项目说明文档

数据结构课程设计

——表达式计算

作 者 姓 名： 陈君

学 号： 2250420

指 导 教 师： 张颖

学院、 专业： 软件学院 软件工程

同济大学

Tongji Universit

# 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最基本问题，就是将一个表达式转化为逆波兰表达式并求值。具体要求是以字符序列的形式从终端输入语法正确的，不含变量的整数表达式，并利用给定的优先关系实现对算术四则混合表达式的求值，并延时在求值过程中运算符栈，操作数栈，输入字符和主要操作变化过程。

要把一个表达式翻译成正确求值的一个机器指令序列，或者直接对表达式求值，首先要能正确解释表达式。任何一个表达式都是由操作符，运算符和界限符组成，我们称它们为单词。一般来说，操作数既可以是常数，又可以是被说明为变量或常量的标识符；运算符可以分成算术运算符，关系运算符和逻辑运算符3类；基本界限符有左右括号和表达式结束符等。为了叙述的简洁，我们仅仅讨论简单算术表达式的求值问题。这种表达式只包括加，减，乘，除4种运算符。

人民在书写表达式时通常采用的是“中缀”表达形式，也就是将运算符放在两个操作数中间，用这种“中缀”形式表示的表达式称为中缀表达式。但是，这种表达式表示形式对计算机处理来说是不大合适的。对于表达式的表示还有另一种形式，称之为“后缀表达式“，也就是将运算符紧跟在两个操作书的后面。这种表达式比较合适计算机的处理方式，因此要用计算机来处理，计算表达式的问题，首先要将中缀表达式转化成后缀表达式，又称为逆波兰表达式。

## 1.2 功能分析

为了实现表达式求值，本项目要求首先读入表达式（包括括号）并创建对应二叉树，其次对二叉树进行前序遍历，中序遍历，后续遍历，输出对应的逆波兰式，中序表达式和波兰表达式。

# 2 设计

## 2.1 数据结构设计

如上述要求所示，表达式的具体计算以及逆波兰式的存储需要用到两个堆栈结构，所以我定义了一个Mystack类用于栈的操作，同时设计了两个结构体，用于表示运算符以及二叉树的结点，同时为了存储逆波兰式用到了树的结构

## 2.2 类结构设计

设计了Mystack类用于栈的操作，设计了mytree用于二叉树的操作。

## 2.3 成员与操作设计

struct Op

{

char ch[10];

int piorty = -1;

};

struct Node

{

Node\* Lnode = NULL;

Node\* Rnode = NULL;

Op \* op;

};

class Mystack

{

public:

bool pop();

bool push(Node \* op);

Node\* gettop() { return top==-1 ? NULL:ops[top]; }

void putout() {

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

cout << ops[i]->op->ch;

cout << "#"<<endl;

}

private:

Node \* ops[20];

int top = -1;

};

class Mytree

{

public:

void init();

void construct();

void getop(Op\* & op,int &i);

void preorder(Node\* root);

void lastorder(Node\* root);

void midorder(Node\* root);

Node\* getroot() { return root; }

private:

Node\* root;

char exp[100];

## };2.4 系统设计

主程序先调用init()函数，提示用户输入表达式，然后调用mytree中的construct函数将表达式用二叉树存储，最后调用三个函数分别使用前序中序后序遍历二叉树，输出波兰表达式，中缀表达式，逆波兰表达式。

