****

**实验报告：表达式转换**

2351041

刘浩田

完成日期:2024.9.28

# 索引

### 概述

### 项目目的

### 项目环境

### 项目背景

### 项目要求

### 项目实现

### 项目总结

# 概述

在计算机科学中，算术表达式的表示方法多种多样，其中最为常见的是前缀表示法、中缀表示法和后缀表示法。中缀表示法因其接近日常数学表达习惯而被广泛使用，但在计算机程序中，后缀表示法因其在解析和执行上的便利性而受到青睐。因此，将中缀表达式转换为后缀表达式是编程中的一项基本任务，它不仅有助于理解不同表达式表示法之间的转换机制，也是实现算术表达式求值算法的重要步骤。

本项目的核心目标是设计并实现一个程序，该程序能够将用户输入的中缀表达式转换为后缀表达式。这一转换过程涉及到对中缀表达式进行解析，并根据运算符的优先级和结合性规则，重新组织表达式中的运算数和运算符，以生成等价的后缀表达式。

在实现这一转换的过程中，我们采用了栈这一数据结构来辅助处理运算符的优先级和括号的匹配问题。具体来说，程序首先读取用户输入的中缀表达式，然后逐个字符分析表达式中的元素，包括数字、运算符和括号。对于每个元素，程序会根据其类型采取不同的处理策略：数字直接添加到后缀表达式中；运算符则需要根据其优先级与栈中已有运算符的优先级进行比较，以决定是直接压栈还是弹出栈中的运算符并添加到后缀表达式中；括号则用于控制运算的顺序，左括号表示新的子表达式的开始，而右括号则触发子表达式的结束和运算符的弹出。

在处理过程中，程序还需注意处理特殊情况，如负数的表示、空格的忽略以及运算符的优先级处理等。此外，程序还需确保在输出后缀表达式时，不同对象之间以空格分隔，且末尾没有多余的空格，以满足输出格式的要求。

为了确保程序的广泛适用性，我们在设计时特别考虑了跨平台兼容性，使得该程序不仅在Windows操作系统上运行流畅，同时也兼容Linux环境，满足了不同用户群体的需求。通过使用标准C++语言和遵循Linux标准的函数，我们的程序能够在两大主流操作系统上提供一致的用户体验。

通过本项目的实现，我们不仅能够加深对栈这一数据结构在实际应用中的理解，还能够掌握中缀表达式和后缀表达式转换的原理和方法。这对于后续学习编译原理、算法设计等领域的知识具有重要的基础性作用。同时，项目也锻炼了我们对字符串处理、算法实现和程序调试的能力，为解决更复杂的编程问题打下了坚实的基础。

# 项目目的

本项目的核心目的在于通过设计和实现一个将中缀表达式转换为后缀表达式的程序，深化学生对数据结构、算法以及栈的应用的理解和应用能力。在计算机科学教育中，理论与实践相结合是培养学生解决实际问题能力的重要途径。通过本实验，学生不仅能够将课堂上学到的理论知识运用到具体项目中，还能够在实践中进一步提升自己的编程技巧和软件设计能力。

首先，实验旨在加强学生对栈数据结构的理解和应用。算术表达式的转换涉及到大量的运算符和运算数的存储、检索、插入和删除操作，这些操作都需要高效的数据结构来支持。通过本实验，学生可以深入理解栈这种数据结构的特点和适用场景，掌握它在实际问题中的应用方法。特别是在处理算术表达式转换时，栈结构的后进先出（LIFO）特性显得尤为重要。

其次，实验强调了算法的重要性。一个优秀的算法能够显著提高程序的执行效率，对于算术表达式转换系统而言，这直接关系到用户体验。例如，在实现中缀表达式转换功能时，选择合适的算法可以加快转换速度；在进行错误处理时，合理的算法可以减少计算的复杂度。通过本实验，学生可以练习设计和优化算法，提高解决实际问题的效率。

