C言語プログラミングの基礎訓練

2015 年 3 月 13 日版

西井 淳

目次

1　プログラム作成上の注意点

1) Makefile を作成し，**make コマンドでコンパイル**すること。Makefile の作り方については付録参照。
2) main 文だけのプログラムは，よほど短いもので無い限り不可。**一つの関数単位は 1 画面程度の大きさを上限**にするのが望ましい。
3) 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。
4) main 関数を書くときの注意
   a) 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。
   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) 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。
   d) 以下は main 関数の構成例

```
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 }
```

5) 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。繰り返し実行する命令は関数にする。
6) 実行のしかたによって Segmentation Fault を起こすプログラムは不可
7) 言語は C++ でもよい。(この場合ファイル入出力の命令は，練習問題中の指定と

は異なる C++ のストリーム入出力の命令を用いてよい)

8) 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　準備運動

2.1　コンパイル・リンク

[**問**] C 言語等のプログラムを書いたとき、それを実行できる形式にするには**コンパイル**と**リンク**が必要である。この「コンパイル」と「リンク」とはどのような作業をすることか説明しなさい。説明には「プリプロセッサ」,「ヘッダファイル」,「ソースプログラム」,「オブジェクトファイル」,「実行形式」というキーワードも使うこと。

2.2　変数とポインタ

以下の各プログラムを実行したとき、各変数のための記憶領域がメモリ空間でどのように確保されて値の受け渡しが行われ，結果はいかに表示されるかを説明しなさい。

問 1

```
1  #include <stdio.h>
2  int z=1;
3
4  void func(int *x)
5  {
6    static int y=8;
7    extern int z;
8    printf("%D,%D,%D\N",(*x)++,y++,z++);
9  }
10 int main(void)
11 {
12   int x=2,y=5,z=4;
13   func(&y);
14   printf("%D,%D,%D\N",x++,y++,z++);
15   func(&z);
16   printf("%D,%D,%D\N",x++,y++,z++);
17   return(0);
18 }
```

問 2

```
1  #include <stdio.h>
2
3  void func(int n1, int *np2)
4  {
5    n1=*np2;
6    *np2=8;
7    np2=&n1;
8  }
9
10 int main(void)
11 {
12   int n1=0, n2=5;
13   int *np1, *np2, *tmp;
14
15   np1=&n2;
16   n2++;
17   tmp=&n1;
18   (*tmp)++;
19   np2=tmp;
20   (*tmp)++;
21
22   printf("N1=%D, N2=%D, *NP1=%D, *NP2=%D\N", n1, n2, *np1, *np2);
23
24   func(*np1,np2);
25   printf("*NP1=%D, *NP2=%D\N", *np1, *np2);
26 }
```

3 基本

3.1 定数の定義

定数はプリプロセッサ (付録参照) を使って

```
1  #define MAX 10
```

と定義する方法と、const を使って以下のように定義する方法がある。

```
1  const int MAX=10;
```

#defineは**プログラムのコンパイル前に単なる文字列置換として実行**され、**const**を用いて宣言した変数は**プログラムの実行時にメモリ領域が確保**される。 このような特徴のため，それぞれ長所と短所がある。

1) 上述した**#define**と **const**の仕様を厳密に満たすコンパイラにより以下のプログラムをコンパイルすると、値が正常に表示されない。その理由を述べ、修正方法を 2 通り (**#define**を使う方法と **const**を使う方法) を述べなさい。

```
1  #include <stdio.h>
2  #define MIN -2
3  int main(void)
4  {
5      printf("%D",3/MIN);
6  }
```

2) 下記のプログラムをコンパイルするとコンパイルエラーが出たり、コンパイルは出来ても実行時に配列 aの値がおかしくなる等の不具合が出ることがある。 配列の大きさはコンパイル時に決まっていないといけないことに注意し，なぜこのような不具合が起きるか説明しなさい。また，**#define**を使ってプログラムを修正しなさい。

```
1  #include <stdio.h>
2  const int MAX=3;
3  int main(void)
4  {
5      int a[MAX],i;
6      for(i=0;i<MAX;i++) a[i]=0;
7      ...
8  }
```

3.2 double と float の演算

1) **double** x=0.0に 0.1 を 10 回足した値は 1.0 にならない．いったいどの程度その値は違うのだろう？ またその理由はなにか？
2) 前問で **double** のかわりに **float** を用いた場合について，同様のこと を議論せよ．
3) 以下のプログラムを作ったところ暴走してしまった．修正案を考えなさい。

```
1  #include <stdio.h>
2
3  int main(void)
4  {
5    double x=0.0;
6    while(1) {
7      x+=0.1;
8    if(x==1.0) break;
9    }
10
```

```
11    return(0);
12  }
```

