プログラミング言語（2）

コンピュータリテラシ

2016年5月20日

建部修見

数の表現

- 負の数の表し方
- 符号付き整数
  - 最上位ビットを符号として用いる
  - 8bitでは-127～+127
- 1の補数
  - 負の数はビットを反転させる
  - 0が二通り
  - 正の数と負の数を足したとき1を加える必要がある
    - 繰り上がった1（キャリー）を足す
  - 最上位ビットで正負が分かる
  - 8bitでは-127～+127

0	0	0	0	0	0	0	1	1
1	1	1	1	1	1	1	0	-1

数の表現(2)

- 2の補数
  - 8bitのとき、負の数は2^8から絶対値を引く
    - 1の補数に1を加える
  - 正の数と負の数の和は繰り上がりを無視
  - 8bitでは-128～+127
- バイアス表現
  - 8bitのとき127をバイアスとして加える

0	0	0	0	0	0	1	0	2
0	0	0	0	0	0	0	1	1
0	0	0	0	0	0	0	0	0
1	1	1	1	1	1	1	1	-1
1	1	1	1	1	1	1	0	-2

1	0	0	0	0	0	0	0	1
0	1	1	1	1	1	1	1	0
0	1	1	1	1	1	1	0	-1

演習（1）

- C言語で負の数はどのように表現されるか調べよう。たとえば以下のプログラムを実行した結果から何がわかるか？
  - %08xは16進数8桁で表示するフォーマットである

```
#include <stdio.h>

int
main()
{
        int i;

        for (i = -4; i < 5; i++)
                printf("%08x = %d¥n", i, i);
        return (0);
}
```

浮動小数点数

- IEEE 754
  - 単精度32bit（符号1bit、指数8bit、仮数23bit）
    - (-1)^s 2^(e-127) (1 + m)
  - 倍精度64bit（符号1bit、指数11bit、仮数52bit）
    - (-1)^s 2^(e-1023) (1 + m)
  - e=255 or 2047のとき
    - 仮数が0なら正か負の無限大
      - 1.0 / 0.0, -1.0 / 0.0
    - 仮数が0でなければ非数NaN（Not a Number）
      - 0.0 / 0.0、$\sqrt{-1}$
  - e=0のときは非正規化数
    - (-1)^s 2^(-126) (0 + m)
    - (-1)^s 2^(-1022) (0 + m)

