# CS205 C/ C++ Program Design

# Assignment 2

Name: 王康

SID: 11510815

## Part 1. Source Code

代码的主要思路是先利用栈将用户输入的中缀表达式转换为后缀表达式（可以去除括号并且方便计算机计算），接着借助一个栈来计算后缀表达式的值。

第一版代码：

```
#include<iostream>
#include<string>
#include<stack>
#include<queue>
using namespace std;
bool num(char c){
    if(c >= '0'&& c <= '9'){
        return true;
    }
    return false;
}

int calculate(char a, char b, char op){
    //用于计算a op b，其中a和b均为char
    int x = a-48;
    int y = b-48;
    if(op=='+'){
        return x+y;
    }
    else if(op=='-'){
        return x-y;
    }else if(op=='*'){
        return x*y;
    }else if(op=='/'){
        return x/y;
    }

}

int main( ){
```

```
    //辅助将中缀表达式转为后缀表达式的栈
    stack<char>symbolStack;
    //队列q用于存储后缀表达式
    queue<char>q;
    cout << "欢迎使用计算器,输入Exit退出" << endl;

    while (1)
    {
        string exp;
        cin >> exp;
        if(exp == "Exit"){
            break;
        }
        int i = 0;
        int size = exp.size();
        //遍历用户输入，将前缀表达式转化为后缀表达式
        while(i<size){
            char c = exp[i];
            i++;
            if(num(c)){
                //如果是数字直接放入队列
                q.push(c);
            }
            else if(c == '+'||c == '-'){
                //对于+-两种优先级最低的符号，符号栈一直需要pop元素到队列直到栈为空或者遇到左括号
                while(!symbolStack.empty()&&symbolStack.top()!='('){
                    q.push(symbolStack.top());
                    symbolStack.pop();
                }
                //符号入栈
                symbolStack.push(c);
            }

            else if(c == '*'||c == '/'){
                //对于*/两种高优先级符号，只有栈顶为*或者/的时候需要pop元素到队列
                while(!symbolStack.empty()&&symbolStack.top()!='('&&
                symbolStack.top()!='+'&&symbolStack.top()!='-'){
                    q.push(symbolStack.top());
                    symbolStack.pop();
                }
                //符号入栈
                symbolStack.push(c);
            }
            else if(c == '('){
                //左括号直接入栈
                symbolStack.push(c);

            }
```

```
            else if(c ==')'){
                //右括号会一直pop栈中的元素到队列直到左括号在栈顶
                while(symbolStack.top()!='('){
                    char p =symbolStack.top();
                        symbolStack.pop();
                        q.push(p);
                }
                //pop出左括号（不加入队列）
                symbolStack.pop();
            }

        }
        //如果符号栈中还有元素，依次pop出到队列中
        while(!symbolStack.empty()){
            q.push(symbolStack.top());
            symbolStack.pop();
        }
        //执行完后已经得到后缀表达式在队列q中

        //需要一个新的辅助栈来求值
        stack<char> cal;
        while(!q.empty()){
            //遍历后缀表达式队列中的所有元素
            if(num(q.front())){
                //遇见数字时入栈
                cal.push(q.front());
            }
            else{
                //遇见操作符，出栈2个数字b,a做a op b运算
                char b = cal.top();
                cal.pop();
                char a = cal.top();
                cal.pop();
                int result = calculate(a,b,q.front());
                cal.push(result+48);
            }
            q.pop();
        }
        //最后留在栈中的元素即为运算结果
        int result = cal.top()-48;
        cout<<result<<endl;

    }
    return 0;

}
```

