MP/1—a FORTRAN macroprocessor*

I. A. Macleod

Department of Computing and Information Science, Queen's University, Kingston, Ontario

This paper describes a macroprocessing system which accepts input and generates output in the standard FORTRAN format. A wide range of macro names is allowed, thereby allowing syntactically meaningful macros to be constructed so that in effect, syntactic extensions to FORTRAN can be developed.

(Received May 1970)

This paper outlines the main features of MP/1, a macroprocessor developed specifically for use with FORTRAN. The main aim of the design has been to produce an efficient macroprocessor for FORTRAN programmers in which operations for particular classes of problems can be expressed more naturally than in the existing FORTRAN syntax. By its relation to FORTRAN MP/1 differs from other recent macroprocessors such as LIMP (Waite, 1967) and ML/1 (Brown, 1967) which are not biased towards any one language, but are intended for more general use. This paper also compares MP/1 with these and other macroprocessors.

As far as possible the notation used by Brown in his recent survey of macroprocessors (Brown, 1969) has been followed. The use of a macro is termed a macro call. A macro definition is made up of a macro name, which defines the syntax of the macro call, and replacement text, which specifies the object code with which the macro call is to be replaced. Operations used inside the replacement text to control the actual generation of code are called macro-time operations. The language generated by the macroprocessor is called the base language.

As well as the question of efficiency, three major aspects need to be taken into consideration in the evaluation of a macroprocessor.

First, the way in which macro names can be defined is of primary importance in macroprocessors designed for use with high level languages where one of the major applications will be to enrich the syntax of the base language. This consideration is less important in macro assemblers such as the 360 macro assembler (Freeman, 1966) where the macros are similar in format to assembler language statements. This paper is, however, more concerned with macroprocessor applications to high level languages. Second, the way in which parameters are handled is important both from the point of view of ease of use and also in that it can affect the range of allowable name constructions. Finally, the macro-time operations should be straightforward and easy to use. This is particularly important if a macroprocessor is to develop beyond being a specialised tool largely restricted to software writers and become a programming aid easily usable by less sophisticated programmers. The tailoring of a macroprocessor to a specific language allows a significant advantage in this respect since the syntax of the macro-time statements can be derived from that of the analogous statements in the base language.

Macro name syntax

In MP/1, a macro name is made up of a set of one or more delimiters separated by a formal parameter. In this respect the name construction differs from that used by Brown who defines the macro name to be the initial delimiter of the construction. In addition the macro may begin or end with a formal parameter.

For example,

ADD @@ TO @@
@@ + @@

are both instances of valid names. (The @@ symbol denotes a formal parameter). Thus MP/1 differs from most other macroprocessors in that a formal parameter can precede the first delimiter of the call. While this necessarily increases the time required to match macro calls, it allows a greater range of macro names. Infix constructions, for example, which would seem to be desirable in a high level language macro facility are allowable. Both GPM (Strachey, 1965) and the current PL/1 macro facility (IBM, C28-6571, 1966) have much more restricted name formats. In PL/1 for example the macro name takes the same form as a procedure name, thereby providing very little in the way of syntactic extension. LIMP allows names to be constructed in much the same way as in MP/1 but has no provision for allowing a parameter to represent a variable length list of elements.

For example, in MP/1 the macro name

ADD @AND@ TO @@

would match both of the calls

ADD A AND B AND C TO X
ADD C TO D

i.e. the formal parameter @AND@ corresponds to a variable length list of actual parameters where each element of the list is separated by a predefined delimiter which in this case is the symbol AND. This facility can however be simulated in LIMP, though presumably less efficiently, using the macro-time facilities available. ML/1 has a similar feature, but macro names representing infix operations cannot be constructed. PL/1 requires macros to have a fixed number of arguments. Macro calls are delimited by a warning character, normally a % character, and the implicit end of line character. Where calls are continued over more than one line, the normal FORTRAN convention for indicating continuation lines is used.

Additionally MP/1 can require arguments to be balanced algebraically with respect to left and right parentheses, a facility also available in Waite's Stage 2 macroprocessor, (Waite, 1970), and also allows default values to be given. For example, in the macro name

ADD @(,) = '1'@ TO @@

the first formal parameter corresponds to a list of balanced arguments separated by commas. If no actual argument is present in the call the default value is 1. Thus for

ADD A,B(1,1), C TO D

there are three elements in the first argument and

*This work was supported in part by the National Research Council of Canada, grant number A7326.

ADD TO D

in which the first argument is omitted, is equivalent to

ADD 1 TO D

Of all the current macroprocessors, XPOP (Halpern, 1967) offers the greatest variety of name constuctions. In addition to the normal delimiter constructions, XPOP allows 'noise words' which may be optionally included in the macro call and keyword parameters, similar in principle to the equivalent facility in the 360 macro assembler, which allow parameters to appear in any order. Thus the same macro call can often be written in many different ways. While this degree of flexibility can sometimes be useful its value in conjunction with a FORTRAN type of base language is debatable.

