数値計算
第一回課題

1G01P035-9 木村浩章

2003 年 4 月 20 日 日曜日

1　課題内容

1.1　課題 1

a を IEEE754 に従う double とする。a の符号、指数部、仮数部を bit パターンとして書き出す C プログラムを作れ。

1.2　課題 2

u, v を IEEE754 の正規化 2 進浮動小数点数か 0 とする。このとき、$|u| \geq |v|$ でオーバーフローとアンダーフローが起きないとして、

- $x = fl(u + v)$
- $w = fl(u - x)$
- $y = fl(w + v)$

と計算すると、厳密に

$$u + v = x + y$$

が成り立つ事を示せ。ただし、$fl()$ はかっこ内の演算を最近点での丸めの下での浮動小数点演算で行うものとする。

2　解答

2.1　課題 1

数値計算のサイトに貼られていた一昨年のレポートをざっと見ると、double の値を char で 8 つに分けて見ていっているようである。しかし、この方法だと一つ問題がある。**リトルエンディアンである Intel 系のプロセッサだと問題がないが、それ以外のプロセッサ（SPARC や Power 系列など）はビッグエンディアンであるため、8bit ずつのかたまりが逆に表示されてしまう**のである。

多くの学生が Intel 系列のプロセッサを使っているためあまり問題にならないかもしれないが、一部にはそれ以外のプロセッサを使っているものもいるし（私もその一人である）、エンディアンの問題を意識してコードを書いていた方が良いだろうと思う。

実際一昨年のレポートを参考にした人にも、何故 char で分ける時にわざわざ逆方向に for 文で参照しているか分からない人も多いと思われる。順方向だとうまくいかないことからエンディアンを意識する…という人もおり、そのような人には SPARC のある端末室での実行を勧めたりした。

解決法は単純で、char のポインタを使わずに、GCC における 64bit の整数型である long long のポインタを使えば良い。こうすればエンディアンの違いを気にせずに書く事が出来る。

long long にキャストしたポインタの各ビットを見ていくために、シフト演算子とビット演算子を利用する。ターゲットとなるビットを 1 の位までシフトし、それと 1 の&をとることによって、ビットの値を計算している。

これだけ分かればすぐにコードはかけるのだが、C のコードを書くにあたって一つ試してみたい事があった。それは、1999 年に “ISO/IEC 9899:1999 - Programming Language C” として新しく策定された機能である。その機能は多岐に渡るため、詳しい事は http://seclan.dll.jp/c99d/などを参照していただきたい。ここで使いたい機能は C++や Java ではおなじみの変数宣言方法である。

Java などを先に学んだものにとって、C の変数の宣言はまだるっこしい。コードブロックの先頭で全てまとめて変数を宣言しなければならないため、for 文で使う変数なども頭で宣言しなければならなかったし、それ故変数のスコープと言う概念もないと言って良いくらいである。

99 年に策定された C では、Java や C++と同じように変数を宣言できるようになっている。GCC も Ver. 3 から対応しているようで、起動オプションとして “-std=c99” を付け加える事によって有効となる。最近このことを知り、使ってみたくなった次第である。

実際には、以下のようなコードになった。ここでは for 文中での i の宣言や、if〜else 文中での double 型の変数宣言をしている。変数のスコープはコードを書く上でとても便利な機能であるので、このような仕様変更は素直に歓迎したい。

