《数据结构》课程实践报告

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 院、系 | 计算机学院 | | 年级专业 | 23软件工程 | 姓名 | 梅子羽 | 学号 | 2327406107 |
| 实验布置日期 | | 2024.10.7 | | 提交  日期 | 2024.10.17 | | 成绩 |  |

课程实践实验2：栈的实现及其应用

## 一、问题描述及要求

表达式求值是编译程序的一个基本问题。设运算符有+、-、\*、/、#和圆括号，其中#为表达式的定界符。

表达式求值需要根据运算符的优先级来确定计算顺序。因此在求值过程中需要保存优先级较低的运算符以及没有参与运算的运算对象，并将当前运算符与以及扫描过的、尚未计算的运算符进行比较，以确定哪个运算符以及哪两个运算对象参与计算。

运算符在两个运算对象的中间（如4+2）称为中缀表达式；

运算符在两个运算对象的后面（如4 2 +）称为后缀表达式，也称逆波兰式。

复杂一点的表达式：

中缀：( 4 + 2 ) \* 3 - 5

后缀：4 2 + 3 \* 5 –

题目：栈的实现及应用

具体要求：

1. 实现自己的顺序栈类

2. 完成中缀表达式求值

要求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和左右括号，并假设该表达式以“#”作为输入结束符。

如

输入：3.5\*(20+4)-20/4#

输出：79

题目（续）

3. 完成后缀表达式求值

（1）输入带#引导符的后缀形式

（2）输入带空格的后缀表达式

如：

输入：3.5 20 4 + \* 20 4 / - #

输出：79

4. 实现前缀表达式求值

【问题描述】设计前缀表达式求值算法double prefix(string prefix\_ex[], int n) ，假设前缀表达式中的各逻辑成分依次存储在长度为n的字符串数组prefix\_ex中，如前缀表达式“\* 5.0 - 20 6”，则n为5，prefix\_ex数组的内容为{"\*","5.0","-","20","6}。

主程序测试时，输入以空格分隔的前缀表达式各个逻辑成分，输出表达式计算结果，如输入表达式不合法，则输出“illegal expression”。表达式中的操作数是浮点数。

【输入形式】以空格分隔的前缀表达式字符串

【输出形式】前缀表达式的值

【样例输入】\* 5 - 20 6

【样例输出】70

## 二、概要设计

### （1）对实验内容的理解

本实验旨在设计一个可以处理各种数学表达式（包括中缀、前缀和后缀表达式）的计算器。通过实现堆栈数据结构和相应的算法，我们将能够解析并计算输入的表达式。此系统将帮助用户理解不同表达式形式的计算过程，同时增强对数据结构和算法的理解。

### （2）系统功能列表

* **基本功能**
  + 支持中缀表达式的计算
  + 支持前缀表达式的计算
  + 支持后缀表达式的计算
* **用户交互**
  + 从键盘输入表达式
  + 提供计算结果
  + 显示计算过程的详细信息
* **数据管理**
  + 支持不同数据类型（如整数、浮点数）
  + 处理基本的算术运算（加、减、乘、除）
* **错误处理**
  + 处理输入错误（如非法字符、错误的括号配对等）

### （3）程序运行的界面设计

Please input the series:

用户输入。

Which expression is it? (p: prefix, i: infix, s: suffix)；

用户输入后，程序将显示相应的计算结果。

### （4）总体设计思路

#### 数据结构

* **栈（Stack）**：用于存储操作数和运算符。根据表达式类型（中缀、前缀、后缀）进行相应的操作。
* **链表节点（ListNode）**：用于实现栈的基础数据结构，存储数据和指向下一个节点的指针。

#### 类设计

* **ListNode**
  + **作用**：用于定义栈节点，存储一个值和指向下一个节点的指针。
  + **方法**：构造函数、析构函数（如有需要）。
* **Stack**
  + **作用**：实现栈的基本操作（入栈、出栈、查看栈顶元素、判断栈是否为空）。
  + **方法**：
    - push(Datatype x): 入栈操作。
    - pop(): 出栈操作，返回栈顶元素。
    - peek(): 返回栈顶元素但不出栈。
    - isEmpty(): 判断栈是否为空。
    - printOut(): 打印栈中所有元素。
* **Calculator**
  + **作用**：解析和计算不同形式的表达式。
  + **方法**：
    - infixSolu(string &s): 计算中缀表达式。
    - suffixSolu(string &s): 计算后缀表达式。
    - prefixSolu(string &s): 计算前缀表达式。
    - operation(double num1, double num2, const string &op): 执行基本运算。

### （5）程序结构设计

#### 程序文件

* **Stack.h**: 定义栈的数据结构和基本操作。
* **Stack.cpp**: 实现栈的功能。
* **main.cpp**: 包含程序的主逻辑，处理用户输入和调用相应的计算函数。

#### 各部分关系描述

* main.cpp 通过包含 Stack.h 来访问栈类，并使用栈类的实例进行计算操作。
* Calculator 类将与 Stack 类紧密结合，以实现不同表达式形式的解析和计算。
* 错误处理将贯穿整个程序，通过用户输入和函数返回值进行反馈。

