プログラミング演習 第 2 回
C で計算をしよう！

2000 年 4 月 13 日

目 次

1 基本データ型 **1**

2 インタラクティブなプログラム **1**

3 整数の演算 **2**

4 繰り返し **3**

5 計算機イプシロン **5**

6 近似値と誤差 **5**

6.1 誤差の限界 5

6.2 いろいろな誤差 6

7 問題 **7**

8 付録 **8**

本日は, C で簡単な数値計算のプログラムを書くことを目標にします。
その前に, C で用いられる変数についてお話しします。

1 基本データ型

C を用いたプログラミングでは，変数を使うときには必ずそのデータの型を指定して，その名前を宣言する必要があります．

変数の名前は，アルファベットで始まる 31 文字以下の文字の並びです．(31 文字を超える変数名を使うと，32 文字目以降は無視される．) 実際にプログラムを動かすときデータは，主記憶に格納されます．機械語のレベルでは，この時に必要な記憶容量を正確に知っておく必要があります．つまり Compiler は，変数の宣言によって必要となる記憶容量を計算し，これを元にして，プログラムの内容を機械語に翻訳していくのです．この記憶要領の計算の元になるのが，データの型と言われるものです．

C 言語におけるもっとも基本的なデータの型は，以下のとおりです．これ以外にも，データの型というものがあるのですが，それらについては後の講義でふれます．

char 1 文字を表す．通常は， 1 byte = 8 bit．皆さんの使っている処理系もそうです．ただ，これでは日本語の漢字等は表示できないので，漢字表示が必要な場合には特別なことをします．この講義ではその事には触れません．

int 整数型．これは数学的な無限の整数を扱える訳ではなく，有限の範囲の整数しか扱えません．扱える範囲は処理系によって様々です．RAINBOW のマシンでは int のサイズは 4 byte = 32 bit です．

float 浮動小数点型．実数を扱うときに使用します．現在ではあまり使いません．RAINBOW のマシンでは float のサイズは 4 byte = 32 bit です．

double 倍精度浮動小数点型．float では計算の誤差が多く，精度が確保できないときに使うために，設けられたのが起源です．しかし現在では，この double を使うのが普通です．RAINBOW のマシンでは double のサイズは 8 byte = 64 bit です．

四則計算にはいる前に, 前回の補足をします.

2 インタラクティブなプログラム

インタラクティブ (interactive) とは，会話的という意味です．正確な意味は辞書に載っています．

実行の途中で文字の入力を要求するようなプログラムを作成します．以下の例では，改行までの入力文字列を char 型の配列に変換する gets() という関数を利用しています．次のプログラムを 2-1.c という名前で作成して下さい．出来たら，コンパイルして実行してみて下さい．

```
#include <stdio.h>

main()
{
    char        nyuuryoku[80];   /* 半角 80 文字まで入力できます */

    printf("文字列を入力して下さい >> ");
```

```
    /* 改行までの入力文字を char 型の配列に変換する */
    gets(nyuuryoku);

    printf("あなたの入力した文字は\n\t%s\n です\n", nyuuryoku);
}
```

printf の新しい使い方が出てきています.

関数 printf の最初の引き数の中の%s は引き数が char 型の配列名のとき，”文字列に直して出力せよ.” という意味です．\t はタブ (右にずらし特定の列に出力する機命令) です．

これはUNIX のecho コマンドの簡単な version です。(実際のecho コマンドはmain関数部分を, main(int argc, char *ar 書くことにより, コマンドラインからの情報を文字列して取得できますが, ポインタの概念が必要なので後で詳しく触れます.)

では、整数の四則計算をしましょう.

3 整数の演算

次のプログラムを 2-2.c という名前で作成して下さい．出来たら，コンパイルして実行してみて下さい．

```
#include <stdio.h>

main()
{
    int     a, b;       /* 変数 a,b が整数型であることを宣言 */
    int     sum, diff, prod, quot;  /* 同じく型宣言 */

    a = 5;  /* a に 5 を代入しています */
    b = 3;  /* 同じく代入しています */

    sum = a + b;        /* sum に a+b を代入します */
    diff = a - b;       /* diff に a-b を代入します */
    prod = a * b;       /* 掛け算は * という記号を使います */
    quot = a / b;       /* 割算は / という記号を使います */

    /* printf は"で囲まれた書式に従って出力します */
    printf("summation = %d, difference = %d\n", sum, diff);
    printf("product = %d, quotient = %d\n", prod, quot);
}
```

関数 printf に渡されるパラメータは，2 重引用符 (”) で囲まれた文字列，変数 sum，変数 diff の 3 つです (プログラミングでは関数のパラメータを引き数といったりします.)

