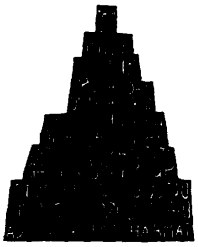

15. マクロ言語†

佐々　政孝††

1. はじめに

マクロ（マクロ言語・マクロ処理系）は，1950年代から，アセンブリ言語に見かけ上新しい命令を追加するために用いられていたが，その後その機能の一般性が注目され，高級プログラミング言語にも応用されるようになった．たとえば，幾つかの文列をひとまとめにすること，方言の吸収と標準化，条件付翻訳，記述性向上，虫取り，言語変換，言語拡張などである．マクロは一方，移植性のあるソフトウェア記述にも利用されてきた．

さて，マクロ言語は，入力テキストから出力テキストへの変換を記述するためのものである．変換の大部分は，マクロ定義の列で与えられる．マクロ定義は，パターン・マッチングの枠組を指定した「(マクロ・)パターン」部と，どのように変換するか（置換テキストとマクロ時機能）を指定した「マクロ・ボディ」部とからなる．マクロ言語で書かれたプログラムが他の汎用プログラミング言語で書かれたプログラムと異なる特徴的な点は，あるマクロが起動されるのが，入力テキスト中にそのマクロ定義のパターンに合致する部分がある時に限られることである．言いかえれば，マクロ・ボディには入力文はないのが普通である．この意味でマクロ処理は「入力駆動」であり，入力テキストは容赦のない流れとして入ってくるのである．

このような理解に基づき，今日では，初期のマクロ・アセンブラ等と比べて，概念や記法が格段に整備された高級言語風のマクロ言語が生まれてきた．

現在までの主要なマクロは，GPM[25]，構文マクロ[13),8),7)]，ML/I[1]，STAGE 2[29]，SIL[11]などである．最近では，これらの流れの上での拡張や強化のほかに，カーソルの概念に基づくマクロ処理機構の定式化(MAX[20])，マクロ機能のプログラミング言語中への埋め込み (SUPERMAC[5]) などの新しい流れも見られる．

マクロの目的は多岐にわたるが，これらは大きく(1)テキストの変換／言語の拡張(2)移植性のあるソフトウェアの記述，の2つに分類できる．マクロの諸機能は，主として(1)の分野のマクロの紹介によりカバーできるので，本解説ではこの分野に属するマクロを中心に述べることとする．

2. 代表的なマクロと最近の動向

2.1 GPM[25]

GPM は General Purpose Macrogenerator の略であり，ごく少数の概念に基づいて様々な処理が実現できることを示した点で興味深い．

GPM には3つの基本機構がある．マクロの呼出し，マクロ引数の挿入（置換テキスト中で仮引数を実引数で置き換えること），リテラル化機構（マクロ呼出しを抑制すること）である．まず，マクロ呼出しは，

§名前，引数1，引数2，……，引数n；

の形で行う．マクロ・ボディ内での引数の挿入は，「～1」，「～2」，……，「～n」のように記述する．リテラル化は，リテラルかっこと呼ぶ「<」と「>」でくくることによって実現される．マクロの定義は，「DEF」という名の組込みマクロを呼び出せばよい．たとえば，

§DEF, A, ⟨A～1 A⟩；

により，「A」というマクロは「A～1 A」という置換テキストを持つことが定義される．これにより次のような変換が行われる．左が入力，右が出力である．

§A, C；　　→ ACA
⟨§A, C；⟩　　→ §A, C；
§A, §A, C；；　→ AACAA

GPM の特徴は，マクロ・ボディには置換テキストしか書けないにもかかわらず，マクロ展開時に行いたい演算のほとんどがマクロ呼出しと置き換えの単純な

† Macro Languages by Masataka SASSA (Institute of Information Sciences and Electronics, University of Tsukuba)
†† 筑波大学・電子情報工学系

機能だけで実現できることを示した点にある．このようなものとして，次の「SUC」マクロは有名である．

```
§DEF, SUC, 〈§1,2,3,4,5,6,7,8,9,10,
                §DEF, 1, 〈〜〉〜1;;〉;
```

