C プログラミングの掟とメモ

2024 年 10 月 2 日版

西井 淳

目次

1 C プログラミングの掟

1) 原則として**一つの関数単位は 1 画面程度の大きさを上限**にする
2) プログラム実行時に, **引数等の不具合で異常終了することが無いように**十分注意すること。
3) main 関数のお作法
   a) main 関数はあまり長くしない。適宜関数を呼び出し，処理のあらすじがわかるようにする。
   b) オプション指定の引数があるときには，フラグ変数に記録して，その後はそのフラグ変数を参照して処理する。
   c) まず引数の数をチェックし，不適切な場合等はプログラムをさっさと終了する。以下は例

```
1   enum {False, True};
2   void Usage(){
3     puts("USAGE: ........")
4   }
5
6   int main()
7   {
8     int sflag=False;
9     if(引数の数のチェック){
10      Usage();
11      exit(1);
12    }
13    if( -s があるか?){
14        sflag=True; /* -s があるとき */
15    }
16
17    /* プログラムの本文 */
18  }
```

4) 同じような命令をプログラム中のあちこちに書いてはいけない。よく似た処理を 2 回以上書いてしまってら，関数を作って簡単に書く方法を考える。
5) Segmentation Fault を起こすプログラムは不可
6) コンパイラ gcc を使う時には、オプションに-Wall -O2をつける。
   - -Wallは，プログラム中で確認が必要なところを表示するオプション。
   - -O2は，実行速度を速くするためのオプション。

2 Makefile

プログラムをコンパイルする手続き等を Makefile に書いておくとコンパイルが楽になる。 以下は一番簡単なサンプル。

```
arg: arg.c                # arg を arg.c から作ることを宣言
    gcc -o arg arg.c      # 具体的な作り方（行頭はタブ）

mycat: mycat.c            # mycat を mycat.c から作ることを宣言
    gcc -o mycat mycat.c # 具体的な作り方
```

Makefile 中の各行において、記号#より後ろはコメントとみなされる。

上記のような内容の Makefile を作り，以下を実行すると，その下に書いている条件がみたされる場合に２行目のコマンドが実行されて，arg コマンドが新しくつくられる。

```
$ make arg
```

- コマンド arg が現在のディレクトリに存在しない
- 現在のディレクトリにあるコマンド **arg が arg.c よりも古い**

たくさんプログラムを書くときには，ずらずらこのような記述を Makefile に並べておけばよい。Makefile 中で，変数の定義もできる。変数の値にアクセスするときには、変数名の頭に記号$をつける。

```
GCC = gcc
CFLAGS=-Wall -O2
Loadlibs=-lm   # 必要に応じて

main: main.o mycat.o    # main は main.o と mycat.o から作る
        ${GCC} ${CFLAGS} -o $@ main.o mycat.o ${Loadlibs}

arg: arg.o              # arg は arg.o から作る
        ${GCC} ${CFLAGS} -o $@ $@.o ${Loadlibs}

main.o: mycat.h main.c
        ${GCC} ${CFLAGS} -c -o $@ main.c

mycat.o: mycat.c
        ${GCC} ${CFLAGS} -c -o $@ mycat.c

clean:
        rm -f *~ *.o
```

最初の 3 行はコンパイラをあらわす変数 GCC, コンパイラに与えるオプション CFLAGS, リンクするライブラリ Loadlibsの設定。各行以降の ${GCC}, ${CFLAGS}, ${Loadlibs}は，ここで設定された値に置換される。 6 行目の$@ は，5 行目の":"で区切られた左の文字 (この場合は main) をさす変数。

make mainを実行すると，Makefile 中の記述のうち， **main に関する文より下の部分の記述から** main.o と mycat.o の作り方が参照され，それが自動的に実行される。 main.o に関する該当箇所を見ると，main.o は，mycat.h, main.c に依存していることがわかる。よって，main.o が無い場合や，mycat.h もしくは main.c が main.o より新しい場合には，新しく main.o が作られる。

また，上の例では以下を実行するとオブジェクトファイル (*.o) や，バックアップファイル *~が消去される。

```
1 $ make clean
```

このように，Makefile は単にプログラムのコンパイルだけでなく，ファイルの消去，コピーその他，さまざまに使うことができる。

先の例は以下のように書き直すことができる。たくさんのプログラムのための Makefile を書くときにはこちらのほうが楽。

```
1  GCC = gcc
2  CFLAGS=-Wall
3  Loadlibs=
4
5  main: main.o mycat.o
6      ${GCC} ${CFLAGS} -o $@ main.o mycat.o ${Loadlibs}
7
8  arg: arg.o
9      ${GCC} ${CFLAGS} -o $@ $@.o ${Loadlibs}
10
11 main.o: mycat.h main.c
12
13 .c.o :
14     ${GCC} ${CFLAGS} -c -o $@ $<
15
16 clean:
17     rm -f *~ *.o
```

