIN DD *

The asterisk (*) is a special code telling the system that data cards immediately follow the DD card. The end of the card deck is indicated by /* card.

CATALOGED PROCEDURE

JCL statements direct the operating system on the processing to be done on a job, and describe all I/O units required. Since these JCL statements are numerous and complex, the cards for frequently used procedures are kept on direct storage as a member of procedure library

named SYS1.PROCLIB. The user invokes these cataloged procedures by giving the system the name of the cataloged procedure rather than submitting the JCL cards. For example, a FORTRAN compilation linkage and execution job contains as many as 20 JCL cards, rather these statements are cataloged with a name FORTCLG. With use of cataloged procedure the JCL cards reduce to

```
//TEST  JOB
//       EXEC  FORTCLG
  (the FORTRAN compile, link edit, go procedure
   is invoked)
//FORT.SYSIN  DD  *
[ source deck ]
/*
//GO.SYSIN    DD  *
[ data cards ]
/*
//
```

When a procedure name is found in JCL statements the system searches the procedure library and the JCL statements are substituted in place of the procedure name, taking the parameters as specified with procedure name.

The procedure is defined by enclosing a set of JCLs between PROC and PEND statements. For example:

```
//RUN    PROC   NAME=LIB1
//STEP1 EXEC    PGM=ONE
//A      DD     DSN=&NAME, ---
//STEP2 EXEC    PGM=TWO
//B      DD     DSN=&NAME, ---
//       PEND
```

This procedure is cataloged in system procedure

library by adding it using the utility IEBUPDTE. Whenever the procedure is referred as

```
//      EXEC    RUN
```

in the JCLs, the procedure is substituted with each appearance of the symbolic parameter &NAME assigning a default value of LIB1. One might override this default for a run as

```
//      EXEC    RUN,PARM=USELIB
```

Each appearance of &NAME in the procedure is replaced by USELIB.

PROC FOR EXTFOR PREPROCESSOR

The complete process of translation from EXTFOR program to the execution may be divided into the following four job steps.

- Preprocessing
- Compilation
- Link Edit
- Execution

To perform these job steps, separate sets of JCL are defined. These job steps constitutes a procedure called FORTPCLG which is cataloged in procedure library SYS1.PROCLIB. This procedure can be used to perform all the 4 steps. In all 4 procedures are cataloged:

FORTP -For preprocessing an EXTFOR program.

FORTPC -For preprocessing and compilation of an EXTFOR program.

FORTPCL -For preprocessing, compilation and linkage edit of an EXTFOR program.

FORTPCLG-For preprocessing to execution of an EXTFOR program.

APPENDIX C

OS - EXTFOR INTERFACE AND OPERATING PROCEDURE

The FORTRAN source of EXTFOR preprocessor is compiled under the local OS environment. After linkage edit the load module which is now in executable form, is stored permanently as a member of processing program library of OS. The load module is preferably stored on the fastest device available. A temporary data set (file) is allocated each time the EXTFOR is loaded, to store the output of EXTFOR. The FORTRAN reads the input source (output of EXTFOR) from this temporary data set; rest of the procedure is same as followed for FORTRAN compilation.

