CS205 C/C++ Programming -- Project 1 Simple Calculator

Name: 陈长信(Chen Changxin)

SID: 12210731

Part0--序言

本报告包含手写高精度算法的方法以及使用`gmp,mpfr`库完成，这个纯调库的解决方法将会放在方法二进行呈现。

----------方法一----------

Part1--问题分析

本项目旨在实现简易计算器功能，以达到加减乘除的基本功能。

如果使用浮点数或者整数类型输入计算，精度低且很容易出现溢出的情况。

故本项目使用高精度算法思想，字符串输入输出的方式对数据进行存储和处理。

思路大概分为两个方向，一个是纯整数型的运算，一个是带有小数型的运算。

对于纯整数型的运算：

1. 对于加法，即先将字符串中字符转化为`0-9`的数字倒序存进数组，然后进行竖式类似的进位运算，如果最后还有剩下`carry`即扩展一位输出。
2. 对于减法，思路与加法类似，不过考虑到可能出现负数的情况，故提前通过字符串`<string.h>`库中的`strcmp`方法对数据进行比较，然后分两类进行借位等运算。
3. 对于乘法，即用关键式`result[i + j] += op1[i] * op2[j]`，核心思想也是模拟乘法的竖式运算按位运算。
4. 对于除法，先使用减法的思想，在一个`while`循环之下不断用被除数减去除数，然后记录减的次数，最后剩余的被除数一定是小于除数，然后再转化为浮点数进行直接除法（此处精度会降低，因为直接使用的是常规浮点数除法），故只能处理高精度/低精度的除法问题。

对于带有小数的运算：

1. 对于加法，首先将整数和小数位分开处理，先处理小数位，将小数位较少的数后面添`0`，然后调用整数运算的方法对小数位进行求和，得到的`carry`位放到整数位，再在整数位进行一遍调用整数求和运算，最后拼起来输出结果。
2. 对于减法，思路是将两个数同时扩大一定的`10`的倍数，然后利用整数型减法的思路作减法最后对小数点的位置进行添加，注意移位的时候可能需要在前面添加`0`。
3. 对于乘法，即计算两个数总共移动了小数点多少位，最后只用调用整数类型的乘法运算，然后在相应的地方补上小数点即可。
4. 对于除法，同乘法一样，计算小数点需要移动的位数，然后调用整数型除法的函数，注意`0`的添加与减少，最后输出结果。

Part2--部分代码及功能介绍

1. 一个逆向由字符串转成数组的函数，在多个地方会进行调用转换

```
int *stringToArray_reverse(const char *str, int *size)
{
    *size = strlen(str);
    int *arr = (int *)malloc(sizeof(int) * (*size));
    int j = 0;

    for (int i = *size - 1; i >= 0; i--)
    {
        arr[j] = str[i] - '0';
        if (arr[j] > 9 || arr[j] < 0)
        {
            printf("The input cannot be interpret as numbers!");
            break;
        }
        j++;
    }

    return arr;
}
```

2. 整数的基础加法就是考虑进位的操作，将多余的进位往后加，然后倒序遍历即可。

```
void Integer_add(int op1[], int op2[], int len1, int len2)
{
    int carry = 0;
    int maxSize = len1 > len2 ? len1 : len2;
    // 这里多留了一位，用来处理有可能最后有进位
    int res[1001] = {0};
    int k = 0;

    int i = 0, j = 0;
    // 这里表示进位的操作，将余数存进答案数组，进位存进carry直到数组原数组遍历完
    while (i <= len1 - 1 || j <= len2 - 1)
    {
        int sum = carry;
        if (i <= len1 - 1)
            sum += op1[i++];
        if (j <= len2 - 1)
            sum += op2[j++];
        res[k++] = sum % 10;
        carry = sum / 10;
    }

    res[maxSize] = carry;
}
```

