CS205 Project1

Name: 张嘉浩

SID: 12010423

Part1 - 问题分析

1. 题目的要求为`Implement a calculator which can add, subtract, multiply and divide two numbers.`，最艰巨的任务便是处理好大数四则运算的准确性。
2. 用户在输入过程中可能遇到的错误如下（具体处理办法见Part2）：
    1. 除法的时候以0为除数
    2. 输入操作符并非`+, -, *, /`
    3. 输入数字不符合格式要求
3. 用户想必不想在出错/进行完一次运算之后，整个程序就停止运行。因此，使用了一个`while(true)`循环，使用户可以不断执行输入操作，直到用户输入`q`后才退出循环。此操作也十分利于我进行程序的debug
4. 因为需要实现大数四则运算，因此我们使用`char[]`来进行数据的读入，并对每一个四则运算分门别类处理。用`char[]`来进行读入使得我们可以正确地处理任意大数字的运算

Part2 - 核心代码展示

数据存储

本project中利用`char num1_str[100], num2_str[100]`的char数组来存储读取的数字，而非简单的double类型

因为通过char数组可以存储到想要的任意大小的数字输入，后续再逐位取出正常的数字进行运算

判断一个str[] 是否是合法的数字：

1. `decimal_point`的flag, 遍历整个数字的过程中只能出现一个小数点
2. 负号`-`只能出现在第一位
3. 每一位都必须`isdigit()`
4. 整个`str[]`至少要有一个digit

```
bool is_number(char str[]) {
    size_t len = strlen(str);
    bool decimal_point = false;

    for (int i = 0; i < len; i++) {
```

```
        if (str[i] == '-') {
            if (i != 0) {
                // - can only in the first position
                return false;
            }
        } else if (str[i] == '.') {
            if (decimal_point) {
                // decimal point can only appear once
                return false;
            }
            decimal_point = true;
        } else if (!isdigit(str[i])) {
            // every char must be digit
            return false;
        }
    }

    // one digit at least
    if (!isdigit(str[len - 1]) && !isdigit(str[0])) {
        return false;
    }

    return true;
}
```

大数相加

使用类似竖式的思路实现高精度的大数加法，整个大数加法的大致思路如下

1. 用`size_t`来存储`strlen()`之后的值，避免`Narrowing conversion from 'unsigned long' to signed type 'int' is implementation-defined`的warning
2. 获取两个字符串中较长的一个`max_len`作为逐个位置加法的边界
3. 逐位相加的时候需要注意进位`carry`的运算
4. 关于`'0'`的使用: 可用于进行ASCII码和实际整数值的转换。例如，字符'5'的ASCII码是53，但我们希望在加法操作中使用整数值5。因此，通过减去ASCII码“0”(即48)，我们就得到了该数字的实际整数值。
5. 注意最后一步可能会有额外的进位，若有则需要将所有位置均向右挪一位，以获得正确的结果
6. 需要使用`'\0'`来标识结束，否则会产生许多意想不到的bug

```
void add(char num1[], char num2[], char answer[]) {
    size_t len1 = strlen(num1);
    size_t len2 = strlen(num2);

    size_t max_len;
    if (len1 > len2) {
```

```
        max_len = len1;
    } else {
        max_len = len2;
    }

    int carry = 0;
    for (size_t i = 0; i < max_len; i++) {
        int digit1;
        if (i < len1) {
            digit1 = num1[len1 - i - 1] - '0';
        } else {
            digit1 = 0;
        }
        int digit2;
        if (i < len2) {
            digit2 = num2[len2 - i - 1] - '0';
        } else {
            digit2 = 0;
        }

        int sum = digit1 + digit2 + carry;
        carry = sum / 10;

        answer[max_len - i - 1] = (char) (sum % 10 + '0');
    }

    if (carry != 0) {
        // shift right
        for (size_t i = max_len; i > 0; i--) {
            answer[i] = answer[i - 1];
        }
        answer[0] = (char) (carry + '0');
        // null terminator
        answer[max_len + 1] = '\0';
    } else {
        // null terminator
        answer[max_len] = '\0';
    }
}
```

大数相减

最初试图尝试使用大数相加的思路去完成大数相减，却遇到许多意料之外的困难，最后求助于GMP库来实现高精度的大数相减

大致思路是使用`mpz_init`进行初始化，使用`mpz_sub`进行减法运算，`gmp_sprintf`进行打印，最后`mpz_clear`进行清空收尾

参考网址：https://gmplib.org/manual/

```
void subtract(char num1_str[], char num2_str[], char answer[]) {
    // init
    mpz_t num1, num2, result;
    mpz_init_set_str(num1, num1_str, 10);
    mpz_init_set_str(num2, num2_str, 10);
    mpz_init(result);

    // sub & print
    mpz_sub(result, num1, num2);
    gmp_sprintf(answer, "%Zd", result);

    // clear
    mpz_clear(num1);
    mpz_clear(num2);
    mpz_clear(result);
}
```