The EXTFOR developed has been interfaced with OS-MFT on EC-1030 System at ORG Systems, Baroda. The EXTFOR load module (JFPREPRO) is stored as a member of library P.LOADLIB on direct access device 2311, volume serial no. LIB400 (system pack). The temporary data set &SRCSET on LIB400 is allocated to contain the output of EXTFOR. The FORTRAN compiler, which has IEYFORT as system name takes the further processing as desired by the user. The following procedure FORTPCLG is defined and cataloged to preprocess, compile, linkage edit, and finally execute the EXTFOR source program.

```
//FORTPCLG PROC NAME=''''NAME=MAIN''',SOURCE=SOURCE,LIST=NOLIST,        X
//             DECK=NODECK,MAP=MAP,LOAD=LOAD,LCNT=,,,LINECNT=50;'',    X
//             LXREF=XREF,LLET=LET,LLIST=LIST
//**
//**       STEP TO PREPROCESS THE EXTFOR SOURCE PROGRAM          **
//FORTP    EXEC PGM=JFPREPRO,COND=EVEN,TIME=2
//STEPLIB  DD   DSN=P.LOADLIB,UNIT=2311,DISP=SHR,VOL=SER=LIB400
//FT06F001 DD   SYSOUT=A
//SYSPRINT DD   SYSOUT=A
//FT01F001 DD   DDNAME=SYSIN
//**       EXTFOR SOURCE ON CARDS        **
//FT03F001 DD   DSN=&SRCSET,DISP=(NEW,PASS),UNIT=2311,                 X
//             SPACE=(CYL,(2,5,1),RLSE),                               X
//             DCB=(RECFM=FB,LRECL=80,BLKSIZE=1600),                   X
//             VOL=SER=LIB400
//**
//**       STEP TO COMPILE FORTRAN CODE       **
//FORT     EXEC PGM=IEYFORT,                                           X
//             PARM=(&NAME,&SOURCE,&LIST,&DECK,                        X
//             &MAP,&LOAD,&LCNT),COND=(4,LT,FORTP)
//SYSPRINT DD   SYSOUT=A
//SYSLIN   DD   DSNAME=&LOADSET,DISP=(NEW,PASS),UNIT=2311,             X
//             SPACE=(80,(200,100),RLSE),DCB=BLKSIZE=80,               X
//             VOL=SER=LIB400
```

```
//SYSIN     DD   DSNAME=&SRCSET,DISP=(OLD,DELETE)
//              DDNAME=SYSIN
//**        SYSIN IS DATA SET NAME USER CAN CONCATENATE       **
//**        WITH PREPROCESSOR OUTPUT                          **
//**
//**        LINKAGE EDIT STEP .     **
//LKED      EXEC PGM=IEWL,PARM=(&LIREP,&LLET,&LLIST),                    X
//              COND=((4,LT,FORTP),(4,LT,FORT))
//SYSPRINT  DD   SYSOUT=A
//SYSLIB    DD   DSNAME=SYS1.FORTLIB,DISP=SHR
//SYSLMOD   DD   DSNAME=&GOSET(MAIN),DISP=(NEW,PASS),UNIT=2311,          X
//              SPACE=(1024,(20,10,1),RLSE),DCB=BLKSIZE=1024,            X
//              VOL=SER=LIB400
//SYSUT1    DD   DSNAME=&SYSUT1,UNIT=2311,SPACE=(1024,(20,10),RLSE),     X
//              DCB=BLKSIZE=1024,VOL=SER=LIB400
//SYSLIN    DD   DSNAME=&LOADSET,DISP=(OLD,DELETE)                       X
//          DD   DDNAME=SYSIN
//GO        EXEC PGM=*.LKED.SYSLMOD,                                     X
//              COND=((4,LT,FORTP),(4,LT,FORT),(4,LT,LKED))
//FT03F001  DD   SYSOUT=A
//FT01F001  DD   DDNAME=SYSIN
//FT06F001  DD   SYSOUT=A
//          PEND
```

To execute the EXTFOR program following set of JCLs is used:

```
//JOBNAM  JOB
//        EXEC  FORTPCLG
//FORTP.SYSIN   DD  *
[EXTFOR source cards]
/*
//GO.SYSIN      DD  *
[input card data]
/*
//
```

User can have EXTFOR source, the input and output data files (data sets) on any media. For source, the DD (Data Definition) card with name FORTP.SYSIN is replaced by DD of data set on which the source program exists. GO.SYSIN DD card defines the input file. The input/output files can be used in program with DD definition as:

```
//GO.FTnnF001   DD  operands
```

The nn is the file reference number used in the program.

However, user can define his own set of procedures and data set definitions to fit his requirements and OS environment.

Following is the EXTFOR sample program:

```
IEF387 DIRECT SYSOUT=A.
//SAMPLE1 JOB
//  EXEC FORTPCLG
//SYSIN  DD  *
```

```
** STATISTICS AT PREPROCESSOR LEVEL **

  0 DIAGNOSTICS GENERATED
```

```
IEF373I STEP /FORTP    / START 80363.1408
IEF374I STEP /FORTP    / STOP  80363.1409 CPU    0MIN 13.58SEC MAIN  54K
```

```
      C*************************************************************************
      C     THIS PROGRAM SORTS A SEQUENCE OF N NUMBERS USING SHELL SORT METHOD *
      C     ILLUSTRATES USE OF FOR,WHILE,DO,REPEAT AND BREAK STATEMENT         *
      C*************************************************************************
      C
0001        INTEGER A(20)
0002        READ  (1,10) N,A
0003        CALL SHLSRT (A,N)
0004        WRITE  (3,20) A
0005   10   FORMAT (21I3)
0006   20   FORMAT (1H ,20X,'THE SORTED SEQUENCE IS'/
           *         1H ,20X,10I5/1H ,20X,10I5)
0007        STOP
0008        END
```