3. 基础整数减法也是同理，就是使用，就是考虑借位`borrow`，但是要通过`strcmp`提前判断一下大小关系，最后添加`-`号

```
void Integer_subtract(char op1[], char op2[], int len1, int len2)
```

```
{
    int is_neg = 0;
    int len = 0;
    int a[1001] = {0}, b[1001] = {0};
    // 判断哪个字符串比较大
    if (len1 < len2 || (strcmp(op1, op2) < 0 && len1 == len2))
    {
        // 这里将字符串判断之后再进行倒序输入
        is_neg = 1;
        for (int i = len2 - 1; i >= 0; i--)
            a[len2 - i - 1] = op2[i] - '0';
        for (int i = len1 - 1; i >= 0; i--)
            b[len1 - i - 1] = op1[i] - '0';
    }
    else
    {
        for (int i = len1 - 1; i >= 0; i--)
            a[len1 - i - 1] = op1[i] - '0';
        for (int i = len2 - 1; i >= 0; i--)
            b[len2 - i - 1] = op2[i] - '0';
    }
    // 将长度赋值为长度较长的数
    if (len1 > len2)
        len = len1;
    else
        len = len2;
    // 核心计算，如果需要借位的话a[i+1]--
    for (int i = 0; i < len; i++)
    {
        a[i] = a[i] - b[i];
        if (a[i] < 0)
        {
            a[i + 1] -= 1;
            a[i] += 10;
        }
    }
}
```

4. 对于乘法操作，同样的就是通过关键式`result[i + j] += op1[i] * op2[j]`进行求和，然后进位操作同加法。

```
for (i = 0; i < len1; i++)
{
    for (j = 0; j < len2; j++)
    {
        result[i + j] += op1[i] * op2[j];
        result[i + j + 1] += result[i + j] / 10; // 进位
        result[i + j] %= 10;                     // 当前位结果
    }
}
```

5. 对于除法操作，先进行减法，最后执行低精度的除法。（这里减法的操作同前面介绍的减法思路，故省略）

```
//这里表示在减法完成之后的除法以及decimal小数点的移位。
sprintf(ans0, "%Lf", double_ans);
```

```
int length = strlen(ans0);
if (decimal <= 0)
{
    printf("0.");
    for (int i = 0; i < abs(decimal); i++)
        printf("0");
    for (int i = 2; i < length; i++)
        printf("%c", ans0[i]);
}
 else
 {
     int start = 2;
     for (int i = 2; i < length; i++)
     {
         if (ans0[i] != '0')
         {
             start = i;
             break;
         }
      }
      for (int i = start; i < abs(decimal) + 2; i++)
          printf("%c", ans0[i]);
      printf(".");
      for (int i = abs(decimal) + 2; i < length; i++)
          printf("%c", ans0[i]);
}
```

6. 在进行科学计数法的计算时候，分开存储并且分开进行乘法以及加法的运算

```
char normalpart1[1001] = {0}, normalpart2[1001] = {0};
char scipart1[1001] = {0}, scipart2[1001] = {0};
// 这里就是把有科学计数法和没有的部分分开，次方位置使用加法，e前面的乘数部分使用小数的乘法计算
strncpy(scipart1, op1 + find_sci1 + 1, len1 - find_sci1 - 1);
strncpy(scipart2, op2 + find_sci2 + 1, len2 - find_sci2 - 1);
strncpy(normalpart1, op1 + 0, find_sci1);
strncpy(normalpart2, op2 + 0, find_sci2);
//乘数部分进行小数乘法
Decimal_multiply(normalpart1, normalpart2, find_sci1, find_sci2);
printf("e");
int newlen1 = len1 - find_sci1 - 1;
int newlen2 = len2 - find_sci2 - 1;
int *num1 = stringToArray_reverse(scipart1, &newlen1);
int *num2 = stringToArray_reverse(scipart2, &newlen2);
//科学计数法进行加法
Integer_add(num1, num2, newlen1, newlen2);
```

7. 在处理带有负数的计算的时候采用检测符号的形式，并转换计算类型，比如正数加负数，可以变成正数减正数的形式。

```
//这里仅展示的是小数加法在遇到带负号数的时候转化为小数减法操作。
else if (op1[0] != '-' && op2[0] == '-')
{
      for (int i = 1; i < len2; i++)
      {
          newop2[i - 1] = op2[i];
      }
      len2 = len2 - 1;
      Decimal_substract(op1, newop2, len1, len2);
}
```

