第3章　UNIXでのプログラミング初歩

本章では，UNIX の CUI(Character User Interface) における原始的なプログラミングの例を解説する。原始的というのは，UNIX 開発以来三十年以上も連綿と使われきたツールのみを使っているからである。それ故にここで述べていることは今後ともそれほど変化がないであろうと予想される。Visual C++や Visual Basic といったお手軽ではあるがめちゃくちゃ高価で重い統合環境でしかプログラミング経験のない若人も，一度はとっつきづらい，でもものすごく軽い CUI を経験してきたいものである。

3.1　最初の一歩

まず，C コンパイラの代表格である gcc コマンド [13] を使ってみる。これは代表的な free software であるので，大概の UNIX 環境で使用可能である。もしなければ，以下の記述のうち gcc の部分を適当な ANSI C コンパイラ (cc コマンド等) に置き換えて読んで頂きたい。
まず，手っ取り早く C プログラムを体験するために，hellow.c プログラムを作ることにする。次のプログラムを自分のホームディレクトリ以下に配置する。

```
 1 : #include <stdio.h>
 2 : #include <stdlib.h>
 3 :
 4 : int main()
 5 : {
 6 :     printf("Hellow, World\n");
 7 : //    return EXIT_SUCCESS;
 8 :     return EXIT_FAILURE;
 9 : }
10 :
```

そしてこれは次の手順に従って，コンパイルし，標準関数ライブラリ (libc.a 等) をリンクし，実行ファイル a.out をソースファイルと同じカレントディレクトリに生成する。行頭の% はコマンドプロンプトを表しているので入力せず，”gcc”から入力すること。

```
%␣gcc␣hellow.c
```

UNIX 環境下ではカレントディレクトリにはパスが通っていないことがあるので，相対ディレクトリ指定を行って実行してみる。以下のように，画面に”Hellow, World!”と表示されれば合格である。

```
% ./a.out
Hellow, World!
%
```

では次に，実行ファイル名が”hellow”となるようにコンパイルする。そのために”-o”オプションを使用する。

```
% gcc -ohellow hellow.c
```

これは

```
% gcc -o hellow hellow.c
```

と指定してもよい。今度は a.out の代わりに hellow という実行ファイルがカレントディレクトリに生成されているので,

```
% ./hellow
```

を入力して実行する。

では，この手順を自動的に行ってくれる make コマンドを使ってみることにする。以下の内容を打ち込んで，hellow.c と同じディレクトリに”Makefile”というファイル名で保存する。この際，最初の”M”は必ず大文字を使うこと。

```
1 : CC=gcc
2 : DEL=rm
3 :
4 : hellow: hellow.c
5 :     $(CC) -o hellow hellow.c
6 :
7 : clean:
8 :     -$(DEL) hellow
```

これによって，make コマンドをオプションなしで実行すると，Makefile の内容が忠実に実行され，1 行目の hellow マクロを実行すべく，gcc コマンドが発行され，エラーがなければ先ほどと同様 hellow という実行ファイルが生成される。

```
% make
% ./hellow
Hellow, World!
%
```

これ以降は，C ソースファイルを作った後，それをコンパイルするための Makefile を作り，make コマンド一発で実行ファイルが生成できるようにする。

3.2 数値計算プログラミングの基礎

コンパイル方法は理解して頂いたので，今度は計算をするプログラムを作って動かしてみることにする。
まずは整数型 (int) を用いて 3 + 2 を計算し，その結果を表示するプログラムである。

```
 1 : #include <stdio.h>
 2 : #include <stdlib.h>
 3 :
 4 : int main()
 5 : {
 6 :     int a, b, c;
 7 :
 8 :     a = 3;
 9 :     b = 2;
10 :
11 :     c = a + b;
12 :
13 :     printf("%d\n", c);
14 :
15 :     return EXIT_SUCCESS;
16 : }
17 :
```

