《数据结构》课程实践报告

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 院、系 | 计算机学院 | | 年级专业 | 21计科 | 姓名 | 方浩楠 | 学号 | 2127405048 |
| 实验布置日期 | |  | | 提交  日期 | 9.26 | | 成绩 |  |

课程实践实验2：栈的实现和应用

## 一、问题描述及要求

**（1）实现顺序栈类并测试正确性；**

**（2）完成后缀表达式求值；**

**如：输入：3.5 20 4+\*20 4/-**

**输出：79**

**不同操作数之间以空格分隔，当表达式输入错误时，给出错误提示。**

**（3）完成中缀表达式求值；**

**中缀表达式以一个字符串的形式读入，可含有加、减、乘、除 运算符和左、右括号。**

**如：输入3.5\*(20+4)-20/4**

**输出：79。**

**当表达式输入错误时，给出错误提示。**

**（4）选做：HTML文档标签匹配**

**超文本标记语言（HTML）中标签的匹配问题与括号匹配问题类似。HTML中的大部分标签以开始（opening tag，<tag>）和结束（closing tag，</tag>）的形式存在，它们必须成对出现以正确描述web文档。以下是一个非常简单的HTML文档。写一个程序，用于检查HTML文档中的开始和结束标签是否匹配。**

**<html>**

**<head>**

**<title>example</title>**

**</head>**

**<body>**

**<center>**

**<h3>Welcome!</h3>**

**<font size = 10>**

**stay hungry, stay foolish!**

**</font>**

**</center>**

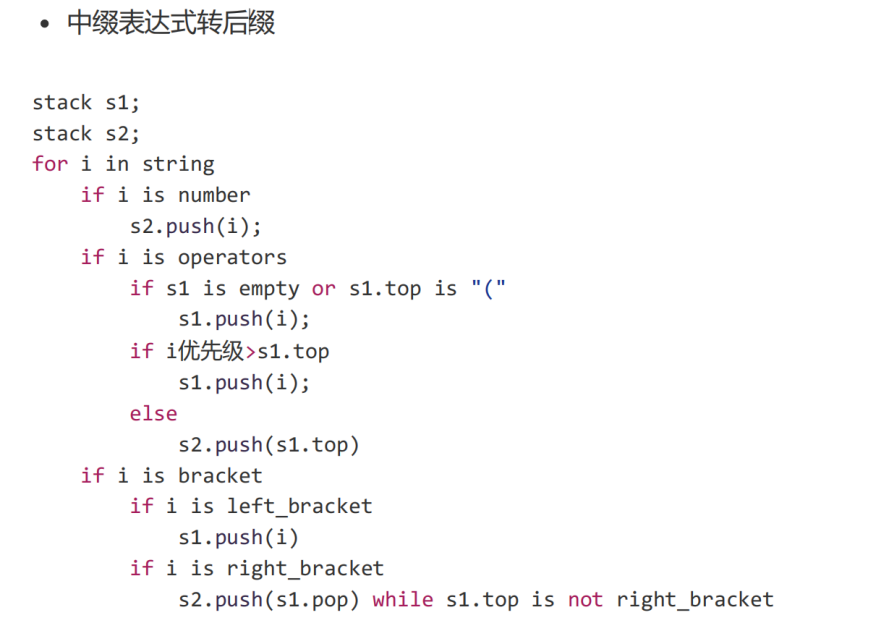
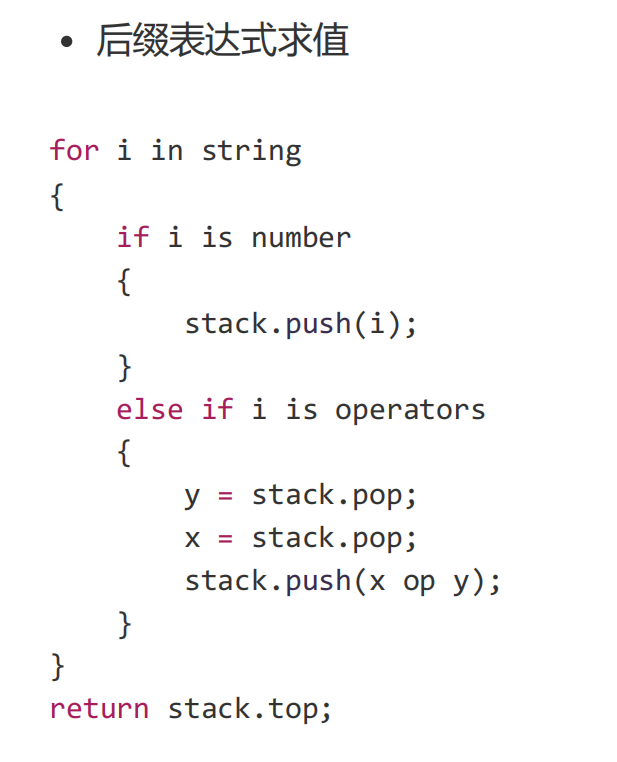
**</body>**