再次，实验着重于提升学生的面向对象编程能力。面向对象编程是一种重要的编程范式，它通过将数据和操作封装成对象，提高了代码的可重用性和可维护性。在本实验中，学生需要构建多个类来表示不同的概念，如栈、运算符、运算数等，并设计类的属性和方法来实现系统的功能。这要求学生不仅要掌握面向对象编程的语法，还要学会如何运用面向对象的思想进行系统设计。

此外，实验还旨在培养学生的软件工程意识。在实际的软件开发过程中，除了编写代码，还需要进行需求分析、系统设计、测试和维护等工作。通过本实验，学生可以体验软件开发的全过程，了解各个阶段的任务和要求。特别是在系统设计阶段，学生需要考虑系统的结构、模块划分、接口设计等问题，这对于提高学生的系统分析和设计能力具有重要意义。

最后，实验还关注于提升学生的用户体验意识。一个好的软件不仅要功能强大，还要易于使用。在本实验中，学生需要设计用户友好的交互界面，确保用户能够方便地进行操作。这要求学生不仅要关注程序的内部逻辑，还要考虑用户的需求和习惯，培养从用户的角度思考问题的能力。

综上所述，本实验目的在于通过实际操作，使学生在多方面得到锻炼和提升，为将来的学习和工作打下坚实的基础。通过本实验，学生不仅能够提高自己的编程能力和系统设计能力，还能够培养良好的软件开发习惯和用户意识，为成为一名优秀的软件工程师奠定基础。

# 项目环境

## 系统：Windows11

## 编译器：VisualStudio2022x86

## 语言：C++

# 项目背景

随着信息技术的不断进步与发展，计算机在处理日常任务时的性能和效率要求越来越高。在编程和算法设计中，算术表达式的处理是一个基础而普遍的问题。传统的中缀表达式虽然易于人类理解，但对计算机来说，解析和计算过程相对复杂。而后缀表达式由于其运算符顺序明确、无需括号区分运算顺序等优点，更适合计算机处理。因此，实现从中缀表达式到后缀表达式的转换对于提高计算机处理算术表达式的效率具有重要意义。

在实际应用中，后缀表达式被广泛应用于计算器、编程语言的编译器、解释器等场景。例如，在编译原理中，将源代码转换为后缀表达式是生成中间代码的重要步骤。此外，在嵌入式系统和移动设备上，由于资源有限，使用后缀表达式可以减少计算的复杂度，提高程序的执行效率。

本项目正是在这样的需求驱动下进行的设计和实现。通过开发一个将中缀表达式转换为后缀表达式的程序，旨在为教育机构提供一个电子化的解决方案，以应对日益增长的算术表达式处理需求，并解决传统处理方式中存在的问题。本项目不仅能够提高数据处理的速度和准确性，还减少了人力资源的消耗，并提高了用户（编程人员及计算机系统）的体验。

在这样的背景下，开发一个功能完善、操作简便、安全可靠的算术表达式转换系统变得尤为重要。本项目正是在这样的需求驱动下进行的设计和实现。该系统旨在为编程人员提供一个电子化的解决方案，以应对日益增长的算术表达式处理需求，并解决传统处理方式中存在的问题。通过本项目，我们期望能够提供一个高效、准确、易于使用的算术表达式转换工具，有效提升算术表达式处理的效率和质量。。

