项目说明文档

数据结构课程设计

——算术表达式求值

同济大学

Tongji University

目 录

[1 分析 1](#_Toc59741200)

[1.1 背景分析 1](#_Toc59741201)

[1.2 功能分析 1](#_Toc59741202)

[2 设计 2](#_Toc59741203)

[2.1 数据结构设计 2](#_Toc59741204)

[2.2 类结构设计 2](#_Toc59741205)

[2.3 成员与操作设计 2](#_Toc59741206)

[2.4 系统设计 3](#_Toc59741207)

[3 实现 4](#_Toc59741208)

[3.1 插入功能的实现 4](#_Toc59741209)

[3.1.1 插入功能流程图 4](#_Toc59741210)

[3.1.2 插入功能核心代码 5](#_Toc59741211)

[3.2 去顶功能的实现 6](#_Toc59741212)

[3.2.1 去顶功能流程图 6](#_Toc59741213)

[3.2.2 去顶功能核心代码 6](#_Toc59741214)

[3.3 取顶功能的实现 7](#_Toc59741215)

[3.3.1 取顶功能流程图 7](#_Toc59741216)

[3.3.2 查找功能核心代码 7](#_Toc59741217)

[3.4 表达式求值功能的实现 9](#_Toc59741218)

[3.4.1 表达式求值功能流程图 9](#_Toc59741219)

[3.4.2 表达式求值核心代码 11](#_Toc59741220)

[3.4.3 表达式求值功能截屏示例 12](#_Toc59741221)

[3.5 总体系统的实现 13](#_Toc59741222)

[3.5.1 总体系统流程图 13](#_Toc59741223)

[3.5.2 总体系统核心代码 14](#_Toc59741224)

[3.5.3 总体系统截屏示例 14](#_Toc59741225)

[4 测试 15](#_Toc59741226)

[4.1 基本测试 15](#_Toc59741227)

[4.1.1 加减乘除功能测试 15](#_Toc59741228)

[4.1.2 取余以及乘方功能测试 16](#_Toc59741229)

[4.1.3 括号功能测试 16](#_Toc59741230)

[4.1.4 单目运算符功能测试 16](#_Toc59741231)

[4.1.5 运算优先级测试 17](#_Toc59741232)

[4.2 边界测试 17](#_Toc59741233)

[4.2.1 初始化无输入数据 17](#_Toc59741234)

[4.2.2 用户不输入等于号 18](#_Toc59741235)

[4.2.3 用户输入a^-b 18](#_Toc59741236)

[4.3 出错测试 19](#_Toc59741237)

[4.3.1 除以0 19](#_Toc59741238)

[4.3.2 输入非法字符 20](#_Toc59741239)

[4.3.3 逻辑错误 20](#_Toc59741240)

[4.3.4 y/n输入错误 20](#_Toc59741241)

# 1 分析

## 1.1 背景分析

表达式求值是程序设计语言编译中的一个基本问题，要能让使用者在输入表达式后，让计算机读取相应表达式，并给出计算结果。

实现相关功能是编程实现计算器的一个基础。

## 1.2 功能分析

从键盘上输入中缀算数表达式，包括括号，计算出表达式的值。根据该需求，首先需要能够读取表达式，再进行运算，考虑到用户可能会在表达式的有效信息之间加入空格等，要能够将这些无关字符去除。

其次该表达式能支持的运算符号包括双目的加减，乘除取余，乘方和括号等操作符，以及单目的+和-，它们的优先级次序为等于<括号<加减<乘除取余<乘方。

最后还要能够识别错误输入，输入错误主要指逻辑错误和非法输入，要能够识别并给出错误提示。

综上所述，该程序需要读入表达式，按照优先级规则运算，还要能够进行错误检测。

# 2 设计

## 2.1 数据结构设计

如上功能分析所述，该系统要求能够处理用户输入的表达式，而表达式求值的实现是栈应用的一个典型例子。可以用算符优先法直接对表达式求值，或者是先将表达式转换成后缀表达式然后再对后缀表达式求值。