**</html>**

## 概要设计

首先需要设计一个顺序栈类,其次利用后缀表达式计算，中缀表达式转换为后缀表达式之后再进行计算。

## 三、详细设计



## 实验结果

测试输入:3.5 20 4 + \* 20 4 / -

预计输出:79

实际输出:79

## 实验分析与探讨

算法的时间复杂度:

算法的空间复杂度:

如何优化:可将顺序栈转换为链栈，这样在对栈进行push操作时可以节省空间

## 六、小结

本次实验我了解到了stack的使用，并且了解到了顺序栈，以及前缀和后缀表达式

## 附录：源代码

实验环境:Visual Studio 2022

2、

文件名:1.cpp

//（1）实现顺序栈类并测试正确性；

#include<iostream>

#include<vector>

using namespace std;

const int stack\_size = 20;

template<typename DataType>

class SeqStack

{

public:

    SeqStack();

    ~SeqStack();

    void Push(DataType x);

    DataType Pop();

    DataType GetTop();

    int Empty();

private:

    DataType data[stack\_size];

    int top; //栈顶的下标

}

void Seqstack<Datatype>::Push(Datatype x)

{

    if(top == stack\_size-1){throw "stackoverflow";} //判断stack是否上溢

    else{data[top++] = x;}

}

Datatype SeqStack<Datatype>::pop()

{

    if(top == -1){throw"stackoverflow";} //判断stack是否下溢

    else{x = data[top--];return x;} //弹出stack的栈顶元素

}

Datatype Seqstack<Datatype>::GetTop()

{

    if(top == -1){throw"stackoverflow";} //判断stack是否下溢

    else{return data[top];}

}

int Seqstack::Empty()

{

    if(top == -1){return 0;}

    else{return 1;}

}

文件名:2.cpp

#include<iostream>

#include<stack>

#include<sstream>

#include<fstream>

using namespace std;

double PostfixExpression(string lst[],int n)

{

    stack<double> st;

    string operators = "+-\*/";

    int i = 0;

    double num;

    while(i<n)

    {

        string m = lst[i];

        if(operators.find(m)==string::npos) //判断第i个元素是否是表达式

        {//无法在operators中找到m，说明第i个元素是数字，需要入栈

            istringstream iss(m);

            iss >> num;

            st.push(num);

        }

        else

        {//在operators中找到了m，说明第i个元素是运算符，需要计算

            double y = st.top(); st.pop();

            double x = st.top(); st.pop();

            double z;

            string temp\_operators = m;

            if(temp\_operators == "+"){z = x+y;}

            else if(temp\_operators == "-"){z = x-y;}

            else if(temp\_operators == "\*"){z = x\*y;}

            else if(temp\_operators == "/"){z = x/y;}

            st.push(z);

        }

        i+=1;

    }

    return st.top();

}

int main()

{

    string postfix\_exression = "3.5 20 4 + \* 20 4 / - "; //待计算的后缀表达式

    string postfix\_exression\_array[20]; //存放后缀表达式的字符串

    istringstream ss(postfix\_exression);

    int i = 0;

    while(ss>>postfix\_exression\_array[i])

    {

        i++; //i为后缀表达式中的元素的数量,

    }

    /\*

    cout<<"输入的后缀表达式为"<<postfix\_exression<<endl;

    cout<<"计算结果为:";

    \*/

    cout<<PostfixExpression(postfix\_exression\_array,i)<<endl;

    system("pause");

    return 0;