# 3 实现

### 3.1 bool Mystack::pop()//

具体实现：判断栈是否为空： 首先，通过检查栈顶指针 top 是否为 -1，来确定栈是否为空。如果栈为空，表示没有元素可以出栈，直接返回 false。

出栈操作： 如果栈不为空，将栈顶元素置为 NULL（或者释放相应的内存，这里似乎是为了简化），然后将栈顶指针 top 减一，表示栈顶元素已经被出栈。

返回结果： 出栈操作成功完成后，返回 true，表示成功出栈一个元素。

## 3.2bool Mystack::push(Node \*op)// 这是一个栈的入栈操作的实现

## 具体思路如下：

判断栈是否已满： 首先，通过检查栈顶指针 top 是否等于 100（数组大小减一），来确定栈是否已满。如果栈已满，表示没有空间可以再入栈，直接返回 false。

入栈操作： 如果栈未满，将待入栈的元素 op 放入数组 ops 中的下一个位置，然后将栈顶指针 top 加一，表示栈顶元素已经被入栈。

返回结果： 入栈操作成功完成后，返回 true，表示成功入栈一个元素。

## 3.3 void Mytree::getop(Op\*& op,int &i)//获取运算符

具体实现：1、判断字符类型： 首先，检查当前字符 exp[i] 是否为操作符。如果不是数字（即小于 '0' 或大于 '9'），表示是操作符。

2、处理操作符： 如果是操作符，将当前字符复制到 Op 对象的 ch 数组中，并根据具体的操作符设置相应的优先级 piorty。

如果是加法或减法（+ 或 -），设置优先级为 5。

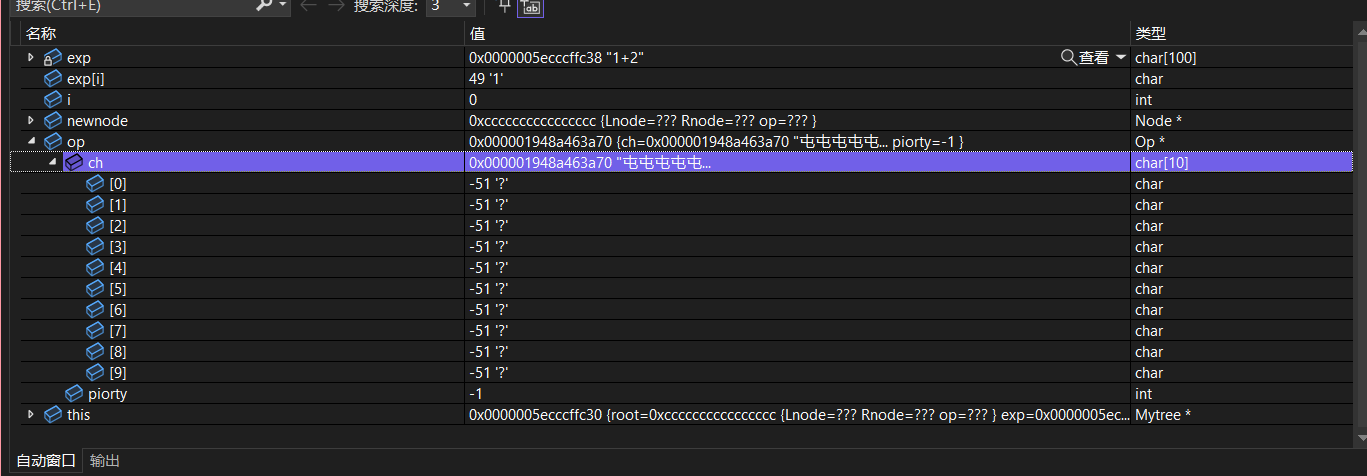
如果是乘法或除法（\* 或 /），设置优先级为 6。

如果是左括号 (，设置优先级为 15。

如果是右括号 )，设置优先级为 0。

3、处理非操作符（数字）： 如果当前字符是数字，说明是操作数，将当前字符复制到 Op 对象的 ch 数组中，同时将优先级 piorty 设置为 -1，表示非操作符。

## 3.3.1 void Mytree::getop(Op\*& op,int &i)//截图展示



## 3.4 void Mytree::construct()//构造二叉树

具体实现：初始化两个栈： 创建两个栈 stack1 和 stack2，其中 stack1 用于存放操作数和最终的树结构，stack2 用于存放操作符。

循环处理中缀表达式： 从表达式的第一个字符开始，进入循环，直到字符串结束（exp[i] != '\0'）。

获取操作符或操作数： 使用 getop 函数获取当前字符的操作符信息或操作数信息，并将其构建成一个 Op 对象。

处理操作数： 如果 Op 对象的 piorty 为 -1，表示是操作数，直接创建一个新节点 newnode，将 Op 对象赋给该节点的 op 成员，并将新节点压入 stack1。