## 2.2 类结构设计

本程序使用顺序栈，即栈的顺序存储结构是利用一组地址连续的存储单元依次存放自栈底到栈顶的数据元素，同时附设指针top指示栈顶元素在顺序栈中的位置。通常的习惯做法是以top=0表示空栈，鉴于C++中数组的下标约定从0开始,则当以C++作描述语言时,如此设定会带来很大不便；另一方面,由于栈在使用过程中所需最大空间的大小很难估计，因此，一般来说,在初始化设空栈时不应限定栈的最大容量。一个较合理的做法是：先为栈分配一个基本容量,然后在应用过程中,当栈的空间不够使用时再逐段扩大。

## 2.3 成员与操作设计

**栈类（STACK）**

**为了使栈能够泛用，使用类模板T**

**私有成员：**

T\* \_top; //表示栈顶的上一个元素

T\* \_base; //表示栈底元素

int stackSize; //表示栈在内存中申请的大小

const int STACK\_INIT\_SIZE = 100; //规定栈初始化时申请的内存

**公有成员：**

Stack(); //构造函数

Status popStack(T&); //弹出栈，不会报错

T pop(); //弹出栈，会报错

Status push(T); //将T加入栈

Status get\_top(T&); //取栈首元素，不会报错

T top(); //取栈首元素，会报错

int size() //取栈的大小

bool empty() //判断栈是否为空

## 2.4 系统设计

系统首先调用printHead()函数实现对屏幕的初始化，使用initMap初始化Map，Map中储存从运算符号到优先级数组指引之间的对应关系（由于运算符号的ascii码都小于128，只需要使用大小为128的数组即可，查询复杂度为O(1)）。之后在循环中通过evaluate\_expression函数来计算表达式的值，并通过用户的输入控制何时退出循环。

# 3 实现

## 3.1 插入功能的实现

### 3.1.1 插入功能流程图

开始

申请空间是否足够

将值插入在top指针所指位置

top指针后移一位

申请新的空间

结束

True

False

### 3.1.2 插入功能核心代码

Status push(T e)

{

if (\_top - \_base >= stackSize)

{

//采取容量扩增的策略，用此方法扩容的平均复杂度为O(1)

\_base = (T\*)realloc(\_base, 2 \* (stackSize) \* sizeof(T));

if (!\_base)return OVERFLOW;

\_top = \_base + stackSize;

stackSize \*= 2;

}

//将e的值赋值给\_top所指，并且\_top指向下一个元素

\*\_top++ = e;

return OK;

}

## 3.2 去顶功能的实现

### 3.2.1 去顶功能流程图

开始

栈空？

\_top指针前移一位

删除失败

结束

True

True

### 3.2.2 去顶功能核心代码

//若栈为空不会报错

Status popStack(T& e)

{

if (\_base == \_top)return ERROR;

e = \*--\_top;

return OK;

}

//若栈为空会报错

T pop()

{

if (\_base == \_top) { cout << "删除失败，因为栈为空！\n"; exit(-1); }

return \*--\_top;

}

## 3.3 取顶功能的实现

### 3.3.1 取顶功能流程图

开始

栈空？

返回\_top前一位所指的元素

取顶失败

结束

True

True

### 3.3.2 查找功能核心代码

//不会报错

Status get\_top(T& e)

{

if (\_base == \_top)return ERROR;

e = \*(\_top - 1);

return OK;

}

//会报错

T top()

{

if (\_base == \_top) { cout << "取栈顶失败，因为栈为空！\n"; exit(-1); }

return \*(\_top - 1);

}

## 3.4 表达式求值功能的实现

### 3.4.1 表达式求值功能流程图

开始

当前读到的字符不是’=’或OPTR栈底不是’=’

OPTR储存运算符，OPND储存运算数

字符为数字

与栈顶元素优先级比较

插入OPND

删除栈顶，读取下一个字符

字符插入OPTR栈，读取下一个字符

去除OPND栈首两个元素，进行运算

读取第一个字符

弹出栈顶，为运算结果

结束

False

True

True

False

<

=

>

### 3.4.2 表达式求值核心代码

Status evaluate\_expression(char \*expression,ll& value)

