ce-15. 計算精度と誤差

（C プログラミング応用）（全１４回）

URL: https://www.kkaneko.jp/pro/c/index.html

金子邦彦

CC BY NC SA

int 型

- C言語では，整数データは int 型などで扱う

  - Microsoft Visual C++ では
  - int　　4バイト

- 扱える値に範囲がある
  - -2147483648 ～ 2147483647 (-231 ～ 231-1)
  - 符号で１ビット，残りで数を表現

オーバーフロー

- それぞれの型 (int, double など）ごとに「表現可能な範囲」が定まる

- 計算の途中で「表現可能な範囲」を越えてしまうこと　＝　オーバーフロー

少数点以下切り捨て

- 整数の演算において除算を行う場合、除算結果の小数点以下は切り捨てられる。
  - (整数)/(整数)の場合のみ、この小数点以下の切り捨てが行われる。
  - (整数）/（浮動小数点数）や(浮動小数点数）/（整数）の場合は浮動小数点数型として計算されるので結果に小数点以下の値も含まれる。
  - 異なる型を含む演算では、必要に応じて自動的に型変換が行われる。
- 予期しない結果を招かないためにも明示的に型変換をした方がよい。

整数データと浮動小数データの違い

	整数データ	浮動小数データ
変数宣言	int kingaku; int en;	double teihen; double takasa;
入力	scanf("%d", &kingaku);	scanf("%lf", &takasa);
出力	printf("kozeni: %d en¥n", en);	printf("takasa: %f ¥n", takasa);
四則演算	四則演算には，+, -, *, / を使う	四則演算には，+, -, *, / を使う
剰余	en = kingaku%1000;	z = fmod(x,y);

1/2 の値は 0

```
#include "stdio.h"
int main()
{
  int ch;
  double r;
  r = 1 / 2;
  printf("r = %f¥n ", r );
  ch = getchar();
  ch = getchar();
}
```

このプログラムの実行結果は，直感とは一致しないかも知れない

```
r = 0.000000
```

右辺に整数の変数しか登場しないので，右辺は整数の精度で計算される

1.0/2.0 の値は 0.5

```
#include "stdio.h"
int main()
{
  int ch;
  double r;
  r = 1.0 / 2.0;
  printf("r = %f¥n ", r );
  ch = getchar();
  ch = getchar();
}
```

「 1 / 2 」 と 「 1.0 / 2.0 」 は，意味が違う

```
r = 0.500000
```

右辺に浮動小数の変数が登場するので，右辺は浮動小数の精度で計算される

1/2 と 1.0/2.0 の違い

- 1/2 は，整数と整数の割り算
  - 文法的には 「2000/30 (値は66） 」と書くのと同じ
  - 1/2 の値は 0 （やはり整数）

- 1.0/2.0 は，浮動小数と浮動小数の割り算
  - 1.0/2.0 の値は 0.5 （浮動小数）

0除算

- ある数を0で割る(0除算）と、プログラムは異常終了する。
  - 調和平均を求める場合、入力データに0が含まれると、0除算が起こり異常終了する。
- 割り算を行う場合は割る数が0でないことを調べてから行う。
  - 入力データが0であることを検知して、調和平均の計算を止めるようする。

浮動小数点数の演算

計算機内での数値の表現

- 数値の桁数は変数の型によって決まっている。
  - int型　　32ビット (Microsoft Visual C++の場合）

$50_{(10)} = 00000000000000000000000000110010_{(2)}$　int 型

負の数

- 2の補数表現であらわす。
  - 絶対値を決められたビット数で表す
  - 0、1を反転する
  - 1を加える

short int 型の例　(-50)

$50_{(10)}$ = $0000000000110010_{(2)}$

0、1を反転する

$1111111111001101_{(2)}$

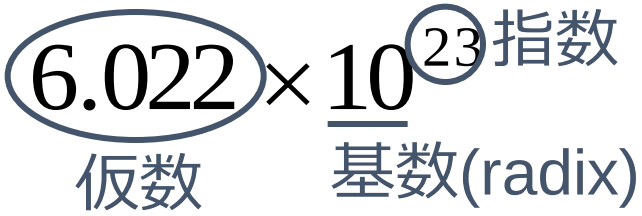
1を加える

$-50_{(10)}$ = $1111111111001110_{(2)}$

浮動小数点数

- 整数（正または負または０）あるいは小数付きの数
  - 0.5 (= 0.5 × 100)

- 仮数(mantissa)部と指数(exponent)部



仮数の整数の桁は1桁で0以外

10進数の0.1を2進数で表すと…

- 無限小数になる。
  - 0.1(10)=
    0.0001100110011001100110011001…(2)
    1.10011001100110011001100…×2-4(2)
- 23ビットで表すために、小数第24桁目で0捨1入をするので、誤差が生じる。(丸め誤差)

0 01111011 10011001100110011001100
= 9.9999994039535522×10^{-2}

0 01111011 10011001100110011001101
= 1.0000000149011612×10^{-1}

10^{-9}程度の誤差

丸め誤差の集積（情報落ち）

- 丸め誤差が集まると，誤差が増える

```
#include "stdio.h"
int main()
{
        double x;
        double s;
        int ch;

        x = 0.0001;
        s = 10000;
        for (int i = 0; i < 100000000; i++) {
          s = s + x;
        }
        printf("1 に 0.0001を100000000回足すと%fになります。¥n", s);
  ch = getchar();
  ch = getchar();
}
```

