2B-06

C言語マクロを前置型の作用素のように使うことについて

中村博文† 渕田孝康‡
都城高専† 鹿児島大学‡

1 あらまし

C 言語 [1]（や C++言語 [2]）で記述されたソースコードの一定の状況において，マクロ機能を用いて，実行の制御が前置型の作用素のように簡潔に記述できることについてささやかな提案をする．具体的な利用場面は，(a) 反復の指定と，(b) 実行・非実行の指定である．(a) は，マクロへの適切な命名によって簡潔で分かりやすい記述に、(b) は，試行錯誤に関わるキー操作の低減につながる．(b) はコンパイルチェックがなされる。

2 作用素のような反復指定の記述

2.1 具体例

反復指定を作用素のように記述できる場合がある．

具体例として，まず，次の，掃き出し法による逆行列計算のコードへの適用例を挙げる．

```
for(i=0; i<N; i++)
  for(j=0; j<N; j++)
     inv[i][j] = ((i==j)? 1.0: 0.0);
for(i=0; i<N; i++){
  w = 1.0/a[i][i];
  for(j=i+1; j<N; j++) a[i][j] *= w;
  for(j=0; j<N; j++) inv[i][j] *= w;
  for(j=0; j<N; j++){
    if(i!=j){
      w = a[j][i];
      for(k=i+1; k<N; k++)
        a[j][k] -= a[i][k]*w;
      for(k=0; k<N; k++)
        inv[j][k] -= inv[i][k]*w;
    }/* if */
  }/* j */
}/* i */
```

このコードのようなアルゴリズムに対して，

```
#define  F(v)  for(v=0; v<N; v++)
#define  Fi  F(i)
#define  Fj  F(j)
#define  Fk  F(k)
```

のようなマクロを定義[1]しておくなら，

```
Fi Fj inv[i][j] = ((i==j)? 1.0: 0.0);
Fi{
  w = 1.0/a[i][i];
  for(j=i+1; j<N; j++) a[i][j] *= w;
  Fj inv[i][j] *= w;
  Fj{
    if(i!=j){
      w = a[j][i];
      for(k=i+1; k<N; k++)
        a[j][k] -= a[i][k]*w;
      Fk inv[j][k] -= inv[i][k]*w;
    }/* if */
  }/* Fj */
}/* Fi */
```

と記述できる[2]．もし更に，

```
#define  Fij  Fi Fj
```

のようなマクロも用意した場合には，冒頭部分を

```
Fij inv[i][j] = ((i==j)? 1.0: 0.0);
```

のようにも記述できる．

2種の繰り返しがある場合に、例としてもしAで始まる名前を使うなら、

```
#define  AvV(v,V)  for(v=0; v<V; v++)
#define  AiN  AvV(i,N)
#define  AjM  AvV(j,M)
```

のようなマクロを定義して用いればよい．

次の例は，線形リストをたどる繰返しである．

```
#define  Trace(v, data)  \
  for(v=HEAD; v!=NULL; v=data[v].next)
#define  Tp  Trace(p, List)
```

のようなマクロを定義しておくと，

```
sum=0;  Tp sum += data[p].value;
```

のように記述できる．

2.2 既存の類似例

文献 [3] に後置も要する例[3]と汎用的な例[4]がある。

Usage of the Programming Language C's Macro such as Prefix Operator

†General Education Division, National Institute of Technology, Miyakonojo College
‡Graduate School of Science and Engineering, National University Corporation Kagoshima University

1制御変数を反復構文の中で定義するかどうかは任意である．

2参考まで述べておくと，例えば，配列 a[i][i] を a(i,j) や aij や Aij のように記述できるようマクロ定義しておくと，コードの記述がより簡潔になり，また数学的表現に近づく．また，デバッグ時に，配列添字等の値の範囲チェックをしたい場合に，コードの随所に記述するのではなく，例えば，#define a(i,j) a[(0<=(i)&&(i)<MAX)?(i):(printf("aij i:%d\n"(i)), OtherProcessing, (i))][…] のように，マクロ定義に盛り込めば記述を一箇所で済ませられる．

3#define loop(n) {int _i_; for(_i_=1;_i_<=(n);_i_ ++) { 及び #define lend }} 。

4#define forto(i,from,to) for(i=(from);i<=(to);i++)　及び #define downto(i,from,to) for(i=(from);i>=(to);i--) 。