这种设计方法将保证系统的可维护性和可扩展性，易于添加新的功能或改进现有功能。

## 三、详细设计

#### 1. 主函数设计

主函数负责整体程序的控制流程。它从标准输入获取一个表达式，然后提示用户选择表达式的类型（前缀、后缀或中缀），根据用户的选择调用相应的解析和计算函数，最后输出结果。

#### 2. 关键算法设计

关键算法包括解析表达式和执行计算，分别对应中缀、后缀和前缀表达式的处理。以下是每种表达式的设计：

* **中缀表达式处理（infixSolu）**
  + 将表达式字符逐个读取，判断是数字还是运算符。
  + 数字通过临时字符串存储并转换为双精度浮点数后入栈。
  + 运算符通过优先级比较决定入栈或计算（利用栈存储运算符和操作数）。
  + 最终结果从操作数栈中弹出并打印。
* **后缀表达式处理（suffixSolu）**
  + 按字符读取，数字入栈，运算符取出两个操作数进行计算。
  + 每次遇到运算符时，从数值栈中弹出两个数进行运算，结果再入栈。
  + 最终结果在数值栈中弹出并打印。
* **前缀表达式处理（prefixSolu）**
  + 将表达式分割成字符串，存入数组。
  + 从数组反向读取，每次遇到数字时入栈，遇到运算符时取出两个数进行计算。
  + 最终结果在数值栈中弹出并打印。

#### 3. 难点分析

* **表达式的优先级处理**：中缀表达式的处理需要考虑运算符的优先级和括号的嵌套。使用栈来管理操作数和运算符，通过比较优先级决定操作顺序。
* **栈的操作**：栈的实现和管理是核心功能，涉及到入栈、出栈、查看栈顶元素等操作，确保在表达式解析过程中不丢失数据。
* **输入格式的灵活性**：要处理不同格式的输入，确保用户输入的表达式能够正确解析，例如处理空格、负号等情况。

#### 4. 设计思路

* **数据结构选择**：使用栈（Stack类）来管理操作数和运算符，通过链表实现栈的功能。
* **类的设计**：
  + **Stack类**：实现栈的基本操作，如入栈、出栈、查看栈顶元素等。
  + **ListNode结构**：作为栈的节点结构，包含一个值和指向下一个节点的指针。
* **类之间的关系**：Stack类依赖于ListNode结构，每个Stack对象内部维护一个指向栈顶节点的指针。

