山东大学 计算机科学与技术 学院

数据结构与算法 课程实验报告

学号：201700130033	姓名： 武学伟	班级： 2017 级 2 班
实验题目：栈		
实验学时：4	实验日期： 2018.11.4	

实验目的：

1.掌握栈结构的定义与实现。

2.掌握栈结构的使用。

软件环境：

Win10home，codeblocks

1. 实验内容（题目内容，输入要求，输出要求）

1.创建栈类，采用数组描述；

2.计算数学表达式的值，输入数学表达式，输出表达式的计算结果。

数学表达式由多位或单位数字和运算符组成，并且假定所有表达式合法。

2. 数据结构与算法描述 （整体思路描述，所需要的数据结构与算法）

数据结构：

栈

算法：

转化为后缀表达式：

①从左往右对中缀表达式进行遍历。

②遇到数字直接放入后缀表达式。

③创建一个用于存储运算符的栈。

④若不是数字，则需要判断操作符种类。若栈为空，不分操作符种类直接插入后缀表达式。否则若是左括号，直接入栈；若是右括号，出栈至遇到左括号，将栈中元素放到后缀表达式当中；若是其他操作符，则判断优先级（设定优先级为'('＜'+'='-'＜'*'＜'/'＜')'，若是栈顶元素的优先级高于中缀表达式中的优先级，则出栈至优先级低于中缀表达式的优先级，然后再入栈中缀表达式的当前操作符，若优先级正确，则直接入栈。

⑤遍历完成后，检查栈是否为空，非空则将剩余的元素放入后缀表达式中。

后缀表达式求值：

①建立一个用于存储数字的栈。

②从后缀表达式中提取字符，若是数字则直接存放进栈中。

③若是字符，则取前栈顶的两个数字进行相应的运算，并将结果存储在栈中，注意'-'和'/'的时候，是栈顶的钱一个元素减去栈顶元素，在取数的时候注意以下

④最后的栈顶元素就是表达式的计算结果。

字符转化为数字：

①在存储为后缀表达式的时候为了区分不同的元素时，在每一个元素后面加''。

②在后缀表达式的计算的时候，判断元素是否为数字，若是则用元素值减去 0 的 ASCII 码值就是对应的整形数字。

③多位数的操作，定义一个变量 sum，每次遍历到一个数字，则对前面的数×10，然后再加上当前的数值。

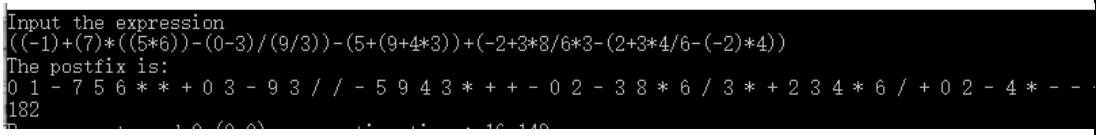
3. 测试结果（测试输入，测试输出，结果分析）

测试一：

输入：

((-1)+(7)*((5*6))-(0-3)/(9/3))-(5+(9+4*3))+(-2+3*8/6*3-(2+3*4/6-(-2)*4))

输出：



The postfix is:

0 1 - 7 5 6 * * + 0 3 - 9 3 / / - 5 9 4 3 * + + - 0 2 - 3 8 * 6 / 3 * + 2 3 4 * 6 / + 0 2 - 4 * - - +

182

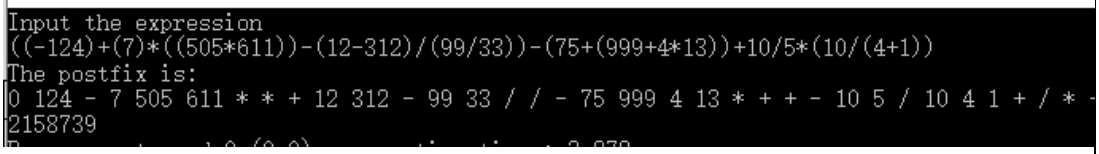
结果正确

测试二：

输入：

((-124)+(7)*((505*611))-(12-312)/(99/33))-(75+(999+4*13))+10/5*(10/(4+1))

输出：



The postfix is:

0 124 - 7 505 611 * * + 12 312 - 99 33 / / - 75 999 4 13 * + + - 10 5 / 10 4 1 + / * +

