

# 项目说明文档

## 数据结构课程设计

### ——表达式转换

作者姓名：\_\_\_\_\_张翔\_\_\_\_\_

学        号：\_\_\_\_\_2352985\_\_\_\_\_

指导教师：\_\_\_\_\_张颖\_\_\_\_\_

学院、专业：\_\_\_\_\_计算机科学与技术学院 软件工程\_\_\_\_\_

同济大学

Tongji University

二〇二四年十二月七日

# 1 项目分析

## 1.1 项目背景分析

在计算机科学中，算术表达式有多种表示形式，其中最常见的有三种：前缀表示法、中缀表示法和后缀表示法。中缀表示法是我们日常生活中最常见的表示方式，运算符位于操作数之间（例如：3 + 5）。然而，计算机在处理中缀表达式时，需要额外处理运算符的优先级和括号，以保证运算顺序的正确性。而后缀表示法（或逆波兰表示法）则省去了运算符优先级和括号的问题，运算符直接跟随操作数（例如：3 5 +）。因此，将中缀表达式转换为后缀表达式是计算机科学中的一个经典问题，它能够简化计算机对表达式的求值过程。

## 1.2 项目要求分析

该项目要求设计一个程序，将用户输入的中缀算术表达式转换为后缀表达式。程序需要能够解析输入的中缀表达式，包括多位数的整数、浮点数、负数以及不同的运算符（如加法、减法、乘法、除法等），并根据运算符的优先级以及括号的配对规则，正确地将其转换成后缀表示法。此外，程序还需要处理常见的错误情况，如无效的表达式或括号配对错误，并输出相应的错误提示。最终，程序应该输出转换后的后缀表达式，供后续的求值程序使用。

## 1.3 项目功能分析

该项目的主要功能是将中缀表达式转换为后缀表达式。

(1) 程序需要能够解析输入的表达式，识别操作数（数字）、运算符（如 +、-、\*、/）和括号（（和））。

(2) 程序需要运用合适的数据结构，将原来的表达式输出一个后缀表达式，该表达式可以直接用于计算或进一步的处理。

# 2 项目设计

## 2.1 结构体和类设计

### 2.1.1 MyLinkNode 类的设计

#### 2.1.1.1 概述

MyLinkNode 是一个模板结构体，表示链表节点。它包含两个成员：存储节点的数据和指向下一个节点的指针，它定义了两个构造函数，一个为默认，另一个为有参数输入。

#### 2.1.1.2 类定义

```
template <typename Type>
```

```

struct MyLinkNode
{
    Type data;
    MyLinkNode<Type>* link;
    MyLinkNode(MyLinkNode<Type>* ptr = nullptr) : link(ptr) {}
    MyLinkNode(const Type& item, MyLinkNode<Type>* ptr = nullptr) : data(item),
link(ptr) {}
};

```

## 2.1.2 MyQueue 类的设计

### 2.1.2.1 概述

MyQueue 是一个基于链表实现的模板队列类，包含 front 和 rear 指针分别指向队列的头部和尾部，count 用于记录队列中元素的个数。该类提供了基本的队列操作，包括构造函数和析构函数。通过 isEmpty() 判断队列是否为空，Size() 返回队列中元素的数量。enqueue() 用于将元素加入队尾，dequeue() 从队头移除并返回元素，getHead() 获取队头元素。

### 2.1.2.2 类定义

```

template <typename Type>
class MyQueue
{
private:
    MyLinkNode<Type>* front;
    MyLinkNode<Type>* rear;
    int count;
public:
    MyQueue() : front(nullptr), rear(nullptr), count(0) {}
    ~MyQueue() { makeEmpty(); }
    bool isEmpty() const;
    void makeEmpty();
    int Size()const;
    void enqueue(const Type& item);
    bool dequeue(Type& item);
    bool getHead(Type& item);
};

```

```
};
```

## 2.1.3 MyStack 类设计

### 2.1.1.1 概述

MyStack 类实现了一个基于链表的栈数据结构。该类包含私有成员 `topNode`，用于指向栈顶元素的节点，`count` 用于记录栈中元素的数量 `max_size` 限制栈的最大容量。构造函数提供了默认栈大小为 100 的选项，也允许通过指定大小来创建栈。析构函数会在销毁栈对象时清空栈内容。主要的成员函数包括：`isEmpty()` 检查栈是否为空，`makeEmpty()` 清空栈，`Size()` 获取栈中元素数量，`Push()` 向栈中压入元素，`Pop()` 从栈中弹出元素，`getTop()` 获取栈顶元素的值。

### 2.1.1.2 类定义

```
template <typename Type>
class MyStack
{
private:
    MyLinkNode<Type>* topNode;
    int count;
    int max_size;
public:
    MyStack() : topNode(nullptr), count(0), max_size(100) {}
    MyStack(int size) : topNode(nullptr), count(0), max_size(size) {}
    ~MyStack() { makeEmpty(); }
    bool isEmpty() const;
    void makeEmpty();
    int Size() const;
    bool Push(Type& item);
    bool Pop(Type& item);
    bool getTop(Type& item);
};
```

