計算機プログラミングI 講義ノート #3

担当：井上 純一 (情報科学研究科棟 8-13), 赤間 清 (情報基盤センター)
URL：http://chaosweb.complex.eng.hokudai.ac.jp/~j_inoue/PROG2007/PROG2007.html

平成 19 年 4 月 20 日

目 次

8 表ジョブと裏ジョブ

第 2 回講義でターミナル上で Xemacs を起動する際

```
xemacs &
```

のように「&」を最後につけて起動しました. この「&」は Xemacs というジョブ (計算機上での作業) を**裏ジョブ (background job)** として実行することを意味します. 一方, この「&」をつけないで Xemacs を起動する場合, Xemacs を**表ジョブ (foreground job)** として実行したことになります. ここで, 両者の違いは, 裏ジョブの場合にはコマンド・タスクの終了を待たずに次のジョブを実行できるという点です. 言い方を変えると, **表ジョブは一つのターミナルで常に一つであるのに対し, 裏ジョブは複数個同時に実行することができる**という点です.

8.1　ジョブの監視

さて, 裏ジョブはターミナルの次の入力コマンドに依存しない形で実行しているわけですから, このジョブは何らかの形で監視できなければなりません (正常に動作しているか, 現時点で何秒間動作しているか, など). そこで, この目的のためにいくつかのコマンドが用意されており, その中の一つは jobs コマンドであり, これは

```
jobs
```

とします. すると, もし Xemacs が裏ジョブとして起動しているのであれば, ターミナル上に

```
+Running xemacs
```

と表示されます. 従って, この jobs コマンドで現時点でどの作業が計算機上で実行されているのかを確認することができるわけです.

もう一つのコマンドは ps であり

```
ps
```

あるいは

```
ps -u [loginID] -l
```

とすれば OK です. 最後の「–l」オプションは「リスト形式で表示せよ」という意味です[1]. 次の練習問題でこれを確認してみましょう.

練習問題 8.1
Xemacs を用いて, test.c と test2.c を裏ジョブとして起動することで開き, jobs コマンド, ps コマンドの各々を用いて, それらのジョブの状態を確認せよ.

9　変数の宣言と代入

ここからは C 言語の各論を学んでいきます. まずは, C 言語で用いる変数について見ていくことにしましょう. その準備のため先週の復習を兼ねて次の練習問題をやってみましょう.

練習問題 9.1
次のプログラムを Xemacs を用いて書き, hensu.c という名前で保存し, 次いでコンパイル, 実行し, 結果を確認せよ.

```
#include<stdio.h>
main()
{
        int add, sub, pro, div, mod, x, y;  /* 変数の宣言 */
```

[1] ps コマンドについてもう少し詳しく知りたいのであれば, 先週学んだオンラインマニュアルを早速使って man ps で調べてみよう.

```
        x = 8; /* x に 8 を代入 */
        y = 4; /* y に 4 を代入 */

        add = x + y; /* x と y を足したものを add に代入 */
        sub = x - y; /* x から y を引いたものを sub に代入 */
        pro = x*y; /* x と y を掛け合わせたものを pro に代入 */
        div = x/y; /* x を y で割ったものを div に代入 */
        mod = x%y; /* x を y で割った余りを mod に代入 */

        printf("%d %d\n", x,y);
        printf("%d %d %d %d %d\n", add,sub,pro,div,mod);
}
```

上記のプログラムで「/*」と「*/」で囲まれた部分は「コメント」であり, コンパイル時には無視される部分です. 従って, この部分には何を書き込んでも良く, その行がどのような処理を行う (行わせようとしている) のかをこまめに書き込んでおくと良いでしょう. 時間が経った後では自分の書いたプログラムでさえも, どのようなものであったかを思い出すことは容易ではないことが結構あります. このような状況を回避するためにも, 自分の書くプログラムにはなるべくコメントを書く習慣をつけると良いと思います.

さて, hensu.c の main() 関数の先頭にある

```
int add, sub, pro, div, mod, x, y;  /* 変数の宣言 */
```

は「add, sub, pro, div, mod, x, y の 7 つの変数は整数型の変数として扱う」ということの宣言です. ここで用いた int 型の他に下表のように変数の性質により型が異なります.

指定子	データタイプ	表現範囲
char	文字	ASCI 文字
int	整数	$-2^{31} \sim 2^{31}-1$
float	単精度浮動小数点	$\pm 3.4 \times 10^{28} \sim \pm 3.4 \times 10^{38}$
double	倍精度浮動小数点	$\pm 1.7 \times 10^{-308} \sim \pm 1.7 \times 10^{308}$

また, hensu.c の printf() 関数ですが, これは前回も見たように, 一般的に printf("....") のような形式をとり, "..."を表示する機能を持ちましたが, この関数はまた

```
printf("%d\n",a);
```

のような形式をとり, **変数 a を 10 進数で画面に表示せよ**という作業をさせることもできます. この %d は printf() 関数の**変換指定子**と呼ばれ, ここでの d は 10 進数を表します. 従って, もう少しこの手の printf() 関数の使い方を一般的に書けば

```
printf("%[変換指定子]\n", 変数);
```

となり, 表示したい変数が複数ある場合には

```
printf("%[変換指定子] %[変換指定子] .....[変換指定子]\n", 変数, 変数, ..., 変数);
```

とすれば良いわけです.

