AN EVALUATION OF ML/I (APS) MACROS FOR STRUCTURED FORTRAN EXTENSIONS

by

Soo Kyung Park

B.A., Ewha Komesans University, 1968

A MASTER'S REPORT

Submitted in partial fulfillment of the requirements for the degree

MASTER OF SCIENCE

Department of Computer Science

KANSAS STATE UNIVERSITY
Manhattan, Kansas
1977

Approved by:

[Signature]
Major Professor
TABLE OF CONTENTS

I. Introduction
   A. Purpose of Paper .............................................. 1
   B. Organization .................................................. 2

II. Macro Preprocessor for FORTRAN Extensions
   A. Design Considerations ....................................... 4
   B. Control Structures .......................................... 6
   C. Use of the ML/I Macro Preprocessor ......................... 10

III. Implementation
   A. Introductory Features of ML/I .............................. 13
   B. Definition of Macros ......................................... 21
   C. How to Execute the Preprocessor ......................... 32
   D. User Constraints ............................................. 37

IV. Evaluation
   A. Problems of Macro Definition ............................ 44
   B. Time and Size of Macros .................................. 56
   C. Conclusions ................................................ 56

APPENDIX
   A. Sample Structured FORTRAN program(TEST SFORT) .... A- 1
   B. Macro Listings(SOONMACRO EPS) .......................... A- 3
   C. Listing of SOO EXEC procedure ........................... A- 5
   D. Listing of a Sample EPSIN EPS ........................... A- 7
   E. Listings of Sample Structured FORTRAN output
      from EPS II ................................................ A-10
ILLEGIBLE DOCUMENT

THE FOLLOWING DOCUMENT(S) IS OF POOR LEGIBILITY IN THE ORIGINAL

THIS IS THE BEST COPY AVAILABLE
FIGURES

Figure II.1 For Use of ML/I Macro Preprocessor .......... 11
Figure III.1 The Description of the File EPSIN EPS ...... 34
Figure III.2 Files Processed by SOO EXBC Using EPS II .. 35
Figure IV.1 NEXT/QUIT Examples ........................... 46
Figure IV.2 IF...FI Examples ............................... 52
Figure IV.3 Examples for CASE...ENDCASE ............... 54
I. INTRODUCTION

A. PURPOSE OF PAPER

The objective of this project is to investigate the use of a general purpose macro processor to implement structured programming extensions to FORTRAN.

The use of macro processors has developed since the 1950's. Using macro processors to modify or extend a compiler language has since occurred to many programmers. Text replacement has been the most distinguishing property of macro processors.

ML/I (Bro67) is a powerful macro processor for conveniently extending source languages. It has been used as a preprocessor to several compilers and assemblers. Its operation is to interprete macro definitions and translate input text to output text. Input to the macro processor is in the form of macro definitions and source text. The output text is derived from the source text by replacing all the macro calls that occur in it.

A version of ML/I, EPS II, is now available at KSU. This system is a load module written in IBM system/360 assembler language. EPS II can be run interactively under CMS.

The idea of introducing structured programming extensions of FORTRAN is not new. There has been a growing awareness of the need to improve unstructured FORTRAN since the 'structured programming' became an important issue in the computer world.
Many well-formed FORTRAN extensions have been defined by eliminating the GO TO and using only sequential statements and control structures in the form of block structures. By using ML/I, structured programming can be implemented to FORTRAN without changing over to a completely new compiler. For this study, a set of macro extensions to FORTRAN will be defined and tested through EPS II. The main study area for structured programming will be focussed on IF THEN ELSE, CASE, WHILE, etc. Also, the efficiencies in terms of time using the macro processor will be investigated and evaluated.

B. ORGANIZATION

The purpose of this paper has been briefly introduced at the beginning. The remainder of this paper is organized as follows. The second section presents the design impact of structured extension to FORTRAN. The extended control structures are described with the use of the macro preprocessor. The third section discusses the implementation procedures which could be viewed as the documentation for the user. Included is explanation of selected features of ML/I. All the macro definitions will be displayed and the process of execution will be shown. In the fourth section, use of ML/I will be evaluated. The problems in defining macros, timing and size, etc, will be topics of discussion. The possibility of further development for FORTRAN extensions using ML/I and future trends
will also be mentioned. The last part of this paper is an appendix which includes a list of macros defined and a test program listing.
II. MACRO PREPROCESSOR FOR FORTRAN EXTENSIONS

A. DESIGN CONSIDERATIONS

Since the term 'structured programming' was first used by Dijkstra (Dij72), the trend has been away from the study of low level languages towards the study of higher levels of programming language structures. A great deal has been published recently about the theory of structured programming.

A broadly drawn list of ideas that have been considered to be aspects of structured programming was given by Abrahams (Abr75). Among those ideas, the notion that GO TO statements must be eliminated became one of the vital points in structured programming. In other words, a high density of GO TO's in a program generally indicates poor programming. So, there has been considerable interest in the possibility of replacing GO TO statements in a program with structured control statements such as IF THEN ELSE, CASE, DO WHILE, to make programs easier to compose, understand and modify.

Without exception, the elimination of GO TO movement has had a strong effect on FORTRAN. Many efforts have been made to define a well-formed FORTRAN through modification or extension. So, instead of attempting the extensive modification of a existing FORTRAN compiler, many language designers invented a FORTRAN-like structured language. By producing a preprocessor or a translator, programs written in this structured language are converted into statements that any existing FORTRAN compiler will accept.
A number of preprocessors and other extensions of FORTRAN have been implemented, such as MORTRAN (Coo76), a structured FORTRAN translator (Hig75), and SFOR which is a precompiler for the implementation of a FORTRAN-based structured language (O'n74). Many attempts have been made to implement the FORTRAN extensions using a macro processor.

ML/I is a macro preprocessor designed for portability. It is written in the descriptive language, LOWL, which is similar to a high-level language. It can be translated into the assembly language of any computer or into a high level language. ML/I was designed for transportability between different computers with differing architectures. Using the bootstrapping technique ML/I has been implemented on a large number of machines. One of these implementations is EPS II on IBM system/360.

ML/I, a general purpose macro processor, is intended to allow the user to extend any language. The fact that ML/I is independent of any base language provides the possibility that it could be used as a preprocessor to the FORTRAN compiler.

The main consideration in this paper is to develop the well-formed structured FORTRAN extensions and to determine the suitability of ML/I for FORTRAN extensions. ML/I is the appropriate preprocessor to use for this study because of its base language independence.
B. CONTROL STRUCTURES

The major constructs of structured FORTRAN extensions which will be implemented by ML/I fall into two categories. One is looping control structures such as LOOP...ENDLOOP, QUIT...LUP and NEXT...LUP. The second category is selection control structures such as IF...FI, SIF...FI and CASE...ENDCASE. The design of these structures is effected by macro definitions. The listing of a sample structured FORTRAN program is included in the Appendix A. The following terms are used:

- Li --- statement label, i=1,2,...,n
- Lexpi --- logical expression, i=1,2,...,n
- Si --- block of statements, i=1,2,...,n

**Looping control structures:**

The repetition statements are tested within looping structure. If the value of the logical expression is changed within the block, the loop immediately terminates. An abnormal exit is also provided. The statement has the following form with the right side target FORTRAN code being generated by the left side statements.

1. LOOP...ENDLOOP:

<table>
<thead>
<tr>
<th>Source Pattern</th>
<th>Target FORTRAN</th>
</tr>
</thead>
<tbody>
<tr>
<td>LOOP L1 WHILE Lexp1 DO S1 ENDLOOP</td>
<td>L1 IF (.NOT.(Lexp1)) GO TO L2 S1 GO TO L1 L2 CONTINUE</td>
</tr>
</tbody>
</table>
The keyword delimiter pattern, 'LOOP...WHILE...DO', defines the beginning of a repetition loop. The logical expression, Lexp1, will be evaluated before entering the loop which will be executed as long as the logical expression is true. The ENDLOOP forms a boundary between the preceding and following groups of statements. Any other sequential statement, selection structure or loop structure may be nested within this loop.

2. QUIT...LUP:

<table>
<thead>
<tr>
<th>Source Pattern</th>
<th>Target FORTRAN</th>
</tr>
</thead>
<tbody>
<tr>
<td>QUIT L1 LUP</td>
<td>GO TO L1</td>
</tr>
</tbody>
</table>

Where L1 and L2 are the same labels in LOOP...ENDLOOP. The above statement should be nested within the loop structure, LOOP...ENDLOOP. LUP is used instead of LOOP because LOOP is the macro name in LOOP...ENDLOOP and the macro name cannot be defined as the delimiter within another structure. This control statement should be nested within selection structure. This statement could be defined for the abnormal termination of the loop in which it is nested. The flow of control always points to ENDLOOP. So, the LOOP...ENDLOOP could have two different exit points. This, QUIT...LUP, statement is very useful to immediately jump to the outside of the loop.

3. NEXT...LUP:

<table>
<thead>
<tr>
<th>Source Pattern</th>
<th>Target FORTRAN</th>
</tr>
</thead>
<tbody>
<tr>
<td>NEXT L1 LUP</td>
<td>GO TO L1</td>
</tr>
</tbody>
</table>
The label L1 should be the same as the one at the loop entry point, LOOP...ENDDO. The statement above is used to pass the flow of control to the entry point of the loop structure, LOOP...ENDDO. The rest of the statements following the NEXT...LUP, within the LOOP...ENDDO, will be ignored whenever this statement executes.

**Selection control structures:**

These structures allow the programmer to test some logical conditions and perform a block of code depending on the truth value of the condition.

