项目说明文档

数据结构课程设计

——算数表达式求解

作 者 姓 名： 杨煜

学 号： 1850217

指 导 教 师： 张颖

学院、 专业： 软件学院 软件工程

同济大学

Tongji University

目 录

[1 分析 1](#_Toc28096468)

[1.1 项目内容 1](#_Toc28096469)

[1.2 项目要求 1](#_Toc28096470)

[2 设计 1](#_Toc28096471)

[2.1 数据结构设计 1](#_Toc28096472)

[2.2 栈结构设计 1](#_Toc28096473)

[2.3 成员与操作设计 1](#_Toc28096474)

[3 实现 2](#_Toc28096475)

[3.1 构造函数的实现 2](#_Toc28096476)

[3.1.1 构造函数 2](#_Toc28096477)

[3.1.2 构造功能核心代码 2](#_Toc28096478)

[3.2 压栈功能的实现 3](#_Toc28096479)

[3.2.1压栈功能 3](#_Toc28096480)

[3.2.2 压栈功能核心代码 3](#_Toc28096481)

[3.3 弹出功能的实现 3](#_Toc28096482)

[3.3.1 弹出功能 3](#_Toc28096483)

[3.3.2 弹出功能核心代码 3](#_Toc28096484)

[3.4 获取顶点的实现 4](#_Toc28096485)

[3.4.1 获取顶点 4](#_Toc28096486)

[3.4.2 获取顶点核心代码 4](#_Toc28096487)

[3.5 计算功能的实现 4](#_Toc28096488)

[3.5.1 计算功能 4](#_Toc28096489)

[3.5.2 计算功能核心代码 4](#_Toc28096490)

[3.6 总体功能的实现 5](#_Toc28096491)

[3.6.1 总体功能 5](#_Toc28096492)

[3.6.2 总体系统核心代码 5](#_Toc28096493)

[3.6.3 总体系统流程图 11](#_Toc28096494)

[4 正常测试 13](#_Toc28096495)

[4.1 正常测试 13](#_Toc28096496)

[4.1.1 正常运算符优先级测试 13](#_Toc28096497)

[4.1.2 嵌套运算测试 13](#_Toc28096498)

[4.1.3 运算数超过1位整数测试 14](#_Toc28096499)

[4.1.4 运算数有正或负号测试 15](#_Toc28096500)

[4.2 出错测试 15](#_Toc28096501)

[4.2.1 左括号不匹配 15](#_Toc28096502)

[4.2.2 右括号不匹配 16](#_Toc28096503)

# 1 分析

## 1.1 项目内容

表达式转换在计算机世界中很常见。该程序需求从键盘上输入一个中缀算数表达式，包括括号，随后计算出表达式的值。

## 1.2 项目要求

1、程序对所有输入的表达式作简单的判断，如表达式有错，能给出适当的提示。支持包括加减，乘除取余，乘方和括号等操作符，其中优先级是等于<括号<加减<乘除取余<乘方

2、能处理单目运算符：+或-。

# 2 设计

## 2.1 数据结构设计

如上述项目要求所述，在这个问题中，我们应该使用栈作为我们的数据结构。栈是一种先进后出的数据结构，是一种输入输出受限制的线性表。使用栈可以轻松的将中缀表达式连同括号转化为后缀表达式，而后缀表达式的计算则十分容易方便，以这种方式来求得最后的表达式的值。

## 2.2 栈结构设计

本程序的栈结构的存储结构使用数组来实现，我们可以使用一个数组作为它的存储结构，同时需要一个指针来指示栈顶位置。输出时从栈顶取出元素，同时指针指向上一位置。

## 2.3 成员与操作设计

**栈类（myStack）**

template<class T>

class myStack

{

public:

myStack(int sz = 100);//构造函数，默认长度100

~myStack() { delete[]elements; }//析构函数

void push(const T& x);//压栈

T pop();//弹出

T getTop();//获取顶点

bool isEmpty() { return(top == -1) ? true : false; }//是否为空

bool isFull() { return(top == maxSize - 1) ? true : false; }//是否为满

int top;//顶点

private:

T\* elements;//数组指针

int maxSize;//栈最大大小

};

# 3 实现

## 3.1 构造函数的实现

### 3.1.1 构造函数

构造函数接受一个默认的参数100，置top为-1，开辟数组空间。

### 3.1.2 构造功能核心代码

template<class T>

myStack<T>::myStack(int sz)

{

top = -1;

maxSize = sz;

elements = new T[maxSize];