处理操作符： 如果 Op 对象的 piorty 不为 -1，表示是操作符。分三种情况处理：

如果是左括号（(），设置 piorty 为 0，创建新节点并将其压入 stack2。

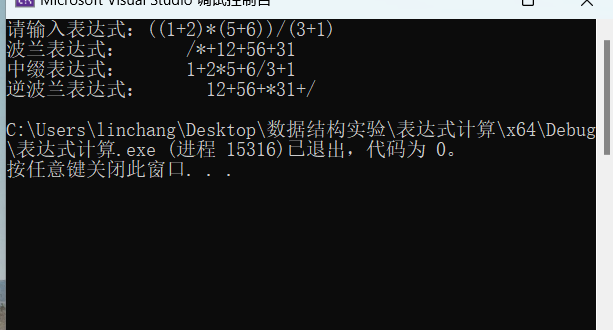
如果是右括号（))，需要将 stack2 中的操作符弹出，并将相应的操作数和操作符构建为新节点，然后将新节点压入 stack1，直到遇到左括号。

对于其他操作符，清空 stack2 中比当前操作符优先级高的操作符，将相应的操作数和操作符构建为新节点，然后将新节点压入 stack1。最后，将当前操作符压入 stack2。

清空栈： 处理完整个中缀表达式后，需要清空 stack2 中的操作符，将相应的操作数和操作符构建为新节点，然后将新节点压入 stack1。此时，stack1 中只会剩下一个节点，即根节点。

设置根节点： 将 stack1 的栈顶节点设为根节点 root。

## 3.4 void Mytree::construct()的截图展示（）



## 3.5void Mytree::preorder(Node\* root)

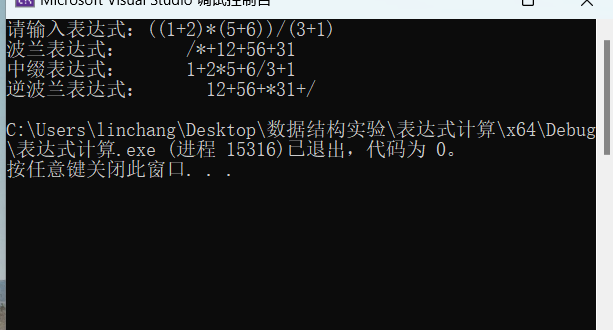
具体实现：判空处理： 如果当前节点 root 为 NULL，直接返回，不进行操作。

输出当前节点： 输出当前节点的操作符或操作数的字符串表示（root->op->ch）。

递归遍历左子树： 调用 preorder 函数，递归地对当前节点的左子树进行前序遍历。

递归遍历右子树： 调用 preorder 函数，递归地对当前节点的右子树进行前序遍历。

截图展示：



## 3.5void Mytree::midorder(Node\* root)

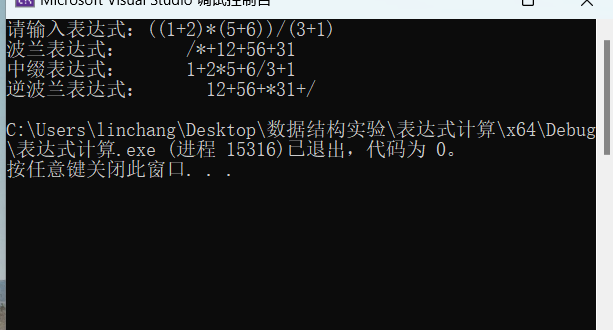
具体实现：判空处理： 如果当前节点 root 为 NULL，直接返回，不进行操作。

递归遍历左子树： 调用 preorder 函数，递归地对当前节点的左子树进行前序遍历。

输出当前节点： 输出当前节点的操作符或操作数的字符串表示（root->op->ch）。

递归遍历右子树： 调用 preorder 函数，递归地对当前节点的右子树进行前序遍历。

截图展示：



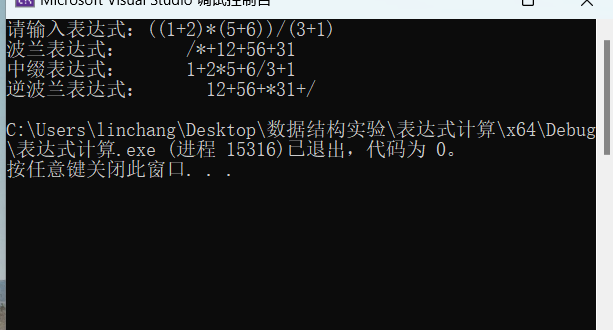
## 3.5void Mytree::lastorder(Node\* root)

