数据结构项目文档

> 题目：表达式转换
>
> 指导教师：张颖
>
> 姓名：王亮
>
> 学号：1653340

一、题目分析

1. 项目简介

算数表达式有前缀表示法，中缀表示法和后缀表示法等形式。日常使用的算术表达式是采用中缀表示法，即二元运算符位于两个运算数中间。请设计程序将中缀表达式转换成为后缀表达式。

2. 功能需求

1. 输入说明：输入在一行中给出以空格分隔不同对象的中缀表达式，可包含+, -, *, /, -, *, /以及左右括号，表达式不超过20个字符（不包括空格）。
2. 输出说明：在一行中输出转换后的后缀表达式，要求不同对象（运算数，运算符号）之间以空格分隔，但是结尾不得有多余空格。
3. 测试用例：

序号	输入	输出	说明
1	2 + 3 * (7 - 4) + 8 / 4	2 3 7 4 - * + 8 4 / +	正常测试6种运算符
2	((2 + 3) * 4 - (8 + 2)) / 5	2 3 + 4 * 8 2 + - 5 /	嵌套括号
3	1314 + 25.5 * 12	1314 25.5 * +	运算数超过1位整数且有非整数出现
4	-2 * (+3)	-2 3 +	运算数有正或负号
5	123	123	只有1个数字

3. 设计思路

首先确定项目采用栈作为数据结构，定义类的成员变量和成员函数；然后实现将中缀表达式转换为后缀表达式的函数；最后完成主函数以验证程序的功能并得到运行结果。

二、设计

1. 数据结构设计

使用栈可以将表达式的中缀表示转换成它的前缀表示和后缀表示。因此本题目使用栈作为数据结构。

2. 类的设计

为了实现表达式表示形式的转换，实现一个顺序栈。

2.1 SeqStack

顺序栈的模板类：

```
const int stackIncreament = 20; //栈溢出时扩展空间的增量

template <typename T>class SeqStack{
public:
    SeqStack(int sz =50);
    ~SeqStack(){
```

```
        delete []elements;
    }
    void Push(const T &x);
    bool Pop(T &x);
    bool getTopData(T &x)const;
    bool getElementByIndex(int index, T &x)const;
    bool IsEmpty()const{
        return top == -1;
    }
    bool IsFull()const{
        return top == maxSize-1;
    }
    int getSize()const{
        return top+1;
    }
    void MakeEmpty(){
        top = -1;
    }
    friend ostream& operator << (ostream &out, SeqStack<T> &s)  {
        out << "Index of top is: " << s.top << endl;
        for (int i = 0; i <= s.top; i ++)
            out << i << ": " << s.elements[i] << endl;
        return out;
    }
private:
    T *elements;
    int top;
    int maxSize;
    void overflowProcess();
};
```