4. **IF...FI:**

<table>
<thead>
<tr>
<th>Source Pattern</th>
<th>Target FORTRAN</th>
</tr>
</thead>
<tbody>
<tr>
<td>IF Lexp1 THEN S1</td>
<td>IF (Lexp1) GO TO L1</td>
</tr>
<tr>
<td>ELSE S2</td>
<td>S2</td>
</tr>
<tr>
<td>FI</td>
<td>GO TO L2</td>
</tr>
<tr>
<td></td>
<td>L1</td>
</tr>
<tr>
<td></td>
<td>L2 CONTINUE</td>
</tr>
</tbody>
</table>

A structured IF statement contains a group of executable statements and one of two blocks of code to be performed. The keyword, IF, followed by a logical expression causes a transfer of control to the next statement which is followed by THEN, if the expression is true. Otherwise, it causes a transfer of control to the statement following ELSE. The structure ends with a FI statement.
5. SIF...FI:

<table>
<thead>
<tr>
<th>Source Pattern</th>
<th>Target FORTRAN</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIF Lexp1 THEN S1</td>
<td>IF (.NOT. (Lexp1)) GO TO L1</td>
</tr>
<tr>
<td>FI</td>
<td>S1</td>
</tr>
<tr>
<td></td>
<td>L1 CONTINUE</td>
</tr>
</tbody>
</table>

If the expression is true, SIF causes a transfer of control to the statement following THEN. Otherwise, the false path consists of transferring control directly from the SIF to the statement following FI. SIF stands for Small IF.

6. CASE...ENDCASE:

<table>
<thead>
<tr>
<th>Source Pattern</th>
<th>Target FORTRAN</th>
</tr>
</thead>
<tbody>
<tr>
<td>CASE L1</td>
<td>IF (.NOT. (Lexp1)) GO TO L2</td>
</tr>
<tr>
<td>GUARD Lexp1 DC</td>
<td>S1</td>
</tr>
<tr>
<td>S1</td>
<td>GO TO L1</td>
</tr>
<tr>
<td>GUARD Lexp2 DO</td>
<td>L2 IF (.NOT. (Lexp2)) GO TO Ln</td>
</tr>
<tr>
<td>S2</td>
<td>S2</td>
</tr>
<tr>
<td>GUARD Lexpn DO</td>
<td>GO TO L1</td>
</tr>
<tr>
<td>Sn ENDCASE</td>
<td>Ln IF (.NOT. (Lexpn)) GO TO L1</td>
</tr>
<tr>
<td></td>
<td>Sn</td>
</tr>
<tr>
<td></td>
<td>L1 CONTINUE</td>
</tr>
</tbody>
</table>

A case structure begins with a CASE statement containing a label. Each block of statements must be preceded by a GUARD statement which is followed the logical expression to
be evaluated. If the comparison is not true, the control transfers to the next GUARD. If no match has occurred after all the logical expressions are examined, control passes to the ENDCASE point. Any other control structures and selection structures may be nested within a CASE structure.

C. USE OF THE ML/I MACRO PREPROCESSOR

The use of macro processors has developed since the early days of programming starting with very simple text replacement facilities and usually in conjunction with assembler language programs. Macros are a rather new features in high-level languages. A number of macro processors have been developed for use with particular high-level languages. In these systems, the macro processor is used as a prepass to the compiler. The general purpose macro processor has been used as a tool to implement high-level language extensions.

Among several different macro processing systems, the macro processor used in work is EPS II, a version of ML/I which is a general purpose macro processor. The ML/I macro processor is essentially designed to be independent of any particular compiler, and thus is useful as a preprocessor for many different languages. ML/I has been used in a variety of different applications, the most common are the extension of existing programming languages and for systematic editing. ML/I is a string handling processor and it is designed to be bootstrapped on to different computer systems.
ML/I could be implemented as a preprocessor to the FORTRAN compiler for the target language, FORTRAN. Figure II.1 shows a general description for use of the ML/I macro preprocessor. The input to the macro preprocessor is a source program written in extension FORTRAN with macro definitions and calls. The output from the macro preprocessor is the FORTRAN program with the macro definitions deleted and the macro calls expanded. This FORTRAN source program is input to the FORTRAN compiler for execution.

Figure II.1. For use of ML/I macro preprocessor

ML/I interprets its input as a stream of atoms and produces as its output another stream of atoms if no macros are invoked. An exception occurs when macro calls are in the input stream, as they are evaluated and the translated tokens are put into the output stream.
There are many useful features in ML/I. This system is intended to allow the user to define his own notation for writing macro calls in order to extend a language (Bro67). Macro calls must commence with the macro name. ML/I organizes a macro by the occurrence of its name and a macro call is taken as the text from the macro name up to its closing delimiter. The user could specify any sequence of characters for each delimiter. The user could define a complex delimiter structure specifies for each delimiter a successor or set of alternative successors. It is possible to have nested macro calls within the arguments of other macro calls. The primary use of ML/I is to allow any existing language to be extended to suit a particular users' requirements. The detailed description of ML/I with its instructions will be presented in the next section.
III. IMPLEMENTATION

A. INTRODUCTORY FEATURES OF ML/I

This section will describe the main features of ML/I with some examples and explain some of the ML/I instructions which will be used for the macro definitions in this paper. For details refer to the EPS II Users Manual (EPSUK).

ML/I terminology:

The user inputs to ML/I some text and definition macros. ML/I performs textual changes; the text generated as a result of the changes is called the value text. The text being evaluated is called the scanned text.

Character set:

ML/I contains the character set for letters, numbers, and some punctuation characters. Space (blank) is treated as a punctuation character. ML/I acts on atoms rather than individual characters. An atom is defined as a single punctuation character or a sequence of letters and/or numbers surrounded by punctuation characters.

Macros and delimiter structure:

System defined macros are called operation macros. The user defines other macros. Each macro must be defined before it is called. When defining a macro the user specifies starting and stopping delimiters and definitions for arguments. Any sequence of atoms could be used for each delimiter. The macro name, that is, the name delimiter, is called the zeroth
delimiter, and the remaining delimiters are called secondary delimiters and closing delimiter. A macro call consists of a macro name and a list of arguments and delimiters. This can be shown as a structure representation:

    IF arg1 THEN arg2 ELSE arg3 FI

The delimiter structure of this macro consists of IF as the name delimiter, and THEN, ELSE and FI as the secondary delimiters, and FI is also the closing delimiter. Macro calls could be expanded to several lines of text.

**Inserts:**

This construction is used to insert a particular argument at the appropriate point. An insert definition consists of an insert name and a closing delimiter, as:

```
%A2.
```

In this example the atom '%' is an insert name with the atom '.' as its closing delimiter. The value text must consist of a flag, which is 'A' in the above example, followed by a macro expression '2'. The various flags used for the macro definition are as follows:

- **A** -- This flag is used within the replacement text of a macro to evaluate and insert the Nth argument of a call of the macro.
- **D** -- The Nth delimiter is inserted.
- **L** -- This is used to place a macro label which is local to the piece of text in which it occurs.
  This is the subject of macro-time GO TO statement and its value is null.
Skips:

Skips are used to inhibit macro replacement within the required context and to take certain strings as literals. A skip definition consists of a skip name and a closing delimiter. There is only one argument, which is treated as a single literal string. Three kinds of option could be defined within skips:

\[ M \] --- Matched option. This is applied where it is desirable for nested skips to be recognized.

\[ D \] --- Delimiter option. The delimiters of the skip are copied over to the value text.

\[ T \] --- Text option. The arguments of the skip are copied over to the value text.

Depending on how the options are set up, the value text may appear in the output text. As an example the macro '⟨ ⟩' is defined with option MT. If the scanned text is given as follows:

\[ ⟨HELLO⟩ \]

The skip name is '⟨' and the closing delimiter is '⟩'. The argument 'HELLO' is copied over to the value text. This special skip name and closing delimiter are called as literal brackets.

Macro variables:

These variables are macro-time integer variables. The user may perform arithmetic on these variables and insert their values into the output text. The two kinds of
macro variables are described below:

Permanent variables --- called P1, P2, ...

These have global scope and can be referred to anywhere. They are reserved at the start of each process and remain in existance throughout.

Temporary variables --- called T1, T2, ...

These have a local scope. Each time a user-defined macro is called, a number of temporary variables is allocated. These are local to the current call. The initial values of three temporary variables are defined by ML/I as for each call:

T1 --- The number of arguments in this call.
T2 --- The total number of all macro calls to this point. (including operation macros)
T3 --- The current depth of nesting of macro calls. (excluding operation macros)

Operation macros:

A number of built-in macros form an integral part of ML/I. The names of all operation macros begin with the letters 'MC' to minimize the confusion with the user-defined macros. A call of an operation macro causes a definition of new constructions or a performance of macro-time arithmetic, etc. The primary operation macros are the followings:
1. MCDEF --- Definition of a local macro, as:
   
   MCDEF arg1 AS arg2

   Where arg1 must be in the form of a structure representation
   which specifies delimiter structure, and arg2 specifies the
   replacement text of the macro being defined.
   Example:
   
   MCDEF HOUSE AS HOME

   By this definition all the occurrences of 'HOUSE' in the
   source text will be replaced as 'HOME'.

2. MCINS --- Definition of a local insert, as:
   
   MCINS arg1

   Where arg1 must be a structure representation.
   Example:
   
   MCINS /*

   '/* defines the insert delimiter and '* ' is the closing
   delimiter. Calling this insert macro adds a new local insert
   definition to the current environment.