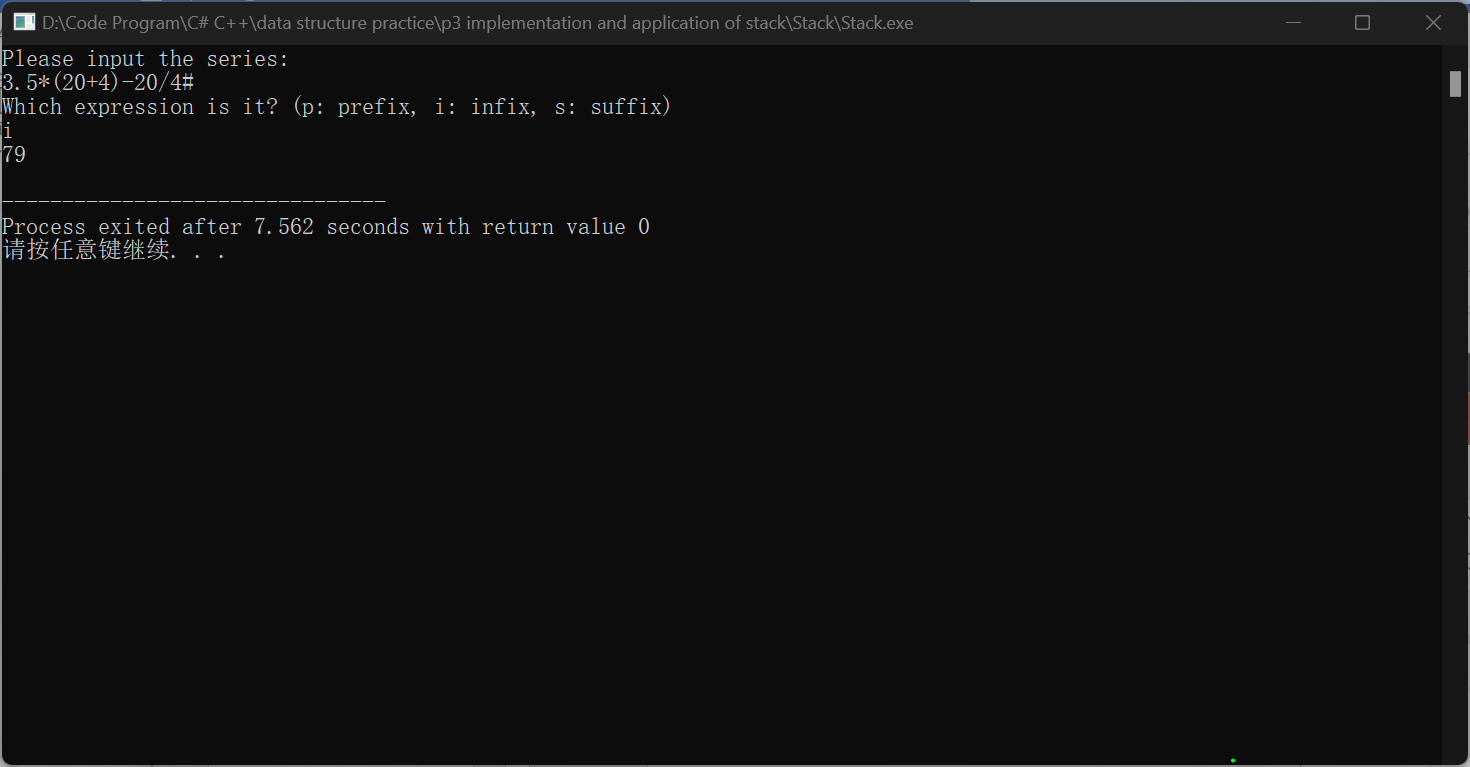
#### 5. 程序结构设计

* **已用程序**：栈的实现（Stack.h和Stack.cpp）。
* **自定义程序文件**：main.cpp实现表达式解析与计算功能。
* **部分关系描述**：
  + main.cpp通过Stack类实现运算符和操作数的管理。
  + 计算功能通过infixSolu、suffixSolu、prefixSolu三个函数分别处理三种表达式类型，形成模块化设计。

