テキストマクロプロセッサMacr055

岸本 誠[a)]

概要：筆者が設計・実装し公開・メンテナンスしている，汎用テキストマクロ処理系 Macr055 について紹介し，テキストマクロの技法とともに，それらの技法に関係するマクロプロセッサの仕様に関する検討について述べる．たとえば，条件付き展開など一見では処理系内蔵でなければ実現できなさそうな機能について，マクロ処理系の一見些細な仕様の違いにより，通常のマクロとして実現可能であったり不可能であったりする（一例としては，*Software Tools*（『ソフトウェア作法』）で紹介されているコードのままの手順ではダメな点があったため仕様を変更している．後で確認したところ，著名な M4 でも筆者によるものと同じような仕様となっていた）．そのため，些細な仕様でも「良い設計」を詰めてゆく必要を感じている．

キーワード：文字ベースマクロプロセッサ，AWK

1. はじめに

筆者は 3 年ほど前から，文字ベースのテキストマクロプロセッサ Macr055（`m55`）を公開し，メンテナンスしている．当初（公開以前）は著名なプロセッサである M4 を参考に，定番書である『ソフトウェア作法』（の，第 8 章）に紹介されている通りの手順によるマクロ処理をほぼそのまま，プログラミング言語を RATFOR から AWK に書き換えたようなものであった．しかしその後 *A General Purpose Macrogenerator* (C. Strachey)[1] を読み，また一昨年の当シンポジウムでの発表「GPM とそのプログラム」と引き続くブログ記事にあった，多くのマクロプロセッサ向け練習問題で動作確認を行ったことにより，実装・マクロプロセッサとしての仕様ともに洗練されたと考えている．

以下，まずテキストマクロプロセッサの分類について説明し（特に，プログラマが最も良く知っている C プリプロセッサとは重要な違いがある），次に AWK 言語による Macr055 の具体的な実装を説明する．その後に，条件付き展開等を題材に，マクロプロセッサのどのような仕様の違いが，マクロの技法に影響するのかを議論する．最後に，マクロプロセッサの今後の展望を述べる．

2. マクロプロセッサの分類

「マクロ」という語が一般的な語であることもあり，ソフトウェアに関する文脈でも多くの場合にマクロという語が使われることから，時折誤解も見られるので，すこし広い分類から見てゆく．

- マクロ言語: アプリケーションソフトの制御スクリプト，4GL や近年では DSL の類など，目的に特化した「プログラミング言語より粗粒度の（マクロな）」記述をする言語
- キーボードマクロ等の類: これは（ユーザが触れるような）言語は持たないこともある
- 変換システムとしてのマクロ: ここで議論す

[a)] ksmakoto@dd.iij4u.or.jp

るテキストフィルタや，Lisp のマクロなど

最後の変換システムとしてのマクロは，次のように分類できる．

- テキスト以外のマクロ: Lisp のマクロ（ S 式 → S 式 ），など
- テキストマクロ (言語・プロセッサ): テキスト（ファイルやストリーム）を入力として，それに何らかの変換をほどこし，テキストを出力するテキストフィルタとして実装される．M4 や C プリプロセッサなど

以下，本稿では「テキストマクロ」のみが議論の対象である．

そして，あまり知られていないが，テキストマクロには以下の 2 種類の分類がある．

- トークンベース: C プリプロセッサ
- 文字ベース: M4（「テキストベース」とも [*1]）

C プリプロセッサ（以下 CPP）は，プリプロセッサ・トークンという用語もあるように（ある程度はコンパイラ本体の lex との違いを明示するための語だが），入力はまずトークンにされ，変換処理本体はトークン列を相手に変換をおこなう．なおそれだけでなく，コメントアウトが効くことや文字列リテラルへの対応などを見てもわかるように，CPP の仕様は「プリプロセスしなくても，見た目が C 言語のようである」入力を前提としている．（アセンブリ言語の）ソースファイルの拡張子が，小文字でなく大文字の `.S` の場合にはコンパイラドライバは CPP を通すなど，目的外使用 [*2] はしばしば見られる．しかし，lexical syntax 的に C 言語と違うような入力を CPP に処理させることには落とし穴がある（詳細は「高品質の C プリプロセッサ mcpp」[3] § 6.2.1）．