{

Stack<ll>OPND; Stack<char>OPTR;//OPND储存运算数，OPTR储存运算符

OPTR.push('=');//先将=入栈

char ch = expression[0];//读取第一个字符

ll num ;

int i = 1;//指向当前读完的下一个字符

while (ch != '=' || OPTR.top() != '=')

{

if (isdigit(ch)) {

readNum(expression, num, i);

OPND.push(num);

ch = expression[i++];

// cout << "input " << num << endl;//测试语句

}

else//是运算符的情况

{

//判断是否是单目运算符，若是-，则改为!，若是+，可以直接忽略

if (i==1||isSingle(expression[i-2]))

{

if (ch == '+') { ch = expression[i++];continue;}

else if (ch == '-') { ch = '!'; }

}

int n1 = Map[int(OPTR.top())], n2 = Map[int(ch)];

//栈顶符号优先级低于新读入符号

if (priority[n1][n2] == '<') {

OPTR.push(ch);

// cout << "input " << ch << endl;//测试语句

ch = expression[i++];

}

//逻辑错误

else if (priority[n1][n2] == '\0') { return LOGIC\_ERROR; }

//栈顶符号优先级等于新读入符号

else if (priority[n1][n2] == '=') {

OPTR.pop();

ch = expression[i++];

// cout << "pop\n";

}

//栈顶符号优先级大于新读入符号

else

{

//单目运算符

if (OPTR.top() == '!')

{

OPTR.pop();

int a = OPND.pop();

OPND.push(operate(a, '!'));

// cout << "calculate -" << a << endl;//测试语句

}

//二目运算符

else

{

char op = OPTR.pop();

if (OPND.size() < 2) { return LOGIC\_ERROR; }

int b = OPND.pop();

int a = OPND.pop();

OPND.push(operate(a, op, b));

// cout << "calculate " << a << " " << op << " " << b << endl;//测试语句

}

}

}

}

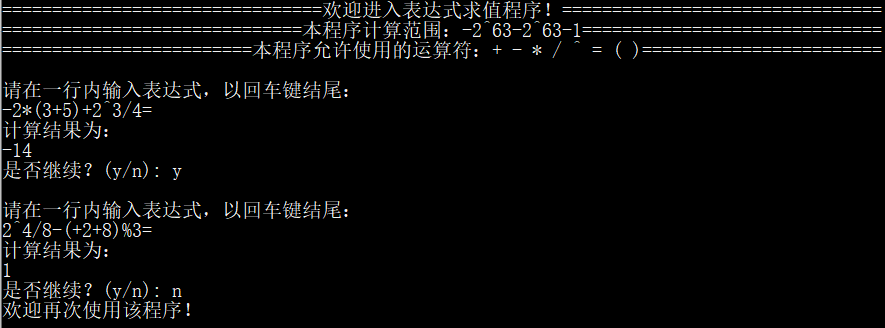
if (OPND.size() == 1)value = OPND.top();//返回栈顶元素

else return LOGIC\_ERROR;

return CORRECT;

}

### 3.4.3 表达式求值功能截屏示例



## 3.5 总体系统的实现

### 3.5.1 总体系统流程图

开始

初始屏幕

初始Map

读入表达式并计算

是否继续

结束

是

否

### 3.5.2 总体系统核心代码

printHead();

initMap();

ll x;

while (1)

{

printf("\n请在一行内输入表达式，以回车键结尾：\n");

//读取表达式兼正误检查

char\* expression = read\_expression\_without\_blank();//

//如果表达式中不含有非法字符

if (expression != NULL)

{

//如果表达式中含有逻辑错误

if (evaluate\_expression(expression, x) != CORRECT)

{

printf("逻辑错误！\n");

}

else cout << "计算结果为：\n" << x << endl;

delete[]expression;

}

printf("是否继续？(y/n): ");

int flag;

//严格的y/n审查部分，用getline读取一整行，这一行里面只能由空格和一个y或n组成，否则报错

while (1)

{

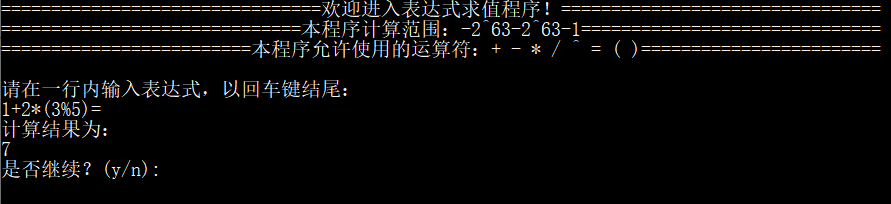
...

