编译器构造实验　Lab04 实验报告

基于表达式的计算器

课程编号：DCS292

学生姓名：傅祉珏

学生学号：21307210

指导老师：李文军 教授

项目截止日期：2025 年 5 月 15 日

2025 年 5 月 14 日

引言

表达式处理是编译器前端的重要组成部分，涵盖了从词法识别、语法结构分析到语义求值的一整套处理机制。为了深入理解编译原理中各模块的设计逻辑与协同运行方式，本实验以“基于表达式的计算器”为核心任务，构建了一个支持复杂表达式求值的分析系统。该系统不仅具备基本的四则运算能力，还进一步支持幂运算、关系与逻辑表达式、一元与三元运算、括号嵌套结构以及预定义函数调用等复杂表达形式，语法覆盖面广、语义约束严格，具有较强的工程实用性与教学示范意义。

在语法分析方面，系统引入算符优先分析（Operator Precedence Parsing, OPP）方法，通过构造优先关系表来明确不同运算符之间的优先级与结合性，从而在面对运算符重载与嵌套表达式时，有效消除语法二义性。词法分析模块基于多态与异常机制实现，对输入字符流进行逐词分类，识别数字（包括科学记数法）、布尔值、逻辑与关系运算符、函数标识符及分隔符等多类 Token，并针对非法输入提供清晰的错误反馈。语义处理部分则承担类型推导与兼容性判断任务，确保表达式在运行前完成类型验证，并在必要时抛出针对性的语义异常。

此外，整个实验采用模块化设计理念，通过分层封装、异常继承、多态扩展等手段，构建了结构清晰、功能完备的表达式解析与求值平台。配套的自动化测试工具覆盖了标准输入、嵌套结构与异常场景三大类别，用于全面验证系统的正确性与健壮性。实验回归测试表明，系统在各类输入下均能稳定解析并输出正确结果，错误响应机制也具备良好的精确性和可维护性。

综上所述，本实验通过构建表达式计算器，系统地集成了编译原理课程中的词法、语法与语义处理理论，并在实践中验证了算符优先分析方法的工程可行性，提供了一个面向表达式语言处理的完整技术实现路径，为后续语言工具开发与编译器设计奠定了扎实基础。

关键词：表达式求值，算符优先分析，语法归约，词法分析，语义处理，类型检查，自动化测试。

目录

基于表达式的计算器 实验报告

傅祉珏 21307210

中山大学计算机学院 广东广州 510006

1 项目概况

1.1 项目背景

本实验是编译原理课程中的一项综合性实验任务，具有较强的实践性与应用导向，围绕表达式语言的处理过程展开，涵盖了词法分析、语法分析与语义分析等编译前端的核心环节。实验通过设计并实现一个支持复杂表达式计算的图形化计算器，全面展示表达式从文本输入到语义求值的完整处理流程，具有较高的教学价值与工程应用参考意义。

本实验以算符优先分析（Operator Precedence Parsing, OPP）技术为核心，要求根据表达式语言的文法构造精确的算符优先关系表，并在此基础上实现语法解析与语义求值功能。支持的表达式类型包括算术运算、关系运算、逻辑运算、一元与三元运算、括号嵌套表达式以及预定义函数调用等，语法结构多样，语义规则严格，对表达式分析与求值提出了较高的设计要求。

本实验实现过程中强调面向对象设计方法的应用，涵盖类的封装、继承、多态与异常处理机制，配合统一的编码规范与模块化测试策略，促进规范化软件开发能力的训练。同时，实验配套提供软装置框架与自动化测试机制，支持表达式处理逻辑的快速集成与验证。通过本实验的实施，有助于建立完整的编译原理知识体系，并提升构建语言处理工具的综合能力。

1.2 实验需求

实验要求构建一个具备表达式解析与求值能力的计算器系统，能够处理包含数值与布尔类型常量的复合表达式。该系统需具备完整的词法分析、语法分析与语义处理能力，并支持嵌套括号、科学记数法、关系与逻辑运算符、预定义函数调用及三元运算等表达式特性，能够正确识别和处理表达式中的各种合法形式及异常情况。

实验需围绕一个明确的语法规范展开，要求基于指定的 BNF 文法对表达式结构进行解析，同时识别其中可能存在的二义性并予以说明或消除。在语法分析过程中，采用算符优先分析技术，构建准确的算符优先关系表，并对运算符重载情况（如一元取负与二元减法）进行区分和正确解析。语义处理部分需实现类型推导与类型兼容性检查，并支持在表达式中自动处理相关语义规则。