## 五、实验结果

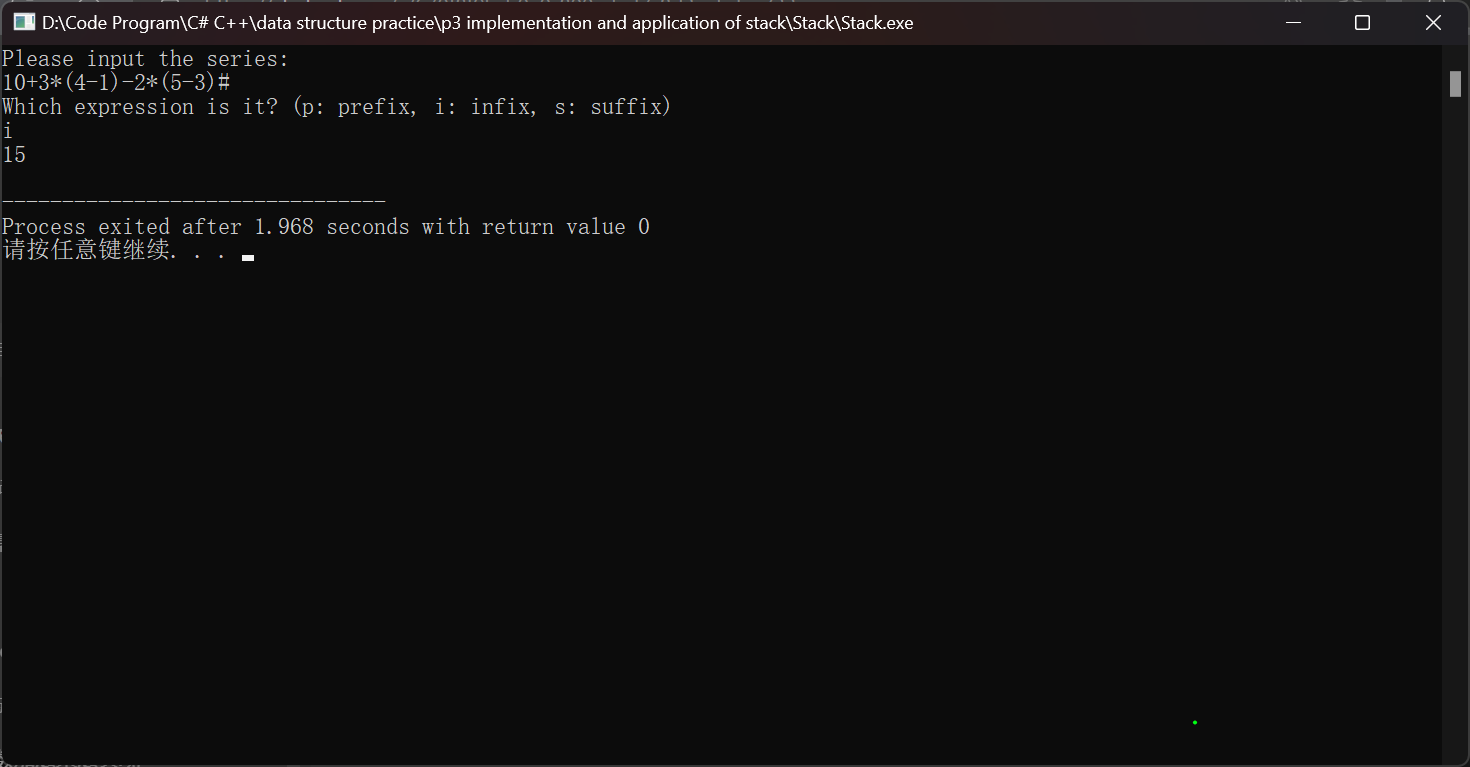
测试一：输入3.5\*(20+4)-20/4#，输出为79

目的为测试中缀表达式对应处理函数的正确性，输出正确



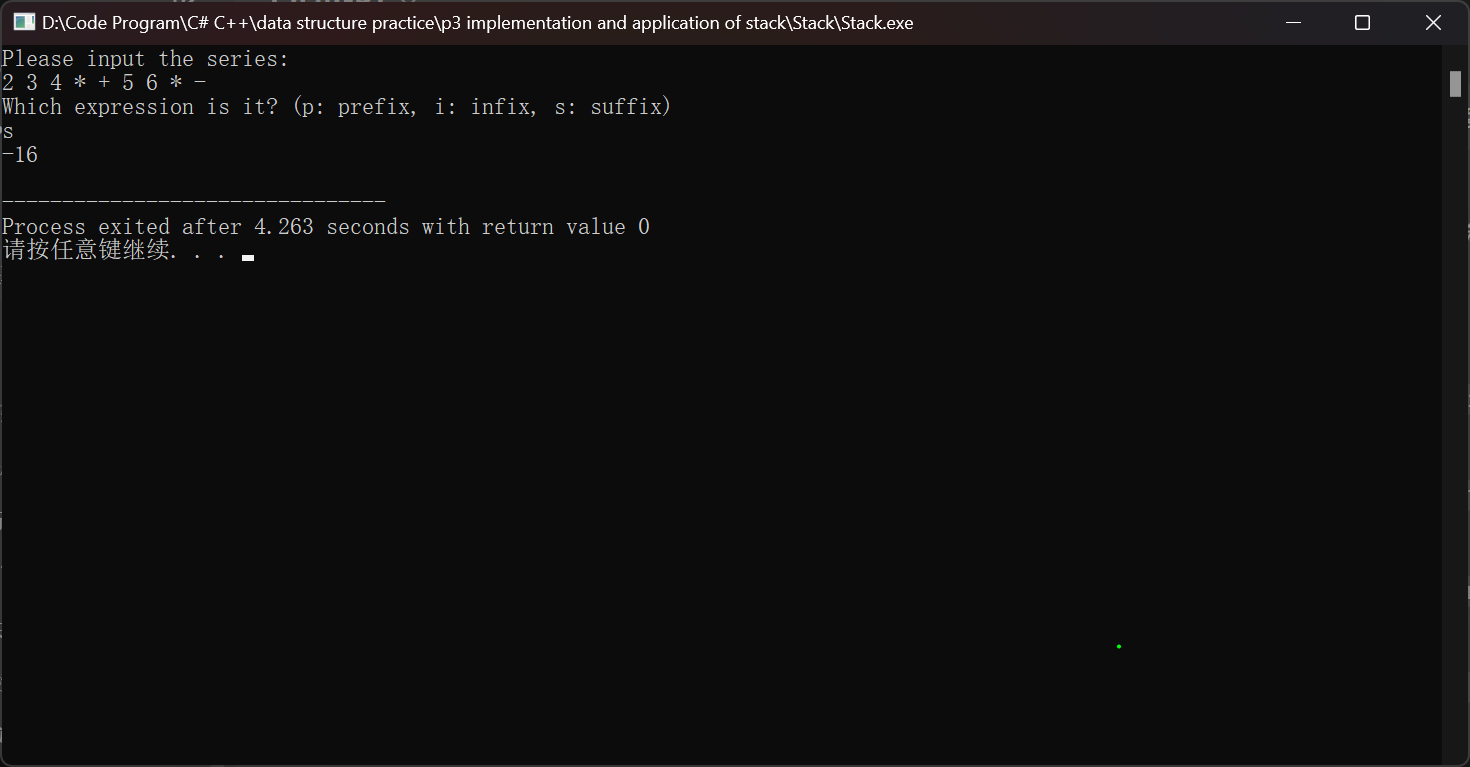
测试二：输入10+3\*(4-1)-2\*(5-3)#，输出为15

目的同上，输出正确



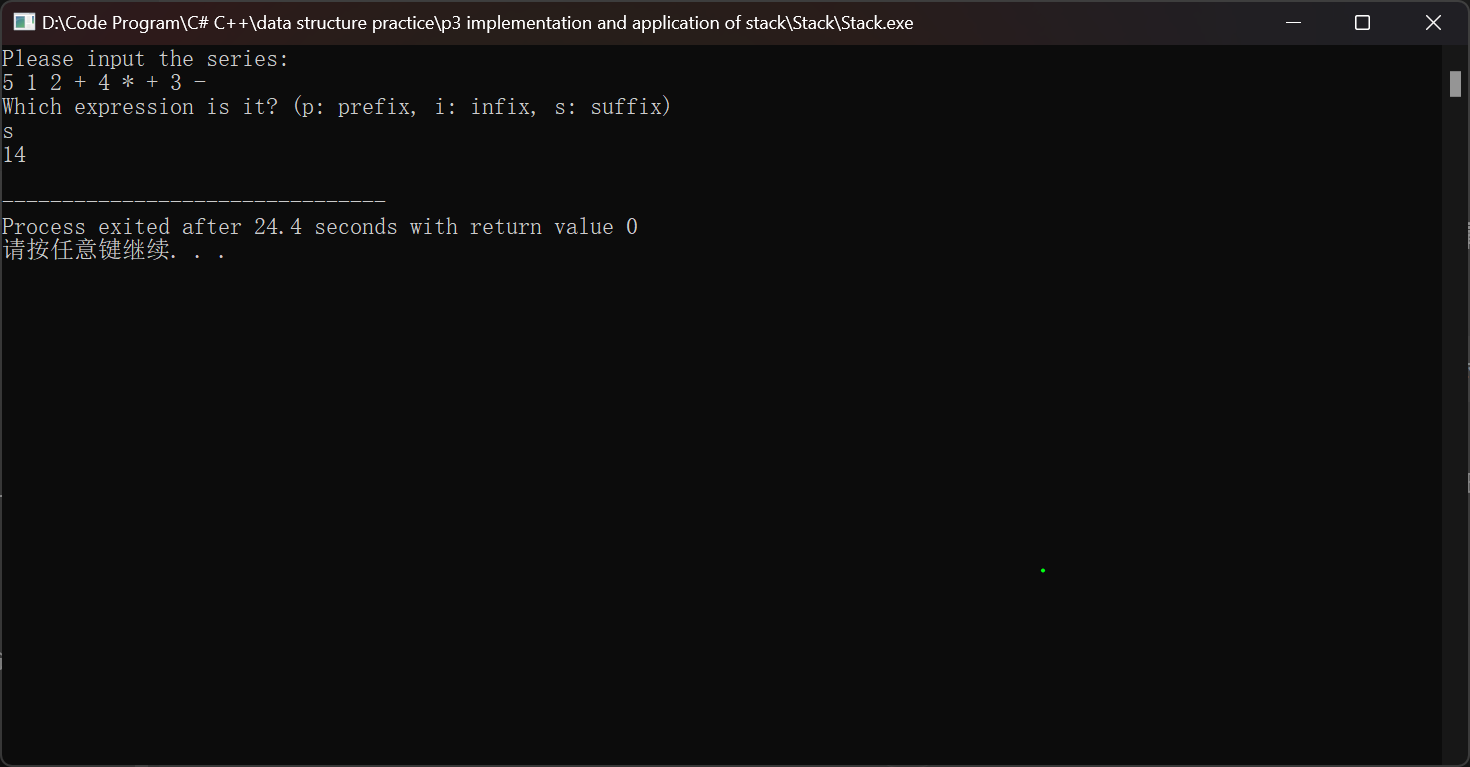
测试三：输入2 3 4 \* + 5 6 \* - ，输出为-16

目的为测试后缀表达式对应处理函数的正确性，输出正确



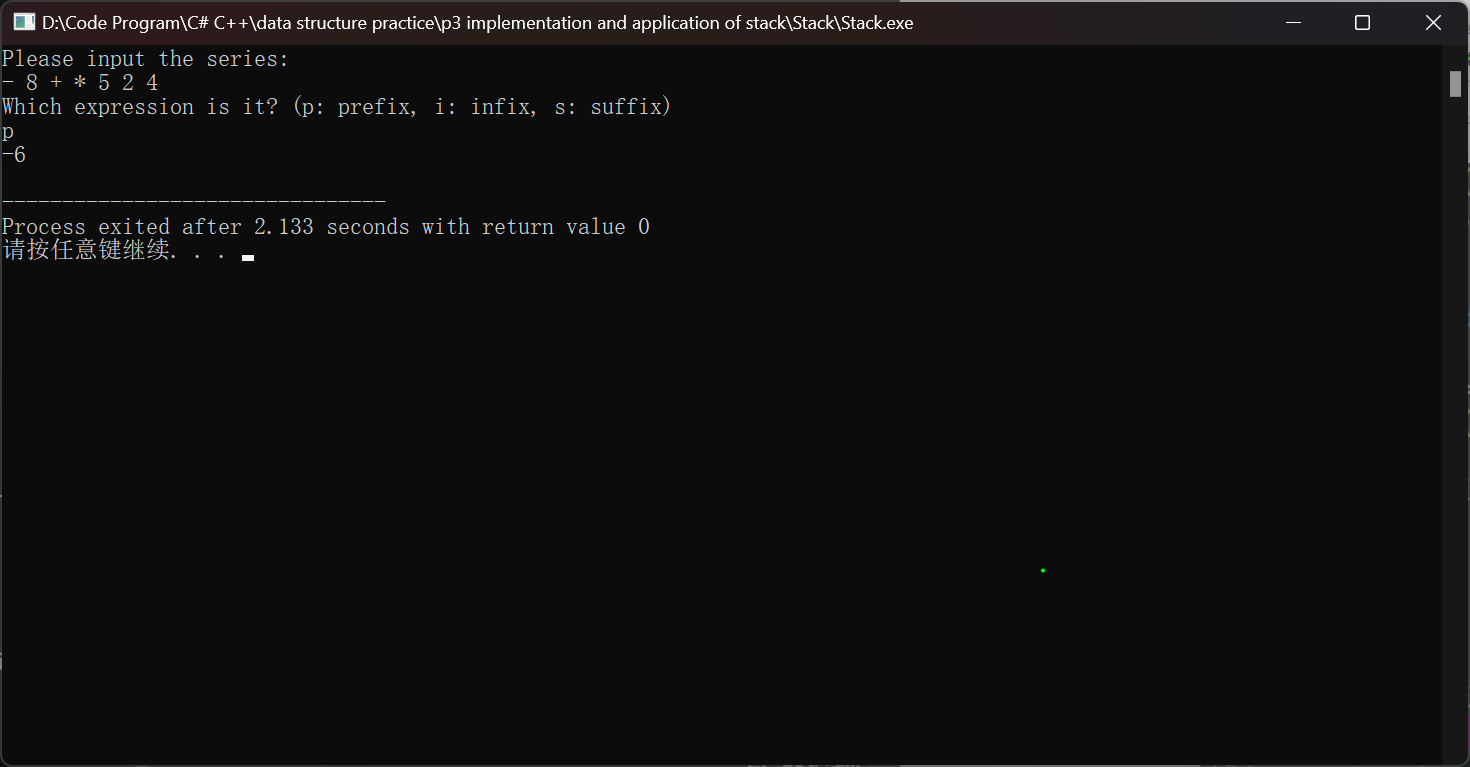
测试四：输入5 1 2 + 4 \* + 3 -，输出为16

目的同上，输出正确



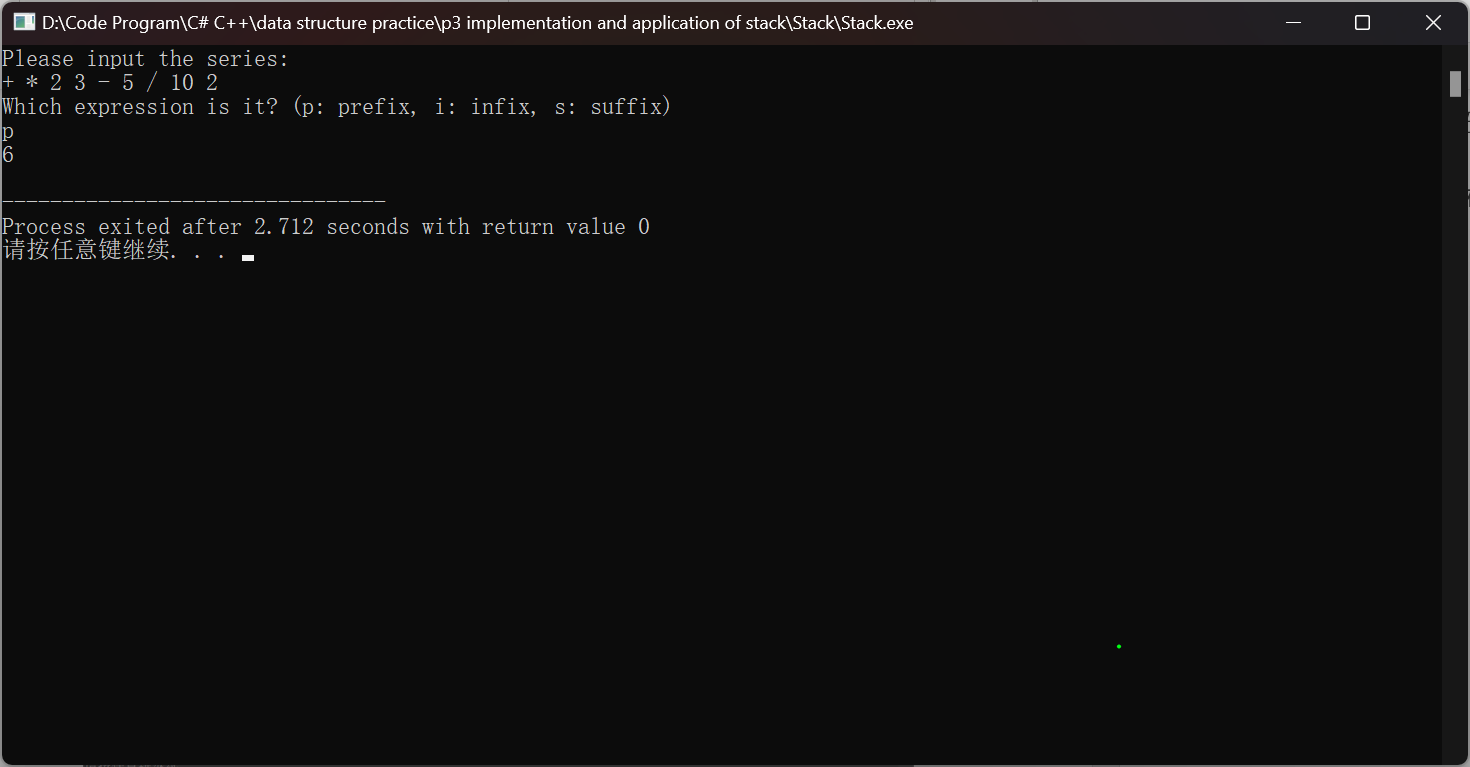
测试五：输入- 8 + \* 5 2 4，输出为-6

目的为测试前缀表达式对应处理函数的正确性，输出正确



测试六：输入+ \* 2 3 - 5 / 10 2，输出为6；

目的为测试前缀表达式对应处理函数的正确性，本来输入是错误的，原因在于我直接将原本的输入字符串倒置，从而类似于10这样以0结尾的数全部错误，后面才改正，使输出正确