丸めモードによる値の差は今回の課題では蛇足だが（本来的には display 関数だけで良い）、一昨年のレポートを参考に（数値に関してはそのままである）実装してみた。とは言っても、丸めを行うために Mac OS X 独自のヘッダファイルをインクルードしてしまったため、これによって x86 系プロセッサではコンパイルできなくなってしまったが…。

```
#include <stdio.h>
#include <architecture/ppc/fenv.h>
#include <string.h>

void display(double data){
    unsigned long long *bytes;

    bytes = (unsigned long long *)&data;

    for ( int i = 0; i < 64; i++){
        printf("%d", (int)((*bytes >> (63-i)) & 1));
        if (i == 0 || i == 11)
            printf(" ");
    }
```

```
    printf("\n");
}

int main (int argc, const char *argv[]) {

//round test mode
    if (argc == 2 && strcmp( *(++argv), "roundTest" ) == 0){

        volatile double a, b, c, d;
        for (int i = 0; i < 4; i++){
        //round mode
            fesetround(i);
            switch(i){
                case 0:
                    printf("nearest mode\n");
                    break;
                case 1:
                    printf("toward_zero mode\n");
                    break;
                case 2:
                    printf("toward_plus mode\n");
                    break;
                case 3:
                    printf("toward_minus mode\n");
                    break;
            }

            printf("    a = ");
            a = 1.0/2/2/2/2/2/2/2/2/2/2/2
            /2/2/2/2/2/2/2/2/2/2/2
            /2/2/2/2/2/2/2/2/2/2/2
            /2/2/2/2/2/2/2/2/2/2/2
            /2/2/2/2/2/2/2/2/2/2/2
            /2/2/2;
            display(a);

            printf("    b = ");
            b = a+1.0/2/2/2/2/2/2/2/2/2/2/2
            /2/2/2/2/2/2/2/2/2/2/2
            /2/2/2/2/2/2/2/2/2/2/2
            /2/2/2/2/2/2/2/2/2/2/2
            /2/2/2/2/2/2/2/2/2/2/2
            /2/2/2/2/2/2/2/2/2/2/2
```

```
    /2/2/2/2/2/2/2/2/2/2
    /2/2/2/2/2/2/2/2/2/2
    /2/2/2/2/2/2/2/2/2/2
    /2/2/2/2/2/2/2/2/2/2
    /2/2/2/2/2;
    display(b);

    printf("    c = ");
    c = a+a/2/2/2/2/2/2/2/2/2/2/2;
    display(c);

    printf("    d = ");
    d = c+1.0/2/2/2/2/2/2/2/2/2/2
    /2/2/2/2/2/2/2/2/2/2
    /2/2/2/2/2/2/2/2/2/2
    /2/2/2/2/2/2/2/2/2/2
    /2/2/2/2/2/2/2/2/2/2
    /2/2/2/2/2/2/2/2/2/2
    /2/2/2/2/2/2/2/2/2/2
    /2/2/2/2/2/2/2/2/2/2
    /2/2/2/2/2/2/2/2/2/2
    /2/2/2/2/2/2/2/2/2/2
    /2/2/2/2/2;
    display(d);

    printf("a + 1 = ");
    a = a+1.;
    display(a);

    printf("b + 1 = ");
    b = b+1.;
    display(b);

    printf("c + 1 = ");
    c = c+1.;
    display(c);

    printf("d + 1 = ");
    d = d+1.;
    display(d);

    printf("\n");
}
```

```
    }else{
        double data;

        printf("Please insert number :");
        scanf("%lf", &data);

        display(data);
    }
    return 0;
}
```

2.1.1 実行結果

まず、“./a.out roundTest” として実行した場合の結果を記す。これは、一昨年のデータを参考に、PowerPC G4 550MHz において 4 種類の丸めモードで演算を実行した結果である。

```
hiroaki% ./a.out roundTest
nearest mode
    a = 0 01111001010 0000000000000000000000000000000000000000000000000000
    b = 0 01111001010 0000000000000000000000000000000000000000000000000001
    c = 0 01111001010 0000000000100000000000000000000000000000000000000000
    d = 0 01111001010 0000000000100000000000000000000000000000000000000001
a + 1 = 0 01111111111 0000000000000000000000000000000000000000000000000000
b + 1 = 0 01111111111 0000000000000000000000000000000000000000000000000001
c + 1 = 0 01111111111 0000000000000000000000000000000000000000000000000001
d + 1 = 0 01111111111 0000000000000000000000000000000000000000000000000001

toward_zero mode
    a = 0 01111001010 0000000000000000000000000000000000000000000000000000
    b = 0 01111001010 0000000000000000000000000000000000000000000000000001
    c = 0 01111001010 0000000000100000000000000000000000000000000000000000
    d = 0 01111001010 0000000000100000000000000000000000000000000000000001
a + 1 = 0 01111111111 0000000000000000000000000000000000000000000000000000
b + 1 = 0 01111111111 0000000000000000000000000000000000000000000000000000
c + 1 = 0 01111111111 0000000000000000000000000000000000000000000000000000
d + 1 = 0 01111111111 0000000000000000000000000000000000000000000000000000

toward_plus mode
    a = 0 01111001010 0000000000000000000000000000000000000000000000000000
    b = 0 01111001010 0000000000000000000000000000000000000000000000000001
    c = 0 01111001010 0000000000100000000000000000000000000000000000000000
    d = 0 01111001010 0000000000100000000000000000000000000000000000000001
a + 1 = 0 01111111111 0000000000000000000000000000000000000000000000000001
```

```
b + 1 = 0 01111111111 00000000000000000000000000000000000000000000000000000001
c + 1 = 0 01111111111 00000000000000000000000000000000000000000000000000000001
d + 1 = 0 01111111111 00000000000000000000000000000000000000000000000000000001

toward_minus mode
    a = 0 01111001010 00000000000000000000000000000000000000000000000000000000
    b = 0 01111001010 00000000000000000000000000000000000000000000000000000001
    c = 0 01111001010 00000000001000000000000000000000000000000000000000000000
    d = 0 01111001010 00000000001000000000000000000000000000000000000000000001
a + 1 = 0 01111111111 00000000000000000000000000000000000000000000000000000000
b + 1 = 0 01111111111 00000000000000000000000000000000000000000000000000000000
c + 1 = 0 01111111111 00000000000000000000000000000000000000000000000000000000
d + 1 = 0 01111111111 00000000000000000000000000000000000000000000000000000000
```