## 3 项目功能实现

### 3.1 输入算术表达式功能实现

#### 3.1.1 输入算术表达式功能实现思路

该部分是用于验证用户输入的算术表达式是否合法，并检查其中是否存在语法错误，具体实现思路如下：

(1) 通过标准输入读取用户输入的表达式并存储到字符数组中。随后，代码进行一系列合法性检查：第一步，检查表达式的起始和结尾字符是否符合规则，例如不能以某些运算符（\*、/）或右括号开头，也不能以左括号或运算符结尾；

(2) 逐字符遍历表达式，通过忽略空格逐步分析每个元素。对于数字，允许小数点并确保小数点不重复，同时检测后续字符是否非法（例如紧跟左括号）；对于运算符，验证其后方不能是另一个运算符或右括号；对于括号，统计左右括号的数量，确保左右括号能正确配对且没有语法错误，例如左括号后直接跟右括号是非法的；其他字符直接判定为非法。

(3) 通过检测括号配对情况判断表达式整体是否合法，并返回表达式格式是否正确。

#### 3.1.2 输入算术表达式功能核心代码

```
bool MyExpression::inputExpression()
{
    std::cout << ">>> 请输入一个算术表达式（在每个运算数/运算符之间用空格隔开）" <<
    std::endl << std::endl;

    std::cin.getline(expression, sizeof(expression));
    size_t len = std::strlen(expression);
    if ((expression[0] == '*' || expression[0] == '/' || expression[0] == ')') ||
        (isOperator(expression[len - 1]) || expression[len - 1] == '(')) {
        return false;
    }

    int openParentheses = 0;
    for (size_t i = 0; i < len; i) {
        char element = expression[i];
        if (element == ' ') {
            i++;
        }
    }
}
```

```

        continue;
    }
    size_t j = 1;
    while (expression[i + j] == ' ') {
        j++;
    }
    if (isOperator(element)) {
        if (isOperator(expression[i + j]) || expression[i + j] == ')') {
            return false;
        }
    }
    else if (std::isdigit(element) || (element == '-' && std::isdigit(expression[i
+ 1]))) {
        // 标记是否已经遇到小数点
        bool hasDecimalPoint = false;
        while (std::isdigit(expression[i + j]) || expression[i + j] == '.') {
            if (expression[i + j] == '.') {
                // 如果已经有一个小数点, 非法
                if (hasDecimalPoint) {
                    return false;
                }
                hasDecimalPoint = true;
            }
            j++;
        }
        if (expression[i + j] == '(') {
            return false;
        }
    }
    else if (element == '(') {
        openParentheses++;
    }

```

```

        if (expression[i + j] == ')') {
            return false;
        }
    }
    else if (element == ')') {
        openParentheses--;
        if (expression[i + j] == '(') {
            return false;
        }
    }
    else {
        return false;
    }
    i += j;
}
return openParentheses == 0;
}

```

## 3.2 中缀表达式转后缀功能实现

### 3.2.1 中缀表达式转后缀功能实现思路

这段代码实现将中缀表达式转换为后缀表达式（逆波兰表达式）的具体思路如下：

(1) 通过输入流逐个解析中缀表达式中的元素（数字、运算符或括号）：

对于数字，直接加入输出队列。

对于运算符，利用栈（s1）管理运算符优先级：当前运算符与栈顶运算符比较优先级，如果栈顶优先级较高或相等，则弹出栈顶运算符到输出队列；否则将当前运算符压入栈中。

对于左括号，直接压入栈；

对于右括号，则持续弹出栈顶元素到输出队列，直到遇到左括号（同时移除左括号）。

对于无法识别的符号，输出警告信息。

(2) 将栈中剩余的运算符依次弹出到输出队列。

(3) 通过输出队列输出后缀表达式，确保表达式中的元素以正确的顺序排列。

### 3.2.2 中缀表达式转后缀功能核心代码

```
void MyExpression::infixToPostfix()
{
    std::istringstream stream(expression);
    char token[MAX_TOKEN];
    while (stream >> token) {
        Token t;
        /* 数字处理 */
        if (std::isdigit(token[0]) ||
            ((token[0] == '-' || token[0] == '+') && std::isdigit(token[1]))) {
            std::strcpy(t.value, token);
            t.isNumber = true;
            q1.enqueue(t);
            continue;
        }
        /* 将运算符或括号拷贝到 Token */
        std::strcpy(t.value, token);
        t.isNumber = false;

        /* 运算符处理 */
        if (isOperator(t.value[0])) {
            while (!s1.isEmpty()) {
                Token top;
                s1.getTop(top);
                /* 比较优先级并处理栈顶元素 */
                if (!top.isNumber && precedence(top.value) >= precedence(t.value)) {
                    Token temp;
                    s1.Pop(temp);
                    q1.enqueue(temp);
                }
            }
            else {
```



```

        break;
    }
}
s1.Push(t);
}
/* 左括号处理 */
else if (std::strcmp(t.value, "(") == 0) {
    s1.Push(t);
}
/* 右括号处理 */
else if (std::strcmp(t.value, ")") == 0) {
    while (!s1.isEmpty()) {
        Token top;
        s1.getTop(top);

        if (std::strcmp(top.value, "(") == 0) {
            s1.Pop(top);
            break;
        }
        Token temp;
        s1.Pop(temp);
        q1.enqueue(temp);
    }
}
/* 未知符号处理 */
else {
    std::cout << "Unrecognized token: " << token << std::endl;
}
}

/* 将栈中剩余的元素全部弹出到队列 */