}

if (flag == 2)break;

}

### 3.5.3 总体系统截屏示例



# 4 测试

## 4.1 基本测试

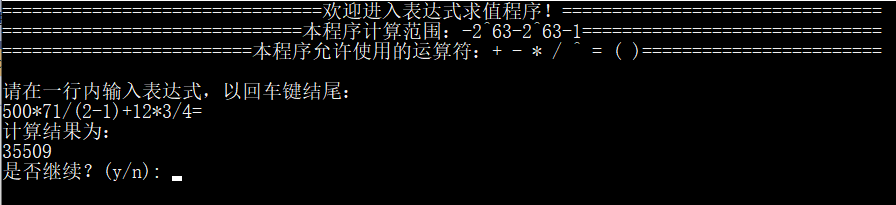
### 4.1.1 加减乘除功能测试

**测试用例**：500\*71/(2-1)+12\*3/4=

**预期结果**：

35509

**实验结果**



### 4.1.2 取余以及乘方功能测试

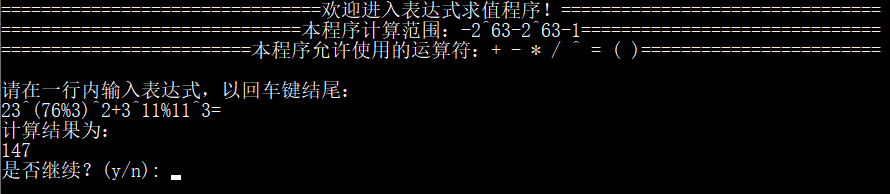
**测试用例：**23^(76%3)^2+3^11%11^3=

**预期结果：**

147

(说明：根据数学中的规定，乘方运算从高往低结合，所以前者为)

**实验结果：**



### 4.1.3 括号功能测试

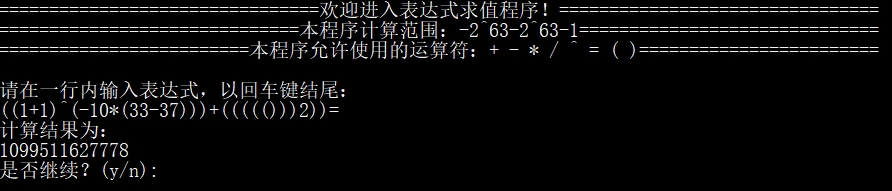
**测试用例：**((1+1)^(-10\*(33-37)))+((((()))2))=

**预期结果：**

1099511627778

(解释：2^40+2)

**实验结果：**



### 4.1.4 单目运算符功能测试

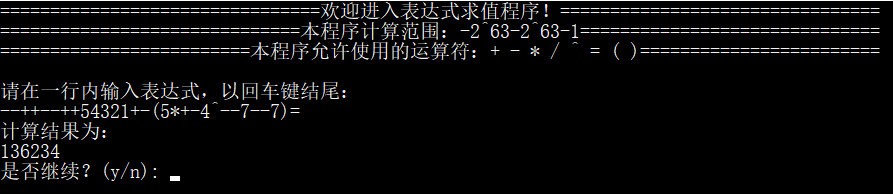
**测试用例：**--++--++54321+-(5\*+-4^--7--7)=

**预期结果：**

136234

(解释：相当于 54321-(5\*((-4)^7)+7)=136234)