3. MCSKIP --- Definition of a local skip, as:
   
   MCSKIP [arg1,] arg2

   Where arg1 is represented as M, D, or T and it is optional,
   and arg2 must be a structure representation.
   Example:
   
   MCSKIP OLD

   This call of macro deletes all occurrences of 'OLD'.
4. **MCSET** --- Macro-time assignment statement, as:

```
MCSET arg1 = arg2
```
Where arg1 must be the name of a macro variable in the current text. Arg2 is a macro expression.

Example:
```
MCSET T1 = /A*
```
The value of T1 is set to the value of inserted argument A1.

5. **MCGO** --- Macro-time GO TO statement, as:

```
MCGO arg1
MCGO arg1 IF arg2 = arg3
```
Where arg1 must consist of the 'L' followed by a macro expression. MCGO can be used to form macro-time loops. A true value results if arg2 and arg3 are identical strings of characters.

Example:
```
MCGO L1
```
This statement is the same as GO TO L1 in macro-time.

```
MCGO L4 IF /D1x$ = NEW
```
If the value of inserted delimiter 1 is identical to 'NEW', then the control goes to L4.

6. **MCLISTS** --- This operation macro enables the programmer to get a complete listing of the source program.

7. **MCLISTT** --- The programmer can get a complete listing of the target program.

8. **MCSTOP** --- Terminate ML/I processing. The MCSTOP macro causes the control to return to the calling program.
Keywords:

Within a structure representation the certain keywords are used to stand for layout characters, for example spaces, newlines, tabs. The layout keyword could be specified as a delimiter within structure representations. These are SPACE, TAB, SPACES, and NL. There are other keywords such as WITH, WITHS, OPT, OR, ALL and any atom commencing with the letter 'N' followed by a digit, none of which can be used as delimiters in structure representations.

Complex structure representation:

In order to define a complicated delimiter structure, there is a mechanism for specifying successors, namely option lists and nodes. Option lists are used to specify that a delimiter has several optional alternatives as successor. The essential form is:

OPT branch1 OR branch2 OR ... OR branchN ALL

The branch is one of the alternative successors following the delimiter which is defined right before this option list. Nodes are used for defining the successor of a delimiter to be a delimiter or option list elsewhere in the structure representation. The following example illustrates this:

Example:

IF N1 OPT THEN N1 OR ELSE N1 OR FI ALL

In this structure representation, the first N1 before the option list is the nodeplace. A node is placed by writing its
name before any delimiter name or option list. The second and the third M1 are the nodego. The nodego is placed after the end of a branch of an option list or after an option list.

The components of structure representation will be described as follows:

Nodeplace --- The placing of a node.
Nodego --- The action of going to a node.
Delspec --- The specification of a delimiter or an option list.

The notations which are used in the ER3 II User Manual indicate parts of syntactic forms:

* --- Constituents may be repeated.
? --- Constituents may be omitted.
**? --- Constituents may be repeated or omitted.

Structure representation of Delspec:

```
[noodeplace ?] [delimiter name]
      [OPT branch [OR [nodeplace ?] branch *?]] ALL
```

branch:

delimiter name [delspec *?] [nodego ?]

This structure representation will be used for the macro definitions following this part.
B. DEFINITION OF MACROS

The following macro definitions are made to facilitate structured FORTRAN programming. Each of the macro definitions will apply an operation macro at the beginning. The replacement text also includes a number of operation macros. Ten macro definitions will be explained thoroughly with some examples. The listing of macros is included in the Appendix B.

1. MCINS %.

This macro is used to add a new local insert definition to the current environment. '%' defines the insert delimiter and '.' is the closing delimiter. The insert macro must be in replacement text. Whenever this macro appears within replacement text, a new value is generated and inserted.

Example:

```
%A2.
```

The second argument (A2) within the structure representation will be inserted to the value text.

```
%D1.
```

The second delimiter (D1) which is the first secondary delimiter in the structure representation will be inserted.

2. MCSKIP MT, < >

The MCSKIP macro is used to ignore some of the macro names or to skip the scanning of some character strings. The matched and text options are set. The skip name '⟨' with closing delimiter '⟩' is defined as a pair of literal brackets.
If this matched option is on, the nested brackets can be
scanned. The text option will literally copy a piece of
text over to the value text and the delimiters, '"' and '"',
will be deleted.

Example:

\[
\text{MCDEF} \langle \text{NEXT LUP} \rangle \text{ AS } \langle \text{MCSET T1 = 100} \\
\langle \text{GO TO} \rangle \neq T1.\rangle
\]

The structure representation of the macro definition
following MCDEF will be enclosed within the brackets from
now on. This saves confusion with the macro name whenever
it needs to be redefined or modified. This is discussed later
in section IV.

In the example, the second '⟨' matches with the last '⟩'
and the nested '⟨' matches with '⟩' following GO TO. The
character string GO TO will be copied over to the value text
as a literal without '⟨⟩'. The value text will be 'GO TO 100'.
The replacement text can involve one or more newlines by
using these literal brackets. To obtain some literal string
in the output text, it is advisable to use the literal brackets
around the string in the definition.

3. MCSET P1 = macro expression

The value of the permanent variable will be set to
value of expression. The value of P1 remains as a global
throughout all of the scanned text.

Example:

\[
\text{MCDEF} \langle \text{IF THEN P1} \rangle \text{ AS } \langle \text{MCSET T1 = P1} \\
\text{MCSET P1 = P1 + 1} \rangle
\]
The value of the temporary variable T1 will be set to the value of P1 which is already defined. The value of P1 will be incremented by 1 by the second macro-time assignment statement.

4. MCDEF \(<;>\) AS <

\(<;> > \)

Where < denotes a blank. This macro is defined in order to write more than one statement per line of source text. The macro name, semicolon, is replaced by a newline with five blanks. Whenever the semicolon appears with a statement in the source text, the next statement following the semicolon will be printed on the next line of the output text.

Example:

```
col. 7
→
A = B; < ; B = C; < ; C = D; < ; D = E
(source)
A = B
B = C
C = D
<D = E
(output)
```

The first statement in the source text starts from column 7 and the rest of the statements follow with a semicolon and a blank except the last one which has two blanks after the semicolon. The output text starts from column 7 by this macro definition, except that the last statement starts in column 8. Note: Need a blank after the semicolon.
5. QUIT...LUP

Structure:

QUIT -- LUP

Macro definition:

MCDEF \langle QUIT LUP \rangle
AS \langle MCSET T1 = \%A1. + 1 \rangle
\langle GO TO \rangle \%T1. \rangle

The delimiter structure of this macro starts from the macro name QUIT with the closing delimiter LUP. Within the replacement text the value of the temporary variable T1 is assigned by the macro-time assignment statement MCSET. This value of T1 is used for the statement label which points to the end of the enclosed loop.

Example:

(source)                        (output)
QUIT 500 LUP                    GO TO 501

6. NEXT...LUP

Structure:

NEXT -- LUP

Macro definition:

MCDEF \langle NEXT LUP \rangle
AS \langle (GO TO) \%A1. \rangle

The macro name NEXT with the closing delimiter LUP will be replaced by a GO TO statement. The inserted value of the argument one (A1) points to the entry of the loop in which this NEXT macro is nested.
Example:

(source)                      (output)
NEXT 500 LUP                   GO TO 500

7. LOOP...ENDLOOP

Structure:

SPACE WITHS LOOP -- WHILE -- DO WITHS NL --

--[ ENDLOOP -------- ]

-- NL WITHS ENDLOOP --

Macro definition:

MCDEF 〈SPACE WITHS LOOP WHILE DO WITHS NL
OPT ENDLOOP OR NL WITHS ENDLOOP ALL〉
AS 〈MCSET T1 = %A1. + 1
%A1. 〈IF〉 (.NOT.(%A2.)) 〈GO TO〉 %T1.
%A3.
〈GO TO〉 %A1.
%T1. CONTINUE〉

The keywords SPACE WITHS within the macro name SPACE
WITHS LOOP specifies an arbitrary number of spaces in front
of the atom LOOP. This macro call makes the space adjustment
possible at the beginning of the replacement text. It means
that the inserted argument %A1. in the second statement of
the replacement text can appear in the same column of the
source text. The third delimiter DO WITHS NL is always
followed by a newline of the source text. The closing delimiter
is defined within the option list. ENDLOOP follows the last
statement of argument 3 on the same line. NL WITHS ENDLOOP begins from the newline which is followed by the last line of argument 3. Argument 1 is specified for the statement label and argument 2 is specified for the logical expression.