}

## 3.2 压栈功能的实现

### 3.2.1压栈功能

压栈函数接受要压入的数，将一个新的数据压入栈中，使顶点指针指向这个数据。如果为满，则操作失败。

### 3.2.2 压栈功能核心代码

template<class T>

void myStack<T>::push(const T& x)

{

if (top == maxSize-1)//判断是否满了

{

cerr<<"错误，栈已满";

}

else

{

elements[++top] = x;//压栈

}

};

## 3.3 弹出功能的实现

### 3.3.1 弹出功能

弹出函数返回值为被弹出栈的值，同时指针左移，指向前一个数。如果为空，则失败。

### 3.3.2 弹出功能核心代码

template<class T>

T myStack<T>::pop()

{

if (top == - 1)//判断是否空了

{

cerr << "错误，栈已空";

}

else

{

return elements[top--];//弹出

}

}

## 3.4 获取顶点的实现

### 3.4.1 获取顶点

返回顶点值

### 3.4.2 获取顶点核心代码

template<class T>

T myStack<T>::getTop()

{

return elements[top];

}

## 3.5 计算功能的实现

### 3.5.1 计算功能

计算函数接受三个参数，两个为数字，一个为运算符，根据运算符做不同性质的运算，返回结果。其中使用switch来进行分支运算。

### 3.5.2 计算功能核心代码

int Cal(char a, int b, int c)//运算函数，a为运算符,b、c为两数

{

switch (a)//根据符号不同，分别做运算

{

case '+':

return (b + c);

break;

case '-':

return (b - c);

break;

case '\*':

return (b \* c);

break;

case '/':

return (b / c);

break;

case '^':

return pow(b,c);

break;

case '%':

return (b % c);

break;

}

}

## 3.6 总体功能的实现

### 3.6.1 总体功能

首先创立两个栈实例，一个为运算符栈，一个为数字栈。读入中缀表达式，首先判断，第一个数字是否为负数，即完成单目运算符-，如果是则通过运算将其变成负数。随后则进行表达式的处理，将数字都存入数字栈。读到运算符时，如果运算符栈不为空，则比较运算符优先级，如果前者为高，则优先进行计算。碰到(入栈，碰到)则进行运算直到运算符栈弹出的内容为（。最后则可以完成该中缀表达式的求值。

### 3.6.2 总体系统核心代码

int main(void)

{

myStack<int> stack1;//建立运算符，数字栈

myStack<char> stack2;

char isContinue = 'y';

while (isContinue == 'y')//判别是否继续

{

cout << "请输入表达式:" << endl;

int i = 0;

string str;//接受表达式

cin >> str;

char a;

int b, c;

if (str[0] == '-')//判断是否为负数

{

stack1.push(-(str[i + 1] - '0'));

i += 2;

}

while (str[i] != '=')//判断是否结束

{

if (str[i] == '-' && (str[i - 1] - '0' < 0 || str[i - 1] - '0' > 9))//判断是否为负数

{

stack1.push(-(str[i + 1] - '0'));

i += 2;

continue;

}

if (str[i] == '+' || str[i] == '-')//判断加法

{

if (stack2.top == -1)//判断运算符栈

{

stack2.push(str[i]);

}

else

{

while (stack2.getTop() == '+' || stack2.getTop() == '-' || stack2.getTop() == '\*' || stack2.getTop() == '/' || stack2.getTop() == '^' || stack2.getTop() == '%')//判断优先级

{

a = stack2.pop();

c = stack1.pop();

b = stack1.pop();

stack1.push(Cal(a, b, c));

}

stack2.push(str[i]);//运算结果入栈

}

}

else if (str[i] == '\*' || str[i] == '/' || stack2.getTop() == '%')

{

if (stack2.top == -1)

{

stack2.push(str[i]);

}

else

{

while (stack2.getTop() == '\*' || stack2.getTop() == '/' || stack2.getTop() == '^' || stack2.getTop() == '%')//判断优先级

{

a = stack2.pop();

c = stack1.pop();

b = stack1.pop();

stack1.push(Cal(a, b, c));

}

stack2.push(str[i]);

}

}

else if (str[i] == '^')

{

if (stack2.top == -1)

{

stack2.push(str[i]);

}

else

{

while (stack2.getTop() == '^')

{

a = stack2.pop();

c = stack1.pop();

b = stack1.pop();

stack1.push(Cal(a, b, c));

}

stack2.push(str[i]);

}

}

else if (str[i] == '(')

{

stack2.push(str[i]);

}

else if (str[i] == ')')