}

文件名:3.cpp

#include <iostream>

#include <stack>

#include <sstream>

#include <fstream>

#include <map>

using namespace std;

int SignToInt(string sign) //将运算符号以及括号用int来表达

{

    if (sign == "+"){return -1;}

    else if (sign == "-"){return -2;}

    else if (sign == "\*"){return -3;}

    else if (sign == "/"){return -4;}

    else if (sign == "("){return -5;}

    else if (sign == ")"){return -6;}

}

stack<double> infixExpression(string lst[], int n)

{

    stack<double> sign\_stack; // sigh\_stack用来存放符号

    stack<double> temp\_stack; // temp\_stack用来存放中间结果

    string operators1 = "+-\*/";

    string operators2 = "+-\*/" ;

    string brackets = "()";

    string left\_bracket = "(";

    string right\_bracket = ")";

    int i = 0;

    double num;

    while (i < n)

    {

        string m = lst[i];

        if (operators1.find(m) == string::npos) //判断第i个元素是否是运算符

        {                                         //无法在operators中找到m，说明第i个元素是数字，需要入栈

            istringstream iss(m);

            iss >> num;

            temp\_stack.push(num);

        }

        else if (SignToInt(m) == -5) //判断第i个元素是否是左括号

        {                            //第i个元素为左括号

                                     //如果是左括号就直接压入sign\_stack

            sign\_stack.push(-5);

        }

        else if (SignToInt(m) == -6) //判断第i个元素是否是右括号

        {                            //第i个元素是右括号

                                     //如果是右括号，则依次弹出sign\_stack的栈顶，将弹出的元素压入temp\_stack，直到遇到右括号为止，并将这一对括号丢弃

            while (true)

            {

                if (sign\_stack.top() != -6)

                {

                    temp\_stack.push(sign\_stack.top());

                    sign\_stack.pop();

                }

                else

                {

                    sign\_stack.pop();

                    break;

                }

            }

        }

        else if (operators2.find(m) != string::npos) //判断第i个元素是否是否是运算符

        {                                              //第i个元素是运算符

        Label1:;

            if (sign\_stack.empty() or sign\_stack.top() == -5)

            { // 1.若sign\_stack为空，或者栈顶运算符为左括号，则直接入栈

                sign\_stack.push(SignToInt(m));

            }

            else if ((SignToInt(m) == -3 or SignToInt(m) == -4) and (sign\_stack.top() == -1 or sign\_stack.top() == -2))

            { // 2.优先级高于栈顶运算符，也直接入栈

                sign\_stack.push(SignToInt(m));

            }

            else

            { // 3.否则，将sigh\_stack栈顶运算符弹出压入temp\_stack，再转到步骤1进行判断

                temp\_stack.push(sign\_stack.top());

                sign\_stack.pop();

                goto Label1;

            }

        }

        i += 1;

    }

    return temp\_stack;

}

string InfixToPost(stack<double> st, string lst[]) //将中缀表达式转换为后缀表达式

{

    for(int i = 0;i<st.size();i++)

    {

        lst+=st.top();

        st.pop();

    }

}

double PostfixExpression(string lst[], int n)

{

    stack<double> st;

    string operators = "+-\*/";

    int i = 0;

    double num;

    while (i < n)

    {

        string m = lst[i];

        if (operators.find(m) == string::npos) //判断第i个元素是否是表达式

        {                                      //无法在operators中找到m，说明第i个元素是数字，需要入栈

            istringstream iss(m);

            iss >> num;

            st.push(num);

        }

        else

        { //在operators中找到了m，说明第i个元素是运算符，需要计算

            double y = st.top();

            st.pop();

            double x = st.top();

            st.pop();

            double z;

            string temp\_operators = m;

            if (temp\_operators == "+")

            {

                z = x + y;

            }

            else if (temp\_operators == "-")

            {

                z = x - y;

            }

            else if (temp\_operators == "\*")

            {

                z = x \* y;

            }

            else if (temp\_operators == "/")

            {

                z = x / y;

            }

            st.push(z);

        }

        i += 1;

    }

    return st.top();

}

int main()

{

    string infix\_exression = "3.5 \* ( 20 + 4 ) - 20 / 4"; //待计算的中缀表达式

    string infix\_exression\_array[20];                     //存放中缀表达式的字符串

    istringstream ss(infix\_exression);

    int i = 0;

    while (ss >> infix\_exression\_array[i])

    {

        i++; // i为中缀表达式中的元素的数量,

    }

    cout << PostfixExpression(infix\_exression\_array, i) << endl;

    system("pause");

    return 0;

}