此外，实验还要求实现一个嵌入式的词法分析程序与语法/语义分析程序，并以模块化方式集成至计算器框架中。程序应具备良好的结构设计和错误处理机制，能在表达式输入非法时抛出精确的异常信息。实验结果需通过配套的测试工具进行验证，涵盖从简单到标准的各类测试用例，以保障系统的正确性与健壮性。

2 语法分析

2.1 BNF 分析

本实验使用的表达式语言基于一种形式化的 BNF（巴科斯-诺尔范式）文法进行定义，其核心结构围绕算术表达式（ArithExpr）和布尔表达式（BoolExpr）展开。BNF 定义涵盖了数值常量、小数、括号表达式、算术运算（加减乘除、幂运算）、一元取负运算、关系与逻辑表达式、三元运算符以及预定义函数调用等内容。文法结构清晰，具备良好的表达能力，能够覆盖大多数常见的表达式结构与语法组合。

尽管该文法采用了标准 BNF 形式进行定义，但由于表达式语言中存在运算符重载、多种优先级运算符以及复杂嵌套结构，文法本身在未进行进一步解析机制限制的情况下，理论上存在语法二义性。例如，表达式 `2 - 3 - 4` 可以被解析为 `(2 - 3) - 4` 或 `2 - (3 - 4)`，而 `2 ^ 3 ^ 2` 也可被理解为 `(2 ^ 3) ^ 2` 或 `2 ^ (3 ^ 2)`，这些不同的语法树将导致不同的求值结果，属于典型的语法歧义问题。此外，重载运算符如一元负号与二元减号“`-`”在表达式中的不同位置也会引发解析歧义，例如 `5 - -4` 中的两个“`-`”分别代表不同的操作含义。

为解决上述可能的二义性问题，实验规范中通过引入明确的“运算符优先级与结合性质”定义，对表达式的解析顺序进行严格约束。根据优先级设定，一元取负运算符优先级高于加减乘除运算符，幂运算符为右结合，三元运算符“`?:`”的优先级最低且为右结合，逻辑运算与关系运算的优先级则位于算术运算之后。通过该优先级体系，可以唯一确定每个表达式的语法结构，有效避免歧义。

以表达式 `2 - 3 - 4` 为例，根据加减运算的左结合性，将其解析为 `(2 - 3) - 4`；而表达式 `2 ^ 3 ^ 2` 则因幂运算为右结合，被解析为 `2 ^ (3 ^ 2)`。类似地，表达式 `5 - -4` 中的第一个“`-`”作为二元减法，第二个“`-`”因其紧跟运算数前被视为一元取负符号。三元运算符“`?:`”亦因右结合性而按照从右至左的方式构造语法结构。通过此类结合性和优先级规则的引导，语法分析器能够在存在运算符重载及嵌套结构的情况下准确解析表达式，避免歧义的产生。

2.2 OPP 构造

在基于表达式的计算器语法分析中，为避免传统 BNF 文法在存在多种运算符及复杂嵌套结构时带来的语法歧义，采用算符优先分析法（Operator Precedence Parsing, OPP）来引导表达式的分析过程。该方法以优先关系表为核心，结合预定义的运算符优先级与结合性，明确表达式中各运算符的移进（Shift）与归约（Reduce）策略，从而实现无二义性的语法树构建。

依据实验文档第 2.3.2 节所定义的优先级规则，表达式中的运算符被分为多个层级，括号拥有最高优先级，其次是预定义函数（如 sin、cos、max、min），之后依次为取负运算符（右结合）、幂运算符（右结合）、乘除运算、加减运算、关系运算、逻辑非（右结合）、逻辑与、逻辑或，最后是三元运算符 `?:`（右结合）。这种优先级设计为算符优先关系表的构造提供了基础框架。

在 `OPP` 类中，`TABLE` 数组即为算符优先关系表，其行列顺序覆盖了所有可能出现在表达式中的关键语法符号，包括括号、函数标识、各种运算符以及结束符 `$`。表中每个元素表示当前栈顶符号（行）与当前输入符号（列）之间的操作决策，其值可为：

- 1：表示移进（SHIFT），即当前符号优先级更高或具有右结合性；

- 2/3/4/5：表示应执行不同形式的归约（REDUCE），通常用于处理具体的语法产生式；
- 0：接受（ACCEPT），表示语法分析成功；
- 负值：表示特定类型的语法错误，如 -1 为三元运算符匹配错误，-2 为缺右括号等。