**实验结果：**



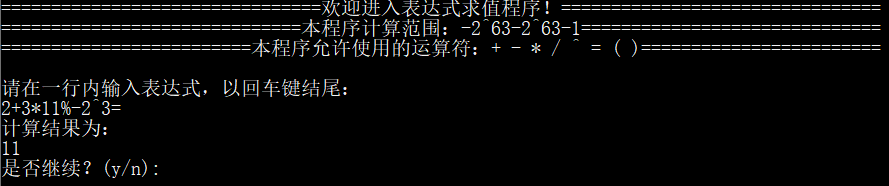
### 4.1.5 运算优先级测试

**测试用例：**2+3\*11%-2^3=

**预期结果：**

11

**实验结果：**



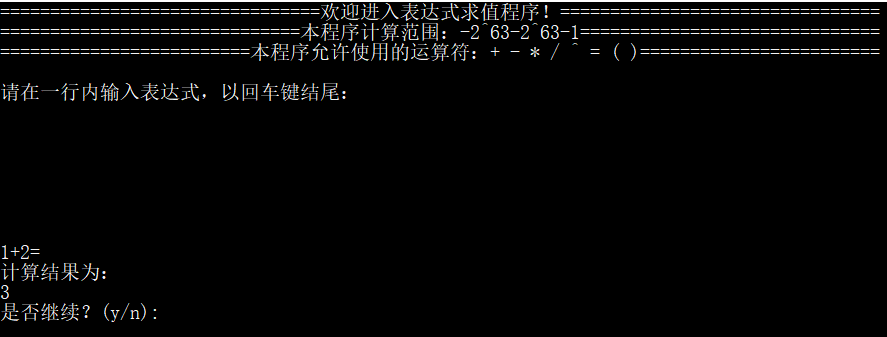
## 4.2 边界测试

### 4.2.1 初始化无输入数据

**测试用例：**初始无输入数据

**预期结果：**直到用户输入为止。

**实验结果：**

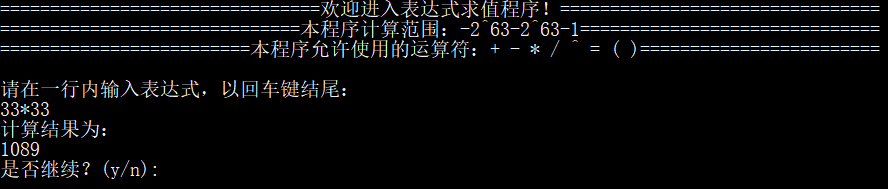


### 4.2.2 用户不输入等于号

**测试用例：**不输入等于号

**预期结果：**自动在末尾添加等于号，程序正常运行。

**实验结果：**

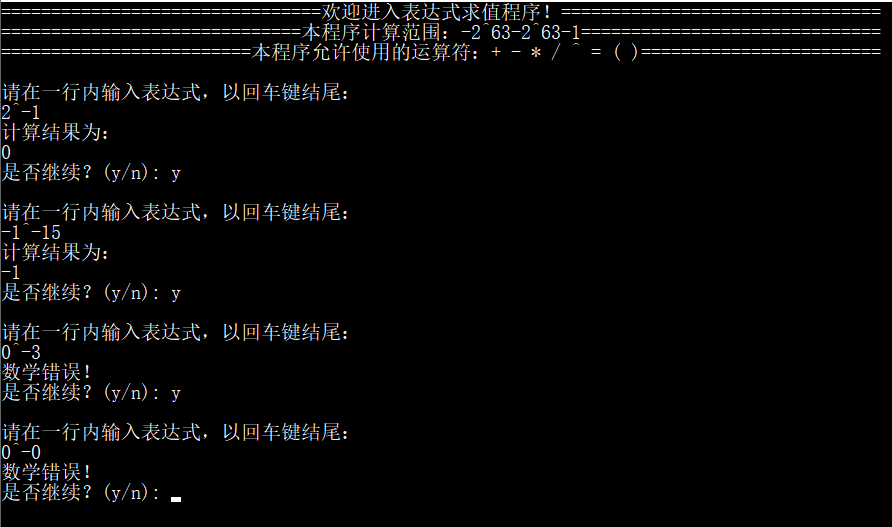


### 4.2.3 用户输入a^-b

**测试用例：**2^-1 -1^-15 0^-3 0^-0

**预期结果：** 0 -1 报错 报错

**实验结果：**



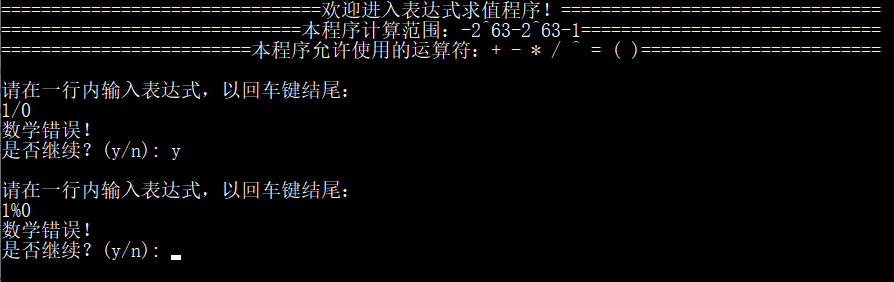