最初の引き数 (”で囲まれた文字列) の中の最初の%という文字とそれに続く文字は 2 番目の引き数の値が代入されます．同様に，2 つ目の%という文字とそれに続く文字は，3 番目の引き数の値に取って代わられます．

1 行目の最初の%d は ”整数 sum の値を出力せよ.” という意味です． 2 番目の%d は整数 diff の値を出力せよ．という意味です． 2 行目のprintf 文についても同様です．実数型 (double) の変数の出力には%f を使います．

このように，関数printf の最初の引き数は，文字列を出力するだけでなく，2 番目以降の引き数の値の表示方法を指定します．

問題 上のprintf の中の書式指定において%d を%f に変えたらどうなるか調べよ.

今度は, キーボード から数を入力して計算させるプログラムを作ります.(つまり、標準入力からのデータに対してのプログラム)

先に, 文字列を入力する場合, 関数gets を利用しました. 今度は, 文字列以外にも適用できる関数scanf をもちいます.

次のプログラムを 2-3.c という名前で作成して下さい．(プログラム 2-2.c を利用して作成せよ.)

```
#include <stdio.h>

main()
{
    int     a, b;        /* 変数 a,b が整数型であることを宣言 */
    int     sum, diff, prod, quot;  /* 同じく型宣言 */

    printf("二つの整数 a,b の四則計算をします.>>");
    printf("a, b に代入する数を空欄をいれて連続して入力して下さい");
    scanf("%d %d", &a, &b);          /* a,b にデータ入力 */


    sum = a + b;         /* sum に a+b を代入します */
    diff = a - b;        /* diff に a-b を代入します */
    prod = a * b;        /* 掛け算は * という記号を使います */
    quot = a / b;        /* ひき算は / という記号を使います */

    /* printf は"で囲まれた書式に従って出力します */
    printf("summation = %d, difference = %d\n", sum, diff);
    printf("product = %d, quotient = %d\n", prod, quot);
}
```

関数 scanf の使い方は,

```
scanf("%d",&a)
```

で, int 型の変数 a にデータを入れます. ただし、double 型の変数 x の場合は,

```
scanf("%lf",&x)
```

となります． "l"(エル) を忘れずに!!.

変数 a の前の記号&は、アンパサンド といい, 変数 a のアド レスを与えています． 今は, おまじないと思って下さい.

4 繰り返し

次に繰り返しを使ったプログラムを作成してみます．

次のプログラムを 2-4.c という名前で作成して下さい．出来たら，コンパイルして実行してみて下さい．

```
#include <stdio.h>

main()
{
    int     i;          /* i = 0, 1, ..., 5 */
    int     sum;        /* 0 から 4 までの和 */

    sum = 0;    /* 変数を初期化します */

    /* for 文は繰り返しの計算に使われます */
    for (i = 0; i < 5; i++) {
        sum = sum + i;
    }

    printf("sum = %d\n", sum);

}
```

まずは，for (...) の後ろの {} で括られた部分を見てみましょう．

```
sum = sum + i;
```

はsum の値にi の値を加えたものを新しいsum の値とせよ．という意味です．この部分は

```
sum += i;
```

と書くことも出来ます．

for 文は次のような形で使用します．

```
for（式１； 式２； 式３）{
    …；
}
```

式 1 は，for 文の中の処理を行う前に 1 度だけ行います．次に式 2 を判定し，その結果が真ならば，{} で囲まれた部分を実行します．その後，式 3 を実行し，再び式 2 を判定します．あとはこの繰り返しです．

上の例の式 3 には

```
i++
```

が書かれています．これは i を 1 増やせという意味でインクリメントと呼ばれます．この部分は，

```
i = i + 1;
```

あるいは，

```
i += 1;
```

と書いても同じことです．

問題 1 から 100 までの奇数の和を求めるプログラムを作成せよ.