# 项目要求

## 1. 功能要求

- 输入处理：程序需要能够接收用户输入的一行中缀表达式，表达式中的元素通过空格分隔，支持加、减、乘、除运算符以及括号。

- 转换功能：程序必须实现将输入的中缀表达式转换为后缀表达式的核心功能。

- 输出处理：程序应输出转换后的后缀表达式，元素间以空格分隔，末尾无多余空格。

## 2. 数据结构要求

- 栈结构：程序应使用栈数据结构来暂存运算符，确保运算符的顺序和优先级得到正确处理。

- 数据完整性：程序需确保输入表达式中的所有数据在转换过程中的完整性和准确性。

## 3. 用户界面要求

- 操作简便：用户界面应直观易用，操作流程简单明了。

- 输入验证：程序应提供输入验证，确保用户输入的数据符合要求。

## 4. 性能要求

- 效率：程序应能够高效地处理表达式转换，即使在处理较长的表达式时也应保持良好的性能。

- 准确性：程序应确保转换结果的准确性，正确处理所有运算符和括号。

## 5. 可复用性和可维护性

- 模块化设计：程序应采用模块化设计，便于后期维护和升级。

- 代码可读性：代码应具有良好的可读性，注释清晰，便于他人理解和修改。

## 6. 测试要求

- 测试用例：为每个功能编写详细的测试用例，确保功能的正确性和稳定性。

- 边界测试：进行边界条件测试，确保程序在极端情况下也能正常工作。

通过满足以上要求，算术表达式转换程序将为用户提供一个功能全面、性能优异、安全可靠、易于使用的算术表达式处理工具，有效提升算术表达式处理的效率和质量。程序的设计和实现将充分考虑用户的需求和体验，确保程序的实用性和可靠性。

