四川大学计算机学院 （软件学院、 智能科学与技术学院）

实验报告

学号：x 姓名： x 专业： 人工智能 班级： 233040701 第 7 周

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 课程名称 | 数据结构与算法分析课程设计 | | 实验课时 | 4课时 |
| 实验题目 | 算数表达式求值 | | 实验时间 | 2024/10/17 |
| 实验目的和要求 | **基本要求：**   1. 采用算符优先级算法，能正确求表达式的值 2. 熟练掌握栈的应用 3. 熟练掌握计算机系统的基本操作方法，了解如何编辑、编译、链接和运行一个C++程序 4. 上机调试程序，掌握查错、排错使程序能正常运行。   已完成的扩展要求：   1. 增加浮点数 2. 单目运算符（阶乘!、负数-） 3. 多种括号 4. 末尾等号可选 5. 科学计算支持（exp、sin、cos） | | | |
| 实验环境 | Clion2024、MinGW11.0、C++17 | | | |
| 算法描述 | **概要描述**：   1. 首先分析表达式字符串，将其拆分为一个vector 2. 再利用栈与符号优先级顺序将这个vector解析为后缀表达式 3. 将后缀表达式解析为抽象语法树(AST)，其中函数可以看成特殊的单目运算符 4. 对抽象语法树进行求值运算，完成整个计算过程   **具体描述**：  **对于第一个问题**，如何将字符串解析为vector，可以将字符串从各种符号处分割(+-\*/!^())，这样就能得到一个字符串列表，每个元素代表着一个单位（数字或符号）。  **对于第二个问题**，由于需要处理不同括号的问题，可以先校验括号合法性，若合法，就可以直接将各种括号都替换为小括号，方便下面转换后缀表达式的处理。对于如何校验括号合法性，可以创建一个栈，从左往右扫描，如果是左括号就入栈，如果是右括号，那么它应当与栈顶相匹配（若不匹配就抛出异常），扫描完成后栈应当为空（不为空也抛出异常）。对于后缀表达式转换，算法如下：   1. 初始化一个空的操作符栈 operatorStack，用于存放暂时无法确定顺序的操作符。 2. 初始化一个空的后缀表达式列表 postfix，用于存放最终转换后的表达式。 3. 遍历输入的中缀表达式 infix 中的每一个元素 i。    * 如果 i 是一个操作符（由 isOperator 函数判断）：      + 如果 i 是左括号 "("，将其直接压入 operatorStack。      + 如果 i 是右括号 ")"，则从 operatorStack 中弹出操作符，直到遇到左括号 "(" 为止，每次弹出的操作符都加入到 postfix 中，左括号 "(" 被弹出但不加入 postfix。      + 如果 i 是其他操作符（如 +、-、\*、/ 等），则将 operatorStack 中优先级大于等于 i 的操作符弹出并加入到 postfix 中，然后将 i 压入 operatorStack。    * 如果 i 不是操作符，即它是一个操作数，则直接将其加入到 postfix 中。 4. 遍历完中缀表达式后，如果 operatorStack 中还有剩余的操作符，依次弹出并加入到 postfix 中。 5. 返回 postfix 作为转换后的后缀表达式。   在这个过程中，getPriority 函数用于获取操作符的优先级，以便在处理操作符时确定弹出的顺序。操作符的优先级通常定义为：\* 和 / 的优先级高于 + 和 -，而括号具有最高的优先级，用于改变操作符的默认顺序。  对于**第三个问题**，通过后缀表达式建立抽象语法树，算法如下：   1. 初始化一个空栈 stack，用于构建抽象语法树。栈中存储的是 AstNode\* 类型的指针，每个 AstNode 代表树中的一个节点。 2. 遍历输入的后缀表达式 postfix 中的每一个元素 i。    * 如果 i 是一个操作符（由 isOperator 函数判断）：      + 如果 i 是一个函数（由 isFunction 函数判断）：        - 检查栈的大小是否小于1，如果是，则抛出异常，因为函数至少需要一个参数。        - 创建一个新的 AstNode，其类型为 ElementType::function，并将其操作符设置为 i。        - 将栈顶的节点作为函数的左子节点（参数），并从栈中弹出该节点。        - 将新创建的函数节点压入栈中。      + 如果 i 不是函数：        - 创建一个新的 AstNode，其类型为 ElementType::op，并将其操作符设置为 i。        - 根据 i 的类型，处理不同的情况：          * 如果 i 是一元操作符（例如阶乘运算 !）：   检查栈是否为空，如果是，则抛出异常，因为一元操作符需要一个参数。  将栈顶的节点作为操作符的左子节点，并从栈中弹出该节点。   * + - * + 如果 i 是减号 -（它可能是一元操作符也可能是二元操作符）：   如果栈的大小大于等于2，则将栈顶的两个节点作为操作符的左右子节点，并从栈中弹出这两个节点。  如果栈的大小等于1，则将栈顶的节点作为操作符的右子节点（表示一元操作符），并从栈中弹出该节点。  如果栈为空，则抛出异常，因为减号至少需要一个参数。   * + - * + 对于其他二元操作符（例如 +、\*、/ 等）：   检查栈的大小是否小于2，如果是，则抛出异常，因为二元操作符需要两个参数。  将栈顶的两个节点分别作为操作符的左右子节点，并从栈中弹出这两个节点。   * + - * 将新创建的操作符节点压入栈中。   + 如果 i 不是操作符，即它是一个立即数（操作数）：     - 创建一个新的 AstNode，其类型为 ElementType::immediate，并将其立即数设置为 i 转换为 double 类型的值。     - 将新创建的立即数节点压入栈中。  1. 遍历完整个后缀表达式后，检查栈的大小：    * 如果栈的大小为1，则返回栈顶的节点，它就是构建完成的抽象语法树的根节点。    * 如果栈的大小不为1，则抛出异常，表示参数过多，无法构建一个有效的抽象语法树。   对于**第四个问题**，对抽象语法树进行求值操作，只需要调用根节点的求值函数，求值函数会递归地对子节点进行求值操作：   1. **递归定义**：在抽象语法树中，每个节点代表表达式的一个子表达式。节点可以是操作数（立即数）、操作符或函数。每个节点都有可能包含子节点，这些子节点又是更小的表达式。这种递归结构允许我们在处理任何节点时，都可以递归地处理其子节点。 2. **后序遍历**：在计算表达式时，先计算子表达式的值，然后再根据操作符或函数来计算当前节点的值。在 AstNode 类的 eval 方法中，这一点体现在以下两个方面：    * 对于操作符节点，首先递归地计算左子节点和右子节点的值，然后根据操作符执行相应的运算。    * 对于函数节点，首先递归地计算参数（即左子节点）的值，然后执行函数运算。 3. **操作符和函数的定义**：在 AstNode 类的 eval 方法中，每个操作符和函数都有明确的数学定义和求值规则。例如，对于加法操作符（+），我们首先计算左子节点和右子节点的值，然后将它们相加。对于函数（如 SIN、COS、EXP），我们首先计算参数的值，然后应用相应的数学函数。 4. **递归终止条件**：递归求值的终止条件是到达立即数节点（ElementType::immediate），此时直接返回该节点的值，因为立即数没有子节点，它代表表达式的最基本单位。   对于**其他方面**，因为构建抽象语法树的过程中使用了堆空间，因此需要创建析构函数，该函数后序递归地将每个节点析构。 | | | |
| 源程序清单 | AstNode.h 定义抽象语法树的节点  ExpressionHandler.h 处理中缀表达式，构建抽象语法树  Main.cpp 程序入口与命令行交互 | | | |
| 运行结果 | 输入Q退出 | | | |
| 实验运行情况分析(包括算法、运行结果、运行环境等问题的讨论) | **运行结果：**  程序能正确地处理小数输入、四则运算、多种括号、科学计算、单目运算符、幂运算、阶乘运算。程序也能检测大部分表达式中的错误：括号不匹配，没闭合，运算符错误等等。  **局限性：**  由于程序是先将中缀表达式转化为后缀表达式，程序只在运行过程中校验了后缀表达式的合法性。也就是说如果输入不是一个标准的中缀表达式（比如就在这里输入了后缀表达式），但是经过了转换变为了合法的后缀表达式，本程序并不会检测到异常。 | | | |
| 指导老师  评议 |  | | | |
| 成绩评定： | 指导教师签名： | | |