出てくる値は 20000 にならない

浮動小数点でのループの終了判定

- ループの制御を浮動小数点型変数を用いて行う場合は注意が必要。
  - 変数 x を 0.001 きざみで0～1まで変化させて何かを処理する場合（面積の近似計算など）
  - 継続条件式を x <= 1 としても、1,000 回目のループが実行されるかどうかは確認が必要。
    - 0.001 を 1,000 回加えても丸め誤差の集積で1以外の数になっている可能性がある。
    - 1より小さい場合は処理が実行される。
    - 1より大きい場合は処理が実行されない。
  - ループ回数で制御することで回避する。

仕様

- 2次方程式を ax2+bx+c=0 として、係数a、b、cを入力してもらう。
- 2次方程式の解の公式に従って、浮動小数点数解があるとき、2つの解x1とx2を計算し、その値を出力する。そのときの(ax2+bx+c)の値も表示する。

$$x1 = \frac{-b + \sqrt{b^2 - 4ac}}{2a} \qquad x2 = \frac{-b - \sqrt{b^2 - 4ac}}{2a}$$

プログラム

```
double a, b, c, d, x1, x2;

printf("Please input a b c : ");
scanf("%lf %lf %lf", &a, &b, &c);

d = b * b - 4 * a * c;      /* 判別式の計算 */
if(d >= 0){                 /* 判別式が正のとき、解を計算 */
      x1 = (-b + sqrt(d)) / (2.0 * a);
      x2 = (-b - sqrt(d)) / (2.0 * a);
      printf("x1 = %17.10e  (%17.10e)¥n",x1,a*x1*x1+b*x1+c);
      printf("x2 = %17.10e  (%17.10e)¥n",x2,a*x2*x2+b*x2+c);
}
else{                       /* 判別式が負のとき、虚数解と表示 */
      printf("x1 = 虚数解¥n");
      printf("x2 = 虚数解¥n");
}
```

プログラムのテスト

- 係数の値として下の値を入力した場合の結果を観察すると、
  - (1 –2 1)
    - x1 = 1.0000000000e+000 (0.0000000000e+000)
    - x2 = 1.0000000000e+000 (0.0000000000e+000)
  - (1 –1000 1)
    - x1 = 9.9999900000e+002 (1.1641532183e-010)
    - x2 = 1.0000010000e-003 (-2.0621060415e-011)
  - (1 –1000000 1)
    - x1 = 1.0000000000e+006 (0.0000000000e+000)
    - x2 = 1.0000076145e-006 (-7.6144923700e-006)
  - (1 –100000000 1)
    - x1 = 1.0000000000e+008 (1.0000000000e+000)
    - x2 = 7.4505805969e-009 (2.5494194031e-001)

方程式の左辺の値

x2の左辺の値が0から離れていく（誤差が大きくなる）

桁落ち

- 有効数字の桁数が減少すること
- 先の例の(1 –1000000 1)では…
  - |b| → 0 412 e848000000000
  - √D → 0 412 e847fffffbce4
  - 差 → 0 412 000000000431c
    0 3ec 0c70000000000

正規化

有効な桁が12ビットになる

もし桁数が限られていなければ、後ろの0が続く40ビットにも値が入るはず。

桁落ち

- 絶対値がごく近い2数を足したり引いたりして、結果の絶対値が小さくなるような計算を行うと桁落ちが起こる。
- 絶対値が小さくなった分だけ相対誤差が大きくなり、有効数字が減る。
- 式の変形などで桁落ちを回避することができることがある。

2次方程式の解で桁落ちを防ぐ

- |b|+√D の方だけを使って1つの解を求める。
  - b≧0 の場合

$$x1 = -\frac{|b| + \sqrt{b^2 - 4ac}}{2a}$$

  - b＜0 の場合

$$x1 = \frac{|b| + \sqrt{b^2 - 4ac}}{2a}$$

- もう1つの解は、解と係数の関係から求める。

$$x2 = \frac{c}{a \cdot x1}$$

改良版プログラム

解と係数の関係より

```
double a, b, c, d, x1, x2;

printf("Please input a b c : ");
scanf("%lf %lf %lf", &a, &b, &c);

d = b * b - 4 * a * c;      /* 判別式の計算 */
if(d >= 0){                 /* 判別式が正のとき、解を計算 */
      if(b >= 0) x1 = -(b + sqrt(d)) / (2.0 * a);
      else       x1 = (-b + sqrt(d)) / (2.0 * a);
      x2 = c / (a * x1);
      printf("x1 = %17.10e  (%17.10e)¥n",x1,a*x1*x1+b*x1+c);
      printf("x2 = %17.10e  (%17.10e)¥n",x2,a*x2*x2+b*x2+c);
} else{                     /* 判別式が負のとき、虚数解と表示 */
      printf("x1 = 虚数解¥n");
      printf("x2 = 虚数解¥n");
}
```