一方の文字ベースマクロは，変換処理本体が直接に文字を相手にする．文字ベースらしさが現れる例としては「マクロに展開されるマクロ」のような，見掛けでは再帰的に展開が行われるような処理を，マクロを展開した結果があたかも入力に存在したものであるかのように押し戻す「プッシュ・バック」[*3] が挙げられる．これにより，再帰的に見える処理が単なる繰返しで行われる．ここで，プッシュ・バックによる（とモデル化すると，わかりやすく単純である）M4 での現象を示す．

M4 は次のような入力に対し，

```
changequote(‘[’, ‘]’)dnl
define([foo], [bar])dnl
define([barbaz], [[quuz]])dnl
foo()baz
```

次のように出力する．

```
quuz
```

入力中にある「`foo()`」が「`bar`」に置き換えられ，それが入力にプッシュ・バックされて，そこから続きの処理が始まるので，「`barbaz`」となって別のマクロ展開が起きている．

トークンベースで処理するには，CPP のように何らかの目的があって，その目的によりトークンがレキシカルに明確でなければならない．このことから「汎用」のマクロプロセッサは，文字ベースであるか，トークンベースであれば [*4]，トークンのレキシカルな定義がプログラマブルでなければならない，と言える．[*5]

なお，CPP が現在のような仕様に明確化されたのは C 言語の標準化の時であり，以前の実装は文字ベース寄りであった（そのため，C コンパイラ

[*1] 文献「高品質の C プリプロセッサ mcpp」中でも表記揺れがある．他にも「キャラクタベース」「ストリングベース」「文字列ベース」などの表現がありえそうである（実際に Google による英語の検索結果ではこの話題に関して，character-based, string-based, text-based の，いずれの表現も見られる）．

[*2] レンチをハンマー代わりに使うなど，以前は「ライフハック」的に気楽に行われた「工具の目的外使用」であるが，近年では機械工場などでは厳しく戒められるようになっている．

[*3] 『ソフトウェア作法』ではそう呼んでいる．

[*4] コマンドライン引数や設定ファイル等，あるいは（架空のマクロ言語であるが）
`deftoken(/[A-Z_a-z][0-9A-Z_a-z]*/)`
といったようにして．

[*5] 同様な点で，マクロ処理の本質ではなく実装の詳細ではあるが，実用上は重要な点として，マクロの引数の区切り文字などといったものも，汎用であるためには（理想的には全て）プログラマブルである必要がある．Macr055 では，一部に頓知のような仕様もあるが（文字コード順で前後の文字に意味を持たせる，等），できる限りプログラマブルに実装している．これは実装のソースコードを見ると処理の振り分けの判定などの部分には，いわゆる「マジックナンバー」的な文字列リテラルなどが現れてはならず，そのような場所は必ず，組込みマクロ等によってユーザから変更可能なグローバル変数でなければならない，ということを意味する．

と別に従来の CPP があるシステムや，互換モードを持つ CPP があるだけでなく，仕様にも名残がある [3]）．一方の M4 も（用途として CPP を強く想定していたためか？）マクロ名の識別や引数における空白の無視など，トークンベース風の所がある．既存のマクロ処理系についての議論はここまでとする．

3. Macr055 の設計と実装

3.1 全体の解説

まず Macr055 という名前について説明する．M4（および M3）という既存の著名な処理系があることから，M5 という略称になる名前などを当初は考えていた．しかし仕様が固まるに従い，M4 の直接の代替にはならないようなものになったため，少し違う名前としてコマンド名が `m55` になるということから，「Macross」[6] を leet 表記 [*6] 風に捻った名前とした．