2158739

结果正确

4. 分析与探讨（结果分析，若存在问题，探讨解决问题的途径）

结果分析：

数据正确，但是因为后缀表达式与前缀表达式是用数组存储的，受限于空间，无法存储过大的数据，所以最好改用数组描述的线性表。

5. 附录：实现源代码（本实验的全部源程序代码，程序风格清晰易理解，有充分的注释）

```
#include <iostream>
#include <cstdio>
#include <string.h>
using namespace std;

template <class T>
class arrayStack
{
```

```
    private:
        int stackTop;  //栈顶
        int arrayLength;   //容量
        T* Stack;   //栈元素
    public:
        arrayStack(int initialCapacity = 10);  //构造函数
        ~arrayStack() {delete []Stack;}   //析构函数
        bool Empty() const {return stackTop == -1;}
        int Size() const {return stackTop+1;}
        T& Top()    //栈顶元素
        {
            if (stackTop == -1)
            {
                cout << "The stack is empty" << endl;
                throw "The stack is empty";
            }
            return Stack[stackTop];
        }
        void Pop()   //出栈
        {
            if (stackTop == -1)
            {
                cout << "The stack is empty" << endl;
                return;
            }
            Stack[stackTop--].~T();
        }
        void Push(T& theElement);     //入栈
};

/*改变数组的长度*/
template <class T>
void changeLength1D(T*& a, int oldLength, int newLength)
{
    if (newLength < 0)    //新数组长度小于0，抛出异常
    {
        cout << "wrong capacity" << endl;
        return;
    }
    T* temp = new T[newLength];  //新数组
    int number;                 //需要复制的元素数
    if (oldLength<newLength)
        number = oldLength;
    else
```

```
        number = newLength;
    for (int i=0; i<number; i++)
        temp[i] = a[i];
    delete []a;    //删除旧数组
    a = temp;      //复制新的数组
}

/*构造函数*/
template <class T>
arrayStack<T>::arrayStack(int initialCapacity)
{
    if (initialCapacity<1)
    {
        cout << "wrong capacity" << endl;
        return;
    }
    arrayLength = initialCapacity;
    Stack = new T[arrayLength];
    stackTop = -1;    //初始化为空栈
}

/*入栈*/
template <class T>
void arrayStack<T>::Push(T& theElement)
{
    if (stackTop == arrayLength-1)   //栈满
    {
        changeLength1D(Stack,arrayLength,2*arrayLength);
        arrayLength *=2;
    }
    Stack[++stackTop] = theElement;
}

/*计算*/
class Calculate
{
    private:
        char infix[1000];   //计算表达式
        char postfix[1000];  //后缀表达式
        arrayStack<char> ope; //存储运算符的栈
        arrayStack<int> num;  //存储计算数的栈
    public:
        void Input ()
        {
```

```
            cout << "Input the expression" << endl;
            cin >> infix;
        }
        bool Isnum(char c)  //判断是否为数字
        {
            if (c>='0' && c<='9')
                return true;
            else
                return false;
        }
        bool compare(char, char);//比较优先级
        void trans();  //转化为后缀表达式
        void calculation();   //计算
};

/*比较优先级*/
bool Calculate::compare(char stack_top, char infix_top)
{//优先级顺序 '(' < '+' = '-' < '*' < '/' < ')'
    if ((stack_top == '+') && (infix_top == '+')) return true;
    if ((stack_top == '+') && (infix_top == '-')) return true;
    if ((stack_top == '-') && (infix_top == '+')) return true;
    if ((stack_top == '-') && (infix_top == '-')) return true;
    if ((stack_top == '*') && (infix_top == '+')) return true;
    if ((stack_top == '*') && (infix_top == '-')) return true;
    if ((stack_top == '*') && (infix_top == '*')) return true;
    if ((stack_top == '*') && (infix_top == '/')) return true;
    if ((stack_top == '/') && (infix_top == '+')) return true;
    if ((stack_top == '/') && (infix_top == '-')) return true;
    if ((stack_top == '/') && (infix_top == '*')) return true;
    if ((stack_top == '/') && (infix_top == '/')) return true;
    if (infix_top == ')') return true;
    return false;
}

/*转化为后缀表达式*/
void Calculate::trans()
{//后缀表达式的每个元素“值”的后面补空格以区别
    int num_postfix = 0;    //后缀表达式数组的下标
    for (int i=0; infix[i]!='\0'; i++)
    {
        if (Isnum(infix[i]))  //如果是数字，直接放入后缀表达式
        {
            postfix[num_postfix++] = infix[i];
            if (!Isnum(infix[i+1]))