これを，先ほどの hellow.c と同時にコンパイルして，hellow と integer という二つ実行ファイルを生成する Makefile は次のようになる。

```
 1 : CC=gcc
 2 : DEL=rm
 3 :
 4 : all: hellow integer
 5 :
 6 : hellow: hellow.c
 7 :     $(CC) -o hellow hellow.c
 8 :
```

```
 9 : integer: integer.c
10 :     $(CC) -o integer integer.c
11 :
12 : clean:
13 :     -$(DEL) hellow
14 :     -$(DEL) integer
```

integerを実行すると

```
% ./integer
5
%
```

となる。
次に，IEEE754倍精度実数型(double)を使用するプログラムをコンパイルしてみる。

```
 1 : #include <stdio.h>
 2 : #include <stdlib.h>
 3 : #include <math.h>
 4 :
 5 : int main()
 6 : {
 7 :     double a, b, c;
 8 :
 9 :     a = 1.0;
10 :     b = 3.141592;
11 :
12 :     c = a + b;
13 :
14 :     printf("%e, %e\n", c, cos(b));
15 :
16 :     return EXIT_SUCCESS;
17 : }
18 :
```

これをコンパイルしてfloatという実行ファイルを生成するMakefileは次のようになる。

```
 1 : CC=gcc
 2 : DEL=rm
 3 :
 4 : LIB=-lm
 5 :
 6 : float: float.c
 7 :     $(CC) -o float float.c $(LIB)
```

```
 8 :
 9 : clean:
10 :     -$(DEL) float
```

ここで重要なことは，libm.a なる標準数学関数ライブラリを明示的にリンクしていることである。もしこれがなければ実行ファイル生成時に

```
/tmp/ccClEG8A.o: In function ‘main’:
/tmp/ccClEG8A.o(.text+0x43): undefined reference to ‘cos’
collect2: ld はステータス 1 で終了しました
```

というエラーを食らうことになる[1]。これは 13 行目の cos 関数がライブラリ内に見あたらない，というメッセージを発している。

float を実行すると

```
% ./float
4.141592e+00, -1.000000e-00
%
```

となる。

3.3 多倍長計算プログラミング

では，本章の最後に，GMP[7] と MPFR[14] を利用した多倍長計算の例を見ることにする。

先ほどの integer.c を GMP が提供する多倍長整数型 (mpz_t) を使うと，次のようなプログラムとなる。

```
 1 : #include <stdio.h>
 2 : #include <stdlib.h>
 3 : #include "gmp.h"
 4 :
 5 : int main()
 6 : {
 7 :     mpz_t a, b, c;
 8 :
 9 :     mpz_init_set_ui(a, 3UL);
10 :     mpz_init_set_ui(b, 2UL);
11 :     mpz_init(c);
12 :
13 :     mpz_add(c, a, b);
```

[1] Vine Linux 2.6r1 での実行例。

```
 14 :     mpz_out_str(stdout, 10, c);
 15 :     printf("\n");
 16 :
 17 :     mpz_clear(a);
 18 :     mpz_clear(b);
 19 :     mpz_clear(c);
 20 :
 21 :     return EXIT_SUCCESS;
 22 : }
 23 :
```

これをコンパイルして実行ファイル gmp-mpz を生成する Makefile は次のようになる。

```
  1 : CC=gcc
  2 : LIB=-lgmp -lm
  3 :
  4 : gmp-mpz: gmp-mpz.c
  5 :     $(CC) -ogmp-mpz gmp-mpz.c $(LIB)
```

gmp-mpz を実行すると，integer と同様に

```
% ./gmp-mpz
5
%
```

という結果を得る。

次に，GMP の多倍長浮動小数点型 (mpf_t) を用いたプログラムを実行してみる。この場合，整数型と異なり，仮数部の桁数 (10 進 50 桁) を指定する必要がある。

```
  1 : #include <stdio.h>
  2 : #include <math.h>
  3 :
  4 : #include "gmp.h"
  5 :
  6 : main()
  7 : {
  8 :     mpf_t a, b, c;
  9 :
 10 :     mpf_set_default_prec(ceil(50/log10(2.0)));
 11 :
 12 :     mpf_init_set_ui(a, 1UL);
 13 :     mpf_init_set_str(b, "3.141592", 10);
 14 :     mpf_init(c);
 15 :
 16 :     mpf_add(c, a, b);
 17 :     mpf_out_str(stdout, 10, 0, c);
```

```
 18 :     printf("\n");
 19 :
 20 :     mpf_clear(a);
 21 :     mpf_clear(b);
 22 :     mpf_clear(c);
 23 : }
 24 :
```