8. 最后附上函数列表

```
  6 > int *stringToArray_reverse(const char *str, int *size)···
 25
 26 > void Integer_add(int op1[], int op2[], int len1, int len2)···
 59
 60 > void Integer_subtract(char op1[], char op2[], int len1, int len2)···
104
105 > void Integer_multiply(int op1[], int op2[], int len1, int len2, int decimal)···
149   // 此函数判断除法操作中被除数是不是继续做减法
150 > int judge_if_continuediv(int a[], int b[], int len)···
166   // 此函数计算除法操作最后的低精度除法
167 > float ints_to_float(int num[], int len)···
179
180 > void Integer_division(char op1[], char op2[], int len1, int len2, int decimal)···
295   // 此函数在小数计算的时候直接判断小数点的位置
296 > int find_decimalpoint(char op[], int len)···
309   // 此函数在于判断字符串中科学计数法的位置
310 > int find_sci(char op[], int len)···
323
324 > void Decimal_add(char op1[], char op2[], int len1, int len2)···
543
544 > void Decimal_substract(char op1[], char op2[], int len1, int len2)···
698
699 > void Decimal_multiply(char op1[], char op2[], int len1, int len2)···
768
769 > void Decimal_division(char op1[], char op2[], int len1, int len2)···
856   // 此函数用来进行整数的计算
857 > void integer_operation(char op1[], char operator, char op2[])···
1056  // 此函数用来进行带有小数的计算
1057 > void decimal_operation(char *op1, char operator, char * op2)···
1245  // 此函数用来判断是否输入的是可以解释的数字
1246 > int judge_if_interpretable(char op[])···
```

Part3--运行结果

Test Case #0

有关于代码的编译

```
daniel@Daniel-Chen:~/YU++/Project1-Calculator$ gcc -c Calculator.c
daniel@Daniel-Chen:~/YU++/Project1-Calculator$ gcc -o Calculator Calculator.o
daniel@Daniel-Chen:~/YU++/Project1-Calculator$ ./Calculator
```

Test Case #1

```
Input: 2 + 3
Output: 2 + 3 = 5
Input: 2 - 3
Output: 2 - 3 = -1
Input: 2 * 3
Ouput 2 * 3 = 6
Input: 2 / 3
Ouput 2 / 3 = 0.66666667
```

```
daniel@Daniel-Chen:~/YU++/Project1-Calculator$ ./Calculator 2 + 3
2 + 3 = 5
daniel@Daniel-Chen:~/YU++/Project1-Calculator$ ./Calculator 2 - 3
2 - 3 = -1
daniel@Daniel-Chen:~/YU++/Project1-Calculator$ ./Calculator 2 '*' 3
2 * 3 = 6
daniel@Daniel-Chen:~/YU++/Project1-Calculator$ ./Calculator 2 / 3
2 / 3 = 0.666667
daniel@Daniel-Chen:~/YU++/Project1-Calculator$
```

Test Case #2

```
Input: 3.14 / 0
Output: A number cannot be divided by zero!
```

```
daniel@Daniel-Chen:~/YU++/Project1-Calculator$ ./Calculator 3.14 / 0
A number cannot be divied by zero!
```

Test Case #3

```
Input: a * 2
Output: The input cannot be interpret as numbers!Please try again!
```

```
daniel@Daniel-Chen:~/YU++/Project1-Calculator$ ./Calculator a '*' 2
The input cannot be interpret as numbers!Please try again!
```

Test Case #4

```
Input: 987654321 * 987654321
Output: 975461057789971041

Input: 1.0e200 * 1.0e200
Output: 1.0e400
```

```
daniel@Daniel-Chen:~/YU++/Project1-Calculator$ ./Calculator 987654321 '*' 987654321
987654321 * 987654321 = 975461057789971041
daniel@Daniel-Chen:~/YU++/Project1-Calculator$ ./Calculator 1.0e200 '*' 1.0e200
1.0e200 * 1.0e200 = 1.00e400
```