AWK（New AWK, `nawk`）で実装しており，分量は 2016 年末の時点で 500 行ちょっとである．GNU AWK（`gawk`）の拡張などは使っていない一方，ユーザ定義関数など New AWK の機能は積極的に使っている．M4 の組込みマクロ `eval` に相当する算術式評価の機能があるが，`bc` コマンドにそのまま投げて受け取るだけという実装にしてある．そういった「大物」（現在の規模に対する相対として）については今後も可能な限り同様にするつもりである．

文字ベースのマクロプロセッサの教科書的な設計例は，『ソフトウェア作法』の第 8 章で詳解されている．基本的にはそれに沿った設計として実装した．まず，AWK 特有のいくつかのノウハウがあるのでそれを紹介する．

(1) 一般に「モジュール」などと呼ばれるような，名前空間を分離する機能が AWK には無い: まずグローバル変数の分離が問題であり，入出力のレイヤ化を飛び越えて関数を呼んでいないかなど，大きなプログラムでは関数も問題である．機能を分離すると同時に名前を付け，それをプレフィックスとして付けることで擬似的な名前空間とした．

(2) ローカル変数が無い: AWK は実引数の数が仮引数と同じかチェックしないので，「余分な仮引数」としてローカル変数に相当するものを宣言する．これは AWK では一般的である．

(3) サブプロセスとの双方向通信ができない: bc コマンドに計算をさせて結果を得る方法として，毎回コマンドとして実行する方法と，パイプで標準入出力からやりとりする方法がある．Unix は，プロセスの生成と終了が比較的重いシステムであるから，算術計算はベンチマーク等で多用するし後者を選びたいが，標準的な AWK では不可能である．そのため名前付きパイプをスクリプトの外で作成し，それを経由している．なお GNU AWK では拡張されていて，`|&`という記号によるパイプで可能である（マニュアルの Two-Way Communications with Another Process という節にある）．

(4) 複合型（データ構造）が連想配列（map）しかなく，それを入子にもできない: マクロプロセッサでは必要なデータ構造は環境フレーム程度などなのでそんなに苦労しないが，とにかくなんとかしてマップするしかない．

(5) エラーハンドリング: そこまで問題になるようだと諦めて Python 等に切り換えるべきか？

ここで用語「ウォーニング・キャラクタ」について説明する．ストレイチーが用いたフレーズ [1] で，入力にそのような文字が存在したら，単に素通しして出力するのではなく，何らかのマクロ展開処理に関係する対応が必要な「要注意」な文字，という意味と思われる．マクロの開始・終了，実引数の区切などである．マクロ内でないのにマクロ終了文字があった，といった場合にどう扱うべきか，といったことを考慮すると，文脈に依存するものと考えられるかもしれない．Macr055 では，マクロ外のマクロ終了文字などは単に無視してそのまま出力している．

コードの説明に入る前に，文字ベースによるマクロ展開のおおまかな流れを説明する．マクロプロセッサは，マクロ外では単に素通しのテキストフィルタとして動作する．マクロ内では，出力はそのまま出力されず，内部のバッファに保存される．

[*6] leet を 1337 とするなど，アルファベットから形状の似た数字への置き換え．

そして，引数の区切文字が現れる度に args[i] のような番号で取り出すことのできる連想配列に保存する．マクロ終了の文字が現れたら，args[0] にある（はずの）マクロ名にもとづいて，そのマクロに定義されている展開に従って引数の置換などを行って展開し，プッシュ・バックする．

続いて，ソースコードの行番号を示しながら追ってゆく（ソースコードは GitHub で公開しているので，こちらの紙面では割愛する）．

メインループは 67 行目の for (;;) {から，133 行目の}までである．入力として 1 文字も読めなかった場合，131 行目の break でメインループから脱出する．メインループからの脱出時に，「マクロの中にいる」ようであればエラーとして異常終了とし，そうでなければ正常終了する．他に異常終了としては，マクロ処理を回避するエスケープであるクウォートやコメント中で入力が尽きた，という場合や，マクロ展開に関係する異常などがある．

