**项目说明文档**

**数据结构课程设计**

**——表达式转换**

### 作 者 姓 名 尹诚成

学 号 2351279

指 导 教 师 张 颖

学 院 专 业 计算机科学与技术学院 软件工程



二〇二三 年 十二 月 十三 日

**目录**

[1.项目分析 1](#_bookmark0)

* 1. .[项目背景分析 1](#_bookmark1)
  2. .[项目需求分析 1](#_bookmark2)
  3. .[项目功能分析 1](#_bookmark3)

1.3.1.构建表达式二叉树功能 1

1.3.2.表达式转换与计算功能 1

1.3.3.异常处理功能 2

[2.项目设计 2](#_bookmark12)

* 1. .[数据结构设计 2](#_bookmark13)
  2. .[结构体与类设计 2](#_bookmark14)

2.2.1.Expression 联合体的设计 2

2.2.2 LinkNode 结构体的设计 2

2.2.3 Stack 类的设计 2

2.2.4 LinkedQueue类的设计 3

2.2.5 BinTreeNode 结构体的设计 3

2.2.6 BinaryTree 类的设计 4

2.2.7 ExpressionTree类的设计 5

* 1. .[项目主体架构设计 5](#_bookmark32)

[3.项目功能实现 6](#_bookmark33)

3.1.项目主体架构的实现 6

3.2.构建表达式二叉树功能的实现 6

3.3.表达式计算功能的实现 7

3.4.表达式输入非法的处理功能的实现 7

4.项目测试 8

4.1.表达式输入合法性验证功能测试 8

4.2.构建表达式二叉树和表达式计算功能测试 8

1. **项目分析**
   1. 项目背景分析

表达式求值是程序设计语言编译的基本问题之一，主要涉及将表达式转化为后缀表达式并进行求值。这一过程要求用户以字符序列的形式输入一个语法正确的整数表达式，不包含任何变量。在求值过程中，表达式的组成元素包括：操作数、算术运算符、关系运算符、逻辑运算符，以及基本界限符如括号和表达式结束符。人们通常采用中缀表达形式编写表达式，即运算符置于两个操作数之间。然而，这种表达形式并不适合计算机处理。相反，后缀表达式，即运算符紧随其操作数之后的形式，更适合计算机处理。因此，将中缀表达式转换成后缀表达式是计算机处理表达式求值问题的关键步骤。

* 1. 项目需求分析

基于以上背景分析，本项目需要实现需求如下：

(1)用户应能通过命令行界面输入一个整数算术表达式。该表达式不包含变量，仅包括整数、算术运算符以及括号；

(2)设计简单直观的控制台界面。

(3)读入表达式并进行解析和验证，程序需要能够解析输入的中缀表达式，并准确识别其中的数字、运算符和括号。还应有能力验证输入表达式的合法性，例如检查括号是否匹配，运算符是否正确使用等；

(4)将中缀表达式转换为后缀表达式并创建表达式二叉树，对表达式二叉树进行前序遍历，中序遍历和后序遍历，分别输出对应的前缀表达式，中缀表达式和后缀表达式；

(5)计算表达式的值并进行输出；

(6)实现异常处理机制，确保系统稳定性和安全性，避免因用户输入错误导致系统崩溃或信息丢失。

* 1. 项目功能分析

本项目旨在通过模拟计算机处理和计算表达式的过程，实现表达式二叉树的构建，并通过前序遍历，中序遍历和后序遍历，分别输出对应的前缀表达式，中缀表达式和后缀表达式，最后进行表达式的计算。下面对项目的功能进行详细分析。

* + 1. 构建表达式二叉树功能

该功能的目的是将用户输入的中缀表达式转换成二叉树形式，以便于执行后 续的遍历和计算。通过二叉树的形式，可以有效地表示和处理表达式中的运算符 优先级和括号结构。

* + 1. 表达式转换与计算

此功能旨在通过遍历表达式二叉树，将中序表达式转化为前缀表达式或后缀表达式，并计算表达式的值。

* + 1. 异常处理功能

实现异常处理机制，处理用户可能输入的非法信息，确保系统的稳定性和安全性。

1. **项目设计**

2.1.数据结构设计

基于项目分析，选择使用二叉树作为数据结构，主要基于以下几个考虑：

(1)表达式的层级结构：二叉树具有天然的层级结构，非常适合表示数学表达式中的操作符和操作数之间的关系。每个节点可以代表一个操作符或操作数，其子节点代表与之相关的表达式部分；