5　計算機イプシロン

計算機イプシロンとは

$$1.0 + \epsilon > 1.0$$

となる最小の正の数 ϵ のことです．もちろん，実数の場合は（有理数でも）上の式をみたす ϵ はいくらでも 0 に近くとれるので意味がありません（数学では最小値は存在しないというのでした．）

計算機の場合，実数型はトビトビの値を取りますから，上記の ϵ が定まります．次のプログラムを 2-5.c という名前で作成して下さい．出来たら，コンパイルして実行してみて下さい．

```
#include <stdio.h>

main()
{
    int         i;      /* 繰り返しの数を数える */
    double      e;      /* 計算機イプシロン */

    e = 1.0;        /* 実数型では，代入する数も少数にします */

    /* 2 で割っていき，1.0 + e = 1.0 となったらループを抜けます */
    for (i = 0; 1.0 + e > 1.0; i++) {
        e /= 2.0;
    }

    /* ループを抜ける一つ手前の値が計算機イプシロンです */
    printf("machine epsilon = 2^(%d)\n", -(i-1));
}
```

プログラム 2-5.c の中で, `e /= 2.0` の意味は、`e = e/2.0` です.

6　近似値と誤差

実際上の問題で，コンピュータなどを用いて数値計算を行う場合には，近似値を扱うのが普通です．そのため，必然的に誤差の問題が生じてきます．

6.1　誤差の限界

真の値 A の近似値 a に対して，近似値と真の値の差

$$a - A$$

を近似値 a の誤差といいます．その絶対値 $|\, a - A \,|$ を誤差ということもあります．また，ある小さい正の数 ϵ に対して

$$|\, a - A \,| \leq \epsilon$$

が成り立つとき，ϵ を近似値 a の誤差の限界といいます．このとき，真の値の存在する範囲は，次のように定まります:

$$a - A \leq A \leq a + \epsilon$$

例えば，$\sqrt{2}$ =1.41421356$\cdots\cdots$ の小数第 6 位を四捨五入した値 $a = 1.41421$ を $\sqrt{2}$ の近似値にとると

$$a - 0.000005 \leq \sqrt{2} < a + 0.000005$$

よって，a は不等式 $\mid a - \sqrt{2} \mid \leq 0.000005$ を満たし，この近似値の誤差の限界は 0.000005，すなわち 5×10^{-6} です．

2 つの地点 P, Q 間の距離の測定値は $100m$，別の 2 地点 R, S 間の距離の測定値は $42195m$ で，これらの誤差の限界はともに $10cm$ であったとします．このとき，誤差の限界は同じでも，P, Q 間の距離と，R, S 間の距離の大きさが違いすぎるため，測定値の近似の程度はかなり異なると考えられます．

このようなとき，真の値 A の近似値 a に対して，近似の程度をみるには，次の相対誤差を考えるといいです:

$$\frac{a - A}{A} \quad \text{または} \quad \mid \frac{a - A}{A} \mid$$

6.2 いろいろな誤差

コンピュータや電卓では，入力された 10 進数は内部で 2 進数に変換されて，計算などの処理が行われています．2 進数でも，10 進数の場合と同様に小数が考えられます．例えば， 2 進数の 101.011 とは，

$$1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 + 0 \times 2^{-1} + 1 \times 2^{-2} + 1 \times 2^{-3}$$

と表される数のことであり，10 進数で表すと 5.375 になります．10 進数の有限小数は，2 進数に変換すると無限小数になる場合があります．例えば，10 進数の小数 0.1 を 2 進数に変換すると

$$0.0001100110011001100110011001100\cdots\cdots$$

と，無限小数となります．コンピュータや電卓には，0 と 1 を記憶する場所が非常にたくさんありますが，無限ではなく限りがあり, そのため，上のような無限小数は，何桁目かで切り捨てや切り上げなどの，いわゆる「 丸める 」操作が施された有限小数として記憶されています．また，計算の途中に現れる数値にも「 丸める 」操作がしばしば施されます．このような操作のために生じる誤差を，丸め誤差といいます．例えば，電卓で $(1 \div 3) \times 3$ の計算を行うとき，丸め誤差のために，電卓によっては正しい答 1 ではなく，0.9999999 と表示されることがあります．

コンピュータは，非常に多い回数の計算を高速でおこないますが，無限回の計算を行うことはできません．弧度法で表された角 x の三角関数 $\sin x$ は

$$\sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \cdots\cdots$$

と，項が無限に続く式で表されます．例えば，この式を用いてコンピュータで $\sin x$ の値を求める場合，無限回の計算は行えないから，右辺を有限個の項で打ち切って，その近似値を求めることになります．このような事情のために生じる誤差を，打ち切り誤差という．