メインループ内のコードを見てゆく．まず最初に，配列のキーにウォーニング・キャラクタを入れる初期化をしている．これは，「入力の，今読んでる位置の先頭がこれらの文字であったら，それを返せ」という関数 m55_input_get に渡すためのもので，ループの先頭で毎回作り直しているのは，組込マクロで変更された場合に対応するためである．

最初の，77 行目から 103 行目までの if 文は，マクロの開始とクウォートの開始のウォーニング・キャラクタへの対応である．マクロの開始では，環境に enter する関数を呼んで，環境に新しいフレームを作る．フレーム内には，マクロへの引数（マクロ自身の名前も，0 番目の引数として同様に扱われる）が，区切が現れるごとに保存される．

クウォートのほうの処理では，「クウォートはネストする」「クウォート内はあらゆるものをそのまま通す」という仕様であるため，内部でループして，対応する「クウォートの終了」が現れるまで入力を読んでそのまま出力している．

続いて 104 行目から 111 行目の if 文は，マクロ中での処理となっている．マクロ中では，引数の区切とマクロ終了のウォーニング・キャラクタに対応した処理が必要である．区切の文字が現れたらそこまでを引数として保存する．マクロ終了の文字が現れたら，マクロを展開し，フレームをひとつ畳む．

メインに続いて 142 行目からある m55_leave_expand が，マクロ展開をおこなう関数である．前半は，AWK ではプラグイン的なものを実装する方法が無いために，組込マクロに関する処理への分岐が羅列してある．後半がマクロ展開の本体で，コードの分量としては引数展開の特殊な場合（引数を連結して展開，引数の個数への展開）などへの対応がほとんどである．

残りはほとんどがハウスキーピング的な関数である．分割を徹底しているので，見通しは少々悪いかもしれない．

3.2 組込マクロの追加方法

続いて，ユーザの実用の観点からは必要性が高いであろう，組込マクロの追加方法を説明することで，Macr055 のコードをミクロな観点から見てゆく．

2016 年 1 月に，組込の ifelse マクロを削除したので，その際の逆差分のパッチを順番に示してゆく．（※コミットの都合で他の差分と一緒になっていたのでそれを除いてある．また，レイアウトの都合でスペーシングなども編集してある）

```
diff --git b/m55.awk a/m55.awk
index 171cff6..868e028 100644
--- b/m55.awk
+++ a/m55.awk
@@ -28,9 +27,9 @@ function m55_main
         M55_DEFTYPE = "df"
         M55_VALTYPE = "vl"
         M55_DNLTYPE = "dn"
+        M55_IFETYPE = "ie"
         M55_ESYTYPE = "es"
         M55_EXPTYPE = "ex"

         M55_CHQTYPE = "cq"
         M55_CHCTYPE = "cc"
```

タグのようなものを決めておく．マクロプロ

セッサの実装で，このようなものが絶対に必要というわけでもないが，『ソフトウェア作法』での手法に倣った．ユニークであることと，先頭の 1 文字が空白（正確には M55_USRTYPE というグローバル変数で決めている）ではないこと，という制限がある．

```
@@ -53,9 +52,9 @@ function m55_main
  m55_macroenv_def("m55_define", M55_DEFTYPE)
  m55_macroenv_def("m55_val", M55_VALTYPE)
  m55_macroenv_def("m55_dnl", M55_DNLTYPE)
+ m55_macroenv_def("m55_ifelse", M55_IFETYPE)
  m55_macroenv_def("m55_esyscmd", M55_ESYTYPE)
  m55_macroenv_def("m55_expr", M55_EXPTYPE)
-
  m55_macroenv_def("m55_changequote", M55_CHQTYPE)
  m55_macroenv_def("m55_changecom", M55_CHCTYPE)
```

シンボルテーブルに，マクロ名が前述のタグに解決されるようなエントリを追加する．

```
@@ -157,12 +156,12 @@
    m55_do_define(args)
  } else if (body == M55_DNLTYPE) {
    m55_do_dnl(args)
+ } else if (body == M55_IFETYPE) {
+   m55_do_ifelse(args)
  } else if (body == M55_ESYTYPE) {
    m55_do_esyscmd(args)
  } else if (body == M55_EXPTYPE) {
    m55_do_expr(args)
  } else if (body == M55_VALTYPE) {
    macroname = args[1]
    if (macroname in locals) {
```