Examples:

<table>
<thead>
<tr>
<th>(source)</th>
<th>(output)</th>
</tr>
</thead>
<tbody>
<tr>
<td>a. LOOP 500 WHILE A.EQ.B DO</td>
<td>500 IF (.NOT.(A.EQ.B)) GO TO 501</td>
</tr>
<tr>
<td>B = C</td>
<td>B = C</td>
</tr>
<tr>
<td>D = B</td>
<td>D = B</td>
</tr>
<tr>
<td>ENDLOOP</td>
<td>GO TO 500</td>
</tr>
<tr>
<td></td>
<td>501 CONTINUE</td>
</tr>
<tr>
<td>b. LOOP 500 WHILE A.EQ.B DO</td>
<td>500 IF (.NOT.(A.EQ.B)) GO TO 501</td>
</tr>
<tr>
<td>B = C</td>
<td>B = C</td>
</tr>
<tr>
<td>D = B ENDLOOP</td>
<td>D = B</td>
</tr>
<tr>
<td></td>
<td>GO TO 500</td>
</tr>
<tr>
<td></td>
<td>501 CONTINUE</td>
</tr>
</tbody>
</table>

8. IF...FI

Structure:

```
IF [-- THEN ------- ] [-- ELSE ------- ]--

[ -- FI ----------- ]

[-- NL WITHS FI ]
```

Macro definition:

```
MCDEF ⟨IF OPT THEN OR NL WITHS THEN ALL
OPT ELSE OR NL WITHS ELSE ALL
OPT FI OR NL WITHS FI ALL⟩
```
AS \textit{MCSET P1 = P1 + 1}  
MCSET T1 = P1  
MCSET P1 = P1 + 1  
MCSET T2 = P1  
\textit{\%NDO. (A1.) (GO TO) \%T1.}  
\textit{\%A3.}  
\textit{(GO TO) \%T2.}  
\textit{\%T1. \%A2.}  
\textit{\%T2. CONTINUE}  

All the secondary delimiters in this IF macro have an alternative successor which is defined within the option list. This IF macro can be written in one line of the source text or written over several lines each beginning with a secondary delimiter. Within the replacement text, the permanent variable P1 is updated to keep track of the value of the statement labels which will be unique throughout the source text. The values of T1 and T2 are temporarily assigned the value of P1 and used for the statement label within the current scanned text. 'W' in \textit{\%NDO.} stands for 'Written'. DO is the delimiter zero which is the macro name. If W is prefixed to D, the inserted text which is the macro name in this case is not evaluated but is inserted literally. (This insert is identical to that of the character string IF, which could have been inserted directly with using the \textit{\%WDO}..)
Examples:

(source)                  (output)

a. IF A.EQ.B THEN C = B ELSE C = A FI  IF (A.EQ.B) GO TO 101
   C = A
   GO TO 102
   101 C = B
   102 CONTINUE

b. IF A.EQ.B
   THEN C = B
   ELSE C = A
   FI
   IF (A.EQ.B) GO TO 103
   C = A
   GO TO 104
   103 C = B
   104 CONTINUE

9. SIF...FI

Structure:

\[
\begin{array}{cc}
\text{SIF} & -[- \text{THEN} \quad \ldots \quad \ldots \quad \text{FI}] \\
\text{NL WITHS THEN} & -[- \text{NL WITHS FI}] \\
\end{array}
\]

Macro definition:

MCDEF <SIF OPT THEN OR NL WITHS THEN ALL
OPT FI OR NL WITHS FI ALL>
AS <MCSET P1 = P1 + 1
MCSET T1 = P1
<IF> (.NOT.(%A1.)) <GO TO> %T1.
%A2.
%T1. CONTINUE>
The macro name SIF stands for Small IF. The reason this SIF is defined separately from the IF macro will be explained with examples in the next section. The SIF macro definition is similar to the IF macro except the SIF macro doesn't have the alternative statement ELSE.

Example:

(source)                                     (output)

a. SIF A.EQ.B THEN C = B FI                  IF (.NOT.(A.EQ.B)) GO TO 105
                                            C = B
                                            105 CONTINUE

b. SIF A.EQ.B
    THEN C = B
    D = A
    FI

    IF (.NOT.(A.EQ.B)) GO TO 106
    C = B
    D = A
    106 CONTINUE

10. CASE...ENDCASE

Structure:

```
CASE -- [NL WITHS GUARD -- [DO -- [DO WITHS NL -- N1]]]

ENDCASE
```

Macro definition:

```
MCDEF <CASE N1 OPT NL WITHS GUARD
OPT DO OR DO WITHS NL ALL N1 OR ENDCASE ALL>
AS <MCSET T1 = 1
MCSET T2 = 2
```
MCSET T3 = %A1.
<IF> (.NOT.(%AT2.)) <GO TO> %T3+1.
%AT2+1.
<GO TO> %A1.
%L1. MCSET T1 = T1 + 2
MCSET T2 = T2 + 2
MCSET T3 = T3 + 1
MCGO L2 IF %DT1+2. = <ENDCASE>
%T3. <IF> (.NOT.(%AT2.)) <GO TO> %T3+1.
%AT2+1.
<GO TO> %A1.
MCGO L1
%L2. %T3. <IF> (.NOT.(%AT2.)) <GO TO> %A1.
%AT2+1.
%A1. CONTINUE>

The node N1 before OPT is the nodeplace. All the delimiters within the inner option list return a pointer to this nodeplace to search the next delimiter, NL WITHS GUARD, or the closing delimiter, ENDCASE. The node N1 after ALL is the nodego. It points to the nodeplace N1 to find the successor of the delimiter defined in the nested option list. In the replacement text, the initial values of the temporary variables are given at the beginning. T1 is used to check for the delimiter, NL WITHS GUARD, T2 checks for the argument between NL WITHS GUARD and the nested option list, DO, and T3 looks
for the statement label. After testing the first NL WITHS GUARD delimiter, END scans iteratively the proper delimiter by the macro-time GO TO statement MCGO L1. This loop points to MCGO L2 if the search of the delimiter ends with ENDCASE. In the above macro definition, if the value of T2 is initially set to 2, the inserted value of %AT2+1. is same as %A3. and the value of argument 3 is inserted in the value text. The closing delimiter ENDCASE is always written in the next line of the last source statement. Following examples demonstrate the above explanation.

Examples:

<table>
<thead>
<tr>
<th>(source)</th>
<th>(output)</th>
</tr>
</thead>
<tbody>
<tr>
<td>a. CASE 500</td>
<td>IF (.NOT.(A.EQ.B)) GO TO 501</td>
</tr>
<tr>
<td>GUARD A.EQ.B DO C = A</td>
<td>C = A</td>
</tr>
<tr>
<td>GUARD A.LT.B DO C = B</td>
<td>GO TO 500</td>
</tr>
<tr>
<td>GUARD A.LT.C DO B = A ENDCASE 501</td>
<td>IF (.NOT.(A.LT.B)) GO TO 502</td>
</tr>
<tr>
<td></td>
<td>C = B</td>
</tr>
<tr>
<td></td>
<td>GO TO 500</td>
</tr>
<tr>
<td>502 IF (.NOT.(A.LT.C)) GO TO 500</td>
<td>B = A</td>
</tr>
<tr>
<td>500 CONTINUE</td>
<td></td>
</tr>
<tr>
<td>b. CASE 500</td>
<td>IF (.NOT.(A.EQ.B)) GO TO 501</td>
</tr>
<tr>
<td>GUARD A.EQ.B DO C = A</td>
<td>C = A</td>
</tr>
<tr>
<td></td>
<td>GO TO 500</td>
</tr>
</tbody>
</table>
GUARD A.LT.B DO 501 IF (.NOT.(A.LT.B)) GO TO 502
C = B C = B
GUARD A.LT.C DO
B = A ENDCASE 502 IF (.NOT.(A.LT.C)) GO TO 500
B = A
500 CONTINUE

C. HOW TO EXECUTE THE PREPROCESSOR

The following part of this section will describe five steps to be used for running a structured FORTRAN program. Underlined commands are typed by the user.

Step 1. After logging on, the user should create or edit the structured FORTRAN. The sample structured FORTRAN program listing, TEST SFORT, is included in the Appendix A.

Step 2. All the files needed for the executing the preprocessor are stored in the file manager space for user VM2G8 (Dr. Hankley). The user needs to bring these files from the VM2G8 file space to the user's A-disk by following commands.

FMR SOOMACRO_FP3_(U VM2G8
FMR EPS MODULE_(U VM2G8
FMR SOO EXEC_(U VM2G8

The user now has 4 different files on A-disk including the file for the structured FORTRAN program. The file SOOMACRO
EPS contains all the macro definitions which were previously described. The listing of this file is included in the Appendix B. The program written by the user will be run through EPS MODULE for preprocessing. SO0 EXEC is the EXEC procedure to combine all the commands used to preprocess the structured FORTRAN program. The listing of SO0 EXEC is included in the Appendix C.

Step 3. The next command the user has to type is

```
SO0 fn_ft1_ft2
```

For example: 'SO0 TEST SFORT WATFIV'

By this command, the EXEC procedure, SO0 EXEC, will be executed. fn specifies the filename of the user's structured FORTRAN program, ft1 is the filetype of that input program, and ft2 is the filetype of the output file from EPS. fn, ft1 and ft2 correspond to &1, &2 and &3 respectively within the SO0 EXEC procedure. The following is a brief explanation of the SO0 EXEC procedure.

The new file, EPSIN EPS, is created by the COPYFILE command that puts the SOOMACRO EPS at the tope of the user's structured FORTRAN program. The Figure III.1 illustrates the file EPSIN EPS. The CMS editor adds an operation macro, MCSTOP, at the end of the EPSIN EPS. This macro will be needed to terminate EPS II processing after EPSIN EPS goes through the EPS II preprocessor and the control returns to the calling program, SO0 EXEC. The listing of EPSIN EPS is in the Appendix D.
Figure III.1 The description of the file EPSIN EPS

EPS II provides six logical I/O files as illustrated in Figure III.2. One of the data definition names, M000, is as yet undefined. M001 is used for the input text EPSIN EPS. M010 signifies the error messages which will be displayed at the user's terminal whenever the errors are encountered.

M011 is the input listing with line numbers which will be placed on the user's A-disk as file '&1 FORTLST'. M012 is the output listing with line numbers which is defined as '&1 FORTLIST'. F027 designates the output to be used as the input to the FORTRAN or WATFIV compiler. The last three output files will be placed on the user's A-disk after the execution of S00 EXEC.
Figure III.2 Files processed by SO0 EXEC using EPS II
After these files are defined, EPS executes the input macro program. Timing lines which indicate virtual CPU, real CPU and clock time will be displayed at the user's terminal both before EPS execution and after execution. Also, the error message listing will be displayed. The user may then calculate the CPU time spent for preprocessing his FORTRAN program.