Macro-time facilities and parameter inserts

In MP/1 the macro-time statements are designed to resemble as far as possible equivalent execution time statements in FORTRAN. For example, the macro name given above could be defined as follows:

```
% DEF   ADD @( , ) = '1'@ TO @@
%       CALL ARGS (@1@,#LV)
%       #LW = 0
%  10   #LW = #LW + 1
        @2@ = @2@ + @1:#LW@
%       IF (#LW.LT.#LV) GO TO 10
% END
```

Variables are preceded by a # symbol, @1@ refers to the first argument and @2@ to the second, while @1:#LW@ refers to the #LWth element of the first argument, which is assumed to be a list. The subroutine ARGS stores the number of elements in the first argument into the variable #LV. All macro-time statements are preceded by a % sign. Thus

ADD A,B,C(1, 2) TO X

generates

```
X = X + A
X = X + B
X = X + C(1, 2)
```

Alternatively, the macro could be written as

```
%DEF ADD @( , ) = '1'@ TO @@
     @2@ = @2@ + @1:+@
%END
```

In this case the above call generates

X = X + A + B + C (1, 2)

This indicates how an argument list can be reproduced with a different argument separator.

MP/1 also allows both string variables and limited string matching operations using the same FORTRAN type logical IF statement. The full range of macro-time statements and facilities are described elsewhere (Macleod, 1969), but the first macro definition given above illustrates their FORTRAN-like nature.

In the PL/1 preprocessor, the macro-time statements are virtually identical to the execution-time PL/1 statements. The replacement text for LIMP, on the other hand, is written in a SNOBOL-like language. While this provides an extremely powerful string processing facility there is a corresponding increase in macro expansion time. It is interesting to note that Waite's second macroprocessor, Stage 2, has abandoned the SNOBOL type statements. Most other macroprocessors have either developed their own syntax for macro-time operations or take the GPM approach of treating macro-time statements as ordinary macros.

Other features

In common with most macroprocessors, MP/1 allows macro calls to appear in non-macro statements and as arguments nested in other macro calls. In both cases the nested macros are delimited by square brackets. The same delimiters are used to denote macro calls inside definitions. One interesting feature of MP/1 is that macro calls can be generated dynamically in the replacement text and this property can often be utilised to avoid bracketing of nested macro calls.

For example, the macro definitions

```
%DEF *@@
     @1@
%END
%DEF *(@( , )@)
     NULIST([*@1], [*@])
%END
%DEF @@- > LIST@( , )@
     @1@ = NULIST([*@2], [*@])
%END
```

associated with the call

L - > LIST A, (B, (C, D)), E

would generate the following steps:

```
L = NULIST([*A], [*(B, C, D))], [*E])
L = NULIST (A, [*(B, (C, D))], [*E])
L = NULIST (A, NULIST([*B], [*C, D)]), [*E])
```

and so on, until finally the expansion is output as

L = NULIST (A, NULIST(B, NULIST(C, D)), E)

This particular series of definitions illustrates how complex macro expansions may be handled by the system. Note in this case that the submacro [*(B, (C, D))] could be taken as a call of either the first or second macros defined. This apparent ambiguity is resolved in the matching process by searching backwards through the defined names so that an attempt will initially be made to match the call against the most recently defined macro. This particular example would thus be matched against the name *(@(,)@) in preference to the name *@@.

As indicated earlier, macro calls are normally preceded by a % character which flags the beginning of a macro call. However, the user may specify that some alternative character is to be used as a warning flag. For example, a blank can be used, in which case the format of a macro call becomes identical to that of a FORTRAN statement.

A further feature of MP/1 is that macro-time statements are compiled rather than interpreted which increases markedly the efficiency of looping operations within the replacement text, such as, for example, indexing through an argument list.

FORTRAN related features of MP/1

Apart from the syntax of the macro-time statements, the tailoring of MP/1 to FORTRAN provides a number of other features which can only be provided awkwardly, if at all, by other macroprocessors.

The format of input to, and output from, MP/1 is exactly the same as in FORTRAN, i.e. continuation times, label fields etc., are treated in the same way as by a FORTRAN compiler. On output, continuation lines are generated automatically if necessary. The user need not make any provision for the generating of unique labels within the replacement text, as this is again handled automatically.

While some FORTRAN compilers allow Hollerith constants to be replaced by character strings, others do not. MP/1 has therefore been given the facility to convert quoted strings in macro arguments and replacement text into Hollerith constants during evaluation.

Where a macro call is labelled this label is normally transferred to the first statement generated by the replacement text. However, if this statement is itself labelled, a FORTRAN CONTINUE statement will be first generated and given the label of the macro call.