コンピュータを利用するときに生じる上のような誤差のほかに，有効数字の極端な桁落ちによる誤差もあります．例えば，有効数字が 5 桁の非常に近い 2 つの数

$$a = 2.1438, \qquad b = 2.1425$$

の差は $a - b = 1.3 \times 10^{-3}$ となり，有効数字が 2 桁に落ちます．そのため，この差に b を掛けた

$$(a - b) \times b = 1.3 \times 10^{-3} \times 2.1425 = 2.78525 \times 10^{-3}$$

では，意味のある有効数字は，最初の 2.7 の 2 つだけになります．

以上のような誤差は，1 回の計算では微小であっても，例えば，コンピュータが得意とする繰り返し計算などでは，計算回数が多くなるにしたがって誤差が累積し，無視できなくなる可能性があることを認識しておくことは大切です．例えば、2 次方程式の解を求めるプログラムを組むとき, 解の公式では誤差が影響するため, 解と係数の関係を用いて作成するのが最良です.

7 問題

レポート問題 **1** *(1) for* 文による繰り返しを使って，2^{10} を求めるプログラムを作成せよ．
(2) 付録のプログラムを参照して 2^n を求めるプログラムを作成せよ.

問題 **2**
計算機イプシロンの定義を

$$\epsilon > 0.0 \text{ となる最小の正の数 } \epsilon$$

と変えたとき、ϵ を求めるプログラムを作成し, 2-5.c の結果と異なることを確認せよ. 理由がわかった人は、レポートとして提出せよ.

8 付録

多項式の計算

繰り返しを使って, n 次の多項式

$$a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0$$

の値を出力するプログラムを作りましょう. 但し、$x, a_n, a_{n-1}, \cdots, a_0$ は実数です.

プログラムを作成する前に, その手順 (アルゴリズム) について考える必要があります. 紙の上では、単純に直接元を多項式に放り込む方法が浮かびますが, コンピュータがする計算量を意識するとあまりよくないことがわかります. 実際, 乗法・加法を合わせて, $\frac{n(n+1)}{2} + n$ 回計算が必要になります (何故か).

計算量を減らして、なおかつ有効なアルゴリズムとして、次のホーナー法があります.

いま, $b_n = a_n$, $b_{n-1} = b_n * x + a_{n-1}$, $\cdots$, $b_0 = b_1 * x + a_0$ と繰り返していくと, b_0 がもとめる多項式の値です. この場合, 乗法・加法に必要な計算量は $2n - 1 + n$ と前のものより少なくすみます. このように, プログラムを組む場合において, 問題を明確化し適切なアルゴリズムをまず作成しなければなりません. また, そのときの計算量を意識しなければならなりません. あまりに大きいと (例えば行列式の計算とか) 時間が途方にかかり, コンピュータがダウンしてしまいます. もう一つ大事なこととして計算誤差があり, それについては後で触れます.

次のプログラムを読んで下さい . 時間がありましたら, 作成し実行してみましょう.

```
#include <stdio.h>

main()
{
     double a, x, r;                          /* 変数 a,x,r が実数型であることを宣言 */
     int n, j;                                /* 同じく型宣言 */

     printf("次数 n を入力して下さい>>");
     scanf("%d", &n);                         /* n にデータを入力 */
     printf("求める値 x を入力して下さい>>");
     scanf("%lf",&x);                         /* x にデータを入力 */

     r= 0.0;                                  /*r の初期化*/
     for(j = n; j > = 0;j--){
      printf("係数 a[%d] を入力して下さい>>",j);
      scanf("%lf",&a);                         /* ホーナー法のアルゴリズム */
      r = x*r + a;
    }
  printf("求める値は、%f.\n", r);
}
```

上の例の式 3 には

```
    j--
```

が書かれています . これは j を 1 減らせという意味です . この部分は ,

```
j = j - 1;
```

あるいは，

```
j -= 1;
```

と書いても同じことです．

また, > =は, ≧の意味です. その他, =(等しい), \not=(等しくない), \leq(以下) は, それぞれ、==, !=, <=と書きます. これらを総称して関係演算子と呼びます. プログラムの中で演算子の記述間違いがあっても, コンパイルがとおることがありますので注意しましょう.