The EXEC procedure will erase the input file EPSIN EPS and close those three output files which were left open after the EPS processing was terminated by MCSTOP. The CMS editor will delete the top part of the input listing containing the macro definitions and it will delete the operation macro, MCSTOP, from the bottom of the input listing. Also, the editor changes C to C$JO3 at the first line of the output file &1 &3 and C to C$ENTRY at the bottom line. The reason this editing has to be done in 300 EXEC will be explained in the user constraints part. As the result of the execution of 300 EXEC, the user gets the three output files on the A-disk. File &1 &3 will be ready to compile using the FORTRAN compiler or using WATFIV.

Step 4. The user will be able to run the output program from EPS through the compiler chosen. The following example used the WATFIV compiler:

WATFIV_TEST

Where TEST was the name of the sample structured FORTRAN program. The FORTRAN G or H compilers could be accessed using the OSJOB command.
Step 5. The user will be able to get the compilation listing after the program is successfully compiled by the WATFIV compiler.

**TY_TEST LISTING**

Where TEST was the filename specified by the user at the beginning of the run.

**D. USER CONSTRAINTS**

Presented here are several points the user needs to be concerned with while writing structured FORTRAN. The constraints are mainly caused by EPS II.

1. The user must write $JOB$ in the first column of the top line of his program and $ENTRY$ on the bottom line. The user may then run his program through WATFIV, FORTRAN G or FORTRAN H compilers. The WATFIV compiler interpretes these two statements as the job control cards, $JOB$ and $ENTRY$. FORTRAN G or H compilers will take the statements as comment statements ignoring the $JOB$ and $ENTRY$. The reason for this constraints is as follows.

In EPS, the punctuation character '$' represents a new line. Whenever this '$' is encountered, the rest of the statement on that line including '$' is deleted and not copied to the output file. This system defined '$' cannot be redefined to some other character by MCA LTER which is one of the operation macros. This macro is used for the alteration of the secondary
delimiters of operation macros or of the keywords used for structure representation. At the time EPS encounters '$' in C$JOB and C$ENTRY, EPS sets $JOB and $ENTRY to null and scans next line of the input text and C is left as the output text from EPS. After EPS preprocessing is done, the CMS editor is invoked to change C to C$JOB and C$ENTRY. This procedure will enable the output file from EPS to run through one of the FORTRAN compilers.

2. One of the constraints is about the statement labels. The user may not use the statement label close to 100.
In the macro definition, the global variable P1 is set to 100. P1 is only incremented by the macro calls, IF and SIF, and the user could see the result after the preprocessing by EPS. In order to prevent the program from the confusion, it is better to start the statement label from 200 or over. The statement label used in the sample structured FORTRAN in the Appendix A starts from 500. So, the user needs to keep this restriction in mind when he does the labelling of other loop structures or case statements within the structured FORTRAN program. Otherwise, compile error will be appeared.

3. As part of the restriction on statement labels, the user must use only three digit labels starting from column 3.
In the following macro definition, the inserted text, \$T2., starts from column 3 and the value of T2 is three digit label.

\begin{verbatim}
MCDEF <IF THEN P1>
AS <MCSET T2 = 333
\$T2,\$CONTINUE>
\end{verbatim}
The output text from EPS always starts from column 3 by this definition.

If the label is two digit number, the last line of replacement text must be written in as the following ways.

\texttt{EPSAT2.Y\textunderscore CONTINUE} or \texttt{EPSAT2.Y\textunderscore DISCONTINUE} or \texttt{EPSAT2.Y\textunderscore DISCONTINUE}

The statement \texttt{CONTINUE} enable to start from column 7 by these definitions. It is preferred to use three digit labels because the number is ranged from 100 through 999. The following examples illustrates this.

Examples:

\texttt{EPSAT2.Y\textunderscore CONTINUE} \hspace{1cm} \texttt{(replacement text)}

If the value of the inserted text T2 is 333, the output text will be

\texttt{EPS333Y\textunderscore CONTINUE}

and \texttt{CONTINUE} starts from column 7 which is legal.

If the value of the inserted text is 33, the output text will be

\texttt{EPS33Y\textunderscore CONTINUE}

and \texttt{CONTINUE} starts from column 6 which is incorrect for FORTRAN convention. If the value of the inserted text T2 is 3333, the output text will be

\texttt{EPS3333Y\textunderscore CONTINUE}

and the last digit of the statement label is typed on the sixth column which is illegal.

4. Indenting was not implemented. It is recommended that the user start all statements at column 7, except the statement labels and the continuation mark and comments. The examples are in the following pages.
The sequential statement within the macro call is not indented as it is specified in the macro definition if another macro call is nested. Another reason for the constraint is this. When the first statement in the replacement text is copied into the output text, it starts from the same column the source text is indented. For example, SIF macro is illustrated:

```
MCDEF \langle SIF ..... \rangle
AS\langle . . \rangle
\langle IF \rangle (.NOT.(\%A1.)) \langle GO TO \rangle \%T1.
\langle . \rangle
```

If the source text is

```
col. 7
\langle SIF A.EQ.B THEN B = A FI
```

The output text will be

```
col. 7
\langle IF (.NOT.(A.EQ.B)) GO TO 102
\langle . \rangle
```

If the source text is

```
col. 10
\langle SIF A.EQ.B THEN B = A FI
```

The output text will be

```
col. 10
\langle IF (.NOT.(A.EQ.B)) GO TO 102
\langle . \rangle
```
However, the above example is not always correct as it is. If this SIF macro is nested within another macro, the indented column is changed by the outer macro definition.

If the following indented structured FORTRAN program which contains the continuation card is given by the user,

```
col. 7
   ↓
LOOP 515 WHILE I.LE.50 DO
   SIF DROP.EQ.0.
   THEN QUIT 515 LUP FI
   R = R + R * SCOEF
   * /2.
   SIF C.GT.11.
   THEN NEXT 515 LUP FI
   C = C - C * CCOEF
   I = I + 1
   ENDLOOP
```

The output text from EPS will be

```
col. 7
   ↓
515 IF (.NOT.(I.LE.50)) GO TO 516
   IF (.NOT.(DROP.EQ.0.)) GO TO 103
   GO TO 516
103 CONTINUE
   R = R + R * SCOEF
   * /2.
   IF (.NOT.(C.GT.11.)) GO TO 104
   GO TO 515
```
104 CONTINUE
C = C - C * 3302F
I = I + 1
GO TO 515
516 CONTINUE

The second IF starts from column 7 because %A3. in LOOP macro
starts from column 7. But the third IF, which is also %A3.,
starts from the same column the second SIF is indented.
The sequential statements followed by another macro calls
within LOOP macro start from the indented column as specified
in the source text. So, the nested macros and the sequential
statements did not follow the definition of LOOP macro.

Another example demonstrates the indented structured
FORTRAN.

col. 7
↓
DO 520 I=1,50
SUM=SUM+C
CASE 525
    GUARD I.LT.5 DO
    SAVE(1)=SUM*1.
    GUARD I.LE.15 DO
    SAVE(2)=SUM*2.
    GUARD I.LE.25 DO
    SAVE(3)=SUM/2.
525 ENDCASE
520 CONTINUE
The output text from EPS will be

```
col. 7
↓
DO 520 I=1,50
   SUM=SUM+C
   IF (.NOT.(I.LT.5)) GO TO 526
   SAVE(1)=SUM*1.
   GO TO 525
526 IF (.NOT.(I.LE.15)) GO TO 527
   SAVE(2)=SUM*2.
   GO TO 525
527 IF (.NOT.(I.LE.25)) GO TO 525
   SAVE(3)=SUM/2.
525 CONTINUE
520 CONTINUE
```

The beginning column of case statements starts from the same column which is typed by the user even the rest of the statement starts from column 7. So, the group of statements are not indented as the user specified. This looks uglier than a program when all the statements starts from column 7 following the usual FORTRAN convention. This indenting problem can not be corrected.
IV. EVALUATION

A. PROBLEMS OF MACRO DEFINITION

There were many difficulties in defining macros for this paper because the ML/I macro processor has some restrictions on the space and line adjustment and on specifying the delimiter structures. The design of structured FORTRAN extensions and the delimiter structure of the macro definition have been effected by these difficulties. The structure of defined macros could have been better if there had not been such restrictions. In this section, some desirable but illegal delimiter structures for the macro definitions will be described and explained with some examples. Also, the reasons these delimiter structures can not be used will be presented.

1. QUIT...NL and NEXT...NL

Desired structure:

   QUIT -- NL

Macro definition:

   MCDEF ⟨QUIT NL⟩
   AS ⟨MCSET T1 = %A1. + 1
   ⟨GO TO⟩ %T1.