Certain FORTRAN compilers require statements to be input in the order specified in the ASA FORTRAN standards and MP/1 accordingly contains macro-time statements which can be used to order statements in the output stack according to the ASA specifications.

Finally MP/1 follows the normal FORTRAN convention of treating blanks as non-significant characters. Thus the macro name

LIST @@

would match either of the macro calls

LIST A

or

LISTA

This means that MP/1 does not follow the ML/1 approach of storing macro delimiters as atoms, and that matching is therefore performed character by character.

Summary

MP/1 was originally implemented on a list structure basis, as is LIMP. While in principle this gives a greater degree of flexibility in designing macro-time operations, it was found that for MP/1 a stack based system was much more efficient and that all the features considered desirable could still be easily incorporated. This finding supports Brown's suggestion (Brown, 1969) that the use of list processing techniques may well generate overheads making the use of the macroprocessor impractical.

As mentioned earlier, MP/1 uses a character by character scan in matching macro calls against the appropriate macro name, this being necessitated by the FORTRAN convention of treating blanks as non-significant characters, thereby preventing the use of an atom based match of the type used by ML/1. Certainly if this consideration were not important a much faster name matching algorithm could be constructed using an atom based system in conjunction with a hash table.

The tailoring of MP/1 to FORTRAN has allowed the implemented system to be both efficient and easy to use, in that the macro-time statements are FORTRAN-like thus following McIlroy's suggested approach to the design of macro systems (McIlroy, 1960). Further, the macro name syntax allows the system to be used to provide a significant extension to the base language. The PL/1 preprocessor, while providing macro-time facilities whose syntax is essentially that of PL/1 itself, has an extremely restricted name format, making the processor unsuitable for syntactic extension. Some of the defects of the current preprocessor are illustrated in a recent work on the incorporation of a pattern directed string processor in PL/1 (Oppen, 1970).

It is the author's belief that the tailoring of a special purpose macroprocessor to a particular language is justified both because the processor can cater more efficiently for the idiosyncrasies of that language, and also because the facilities of the processor can be incorporated into a syntax already familiar to programmers in that language. In the context of language extendability it is certain that some sort of macro-like facility will become common in future programming languages. It is noteworthy that a recent proposal (Davies and Paradine, 1969), has advocated extensive changes in the PL/1 preprocessor which will allow the construction of 'trigger' macros. The macro name syntax in this case is similar to that of ML/1 but the arguments of trigger macro may themselves be macros, called syntax macros, which have a less restrictive notation and can for example be used to represent infix expressions.

While it is not claimed that MP/1 is the ideal macroprocessor, it is hoped that it will be of use in the design of similar facilities in both present and future programming languages.

Applications of MP/1 include the incorporation of a SNOBOL like extension into FORTRAN-IV (Macleod, 1970). The system is currently being extended to allow its use in developing interactive problem solving languages (Mandil, 1970).

Acknowledgements

The author wishes to thank Professor R. M. Pengelly for many helpful suggestions made during the course of this work and also the referee for several useful criticisms.

References

Brown, P. J. (1967). The ML/1 Macro Processor, *CACM*, Vol. 10, pp. 618-623.
Brown, P. J. (1969). A Survey of Macroprocessors in *Automatic Programming*, Vol. 6, Pergamon Press Ltd., Oxford.
Davies, K. E., and Paradine, C. (1969). *Extensions to the Macro Language*, IBM Ltd., Hursley, England.
Freeman, D. N. (1966). Macro Language Design for System/360, *IBM Systems Journal*, Vol. 5, pp. 63-77.
Halpern, M. I. (1967). *A Manual of the XPOP Programming System*, Lockheed Missiles and Space Company, California.
IBM (1966). *PL/1 Language Specifications* C28-6571, IBM Corporation, Poughkeepsie, New York.
Macleod, I. A. (1970). SP/1—A FORTRAN Integrated String Processor, *The Computer Journal*, Vol. 13, pp. 255-260.
Macleod, I. A. (1969). *An Information Processing Language*, Ph.D. thesis, Queen's University, Belfast.
Mandil, S. (1970). Ph.D. thesis, Queen's University, Belfast (in preparation).
McIlroy, M. D. (1960). Macro Instruction Extension of Compiler Languages, *CACM* Vol. 3, pp. 214-220.
Oppen, D. (1970). *A String Processing Extension of PL/1*, M.Sc. thesis, Queen's University, Kingston.
Strachey, C. (1965). A General Purpose Macro Generator, *The Computer Journal*, Vol. 8, pp. 255-251.
Waite, W. M. (1967). A Language Independent Macro Processor, *CACM* Vol. 10 pp. 433-440.
Waite, W. M. (1970). The Mobile Programming System: Stage 2, *CACM*, Vol. 13, pp. 415-421.