(2)便于表达式的遍历和计算：二叉树结构便于实现表达式的前序、中序和后序遍历，这对于不同类型的表达式输出和求值至关重要；

(3)灵活性和扩展性：使用二叉树结构，可以轻松添加或修改表达式的组成部分；

(4)易于理解和实现：二叉树作为一种基础的数据结构，其原理和操作相对简单明了，便于程序设计和调试；

2.2.结构体与类设计

2.2.1.Expression联合体的设计

Expression联合体是一个用于表示表达式中单个元素的联合体，用于表示单个元素的值或表示的运算符，其数据成员定义及含义如下：

char type：运算符类型

double value：运算数的值

2.2.2.LinkNode结构体的设计

LinkNode 结构体是一个用于构建链表节点的模板结构体。该结构体用于表示链表中的每个节点，其中包括节点存储的数据以及指向下一个节点的指针。本项目希望链表结点类可以直接访问链表结点，所以使用 struct 而不是 class 描述链表结点类。其数据成员和构造函数定义及含义如下：

T data：数据域，存储节点的数据。

LinkNode<T>\* link：指针域，指向下一个节点的指针。

LinkNode(LinkNode\* ptr = NULL)：构造函数，初始化指针域。

LinkNode(const T& item, LinkNode\* ptr = NULL)：构造函数，初始化数据域和指针域。

2.2.3.Stack类的设计

该通用模板类MyStack 用于创建和操作堆栈数据结构。堆栈是一种基本的数据结构，它遵循先进后出的原则，允许用户推入和弹出元素，元素（即链表节点）由MyLinkNode结构体表示，其中包含数据和指针域。此模板类MyStack实现了堆栈的基本功能，并提供了一些辅助方法以方便操作。其数据成员、构造函数、析构函数、公有成员函数，运算符重载定义及含义如下：

LinkNode<T>\* top：指向栈顶元素的指针

Stack() : 默认构造函数，创造一个空的栈。

~Stack() ：析构函数，清空栈并释放内存。

void push(T item)：用于将新元素推入堆栈。它会在堆栈的顶部创建一个新节点，将元素存储在 其中，并将新节点的link指向旧的堆栈顶部。

bool pop(T& item)：从堆栈中弹出顶部元素，将其存储在传入的参数中，并释放相应的节点。如 果堆栈为空，它将返回false，否则返回true。

bool getTop(T& item) const：用于获取堆栈的顶部元素，但不弹出它。如果堆栈为空，它将返回false， 否则返回true。

bool isEmpty() const：判断堆栈是否为空，如果堆栈为空，它将返回true，否则返回false。

int getSize() const：计算堆栈中的元素数量，通过遍历堆栈节点并计数节点的数量来实现。

void makeEmpty()：清空堆栈，释放所有分配的内存。它通过遍历堆栈的节点，释放它们，并将 堆栈的top指针置为NULL来实现。

2.2.4.LinkedQueue 类的设计

该通用模板类LinkedQueue用于存储和管理数据元素，遵循先进先出的原则。该队列是基于链表数据结构实现的，链表节点由MyLinkNode 结构体表示，其中包含数据和指针域。此模板类LinkedQueue允许在队列的前端添加元素，以及从队列的前端移除元素。其数据成员、构造函数、析构函数、公有成员函数，运算符重载定义及含义如下：

LinkNode<T>\* front：指针指向队列的前端，即队列中的第一个元素

LinkNode<T>\* rear：指针指向队列的末端，即队列中的最后一个元素

LinkedQueue()：默认构造函数，创建一个空的队列

~LinkedQueue()：析构函数，清空队列并释放内存。

bool EnQueue(const T item)：将元素添加到队列的末尾

bool DeQueue(T& item)：移除队列的前端元素并将其值通过引用返回

bool getFront(T& item) const：获取队列前端元素的值

void makeEmpty()：清空队列，释放所有元素占用的内存

bool IsEmpty() const：检查队列是否为空

int getSize() const：获取队列中元素的数量

2.2.5.BinTreeNode 结构体的设计

BinTreeNode 结构体是一个模板结构体，用于构建和表示一个二叉树的节点。二叉树是一种重要的数据结构，广泛应用于各种计算机科学领域。在这个结构体中，每个节点包含一个数据元素和两个指向其子节点的指针：左子节点和右子节点。其数据成员，构造函数定义及含义如下：

