《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 | 彭光 | 学号 | 2127405052 |
| 实验布置日期 | | 2022-09-06 | | 提交  日期 | 2022-09-26 | | 成绩 |  |

课程实践实验2：栈的实现与运用

## 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。

## 概要设计

对实验内容的理解：

实验（1）主要是让我们自己构建一个stack类来实现进栈出栈等操作；

实验（2）主要是让我们知道遇到一个运算符时，需要弹出栈里的两个元素进行运算。最后如果栈内只剩下一个double数据就将它进行输入，如果栈是空的或者有两个或以上元素，就输出“illegal expression”提示。

实验（3）主要是从两个方面进行切入，先是中缀表达式转后缀表达式存在string列表中，在运用实验（2）中的prefix函数进行后缀表达式求值。

系统的功能列表：

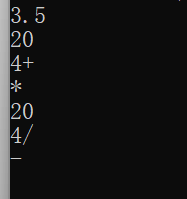
1. 中缀表达式求值
2. 后缀表达式求值
3. 中缀表达式转或追表达式

程序运行的界面设计：

首先出现屏幕提示，输入一则字符串，如果是后缀表达式，直接运行即可；如果是中缀表达式，同样输入字符串，直接运行就行。

总体设计思路：

首先来说明一下后缀表达式求值，这里面我只调用了一个函数（prefix），在这个函数里面的接口为string数组和里面的元素个数。这里面输入要求是不同操作数之间以空格分隔，即3.5 20 4+\*20 4/-，需要解决运算符与运算符之间无空格的问题，这里我采用的是首先将操作数给分开也就是用空格作为标识符，string数组里面都为空字符串，采用prefix\_array[k] += prefix\_ex[i];的方法将不是+-\*/这几个运算符的东西加到一个空字符串内，不断地走。这里出现了一个问题就是

像这样存在操作数与运算符在一起。对于这个问题我是去运用if ((prefix\_ex[i + 1] == '+' || prefix\_ex[i + 1] == '-' || prefix\_ex[i + 1] == '\*' || prefix\_ex[i + 1] == '/'))

{

k++;

}这样的方法（看i+1是否是运算符，如果是，k继续++）

接下来prefix函数部分我运用了将操作数进栈的方法，如果遇到运算就将栈里的两个元素pop出来，注意是后者操作符前者，然后不断地进行运算，直到栈里只剩一个double数据输出，如果出现其他情况，就出现不合法提示。

其次说明一下中缀表达式求值：

数显是输入方面，要求的是输入一则字符串，但是字符串里没有空格隔开，对于这个要求，我们利用+-\*/（）这六个运算符来隔开其中的操作数，但是有的是两个运算符在一起，我用的方法是继续进行k++，虽然会string数组里会出现空字符串，我用的方法是再继续调用vector<string>的向量来push进去，再倒置，将有空字符串的进行删除，最后再存入另一个string数组里。就像这样vector<string> y;

for (int i = 0; i <=k; i++)

{

y.push\_back(p[i]);

}

for (int i = y.size() - 1; i >= 0; i--)

{

if (y[i] == "")

{

y.erase(y.begin() + i);

}

}

for (int i = 0; i < y.size(); i++)

{

q[i] = y[i];

}

接下来就是中缀转后缀的方式。这里我先运用了一个fuzhi函数来给运算符赋值，因为是要比较运算符的优先级，所以我定义”（“和#为0，使它的优先级最低，在中缀转后缀的过程中用了一个ixx函数，首先如果是”（“要进栈，如果是”）“的话要把栈里的元素都给pop出来直到遇见”（“才截止；如果是+-\*/就要判断优先级了，调用fuzhi函数：ifa[i]优先级比y1的栈顶优先级高，a[i]就进栈，如果不是就要求y1一直出栈，直到遇见比a[i]低的，再让a[i]进栈。到最后让y2出栈。因为y2都是倒放的，所以我们要建一个string数组将他们倒放，才能转正。最后调用prefix函数求出值。

## 三、详细设计

后缀表达式求值主函数：先把”不同操作数用空格间隔“的字符串转化为string数组，然后调用prefix函数来进行求值。

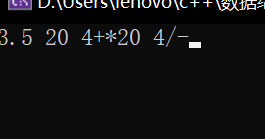
后缀表达式算法：运用遇到一个运算符时，需要弹出栈里的两个元素进行运算。最后如果栈内只剩下一个double数据就将它进行输入，如果栈是空的或者有两个或以上元素，就输出“illegal expression”提示。