SUBPROGRAMS CALLED

SYMBOL	LOCATION	SYMBOL	LOCATION	SYMBOL	LOCATION	SYMBOL	LOCATION	SYMBOL	LOC
BCOM#	90	SHLSRT	94						

SCALAR MAP

SYMBOL	LOCATION	SYMBOL	LOCATION	SYMBOL	LOCATION	SYMBOL	LOCATION	SYMBOL	LOC
	A0								

ARRAY MAP

SYMBOL	LOCATION	SYMBOL	LOCATION	SYMBOL	LOCATION	SYMBOL	LOCATION	SYMBOL	LOC
	A4								

FORMAT STATEMENT MAP

SYMBOL	LOCATION	SYMBOL	LOCATION	SYMBOL	LOCATION	SYMBOL	LOCATION	SYMBOL	LOC
10	F4	20	FA						

```
*OPTIONS IN EFFECT*   ID,EBCDI,SOURCE,NOLIST,NODECK,LOAD,MAP
*OPTIONS IN EFFECT*   NAME = MAIN      , LINECNT =      50
*STATISTICS*     SOURCE STATEMENTS =          8,PROGRAM SIZE =      494
*STATISTICS*  NO DIAGNOSTICS GENERATED
```

```
0001                SUBROUTINE SHLSRT (SEQ,NUM)
0002                INTEGER SEQ(20),NUM,DIST,TEMP
              C+    FOR (DIST=4;(DIST.LT.NUM);DIST=DIST+DIST)
0003                DIST=4                                                  ++
0004              90000 IF(.NOT.(DIST.LT.NUM)) GO TO 90002                  ++
0005              90001 DIST=DIST+DIST                                      ++
0006                GO TO 90000                                             ++
0007              90002 CONTINUE                                            ++
0008                DIST=DIST-1
              C+    WHILE (DIST.GT.1) DIST=DIST/2
0009              90004 IF(.NOT.(DIST.GT.1)) GO TO 90005                    ++
0010                DIST=DIST/2
0011                NO=NUM-DIST
0012                DO 90006 K=1,NO                                         ++
0013                I=K
              C+    REPEAT
0014              90008 CONTINUE                                            ++
0015                J=I+DIST
              C+    IF (SEQ(I).LE.SEQ(J)) THEN BREAK
0016                IF(.NOT.(SEQ(I).LE.SEQ(J))) GO TO 90012                 ++
              C+    BREAK
0017                GO TO 90010                                             ++
0018              90012 CONTINUE                                            ++
0019                TEMP=SEQ(I)
0020                SEQ(I)=SEQ(J)
0021                SEQ(J)=TEMP
0022                I=I-DIST
              C+    UNTIL (I.GE.1)
0023              90009 IF(I.GE.1)                                          ++
                     *GO TO 90008                                           ++
0024              90010 CONTINUE                                            ++
              C+    END
0025              90006 CONTINUE                                            ++
0026              90007 CONTINUE                                            ++
              C+    END
0027              90003 GO TO 90004                                         ++
0028              90005 CONTINUE                                            ++
0029                RETURN
0030                END
```

SCALAR MAP

SYMBOL	LOCATION	SYMBOL	LOCATION	SYMBOL	LOCATION	SYMBOL	LOCATION	SYMBOL	LO
DIST	0C	NUM	E0	N0	E4	K	E8	I	
J	F0	TEMP	F4						

ARRAY MAP

SYMBOL	LOCATION	SYMBOL	LOCATION	SYMBOL	LOCATION	SYMBOL	LOCATION	SYMBOL	LO
SEQ	F8								

```
*OPTIONS IN EFFECT*   ID,OKOI,SOURCE,NOLIST,NODECK,LOAD,MAP
*OPTIONS IN EFFECT*   NAME = SHLSRT   , LINECNT =      50
*STATISTICS*     SOURCE STATEMENTS =      30,PROGRAM SIZE =     724
*STATISTICS*  NO DIAGNOSTICS GENERATED

*STATISTICS*  NO DIAGNOSTICS THIS STEP
```

```
IEF373I STEP /FORT     / START 80363.1409
IEF374I STEP /FORT     / STOP  80363.1409 CPU    0MIN 12.84SEC MAIN 106K
```

```
IEF373I STEP /L*FD     / START 80363.1409
IEF374I STEP /L*FD     / STOP  80363.1410 CPU    0MIN 07.68SEC MAIN  68K
//GO.SYSIN DD *
                    THE SORTED SEQUENCE IS
                       2    3    4    7    8    8   21   32   35   59
                      62   83   95  123  236  365  532  568  586  986
```

```
IEF373I STEP /GO       / START 80363.1410
IEF374I STEP /GO       / STOP  80363.1411 CPU    0MIN 03.76SEC MAIN  48K
IEF375I  JOB /SAMPLE1 / START 80363.1408
IEF376I  JOB /SAMPLE1 / STOP  80363.1411 CPU    0MIN 37.86SEC
```

ERROR MESSAGES

The EXTFOR preprocessor gives error messages in the following format:

C****nn) JFPmmmI text of the message

nn - the serial number of error in the current statement.

mmm - the error code number.

The text of the message may be any one of the following list.