どうやらこれを見る限り、うまく丸めを行っているようである。

次に、今回の課題である “浮動小数点を入力し、そのビットパターンを出力する” テストをしてみる。

```
hiroaki% ./a.out
Please insert number :0
0 00000000000 00000000000000000000000000000000000000000000000000000000

hiroaki% ./a.out
Please insert number :-0
1 00000000000 00000000000000000000000000000000000000000000000000000000

hiroaki% ./a.out
Please insert number :0.1
0 01111111011 10011001100110011001100110011001100110011001100110011010

hiroaki% ./a.out
Please insert number :0.5
0 01111111110 00000000000000000000000000000000000000000000000000000000
```

例が少ないが、どうやらこれもうまくいっているようである。

2.1.2 実行環境など

コードは Mac OS X 上で書いた。Macintosh のプロセッサは課題 1 の解答でも指摘のとおり、ビッグエンディアン方式の PowerPC である。コンパイラとして、Mac OS X の Developer Tools に付属の GCC Ver. 3.1 を使用した。

リトルエンディアンでの実行環境として、Pentium 4 の AT 互換機で RedHat Linux 8 を動かし、上と同じように GCC でコンパイルして確かめた（ただし、丸めモードに関しては実行していない)。どうやらきちんとうまくいっているようである。

コマンドは単純に、“gcc -std=c99 main.c” である。

2.2 課題 2

感覚的には理解できているような気がしないでもないのだが、それをきちんと証明する事が出来ない。もう少し考えてみたいと思う。

3 参考

プログラミング言語 C の新機能 http://seclan.dll.jp/c99d/にあるウェブサイトで、99 年策定の新しい C 言語の仕様を知る事が出来る。本家 GNU でも、英語ながら該当ページがある (http://gcc.gnu.org/gcc-3.1/c99status.html)。

HAPPY Macintosh Developing TIME! http://homepage.mac.com/mkino2/にあるウェブサイトで、GCC の “std=c99” オプションがあることを BBS にて知る事が出来た。

構造化コンピュータ構成 一年次の情報科学概論の授業のときの教科書で、主にコンピュータアーキテクチャについて詳しく書かれている。今回はエンディアンの違いについて参考にした。

2001 年度数値計算第一回課題解説 float.h における丸めモードの設定や、IEEE754 規格を満たす bit パターンのサンプルなどを参考にした。

4 感想

課題 1 は、bit 列を吐き出すだけではあまりに一瞬で終わってしまうので、一昨年の課題同様丸めモードについても調べてみた。CPU が違い、OS も Mac OS X ということで、調べてみる価値があると思ったのである。

が、この丸めモード変更関数が記述されているヘッダファイルが見つからず、意外なところで苦労してしまった。/usr/include/内のヘッダファイル全ての内容を検索にかけ、どうやら/usr/include/architecture/ppc/fenv.h だと気付くまでに 1 時間近くかかってしまった…。

IEEE754 規格がハードウェア的に実装されているというのはこの講義で初めて知ったが、その丸めモードを C で変更できると言うのはなかなか面白く、ほんのひとかけらではあるものの、ハードウェアのプリミティブな部分にアクセス出来るのだという驚きも実感できた。

課題 2 はどうしてもうまく説明できないため、出来るだけ早めにリベンジしようと思う…。