 Desired structure:

   NEXT -- NL
Macro definition:

```
MDEFINE \$NEXT NL
AS \$GO TO S$A1.

>
```

These two macro definitions are rather straightforward in writing the source text compared with QUIT...LUP and NEXT...LUP. The layout keyword NL in both macro definitions is replaced by the carriage return within the replacement text. The closing delimiter '>' of the skip macro is placed on the new line of the replacement text. In the source text, the carriage return is pressed to specify NL instead of writing NL after the argument 1. The statements in the replacement text, except the NL, are the same as the ones in QUIT...LUP or NEXT...LUP structures. The above macro definitions work properly by themselves in the following examples:

<table>
<thead>
<tr>
<th>(source)</th>
<th>(output)</th>
</tr>
</thead>
<tbody>
<tr>
<td>QUIT 500</td>
<td>GO TO 501</td>
</tr>
<tr>
<td>NEXT 500</td>
<td>GO TO 500</td>
</tr>
</tbody>
</table>

However, if these macro calls are nested within a loop such as LOOP...ENDLOOP in the source text, the layout keyword 'NL' of this nested macro call is substituted as a new line on the output text (Figure IV. 1.). If this macro call is followed by a regular statement which is not a macro call, there is no immediate space after the macro call statement. Even though these two macros could be defined as shown in this section (i.e. they are not illegal), the irregular spacing of the output text can not be accepted for FORTRAN programs.
Figure IV.1. NEXT/QUIT examples

b -- correct output  a, c -- output is incorrect.
2. LOOP...ENDLOOP

Desired structure:

LOOP -- WHILE -- DO WITHS NL -- NL -- ENDLOOP

Macro definition:

MCDEF ⟨LOOP WHILE DO WITHS NL NL ENDLOOP⟩
AS ⟨MCSET T1 = %A1. + 1
%A1. ⟨IF⟩ (.NOT.(%A2.)) ⟨GO TO⟩ %T1.
%A3.
⟨GO TO⟩ %A1.
%T1. CONTINUE⟩

The delimiter structure of this macro definition is similar to the one already defined in the last section with the exception of the macro name and the secondary delimiter, NL.

This structure is simpler than the previously defined one. However, the difficulty lies on the column adjustment in the output text. The following example demonstrates this spacing problem:

(source)                          (output)

LOOP 500 WHILE A.EQ.B DO        500 IF (.NOT.(A.EQ.B)) GO TO 501
B = D                             B = D
C = B                             C = B
500 ENDLOOP                      GO TO 500

501 CONTINUE
The first statement label in the output text did not appear at the second column because the macro did not start at the second column. The first statement label always starts from the same column the source text starts. The spacing has to be done properly to match the FORTRAN convention for the statement label.

Another problem is with the NL delimiter. The following example illustrates the difficulty using the delimiter, NL.

SPACE WITHS LOOP is used as the macro name.

\[
\begin{array}{ll}
\text{(source)} & \text{(output)} \\
\text{LOOP 515 WHILE I.LE.50 DO 515 IF (.NOT.(I.LE.50)) GO TO 516} & 515 \text{ IF (.NOT.(I.LE.50)) GO TO 516} \\
\text{SIF DROP.EQ.0.} & \text{IF (.NOT.(DROP.EQ.0.)) GO TO 103} \\
\text{THEN QUIT 515 LUP FI} & \text{GO TO 516} \\
\text{R = R + R* SCOEF} & \text{103 CONTINUE} \\
\text{SIF C.GT.11.} & \text{GO TO 515} \\
\text{THEN NEXT 515 LUP FI} & \text{516 CONTINUE} \\
\text{C = C - C * GCOEF} & \\
\text{I = I + 1} & \\
\text{515 END LOOP} & \\
\end{array}
\]

Those statements which follow any macro call within this loop structure are deleted. The reason for this is not clear.

3. IF...FI and SIF...FI

Instead of designing two separate delimiter structures, it is better to combine these two structures into one macro name IF (although this seems to be impossible). Two cases of preferred delimiter structures will be described.
Desired structure:

```
IF              [ THEN ]
              [ NL WITHS THEN ]

ELSE --- NL WITHS ELSE --- N1

N1 --- [ FI ]
       [ NL WITHS FI ]
```

Macro definition:

```
MCDEF ⟨ IF OPT THEN OR NL WITHS THEN ALL 
OPT OPT ELSE OR NL WITHS ELSE ALL N1
OR N1 OPT FI OR NL WITHS FI ALL ALL ⟩
A3 ⟨ MCSET P1 = P1 + 1 
MCSET T1 = P1 
MCSET P1 = P1 + 1 
MCSET T2 = P1 
MCGO L1 IF %D2. = ⟨ ELSE ⟩ 
MCGO L1 IF %D2. = ⟨ NL WITHS ELSE ⟩ 
%ndo. (.NOT.(%A1.)) ⟨GO TO⟩ %T2. 
%A2. 
MCGO L2 
%L1.%ndo. (%A1.)) ⟨GO TO⟩ %T1. 
%A3. 
⟨GO TO⟩ %T2. 
%T1. %A2. 
%L2. %T2. CONTINUE⟩
```
This structure has four option lists. Among them two option lists are nested within a big option list and are chained together by the node N1. Basically, this structure can be written in four different ways such as IF...THEN...ELSE...FI, IF...NL WITHS THEN...NL WITHS ELSE...NL WITHS FI, and IF...NL WITHS THEN...NL WITHS FI. Also it is possible to write a similar type of delimiter in an option list, i.e., substituting NL WITHS THEN to THEN. The node N1 pointed by the arrow is the nodego. The other N1 followed by an option list, FI or NL WITHS FI, is the nodeplace. The replacement text of this macro definition is the combination of two separate replacement texts which were defined in the previous section. The only differences are that the macro-time GO TO statement, MCGO, and the macro-time labels are used to match and find the proper character string for the delimiter.

Even though this delimiter structure has a better form, the macro definition by this structure is illegal. The keyword OPT may not be followed by another OPT. The nodeplace N1 after OR has to be followed by a delimiter name not by the keyword OPT. Another trouble point is that the layout keywords (NL) and the keywords (WITHS) can not be used within the replacement text. So, there is no way to test the alternative delimiter in the option list if one of them is constructed with keywords.

Another desirable delimiter structure is as follows:
Macro definition:

\[ \text{MCDIF IF OPT THEN OR NL WTHS THEN ALL OPT ELSE N1 OR NL WTHS ELSE N1 OR N1 FI ALL} \]

AS

The replacement text is the same as the one previously defined in this section.

This structure can be written in several ways described in the first example above except that the closing delimiter has no alternative but FI. This macro has two option lists one of which includes the nodego and the nodeplace. The delimiter following the nodeplace N1 has to be a closing delimiter within that option list. The delimiter structure of this macro definition is correct but the replacement text, which is identical to the one previously mentioned, contains an illegal conditional statement. Because the test of \[ \text{NL WTHS ELSE} \] fails all the time (illegal NL in the replacement text), the control of the replacement text will never go to the inserted value of %L1. as long as the delimiter is specified as \[ \text{NL WTHS ELSE} \]. As a result, the output statements will be the ones following the statement of failed test. The following example will demonstrate the above explanation.
(source) 

a. IF A.EQ.3 THEN C = A ELSE C = 3 FI

IF (A.EQ.3) GO TO 101
C = B
GO TO 102
101 C = A
102 CONTINUE

b. IF A.EQ.3 THEN C = A ELSE C = B FI

IF (.NOT.(A.EQ.3)) GO TO 104
C = A
104 CONTINUE

c. IF A.EQ.3 THEN C = A FI

IF (.NOT.(A.EQ.3)) GO TO 106
C = A
106 CONTINUE

d. IF A.EQ.3 THEN C = A FI

IF (.NOT.(A.EQ.3)) GO TO 108
C = A
108 CONTINUE

(output) 

Figure IV.2. IF...FI examples

a --- correct output  b --- output is incorrect.
c --- correct output  d --- correct output
4. CASE...ENDCASE

Desired structure:

```
CASE -- \textcircled{N1} \rightarrow [NL WITHS GUARD \rightarrow \text{DO} \rightarrow \text{DO WITHS NL} \rightarrow N1]
```

--- ENDCASE

Macro definition:

```
MCDDEF <CASE N1 OPT NL WITHS GUARD
OPT DO OR DO WITHS NL ALL N1 OR NL ALL ENDCASE>
AS <MCSET T1 = 1
MCSET T2 = 2
MCSET T3 = \%A1.
<IF> (.NOT. (\%AT2.)) <GO TO> \%T3+1.
\%AT2+1.
<GO TO> \%A1.
\%L1.MCSET T1 = T1 + 2
MCSET T2 = T2 + 2
MCSET T3 = T3 + 1
MCGO L2 IF \%DT1+3. = <ENDCASE>
\%T3. <IF> (.NOT. (\%AT2.)) <GO TO> \%T3+1.
\%AT2+1.
<GO TO> \%A1.
MCGO L1
\%L2. \%T3. <IF> (.NOT. (\%AT2.)) <GO TO> \%A1.
\%AT2+1.
\%A1. CONTINUE >
```
This CASE macro has NL as the second delimiter. There is the same problem as with the LOOP macro. The NL causes difficulties if other macros are nested within this CASE structure. The following examples are for the CASE macro.

(source)  

a. CASE 500
    GUARD A.EQ.3 DO
    A = B
    GUARD A.LT.3 DO
    QUIT 500 LUP
    C = A
    GUARD A.GT.3 DO
    B = C
500 ENDCASE

b. CASE 500
    GUARD A.EQ.3 DO
    A = B
    IF P.LT.C THEN C = B
    ELSE C = A FI
    GUARD A.LT.3 DO
    C = A
    GUARD A.GT.3 DO
    B = C
500 ENDCASE

(output)  

IF (.NOT.(A.EQ.3)) GO TO 501
A = B
GO TO 500
501 IF (.NOT.(A.LT.3)) GO TO 500
GO TO 501
500 CONTINUE

ERROR(S)
D6 IS ILLEGAL MACRO ELEMENT
::
::
::
(error messages deleted)

Figure IV.3. Examples for CASE...ENDCASE
The example (a) in Figure IV.3 displays the identical error which was illustrated in the example for LOOP...ENDLOOP macro. Those statements which follow a nested macro call within this CASE macro are deleted. In the second example in Figure IV.3, the error messages are given. It is a very long message and is difficult to figure it out.