类中主要包括三个成员数据：元素数组指针、栈顶位置下标、最大栈空间。

求解表达式转换问题主要用到类中的 `Push(const T& x)` 和 `Pop(T &x)` 两个方法。

3. 函数设计

3.1 表达式转换

为了实现表达式的转换，需要考虑各操作符的优先级。

操作符ch	#	(	*, /, %	+, -	)
isp	0	1	5	3	6
icp	0	6	4	2	1

isp叫做栈内优先数，icp叫做栈外优先数。

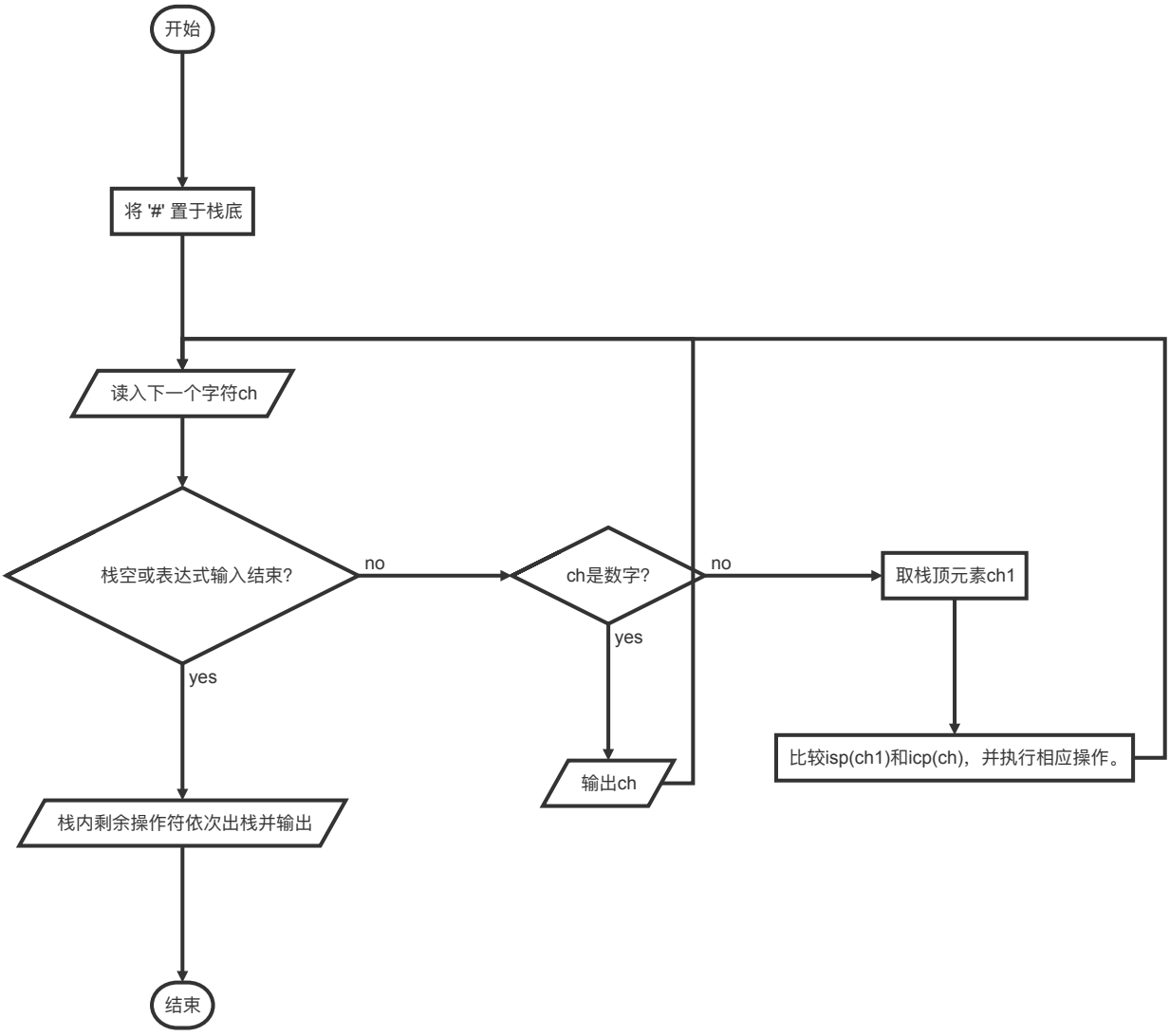
扫描中缀表达式，将它转换成后缀表达式的算法描述如下：

1. 操作符初始化，将结束符#进栈。然后读入中缀表达式字符流的首字符ch。
2. 重复执行以下步骤，知道ch='#'，同时栈顶的操作符也是'#'，停止循环。
    1. 若ch是操作数直接输出，读入下一字符ch。
    2. 若ch是操作符，判断ch的优先级isp和当前位于栈顶的操作符op的优先级isp：
        - 若icp(ch) > isp(op)，令ch进栈，读入下一个字符ch。
        - 若icp(ch) < isp(op)，退栈并输出。
        - 若icp(ch) == isp(op)，退栈但不输出，若退出的是'('号读入一下个字符ch。
    3. 算法结束，输出序列即为所需的后缀表达式

三、实现

1. 表达式转换

1.1 表达式转换流程图


开始
将 '#' 置于栈底
读入下一个字符ch
栈空或表达式输入结束?
no
ch是数字?
no
取栈顶元素ch1
yes
yes
比较isp(ch1)和icp(ch)，并执行相应操作。
输出ch
栈内剩余操作符依次出栈并输出
结束