Test Case #5

```
Input: 2 + 3
Output: 2 + 3 = 5
Input: 2 * 3
Output: 2 * 3 = 6
Input: quit
```

```
daniel@Daniel-Chen:~/YU++/Project1-Calculator$ ./Calculator
(Please Type quit to terminate)Enter operand 1:2
Enter operator: +
Enter operand 2: 3
2 + 3 = 5
(Please Type quit to terminate)Enter operand 1:2
Enter operator: *
Enter operand 2: 3
2 * 3 = 6
(Please Type quit to terminate)Enter operand 1:quit
daniel@Daniel-Chen:~/YU++/Project1-Calculator$
```

Part4--项目困难以及解决方法总结

1. 本项目的第一个难点在于计算大数计算的高精度算法，加减法就利用基础竖式计算的方式进行计算，带有小数点的计算就通过小数位和整数位分开计算的方法；乘法通过竖式计算的思路，除法就通过不断的减法最后通过低精度除法的思路完成，带有小数的计算小数点的移位即可。
2. 对于不同的计算器模式，就是通过主函数的`argc以及*argv`来判断命令行的读入。
3. 对于科学计数法的计算就是分两部分进行计算，最后进行拼凑合并输出。

Part5--补充说明以及项目总结

1. 本项目部分算法思路来自网络或者`ChatGPT`，本文档均为自行撰写。
2. 本项目对于除法的运算还存在低精度的情况，没有实现高精度/高精度的算法。
3. 另外，本项目在部分操作比如处理负数计算的时候代码部分显得冗长，还有改进提升空间。

----------方法二----------

Part 1 -- 问题分析

本项目实现简易计算器，以达到加减乘除的基本功能。

如果只是使用语言自带的运算，精度低且在进行大数计算的时候容易出现溢出。

故使用高精度整数计算库`gmp`以及高精度浮点数计算库`mpfr`进行辅助计算。

考虑到计算结果的精确性，我考虑在整数计算的时候调用`gmp`库，只在涉及到小数计算的时候采用`mpfr`库。

在除法运算的时候判断除数是否为`0`，以及由于除法运算的时候判断容易出现整数除整数出现浮点数的情况，故一律使用`mpfr`进行运算。

利用`mpfr`库中函数`mpfr_set_str`进行判断输入是否为数字，如果不是数字，给出相应的报错信息。

另外，考虑到结果的美观性，在浮点数运算结果超过`^10`或`^-10`的时候采用科学计数法输出，以及对所有的结果输出`10`位小数。

Part2--代码呈现

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <gmp.h>
#include <mpfr.h>

void integer_operation(char *op1_str, char operator, char * op2_str)
{
    // gmp库里的高精度整数类型
    mpz_t op1, op2, result;

    // 初始化这些值
    mpz_inits(op1, op2, result, NULL);

    // 这里把输入的字符串转为十进制的高精度整数类型
    mpz_set_str(op1, op1_str, 10);
    mpz_set_str(op2, op2_str, 10);

    // 关于不同的operator进行高精度的加减乘除运算
    switch (operator)
    {
    case '+':
        mpz_add(result, op1, op2);
        gmp_printf("%s + %s = %Zd\n", op1_str, op2_str, result);
        break;
    case '-':
        mpz_sub(result, op1, op2);
        gmp_printf("%s - %s = %Zd\n", op1_str, op2_str, result);
        break;
    case '*':
        mpz_mul(result, op1, op2);
        gmp_printf("%s * %s = %Zd\n", op1_str, op2_str, result);
        break;
    case '/':
        // 这里通过特殊函数判断高精度类型的被除数是否与0相等
        if (mpz_cmp_ui(op2, 0) == 0)
            printf("A number cannot be divied by zero\n");
        else
        {
            mpz_div(result, op1, op2);
            gmp_printf("%s / %s = %Zd\n", op1_str, op2_str, result);
        }
        break;
    default:
        printf("Error: Invalid operator\n");
    }

    // 最后清除内存
```

```
    mpz_clears(op1, op2, result, NULL);
}

void decimal_operation(char *op1_str, char operator, char * op2_str)
{

    mpfr_t op1, op2, result;

    // 这里用inits2函数将三个高精度浮点数值都初始为256位
    mpfr_inits2(256, op1, op2, result, NULL);