例如，在一元负号“-”与二元减法运算符的处理上，该表通过结合上下文符号的不同位置，结合优先级（取负高于减法）和结合性（取负为右结合），使其在分析过程中根据栈顶和输入符号的相对优先级作出正确的移进或归约决策，从而完成区分。此外，幂运算符 ^ 的右结合性亦通过将其与自身比较时表项设为移进（而非归约）来体现，从而正确处理如 2 ^ 3 ^ 2 之类的表达式，使其被解析为 2 ^ (3 ^ 2)。

表 1: 算符优先关系表（OPP 表）

	(	)	func	-	^	md	pm	cmp	!	&	|	?	:	,	$
(	1	5	1	1	1	1	1	1	1	1	1	1	-1	1	-2
)	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5
func	1	-3	-3	-3	-3	-3	-3	-3	-3	-3	-3	-3	-3	-3	-3
-	1	5	1	1	2	2	2	2	-6	-4	-4	2	2	2	2
^	1	5	1	1	1	3	3	3	-6	-4	-4	3	3	3	3
md	1	5	1	1	1	3	3	3	-6	-4	-4	3	3	3	3
pm	1	5	1	1	1	1	3	3	-6	-4	-4	3	3	3	3
cmp	1	5	1	1	1	1	1	-4	-6	3	3	3	-1	-3	3
!	1	5	-4	-4	-4	-4	-4	1	1	2	2	2	-1	-3	2
&	1	5	-4	-4	-4	-4	-4	1	1	3	3	3	-1	-3	3
|	1	5	-4	-4	-4	-4	-4	1	1	1	3	3	-1	-3	3
?	1	-1	1	1	1	1	1	1	1	1	1	1	-1	-1	-1
:	1	5	1	1	1	1	1	1	-1	-1	-1	1	-1	-1	4
,	1	5	1	1	1	1	1	-3	-3	-3	-3	1	-1	1	-3
$	1	-7	1	1	1	1	1	1	1	1	1	1	-1	-3	0

优先关系表中也专门对三元运算符“?:”进行了处理。由于其语法结构复杂、结合方式为右结合，且存在问号与冒号匹配的问题，该表在“?”对“:”的表项中设为移进，而在“:”对“$”或逗号的交叉项中设定特定归约编号或错误码，以精确管理三元运算的归约时机和异常情况。

函数调用亦在表中设有专门处理。例如，函数符号 func 在跟随符号为)、:、, 等语法符号时对应的表项设为函数错误（-3），以准确捕捉函数调用格式异常；同时，在函数符号后遇到表达式起始符如 (或数字时则触发移进。

3 程序设计及验证

3.1 词法分析器设计

词法分析器作为表达式计算系统的基础模块，负责将用户输入的字符流分解为一系列有意义的词法单元（Token），为后续的语法分析提供结构化的输入。在本实验中，词法分析器依据表达式语言的定义，实现了对数值常量、布尔值、运算符、函数标识符等多种类型的

Token 的识别与提取，并配合完善的异常处理机制，有效提升了系统的健壮性与错误定位能力。

词法分析任务需要识别的 Token 类型包括五大类：数值常量（如整数、小数、科学记数法表示的浮点数），布尔常量（`true`、`false`），运算符（如算术运算符 `+`、`-`、`*`、`/`、`^`，关系运算符 `>`、`<`、`>=`、`<=`、`==`、`!=`、`<>`，逻辑运算符 `&&`、`||`、`!`，以及条件运算符 `?` 和 `:`），函数（包括 `sin`、`cos`、`max`、`min`）以及输入结束符（`$`）。每类 Token 在分析过程中均有专门的处理方法进行识别。

在扫描到数字字符时，词法分析器会调用数字解析逻辑进行处理，该逻辑不仅能够识别整数和小数，还支持科学记数法格式（如 `3.14e-2`）。为确保格式合法，解析器会判断小数点及指数符号是否重复或位置错误，若发现异常，如连续出现两个小数点或缺失指数部分，则抛出非法小数字面量异常。布尔值的识别则通过判断字符串是否精确匹配“`true`”或“`false`”，若不符合规范，则会报告非法标识符错误。