大数相乘

参考思路来源：https://blog.csdn.net/weixin_41376979/article/details/79197186

与加法类似，均是竖式运算的操作思路以实现高精度的大数乘法

1. 使用strlen函数计算输入数字的长度，并计算乘积的最大长度。
2. 创建一个长度为max_len的数组数字来存储结果
3. 从右到左遍历num1的每个数字，并将其与从右到左的num2的每个数字相乘，将结果添加到数字数组中的相应数字。同时跟踪每次乘法中产生的进位，并将其添加到下一个乘积中
4. 乘法完成后，从数组中消灭前导零，并按倒序将数字复制到答案数组

```
void multiply(char num1[], char num2[], char answer[]) {
    size_t len1 = strlen(num1);
    size_t len2 = strlen(num2);
    size_t max_len = len1 + len2;

    int digits[max_len];
    for (size_t i = 0; i < max_len; i++) {
        digits[i] = 0;
    }

    for (size_t i = 0; i < len1; i++) {
        int carry = 0;
        int digit1 = num1[len1 - i - 1] - '0';

        for (size_t j = 0; j < len2; j++) {
```

```
            int digit2 = num2[len2 - j - 1] - '0';
            int product = digit1 * digit2 + carry + digits[i + j];

            carry = product / 10;
            digits[i + j] = product % 10;
        }

        if (carry > 0) {
            digits[i + len2] += carry;
        }
    }

    // remove leading zeroes
    size_t i = max_len - 1;
    while (i > 0 && digits[i] == 0) {
        i--;
    }

    // copy digits to answer in reverse order
    for (size_t j = 0; j <= i; j++) {
        answer[j] = (char) (digits[i - j] + '0');
    }

    // null terminator
    answer[i + 1] = '\0';
}
```

大数相除

与减法一样，通过对GMP库的利用，实现高精度的大数除法

```
mpz_t num1, num2, quotient, remainder;
mpz_inits(num1, num2, quotient, remainder, NULL);

mpz_set_str(num1, num1_str, 10);
mpz_set_str(num2, num2_str, 10);

if (mpz_cmp_si(num2, 0) == 0) {
    printf("A number cannot be divided by zero.\n");
    continue;
}
mpz_tdiv_qr(quotient, remainder, num1, num2);

char quotient_str[101], remainder_str[101];
mpz_get_str(quotient_str, 10, quotient);
mpz_get_str(remainder_str, 10, remainder);
printf("%s / %s = %s with remainder %s\n", num1_str, num2_str, quotient_str,
remainder_str);
```

```
mpz_clears(num1, num2, quotient, remainder, NULL);
```

Part3 - 结果&验证

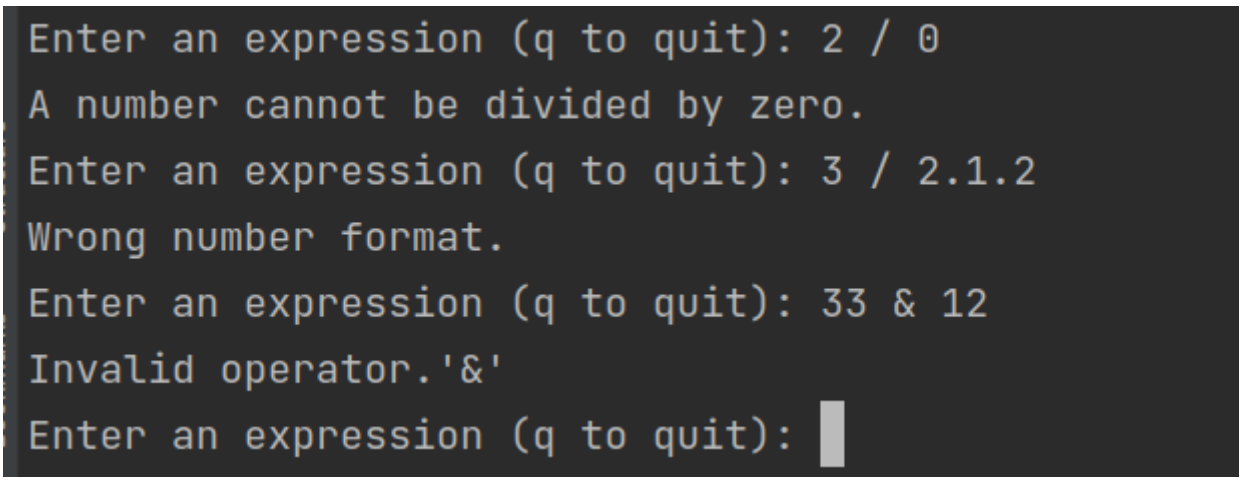
以下结果经过与`卡西欧科学计算器FX-991`的对照检验，程序均给出了正确的答案