T data：数据域，存储节点的数据

BinTreeNode<T>\* leftChild：指针域，指向左子节点的指针

BinTreeNode<T>\* rightChild：指针域，指向右子节点的指针

BinTreeNode() : data(T()),leftChild(NULL), rightChild(NULL) {}：构造函数。

BinTreeNode(T item, BinTreeNode<T>\* l = NULL, BinTreeNode<T>\* r = NULL) : data(item), leftChild(l), rightChild(r) {}：构造函数，初始化数据域和指针域。

2.2.6.BinaryTree 类的设计

BinaryTree 类是一个模板类，用于创建和管理二叉树结构。二叉树是一种基础且重要的数据结构，广泛应用于多种场景，如管理、排序和搜索算法等。本模板类提供了二叉树的基本操作，包括插入、查找、遍历、修改和删除节点等功能。其数据成员、构造函数、析构函数、公有成员函数，运算符重载定义及含义如下：

BinTreeNode<T>\* root：指向树的根节点的指针

BinTreeNode<T>\* copy(const BinTreeNode<T>\* subTree)：深拷贝一个子树。

BinaryTree(): root(NULL) {}：默认构造函数，初始化一个空的二叉树，即根节点设置为NULL。

BinaryTree(T& item)：转换构造函数，创建一个二叉树，并设置根节点的数据为提供的 item，适 用于已知根节点数据的情况，便于直接构建含有根节点的二叉树。

BinaryTree(BinaryTree<T>& other)：复制构造函数，创建一个新的二叉树，其结构和数据是另一个二叉树（other） 的深拷贝。

~BinaryTree()：析构函数，用于在 BinaryTree 类的对象不再需要时，安全地销毁该对 象。它负责释放二叉树中所有节点占用的内存资源，防止内存泄漏。

void destroy(BinTreeNode<T>\*& subTree)：递归地销毁整个子树。

bool isEmpty(void)：检查树是否为空。

int getHeight(BinTreeNode<T>\* subTree)：计算树或子树的高度。

int getSize(BinTreeNode<T>\* subTree)：计算树或子树的高度。

BinTreeNode<T>\* getRoot(void)：获取树的根节点。

BinTreeNode<T>\* getParent(BinTreeNode<T>\* current, BinTreeNode<T>\* subTree)：查找指定节点的父节点。

BinTreeNode<T>\* getLeftChild(BinTreeNode<T>\* current)：查找指定节点的左子节点。

BinTreeNode<T>\* getRightChild(BinTreeNode<T>\* current)：查找指定节点的右子节点。

BinTreeNode<T>\* findNode(const T& item, BinTreeNode<T>\* subTree)：在树中查找包含特定数据的节点。

void inOrder(BinTreeNode<T>\* subTree)：前序遍历树。

void preOrder(BinTreeNode<T>\* subTree, bool lastNode)：中序遍历树。

void postOrder(BinTreeNode<T>\* subTree)：后序遍历树。

void levelOrder(BinTreeNode<T>\* subTree)：层序遍历树。

bool insertFrontLeft(BinTreeNode<T>\*& subTree, const T& item)：在指定节点插入左子节点。

bool insertFrontRight(BinTreeNode<T>\*& subTree, const T& item)：在指定节点插入右子节点。

void inOrderOutput(BinTreeNode<T>\* subTree)：前序遍历树输出。

void preOrderOutput(BinTreeNode<T>\* subTree)：中序遍历树输出。

void postOrderOutput(BinTreeNode<T>\* subTree)：后序遍历树输出。

void levelOrederOutput(BinTreeNode<T>\* subTree)：层序遍历树输出。

BinaryTree<T>& operator=(const BinaryTree<T>& other)：重载赋值运算符，用于实现树的深拷贝。

2.2.7.ExpressionTree 类的设计

ExpressionTree 类是一个为算术表达式处理和计算设计的类，它继承自 MyBinaryTree<Expression>类。该类的主要功能是将一个算术表达式转换成一个二叉树结构，并提供计算该表达式的结果的方法。其数据成员、构造函数、析构函数、公有成员函数，运算符重载定义及含义如下：