マクロ名を解決した結果が前述のタグであれば，対応する処理をするよう，振り分けに追加する．ごく短ければ直接書いてしまってもよいかもしれないが，一般に別の関数にしたほうが無難だろう．

```
@@ -233,6 +227,21 @@ function m55_do_dnl
    } while ((c != "") && (c != "\n"))
 }

+function m55_do_ifelse(args,  base) {
+ base = 1
+ for (;;) {
+   if (args[base+0] == args[base+1]) {
+     m55_input_pushback(args[base+2])
+     return
+   }
+   if ( ! ((base+4) in args)) {
+     m55_input_pushback(args[base+3])
+     return
+   }
+   base += 3
+ }
+}
+
 function m55_do_esyscmd(args,  cmd, result) {
    cmd = args[1]
    cmd | getline
```

組込マクロの展開処理の本体．M4 の ifelse と同様の仕様である．

4. マクロの技法とプロセッサの仕様

4.1 技法 (1) if

Macr055 の構文は，デフォルトでは次のようなものになっている．

```
{macroname|arg1|arg2|arg3}
```

空白類などの特別扱い等も一切無い．C プリプロセッサや M4 の，名前を基にその名前のマクロが無ければそのままとするという仕様は，C 言語のソースコードなどで関数名をそのままマクロ名として定義して，マクロ展開によりコードをすり替える，といった場合に便利である．しかし，ちょっとした名前が意図せずマクロによって変えられてしまうかもしれない（CPP における unix とか）というような深刻な副作用もある．それに対し，このように，マクロの開始が明示される構文は，マクロが無ければエラーになることを示唆するもののように思われるので，そのような仕様としている．（これに関する決定は結局，目的次第なのかもしれない）

2 個の引数が同じか異なるか，によって展開結果を変えるようなマクロ ifelse を定義して使用する例を示す．

```
{m55_define|ifelse|‘{$1|
  {m55_define|$1|$4}
  {m55_define|$2|$3}}’}{m55_dnl}
{ifelse|foo|foo|equal|not-equal}
{ifelse|foo|bar|equal|not-equal}
```

m55 からの出力は以下のようになる．

```
equal
not-equal
```

『ソフトウェア作法』で示されている手順に従っているプロセッサでは，実はこの展開はできない [*7]．なぜかというと，マクロの先頭にあるマクロ名（かもしれないもの）を読み込んだ時点で，そのようなマクロが存在するかどうかのチェックとしてマクロの展開結果そのものを取得してしまうのである．この技法では，それに引き続く引数内にあるマクロ定義によって，その展開結果を定義しているわけであるから，それではうまくない．元々は Macr055 もそのような仕様であったが，これをうまく扱えるように仕様を変更した．

M4 では実装によって，この技法がうまく使えないものもあるようだが（GNU M4 ではできなかった），FreeBSD の/usr/bin/m4 にある M4 では，次のようにダミーを入れてやればうまくいく．

```
define(‘eq’, ‘define($1, ‘‘dummy’’)$1(
  define(‘$1’, ‘$3’)define(‘$2’, ‘$4’))’)dnl
eq(‘foo’, ‘foo’, ‘equal’, ‘not-equal’)
eq(‘foo’, ‘bar’, ‘equal’, ‘not-equal’)
```