    // 设置op1以及op2为10进制数，并且设置为四舍五入模式(MPFR_RNDN)
    mpfr_set_str(op1, op1_str, 10, MPFR_RNDN);
    mpfr_set_str(op2, op2_str, 10, MPFR_RNDN);

    switch (operator)
    {
    case '+':
        mpfr_add(result, op1, op2, MPFR_RNDN);
        mpfr_printf("%s + %s = ", op1_str, op2_str);
        break;
    case '-':
        mpfr_sub(result, op1, op2, MPFR_RNDN);
        mpfr_printf("%s - %s = ", op1_str, op2_str);
        break;
    case '*':
        mpfr_mul(result, op1, op2, MPFR_RNDN);
        mpfr_printf("%s * %s = ", op1_str, op2_str);
        break;
    case '/':
        if (mpfr_cmp_ui(op2, 0) == 0)
        {
            printf("Error: Division by zero\n");
            return;
        }
        mpfr_div(result, op1, op2, MPFR_RNDN);
        mpfr_printf("%s / %s = ", op1_str, op2_str);
        break;
    default:
        printf("Error: Invalid operator\n");
        return;
    }

    // 获取结果的指数部分
    mpfr_exp_t exponent = mpfr_get_exp(result);

    // 如果指数部分大于等于 10 或小于等于 -10，则输出科学计数法表示，否则直接输出结果
    if (exponent >= 10 || exponent <= -10)
    {
        mpfr_printf("%.10Re\n", result);
    }
    else
    {
        mpfr_printf("%.10Rf\n", result);
    }

    // 清除内存
    mpfr_clears(op1, op2, result, NULL);
```

```
}

int main()
{
    mpfr_t num;
    char op1_str[100], op2_str[100];
    char operator;

    mpfr_init(num);
    while (1)
    {
        printf("(Please Type quit to terminate)Enter operand 1:");
        scanf("%s", op1_str);

        if (strcmp(op1_str, "quit") == 0)
            break;

        printf("Enter operator (+, -, *, /): ");
        scanf(" %c", &operator);

        printf("Enter operand 2: ");
        scanf("%s", op2_str);

        // 判断输入是不是数字，用mpfr库判断，如果不是，就输出不符合要求
        if (mpfr_set_str(num, op1_str, 10, MPFR_RNDN) != 0 || mpfr_set_str(num,
op2_str, 10, MPFR_RNDN) != 0)
        {
            printf("The input cannot be interpret as numbers! Please try
again!\n");
        }

        else
        {
            // 判断输入的数值类型（整数或小数）并选择相应的库函数进行运算
            if (strchr(op1_str, '.') == NULL && strchr(op2_str, '.') == NULL &&
operator!= '/')
            {
                // 整数运算
                integer_operation(op1_str, operator, op2_str);
            }
            else
            {
                // 小数运算
                decimal_operation(op1_str, operator, op2_str);
            }
        }
    }

    return 0;
}
```

Part3--运行结果

Test Case #1

```
Input: 2 + 3
Output: 2 + 3 = 5
Input: 2 - 3
Output: 2 - 3 = -1
Input: 2 * 3
Ouput 2 * 3 = 6
Input: 2 / 3
Ouput 2 / 3 = 0.66666667
```

```
daniel@Daniel-Chen:~/YU++/Project1-Calculator$ gcc -o Calculator Calculator.c -lgmp -lmpfr
daniel@Daniel-Chen:~/YU++/Project1-Calculator$ ./Calculator
(Please Type quit to terminate)Enter operand 1: 2
Enter operator (+, -, *, /): +
Enter operand 2: 3
2 + 3 = 5
(Please Type quit to terminate)Enter operand 1: 2
Enter operator (+, -, *, /): -
Enter operand 2: 3
2 - 3 = -1
(Please Type quit to terminate)Enter operand 1: 2
Enter operator (+, -, *, /): *
Enter operand 2: 3
2 * 3 = 6
(Please Type quit to terminate)Enter operand 1: 2
Enter operator (+, -, *, /): /
Enter operand 2: 3
2 / 3 = 0.6666666667
(Please Type quit to terminate)Enter operand 1: quit
daniel@Daniel-Chen:~/YU++/Project1-Calculator$
```

Test Case #2

