PUC

Series: Monographs in Computer Science
and Computer Applications

Nº 1/76

EXTENDING THE CONTROL STRUCTURES OF FORTRAN VIA A MACRO-GENERATOR

by

Rubens N. Mello

and

Daniel Schwabe

Departamento de Informática

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

Series: Monographs in Computer Science
and Computer Applications

Nº 1/76

EXTENDING THE CONTROL STRUCTURES OF FORTRAN VIA A MACRO-GENERATOR*

by

Rubens N. Mello

and

Daniel Schwabe

Series Editor: Sergio E.R. Carvalho

January, 1976

*This work was supported in part by the Brazilian government Agency FINEP under contract Nº 244/CT and developed by the Information Systems Group of the Departamento de Informática-PUC/RJ.

Copies may be requested from:

Rosane T.L. Castilho, Head
Setor de Documentação e Informação
Deptº de Informática - PUC - RJ
R. Marquês de São Vicente, 209 - Gávea
20000 - Rio de Janeiro - RJ - Brasil

ABSTRACT:

A technique for extending programming languages via a general purpose macro-generator is introduced. Particularly an implementation of important control structured programming in FORTRAN and some text substitution facilities are presented. The portability and case of implementation issues are discussed as well as possibilities of further extensions.

KEY WORDS:

FORTRAN, macro-generators, portability, language extensions.

RESUMO:

Uma técnica de extensão de linguagens via um gerador de macros de propósito geral é apresentada. De modo particular a implementação de estruturas de controle importantes para a programação estruturada em FORTRAN e facilidades de substituição de texto são mostradas. Portabilidade, facilidade para novas extensões e adaptabilidade a outras linguagens são consideradas as principais características do método apresentado.

PALAVRA CHAVE:

FORTRAN, macro-expansores, portabilidade, extensões de linguagem.

CONTENTS

1 INTRODUCTION

FORTRAN is one of the oldest programming languages and probably one of most widespread use.

As one reads the present literature on programming languages, it is almost impossible to avoid coming across with terms like "structured programming", "go-to-less-programs", etc. Regardless of what these terms actually mean, some programming languages mechanisms proposed by the "structuralists" have been widely tested and there is now some consensus about their usefullness. These mechanisms include some recently proposed control structures, data type definition facilities and so forth.

No matter how bad FORTRAN is, it is still used in many applications for efficiency and portability reasons and this is why it is probably still going to be used for a much longer period than many would like.

Along these many years of existence, its definition has suffered a great number of modifications, ranging from the "general" to the very specific extensions. Some of those extensions have been implemented as new languages, others simply as extended FORTRAN.

Nevertheless, various recent extensions have addressed similar problems [1,2], such as the lack of appropriate control structures. In this report we propose an extension to FORTRAN and although one may argue about the validity of extending FORTRAN (in any sense) we feel that our proposed extension is justified, at least as an adequate notation.

In the following sections we describe a FORTRAN extension - called FORTS - developed at PUC/RJ, and discuss the features introduced in the language and the method of implementation. A user's manual is also included in the appendix.

2 FEATURES OF FORTS

2.1 General Description

FORTS accepts and translates into any dialect of FORTRAN IV. It is

an extension to FORTRAN in the sense that it adds "structured" commands to the language. The following keywords are reserved: IF, THEN, ELSE, FI, DOWHILE, OD, REPEAT, UNTIL, FOR, ROF, FROM, TO, BY, DOCASE, ESACOD, CASE, ESAC. Any valid FORTRAN program is accepted by FORTS.

All "structured" commands (keywords) are preceded by a special character. Currently the '_' (underscore) is being used. Therefore, the above mentioned keywords are reserved only when preceded by this character. Statement numbers should not be greater than 87000.

Only one such command may appear in each card and there is no possibility of statements spreading over more than one line. Comments may appear anywhere in the program, as in standard FORTRAN.

Each particular command has its own way of grouping statements. The grouping is done according to keywords.

The user may ask for a listing of the input program. The listing produced is exactly the input. For each of the pairs _IF _FI, _DOWHILE _OD, _DOCASE _ESACOD, _FOR _ROF, a separate nesting number is printed, along with the corresponding command.

2.2 Statements Available (*)

(i) SELECTION

_IF B	IF (.NOT.(B)) GO TO 100
_THEN	
S_1	S_1
S_2	S_2
.	.
.	.
.	.
S_n	S_n
_IF	100 CONTINUE

_IF B	IF (.NOT.(B)) GO TO 100
_THEN	
S_1	S_1
.	.
.	.
.	.
S_n	S_n
	GO TO 200
_ELSE	100 CONTINUE
S_{n+1}	S_{n+1}
.	.
.	.
.	.
S_m	S_m
_FI	200 CONTINUE

---

* For the sake of simplicity, the statement numbers in the examples are not equal to those that are actually generated.