1.2 关键代码

```
void postFix(SeqStack<char>& s)
{
    cout<<"请输入中缀表达式（每个字符间用空格分隔，结尾没有多余的空格）："<<endl;
    bool haveCheckedNeviOrPosi = false;
    string tempDigit = "";
    char ch = '#', ch1, op;
    s.Push(ch);
    cin.get(ch);
    while (s.IsEmpty() == false)
    {
        //清空tempDigit
        tempDigit = "";

        if( isDigit(ch, tempDigit, haveCheckedNeviOrPosi) )
        {
            cout<<tempDigit;

            if (ch=='\n')
            {
                if(s.getSize()>1)
                    cout<<' ';
                break;
            }
            cout<<' ';
            cin.get(ch);
        }
        else
        {
            if(tempDigit == "-")
                ch = '-';
            if(tempDigit == "+")
                ch = '+';

            s.getTop(ch1);
            if(icp(ch) > isp(ch1))
            {
                s.Push(ch);

                if(ch=='-' || ch=='+')
                    haveCheckedNeviOrPosi = false;

                if(ch != '-' && ch != '+')
                    cin.get(ch);

                if (ch=='\n')
                    break;
                cin.get(ch);
            }
            else if(icp(ch) < isp(ch1))
```

```
            {
                s.Pop(op);
                cout<<op<<' ';

            }
            else{
                s.Pop(op);
                if(op == '(')
                {
                    cin.get(ch);
                    if (ch=='\n')
                        break;
                    cin.get(ch);
                }
            }
        }
    }

    //操作符栈内剩余的操作符依次出栈并输出
    while(s.getSize()>1)
    {
        s.Pop(op);
        cout<<op;
        if(s.getSize()>1)
            cout<<' ';
    }
}
```

1.3 特殊情况处理

数字可能是负数、带正号的整数、小数。在判断是否数字的函数isDigit中处理。

2. 判断是否是数字

由于数字不仅有普通的正整数，还可能出现负数、带正号的整数和小数。

负数情况需要将符号与减号进行区分。

相关代码：

```
/*
 *@brief: 判断是否是数字
 */
bool isDigit(char& ch, string& tempDigit, bool& haveChecked)
{
    if(ch == '-' || ch == '+')
    {
        if(haveChecked)
            return false;
        tempDigit += ch;
```

```
            cin.get(ch);
        }
        while(isdigit(ch) || ch=='.')
        {
            tempDigit += ch;
            cin.get(ch);
        }

        if(tempDigit.length() > 0 && tempDigit != "-" && tempDigit != "+")
            return true;
        else
        {
            if(tempDigit == "-" || tempDigit == "+")
                haveChecked = true;
            return false;
        }
    }
```

四、测试

1. 功能测试（包含各种边界情况）

1.1 正常测试六种运算法

测试结果：

```
请输入中缀表达式（每个字符间用空格分隔，结尾没有多余的空格）：
2 + 3 * ( 7 - 4 ) + 8 / 4
2 3 7 4 - * + 8 4 / +Program ended with exit code: 0
```

测试结果正确。

1.2 嵌套括号

测试结果：

```
请输入中缀表达式（每个字符间用空格分隔，结尾没有多余的空格）：
( ( 2 + 3 ) * 4 - ( 8 + 2 ) ) / 5
2 3 + 4 * 8 2 + - 5 /Program ended with exit code: 0
```

测试结果正确。

1.3 运算数超过1位整数且有非整数出现

测试结果：

测试结果正确。

1.4 运算数有正号或负号

测试结果：

测试结果正确。

1.5 只有1个数字

测试结果：

测试结果正确。

五、性能评价

该算法对输入表达式只进行一次自左向右的扫描，对每个操作数只执行一次输出，其执行时间为O(1)。对每一个操作数，执行进栈和退栈各一次，其时间也为O(1)。若设表达式中的符号的总数为n，则总的实行时间复杂度为O(n)。