总结：三个主要函数的时间复杂度都是O(n)，这是栈处理的一大优势。本次实验中的难点我觉得主要在于中缀和前缀的处理。中缀需要更多判断和逻辑自洽，尤其是对于mymap的构建，将符号比较抽象成数值比较；而前缀的处理，在没有提示要倒过来时，是很难写完美的。因为符号除了先后进栈，还有优先顺序，而数字没有。如果按照后缀的从左往右读，先读入的就变成了很难先处理的符号，这困扰了我很久。没有一下想到反向处理的一大原因是我觉得只有符号翻转，数字没有翻转会影响，实际上仅仅需要更改一下弹出栈的两个数而已。

## 六、小结

在本次设计与实现过程中，我成功开发了一个简单的表达式求解器，能够处理前缀表达式、中缀表达式和后缀表达式。整个过程涉及以下几个关键步骤：

1. **需求分析**：明确系统需要支持的表达式类型和操作，包括加、减、乘、除等基本运算。
2. **数据结构设计**：选择链表实现的栈结构，用于存储操作数和操作符，以便进行后续的计算。
3. **算法设计**：根据不同的表达式类型，设计相应的求解算法。通过逆波兰表达式、运算符优先级等概念实现了中缀到后缀的转换，并使用栈进行表达式的计算。
4. **界面设计**：简单的控制台界面提示用户输入表达式及选择表达式类型，易于交互。

#### 完成的内容

* 实现了支持前缀、中缀、后缀表达式的求解器。
* 编写了栈类的实现，支持基本的栈操作（如压栈、出栈、查看栈顶元素）。
* 进行了简单的异常处理，确保在栈为空时不会出现错误。
* 提供了控制台界面，用户可以通过输入选择表达式类型。

#### 未完成的内容

* **输入验证**：当前版本未对用户输入进行充分的验证，例如不合法字符的处理和表达式的语法检查。
* **更多运算符支持**：尚未实现对更复杂运算符（如幂运算、取余等）的支持，未来可以考虑添加。