```

```
                postfix[num_postfix++] = ' ';  //多位的数在最后一位后面插入空值以区分
        }
        else
        {
            if (ope.Empty())
                ope.Push(infix[i]);  //若栈为空，则直接让操作符入栈
            else
            {
                if (infix[i] != '(' && infix[i] != ')')    //若操作符不是括号。
                {
                    if(compare(ope.Top(),infix[i]))         //需要判断优先级。
                    {
                        while((!ope.Empty())                          && (compare(ope.Top(),infix[i])))    //栈中优先级高于等于中缀表达式时，需要出栈到后缀表达式
                        {
                            postfix[num_postfix++] = ope.Top();
                            postfix[num_postfix++] = ' ';
                            ope.Pop();
                        }
                        ope.Push(infix[i]);     //然后入栈优先级符合规定的操作符
                    }
                    else   //优先级正确，直接入栈
                        ope.Push(infix[i]);
                }
                if (infix[i] == '(')  //若当前元素为(,则直接入栈
                {
                    if (infix[i+1]=='+' || infix[i+1]=='-')
                    {
                        postfix[num_postfix++] = '0';
                        postfix[num_postfix++] = ' ';
                    }
                    ope.Push(infix[i]);
                }
                if (infix[i] == ')')   //若当前元素为),准备出栈
                {
                    while((!ope.Empty()) && (ope.Top()!='('))   //出栈至(,将出栈的元素放入输出的数组中
                    {
                        postfix[num_postfix++] = ope.Top();
```

```
                        postfix[num_postfix++] = ' ';
                        ope.Pop();
                    }
                    if ((!ope.Empty()) && (ope.Top()=='('))
                        ope.Pop();
                }
            }

        }
    }
    while(!ope.Empty())  //栈中剩余元素放入后缀表达式
    {
        postfix[num_postfix++] = ope.Top();
        postfix[num_postfix++] = ' ';
        ope.Pop();
    }
    postfix[num_postfix] = '\0';
    cout << "The postfix is: " << endl;
    cout << postfix;
    cout << endl;
}

/*计算*/
void Calculate::calculation()
{
    int i = 0;
    while (postfix[i]!='\0')  //遍历后缀表达式
    {
        if (postfix[i] == ' ')  //为空格，跳过（空格为了区别不同的元素
        {
            i++;
            continue;
        }
        else if (Isnum(postfix[i]))  //如果是数字，直接压入 int 型栈中
        {
            int sum = 0;
            while (Isnum(postfix[i]))  //对多位数的操作
            {
                sum = sum*10 + postfix[i]-'0';
                i++;
            }
            num.Push(sum);
```

```
        }
        else  //如果是操作符，从栈中取两个数进行相应操作，然后将结果压入栈，注意减法与除法的运算数顺序
        {
            if (postfix[i] == '+')
            {
                int a1 = num.Top();
                num.Pop();
                int a2 = num.Top();
                num.Pop();
                int res = a1+a2;
                num.Push(res);
                i++;
            }
            else if (postfix[i] == '-')
            {
                int b1 = num.Top();
                num.Pop();
                int b2 = num.Top();
                num.Pop();
                int res = b2-b1;
                num.Push(res);
                i++;
            }
            else if (postfix[i] == '*')
            {
                int c1 = num.Top();
                num.Pop();
                int c2 = num.Top();
                num.Pop();
                int res = c1*c2;
                num.Push(res);
                i++;

            }
            else if (postfix[i] == '/')
            {
                int d1 = num.Top();
                num.Pop();
                int d2 = num.Top();
                num.Pop();
                int res = d2/d1;
                num.Push(res);
                i++;
```

```
            }
        }
    }
    cout << num.Top();  //输出计算结果
    num.Pop();//清空栈
}

int main()
{
    Calculate ca;
    ca.Input();
    ca.trans();
    ca.calculation();
    return 0;
}
```