詳細は略すが，これは1桁の引数（1〜9）に1を加えた数字を出力する．たとえば「§ SUC, 7;」は「8」となる．また，条件付展開もマクロで行えることが示されている．

しかし，このような実現法は，オーバーヘッドが大きく，GPM は実用にはあまり用いられていない．

2.2 構文マクロ (syntax macros) の系列[7),8),13)]

GPM のように，マクロそれ自体でほとんどの機能を実現しようとするのとはまた異なるアプローチとして，もととなる言語（基底言語と呼ぶ）の構文を拡大することにより，言語拡張を行おうとするのが構文マクロである．この方式では，ユーザはまず，基底言語の BNF 記述を与える．この記述は厳密である必要はなく，マクロ定義で使用する構文記号が正しく構文解析できる程度の，核の部分を記述すればよい．その上で，拡張したい構文ごとに，それをどう基底言語に落とすかをマクロ定義として記述する．たとえば **for** 文を導入するには，次のようなマクロ定義を書く[13)]．

```
smacro for variable:=expression to expression
                                    do statement
define
  begin $1:=$2;
    L1: if $1≦$3 then
          begin $4; $1:=$1+1; go to L1 end
  end
endmacro
```

ここで，variable, expression, statement は，基底言語の構文記号である．$1, $2, $3, $4 は引数で，それぞれ variable, 1 番目，2 番目の expression, statement に対応する．

具現化の方法は，下降型の構文解析を行いながら，マクロの第1区切り記号（上例の **for**）が現れると解析を中断してマクロ処理に移る．その際，マクロ引数は，指定した構文記号に合うように構文解析して取り込む．

Leavenworth の構文マクロ[13)]は，拡張できる構文記号を〈文〉と〈関数〉だけに限っているなど種々の制限があったが，Campbell はそれらの制限の多くを取り除いたシステムを作成した[7)]．ただ，これらの構文マクロの限界として，新しいデータ構造や新しいインフィックス演算子の導入はかなり困難であることを述べておく．そのほか，文献[2),24)]を見られたい．

2.3 ML/I[1)]とその流れ

すべてをマクロ呼出しに頼る GPM と，構文解析に基づく構文マクロとを，考えられるアプローチのある極限とすると，ML/I はいわばその中間に位置し，マクロ本来の領分を守りつつ，その可能性を大きく広げた．

マクロ定義の一例を挙げよう．

```
MCDEF
  IF = THEN OPT ELSE FI OR FI ALL
       AS                〈マクロ・ボディ〉;
```

このマクロ定義のマクロ・パターン部は，

```
IF 引数1=引数2 THEN 引数3
                    (ELSE 引数4 FI|FI)
```

を意味し，ELSE 部が省略可能な IF 文を扱うものである．マクロ・パターン部には，このほかに繰返しも記述できる．

このように，ML/I で扱うことのできるマクロ呼出しは，マクロ名（上例では IF）で始まり，閉じ区切り記号の1つ（上例ではいずれも FI）で終るような形のものである．この制限はややきついが，扱う言語が Algol 68 や Ada など，閉じ区切り記号のあるものであれば，構文マクロにおけるように構文則のことを気にする必要がないので，実用上はこの方が楽である．

マクロ・ボディ内では，大域的・局所的マクロ変数，整数算術式と代入文，条件付飛越し文などを利用できる．これらはたとえば一意的なラベルを発生するのに必要である．ただ，記法がマクロ・アセンブラ式であるのが残念である．

マクロ呼出しを抑制するためには，GPM のリテラルかっこよりずっと強力な skip という仕組みを取り入れた．これにより，コメントの削除や文字列のコピーが行える．

ML/I の応用は数多いが，たとえば文献[26)]などがある．

ML/I の流れに属するものとして PM[22)]がある．PM では，マクロ・パターンの記述は正規表現に基づいており，マクロ・ボディ中の記法も Algol 風の読みやすいものとなっている．また，言語依存部分を扱えるように種々の指定が可能である．