Stack<char> operatorStack：用于存储操作符的栈

Stack<BinTreeNode< Expression>\*> operandStack: 用于构建表达式树的节点栈。

int setPriority: 用于确定不同操作符的优先级。

double calculateExpression (BinTreeNode<Expression>\* node): 递归地计算表达式树中每个节点的值。

void processOperatorStack():取一个操作符和两个操作树来构建二叉树节点。

void createExpressionTree(union Expression\* expressionStruct): 将表达式转换为一个二叉树结构，每个节点代表一个操作数或操作符。

double calculate():对已构建的表达式树进行计算，返回表达式的数值结果。

2.3.项目主体架构设计

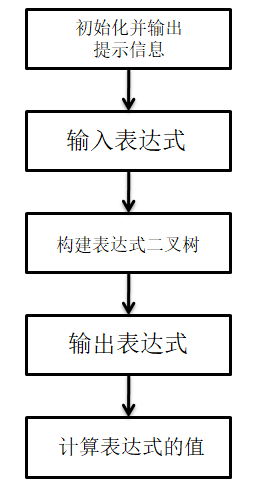


图 2.3 项目主体架构设计流程图

项目主体架构设计为：

(1)初始化并输出提示信息；

(2)输入表达式并构建表达式二叉树；

(3)输出不同形式的表达式；

(4)计算表达式。

**3.项目功能实现**

3.1.项目主体架构的实现

项目主体架构实现思路为：

(1)初始化和提示信息：程序开始时，首先通过outputStart函数展示用户界面和输入要求，包括表达式的字符限制、有效字符种类、括号匹配要求等。这 一步骤为用户提供了清晰的指导，确保输入的表达式符合程序处理的格式；

(2)输入表达式：用户被提示输入一个表达式。这个表达式被存储在一个字符数组expression中，其长度限制为EXPRESSION\_MAX\_CHAR。循环检查输入的表达式是否有效（通过checkExpressionIsVaild函数）。这包括检查表达式的格式是否正确、 是否存在非法字符等；

(3)构建表达式二叉树：一旦输入有效，程序使用ExpressionTree类的实例来创建一个表达式树。这是通过调用createExpressionTree 方法实现的，该方法将输入的表达式转换为二叉树的形式。在这个阶段，表达式中的每个数字和操作符都被作为树的节点处理，构建出反映表达式结构的二叉树；

(4)输出不同形式的表达式：程序输出原始的中缀表达式、 转换后的前缀表达式和后缀表达式。这些输出是通过对表达 树进行前序、中序和后序遍历得到的，展示了表达式在不同格式下的表示；

(5)计算表达式的值：程序通过调用 ExpressionTree 类的 calculate 方法计算表达式的数值结果。该方法内部使用递归遍历表达式树，根据树节点的操作符和操作数进行计算；

(6)等待用户退出：最后，程序等待用户按下回车键退出。

3.2.构建表达式二叉树功能的实现

构建表达式二叉树功能的函数为 ExpressionTree 类的成员函数 createExpressionTree，构建表达式二叉树功能实现的思路为：

(1)初始化后缀表达式数组和索引：创建一个字符数组来存储后缀表达式，并设置一个索引变量用于跟踪数组的当前位置；

(2)遍历表达式：对输入的中缀表达式中的每个字符进行遍历；

①处理操作数：如果当前字符是操作数，则直接将其添加 到后缀表达式数组中；

②处理左括号：遇到左括号“(”时，将其推入栈中；

③处理右括号：遇到右括号“)”时，调用processOperatorStack函数创建一个二叉树节点，直到遇到左括号“(”。然后弹出左括号；

④处理运算符：对于运算符，当栈顶运算符的优先级大于或等于当前运算符时，弹调用processOperatorStack函数创建一个二叉树节点；

(4)设置根节点：遍历完成后，栈顶元素即为表达式树的根节点。

3.3.表达式计算功能的实现

表达式计算功能的函数为ExpressionTree 类的成员函数calculate，表 达式计算功能实现的思路为：

(1)计算入口：calculate函数作为入口，调用calculateExpression 函数并传入根节点，开始递归过程；

(2)递归基准情况：如果当前节点为NULL，返回0。如果当前节点是叶子节点（即操作数），返回其数值；

(3)递归处理：对左子节点和右子节点递归调用calculateExpression函数，得到左右子树的计算结果。根据当前节点的运算符（+, -, \*, /），对左右子树的计算结果执行相应的运算；

3.4. 表达式输入非法的处理功能的实现

表达式输入要求为：

(1)输入在一行中以空格分隔不同对象的中缀表达式

(2)表达式只能包含数字、+、-、\*、/、.、以及左右括号

(3)表达式不超过256个字符(不包括空格), 单个运算数不能超过8位数

(4)表达式中的括号嵌套匹配

(5)表达式中每个运算符前后必须连接数字，"-n"请改写成"0-n"