Code	Message Text
1	WARNING--LABEL ILLEGAL HERE

In some of the EXTFOR extended features the statement label is not allowed, for example labelled ELSE, UNTIL parts of control structures.

```
      IF (AMONT.GT.1000) THEN BREAK
10    ELSE  INTRST = (AMONT*RATE*YRS)/100
```

The statement number 10 with ELSE is not allowed.

2 '(' - MISSING

The condition part of IF, FOR, WHILE, UNTIL is not preceded by '('.

3 UNMATCHING '('S

The number of '(' and ')' are not same in condition part of the control structure. For example

```
      IF ((LASTC.LT.72).AND.(C.NE.BLANK)$THEN
                          TOKEN(I) = C
```

The $ points where a ')' should have been to match the outermost '('.

4 UNRECOGNIZABLE STATEMENT
The current statement has the first symbol as a valid keyword but the rest of the statement results an unpredicatable syntax.

5 TOO MANY NESTED 'FOR' STATEMENTS
Maximum of 4 nested 'FOR' are allowed.

6 SYNTAX ERROR IN 'FOR' STATEMENT
Self explanatory.

7 TOO LARGE A CONDITION/STATEMENT
The maximum length of a condition or a statement is 1254 characters.

8 CONDITIONAL 'END/ELSE/UNTIL'
These keywords should not be used in the conditional part of a control structure. For example

```
IF (X(I).LT.0) THEN END
                    ELSE UNTIL
```

9 UNMATCHING ELSE/UNTIL
Self explanatory
Example:

```
IF (I.EQ.10) THEN BREAK
A = 2
DO I=1,5
B(I)=0
ELSE -- error
END
```

10 UNEXPECTED END OF SUBPROGRAM
The source text is over or a new subprogram has appeared in the program without the matching END for WHILE, DO or BEGIN structures in the previous subprogram.

12 ILLEGAL LEVEL SPECIFIED

The level in BREAK or NEXT statement is not positive integer. For example

```
BREAK - 5
NEXT  A
```

13 FORTRAN 'DO' STATEMENT ILLEGAL HERE

The FORTRAN DO statement is found just after THEN or ELSE in the IF statement or in FOR statement which is not allowed in EXTFOR.

14 BREAK/NEXT ILLEGAL HERE

The statement BREAK or NEXT is encountered without an enclosing loop or block structure.

15 SYNTAX

The error message points to the symbol in the statement by a "$" where the error is encountered.

For errors at FORTRAN compilation level, refer to the FORTRAN manual.

At the end the EXTFOR specifies the number of errors encountered during preprocessing the source program. The message is

```
** STATISTICS AT PREPROCESSOR LEVEL **
nnn  DIAGNOSTICS GENERATED
```

REFERENCES

[1] Aho and Ullman, 'Principle of Compiler Design', Addison-Wesley Publishing Company, New York.

[2] B.W. Kernighan, 'RATFOR - a Preprocessor for Rational FORTRAN', Software-Practice and Experience, 5,No.4, 395-406(1975).

[3] B.W. Kernighan, and P. Plauger,'Software Tools', Addison-Wesley Publishing Company, New York.

[4] Cook, J.A. and Shustek, L.J., 'A User's Guide to MORTRAN2', Computation Research Group, SLAC, GM No. 165(1975).

[5] C28-6539, IBM SYSTEM/360 OS - JOB CONTROL LANGUAGE

[6] C28-6817, IBM SYSTEM/360 FORTRAN IV (G and H) Programmer's Guide.

[7] Dahal O.J., C.A.R. Hoare and E.W. Dijkstra, 'Notes on Structured Programming', Academic Press, New York 1972.

[8] David Gries, 'Compiler Construction for Digital Computers', Wiley, New York.

[9] Dijkstra, E.W., 'A Discipline of Programming', Prentice-Hall.

[10] Douglas Comer, 'MOUSE4: An Improved Implementation of the RATFOR Preprocessor', Software-Practice and Experience, vol.8, 35-40 (1978).

[11] Irving B. Elliolt, 'DLP, A Design Language Pre-Processor', SIGPLAN Notices, vol.14, No.2, Feb 1979 (14-20).

[12] Makoto Arisawa, and Minoru Iuchi, 'FORTRAN + Preprocessor = Utopia 84', Computer Science Deptt., Yamanashi University, Kofu, Japan, SIGPLAN Notices, vol.4, No.1, Jan. 1979 (Page 14).

[13] Meeson,R. and Arther Pyster, 'Overhead in FORTRAN Preprocessors', Software - Practice and Experience, vol.9, 987-999 (1979).

[14] Santa Barbara, 'IFTRAN-3 User's Guide', General Research Corporation, CA (1978).

[15] Terry Beyer, 'FLECS: User's Manual', Deptt. of Computer Sciences, University of Oregon, (1975).

[16] Ullman, J.D., 'Fundamental Concepts of Programming Systems', Addison-Wesley Publishing Company,N. York.