# 项目实现

项目通过各类、函数、结构共同设计，完成了项目要求。

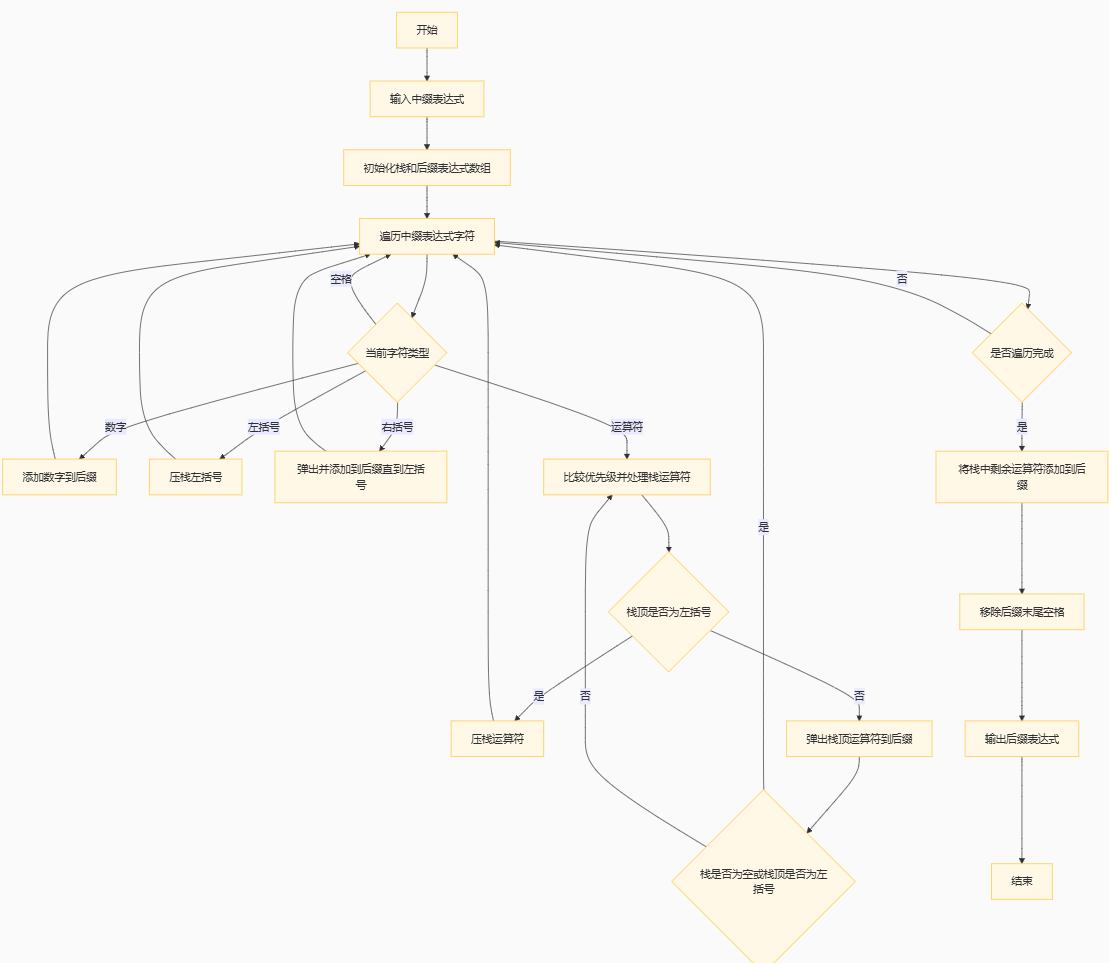


图 项目流程图

## 7.1类设计

本项目主要实现了一个栈（Stack）类模板，用于中缀表达式到后缀表达式（逆波兰表达式）的转换。以下是类的详细设计与实现：

|  |
| --- |
| 附件、Stack类 |
| template <class T>  class Stack {  public:  // 构造函数，默认大小为kMaxSize  explicit Stack(int size = kMaxSize);  // 将元素压入栈顶  void Push(const T& item);  // 弹出并返回栈顶元素  T Pop();  // 获取栈顶元素  T GetTop();  // 清空栈  void MakeEmpty();  // 检查栈是否为空  bool IsEmpty() const;  // 检查栈是否已满  bool IsFull() const;  // 获取当前栈顶索引  int GetTopIndex() const;  private:  int top\_; // 栈顶指针  T\* elements\_; // 存储元素的数组  int max\_size\_; // 栈的最大容量  }; |

Stack类是一个模板类，用于实现栈数据结构。它的主要特点包括：  
1. 私有成员：  
- `top`：栈顶指针  
- `elements`：存储栈元素的数组  
- `maxSize`：栈的最大容量  
2. 公有成员函数：  
- `Stack(int = kMaxSize)`：构造函数，默认大小为kMaxSize  
- `Push(const T& item)`：将元素压入栈顶  
- `Pop()`：弹出栈顶元素并返回  
- `GetTop()`：获取栈顶元素但不删除  
- `MakeEmpty()`：清空栈  
- `IsEmpty()`：检查栈是否为空  
- `IsFull()`：检查栈是否已满  
- `GetTopIndex()`：检查栈顶索引  
Stack类的实现细节：  
1. 构造函数：

|  |
| --- |
| 附件、构造函数： |
| template <class T>  Stack<T>::Stack(int size) : top\_(-1), max\_size\_(size) {  // 分配内存给元素数组  elements\_ = new T[max\_size\_];  // 断言确保内存分配成功  assert(elements\_ != nullptr);  } |

构造函数初始化栈，动态分配内存并使用assert确保内存分配成功。  
  
2. Push操作：

|  |
| --- |
| 附件、Push函数： |
| template <class T>  void Stack<T>::Push(const T& item) {  // 断言确保栈未满  assert(!IsFull());  // 将元素放入栈顶并增加栈顶指针  elements\_[++top\_] = item;  } |

Push操作首先检查栈是否已满，然后将新元素添加到栈顶。  
  
3. Pop操作：

|  |
| --- |
| 附件、Pop函数： |
| template <class T>  T Stack<T>::Pop() {  // 断言确保栈非空  assert(!IsEmpty());  // 返回栈顶元素并减少栈顶指针  return elements\_[top\_--];  } |

Pop操作首先检查栈是否为空，然后返回并删除栈顶元素。  
  
4. 其他操作如GetTop、MakeEmpty、IsEmpty、IsFull等都有相应的实现，确保了栈的正确操作。

## 7.2 主函数实现

主函数实现了中缀表达式的输入和后缀表达式的函数调用与输出。

|  |
| --- |
| 附件、main函数： |
| int main() {  char infix[kMaxSize]; // 存储输入的中缀表达式  char postfix[kMaxSize \* 2]; // 存储转换后的后缀表达式  // 提示用户输入中缀表达式  cout << "请输入中缀表达式：" << endl;  cin.getline(infix, kMaxSize);  // 调用函数将中缀表达式转换为后缀表达式  InfixToPostfix(infix, postfix);  // 输出转换后的后缀表达式  cout << "转换后的后缀表达式：" << endl;  cout << postfix << endl;  #ifdef \_WIN32  system("pause");  #elif defined(\_\_linux\_\_) || defined(\_\_APPLE\_\_)  cout << "按回车键继续..." << endl;  cin.get();  #endif  return 0;  } |

