CS205 Project 1

思路分析

本次作业要求实现一个简单的计算器，实现两位数的加减乘除。其难点在于如何实现高精度，直接引用C中现有的数据类型显然是不能满足要求的，因此我想到可以把输入的字符当作char数组，逐位进行加减乘除运算，在此基础上加上进位、补位等操作即可实现理论上的无限精度（只要有无限的内存即可）。

具体代码及实现方法

输入参数的格式

输入参数应为 数字 + 操作符 + 数字，其中数字可以为小数或整数，操作符可以为：

加（+）、减（-）、乘（x<小写的x>）、除（/）

```
// 传入参数
    char *num11 = argv[1];
    char op = *argv[2];
    char *num22 = argv[3];
```

输出参数的格式形如 a + b = c（如下图）

```
D:\CS205P1\cmake-build-debug\CS205P1.exe 23344 / 223
23344 / 223 = 104.681614
```

前置工作

在正式进行代码的运算之前，需要解决几个问题：

1. 输入的两个数字位数应当一一对齐，若传入时参数长度不同，则应补齐。其中整数部分应当在前补0，小数部分应当在后补0。

```
// 对齐整数函数
char *orderInt(int width, char c[]) {
    int len = strlen(c);
    char *buffer = (char *) malloc((width + 1) * sizeof(char));
    int i, j;

    // 将buffer数组初始化为'0'
    for (i = 0; i < width; i++) {
        buffer[i] = '0';
    }
    buffer[width] = '\0';// 添加字符串结尾符
```

```
    // 将字符串c插入到buffer数组的右端
    i = width - 1;
    j = len - 1;
    while (j >= 0 && i >= 0) {
        buffer[i] = c[j];
        j--;
        i--;
    }
    return buffer;
}

// 对齐小数函数
char *orderDeci(int width, char c[]) {
    int len = strlen(c);
    char *buffer = (char *) malloc((width + 1) * sizeof(char));
    int i, j;

    // 将buffer数组初始化为'0'
    for (i = 0; i < width; i++) {
        buffer[i] = '0';
    }
    buffer[width] = '\0';// 添加字符串结尾符

    // 将字符串c插入到buffer数组的左端
    i = 0;
    j = 0;
    while (i < width && j < len) {
        buffer[i] = c[j];
        i++;
        j++;
    }
    return buffer;
}
```

2. 输入是以char数组形式存储，则输出也应是char数组形式。若想将其打印，应当遍历数组，为简化代码，故应写出打印数组的函数。

```
// 打印数组
void printArray(char *s) {
    int i = 0;
    while (s[i] != '\0') {
        printf("%c", s[i]);
        i++;
    }
}
```

3. 我的方法是整数与小数通过token方法分割成两个部分，先计算小数部分，再根据进位或补位情况进行整数部分的计算。因此可能涉及在数组前插入小数点或插入符号的情况，因此应先写出在数组最前面插入一个字符的方法。

```
// 在数组前面插入一个字符
char *insertAns(char num, char c[]) {
    char *buffer = (char *) malloc((strlen(c) + 2) * sizeof(char));

    // 将buffer第0位设为num，其余与c一一对应
    buffer[0] = num;
    for (int i = 1; i < strlen(buffer); i++) {
        buffer[i] = c[i - 1];
    }
    buffer[strlen(c) + 1] = '\0';  // 添加字符串结尾符
    return buffer;
}
```

4. 为避免计算中出现负号，因此减法的计算应当先得出绝对值，再根据实际情况决定是否要添加负号。要得出绝对值则需要先比较两个数的大小关系，因C语言没有max方法，故重写max方法，再根据实际情况决定参数类型。

```
// 比较最大值函数(char*类型)
char *maxChar(char *a, char *b) {
    // 先比较位数
    if (strlen(a) > strlen(b)) {
        return a;
    } else if (strlen(a) < strlen(b)) {
        return b;
    } else { // 如果二者位数相等
        for (int i = 0; i < strlen(a); i++) {
            if (a[i] > b[i]) {
                return a;
            } else if (a[i] < b[i]) {
                return b;
            }
        }
        return a;
    }
}
//int类型
int maxInt(int a, int b) {
    if (a >= b) {
        return a;
    } else {
        return b;
    }
}
```

5. 需要预声明的变量

```
    char *token1, *token2; // 分割字符串
    char *int1, *int2; // 数字的整数部分
    char *deci1, *deci2; // 数字的小数部分
    char *resultInt; // 整数部分计算结果
    char *resultDeci; // 小数部分计算结果
    char *result = "0"; // 最终输出结果
```