変換指定子と表現形式の対応は次の通りです.

変換指定子	表現形式
d,i	10 進数
u	符号無し 10 進数
o	符号無し 8 進数
x,X	符号無し 16 進数
c	文字
s	文字列
f	通常の小数形式 (例: 123.4567)
e,E	指数形式 (例: 1.234567E+02)
g,G	指数部の桁数により, f 形式と g 形式を自動振り分け

練習問題 9.2 (※ LMS 入力課題) [2]
A,B,C さんの英語, 数学, 国語, 理科, 社会の点数は以下の表のようであった.

名前	英語	数学	国語	理科	社会
A	70	60	53	67	82
B	40	93	48	81	30
C	94	30	87	36	85

このとき, A, B, C のそれぞれに対し, 5 教科平均を算出し

```
score_A=*** score_B=*** score_C=***
```

の形式で表示するプログラムを作成せよ.

9.1 変数の名前

C 言語における変数の命名規則を次に列記しておきます.

(1) 最初の文字は英字か下線 (_) でなければならない.
(2) 2 番目以降の文字は英字か下線か数字でなければならない.
(3) 大文字と小文字は区別される.
(4) 変数名の長さには制限はないが, 最初の 31 文字までが有効である.
(5) 途中に英字, 下線, 数字以外の文字があってはいけない.
(6) 途中にスペース, タブ, 改行記号があってはいけない.

また, int, printf, main 等, C 言語の文法上で明確に定義され, 予め意味を持ち, 予約されたもの (所謂「予約語」) は変数として用いることができません. このような変数を用いた場合にはコンパイル時エラーが出るはずですから, その際にはもう一度変数名を考え直し, 適切な変数名を用いるようにします.

[2] 「※ LMS 入力課題」と書かれた練習問題は端末で実行するだけではなく, LMS の課題として投稿してください. 詳しくは講義中に指示します.

9.2　代入による型変形

ここでは, ある型で宣言した変数に, その宣言とは異なる型のデータが代入された場合にどのようなことが生じるのかを調べてみましょう. まず, 次のケースを考えます.

```
float x; /* x という単精度浮動小数点型変数を宣言する */
x = 12; /* 浮動小数点型変数 x に整数型データを代入する */
```

この場合には, **整数型が型変形され, 浮動小数点型になって x に代入される**ことになります.
また, 次のケースはどうでしょうか.

```
int k; /* k という整数型の変数を宣言する */
k=12.3456; /* 実数データを整数型の変数 k に代入する */
```

この場合には**浮動小数点データが整数型に型変換される**ことになります. 次の練習問題を見てみましょう.

練習問題 9.3
次のプログラムを書き, katahenkei.c という名前で保存し, コンパイル・実行せよ. そこで得られた結果について考察せよ.

```
#include<stdio.h>
main()
{
     int i;
     double x,y;
     x=i=y=3.141592;
     printf("x=%lf y=%lf i=%d",x,y,i);
}
```

9.3　キャスト演算

一般的に次の形:

```
(型名) 式;
```

によって, まずは「式」が評価され, その値が型名で指定された型に変換されます. このような変換を**キャスト (cast)** と呼び, 型名を () でくくったものを**キャスト演算子 (cast operator)** と言います. **キャスト演算は剰余算よりも先に行われる**ことを注意しておきましょう. 実際の使い方は次のプログラムのようになります.

```
#include<stdio.h>
main()
{
     int sum,N;
     float f;
     sum=314;
     N=100;
```

```
    f = (float)sum/N;
    printf("f=%f\n",f);
}
```

このプログラムでは整数型の変数 sum をキャスト演算子で単精度浮動小数点に直し, その値を整数 N で割った値を浮動小数点型の変数 f に代入し, それを表示させています. これをふまえて, 次の練習問題をやってみましょう.

練習問題 9.4 (※ LMS 入力問題)
次のプログラムを書き, 答えが正しいかどうかを確認せよ. また, 正しくない場合にはキャスト演算子を用いて正しい結果が出力されるようにプログラムを変更せよ.

```
#include<stdio.h>
main()
{
    int i,j;
    double f;
    i=7;
    j=2;
    f=i/j;
    printf("f=%lf\n",f);
}
```

9.4　混合演算による型変形

異なる型どうしの演算 (混合演算) の場合, 精度の高い方への型変換が行われます. 次の例で確認しておきましょう.

```
#include<stdio.h>
main()
{
        int k;
        float x;
        double y;

        k+1; /* 型変形無し */
        k+x; /* k が float に型変換される */
        k+y; /* k が double に型変換される */
        x+y; /* x が y に型変換される */
        ..................
        .........
        ......
}
```

9.5　様々な演算子とその使い方

教科書の pp. 21-27 にリストアップされている各種演算子の中でさしあたり比較的使用頻度の高いものをピックアップして説明します.