先ほどの Makefile を実行ファイル gmp-mpf も同時に生成するように変更する。

```
  1 : CC=gcc
  2 : LIB=-lgmp -lm
  3 :
  4 : all: gmp-mpz gmp-mpf
  5 :
  6 : gmp-mpz: gmp-mpz.c
  7 :     $(CC) -ogmp-mpz gmp-mpz.c $(LIB)
  8 :
  9 : gmp-mpf: gmp-mpf.c
 10 :     $(CC) -ogmp-mpf gmp-mpf.c $(LIB)
```

実行ファイル gmp-mpf を実行すると

```
% ./gmp-mpf
0.414159199999999999999999999999999999999999999999999999999998e1
%
```

という結果を得る。

最後に，IEEE754 互換の多倍長浮動小数点演算環境を提供してくれる MPFR[14] を使ってみることにする。先ほどの gmp-mpf.c を MPFR を使うように変更すると次のようになる。

```
  1 : #include <stdio.h>
  2 : #include <math.h>
  3 :
  4 : #include "gmp.h"
  5 : #include "mpfr.h"
  6 : #include "mpf2mpfr.h"
  7 :
  8 : main()
  9 : {
 10 :     mpf_t a, b, c;
 11 :
 12 :     mpf_set_default_prec(ceil(50/log10(2.0)));
 13 :
```

```
 14 :     mpf_init_set_ui(a, 1UL);
 15 :     mpf_init_set_str(b, "3.141592", 10);
 16 :     mpf_init(c);
 17 :
 18 :     mpf_add(c, a, b);
 19 :     mpf_out_str(stdout, 10, 0, c);
 20 :     printf("\n");
 21 :
 22 :     mpf_clear(a);
 23 :     mpf_clear(b);
 24 :     mpf_clear(c);
 25 : }
 26 :
```

これをコンパイルして実行ファイル mpfr を生成するよう Makefile を変更すると

```
  1 : CC=gcc
  2 : LIB=-lmpfr -lgmp -lm
  3 :
  4 : all: gmp-mpz gmp-mpf mpfr.c
  5 :
  6 : gmp-mpz: gmp-mpz.c
  7 :     $(CC) -ogmp-mpz gmp-mpz.c $(LIB)
  8 :
  9 : gmp-mpf: gmp-mpf.c
 10 :     $(CC) -ogmp-mpf gmp-mpf.c $(LIB)
 11 :
 12 : mpfr: mpfr.c
 13 :     $(CC) -ompfr mpfr.c $(LIB)
```

となる。mpfr を実行すれば

```
% ./mpfr
4.14159200000000000000000000000000000000000000000000
%
```

という結果を得る。

以降の多倍長計算は，GMP と MPFR を使うことにする。

演習問題

1. float.c を改良し，IEEE754 倍精度で $\sqrt{2} + \sqrt{3}$ を計算プログラムを作れ。[ヒント: `sqrt(2.0)` で $\sqrt{2}$ の計算を実行できる。]

2. gmp-mpf.c を改良し，10 進 40 桁で $\sqrt{2}+\sqrt{3}$ を計算プログラムを作れ。[ヒント: mpf_t 型の変数 `ret` と `src` を用意しておき，`mpf_set_ui(src, (unsigigned long)2); mpf_sqrt(ret, src);` とすることで $\sqrt{2}$ の計算を多倍長で実行できる。]

3. mpfr.c を改良し，10 進 40 桁で $\sqrt{2}+\sqrt{3}$ を計算プログラムを作れ。また，その結果は gmp-mpf.c とどのように違うか，考察せよ。