浮点数运算模拟

3180103772 张溢弛

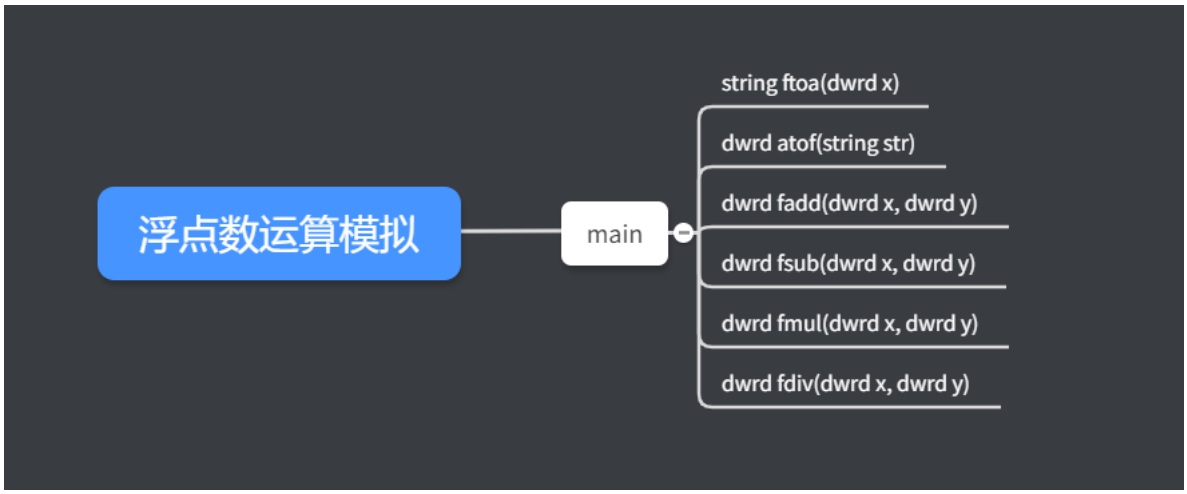
0.实验描述

浮点数的表示与算术运算算法分析，要求理论推导与程序模拟。 算术运算包括字符串到浮点数、浮点数到字符串的转换，加、减、乘、除四则运算等。要有说明文档，包括算法证明、程序框图、使用方法、特殊处理(溢出、数位扩展)、实例分析等等。字符串转换可能稍难。

```
typedef unsigned int dwrd;
char* ftoa(dwrd);
dwrd atof(char*);
dwrd fadd(dwrd, dwrd);
dwrd fsub(dwrd, dwrd);
dwrd fmul(dwrd, dwrd);
dwrd fdiv(dwrd, dwrd);
```

1.程序结构

- 本程序的结构图如下



2.算法分析

- 加法和减法
  - 浮点数加法的运算规则为小阶向大阶看齐，根据浮点数的格式提取出ex，ey分别为浮点数x和y的阶码，而mx，my为尾数，将小阶对应的尾数右移后向大阶看起，然后将位数相减得到新的尾数，相减后进行重新的规格化。
  - 减法可以将被减数转化为相反数相加，只要改变被减数的符号位后再进行加法就可以

```
dwrd fadd(dwrd x, dwrd y)
{
    if(x == 0) return y;
    else if(y == 0) return x;
    if((x << 1) < (y << 1)) return fadd(y , x);
    dwrd ex = (x >> 23) & 0xFF;
```

```
    dwrd ey = (y >> 23) & 0xFF;
    dwrd mx = (x & 0x7FFFFF) | 0x800000;
    dwrd my = (y & 0x7FFFFF) | 0x800000;
    my >>= (ex - ey);
    if((x ^ y) & 0x80000000) {
    mx -= my;
    if(mx == 0) return 0;
    mx <<= 1;
    ex--;
    }
    else {
    mx += my;
    if (mx == 0) return 0;
    while (mx & 0xFF000000) {
        mx >>= 1;
        ex++;
    }
    }
    return (x & 0x80000000) | (ex << 23) | (mx & 0x7FFFFF);
}

dwrd fsub(dwrd x, dwrd y)
{
    return fadd(x, y ^ 0x80000000);
}
```

- 乘法
  - 浮点数乘法的运算规则为
    - $E(x) = x + M, E(xy) = x + y + m = E(x) + E(y) - M$
    - 根据真值表，浮点数乘法运算的符号位是两个因数最高位的异或
  - 乘法的尾数即为两个浮点数的尾数值相乘，但是由于一个浮点数的尾数有24位，两个浮点数相乘的时候会产生48位的尾数，容易导致溢出，所以只保留相乘的前24位

```
dwrd fmul(dwrd x, dwrd y)
{
    if(x == 0 || y == 0) return 0;
    dwrd ex = (x >> 23) & 0xFF;
    dwrd ey = (y >> 23) & 0xFF;
    dwrd mx = (x & 0x7FFFFF) | 0x800000;
    dwrd my = (y & 0x7FFFFF) | 0x800000;
    dwrd s = (x >> 31) ^ (y >> 31);
    dwrd e = ex + ey - 127;
    dwrd result = 0;
    while(my != 0) {
    result >>= 1;
    if(my & 1) result += mx;
    my >>= 1;
    }
    while(result & 0xFF000000) {
    result >>= 1;
    e++;
    }
    return (s << 31) | (e << 23) | (result & 0x7FFFFF);
}
```