10　関数 scanf() の使い方

変数に関して学んだところで, scanf() という関数の使い方について見ておきましょう. scanf() 関数は

```
scanf(書式, &変数名 1, &変数 2, ... ,&変数 n);
```

のようにプログラム中で用います. この機能としては, **キーボードからの文字列として与えられるデータが, 書式に記されている各変数指定に従って, 計算機内部の値に変換され, 変数 1, 変数 2, ... , 変数 n に順次格納される**ということです.

具体的には次のような使い方をします.

```
#include<stdio.h>
main()
{
            int data1,data2,add;
            scanf("%d %d",&data1,&data2);
            add=data1+data2;
            printf("add=%d\n",add);
}
```

これを例えば, examScanf.c という名前で保存し

```
gcc examScanf.c -o b.out -lm
```

としてコンパイルし, その後

```
./b.out
```

として実行すると, 画面は入力待ちの状態になるので, そこに

```
12 34
```

と打ち込むと, 画面には

```
add=46
```

と出力されるはずです. つまり, キーボードから打ち込まれた 12 と 34 はそれぞれ変数 data1, data2 に格納され, これを足したものが add に代入された後, printf() 関数により画面に表示されるわけです.

11　流れの制御

計算機に何かの作業をさせたい場合, ある条件に合致するときのみ, それを実行させたいとか, いくつかの場合を想定し, その場合に応じて異なる処理を行わせたいというようなことがたびたびあります. ここからは C 言語でそのような「条件分岐」を含む処理をどのようにして行わせるのかについて詳しく見ていきます.

11.1　if 文

まず, 最も基本的な **if 文**と呼ばれる制御法の使い方はプログラム中で次のように書きます.

```
if(式) 文;
```

ここで, この if 文の機能は () 内が評価され, その値が 0 でなければ (真ならば), () に続く文が実行されます. もし, () 内が 0 ならば (偽ならば), () に続く文は実行されず, この if 文に続く次の文が処理の対象になり, 実行されます.

実際には次のようにして使うことになります.

```
#include<stdio.h>
main()
{
            float h,w,s;
            scanf("%f",&h);  /* キーボードから身長を読み込む */
            scanf("%f",&w); /* キーボードから体重を読み込む */
            s=(h-100)*0.9;  /* 標準体重を算出 */

    /* もし，体重から標準体重を引いた値が 10 以上であれば警告を促す */
            if(w-s >=10.0)
            printf("Be careful ! You are overweight! \n");
}
```

この例で見るように, if 文は () 内の関係式 (あるいは等価式) が成立するときのみ, それに続く文を実行する働きを持ちます. 関係式, 等価式に用いる**関係演算子**, **等価演算子**は以下の通りです.

関係演算子	一般形	意味
$<$	e1 $<$ e2	$e_1 < e_2$ なら 1, 違えば 0
$>$	e1 $>$ e2	$e_1 > e_2$ なら 1, 違えば 0
$<=$	e1 $<=$ e2	$e_1 \leq e_2$ なら 1, 違えば 0
$>=$	e1 $>=$ e2	$e_1 \geq e_2$ なら 1, 違えば 0
等価演算子	**一般形**	**意味**
$==$	e1 $==$ e2	$e_1 = e_2$ ならば 1, 違えば 0
$!=$	e1 $!=$ e2	$e_1 \neq e_2$ ならば 1, 違えば 0

※ 特に等価演算子の== と代入を意味する= を間違うケースが非常に多いですから注意してください.

11.2 if else 文

条件分岐をさせるには次のような **if else 文**を用いることもできます.

```
#include<stdio.h>
main()
{
            float h,w,s;
            scanf("%f",&h);  /* キーボードから身長を読み込む */
            scanf("%f",&w); /* キーボードから体重を読み込む */
            s=(h-100)*0.9;  /* 標準体重を算出 */

    /* もし，体重から標準体重を引いた値が 10 以上であれば警告を促す */
           if(w-s >=10.0)
           printf("Be careful ! You are overweight! \n");

    /* そうでなければダイエットの必要はないと表示する */
            else
            printf("OK. You do not need diet. \n");

}
```

if else 文は次のような「入れ子」の形で用いることもできます.

```
if (式 1) {文 1;
}else if (式 2){文 2;
}else if (式 3){文 3;
.....................
.....................
}else if (式 n){文 n;}
```

今週の LMS 入力課題

2 次方程式 $x^2 + ax + b = 0$ の係数 a, b を尋ねられ, キーボード入力により, a, b の値を入力すると, この 2 次方程式の解が 2 つの実根を持つ場合には 2 つの解を出力し, 重根の場合には 1 つの解のみを表示し, 虚数解を持つ場合には “No Solution” と表示するプログラムを作成せよ.

※ 注意：ルートを使う場合には, 例えば, 実数 a のルートを実数 y に代入したければ

```
y=sqrt(a);
```

とします. また, この際には

```
#include<math.h>
```

の一行を先頭に加えてください.