This is the usual if-then-else construct. In our case, B is any logical expression valid in FORTRAN S_1, ... , S_n, S_{n+1}, ... S_n can be any valid statement, including "structured" ones.

If the logical expression B is .TRUE., then the group of statements S_1, ... S_n is executed, and the next command is the one following the _FI statement. If B is .FALSE., then the group of statements S_{n+1}, ..., S_m is executed, followed by the statement after_FI.

When the ELSE clause is not present and B is .FALSE., the statement after _FI is executed. Only the _IF statement may be labelled.

(b) _DOCASE

_CASE B_1

S_{11}

⋮

S_{1n_1}

_ESAC

_CASE B_2

S_{21}

⋮

S_{2n_2}

_ESAC

⋮

_CASE B_m

S_{m_1}

⋮

S_{mn_m}

_ESAC

_ESACOD

GO TO 100

100 IF (.NOT.(B_1)) GO TO 200

S_{11}

⋮

GO TO 999

200 IF (.NOT.(B_2)) GO TO 300

S_{21}

⋮

S_{2n_2}

GO TO 999

⋮

IF (.NOT.(B_m)) GO TO 999

S_m

⋮

S_{mn_m}

GO TO 999

999 CONTINUE

This is the case statement. B_1, ... , B_m are valid FORTRAN logical expressions; S_{11}, ... , S_{mn_m} are valid FORTS statement (including "structured" ones). For the first B_i that is.TRUE., the group S_{11}, ... , S_{in_i} is executed.

The equivalent FORTRAN code is on the right side of the figure above. No statement are allowed between an _ESAC and the following _CASE or between _DOCASE and _CASE or _ESAC and ESACOD. Only the _DOCASE statement may be labelled.

(ii) ITERATION

(a)

```
_DOWHILE  (B)                    100 IF (.NOT. (B))  GO TO 200
    S1                                  S1
    :                                   :
    Sn                                  Sn
_OD                                  GO  TO  100
                                 200  CONTINUE
```

This is the do-wwhile statement. B is any valid FORTRAN logical expression, and S_1, ..., S_n are valid FORTS statements (including "structured" ones). Statements S_1 through S_n are executed repeatedly as long as B is .TRUE. . The _DOWHILE statement cannot be labelled.

(b)

```
_REPEAT                          100  CONTINUE
    S1                                  S1
    :                                   :
    Sn                                  Sn
_UNTIL (B)                       IF (.NOT.(B)) GO TO 100
```

This is the repeat-until statement. B and S_1, ..., S_n are exactly as in the do-while statement. Statements S_1, ..., S_n are executed repeatedly until B is .TRUE.. The REPEAT statement cannot be labelled.

(c)

_FOR I_1 _FROM I_2 _TO I_3 _BY I_4	
	$I_1 = I_2$
	100 CONTINUE
S_1	S_1
⋮	⋮
S_n	S_n
	$I_1 = I_1 + I_4$
_ROF	IF((I_4 .GE.0.AND.I_1.LE.I_3).OR.
	(I_4.LT.0.AND.I_1.GE.I_3)) GO TO 100

This is equivalent to the for statement. I_1, I_2, I_3 and I_4 are always of the same type; I_1 is a valid FORTRAN variable; I_2, I_3 and I_4 are valid FORTRAN arithmetic expressions. The statements S_1, ..., S_n are executed repeatedly, varying the value of I_1 from I_2 to I_3 by I_4. The increment (I_n) may be negative. The _FOR statement cannot be labelled.

2.3 Text Substitution Facilities

Since FORTS is an extended general purpose macro-generator, all the usual macro-facilities are available; the syntax for these macro-statements is very similar as that of Strachey's GPM [3]. The important detail is that cards containing these statements must have a special character in column 1. The user may place frequently used macros in a library, which is maintained by built-in macros. See the appendix for more details.

2.4 Error Detection and Messages

The translator detects "structured" command errors such as lack of pairing keywords, wrong number of arguments, inexistent commands. The translator does not check for syntax errors in the logical expressions, argument compatibility and so forth. The translator tries to detect as many errors as possible, and translation continues until the input is ended.

