表达式计算

1452764　何冬怡

目录

项目简介

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。

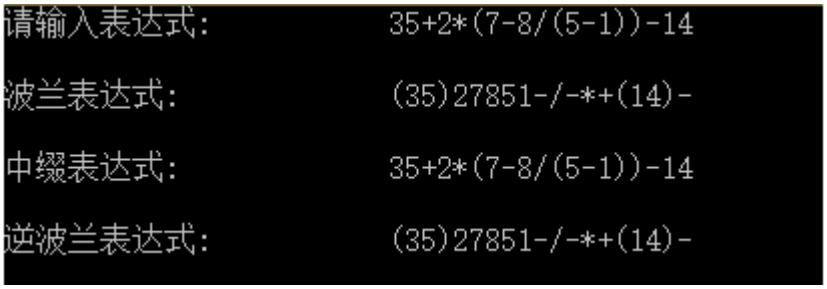
人民在书写表达式时通常采用的是“中缀”表达形式，也就是将运算符放在两个操作数中间，用这种“中缀”形式表示的表达式称为中缀表达式。但是，这种表达式表示形式对计算机处理来说是不大合适的。对于表达式的表示还有另一种形式，称之为“后缀表达式“，也就是将运算符紧跟在两个操作书的后面。这种表达式比较合适计算机的处理方式，因此要用计算机来处理，计算表达式的问题，首先要将中缀表达式转化成后缀表达式，又称为逆波兰表达式。

项目要求

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

使用手册

1. 打开 exe 文件后，出现用户界面，输入目标表达式，enter 输入即输出对应表达式。
2. 注意：本系统支持自然数的加减乘除（+−*/）以及多层括号运算。波兰表达式和逆波兰表达式输出时会将高于一位的数字用括号括起。



程序概述

1. 数据结构

利用二叉树储存表达式。节点类 Node 储存数值（负数为宏定义的运算符），左右结点为左右操作数。二叉树类 ExpTree 成员变量为二叉树根节点，为 Node 的友元类。

```
#define PLUS    -1  //+
#define MINUS   -2  //-
#define MULTI   -3  //*
#define DIVI    -4  //÷
```

2. 算法思路

用字符串储存表达式，使用迭代器遍历字符串完成解析。然后使用三种遍历方式输出。

1) 表达式二叉树的创建

考虑到算数优先级，算数优先级越高的应该离根节点越远，故需要从右向左遍历。在字符串前面加入一个占位字符‘a’,将迭代器的起点和终点分别设置为 end()−1 和 begin()，利用迭代器自减完成遍历。第一轮遍历找到第一个（最右边的）+−运算，若无则再次遍历寻找*÷运算，然后将字符串左右两侧再递归分别解析。遍历过程中需要跳过括号，而当遍历内容恰好被括号完全括起则需要将其删除。若解析内容无运算符，则可以肯定为自然数，将其转化为数字保存。

递归参数为迭代器的起点和终点。

2) 表达式二叉树的中序输出

使用递归输出。需要对下一层的运算符判断，考虑是否加括号。包括：减法右侧的减法；除法右侧除法；乘除法左侧和右侧的加减法。

3) 表达式二叉树的前序输出和后序输出

这两种输出方式比较接近，而且由于运算符按顺序输出，所以不需要使用括号标注运算优先级。但由于操作数之间没有间隔，所以对于高于一位的数字需要加括号输出。

3. 文件目录

可执行文件7_1452764_hedongyi.exe	类定义声明7_1452764_hedongyi.h
主文件 7_1452764_hedongyi.cpp	项目文档 7_1452764_hedongyi.pdf

成员变量/函数接口/宏

成员变量名	数据类型	功能说明	
_left	BSTNode*	左节点指针	
_right	BSTNode*	右节点指针	
_data	int	操作数（运算符）值	
成员函数名	**功能**	**参数**	**返回值**
Node(int)	构造对应关键码的节点	int 关键码	Node

表 1　Node 类接口

宏定义
`#define PLUS  -1     //+`
`#define MINUS -2     //-`
`#define MULTI -3     //*`
`#define DIVI  -4     //÷`
`#define L cout<<'('`
`#define R cout<<')'`
`typedef decltype(aa.begin()) ITER;`

表 2 宏定义

成员变量名	数据类型	功能说明	
_root	Node*	二叉树根节点指针	
成员函数名	**功能**	**参数**	**返回值**
ExpTree(Node* p)	以 p 为根节点构造树	Node* 目标根节点	ExpTree
setroot(Node* root)	设置根节点	Node* 目标根节点	void
parse(ITER begin, ITER end)	遍历 begin 到 end，解析建立二叉树	ITER begin 开始 ITER end　终点	Node*
skip(ITER &it)	跳过空格	ITER 当前迭代器	void
convertToInt(ITER begin, ITER end)	将范围内转化为 int	ITER begin 开始 ITER end　终点	int
printDLR(Node *root)	前序递归输出	Node* 目标二叉树根节点	void
printLDR(Node *root)	中序递归输出	Node* 目标二叉树根节点	void
printLRD(Node *root)	后序递归输出	Node* 目标二叉树根节点	void
printDLR()	启动全树前序输出	空	void
printLDR()	启动全树中序输出	空	void
printLRD()	启动全树后序输出	空	void

表 3　ExpTree 类接口