4) x=1.0を 0.1 倍してから 10.0 倍するという演算を何度か繰り返す場合に付いて，xが double の場合と float の場合でどのくらい演算時間の差があるかを調べよ。プログラムの実行時間は time コマンドで計測できる ($ time <command name>)。

コメント) 結果は処理系によって違うが，**double**と**float**の演算にかかる時間は同じになる処理系も多い。また、**float**は精度が非常に悪いため、メモリ量が少ないとき以外には**float**はほとんど使われない。

3.3 main 関数への引数の処理

1) プログラムに与えた引数の数と，引数を表示するプログラム showargを作りなさい。実行結果の例は以下の通り。

```
1  $ ./showarg a b c
2  4
3  ./showarg a b c
```

2) 引数として与えた 2 つの実数の和を出力するプログラムを作りなさい。($ ./sum 1 2 と実行すれば"3" と表示されるプログラム)。引数が 2 個与えられなかったときには，以下のようなメッセージを出して終了すること。

```
1  $ ./sum 1
2  Usage: sum <num1> <num2>
```

注）main 関数への引数は全て文字列として受け取られる。文字列を **double**に変換するには関数atof() を用いる。

3.4 動的なメモリ確保

1) 引数で整数 n が与えられたとき，calloc を用いて配列 **int** a[n]を確保し，全ての $i < n$ について a[i]=i を代入した後，a[] の内容を表示するコマンド arrayをつくりなさい。なお，この節でのプログラムでは以下に気を付けること。
   a) メモリ確保に失敗したときには，エラー出力を行って終了すること。
   b) 必ず確保したメモリーは必ずプログラム終了時までに開放すること。

2) 引数で整数 n が与えられたとき，２次元配列 **int** a[n][n]を確保し，その配列を単位行列として、配列 aの内容を表示するコマンド array2をつくりなさい。

4　ファイル入出力と分割コンパイル

1) 以下の関数群をつくり，mylib.cという名前で保存しなさい。
   a) 引数で指定したファイル名の関数を読込みモードで開き，そのファイルへのファイルポインタを返す関数 fRopen。ファイルを開くのに 失敗したときには、“Failed to open ○○”(○○には引数で与えたファイル名が入る) と**標準エラー出力**に出力してプログラムを終了。

```
1    FILE* fRopen(char* fname)
```

   b) 引数で指定したファイル名の関数を書込みモードで開き，そのファイルへのファイルポインタを返す関数 fWopen。ファイルを開くのに失敗したときには、“Failed to open ○○” と**標準エラー出力**に出力してプログラムを終了。

```
1    FILE* fWopen(char* fname)
```

2) mylib.cの中の関数のプロトタイプ宣言 (関数原型宣言) のみを記載したファイル mylib.hをつくりなさい。
3) make cleanを実行したら，末尾に~ がついた名前のファイルと拡張子が.oであるファイルが削除されるように Makefile を作りなさい。

5　乱数とデータ処理

1) 引数で与えた回数だけ 0 以上 1 以下の一様乱数を発生し、結果を各行に表示するプログラム genrandをつくりなさい。また，100 回乱数を発生させた結果をリダイレクト (UNIX 関連のドキュメント参照) を用いて保存しなさい。
2) 数値データをファイルから読み込み，以下を行うプログラム getdistをつくりなさい。 ただし，以下の仕様を満たすこと。
   b) –hオプションもしくは、オプションが不適切なときには，使い方 (Usage) を以下のように表示して終了する。

```
1  $ ./getdist -h
2    Usage: getdist [option] <file>
```

```
3    option:
4    -h) Show this message
5    -n) with line number
6    ...(適宜追加)...
```

c) –a オプションでデータの平均，標準偏差，最小値，最大値を表示

d) –g オプションでデータのヒストグラムを出力 (設定した刻み幅で, 度数頻度を'*' を使って表示する)。例えば設定した刻みを 0.1 にした場合には

```
1    0-0.1:    **
2    0.1-0.2:  ****
3    0.2-0.3:  ***
4    ...
```