## 7.3转换步骤和逻辑

在本项目中，实现中缀表达式到后缀表达式的转换是核心功能。这一过程涉及到对输入的中缀表达式进行解析，并按照后缀表达式的规则重新组织运算符和运算数。以下是详细的转换步骤和逻辑：

|  |
| --- |
| 附件、InfixToPostfix函数： |
| void InfixToPostfix(const char\* infix, char\* postfix) {  Stack<char> op\_stack; // 用于存储运算符的栈  int postfix\_index = 0; // 后缀表达式的当前索引  bool expect\_operand = true; // 标志，用于判断是否期望一个操作数  // 遍历中缀表达式的每个字符  for (int i = 0; infix[i] != '\0'; i++) {  // 跳过空格  if (infix[i] == ' ') {  continue;  }  // 处理数字、负号和小数点  if (isdigit(infix[i]) || (infix[i] == '-' && expect\_operand) || infix[i] == '.') {  // 如果是负号且期望操作数，则将负号加入后缀表达式  if (infix[i] == '-' && expect\_operand) {  postfix[postfix\_index++] = infix[i++];  }  // 将连续的数字和小数点添加到后缀表达式  while (isdigit(infix[i]) || infix[i] == '.') {  postfix[postfix\_index++] = infix[i++];  }  // 数字后添加空格作为分隔符  postfix[postfix\_index++] = ' ';  i--; // 回退一个字符，因为for循环会再次i++  expect\_operand = false; // 不再期望操作数  }  // 处理左括号  else if (infix[i] == '(') {  op\_stack.Push(infix[i]);  expect\_operand = true; // 左括号后期望操作数  }  // 处理右括号  else if (infix[i] == ')') {  // 将栈中运算符弹出并加入后缀表达式，直到遇到左括号  while (!op\_stack.IsEmpty() && op\_stack.GetTop() != '(') {  postfix[postfix\_index++] = op\_stack.Pop();  postfix[postfix\_index++] = ' ';  }  // 弹出左括号（如果存在）  if (!op\_stack.IsEmpty() && op\_stack.GetTop() == '(') {  op\_stack.Pop();  }  expect\_operand = false; // 右括号后不期望操作数  }  // 处理运算符  else if (IsOperator(infix[i])) {  // 忽略正号  if (infix[i] == '+' && expect\_operand) {  continue;  }  // 将优先级更高或相等的运算符从栈中弹出并加入后缀表达式  while (!op\_stack.IsEmpty() && op\_stack.GetTop() != '(' &&  GetOperatorPriority(op\_stack.GetTop()) >= GetOperatorPriority(infix[i])) {  postfix[postfix\_index++] = op\_stack.Pop();  postfix[postfix\_index++] = ' ';  }  // 将当前运算符压入栈  op\_stack.Push(infix[i]);  expect\_operand = true; // 运算符后期望操作数  }  }  // 将栈中剩余的运算符添加到后缀表达式  while (!op\_stack.IsEmpty()) {  postfix[postfix\_index++] = op\_stack.Pop();  postfix[postfix\_index++] = ' ';  }  // 移除末尾多余的空格  if (postfix\_index > 0 && postfix[postfix\_index - 1] == ' ') {  postfix\_index--;  }  // 添加字符串结束符  postfix[postfix\_index] = '\0';  } |

**1. 初始化和输入处理**

首先，程序初始化一个栈（Stack<char> op\_stack）用于存储运算符，并定义一个字符串（char postfix[kMaxSize \* 2]）用于存储转换后的后缀表达式。程序提示用户输入中缀表达式，并使用cin.getline函数读取用户输入，存储在infix数组中。