演習（2）

- 以下を実行してみよう
  - %g は倍精度浮動小数点を表示するフォーマット
  - sqrt()は平方根を計算する関数で、math.hで宣言されている

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

int
main()
{
        printf("%g¥n", 0.0 / 0.0);
        printf("%g¥n", 1.0 / 0.0);
        printf("%g¥n", sqrt(-1.0));
        return (0);
}
```

誤差

- 丸め誤差
  - $0.1_{(10)} = 0.0001100110011001\ldots_{(2)}$
  - 最後の桁が0捨1入（丸め）される
    - 切り捨て、切り上げ、0方向に丸めなどモードがある
- 情報落ち
  - 絶対値の大きい数と小さい数を加減算して、小さい数が無視される
    - 1e+8 + 1e-9
- 桁落ち
  - ほぼ等しい値を減算すると有効数字が減る
    - 1e+8 + 1e-8 - 1e+8
- 打ち切り誤差
  - 計算を途中でやめることによる誤差
    - $\sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \cdots$

演習（3）

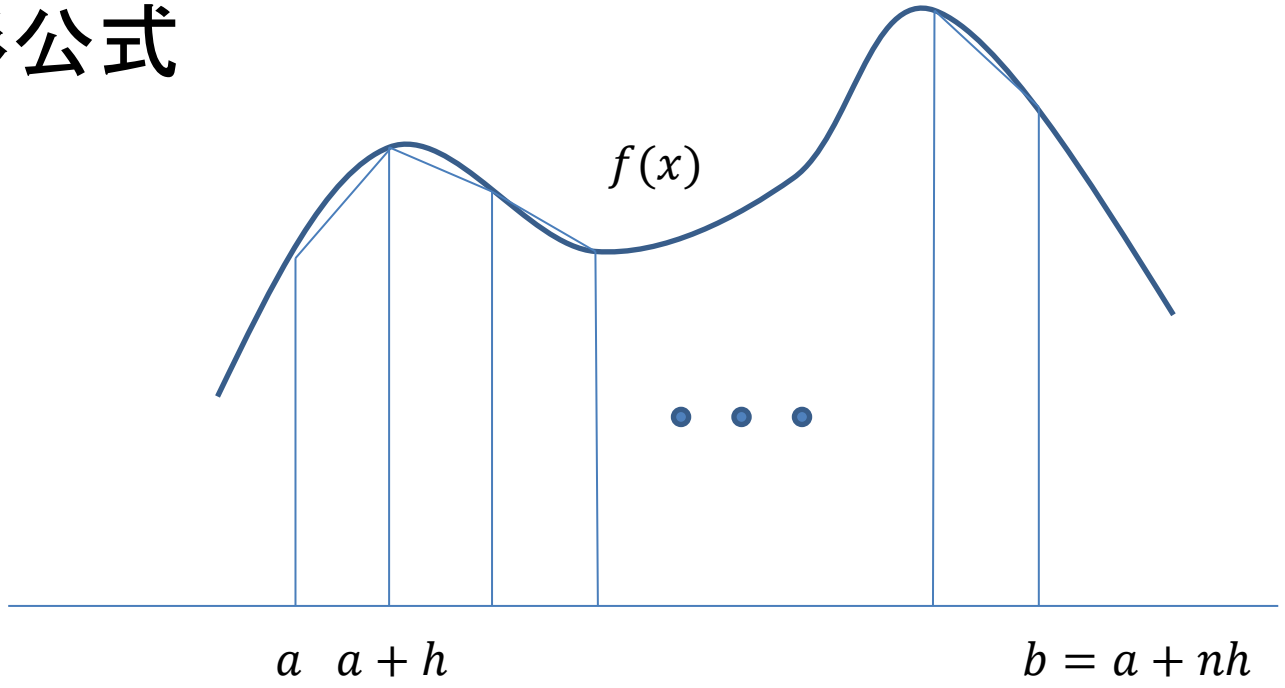
- 以下を実行して、それぞれ何が起きているか説明してみよう

```
#include <stdio.h>

int
main()
{
        printf("%g¥n", 0.1 * 3 – 0.3);
        printf("%g¥n", 1e+8 + 1e-9);
        printf("%g¥n", 1e+8 + 1e-9 – 1e+8);
        printf("%g¥n", 1e+8 + 1e-8 - 1e+8);
        printf("%g¥n", 1e+8 - 1e+8 + 1e-8);
        return (0);
}
```

数値積分

- 台形公式



$$S=\sum_{i=0}^{n-1}\frac{f(a+ih)+f(a+(i+1)h)}{2}h=\frac{f(a)+f(b)}{2}h+\sum_{i=1}^{n-1}f(a+ih)h$$

演習（４）

- $\int_0^1 \frac{4}{1+t^2}\,dt$を台形公式で計算するプログラムを作成し、刻み幅を10から1,000,000,000まで10倍ずつ増やしながら計算してみよう
  - 16桁の倍精度小数点数を表示するprintfのフォーマットは %.16g である
  - 次のスライドにプログラム例があるので参考にしても良い

```
#include <stdio.h>

double
f(double x)
{
        return (4.0 / (1.0 + x * x));
}

double
integral(double a, double b, int n)
{
        double s, h = (b - a) / n;
        int i;

        s = .5 * (f(a) + f(b));
        for (i = 1; i < n; i++)
                s += f(a + i * h);
        return (s * h);
}

int
main()
{
        int i;

        for (i = 10; i <= 1000000000; i = i * 10)
                printf("%.16g¥n", integral(0.0, 1.0, i));
        return (0);
}
```

オプション演習

- OpenMPを用いるとマルチコアで並列実行できる。そのためには
  - integral()の以下の部分の直前に一行加える

```
        for (i = 1; i < n; i++)
                s += f(a + i * h);
```

```
#pragma omp parallel for reduction(+:s)
        for (i = 1; i < n; i++)
                s += f(a + i * h);
```

  - コンパイル時に-fopenmpオプションを追加する
  - COINS環境ではccはバージョンが古くサポートしていないためclang-mp-3.8を用いる

    $ clang-mp-3.8 -O –fopenmp …

  - timeコマンドで時間を比較してみよう