2.4 SIL[11)]と移植用ソフトウェア記述言語

これまで述べてきたテキスト／言語の変換のほか

に，マクロの大きな利用分野は移植性のあるソフトウェアの記述である．その一例として，SIL (SNOBOL4 Implementation Language) について述べる．SIL は SNOBOL 4 (本誌の SNOBOL の項参照) の移植用処理系 (解釈実行方式) の記述用に設計された．SIL は，SILM と呼ぶ仮想計算機のアセンブリ言語に当たり，その文は 131 種類ある．移植者は，SIL の各文をマクロ・アセンブラで展開することによって対象機種上の機械命令に落とすことができる．一例が文献[23]にある．それによれば移植作業全体の労力は約3人月であったという．これは大きいようにも思われるが，SNOBOL 4 の複雑さから見れば，移植コストはむしろ少ないと考えられるべきである．Griswold によれば移植用システムにしたことによる非効率化率はおよそ 20% と見積もられている．SIL 言語は，大部分のマクロ・アセンブラで変換できるよう設計されているが，番地と整数とが区別されず同じ命令で扱われるなど，機械依存部分も残されている[23]．

マクロ処理を想定した移植用記述言語としては，このほかに，ML/I を記述した L や LOWL[2]，LIMP[28] を記述した WISP，STAGE 2[29] やそれを記述した FLUB などがある．BCPL を記述した OCODE をこの一種とみなしてマクロ処理した例もある[12]．その他，文献[2),3),15)]を参照されたい．

3. マクロの諸機能

3.1 マクロに対する要求

これまでの例からわかる通り，マクロを実用に供しようとすれば，単純な置換以上の種々の機能が要求される，その幾つかを以下に挙げよう．

(1) マクロ・パターンにおける省略 (option)，選択 (alternative)，繰返し (または任意個の引数)，また，引数をかっこに対してバランスさせること．

(2) マクロ・ボディにおける，省略，選択，繰返しのある引数の挿入．

(3) マクロ・ボディにおける，大域的・局所的マクロ時変数の導入 (整数型と，できれば文字列型)，マクロ時変数への代入，整数演算．

表-1 マクロ言語における諸機能の比較

マクロ言語	特徴・目的	パターン・マッチングと呼出し					マクロ時機能	動的マクロ定義 (注 4)	扱う言語	具現化と記述言語で注目すべき点
		形式 (prefix 型, template 型など) (注 1)	選択	繰返し	入れ子呼出	w/f (注 2)	c/t (注 3)			
GPM[25]	マクロ呼出しだけでマクロ時演算の機能を実現．	§ 名前，引数，……；固定形式	×	×	○	w	c	マクロ呼出しで実現	○	(GPM)	CPL で記述した処理系が論文25)に載っている．
TRAC[17]	対話型処理用，文字列読込みをプログラムで制御できる．	# (cl, 名前，引数，…) 固定形式	×	×		w	t	多少あり		TRAC	
構文マクロ[13),6)]	高級プログラム言語の構文を拡張．基底言語の BNF 記述を書いておく．	prefix．ほぼ LL(1)構文解析	○	×	○	f	t	弱い	×	任意	
ML/I[1]	汎用マクロの代表的なもの．call by name 可．	prefix	○	○	○	選べる	t	強力	○	任意	L や LOWL (ML/I に類似) で記述．
PL/I	PL/I プログラムの前処理．	PL/I の関数と手続きの形のみ	×	×		f	t	PL/I と同様	×	PL/I	
STAGE 2[29]	LIMP[28]の発展．移植用ソフトウェアの処理にも適する．I/O ファイルが扱えるので多重パスが可能．	template 行単位	○	×	×	f	c	強力	×	任意	FLUB という仮想機械の言葉で記述．移植性良．
MP/1[14]	Fortran 用．	template 行単位	×	○	○	w	c	Fortran と同様	×	Fortran	
MAX[26]	マーカー (カーソル) を考えて，マクロ評価法を定式化．	α□a→ □β(注 5)	×	×	×	f	t	(他の機能で代用)	○	任意	
改良構文マクロ[7]	任意の構文記号を拡張できるようにした．マクロ時演算中で属性を利用．	LL(1) 構文解析．その範囲で infix．	○	○	○	f	t	強力	×	任意	コード木のレベルで処理．
PM[22]	パターンは正規表現．言語依存部分を扱える．	パターンは正規表現．prefix	○	○	○	f	t	強力	○	任意	PM マクロで拡張された Algol 68 風 Fortran で記述．
SUPERMAC[3]	指定した入力ファイルから指定した出力ファイルへのマクロ展開．多重パス可．		×	○		f	t	基底言語と同様	△	BASIC BCPL	プログラム言語の実行時ルーチンとして組込む．