输入参数格式有误

考虑了作业文档中的四种情况，并且有相应的方法实现，代码如下：

```
if (argc != 4) {
        printf("Arguments number illegal! Please check the number of your
argument!\n"); // 输入参数不正确
        return -1;
    }
    if (*argv[2] != '+' && *argv[2] != '-' && *argv[2] != 'x' && *argv[2] != '/') {
        printf("illegal operator!\n"); // 操作符有误
        return -1;
    }
    int i = 0;
    while (argv[1][i] != '\0') {
        if (argv[1][i] > '9' || argv[1][i] < 0) {
            printf("The input cannot be interpret as numbers!\n"); // 输入不是数字
            return -1;
        }
        i++;
    }
    i = 0;
    while (argv[3][i] != '\0') {
        if (argv[1][i] > '9' || argv[1][i] < 0) {
            printf("The input cannot be interpret as numbers!\n"); // 输入不是数字
            return -1;
        }
        i++;
    }
    if (*argv[2] == '/' && argv[3][0] == '0') {
        printf("Cannot divided by zero.\n"); // 分母不能为0
        return -1;
    }
```

示例如下：

```
wyuuu@LAPTOP-AF3F78HF:/mnt/d/资料/C++/project/project1$ ./test 2 + 3 + 3
Arguments number illegal! Please check the number of your argument!
```

```
wyuuu@LAPTOP-AF3F78HF:/mnt/d/资料/C++/project/project1$ ./test a x 2
The input cannot be interpret as numbers!
```

```
wyuuu@LAPTOP-AF3F78HF:/mnt/d/资料/C++/project/project1$ ./test 5 / 0
Cannot divided by zero.
```

```
wyuuu@LAPTOP-AF3F78HF:/mnt/d/资料/C++/project/project1$ ./test 3 s 4
illegal operator!
```

加法方法

加法的实现

具体方法为先创造一个char*类型，并为其分配足够的内存，以便存储得到的结果数组。

加法的实现则为遍历两个char数组，在各自对应的位置进行加法计算，并且加上先前的进位，将得出的结果填充到结果数组中。若循环结束后仍有进位，则将进位与结果数组拼接形成新的结果数组。

具体代码如下：

```
// 加法函数
char *addition(char *s1, char *s2) {
    int len1 = strlen(s1);
    int len2 = strlen(s2);
    int len = maxInt(len1, len2);
    char *result = (char *) malloc((len + 2) * sizeof(char)); // 分配足够的内存

    int carry = 0;
    int i, j, k;

    for (i = len1 - 1, j = len2 - 1, k = len - 1; i >= 0 || j >= 0; i--, j--, k--) {
        int num1 = (i >= 0) ? s1[i] - '0' : 0;
        int num2 = (j >= 0) ? s2[j] - '0' : 0;
        int sum = num1 + num2 + carry;
        result[i] = sum % 10 + '0';
        carry = sum / 10;
    }
    if (carry > 0) { // 如果最高位有进位，则需要在结果的最前面再加上进位的数字
        char *carryStr = (char *) malloc((len + 2) * sizeof(char));
        carryStr[0] = carry + '0';
        carryStr[1] = '\0';
        strcat(carryStr, result); // 将carryStr字符串连接到result字符串的前面
        result = carryStr; // 将carryStr的地址赋值给result
        len++; // 字符串长度加1
    }
    result[len] = '\0'; // 添加字符串结尾符
    return result;
}
```

实例如下：

```
wyuuu@LAPTOP-AF3F78HF:/mnt/d/资料/C++/project/project1$ ./test 12234.23 + 233.34
12234.23 + 233.34 = 12467.57
```