具体实现：判空处理： 如果当前节点 root 为 NULL，直接返回，不进行操作。

递归遍历左子树： 调用 lastorder 函数，递归地对当前节点的左子树进行后序遍历。

递归遍历右子树： 调用 lastorder 函数，递归地对当前节点的右子树进行后序遍历。

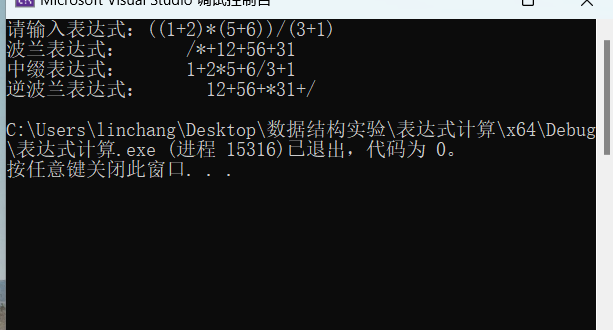
输出当前节点： 输出当前节点的操作符或操作数的字符串表示（root->op->ch）。截图展示：



## 3.6 总体系统的实现

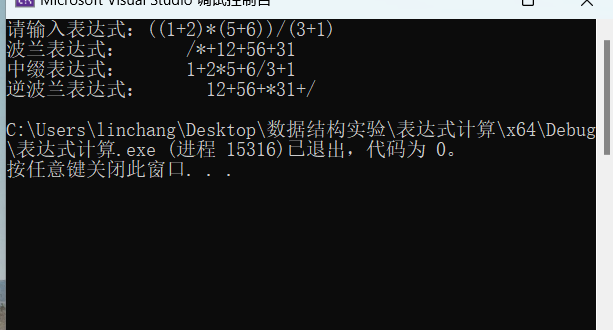
主程序先调用init()函数，提示用户输入表达式，然后调用mytree中的construct函数将表达式用二叉树存储，最后调用三个函数分别使用前序中序后序遍历二叉树，输出波兰表达式，中缀表达式，逆波兰表达式。

### }3.6.1 总体系统截屏示例



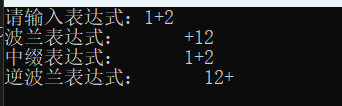
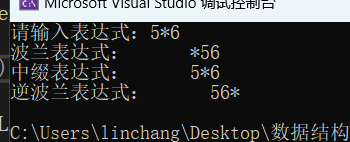
# 4 测试

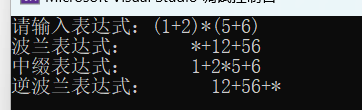
## 4.1 系统总体功能测试功能测试



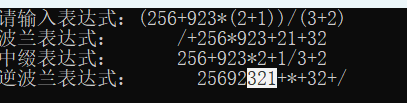
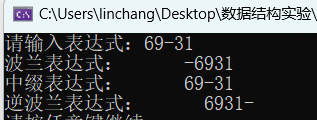
4.2基础功能的测试

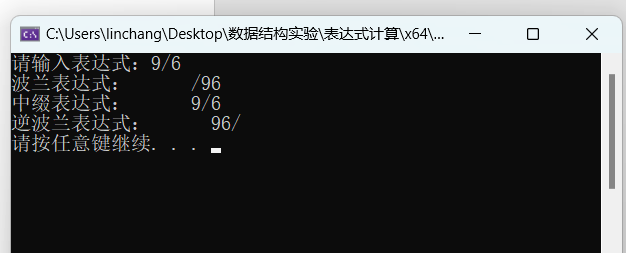
加法：

乘法：

简单的括号使用：(1+2)\*(5+6)

复杂计算(256+923\*(2+1))/(3+2)

减法：

除法：

复杂计算：(256+923\*(2+1))/((6+3+2)\*5+6+7)