空白の所は不明

注 1. prefix 型：マクロ名ではじまる．infix なマクロを扱うのが難しい．template 型：行ごとにマッチングさせる．行の概念に依存．

注 2. マクロ呼出しに警告記号が必要か(w)，自由モードか(f)．

注 3. 文字単位(c)かトークン単位(t)か．

注 4. マクロ定義をインタプリトするかコンパイルするかに関係が深い．

注 5. □はマーカー，α と β はトークンの列，a はトークン．

（4） マクロ時変数や引数に対する条件文，マクロ時のループ機構．

（5） 入れ子構造と再帰性．たとえばマクロ呼出しの引数中でのマクロ呼出し．

（6） 入力テキストの一部でマクロ呼出しを抑制する機能．リテラルかっこのほか，たとえばコメント，文字列の扱い．

以上のほか，詳しい議論が文献[2]にある．

3.2 マクロ言語における諸機能の比較

表-1 を見て頂きたい．この表から，これまで述べたマクロの諸機能のほか，マクロのデザインチョイスを読み取って頂ければ幸いである．表にないマクロについては文献[2),6),9),10),15),27),28)]を参照されたい．

4. 今後の方向

マクロは古くからプログラミング言語やコンパイラとの関係が深い．初期のマクロの目的であった，「アセンブリ言語を読み書きしやすくする」という方向は，今日のシステム記述言語につながっている．同様に，マクロによって導入されることの多かった新しい制御構造も，どしどしプログラミング言語に取り入れられてきている．これではマクロはいずれ不要になると思われるかもしれない．しかし，言語や命令をユーザ独自のものに合わせたり，拡張したり，1つのシステムから種々の版を発生したり，テキストを系統的に編集したりするために，マクロに対する需要は依然として続くであろう．これらのためにも，種々の応用にこたえられる標準的な汎用マクロの出現が待たれる．

今後の研究の方向と，その萌芽について，紙数も尽きたので箇条書きに挙げる．

（1） コンパイラなどと比較したマクロの能力と限界についての議論[2),22)]，マクロ処理機構の定式化（たとえば MAX[20)]），GPM よりはもっと実用に近いレベルでの直交設計（少数の基本的機構で多くの要求を満たすこと）．