整数与小数部分的拼接

整数与小数部分各自计算完毕后，需要进行二者的拼接，尤其需要注意的是小数点的拼接，若二者均为整数，则无需小数点。

具体代码如下：

```
    /*
    * 加法按字符串位置逐个计算，先计算小数，
    * 得出整数部分后，再进行整数部分的运算
    */
    if (op == '+') {

        if (deci1 != NULL && deci2 != NULL) {
            char carry[1] = {'0'}; // 小数进位部分
            resultDeci = addition(deci1, deci2); // 小数部分求和计算
            // 若有进位，则将进位部分取出,并且加上小数点
            if (strlen(resultDeci) > strlen(deci1)) {
                int1 = addition(int1, "1");
                resultDeci++;
            }

            // 计算整数部分
            resultInt = addition(int1, int2);

            // 拼接整数和小数的字符串，输出最终结果
            result = strcat(resultInt, ".");
            result = strcat(result, resultDeci);
        } else if (deci1 == NULL && deci2 != NULL) {
            deci1 = "0";
            resultInt = addition(int1, int2);
            resultDeci = insertAns('.', deci2);
            result = strcat(resultInt, resultDeci);
        } else if (deci2 == NULL && deci1 != NULL) {
            deci2 = "0";
            resultInt = addition(int1, int2);
            resultDeci = insertAns('.', deci1);
            result = strcat(resultInt, resultDeci);
        } else {
            resultInt = addition(int1, int2);
            result = resultInt;
        }
        // 输出结果
        printArray(num11);
        printf(" %c ", op);
        printArray(num22);
        printf(" = ");
        printArray(result);
        printf("\n");

    }
```

减法方法

```
char *subtract(char *s1, char *s2) {
    int len1 = strlen(str1);
    int len2 = strlen(str2);
    int len = maxInt(len1, len2);
    char *result = (char *) malloc((len) * sizeof(char)); // allocate memory
    int carry = 0;
    int i, j, k;
    char *s1 = maxChar(str1, str2);
    char *s2 = maxChar(str1, str2) == str1 ? str2 : str1;

    for (i = len1 - 1, j = len2 - 1, k = len - 1; i >= 0 || j >= 0; i--, j--, k--) {
        int num1 = (i >= 0) ? s1[i] - '0' : 0;
        int num2 = (j >= 0) ? s2[j] - '0' : 0;
        int sum = num1 - num2 - carry;
        if (sum < 0) {
            sum = -sum;
            carry = 1;
        }else{
            carry = 0;
        }
        result[i] = sum + '0';
    }
    if (carry > 0) { // 如果最高位有进位，则需要在结果的最前面再加上进位的数字
        char *carry_str = (char *) malloc((len + 2) * sizeof(char));
        carry_str[0] = carry + '0';
        carry_str[1] = '\0';
        strcat(carry_str, result); // 将carry_str字符串连接到result字符串的前面
        result = carry_str; // 将carry_str的地址赋值给result
        len++; // 字符串长度加1
    }
    result[len] = '\0'; // 添加字符串结尾符
    while (*result == '0' && *(result + 1) != '\0') {
        result++;
    }
    return result;
}
```

实例如下：

```
/Users/natyiano/CLionProjects/untitled1/cmake-build-debug/untitled1 13.32 - 14.89
13.32 - 14.89 = 1.57 NEGATIVE
```

乘法方法

因时间问题及乘除法相对复杂，最终没有实现乘法与除法的高精度计算，但是仍然有判断输入是小数还是整数的判断，并将小数转为double类型，整数转为long long int类型，以保证计算仍有较高的精度。并且用printf的方法去除多余的0。

具体代码如下：

```
// 乘法方法
    else if (op == 'x') {
        bool bl = false;// 判断小数还是整数
        for (int i = 0; i < strlen(num11); i++) {
            if (num11[i] == '.' || num22[i] == '.') {
                bl = true;
            }
        }

        if (bl == true) {
            double a = atof(num11);
            double b = atof(num22);
            double result1 = a * b;
            printArray(num11);
            printf(" %c ", op);
            printArray(num22);
            printf(" = ");
            printf("%.10g", result1);
            printf("\n");
        } else{
            long long int a = atoll(num11);
            long long int b = atoll(num22);
            long long int result1 = a * b;
            printArray(num11);
            printf(" %c ", op);
            printArray(num22);
            printf(" = ");
            printf("%lld", result1);
            printf("\n");
        }

    }
```

实例如下：

```
wyuuu@LAPTOP-AF3F78HF:/mnt/d/资料/C++/project/project1$ ./test 234.4 x 212
234.4 x 212 = 49692.8
```

除法方法

除法实现方法与乘法类似，具体代码如下：

```
// 除法(同乘法)
    else if (op == '/') {
        double a = atof(num11);
        double b = atof(num22);
        double result1 = a / b;
        printArray(num11);
        printf(" %c ", op);
        printArray(num22);
        printf(" = ");
        printf("%.10g", result1);
        printf("\n");
    }
```

实例如下：

```
wyuuu@LAPTOP-AF3F78HF:/mnt/d/资料/C++/project/project1$ ./test 234 / 23
234 / 23 = 10.17391304
```