{

while (stack2.getTop() != '(')

{

a = stack2.pop();

c = stack1.pop();

b = stack1.pop();

stack1.push(Cal(a, b, c));

if (flag == 0)

{

break;

}

}

stack2.pop();

}

else//多位的整数

{

if (str[i] - '0' <= 0 || str[i] - '0' >= 9)

{

cout << "错误，请输入整数" << endl;

flag = 0;

break;

}

int t = i;

int k = 0;

while (str[t] - '0' >= 0 && str[t] - '0' <= 9)

{

t++;

}

for (int p = i; p < t; p++)

{

k += (str[p] - '0') \* (int)pow(10, t - p - 1);

}

stack1.push(k);

i = t;

continue;

}

i++;

}

if (flag == 1)

{

while (stack2.top != -1)//计算结果

{

a = stack2.pop();

c = stack1.pop();

b = stack1.pop();

stack1.push(Cal(a, b, c));

}

cout << stack1.getTop() << endl;

}

cout << "是否继续（y，n）？";

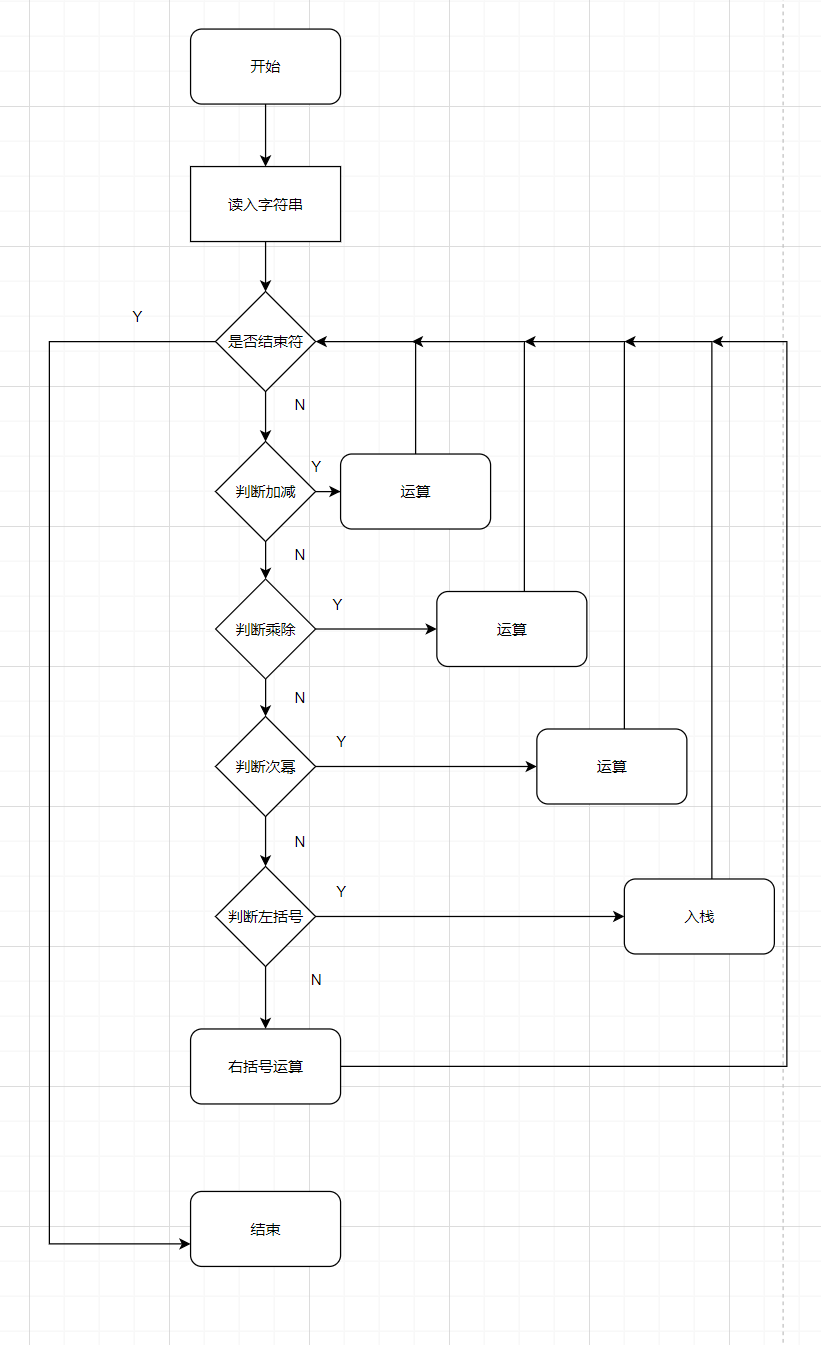
cin >> isContinue;