5. Brackets around macro definitions:

If a delimiter structure which was defined without the literal brackets is redefined (for example, to change the replacement text), then the original definition is not erased from the run time environment. That causes some errors. The following examples illustrate this:

a. MCDEF NEXT LUP AS ⟨⟨GO TO⟩ %A1.⟩
   (source)                                       (output)
   NEXT 500 LUP                                   GO TO 500

b. MCDEF NEXT LUP AS ⟨⟨JUMP TO⟩ %A1.⟩
   (source)                                       (output)
   NEXT 500 LUP                                   GO TO 500

c. MCDEF NEXT LUP AS ⟨GO TO⟩ %A1.
   Error(s)
   A1 IS ILLEGAL MACRO ELEMENT
   :
   :
   (Error messages deleted)

In the second example (b), the first macro definition (a) is not erased and the output is the same one by the original macro definition. The third example shows error message.
B. TIME AND SIZE OF MACROS

The size of macros defined in this paper which is the file, 300MACRO.EPS, is 57 records. EPS TEXT contains 570 records which has the logical record length 80. The size of EPS MODULE is 4 records with the record length 65535.

About the timing, the results coming from EPS execution indicates that it takes about 5 to 6 times of execution time compared to the timing through the compiler. For example, the timing information of the sample structured FORTRAN through EPS preprocessing is as follows:

T=0.23/0.55 10:40:53   (before EPS preprocessing)
T=1.44/1.87 10:41:10   (after EPS preprocessing)

About 1.32 seconds are used for the real CPU time.
The compile time by WATFIV took about 0.22 second. It appears that ML/I is not a suitable preprocessor for the high level languages like PL/I or FORTRAN as far as the timing concerns.

C. CONCLUSIONS

This project provides a possibility that a well-structured FORTRAN could be produced by programmers using the preprocessor. It is obvious that the impact of structured programming upon FORTRAN implementing general block structure does not conflict with the features of the existing FORTRAN. Using ML/I as a preprocessor for the study of the structured FORTRAN extensions
was a success except for some limited features previously illustrated.

There are several good aspects of ML/I. It has a flexible format and its use of keywords makes it easy to read. The methods of specifying repetitions of delimiters and the branching technique to groups of alternatives is a powerful facility.

However, there has been some difficulties. Defining macros according to the FORTRAN convention was the part of the difficulties. Some of the features which was not clear and undefined by ML/I have some conflict on writing macros and running ML/I. If these bad elements could be discovered in the future, ML/I will be a nice preprocessor for the structured FORTRAN extensions.

As far as considering the concept behind this project, this project was a quite successful one. To overcome the undesirable features described previously, it needs to be more study on ML/I and on the area of structured FORTRAN program.
REFERENCES


O'neill, Dennis M.  

Vaughn, W.C.M.  
APPENDIX A

SAMPLE STRUCTURED FORTRAN PROGRAM (TEST SFORT)
C$JOB
DIMENSION R(5), C(5), SAVE(5)
DROP=1; COEF=0.01; CCOREF=0.01
RAT=6.02/6.; R(6)=6.; I=6
LOOP 500 WHILE I.GT.1 DO
R(I-1)=R(I-1)/RATIO; I=I-1
ENDLOOP
DJ 505 I=7,5
505 R(I)=R(I-1)*RATIO
J=1
LOOP 510 WHILE J.LE.50 DO
IF J.LT.6 THEN C(J)=5.
ELSE C(J)=10. FI
J=J+1
ENDLOOP
I=1
LOOP 515 WHILE I.LE.50 DO
IF DROP.EQ.0.
THEN QUIT 515 LUP F1
R(I)=R(I)+(R(I)*COEF)
*/2.
IF C(I).GT.11.
THEN NEXT 515 LUP F1
C(I)=C(I)-C(I)*CCOREF
I=I+1
ENDLOOP
SUM=0.
DO 520 I=1,50
SUM=SUM+C(I)
CASE 529
GUARD I.LT.5 DO
SAVE(1)=SUM+1.
GUARD I.LE.15 DO
SAVE(2)=SUM+2.
GUARD I.LE.25 DO
SAVE(3)=SUM+2.
GUARD I.LE.35 DO
SAVE(4)=SUM+3.
GUARD I.LE.50 DO
SAVE(5)=SUM+4.
ENDCASE
520 CONTINUE
IF SAVE(5).GT.50. THEN DROP=0. FI
STOP
END
C$ENTRY
APPENDIX B

MACRO LISTINGS (SOOMACRO.EPS)
APPENDIX C

LISTING OF 300 EXEC PROCEDURE
FILE: Soo EXEC A1

&CCNTPOL OFF NMSG
COPYFILE SCOMACRO EPS A1 &1 &2 A1 EPSIN EPS A1
&STACK HT
&BEGSTACK LIFO
FILE
INPUT MSTOP
BOTTOM
END
EDIT EPSIN EPS
&STACK RT
FILEDEF &UO00 TERM
FILEDEF &O01 DISK EPSIN EPS
FILEDEF &O10 TERM
FILEDEF &O11 DISK &1 SFORTLIST
FILEDEF &O12 DISK &1 FORTLIST
FILEDEF FC27 DISK &1 &3
&TIME = GLITERAL &TIME
&TIME TYPE
EPS
&TIME = GLITERAL &TIME
&TIME TYPE
ERASE EPSIN EPS
FINIS &1 SFORTLIST
FINIS &1 FORTLIST
FINIS &1 &3
&STACK HT
&BEGSTACK LIFO
FILE
DELETE
BOTTOM
DELETE &9
DO 2
TOP
END
EDIT &1 SFORTLIST
&BEGSTACK LIFO
FILE
CHANGE /C/C$ENTRY
BOTTOM
CHANGE /C/C$JOB
DO 1
TOP
END
EDIT &1 &3
APPENDIX D

LISTING OF A SAMPLE EPSIN EPS
MCLISTS
MCLIST
MCINS T.
MSKIN MT. < >
MODEF <: AS <>
MCSET P1 = 103
MODEF <QUIT LUP>
AS <CSET T1 = TAI + 1
<GO TO> IT1.>
MODEF <QUIT LUP>
AS <GO TO> RAI.
MCDEF <SPACE WITHS LOOP WHILE DO WITHS NL
OPT ENDELCP OR NL WITHS ENDLUP ALL>
AS <MCSET T1 = CAI + 1
TAI. <IF> (NOT (CA2.)) <GO TO> ST1.>
AA3. 
<GO TO> RAI.
IT1. <CONTINUE>
MCDEF <IF OPT IFN OR NL WITHS THEN ALL
OPT ELSE CR NL WITHS ELSE ALL
OPT {P1 OR NL WITHS {P1 ALL}>
AS <MCSET P1 = P1 + 1
MCSET T1 = P1
MCSET P1 = P1 + 1
MCSET T2 = P1
A1DD. <TAI.1. <GO TO> ST1.>
AA3. 
<GO TO> ST2.
ST1. <A2.>
ST2. <CONTINUE>
MCDEF <IF OPT IFN OR NL WITHS THEN ALL
OPT P1 OR NL WITHS P1 ALL>
AS <CSET P1 = P1 + 1
MCSET T1 = P1
<IF> (NOT (TAI.1.)) <GO TO> ST1.>
A12. 
ST1. <CONTINUE>
MCDEF <CASE NL OPT NL WITHS GUARD
OPT DC OR DU WITHS NL ALL NI OR ENDCASE ALL>
AS <MCSET T1 = 1
MCSET T2 = 2
MCSET T3 = TAI.
<T1. <IF> (NOT (TAT2.)) <GO TO> ST3+1.
TAT?+1.
<GO TO> RAI.>
A11. MCSET T1 = T1 + 2
MCSET T2 = T2 + 2
MCSET T3 = T3 + 1
A1DD L2 IF TCT1+2. = <ENDCASE>
T3. <IF> (NOT (TAT2.)) <GO TO> ST3+1.
TAT2+1.
<GO TO> RAI.>
A12. ST3. <IF> (NOT (TAT2.)) <GO TO> TAI.
TAT2+1. 
A11. <CONTINUE>
CSJOB

DIMENSION R(51),C(50),SAVE(5)
DROP=1.; SCOEF=0.01; Ccoef=0.01
RATIO=6.02/6.; R(6)=6.; I=6
LOOP 500 WHILE I.GT.1 DO
R(I-1)=R(I)/RATIO; I=I-1
ENDCOP
DO 505 I=7,51

505 P(I)=R(I-1)*RATIO
J=1
LOOP 510 WHILE J.LE.50 DO
IF J.LE.6 THEN C(J)=5.
ELSE C(J)=10. FI
J=J+1
ENDCOP
I=1
LOOP 515 WHILE I.LE.50 DO
SIP DROP=EQ.0.
THEN QUIT 515 LUP FI
R(I)=R(I)+R(I)*SCOEF
*/2.
SIF C(I).GT.11.
THEN NEXT 515 LUP FI
C(I)=C(I)-C(I)*Ccoef
I=I+1
ENDLoOP
SUM=0.
DO 520 I=1,50
SUM=SUM+C(I)
CASE 525
GUARD I.LT.5 DO
SAVE(1)=SUM*1.
GUARD I.LE.15 DO
SAVE(2)=SUM*2.
GUARD I.LE.25 DO
SAVE(3)=SUM*3.
GUARD I.LE.30 DO
SAVE(4)=SUM/3.
GUARD I.LE.50 DO
SAVE(5)=SUM/4. ENDCASE
520 CONTINUE
SIP SAVE(5).GT.50. THEN DROP=0. FI
STOP
END
C$ENTRY
MCSTCP
APPENDIX E