```

```

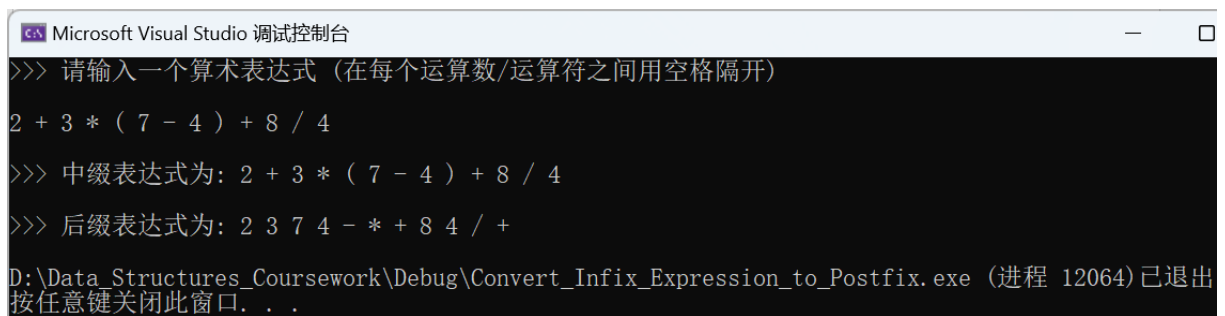
while (!s1.isEmpty()) {
    Token t;
    s1.Pop(t);
    q1.enqueue(t);
}

std::cout << std::endl << ">>> 后缀表达式为: ";
/* 输出后缀表达式 */
while (!q1.isEmpty()) {
    Token t;
    q1.dequeue(t);
    std::cout << t.value;
    if (!q1.isEmpty()) {
        std::cout << " ";
    }
}

std::cout << std::endl;
}

```

## 4 项目测试



```

Microsoft Visual Studio 调试控制台
>>> 请输入一个算术表达式 (在每个运算数/运算符之间用空格隔开)
2 + 3 * ( 7 - 4 ) + 8 / 4
>>> 中缀表达式为: 2 + 3 * ( 7 - 4 ) + 8 / 4
>>> 后缀表达式为: 2 3 7 4 - * + 8 4 / +
D:\Data_Structures_Coursework\Debug\Convert_Infix_Expression_to_Postfix.exe (进程 12064) 已退出
按任意键关闭此窗口. . .

```

### 4.1 正常 6 种运算符测试

```
Microsoft Visual Studio 调试控制台
>>> 请输入一个算术表达式 (在每个运算数/运算符之间用空格隔开)
( ( 2 + 3 ) * 4 - ( 8 + 2 ) ) / 5
>>> 中缀表达式为: ( ( 2 + 3 ) * 4 - ( 8 + 2 ) ) / 5
>>> 后缀表达式为: 2 3 + 4 * 8 2 + - 5 /
D:\Data_Structures_Coursework\Debug\Convert_Infix_Expression_to_Postfix.exe (进程 15228) 已退出
按任意键关闭此窗口. . .
```

#### 4.2 嵌套括号测试

```
Microsoft Visual Studio 调试控制台
>>> 请输入一个算术表达式 (在每个运算数/运算符之间用空格隔开)
1314 + 25.5 * 12
>>> 中缀表达式为: 1314 + 25.5 * 12
>>> 后缀表达式为: 1314 25.5 12 * +
D:\Data_Structures_Coursework\Debug\Convert_Infix_Expression_to_Postfix.exe (进程 37780) 已退出
按任意键关闭此窗口. . .
```

#### 4.3 运算数超过 1 位整数且有非整数出现测试

```
Microsoft Visual Studio 调试控制台
>>> 请输入一个算术表达式 (在每个运算数/运算符之间用空格隔开)
-2 * ( +3 )
>>> 中缀表达式为: -2 * ( +3 )
>>> 后缀表达式为: -2 +3 *
D:\Data_Structures_Coursework\Debug\Convert_Infix_Expression_to_Postfix.exe (进程 2136) 已退出
按任意键关闭此窗口. . .
```

#### 4.4 运算数有正或负号测试

## 5 集成开发环境与编译运行环境

Windows 系统: Windows 11 x64

Windows 集成开发环境: Microsoft Visual Studio 2022 (Release 模式)

Windows 编译运行环境: 本项目适用于 x86 架构和 x64 架构