运算符的识别使用了双字符匹配策略，能正确区分如 `>=`、`<=`、`!=`、`&&` 等多字符运算符，同时也能够对“`-`”符号的上下文进行判断，从而识别其为一元负号还是二元减法符号。在处理三元运算符 `?:` 或括号表达式时，词法分析器还会检测语法结构是否完整，若发现如“`?)`”或“`:,`”等格式错误，便立即抛出缺失操作数异常。对于函数识别，分析器会尝试提取三字符前缀，并判断其是否属于允许的函数名，未能匹配时将视为非法标识符处理。

词法分析过程中可能遇到非法字符、格式错误或未知标识符等问题。为此，系统定义了丰富的异常类型，如空表达式异常、非法小数异常、非法符号异常、非法标识符异常和通用词法异常等，这些异常覆盖了表达式输入中的大多数错误类型。分析器在扫描过程中会主动捕获这些异常并抛出，保证系统在面对不合法输入时能够做出精确响应，提升用户体验与系统鲁棒性。

3.2 语法分析器设计

语法分析器是表达式求值计算器的核心模块之一，主要功能是依据算符优先规则（OPP）对词法分析阶段产生的 Token 序列进行结构化分析，并在分析过程中同步执行语义处理。该模块不仅实现了对一元、二元和三元运算的归约规则，还包括函数调用、括号结构与类型兼容性检查等语义语法联合控制机制。

语法分析器基于经典的算符优先分析法构建，通过两个栈结构分别维护运算符栈（operators）与操作数栈（operands）。整个分析过程以一个优先关系表（OPP 表）为驱动核心，在每轮循环中，根据当前输入符号与栈顶符号之间的优先关系作出移进（Shift）或归约（Reduce）等动作决策。

在语法分析的每一步中，分析器会实时执行归约操作，将已解析的 Token 合并为新的 Token 并推入操作数栈。对于不同复杂度的表达式结构，系统定义了以下三类主要归约函数：

- 一元运算归约（reduce1）

用于处理如 `-x` 或 `!x` 的表达式。分析器弹出一个操作数与一个运算符，执行数值取负或布尔取反运算，处理完成后将结果重新压入操作数栈。如遇栈空等异常情况，将抛出 `MissingOperandException`。

- 二元运算归约（reduce2）

对标准的二元运算表达式如 `x + y`, `x * y`, `x && y`, `x > y` 等进行处理。运算执行前会首先检查操作数的类型是否匹配。若为算术运算，要求操作数为数值类型；若为逻辑运算，

则要求为布尔类型；若为关系运算，则进行数值比较并返回布尔值。若发生类型不匹配或除以零等错误，则分别抛出 `TypeMismatchedException` 与 `DividedByZeroException`。

- 三元运算归约（reduce3）

用于处理条件表达式 `cond ? a : b`，要求操作符依次为 `?` 和 `:`，且 `cond` 为布尔类型，`a` 与 `b` 为数值。若结构不符，将抛出 `TrinaryOperationException`，若类型不兼容，则抛出 `TypeMismatchedException`。

Algorithm 1 主控算法框架

Require: 初始化运算符栈与操作数栈
Ensure: 最优化向量，每次迭代的误差列表

初始化运算符栈与操作数栈
将起始符号 `$` 压入运算符栈
loop
    读取下一个 Token `curToken`
    获取运算符栈顶符号 `topToken`
    判断 `action = OPP[topToken][curToken]`
    if `action` 为 `SHIFT` **then**
        将 `curToken` 压入运算符栈
    end if
    if `action` 为 `REDUCE` **then**
        根据 OPP 表中的编号调用相应的归约函数
    end if
    if `action` 为 `BRACETREDUCE` **then**
        处理括号或函数调用归约逻辑
    end if
    if `action` 为 `ACCEPT` **then**
        返回最终计算结果
    end if
    if `action` 为 `ERROR` **then**
        抛出对应的语法或语义异常
    end if
end loop

此外，函数调用的语法归约（reduce0）也在括号归约中统一处理。当运算符栈中出现函数标识符与括号包裹的参数列表时，分析器将识别并统计参数数量，并根据函数名称执行相应计算。目前系统支持 `sin`、`cos`（一元），`max`、`min`（多元）四种预定义函数。参数不足或函数名称非法将分别触发 `MissingOperandException` 或 `FunctionCallException`。