## 4.3 出错测试

### 4.3.1 除以0

**测试用例：**1/0 1%0

**预期结果：**报错

**实验结果：**

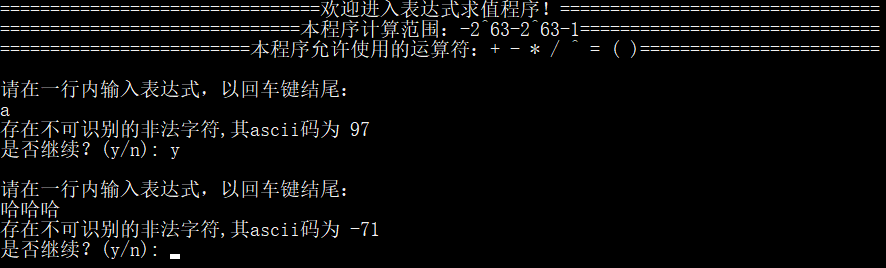
****

### 4.3.2 输入非法字符

**测试用例：**a 哈哈哈

**预期结果：**报错

**实验结果：**

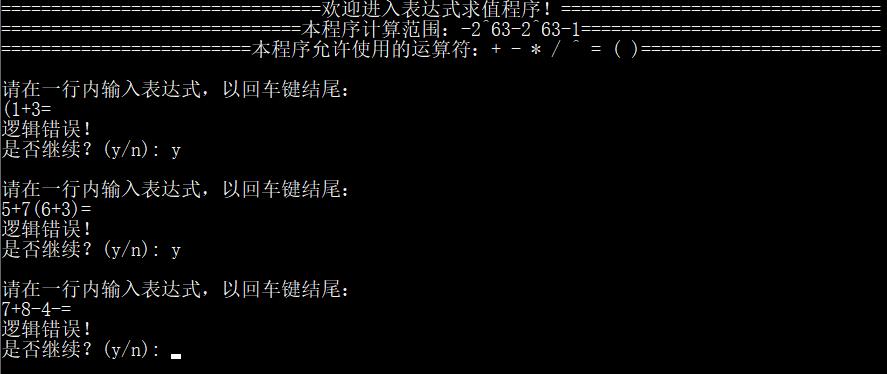
****

### 4.3.3 逻辑错误

**测试用例：**(1+3= 5+7(6+3)= 7+8-4-=

**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：**

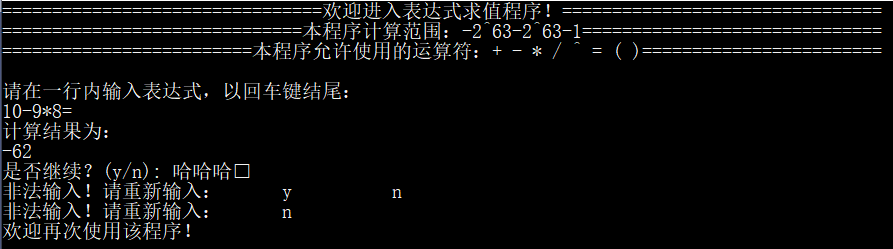


### 4.3.4 y/n输入错误

**测试用例：**乱码输入 不干净输入

**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：**

****