出力結果は同じなので省略する．

4.2 技法 (2) 1 桁の数の計算

ストレイチーの GPM での解法がある [4] ので，それが Macr055 ではどのようになるのかを示す．次のようになる．マクロ名に使える文字に制限は無いので，1 とか -1 という名前が使えている．

```
{m55_define|1+|‘{1|2|3|4|5|6|7|8|9|10|
  {m55_define|1|$$$1}}’}{m55_dnl}
{m55_define|1-|‘{-1|0|1|2|3|4|5|6|7|8|
  {m55_define|-1|$$$1}}’}{m55_dnl}
{m55_define|+|‘{$1|
  {m55_define|$1|‘{1+|{+|{1-|$1}|$2}}’}
  {m55_define|0|$2}}’}{m55_dnl}
{m55_define|-|‘{$2|
  {m55_define|$2|‘{-|{1-|$1}|{1-|$2}}’}
  {m55_define|0|$1}}’}{m55_dnl}
{m55_define|*|‘{$1|
  {m55_define|$1|‘{+|{*|{1-|$1}|$2}|$2}’}
  {m55_define|0|0}}’}{m55_dnl}
{m55_define|lt|‘{$1|
  {m55_define|$1|‘{p|$1|$2|{m55_define|p|‘{lt|{1-|$1}|$2}’}}
  {m55_define|-1|t}
  {m55_define|$2|f}}’}{m55_dnl}
{m55_define|r|‘{{lt|$1|$2}|
  {m55_define|t|$1}
  {m55_define|f|‘{r|{-|$1|$2}|$2}’}}’}{m55_dnl}
```

この例では，クウォートによるエスケープと引数の置換に関する振舞が GPM と違うらしいことがあきらかになった．Macr055（に限らず，M4 など *Software Tools* の影響下にあるマクロプロセッサ）は，どれだけクウォートが掛かっていようと，その中にある引数の置換は行なってしまう．そのため，$に置換される特殊置換パターンの$$を使っている．これに関しては Macr055 を変更するつもりは（今のところ）無い．その他の練習問題についても，結果をブログで公開している?．エイトクイーンには 3 時間ほどを要した．

5. 展望

正直なところ機動力といった点では，Python, Ruby, Perl といったスクリプティング言語や erb のようなそれらの言語の埋込みテンプレートシステムを，対応したい問題に応じて使ったほうが良いであろう．

それに対し，テキストマクロプロセッサは，M4 があまりにも代表的であったために，M4 の構文がうまく適応しない分野では最初から諦められていた，といった面があったのではないかと思う．その点で汎用テキストマクロは，広い「ブルーオー

[*7] マクロ定義のスコープの問題もあるが，ここではそれには触れない．

シャン」ではないかもしれないけども，競合が少なくまた従来省みられなかった「ブルーポンド」ぐらいの分野かもしれない．

また M4 は，枯れているツールではあるが，Autotools のように酷使しているツールもある．なお Autotools は GNU M4 を使うよう指定している．その GNU M4 は将来のバージョンへの言及が以前よりドキュメントのあちこちに見られるものの，リポジトリにはここ 2 年ほど動きが無いように見える．

6. まとめ

筆者が設計・開発しメンテナンスしている文字ベースのテキストマクロプロセッサ Macr055 の設計と実装について紹介し，テキストマクロの技法がマクロプロセッサのちょっとした振舞の違いに影響を受ける例などについて議論した．既存のよく広まっているツール（M4 等）にとらわれず，仕様を検討する必要といったことも示せたと思う．

参考文献

[1] C. Strachey: **A General Purpose Macrogenerator**. *Computer Journal* 8, 3 (1965), 225-41.

[2] B. W. Kernighan, P. J. Plauger, 木村泉訳: **ソフトウェア作法**. 共立出版, 1981.

[3] 松井潔: **高品質の C プリプロセッサ mcpp**. 2008. (http://prdownloads.sourceforge.net/mcpp/mcpp-summary-272-jp.pdf)

[4] 和田英一: **パラメトロン計算機: Christopher Strachey の GPM**. 2015. (http://parametron.blogspot.jp/2015/01/cristopher-stracheygpm.html)

[5] 岸本誠: **ksmakoto の hatenadiary**. 2016. (http://ksmakoto.hatenadiary.com/entry/2016/02/17/191433 他)

[6] きだあきら: **LSI C-86 用改造版 cpp 解説書，あるいは与太話**. 1994. (「LSI C-86 用改造版 cpp」添付文書)

[7] スタジオぬえ, タツノコプロ, 他: **超時空要塞マクロス**. 毎日放送他, 1982-.