编译过程

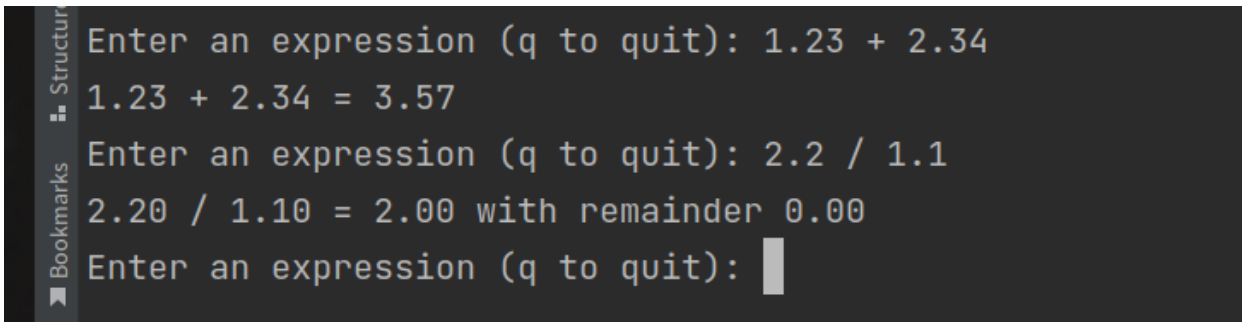
需要用到`-lgmp`以及`-lm`指令，与math库和GMP库相链接

```
excelsior@ubuntu:~/Desktop/C_repo/CS205_Project1$ gcc calculator.c -o
calculator -lgmp -lm
excelsior@ubuntu:~/Desktop/C_repo/CS205_Project1$ ./calculator
```

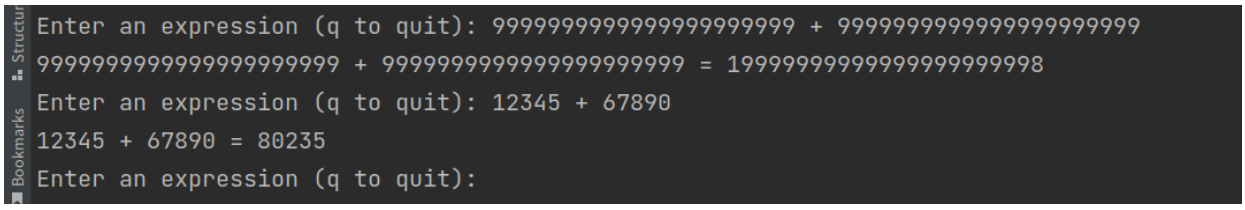
合法性检验



浮点数运算



加法



减法

```
Enter an expression (q to quit): 98765 - 43210
98765 - 43210 = 55555
Enter an expression (q to quit): 10 - 20
10 - 20 = -10
Enter an expression (q to quit): 12.5 - 13.5
12.50 - 13.50 = -1.00
```

乘法

```
Enter an expression (q to quit): 987654321 * 987654321
987654321 * 987654321 = 975461057789971041
Enter an expression (q to quit): 12345678987654321 * 12345678987654321
12345678987654321 * 12345678987654321 = 152415789666209420210333789971041
```

除法

```
Enter an expression (q to quit): 1001 / 7
1001 / 7 = 143 with remainder 0
Enter an expression (q to quit): 41823479801 / 34219876
41823479801 / 34219876 = 1222 with remainder 6791329
Enter an expression (q to quit): 2.22 / 1.11
2.22 / 1.11 = 2.00 with remainder 0.00
```

终态

```
Enter an expression (q to quit): 2.22 / 1.11
2.22 / 1.11 = 2.00 with remainder 0.00
Enter an expression (q to quit): q
Quiting...
```

Part 4 - 遇到的问题&解决方案

1. 开始写这个project时，对C中十分重要的概念“指针”一窍不通，在网上寻求思路的时候却又经常遇到含有指针的代码，导致理解缓慢而花费大量的时间。最后通过对b站于教授lecture5的指针网课的初步自学，以及对会C++同学的咨询才得以完成此次project。不过由于对指针的掌握尚未熟练，本次project的实现中尚未使用指针，不过对于指针的初步了解极大地降低了我搜寻参考思路的困难。

2. 通过与科学计算器的对比，发现许多bug是由于对整数类型的范围不熟练造成的，整数类型看似简单却十分容易出错。如因为`strlen`引起的：`Narrowing conversion from 'unsigned long' to signed type 'int' is implementation-defined`等warnings，最后可将变量声明为`size_t`类型得以解决。
3. 编程过程伊始对ASCII码仅有抽象的概念，在写完加减法运算之后就已变得熟练了许多（在初版的代码中，有许多位置忘记进行`+'0'/-'0'`的操作，后续在不断的debug中留下了深刻的印象）
4. 每个string的最后需要跟一个`\0`, 否则编译器不知道string何时结束
5. 对加法和减法的实现思路几乎一致，因此存在大量的duplicated code（后期发现实现的减法在进行负数运算时存在问题，便没有使用字符串逐位运算减法的方法。在使用GMP库之后，不会出现duplicate code了，后续分析仅针对最初版减法实现）。在设计模式中重构的学习时，duplicated code主要有以下四种解决办法 1. extract method 2. extract class 3. template method pattern 4. strategy pattern。经过初步的判断，我认为这次的实现其实可以通过extract method和构造template method pattern来解决，但由于个人对C++中函数仍使用十分不熟练，在本project尝试重构代码的过程中引起了大量其他答案正确性问题，遂最后作罢。