浮点数运算

- 程序介绍

  本次实验实现了要求的六个子程序。用户通过在控制台输入两个浮点数字符串，可以首先通过atof函数得到浮点数，然后进行四则运算，最后得到ftoa的结果。

  因为C语言内置有atof函数，这里我把名字改成了myAtof。

- 运算算法和证明
  - 基本程序结构

    ```
    union fd;
    typedef union fd FloatUnion;
    typedef unsigned int word;   //16-bit
    typedef unsigned long dwrd;  //32-bit

    union fd{
        float f;
        dwrd d;c
    };

    char* ftoa(dwrd);
    dwrd myAtof(char* str);
    dwrd fadd(dwrd, dwrd);
    dwrd fsub(dwrd, dwrd);
    dwrd fmul(dwrd, dwrd);
    dwrd fdiv(dwrd, dwrd);
    int isZero(dwrd);
    ```

    定义并利用联合体union储存数据。

  - fadd

    首先检测两个数字是否为0的情况。这一部分较容易处理。

    然后进行加法运算。首先要进行对阶进行处理，将指数较小的通过移位把指数变大，之后根据符号位的异或结果，将尾数相加或相减，如果相减，还要在小数减大数时，将符号位变号。另外，还需要检查是否为非规格化，以及是否为无穷和NaN。

    最后进行尾数规格化。首先检测尾数是否超过24位，如果超过24位，需要进行右规，由于右规会损失精度，所以采取舍0进1的舍入策略，当然，有些简单粗暴，以后优化。如果不足24位，进行左规。

  - fsub

    减法把减数的符号位取反，调用fadd再相加即可。

  - fmul

    先处理存在0的情况，直接返回0.

    然后指数相加，尾数进行无符号数乘法。计算结果的二进制位数，将它的移位到32+24=56位，之后再左规右规规格化，还是简单粗暴的0舍1入策略。

  - fdiv

首先判断被除数与除数是否同时为0，如果同时为0则返回nan，如果被除数为0，返回0，如果除数为0，返回对应符号的无穷大。

之后首先指数作差，尾数将被除数拓展为64位并左移32位做定点小数除法，由于是C语言模拟，可以使用恢复除数除法。然后对商移位成24位，然行规格化就运算完成了。

- 示例
  - Sample

```
Please input two string of number as two float numbers:
You can only input " - . " and numbers.
Divided the two numbers using a space.
123.1234 -123.4567899
The float number of string you input is:
123.1234--->123.123398 and -123.4567899--->-123.456787
Now calculate the arithmetic result of the two numbers.
Sum        of 123.123398 and -123.456787 is -0.333389 or -0.333389 (C float calculation)
Difference of 123.123398 and -123.456787 is 246.580185 or 246.580185 (C float calculation)
Product    of 123.123398 and -123.456787 is -15200.421875 or -15200.418945 (C float calculation)
Quotient   of 123.123398 and -123.456787 is -0.997299 or -0.997300 (C float calculation)
Now output the string of operand1 and operand2 again:
123.123398--->123.123398 and -123.456787--->-123.456787
```

这里我们既显示了我们自己计算的结果，也显示了浮点计算的结果，进行对比，发现还是很准确的。

  - 大数

```
Please input two string of number as two float numbers:
You can only input " - . " and numbers.
Divided the two numbers using a space.
12345678901234567890123456789.123456789
12345678901234567890123456789.123456789
The float number of string you input is:
12345678901234567890123456789.123456789--->12345677973799631000000000000.000000 and 12345678901234567890123456789.123456789--->12345677973799631000000000000.000000
Now calculate the arithmetic result of the two numbers.
Sum        of 12345677973799631000000000000.000000 and 12345677973799631000000000000.000000 is 24691355947599261000000000000.000000 or 24691355947599261000000000000.000000 (C f
loat calculation)
Difference of 12345677973799631000000000000.000000 and 12345677973799631000000000000.000000 is 0.000000 or 0.000000 (C float calculation)
Product    of 12345677973799631000000000000.000000 and 12345677973799631000000000000.000000 is -0.000000 or 1.#INF00 (C float calculation)
Quotient   of 12345677973799631000000000000.000000 and 12345677973799631000000000000.000000 is 1.000000 or 1.000000 (C float calculation)
Now output the string of operand1 and operand2 again:
12345677973799631000000000000.000000--->12345677973799631000000000000.000000 and 12345677973799631000000000000.000000--->12345677973799631000000000000.000000
```

大数据的计算结果较为准确

  - 小数

```
lease input two string of number as two float numbers:
ou can only input " - . " and numbers.
ivided the two numbers using a space.
.0000000001 -0.0023000000
he float number of string you input is:
.0000000001--->0.000000 and -0.0023000000--->-0.002300
ow calculate the arithmetic result of the two numbers.
um         of 0.000000 and -0.002300 is -0.002300 or -0.002300 (C float calculation)
ifference  of 0.000000 and -0.002300 is 0.002300 or 0.002300 (C float calculation)
roduct     of 0.000000 and -0.002300 is -0.000000 or -0.000000 (C float calculation)
uotient    of 0.000000 and -0.002300 is -0.000000 or -0.000000 (C float calculation)
ow output the string of operand1 and operand2 again:
.000000--->0.000000 and -0.002300--->-0.002300
```

可以看出这里的计算也较为准确。但是数字输入过于小的时候，会直接当作0进行处理，即把尾数忽略。