改进后的代码

```cpp
#include<iostream>
#include<string>
#include<stack>
#include<queue>
using namespace std;
bool num(char c){
    if(c >= '0'&& c <= '9'){
        return true;
    }
    return false;
}

bool isNumber(string s){
    if(s=="+"||s=="-"||s=="*"||s=="/"){
        return false;
    }
    return true;
}

string calculate(string a, string b, string op){
    //用于计算a op b
    //将字符串转为double
    double x = atof(a.c_str());
    double y = atof(b.c_str());
    if(op=="+"){
        return to_string(x+y);
    }
    else if(op=="-"){
        return to_string(x-y);
    }else if(op=="*"){
        return to_string(x*y);
    }else if(op=="/"){
        return to_string(x/y);
    }

}

int main(){
    //辅助将中缀表达式转为后缀表达式的栈
    stack<char>symbolStack;
    //队列q用于存储后缀表达式
    queue<string>q;
    cout << "欢迎使用计算器,输入Exit退出" << endl;

    while (1)
    {
        string exp;
        cin >> exp;
        if(exp == "Exit"){
```

```
            break;
        }
        int i = 0;
        int size = exp.size();
        //遍历用户输入，将前缀表达式转化为后缀表达式
        while(i<size){
            char c = exp[i];
            i++;
            string b(1,c);
            if(num(c)){
                if(!num(exp[i])&&exp[i]!='.'){
                    //如果是一位数的数字，直接放入队列
                    q.push(b);
                }
                else{
                    string d(1,exp[i]);
                    string temp =b+d;
                    //如果遇到连续的数字
                    i++;
                    if(i<size){
                        while(num(exp[i])||exp[i]=='.'){
                            string f(1,exp[i]);
                            temp+=f;
                            i++;
                            if(i==size){
                                break;
                            }
                        }
                    }
                    q.push(temp);
                }


            }
            else if(c == '+'||c == '-'){
                //对于+-两种优先级最低的符号，符号栈一直需要pop元素到队列直到栈为空或者遇到左括号
                while(!symbolStack.empty()&&symbolStack.top()!='('){
                    string a(1,symbolStack.top());
                    q.push(a);
                    symbolStack.pop();
                }
                //符号入栈
                symbolStack.push(c);
            }

            else if(c == '*'||c == '/'){
                //对于*/两种高优先级符号，只有栈顶为*或者/的时候需要pop元素到队列
                while(!symbolStack.empty()&&symbolStack.top()!='('&&
```

```
            symbolStack.top()!='+'&&symbolStack.top()!='-'){
                string a(1,symbolStack.top());
                q.push(a);
                symbolStack.pop();
            }
            //符号入栈
            symbolStack.push(c);
        }
        else if(c == '('){
            //左括号直接入栈
            symbolStack.push(c);

        }
        else if(c ==')'){
            //遇到右括号会一直pop栈中的元素到队列直到左括号在栈顶
            while(symbolStack.top()!='('){
                string p (1,symbolStack.top());
                symbolStack.pop();
                q.push(p);
            }
            //pop出左括号（不加入队列）
            symbolStack.pop();
        }

    }
    //如果符号栈中还有元素，依次pop出到队列中
    while(!symbolStack.empty()){
        string p (1,symbolStack.top());
        q.push(p);
        symbolStack.pop();
    }
    //执行完后已经得到后缀表达式在队列q中

    //需要一个新的辅助栈来求值
    stack<string> cal;
    while(!q.empty()){
        //遍历后缀表达式队列中的所有元素
        //Excout<<q.front()<<endl;
        if(isNumber(q.front())){
            //遇见数字时入栈
            cal.push(q.front());
        }
        else{
            //遇见操作符，出栈2个数字b,a做a op b运算
            string b = cal.top();
            cal.pop();
            string a = cal.top();
            cal.pop();
            string result = calculate(a,b,q.front());
```

```
                cal.push(result);
            }
            q.pop();
        }
        //最后留在栈中的元素即为运算结果
        string result = cal.top();
        cout<<result<<endl;

    }
    return 0;

}
```

# Part 2. Result & Verification

（第一版程序的测试结果如下）


## Test case #1:

Input: 2*(3+5)

Output: 16

## Test case #2:

Input: 8-4/2

Output: 6

### Test case #3:

Input: (2-7)*2

Output: -10

（改进后的程序测试如下）


### Test case #1:

Input: 1.378+23

Output: 24.378000

### Test case #2:

Input: (3.2+7.6)*2.3

Output: 24.840000

### Test case #3:

Input: 9.1/2+6+(6/2)

Output: 13.550000

## Part 3. Difficulties & Solutions, or others

第一版程序的缺点在于不支持超过一位数的数字运算比如23+3或者1.3+2，系统只可以处理诸如7+8=15之类的运算。因为当时读取数字是按char读取单个数字。

第二版程序想到了改进办法，读取字符串并识别下一个char是不是也在‘0’和‘9’之间。这样带来的问题是之前储存‘+’，‘1’可以直接采用char类型的栈和队列，现在则必须改成string类型以储存‘+’和“10”两种本不同类型的数据，读取时加以判断类型即可恢复。

第三版程序在第二版的基础上加以改进，支持用户输入小数（在读取字符串时将‘.’与数字类似处理），将存储类型从int改为了double类型。