（2） PL/I では必ずしも成功しなかった，マクロのプログラミング言語への埋め込み（たとえば SUPERMAC[4),5)]）．

（3） これまでの成果を集大成し，種々の応用にこたえられる標準的な汎用マクロの設計と普及．

（4） 拡張言語との係わりの追究[2),18),19),24)]．

（5） 伝統的な利用法を越えた広い分野への応用（コンパイラ・コンパイラ[26)]，属性文法[21)]など[16)]）．

参 考 文 献

参考文献は数多いが，少し以前のものまではブラウンの本[2)]によくまとめてある．概説書は 2),6),9) である．

1) Brown, P. J.: The ML/I macro processor, Comm. ACM, Vol. 10, No. 10, pp. 618-623 (1967).
2) Brown, P. J.: *Macro processors and techniques for portable software*, Jown Wiley (1974). 和訳：鳥居(他)：マクロ・プロセサとソフトウェアの移植性，近代科学社．
3) Brown, P. J. (Ed.): *Software portability-an advanced course*, Cambridge Univ. Press (1977).
4) Brown, P. J.: Macros without tears, Software-Practice and Experience, No. 9, pp. 433-437 (1979).
5) Brown, P. J.: SUPERMAC-A macro facility that can be added to existing compilers, Software-Practice and Experience, No. 10, pp. 431-434 (1980).
6) Campbell-Kelly, M.: *An introduction to macros*, Macdonald・London & American Elsevier (1973).
7) Campbell, W. R.: A compiler definition facility based on the syntactic macro, Computer Journal, Vol. 21, No. 1, pp. 35-41 (1978).
8) Cheetham, T. E.: The introduction of definitional facilities into higher level programming languages, AFIPS Conference Proceedings (FJCC), No. 29, pp. 623-637 (1966).
9) Cole, A. J.: *Macro Processors*, Cambridge Univ. Press (1976).
10) Comer, D.: MAP: A Pascal macro preprocessor for large program development, Software-Practice and Experience, No. 9, pp. 203-209 (1979).
11) Griswold, R. E.: *The macro implementation of SNOBOL 4*, Freeman (1972).
12) 林 恒俊：目的コード生成過程記述のためのソフトウェア・トゥール及びその適用，第 22 回プログラミング・シンポジウム報告集，pp. 141-150 (1981).
13) Leavenworth, B. M.: Syntax macros and extended translation, Comm. ACM, No. 9, pp. 790-793 (1966).
14) Macleod, J. A.: MP/1-a FORTRAN macroprocessor, Computer Journal, Vol. 14, No. 3, pp. 229-231 (1971).
15) Metzner, J. R.: A graded bibliography on macro systems and extensible languages, SIGPLAN Notices, Vol. 14, No. 1, pp. 57-68 (1979).
16) Middleton, A. G.: On the use of macros for

iteration, Computer Journal, Vol. 19, No. 2, pp. 170-172 (1976).
17) Mooers, C. N.: TRAC, procedure describing language for the reactive typewriter, Comm. ACM, Vol. 9, No. 3, pp. 215-219 (1966).
18) Nagata, H.: FORMAL: A language with a macro-oriented extension facility, Computer Languages, Vol. 5, No. 2, pp. 65-76 (1980).
19) Napper, R. B. E. and Fisher, R. N.: ALEC-A user extensible scientific programming language, Computer Journal, Vol. 19, No. 1, pp. 25-31 (1976).
20) Nudds, D.: The design of the MAX macro-processor, Computer Journal, Vol. 20, No. 1, pp. 30-36 (1977).
21) Papakonstantinou, G.: A poor man's realization of attribute grammars, Software-Practice and Experience, No. 9, pp. 719-728 (1979).
22) Sassa, M.: A pattern matching macro processor, Software-Practice and Experience, No. 9, pp. 439-456 (1979).
23) 白濱律雄：SNOBOL 4 処理系の移植経験，情報処理学会計算言語学研究会 CL 9-3 (1977).
24) Solntseff, N. and Yezerski, A.: A survey of extensible programming languages, *Annual Review in Automatic Programming*, No. 7, Part 5 (1974).
25) Strachey, C.: A general purpose macrogenerator, Computer Journal, Vol. 8, No. 3, pp. 225-241 (1965).
26) Tanenbaum, A. S.: A general-purpose macro processor as a poor man's compiler-compiler, IEEE Trans. SE, SE-2, No. 2, pp. 121-125 (1976).
27) Teskey, N.: KATE: A macro-processor for extending command languages, Computer Journal, Vol. 20, No. 2, pp. 187-189 (1977).
28) Waite, W. M.: A language-independent macro processor, Comm. ACM, Vol. 10, No. 7, pp. 433-440 (1967).
29) Waite, W. M.: The mobile programming system: STAGE 2, Comm. ACM, Vol 13, No. 7, pp. 415-421 (1970).

(昭和56年2月12日受付)