中缀转后缀主函数：先把无任何间隔的字符串转化为string数组再进行ixx求值。

中缀转后缀算法：先是中缀表达式转后缀表达式存在string列表中，在运用实验（2）中的prefix函数进行后缀表达式求值。

## 四、实验结果

测试输入：



测试目的：看结果是否出现错误

正确输出：79

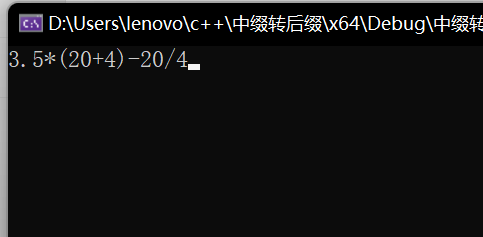
实际输出：

屏幕截图 2022-09-26 192148

错误原因：无

测试结论：通过

测试输入：



测试目的：看是否会出现错误

正确输出：

79

实际输出：

屏幕截图 2022-09-26 192400

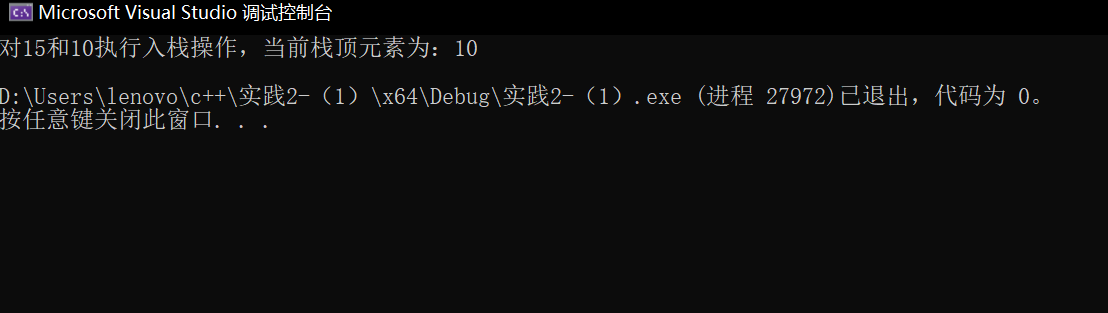
错误原因：无

测试结论：通过

测试输入：无

测试目的：检查顺序栈类的实行

正确输出：10

实际输出：

错误原因：无

测试结论：通过

## 五、实验分析与探讨

测试结果分析：

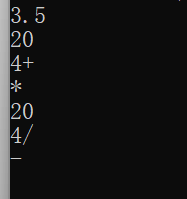
时间复杂度：O（n\*\*2）

空间复杂度：O（n）

探讨：我觉得利用两个程序进行分批的求值可能会对用户造成不小的困扰，我觉得可以利用string数组和元素计数这个接口来将prefix这个函数作为公用部分来进行求值，我还觉得应该用一些cout文字说明让用户自己来选择使用那种表达式求值，让用户达到最好的体验。

问题与解决方法：

这里出现了一个问题就是

像这样存在操作数与运算符在一起。对于这个问题我是去运用if ((prefix\_ex[i + 1] == '+' || prefix\_ex[i + 1] == '-' || prefix\_ex[i + 1] == '\*' || prefix\_ex[i + 1] == '/'))

{

k++;

}这样的方法（看i+1是否是运算符，如果是，k继续++）

## 小结

通过这个实验我了解到了顺序栈的运用方式，我希望能够更多的了解顺序栈。完成了1，2，3部分，选做部分未完成，程序没有放在一起，有点不太方便。

## 附录：源代码

**1、实验环境：Visual Studio 2022**

2、

（1）ixx.cpp

#include "ixx.h"

#include<iostream>

#include<string>

#include<stack>

#include<sstream>

#include<algorithm>

using namespace std;

double prefix(string p[], int h)

{

stack<double> x;

for (int i = 0; i < h; i++)

{

if (!(p[i] == "\*" || p[i] == "+" || p[i] == "-" || p[i] == "/"))

{

x.push(stod(p[i]));

}

else

{

if (x.empty() || x.size() == 1)

{

cout << "illegal expression" << endl;

exit(0);

}

double a = x.top();

x.pop();

double b = x.top();

double c;

x.pop();

if (p[i] == "+") c = a + b;

else if (p[i] == "-") c = b - a;

else if (p[i] == "\*") c = a \* b;

else if (p[i] == "/") c = b / a;

x.push(c);

}

}

if (x.size() != 1)

{

cout << "illegal expression" << endl;

exit(0);

}

return x.top();

}

int main() {

string prefix\_ex;

getline(cin, prefix\_ex);

string prefix\_array[20];

/\*stringstream ss(prefix\_ex);

int i = 0;

while (ss >> prefix\_array[i])

i++;\*/

int k = 0;

for (int i = 0; i < prefix\_ex.size(); i++)

{

if (prefix\_ex[i] != ' ')

{

if (prefix\_ex[i] == '+' || prefix\_ex[i] == '-' || prefix\_ex[i] == '\*' || prefix\_ex[i] == '/')

{

if ((prefix\_ex[i + 1] == '+' || prefix\_ex[i + 1] == '-' || prefix\_ex[i + 1] == '\*' || prefix\_ex[i + 1] == '/'))

{

k++;

}

prefix\_array[k] += prefix\_ex[i];

k++;