といった表示を行えるようにする。刻み幅の数は**#define**で用いて定義することにより，可変にする。

e) 引数に –n オプションが指定されたときには各行に行番号をつける。

f) C 言語でのオプション処理は通常の文字列処理関数を使って行っても，gcc の標準ライブラリ getoptを使っても良い。getoptの使い方は各自調べてください。

g) プログラムのはじめのほうで，どのような引数があるかをチェックし，引数が不適切な場合には，ただちに Usage を表示して終了する。

h) 既に作った mylib.cにあるファイルを開く関数を利用し、分割コンパイルを行うこと。

i) make getdistを実行すればコマンドgetdistをコンパイルできるように Makefile を作りなさい。

3) getdist .cを改造して，マクロ変数 FILE(付録参照) を定義しているときにはファイルresult .datに結果を出力するようにし，定義してないときにはこれまで通り標準出力に結果を表示するようにしなさい。

ヒント：FILE を定義した場合には出力ファイルポインタを指定したファイルに、定義していないときには標準出力に設定する。**#ifdef**の分岐は一カ所のみですむはず。

4) コンパイル時に gcc のオプションに, 例えば–DFILEを追加すると、プログラム中で**#define** FILEとマクロ定義したのと同じ意味になる。このことを利用して、make fgetdistとすれば、getdist .cをコンパイルして, 結果をファイルに出力するコマンドfgetdist を，make getdistとすれば結果を標準出力するgetdist を作るように Makefile を修正しなさい。

5.1 パイプ

1) 関数 fRopenをパイプ (付録参照) に対応できるようにした関数 fRPopenを作り, mylib.c, mylib.hに追加しなさい。
2) パイプを用いてデータファイルを受け取れるように，前問で作成したgetdistを改良したgetdist2を作りなさい。その動作は以下を試して確認しなさい。

```
$ genrand 100 | getdist2
$ genrand 100 | getdist2 -g
$ getdist2 <file name>
$ getdist2 -g <file name>
```

5.2 微分方程式

1) 微分方程式 $\dot{x} = x$ について以下を行いなさい。
   a) 微分方程式の解を（解析的に）求めよ。その解の挙動のグラフを書き，そのようなグラフの得られる理由について考察せよ。
   b) 微分方程式の解 $(x(t))$ を求めるプログラムを作成し，それをもとに自然定数 e を求めよ。
2) 以下の微分方程式について各演習を行いなさい。

$$\frac{d}{dt}\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix}\begin{pmatrix} x \\ y \end{pmatrix}$$

   a) 微分方程式の解を（解析的に）求めよ。その解の挙動のグラフを書き，そのようなグラフの得られる理由について考察せよ。
   b) 微分方程式の解 $((x(t), y(t))$ を求め，結果をグラフ表示するプログラムを作成せよ。結果のグラフ表示の方法は別途説明する。

6 解説

6.1 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。

```
1 arg: arg.c                # arg を arg.c から作ることを宣言
2     gcc -o arg arg.c      # 具体的な作り方（行頭はタブ）
3
4 mycat: mycat.c            # mycat を mycat.c から作ることを宣言
5     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。

上記のような記述をした Makefile をつくっておき，以下を実行すると，以下の場合に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。

```
1     $ 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。

```
1  GCC = gcc
2  CFLAGS=-Wall -O2
3  Loadlibs=-lm   # 必要に応じて
4
5  main: main.o mycat.o    # main は main.o と mycat.o から作る
6          ${GCC} ${CFLAGS} -o $@ main.o mycat.o ${Loadlibs}
7
8  arg: arg.o              # arg は arg.o から作る
9          ${GCC} ${CFLAGS} -o $@ $@.o ${Loadlibs}
10
11 main.o: mycat.h main.c
12         ${GCC} ${CFLAGS} -c -o $@ main.c
13
14 mycat.o: mycat.c
15         ${GCC} ${CFLAGS} -c -o $@ mycat.c
16
17 clean:
18        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。

6.2 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。

6.2.1 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 }
```

6.2.2 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。

6.2.3 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。

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

```
4 ...
5 #else
6 ...
7 #endif
```

6.3 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．

```
1 $ 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。

```
1 $ 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。

```
1     ....
2     int main(int argc, char **argv)
3     {
4        char *filename;
5        FILE *fp;
6
7        filename=argv[argc-1];        /* 最後の引数をファイル名と思って取
          得 */
8
9        if ( filename[0]=='-' || argc==1 ) { /* 引数が1個 または
          filenameの
10                                          1文字目が'-'(オプション指定)の
          とき*/
11          fp = stdin;
12       } else {                      /* ファイル指定あり */
13          fp = fopen( filename, "r" );
14          if ( fp == NULL ) {
15             fprintf( stderr, "'%s'が読み込めません.\n", filename );
16             exit(1);                /* 異常終了 */
```

```
17              }
18          }
19          ....
20      }
```

7　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。