3. THE IMPLEMENTATION

As it was mentioned before, the generation of the FORTRAN statements corresponding to the structured statements was done via macro calls. One of the main reasons for using this technique was the availability of a general purpose macro-generator written in PL/I. The addition of a single routine permitted us to achieve immediately the first running version of the system.

The first experiments with this pre-processor revealed some difficulties, mainly concerning speed. Since the PL/I program was written in a very simple manner, we decided to translate it into FORTS to be able to get a portable and efficient system.

Using this structured FORTRAN program as input to the PL/1 version of the preprocessor, we finally produced a second version which is an ANSI 1966 standard FORTRAN with the exception that it uses INTEGER*2 statements which were included only for economy of space.

The program has about 800 statements requiring approximately 70K with code generated by IBM's FORTRAN H compiler, and it expands itself in about 7 seconds in our IBM/370-165, with OS-MVT.

In this way we finally obtained a structured FORTRAN system which is easily implemented in any installation having a standard FROTRAN IV compiler.

In summary, we feel that the choice of implementation of extensions via macros instead of direct modification of the FORTRAN compiler has some clear advantages. First, it allows a standard definition of these extensions (there are groups presently working towards this same goal [2]). Second, macros allow more flexibility in the creation of new statements or modifications to existing ones - it suffices to alter the macro definitions. Moreover, the extension of the language in this way is actually open-ended, giving us the freedom to experiment also with different features such as new data types, etc.

Evidently there are disavantages, some of which are inherent to preprocessing, as for instance difficulties in error detection and increase in overall processing time. One important disadvantage of our particular preprocessor is the limitation imposed on the syntax, because it is not sintax-directed.

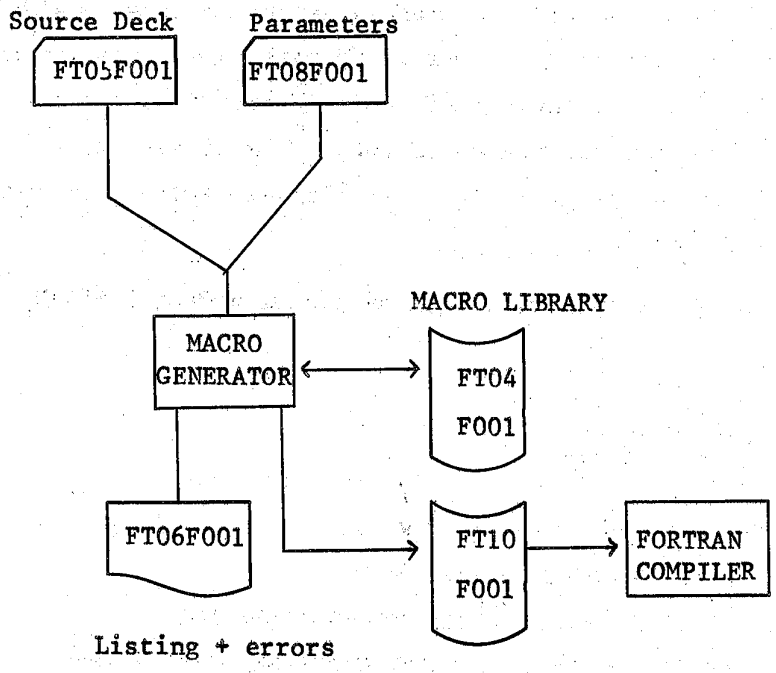
4. CONCLUSION

FORTS is a simple extension to FORTRAN which has proved to be an excellent tool for structured programming in FORTRAN. Maybe it should be regarded as more than just a simple extension to FORTRAN because during the preprocessing phase the user has available not only the structured statements but also a very powerful macro-processor. In this way the user has the freedom to define and experiment with any new statement. This extensibility has led us to follow the intent of adding more features to the system. Among these additions we include the introduction of some data structures not available in FORTRAN and facilities for program instrumentation. It is also worth mentioning the existence of a separate project that is working towards the addition of a DML facility for a data base system based on the proposals of DBTG [4]. As yet another application of FORTS we should mention that is has been successfully used as a programming tool in a project that involves the construction of basic software for a minicomputer [5].

APENDIX

USER'S MANUAL

1. FLOW OF DATA



2. DESCRIPTION OF FILES

FT05F001 - Contains the source deck to the Macro Generator; the source format was described previously.

FT08F001 - Contains the parameters for the macro generator. Presently, these parameters are

<$!;@>'|"/_13

These parameters control the macro expansion; the library is dependent on the parameters. If the user desires only a source listing of his program, without the corresponding FORTRAN program output, he should specify 03 instead of 13 in the parameter card. For more details on the other parameters,see the Macro Facilities section.