}

return 0;

}

### 3.6.3 总体系统流程图



# 4 正常测试

## 4.1 正常测试

### 4.1.1 正常运算符优先级测试

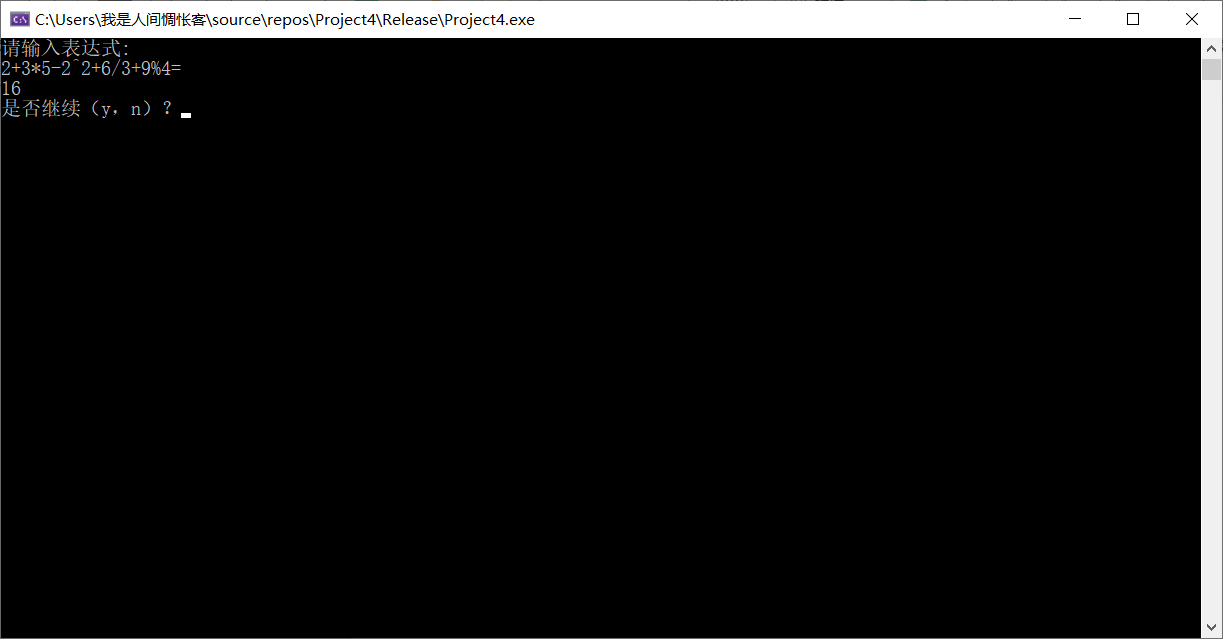
**测试用例**：

2+3\*5-2^2+6/3+9%4=

**预期结果**：

16

**实验结果**



### 4.1.2 嵌套运算测试

**测试用例：**

((2\*(3+4)+5)-2)\*3=

**预期结果：**

**51**

**实验结果：**



### 4.1.3 运算数超过1位整数测试

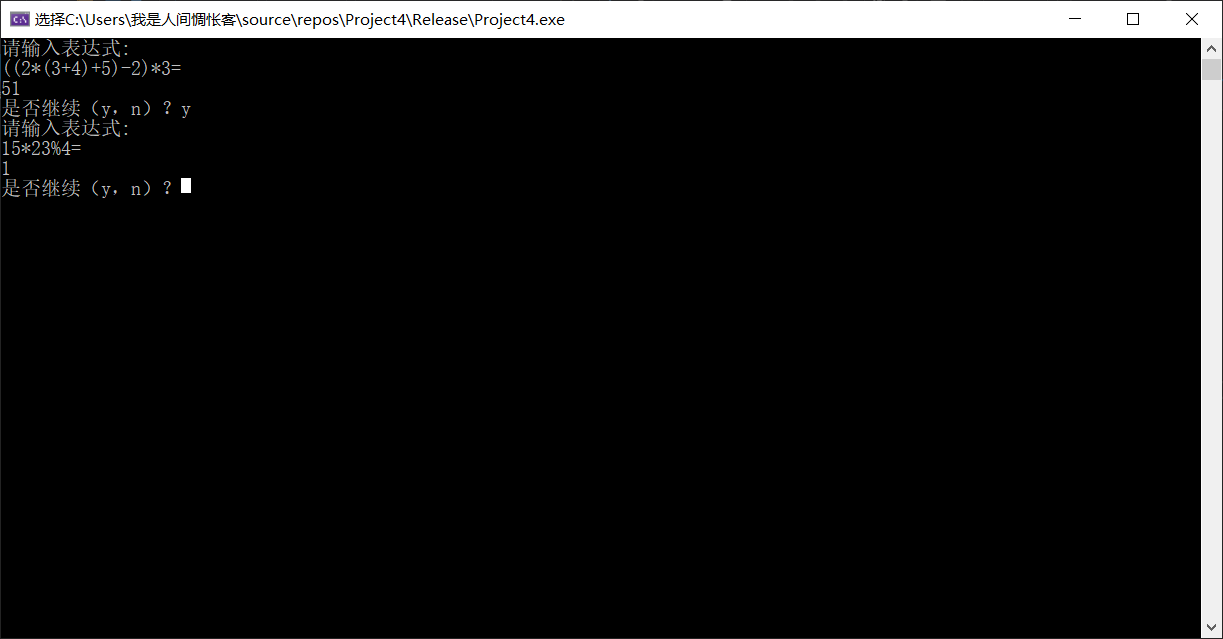