**2. 遍历中缀表达式**

程序遍历输入的中缀表达式中的每个字符。对于每个字符，程序执行以下操作：

**跳过空格**：如果字符是空格，则跳过，继续处理下一个字符。

**处理数字和负号**：如果字符是数字或负号（且负号出现在期望操作数的位置），则将其添加到后缀表达式中。如果是负号，还需要检查下一个字符是否为数字，以确保正确处理负数。

**处理左括号**：如果字符是左括号(，则将其压入栈中。

**处理右括号**：如果字符是右括号)，则将栈中的运算符弹出并添加到后缀表达式中，直到遇到左括号为止。左括号从栈中弹出但不添加到后缀表达式中。

**处理运算符**：如果字符是运算符（加、减、乘、除），则比较其优先级与栈顶运算符的优先级。如果栈顶运算符的优先级更高或相等，且栈顶不是左括号，则将栈顶运算符弹出并添加到后缀表达式中。然后将当前运算符压入栈中。

**3. 处理运算符优先级**

在处理运算符时，程序使用GetOperatorPriority函数来获取运算符的优先级。乘法和除法的优先级为2，加法和减法的优先级为1。如果运算符的优先级相同或更高，程序会将栈中的运算符弹出并添加到后缀表达式中，直到遇到优先级较低的运算符或左括号。

**4. 完成转换**

遍历完中缀表达式后，栈中可能还剩余一些运算符。程序将这些运算符弹出并添加到后缀表达式中，直到栈为空。

**5. 移除末尾空格和添加字符串结束符**

最后，程序检查后缀表达式末尾是否有多余的空格，并将其移除。然后，在后缀表达式的末尾添加字符串结束符\0，以标记字符串的结束。

通过上述步骤，程序能够将用户输入的中缀表达式转换为后缀表达式。这个过程不仅涉及到栈的操作，还涉及到对运算符优先级和括号匹配的处理，确保了转换的正确性和效率。

# 项目总结

本次实验项目是设计并实现一个中缀表达式转换为后缀表达式的程序。通过这个项目，我深入理解了数据结构、算法以及面向对象编程在解决实际问题中的应用，同时也提升了自己的编程技巧和软件设计能力。

**8.1学习体会**

通过将课堂上学到的数据结构和算法知识应用到具体项目中，我深刻体会到了理论知识在解决实际问题中的重要性。

项目让我更加熟悉了面向对象的设计原则，如封装、继承和多态。通过定义清晰的类和对象，我提高了代码的可读性和可维护性。

在项目开发过程中，我学习了如何规划和设计一个完整的系统，包括需求分析、系统架构设计、模块划分等。

**8.2技术收获**

通过本项目，我提高了C++编程技能，特别是在STL的使用、异常处理和输入输出操作等方面。

我深入理解了栈等数据结构的特点和适用场景，掌握了它们在实际问题中的应用方法。

项目中实现了多种算法，如排序算法、查找算法等，提高了我的算法设计和优化能力。

**8.3问题与挑战**

在用户输入错误或系统异常情况下，如何保证系统的健壮性是一个难题。我通过增加输入验证和异常捕获机制来提高系统的稳定性。

设计一个用户友好的界面也是项目中的一个挑战。我通过不断迭代和改进用户界面，使其更加直观和易用。

**8.4改进方向**

未来可以增加更多功能，如数据导出、数据导入、表达式计算等，使其更符合实际应用需求。

可以探索更高效的数据结构和算法，进一步提高系统的处理速度和响应能力。

可以进一步优化用户界面和交互设计，提供更加友好和直观的用户体验。

考虑到系统处理的是个人信息，未来可以加入更多的安全特性，如数据加密、访问控制等。

**8.5结论**

这次课程设计是一次宝贵的学习经历。它不仅让我将所学知识应用于实践，还让我学会了如何分析和解决实际问题。通过团队合作，我克服了各种挑战，最终完成了一个功能完善、性能良好的表达式转换系统。我相信，这次经历将对我未来的学习和工作产生积极的影响。