```
Input: 3.14 / 0
Output: A number cannot be divided by zero!
```

```
daniel@Daniel-Chen:~/YU++/Project1-Calculator$ gcc -o Calculator Calculator.c -lgmp -lmpfr
daniel@Daniel-Chen:~/YU++/Project1-Calculator$ ./Calculator
(Please Type quit to terminate)Enter operand 1: 3.14
Enter operator (+, -, *, /): /
Enter operand 2: 0
A number cannot be divided by zero
(Please Type quit to terminate)Enter operand 1: quit
```

Test Case #3

```
Input: a * 2
Output: The input cannot be interpret as numbers!Please try again!
```

```
daniel@Daniel-Chen:~/YU++/Project1-Calculator$ gcc -o Calculator Calculator.c -lgmp -lmpfr
daniel@Daniel-Chen:~/YU++/Project1-Calculator$ ./Calculator
(Please Type quit to terminate)Enter operand 1: a
Enter operator (+, -, *, /): *
Enter operand 2: 2
The input cannot be interpret as numbers! Please try again!
(Please Type quit to terminate)Enter operand 1: 2
Enter operator (+, -, *, /): +
Enter operand 2: bbb
The input cannot be interpret as numbers! Please try again!
(Please Type quit to terminate)Enter operand 1: SUSTech
Enter operator (+, -, *, /): /
Enter operand 2: Tech
The input cannot be interpret as numbers! Please try again!
(Please Type quit to terminate)Enter operand 1: quit
daniel@Daniel-Chen:~/YU++/Project1-Calculator$
```

Test Case #4

```
Input: 2 $ 3
Output: Error: Invalid operator!
```

```
daniel@Daniel-Chen:~/YU++/Project1-Calculator$ gcc -o Calculator Calculator.c -lgmp -lmpfr
daniel@Daniel-Chen:~/YU++/Project1-Calculator$ ./Calculator
(Please Type quit to terminate)Enter operand 1: 2
Enter operator (+, -, *, /): $
Enter operand 2: 3
Error: Invalid operator
(Please Type quit to terminate)Enter operand 1: quit
```

Test Case #5

```
Input: 987654321 * 987654321
Output: 975461057789971041

Input: 1.0e200 * 1.0e200
Output: 1.0e400
```

```
daniel@Daniel-Chen:~/YU++/Project1-Calculator$ gcc -o Calculator Calculator.c -lgmp -lmpfr
daniel@Daniel-Chen:~/YU++/Project1-Calculator$ ./Calculator
(Please Type quit to terminate)Enter operand 1:987654321
Enter operator (+, -, *, /): *
Enter operand 2: 987654321
987654321 * 987654321 = 975461057789971041
(Please Type quit to terminate)Enter operand 1:1.0e200
Enter operator (+, -, *, /): *
Enter operand 2: 1.0e200
1.0e200 * 1.0e200 = 1.0000000000e+400
(Please Type quit to terminate)Enter operand 1:quit
daniel@Daniel-Chen:~/YU++/Project1-Calculator$
```

Test Case #6

```
Input: 2 + 3
Output: 2 + 3 = 5
Input: 2 * 3
Output: 2 * 3 = 6
Input: quit
```

```
daniel@Daniel-Chen:~/YU++/Project1-Calculator$ gcc -o Calculator Calculator.c -lgmp -lmpfr
daniel@Daniel-Chen:~/YU++/Project1-Calculator$ ./Calculator
(Please Type quit to terminate)Enter operand 1:2
Enter operator (+, -, *, /): +
Enter operand 2: 3
2 + 3 = 5
(Please Type quit to terminate)Enter operand 1:2
Enter operator (+, -, *, /): *
Enter operand 2: 3
2 * 3 = 6
(Please Type quit to terminate)Enter operand 1:quit
daniel@Daniel-Chen:~/YU++/Project1-Calculator$
```

Part4--项目困难以及解决方案

本项目难点之一在于如何处理大数的加减乘除，本质就是利用字符串的高精度计算方法解决。故使用`gmp`以及`mpfr`库进行高精度计算。