(6)表达式请在一行中输入，按回车键表示输入结束

这段代码会等待用户输入表达式，并对表达式合法性进行检查。如果表达式合法，则函数返回true，如果表达式不合法，则函数返回false。具体执行逻辑如下：

(1)检查空表达式：如果表达式为空（即第一个字符为终止符’\0’），输出提示信息并返回false；

(2)字符合法性检查：遍历表达式中的每个字符。如果字符不是数字（0-9）、运算符（+, -, \*, /）或括号（(, )），输出提示信息并返回false；

(3)检查表达式开头：如果表达式以运算符（+, -, \*, /）开头，输出提示信息并返回false；

(4)处理括号和运算符逻辑：使用栈（expressionStack）来确保表达式中的括号正确匹配，使用标志变量lastIsOperand和lastIsOperator来追踪上一个字符是否为运算符或操作数。对于每个左括号“(”，检查后面是否直接跟 随一个右括号“)”（空括号），如果是，则输出提示信息并返回false。对于每 个右括号“)”，检查栈是否为空或栈顶是否不是左括号“(”，如果是，则输出提示信息并返回false。对于每个运算符，检查前一个字符是否不是操作数，如果是，则输出提示信息并返回false；

(5)结束时的检查：遍历结束后，检查是否所有的括号都已匹配，并检查表达式是否以运算符结束；

(6)返回结果：如果以上所有检查都通过，则函数返回true，表示表达式合法；否则，在任一步骤中遇到非法情况时，函数返回false。

**4.项目测试**

4.1.表达式输入合法性验证功能测试

可以验证程序对表达式输入非法的情况进行了处理。

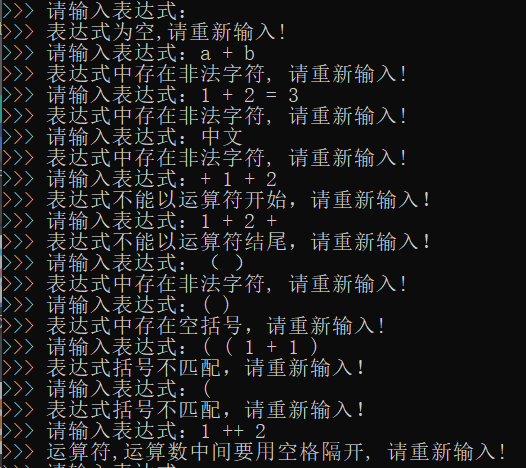


图 4.1 表达式输入合法性功能验证

4.2.构建表达式二叉树和表达式计算功能测试

将中缀表达式转换为后缀表达式并创建表达式二叉树，对表达式二叉树进行前序遍历，中序遍历和后序遍历，分别输出对应的前缀表达式，中缀表达式和后缀表达式，计算表达式的值并进行输出。可以验证程序正确实现了构建表达式二叉树功能和表达式计算功能。

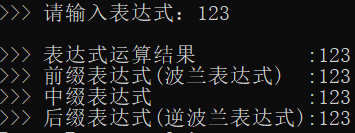


图 4.2.1 构建表达式二叉树和表达式计算功能测试（测试一）

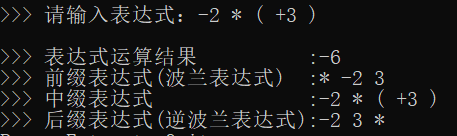


图 4.2.2 构建表达式二叉树和表达式计算功能测试（测试二）

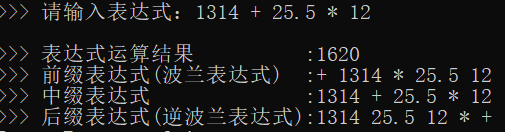


图 4.2.3 构建表达式二叉树和表达式计算功能测试（测试三）

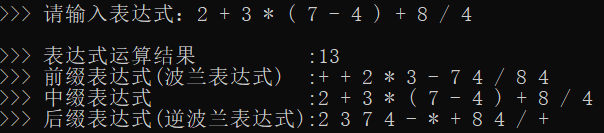


图 4.2.4 构建表达式二叉树和表达式计算功能测试（测试四）

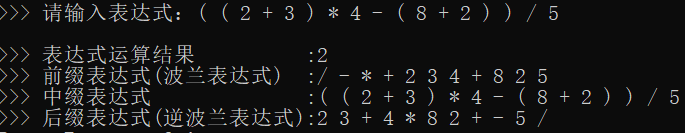


图 4.2.5 构建表达式二叉树和表达式计算功能测试（测试五）