LISTINGS OF SAMPLE STRUCTURED FORTRAN OUTPUT

FROM EPS II
ERROR MESSAGES LISTING

SOURCE PROGRAM LISTING

DIMENSION R(S1),C(S0),SAVE(5)
DROPO=1.1 SCOE=0.01: CC=0.01
RATIO=6.07:0.1 R(B)=6.1: I=6
LEA 500 WHILE I.GT.1 DC
R(I-1)=R(I)/RATIO: I=1-1
EANLOP
DO 505 I=7,51
      505 R(II)=R(I-1)*RATIO
      J=J+1
      LOOP 510 WHILE J.LE.50 DO
      IF J.LT.6 THEN G(I)=5.
      ELSE G(I)=10. FI
      J=J+1
      ENL0CP
      U(I)=
      LOOP 515 WHILE I.LE.50 DD
      SIF DROPO=0.
      THEN QUIT 515 LOP FI
      R(I)*=R(1)+(A(II)*SCOE)
      */2.
      SIF C(I)GT.11.
      THEN NEXT 515 LOP FI
      C(I)=C(I)-(C(I)/CCOE)
      I=I+1
      ENL0CP
      SUM=0.
      DO 520 I=1,50
      SUM=SUM+C(I)
      CASE 525
      EXIT 525
      520 CONTINUE
      SAVE(I)=SUM+1.
      EXIT 525
      QUAD 1.0 LE.15 DO
      SAVE(2)=SUM+2.
      EXIT 525
      QUAD 1.0 LE.12.5 DD
      SAVE(3)=SUM+3.
      EXIT 525
      QUAD 1.0 LE.50 DD
      SAVE(5)=SUM+4.
      ENCASE
      SIF SAVE(5).GT.50. THEN DROPO=0. FI
      STOP
      END
TARGET PROGRAM LISTING

C
DIMENSION R(51), C(50), SAVE(5)
DROP=1,
SŒOF=0.01
CŒOF=0.01
RATIQ=0.02/6.
R(6)=6.
00008861 I=6
00008862 500 IF (.NOT.(I.GT.1)) GO TO 501
+ R(I-1)=R(I)/RATIO
+ I=1-1
+ GO TO 520
+ 00000064 501 CONTINUE
00000065 DO 505 I=7,51
00000066 505 R(I)=R(I-1)*RATIO
00000067 J=I
00000068 510 IF (.NOT.(J.LE.50)) GO TO 511
+ IF (J.LT.6) GO TO 101
+ C(J)=10.
+ GO TO 102
+ 101 C(J)=5.
+ 102 CONTINUE
+ J=J+1
+ GO TO 510
00000072 511 CONTINUE
00000073 I=1
00000074 515 IF (.NOT.(I.LT.50)) GO TO 516
+ IF (.NOT.(DROP.EQ.0.)) GO TO 103
+ GC TO 516
+ 103 CONTINUE
+ R(I)=R(I)+R(I)*SŒOF
+ #/2.
+ IF (.NOT.(C(I).GT.11.)) GO TO 104
+ GO TO 515
+ 104 CONTINUE
+ C(I)=C(I)-(C(I)*CŒOF)
+ I=I+1
+ GO TO 515
00000081 516 CONTINUE
00000084 SUM=0.
00000085 DC 520 I=1,50
00000086 SUM=SUM+C(I)
00000087 IF (.NOT.(I.LT.51)) GO TO 526
+ SAVE(I)=SUM+1.
+ GO TO 525
+ 526 IF (.NOT.(I.LE.15)) GO TO 527
+ SAVE(2)=SUM+2.
+ GC TO 525
+ 527 IF (.NOT.(I.LE.25)) GO TO 528
+ SAVE(3)=SUM+3.
+ GO TO 525
+ 528 IF (.NOT.(I.GT.30)) GO TO 529
+ SAVE(4)=SUM/3.
+ GO TO 525
+ 529 IF (.NOT.(I.LE.53)) GO TO 525


R:
TY TEST WATFIV

C$JOB
DIMENSION R(51), C(50), SAVE(5)

DROP=1.
SCOEF=0.01
CDEF=0.01
RAT=0.02/6.
R(6)=6.
I=6

500 IF (.NOT.(I.GT.11)) GO TO 501
   R(I-1)=R(I)/RAT
   I=I-1
   GO TO 500

501 CONTINUE
   GO 505 I=7,51

505 P(I)=R(I-1)*RAT
   J=1

510 IF (.NOT.(J.LT.50)) GO TO 511
   IF (J.LT.6) GO TO 101
   C(J)=10.
   GO TO 102

101 C(J)=5.
102 CONTINUE
   J=J+1
   GO TO 510

511 CONTINUE
   I=1

515 IF (.NOT.(I.LE.50)) GO TO 516
   IF (.NOT.(DROP.EQ.0.)) GC TO 103
   GO TO 516

103 CONTINUE
   R(I)=R(I)+(R(I)*SCOEF)
   */.2.
   IF (.NOT.(C(I).GT..11.)) GO TO 104
   GO TO 515

104 CONTINUE
   C(I)=C(I)-C(I)*CDEF
   I=I+1
   GO TO 515

516 CONTINUE
   SUM=0.
   GO 520 I=1,50
   SUM=SUM+C(I)
   IF (.NOT.(I.LT.51)) GO TO 526
   SAVE(I)=SUM=1.
   GO TO 525

526 IF (.NOT.(I.LE.15)) GO TO 527
   SAVE(2)=SUM=2.
GO TO 525
527 IF (.NOT.(I.LE.25)) GO TO 528
SAVE(3)=SUM/2.
GO TO 525
528 IF (.NOT.(I.LE.30)) GO TO 529
SAVE(4)=SUM/3.
GO TO 525
529 IF (.NOT.(I.LE.50)) GO TO 525
SAVE(5)=SUM/4.
525 CONTINUE
520 CONTINUE
IF (.NOT.(SAVE(5).GT.50.1)) GO TO 105
DROP=0.
105 CONTINUE
STOP
END

C$ENTRY

R:
MATHIV TEST
R:
TY TEST LISTING
C$JOB
1 DIMENSION R(51),C(50),SAVE(5)
2 DROP=1.
3 SCDEF=0.01
4 CCOEF=0.01
5 RATIO=0.02/6.
6 R(0)=0.
7 I=6
8 500 IF (.NOT.(I.GT.1)) GO TO 501
9 R(I-1)=R(I)/RATIO
10 I=I-1
11 GO TO 500
12 501 CONTINUE
13 DO 505 I=7,51
14 505 R(I)=R(I-1)*RATIO
15 J=1
16 510 IF (.NOT.(J.LE.50)) GO TO 511
17 IF (J.LT.6) GO TO 101
18 C(J)=10.
19 GO TO 102
20 101 C(J)=5.
21 102 CONTINUE
22 J=J+1
23 GO TO 510
24 511 CONTINUE
25 I=1
26 515 IF (.NOT.(I.LE.50)) GO TO 516
27 IF (.NOT.(DROP.EQ.0.1)) GO TO 103
28 GO TO 516
29 103 CONTINUE
30 R(I)=R(I)+(R(I)*SCDEF)
31 #/2.
32 IF (.NOT.(C(I).LT.11.1)) GO TO 104
33 GO TO 515
34 104 CONTINUE
35 C(I)=C(I)-(C(I)*CCOEF)
36 I=I+1
GO TO 515
516 CONTINUE
SUM=0.
GO TO 520 I=1,50
SUM=SUM+C(I)
IF (.NOT.((I .LT. 5)) GO TO 526
SAVE(1)=SUM/1.
GO TO 525
526 IF (.NOT.((I .LE. 15))) GO TO 527
SAVE(2)=SUM/2.
GO TO 525
527 IF (.NOT.((I .LE. 25))) GO TO 528
SAVE(3)=SUM/3.
GO TO 525
528 IF (.NOT.((I .LE. 30))) GO TO 529
SAVE(4)=SUM/4.
GO TO 525
529 IF (.NOT.((I .LE. 50))) GO TO 525
SAVE(5)=SUM/5.
GO TO 525
525 CONTINUE
520 CONTINUE
57 IF (.NOT.(SAVE(5) .GT. 50.)) GC TO 105
58 DROP=0.
59 105 CONTINUE
STOP
END

ENTRY

CORE USAGE OBJECT CODE= 1640 BYTES, ARRAY AREA= 424 BYTES, TOTAL AREA AVAILABLE= 22096 BYTES
DIAGNOSTICS NUMBER OF ERRORS= 0, NUMBER OF WARNINGS= 0, NUMBER OF EXTENSIONS= 0
COMPIL TIME= 0.22 SEC, EXECUTION TIME= 0.02 SEC, 10.53.05 TUESDAY 22 MAR 77 WATF IV - JAN 1976 V1L5

R:
SPCCL CONSOLE CLOSE STOP
ABSTRACT

The objective of this project is to investigate use of a general purpose macro processor to implement extensions to FORTRAN.

The macro processor is a software translator which interprets macro definitions and translates input text to output text. Input to the macro processor is in the form of macro definitions and source text. The output text is derived from the source text by replacing all the macro calls that occur in it. Available at KSU is EPS II, a version of ML/I (Bro67) which is a general purpose macro processor system and which can be easily applied to all existing programming languages. EPS II can be run interactively under CMS as a preprocessor for FORTRAN extensions.

For this study, a set of some macro extensions to FORTRAN were defined to facilitate structured programming in FORTRAN. Structures examined were IF THEN ELSE, WHILE, and CASE.