**测试用例：**

**15\*23%4=**

**预期结果：**

1

**实验结果：**



### 4.1.4 运算数有正或负号测试

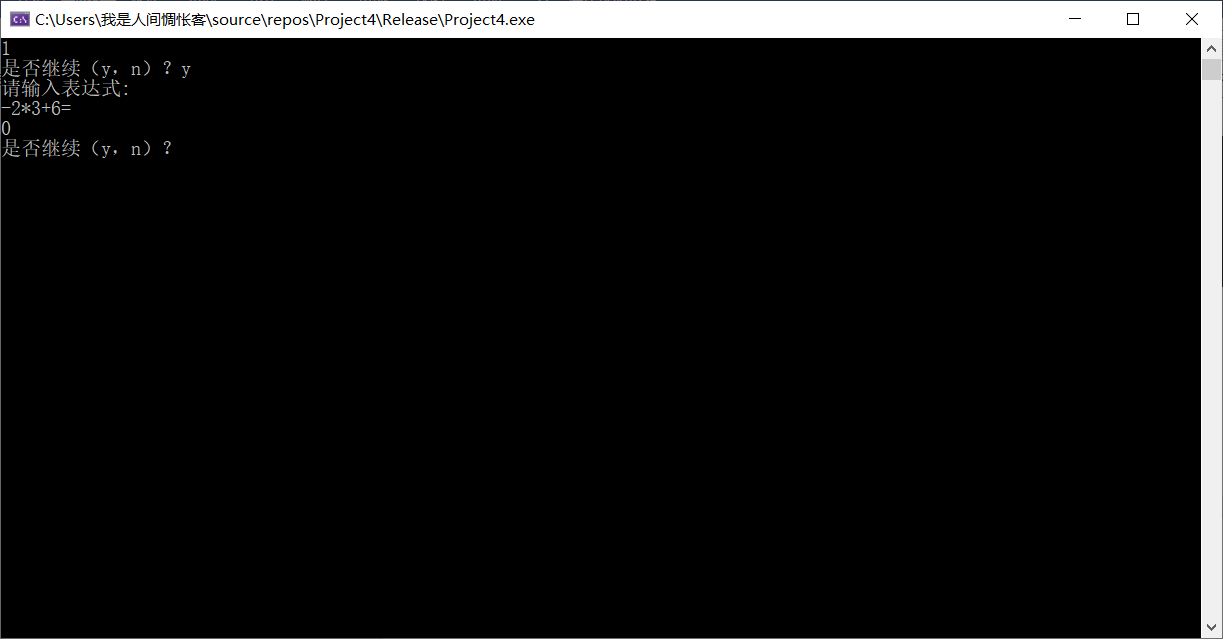
**测试用例：**

**-2\*3+6=**

**预期结果：**

0

**实验结果：**



## 4.2 出错测试

### 4.2.1 左括号不匹配

**测试用例：**

2+3)=

**预期结果：**程序给出提示信息，程序正常运行不崩溃。

**实验结果：**



### 4.2.2 右括号不匹配

**测试用例：**

(2+3=

**预期结果：**程序给出错误信息，程序正常运行不崩溃。

**实验结果：**