Linux コード [4] でフルスペルの命名の例[5]がある。

3 コードの実行・非実行指定の作用素のような記述

3.1 1文ごとの実行・非実行の指定

通常，/*と*/で囲んで，例えば

```
proc_a(); /*proc_b();*/ proc_c();
```

のようにコメント化しているのを，

```
#define  D  if(0)
```

のような定義のマクロを用意しておくと

```
proc_a(); D proc_b(); proc_c();
```

のように記述できる[6]．

3.2 複数文や複数行の実行・非実行の指定

Dのようなマクロを，1行内でも複数行内でも，複数の文に対して適用したい場合には，{と}で囲んで

```
D{
 :
}
```

のように使用する．

3.3 実行と非実行の切り替え

コードの実行と非実行の反転は，Dという文字の入力や削除でも可能であるが，そうではなくて，

```
#define  DD  if(1)
```

または，空列にマクロ展開させる

```
#define  DD
```

のようなマクロも用意しておくと，反転が比較的頻繁な場合でも挿入や削除の字数が常に1字で済む．

勿論，すべてのDを実行化したい場合には，コード中のDをすべて削除するのではなく，Dの定義を

```
#define  D  if(1)
```

と修正するか，空列にマクロ展開するよう

```
#define  D
```

と修正して，一箇所の変更で済ます選択肢もある．

チェック用のコードであったり，使ったり使わなかったりするコードを無効化するのに，コメント化や，#if と#endif で囲むこともよく行われている．

しかし，これは次のような不都合なことが起こり得る．それは，もしデータ構造が変わるなどの変更があった場合に，コメント化した部分についてその反映を忘れる可能性である．いざアンコメント化して実行させたいというときに，エラーが出てやっと気づくことになる可能性がある．

もしコメントにしていなかったら，データ構造などの変更後の最初のコンパイルでエラーが出て，そのときに他の部分と一律に対応していた，というようなことがあり得る．

このようなことを避けるには，実行はしないが，せめてコンパイルチェックだけでも通過する方が不整合に早く気付く可能性が高まる．この点でも以上のようなマクロの利用は些細ではあるが有益である．

3.4 既存の類似例

デバッグ用のコードへの扱いとして、Linux コード [4] の nfsd.h で類似の例がある。

```
#ifdef CONFIG_SUNRPC_DEBUG
# define ifdebug(flag) if (nfsd_debug & \
                            NFSDDBG_##flag)
#else
# define ifdebug(flag) if (0)
#endif
```

3.5 留意事項

D や DD，DEBUG_xxxxx のようなマクロを用いる場合，if構文やelse構文の中ではその構文の支配範囲を変えてしまう．マクロ名に if という文字を含んでいなくても，if 文で実現しているということを忘れないよう留意が必要である．

また，Dや DD，DEBUG_xxxxx のようなマクロの効果を;の挿入で遮り無効にするのは，前側の制御構文の支配下にある場合には，してはならない．

Dや DD のような実行・非実行の制御用のマクロは，試行錯誤やデバッグの後も残っていると，通常は美しいプログラムとは言えない点も留意が必要である．

製品版など実行プログラムを小さくしたい場面では，if(0) に対してコンパイラのデッドコード生成の抑止機能がデフォルトか否かに留意が必要である．

4 おわりに

C 言語のマクロ機能を用いて，(a) 反復の指定と，(b) コードの実行・非実行の指定が，前置型の作用素のように簡潔に記述できることを提案した．

文献

[1] Kernighan, B.W. and Ritchie, D.M.(著)，石田晴久(訳)：プログラミング言語 C，共立出版 (1989).

[2] Stroustrup, B.(著)，長尾高弘 (訳)：プログラミング言語 C++，アスキー (1998).

[3] 林晴比古：C プリプロセッサ・パワー第 4 刷，日本ソフトバンク (1989).

[4] The Linux Kernel Organization：linux-4.19.3，入手先 ⟨ The Linux Kernel Archives，https://www.kernel.org/⟩ (参照 2018-11-22).

5 os.h の #define CATCH_EINTR(expr) while ((errno = 0, ((expr) < 0)) && (errno == EINTR)) や expr.h の #define for_all_symbols(i, sym) for (i = 0; i < SYMBOL_HASHSIZE; i++) for (sym = symbol_hash[i]; sym; sym = sym->next) if (sym->type != S_OTHER) など。

6 tricky-code.net の http://tricky-code.net/nicecode/code10 .php (参照 2018-7-11) に 3 項演算子を用いた同目的の定義例 #define debug 1 ? (void)0 :がある．しかし，3 項演算子は，for 文や while 文などの命令や，{ と } で囲んだ複文に対してはエラーとなる。