* **浮点数精度处理**：在进行浮点数计算时，可能会遇到精度问题，尚未针对这一问题进行优化。

#### 选做部分

* 目前未实现输入文件的读取功能，用户需要手动输入表达式。未来可以考虑扩展此功能，允许从文件中读取表达式进行计算。

#### 程序的局限性

* 当前程序仅能处理单个表达式，未能实现连续计算或链式操作。
* 对于非常复杂的表达式，可能会因为栈的深度限制而导致运行时错误。
* 输入表达式的格式要求较为严格，用户输入不规范可能导致程序崩溃。

#### 展望

在未来的学习与完善中，可以考虑以下几个方向：

1. **输入增强**：实现更灵活的输入方式，包括从文件读取和从命令行传递参数。
2. **错误处理**：增加对输入错误和异常的处理机制，提供更友好的用户提示。
3. **扩展功能**：考虑实现更多的数学函数（如三角函数、对数函数等）和运算符的支持。
4. **性能优化**：在算法设计上进行进一步优化，提高程序的执行效率，特别是在处理长表达式时。
5. **用户界面**：若时间允许，可以考虑使用图形用户界面（GUI）框架，提高用户体验。

#### 收获与感想

通过本次项目的设计与实现，我深入理解了数据结构和算法的应用，特别是在表达式求解方面的应用。实现栈的结构加深了我对链表和动态内存管理的理解。同时，我也意识到在程序设计中需要充分考虑用户体验和输入的灵活性。未来，我希望能将这些经验应用到更复杂的项目中，继续提升自己的编程能力。

#### 必要的补充说明

* **输入文件格式**：若后续实现文件读取功能，建议输入文件每一行包含一个完整的表达式，行与行之间无空行，且支持的运算符、数字和空格应符合前述要求。
* **表达式的格式**：要求表达式中运算符与操作数之间可以有空格，但不允许出现多个连续的空格或不支持的字符。

这次项目的实现过程为我今后的学习和实践提供了宝贵的经验，我将继续努力，完善和扩展我的程序！

## 附录：源代码

1.实验环境：Dev-C++ 5.11集成开发环境，C++11标准。

2、

（1）//Stack.h

#ifndef STACK\_H

#define STACK\_H

#include <iostream>

#include <string>

#include <cmath>

#include <iomanip>

#include <ctime>

#include <vector>

#include <algorithm>

#include <stdio.h>

#include <cstdlib>

#include <map>

#include <fstream>

#include <sstream>

#include <ctime>

using namespace std;

template <typename Datatype>

struct ListNode

{

Datatype val;

ListNode\* next;

ListNode(): val(), next(nullptr) {}

ListNode(Datatype x): val(x), next(nullptr) {}

};

template <typename Datatype>

class Stack

{

private:

ListNode<Datatype>\* p;

public:

Stack();

void push(Datatype)const;

Datatype pop();

Datatype peek()const;

bool isEmpty()const;

void printOut()const;

};

#endif

（2）//Stack.cpp

#include "Stack.h"

#include <iostream>

#include <string>

#include <cmath>

#include <iomanip>

#include <ctime>

#include <vector>

#include <algorithm>

#include <stdio.h>

#include <cstdlib>

#include <map>

#include <fstream>

#include <sstream>

#include <ctime>

using namespace std;

template <typename Datatype>

Stack<Datatype>::Stack()

{

p = new ListNode<Datatype>();

}

template <typename Datatype>

void Stack<Datatype>::push(Datatype x)const

{

ListNode<Datatype>\* temp = new ListNode<Datatype>(x);

temp->next = p->next;

p->next = temp;

}

template <typename Datatype>

Datatype Stack<Datatype>::pop()

{

if(p->next == nullptr)

{

cout << "Stack is empty!" << endl;

return 0;

}

else

{

Datatype num = p->next->val;

ListNode<Datatype>\* temp = p->next;

p->next = p->next->next;

delete temp;

return num;

}

}

template <typename Datatype>

Datatype Stack<Datatype>::peek()const

{

if(p->next == nullptr)

{

cout << "Stack is empty!" << endl;

return 0;

}

else

{

return p->next->val;

}

}

template <typename Datatype>

bool Stack<Datatype>::isEmpty()const

{

return (p->next == nullptr);

}

template <typename Datatype>

void Stack<Datatype>::printOut()const

{

ListNode<Datatype>\* pc = p->next;

while(pc != nullptr)

{

cout << pc->val << " ";

pc = pc->next;

}

cout << endl;

}

template class Stack<double>;

template class Stack<string>;

（3）//main.cpp

#include "Stack.h"

#include <iostream>

#include <string>

#include <cmath>

#include <iomanip>

#include <ctime>

#include <vector>

#include <algorithm>

#include <stdio.h>

#include <cstdlib>

#include <map>

#include <fstream>

#include <sstream>

#include <ctime>

using namespace std;

map<string, int> mymap =

{

{"+", 3}, {"-", 3},

{"\*", 4}, {"/", 4},

{"(", 2}, {")", 2},

{"#", 1}

};

double operation(double num1, double num2, const string &x)

{

if (x == "+")

return (num1 + num2);

else if (x == "-")

return (num1 - num2);

else if (x == "\*")

return (num1 \* num2);

else

return (num1 / num2);

}

void infixSolu(string &s)

{

Stack<double> infNum;

Stack<string> infOpe;

infOpe.push("#");

string temp = "";

for (auto &x : s)

{

if (!mymap.count(string(1, x)))

temp += x;

else

{

if (temp != "")

{

infNum.push(stod(temp));

temp = "";

}

string ope1 = string(1, x);

string ope2 = infOpe.peek();

while (ope1 != "#" || ope2 != "#")

{

if (ope1 == "(")

{

infOpe.push(ope1);

break;

}

else if (ope2 == "(" && ope1 == ")")

{

infOpe.pop();

break;

}

else if (mymap[ope1] > mymap[ope2])

{

infOpe.push(ope1);

break;

}

else

{

double num2 = infNum.pop();

double num1 = infNum.pop();

infNum.push(operation(num1, num2, ope2));

infOpe.pop();

ope2 = infOpe.peek();

}

}

}

}

cout << infNum.pop() << endl;

}

void suffixSolu(string &s)

{

Stack<double> sufNum;

string temp = "";

int x = 0;

while (x < s.size())

{

while (!mymap.count(string(1, s[x])) && s[x] != '#')

{

if (s[x] != ' ')

{

temp += s[x];

}

else

{

sufNum.push(stod(temp));

temp = "";

}

x++;

}

while (mymap.count(string(1, s[x])) || s[x] == ' ')

{

if (s[x] != ' ' && s[x] != '#')

{

double num2 = sufNum.pop();

double num1 = sufNum.pop();

sufNum.push(operation(num1, num2, string(1, s[x])));

}

x++;

}

}

cout << sufNum.pop() << endl;

}

void prefixSolu(string &s)

{

vector<string> s1;

stringstream ss(s);

string temp;

while (ss >> temp) {

s1.push\_back(temp);

}

Stack<double> prefNum;

int x = s1.size() - 1;

while (x >= 0)

{

while (x >= 0 && !mymap.count(s1[x]))

{

prefNum.push(stod(s1[x]));

x --;

}

while (x >= 0 && mymap.count(s1[x]))

{

double num1 = prefNum.pop();

double num2 = prefNum.pop();

prefNum.push(operation(num1, num2, s1[x]));

x --;

}

}

cout << prefNum.pop() << endl;

}

int main()

{

string s;

cout << "Please input the series:\n";

getline(cin, s);

string x;

cout << "Which expression is it? (p: prefix, i: infix, s: suffix)\n";

cin >> x;

if (x == "i")

infixSolu(s);

else if (x == "p")

prefixSolu(s);

else if (x == "s")

suffixSolu(s);

return 0;

}