FT06F001 - Contains the source listing with error messages.

FT04F001 - Is the Macro-library; this is a direct access FORTRAN file, containing the macro definitions for the structured statements. It is described by the following DEFINE FILE:

DEFINE FILE 4(500, 125, U, KK)

Its first 4 records are directory blocks. The directory should be regarded as an integer matrix of ten columns in which each row describes the name of a macro (8 positions), the number of the first record containing its body (1 position) and the length of the body in characters (1 position). The library is maintained by built-in macros (see Macro Facilities section) or optionally via utility programs.

FT10F001 - Contains the output, i.e., the corresponding FORTRAN program.

3. SAMPLE JCL FOR OS/360-MVT

```
//  NAME JOB----
//  FORTS EXEC PGM = MACF1, REGION = 75K
//* THIS IS THE LOAD MODULE OF THE MACRO GENERATOR
//  STEPLIB  DD  DSN = CPG3264.MACRO.EXPANSOR, DISP = SHR
//  FT05F001 DD  DDNAME = SYSIN
//  FT06F001 DD  SYSOUT A
//  FT10F001 DD  DSN = &&TEMP, DISP = (NEW, PASS), SPACE(80, (250, 250)),
//  UNIT SYSSQ
//  FT04F001 DD  DSN = CPG3264.MACRFORT, DISP = SHR
//* THIS IS THE MACRO LIBRARY
//  FT08F001 DD  *
<$!; @>'|"/_13
/*
//  SYSIN DD *
        ]----     SOURCE DECK
```

```
/*
// COMP   EXEC   FORTJ,---
// FORT.SYSIN    DD  DSN = &&TEMP, DISP =(OLD, DELETE)
          .
          .
          .
//
```

4. MACRO FACILITIES

The user has available to him a full general purpose macro-generator, with sintax very much like Strachey's GPM [Strachey, Gries].

All cards containing statements directed to the macro generator begin with the character ' " ' (which may be redefined in the parameters to be any other character); it should be noted that this does not include the cards containing "structured" statements.

For elementary text substitutions, the following mechanisms will suffice.

(i) - Precede the program with cards of the form

"$DEFINE!name!string; or _DEFINE!name!string

(ii) - In the program text, the user may write the symbol _name instead of the string.

For example, if we wish to write the constant PI in the program, and have the value 3.14159 uniformly substituted for it, we write

```
"$DEFINE!PI!3.14159;
```

and the program text

```
AREA=_PI*R*2
      .
      .
      .
_IF ANG.LT._PI
   _THEN
      .
      .
      .
```

For more sophisticated expansions, see [3]

Note that a whole subroutine may be inserted in this way. Frequently used texts (subroutines, commons, etc...) may be placed in the macro-library, trough the built-in macro SYST. For example, if we wanted to put the macro PI defined previously in the library, we should write:

"$SYST!PI; or _SYST!PI

For a macro to be put in the library, it must have been previously defined in the same run; furthermore, it must not have been filed before. If the user whishes to update the body of a macro which is already in the library, he must first delete this macro from the library through the built-in macro DSYST, and then file it again through a call to SYST after having updated its body using the built-in UPDATE macro. For example, if we want to change the value of PI that is in the library (3.14159) to 3.1416, we must write

"$UPDATE!PI!3.1416; or _UPDATE!PI!3.1416

"$DSYST!PI!$SYST!PI; or _DSYST!PI_SYST!PI

Both SYST and DSYST accept more than one macro to be filed deleted in one call. For example, if E is also another macro previously defined (but not filed) it is valid to write

"$SYST!PI!E; or _SYST!PI!E

The advantage of filing a macro in the library is that, once that has been done, there is no need to repeat the definition each time the program runs; this definition is automatically retrieved from the library when necessary.

REFERENCES

1 - MEISSNER, L.P. On extending FORTRAN structures to facilitate structured programming. SIGPLAN Notices, New York, 10(9) sept. 1975.

2 - FORWORD: FORTRAN development newsletter, (5), oct. 1975 (ad hoc comitte on FORTRAN-SIGPLAN)

3 - STRACHEY, C. A general purpose macro-generator. Computer I., London (Journal(?)), 8 : 225-41, 1965.

4 - CODASYL Data Base Task Group. New York, ACM, 1971.

5 - UM ASSEMBLER para o mini-computador G-10; Manual de lógica. Rio de Janeiro, PUC, CONSULPUC/Projeto Guaranys. (to be published) 76