- 除法
  - 除法的阶码运算规则如下
    - $E(x/y) = x - y + M = E(x) - E(y) + M$
    - 尾数的除法采用恢复除数的除法
      - 首先算出阶码
      - 然后对尾数做出除法
      - 为了处理最高位商0导致移位失败，我们在开始时通过不断地位移使得mx>my

```
dwrd fdiv(dwrd x, dwrd y)
{
    if(x == 0) return 0;
    if (y == 0) return (x & 0x80000000) | 0x7F800000;
    dwrd ex = (x >> 23) & 0xFF;
    dwrd ey = (y >> 23) & 0xFF;
    dwrd mx = (x & 0x7FFFFF) | 0x800000;
    dwrd my = (y & 0x7FFFFF) | 0x800000;
    dwrd s = (x >> 31) ^ (y >> 31);
    dwrd e = ex - ey + 127;
    dwrd result = 0;
    while(mx < my) {
    mx <<= 1;
    e--;
    }
    while(!(result & 0xFF800000)) {
    if(mx < my) result <<= 1;
    else {
        mx -= my;
        result = (result << 1) | 1;
    }
    mx <<= 1;
    }
    return (s << 31) | (e << 23) | (result & 0x7FFFFF);
}
```

- ftoa的实现
  - 浮点数转换成字符串的过程比较麻烦，首先要考虑特殊情况，如果是0则直接输出0，如果阶码是255则表示无穷大，用“INF”来表示
  - 然后考虑符号位，根据最高位判断符号位添加到字符串中
  - 接下来分两种情况讨论，看x是大于1的数还是小于1的数
    - 如果大于1，需要分成整数部分和小数部分进行转换
    - 如果小于1，则不需要管整数的部分

```
string ftoa(dwrd x)
{
    string result;
    dwrd sx = x >> 31;
    dwrd ex = (x >> 23) & 0xFF;
    dwrd mx = (x & 0x7FFFFF) | 0x800000;
    if(ex == 0) {
    if(mx == 0) result = "0";
    else {

```

```
    }
    }
    else if(ex == 0xFF) {
    if(mx == 0) {
        if(!sx) result = "INF";
        else result = "-INF";
    }
    else result = "NaN";
    }
    else if(ex >= 127) {
    dwrd e = ex - 127;
    string integer = "1";
    string decimal = ".";
    int i = 0;
    while(i < 24) {
        if(i < e) {
        if((mx >> (22 - i)) & 1) integer += "1";
        else integer += "0";
        }
        else {
        if((mx >> (22 - i)) & 1) decimal += "1";
        else decimal += "0";
        }
        i++;
    }
    if(!sx) result = trans1(integer) + "." + trans2(decimal);
    else result = "-" + trans1(integer) + "." + trans2(decimal);
    }
    else {
    dwrd e = 127 - ex;
    string decimal = ".";
    int i = 0, j = 0;
    while(i < e) {
        decimal += "0";
        i++;
    }
    while(j < 24) {
        if((mx >> (22 - j)) & 1) decimal += "1";
        else decimal += "0";
        j++;
    }
    if(!sx) result = "0." + trans2(decimal);
    else result = "-0." + trans2(decimal);
    }
    return result;
}
```

- atof的实现
    - 在将字符串转换成浮点数之前，首先可以通过正则表达式来验证是否是一个合法的浮点数
    - 为了能够分别计算整数部分和小数部分，需要将字符串分成符号位，整数和小数三个部分，拆分后需要考虑浮点数为0的特殊情况，此时应该直接返回0
    - 整数部分的转化思路为除2取余，得到整数部分的二进制串，小数部分的转换比较麻烦，通过字符串切割得到小数部分将其当作整数转换成二进制一样的操作来转换，最后进行拼接

```
dwrd atof(string str)
{
```

```
    string binary;
    dwrd s, e, m = 0;
    if(str[0] == '-') {
    s = 1;
    str.erase(str.begin());
    }
    else s = 0;
    int i = 1, temp = (int)str.find('.');
    if(temp == -1) binary = form1(str, 0, (int)str.length() - 1);
    else {
    binary = form1(str, 0, temp) + '.' + form2(str, temp + 1,
int(str.length()));
    }
    if(binary.find('.') != -1) {
    e = (dwrd)binary.find('.') + 126;
    binary.erase(binary.find('.'), 1);
    }
    else e = (dwrd)binary.find('.') + 126;
    int len = (int)binary.length();
    while(i < 24 && i < len) {
    if(binary[i] == '1') m += 1 << (23 - i);
    i++;
    }
    return (s << 31) | (e << 23) | m;
}
```

3.实验结果

- 根据提示在一行中输入两个浮点数
- 两个小数部分为0的整数

```
Please input 2 float number in a line
1.0 2.0
num1 =                                    1.0000000000000000000
num2 =                                    2.0000000000000000000
add =                                    3.0000000000000000000
sub = -                                   1.0000000000000000000
mul =                                    2.0000000000000000000
div = 0.0000000000000000000
```

- 普通的两个数

```
Please input 2 float number in a line
3.4 5.6
num1 =                                     3.0000000002340586730
num2 =                                     5.0000000000899963036
add =                                     8.0000000003240549766
sub = -                                    2.0000000003781210424
mul =                                    19.0000000001312362178
div = 0.0000000004110185660
```

- 包含负数

```
-1.2 2.5
num1 = -                                    1.0000000003781210424
num2 =                                     2.0000000001156841472
add =                                     1.0000000002597465166
sub = -                                    3.0000000000643084600
mul = -                                    2.0000000003240549766
div = -0.0000000002533156706
```

4.心得体会

- 这次实验通过编程模拟计算机中的浮点数运算，我收获颇丰，更透彻理解了计算机中的一些底层工作原理，希望在下周开始的CPU工作原理等内容中能学到更多东西。