}

else

{

prefix\_array[k] += prefix\_ex[i];

}

}

else

{

k++;

}

}

/\*for (int i = 0; i <= k; i++)

{

cout << prefix\_array[i] << endl;

}\*/

cout << prefix(prefix\_array, k) << endl;

}

（2）源.cpp

#include<stack>

#include<string>

#include<iostream>

#include<vector>

using namespace std;

int fuzhi(string& x)

{

if (x == "("||x=="#")

return 0;

else if (x == "+" || x == "-")

return 1;

else if (x == "\*" || x == "/")

return 2;

}

double prefix(string p[], int h)

{

stack<double> x;

for (int i = 0; i < h; i++)

{

if (!(p[i] == "\*" || p[i] == "+" || p[i] == "-" || p[i] == "/"))

{

x.push(stod(p[i]));

}

else

{

if (x.empty() || x.size() == 1)

{

cout << "illegal expression" << endl;

exit(0);

}

double a = x.top();

x.pop();

double b = x.top();

double c;

x.pop();

if (p[i] == "+") c = a + b;

else if (p[i] == "-") c = b - a;

else if (p[i] == "\*") c = a \* b;

else if (p[i] == "/") c = b / a;

x.push(c);

}

}

if (x.size() != 1)

{

cout << "illegal expression" << endl;

exit(0);

}

return x.top();

}

double ixx(string a[],int m)

{

stack<string> y1,y2;

y1.push("#");

for (int i = 0; i <= m; i++)

{

if (!(a[i] == "+" || a[i] == "-" || a[i] == "\*" || a[i] == "/" || a[i] == "(" || a[i] == ")"))

{

y2.push(a[i]);

}

else

{

//if (y1.top() == "#")

//y1.push(a[i]);

if (a[i] == "(")

y1.push(a[i]);

else if (a[i] == ")")

{

while (y1.top() != "(")

{

y2.push(y1.top());

y1.pop();

}

y1.pop();

}

else

{

if (fuzhi(a[i]) >= fuzhi(y1.top()))

{

y1.push(a[i]);

}

else

{

while (fuzhi(a[i]) < fuzhi(y1.top()))

{

y2.push(y1.top());

y1.pop();

if (fuzhi(a[i]) >= fuzhi(y1.top()))

{

y1.push(a[i]);

break;

}

}

}

}

}

}

vector<string> b;

while (y1.top() != "#")

{

y2.push(y1.top());

y1.pop();

}

while (!y2.empty())

{

b.push\_back(y2.top());

y2.pop();

}

string d[100];

int z = 0;

for (int i = b.size() - 1; i >= 0; i--)

{

d[z] = b[i];

z++;

}

return prefix(d,z);

}

int main()

{

string s;

getline(cin, s);

string p[100],q[100];

int k = 0;

for (int i = 0; i < s.size(); i++)

{

if (!(s[i] == '+' || s[i] == '-' || s[i] == '\*' || s[i] == '/' || s[i] == '(' || s[i] == ')'))

{

p[k] += s[i];

}

else

{

k++;

p[k] += s[i];

k++;

}

}

int h = 0;

vector<string> y;

for (int i = 0; i <=k; i++)

{

y.push\_back(p[i]);

}

for (int i = y.size() - 1; i >= 0; i--)

{

if (y[i] == "")

{

y.erase(y.begin() + i);

}

}

for (int i = 0; i < y.size(); i++)

{

q[i] = y[i];

}

int count = y.size() - 1;

//for (int i = 0; i <= count; i++)

//cout << q[i] << endl;

cout << ixx(q,count) << endl;

return 0;

}

1. 实验1.cpp

#include<iostream>

#include<vector>

using namespace std;

const int StackSize = 100;

template<typename DataType>

class SeqStack

{

public:

SeqStack();

~SeqStack();

void Push(DataType x);

DataType Pop();

DataType GetTop();

int empty();

DataType\* data;

int top;

};

template<typename DataType>

SeqStack<DataType>::SeqStack()

{

data= new DataType[StackSize];

top = -1;

}

template<typename DataType>

SeqStack<DataType>::~SeqStack()

{

if (data) delete data;

}

template<typename DataType>

void SeqStack<DataType>::Push(DataType x)

{

if (top == StackSize - 1) throw"上溢";

data[++top] = x;

}

template<typename DataType>

DataType SeqStack<DataType>::Pop()

{

DataType x;

if (top == -1) throw"下溢";

x = data[top--];

return x;

}

template<typename DataType>

DataType SeqStack<DataType>::GetTop()

{

if (top != -1) {

return data[top];

}

else throw"栈已空";

}

template<typename DataType>

int SeqStack<DataType>::empty()

{

if (top == -1)return 1;

else

return 0;

}

int main()

{

SeqStack<int> S;

cout << "对15和10执行入栈操作，";

S.Push(15);

S.Push(10);

cout << "当前栈顶元素为：" << S.GetTop() << endl;

return 0;

}