.c.oと書いたエントリーは*.cから*.oを作る一般的な作り方であることを示す。 例えば mycat.o は mycat.c から作られるが、その作り方は.c.oが参照されて以下のように解釈される。

```
1 mycat.o : mycat.c
2     ${GCC} ${CFLAGS} -c -o $@ $<
```

ここで$@ は一行目の：の左の mycat.o に，$< は：の右の mycat.c に置換され，mycat.o をつくる処理が実行される。 main.o は mycat.h と main.c に依存することが記載されているが、その作成方法は具体的に書かれてないので.c.oが参照される。

3 プリプロセッサ

プリプロセッサとは、C 言語等のプログラムの**コンパイル前に**前処理を行うための簡易なプログラム言語である。

3.1 マクロ定義

```
1 #define マクロ名 置換文字列
```

#defineを用いて、文字列の置換を行うことができる。

```
1 #define MAX 100
```

と書けば、この行より下の MAXという文字列は、コンパイル前に 100に置き換えられる。 以下は例。

```
1 #define MAX 10
2 int main(void)
3 {
4 int i;
5 char buf[MAX];
6 for (i=0;i<MAX;i++) buf[i]=0;
7 }
```

3.2 マクロ関数定義

マクロ定義を使って関数等の定義もできる。

1) 引数 x の二乗を計算するマクロ定義例

```
1 #define sqr(x) ((x)*(x))
```

() がなぜ必要かは各自考えよ。

2) 2 つの文字の大きい方を返すマクロ関数定義例

```
1 #define maxof(x,y) ((x>y)?x:y)
```

3) 与えた回数ループをするマクロ関数定義例

```
1 #define loop(n) for(i=0;i<n;++i)
```

ただし、あらかじめ **int** i;が宣言されていることが必要。C++ なら変数の局所定義も OK なので、以下のようにも出来る。

```
1 #define loop(n) for(int i=0;i<n;++i)
```

4) 先の例で、ループの変数名も与えるようにした例

```
1 #define loop(i,n) for(i=0;i<n;++i)
```

マクロ関数定義では、引数の型を意識しなくていいので便利。

3.3 条件分岐

```
1 #ifdef マクロ名
2 ...
3 #elif
4 ...
5 #endif
```

#ifdefを用いて、コンパイルする場所の切替えを行うことができる。以下の例では、マクロ変数 DEBUG を定義しているときには、“デバッグ中” と表示される。

```
1 #define DEBUG
2 int main(void)
3 {
4 #ifdef DEBUG
5 printf("デバッグ中);
6 #ENDIF
7 }
```

このようにすればデバッグ中にいろんな情報を表示することもできる。また、よく似た、でも少し違うコマンドを作りたいときにも便利。条件分岐に用いられるマクロ変数には、上記の例のように置換文字列を与えてなくてもよい。

あるマクロ文字が**定義されてない場合**にのみコンパイルしたい部分は**#ifndef**を使う。

```
1 #ifndef マクロ名
2 ...  //指定したマクロが定義されていないときにコンパイルされる部分
3 #endif
```

さらにマクロ変数の値に応じた分岐も可能である。

```
#if マクロ名==値1
...
#elif マクロ名==値2
...
#else
...
#endif
```

4 パイプ

UNIX 上で，あるファイル (例えば data.txt) の中身を less で見たければ，以下を実行すればよい.

```
$ less data.txt
```

この例のように引数としてファイル名を渡すなら，main 関数の引数を用いればよい．同様の機能は，パイプ (「UNIX の基本操作」参照) を用いて，以下のように実行することもできる。

```
$ cat data.txt |less
```

cat コマンドは data.txtを標準出力に表示するコマンドである．上の例では，この標準出力への出力をlessが受け取って処理している．もう少し正確に言うと，パイプはcatからの標準出力をlessコマンドに対する標準入力に切替えている．この場合lessは標準入力から data.txtの中身を読み込んでいることになる.

以下はパイプ出力を受け取ることができるプログラム例。

```
    ....
    int main(int argc, char **argv)
    {
        char *filename;
        FILE *fp;

        filename=argv[argc-1];          /* 最後の引数をファイル名と思って取得 */

        if ( filename[0]=='-' || argc==1 ) { /* 引数が1個 またはfilenameの
                                                1文字目が'-'(オプション指定)のとき*/
            fp = stdin;
        } else {                        /* ファイル指定あり */
            fp = fopen( filename, "r" );
            if ( fp == NULL ) {
                fprintf( stderr, "'%s'が読み込めません.\n", filename );
                exit(1);                /* 異常終了 */
            }
```

```
18          }
19          ....
20      }
```

5　このドキュメントの著作権について

1) 本稿の著作権は西井淳 nishii@sci.yamaguchi-u.ac.jp が有します。
2) 非商用目的での複製は許可しますが，修正を加えた場合は必ず修正点および加筆者の氏名・連絡先，修正した日付を明記してください。また本著作権表示の削除は行ってはいけません。
3) 本稿に含まれている間違い等によりなんらかの被害を被ったとしても著者は一切責任を負いません。

間違い等の連絡や加筆修正要望等の連絡は大歓迎です。