语义分析的核心是类型兼容性检查与类型推导。所有运算都基于 Token 的类型信息进行严格检查，布尔型和数值型在运算中不可混用，比较操作只接受数值型参与，而逻辑运算仅适用于布尔型。类型推导则基于运算符语义，例如加减乘除的结果为数值，逻辑运算结果为布尔值，三元表达式根据条件返回两个数值之一。

最终，当分析器识别到输入结束符 `$` 且操作数栈中只剩下一个 `Decimal` 类型的结果时，视为语法分析成功，并返回表达式的计算结果。若结果类型不正确或仍有未处理的符号，将

抛出 `TypeMismatchedException` 或 `MissingOperatorException`。

3.3 实验验证

为了验证表达式计算器的语法分析与语义处理功能是否正确实现，系统构建了一套覆盖面广、针对性强的测试用例集，涵盖了标准计算、逻辑推理、函数调用以及各类错误检测场景。测试采用回归验证的方式运行，通过自动化比对预期输出与实际结果，确保程序在不同输入下均能给出正确响应。

功能性测试部分包括基本算术表达式、多层嵌套的科学记数法计算、布尔逻辑组合、条件运算符选择及函数嵌套调用等用例。其中，示例 C001 到 C003 验证了四则运算、乘方及科学计数的正确性，如表达式 `2.25E+2 - (55.5 + 4 * (10 / 2) ^ 2)` 成功输出 `69.5`，确认了语法解析器对科学计数、小数精度与优先级规则的处理能力。C004 使用 `&` 逻辑与运算符结合三元表达式 `?:` 结构，验证了条件表达式在语义层的布尔控制逻辑。C005 和 C006 则聚焦于函数调用，测试了 `sin`、`cos`、`max`、`min` 等预定义函数的组合调用与嵌套表达，输出结果高度精确，说明函数参数提取与栈结构归约逻辑工作正常。

异常检测用例从 E001 到 E010 共设计了十类不同语法或语义错误场景，用于验证系统的健壮性与异常响应机制。包括括号缺失（E001、E002）、缺少操作符或操作数（E003、E004）、除零错误（E005）、类型不匹配（E006）、非法小数（E007）、未定义函数调用（E008）、函数参数数量错误（E009、E010）等情况。每一类错误均成功触发对应异常类型，如 `MissingLeftParenthesisException`、`DividedByZeroException`、`TypeMismatchedException` 等，表明异常设计与抛出机制运行可靠。

所有测试均使用自动化验证工具运行，共执行 16 个测试用例，覆盖率 100%。统计结果显示：通过用例数为 16，失败用例数与警告用例数均为 0，整体通过率达 100.0%。在另一组精简测试中，选取典型的正确与错误表达式共 8 个用例，也全部成功通过，进一步说明系统在不同规模和复杂度输入下都具备稳定的处理能力。

4 总结与展望

4.1 实验总结

本实验通过构建一个具备完整表达式求值能力的计算器系统，综合验证了词法分析、语法分析和语义处理各模块的功能实现情况。实验系统支持包括科学计数法、小数、逻辑与关系表达式、三元条件结构以及预定义函数在内的多种表达式形式，具备较高的语法覆盖率和语义准确性。回归测试结果表明，该系统能够在多种复杂表达式结构下保持稳定运行，并对各种语法与语义异常作出明确诊断，具备良好的健壮性与可维护性。

在测试过程中，所有功能性用例均按照预期计算出准确结果，表明算符优先关系表设计合理，语法归约逻辑清晰，语义处理流程完善。系统对括号缺失、操作符与操作数缺失、除零、类型不匹配、非法标识符、函数调用错误等常见输入异常的响应准确，异常信息定位精确，反映出异常处理机制设计的完整性。整体回归测试通过率达 100%，验证了系统整体设计的正确性和鲁棒性。

4.2 未来展望

在当前实验成果基础上，表达式计算器尚有多个潜在扩展方向可供深入研究。首先，可进一步扩展表达式语言的语义能力，如引入变量声明与作用域支持，实现带有环境上下文的

表达式求值机制，从而向解释器的方向演进。其次，可引入用户自定义函数机制，支持函数定义与嵌套调用，以提升语言的表达力和灵活性。同时，考虑引入更多数据类型（如字符串、数组、布尔向量等）及其运算支持，有助于构建更通用的表达式求值框架。

在语法分析方面，未来可探索使用更高阶的语法分析方法（如 LALR、递归下降等）替代算符优先分析，从而提升对非运算主导语法结构的适应能力。此外，若结合图形界面模块（GUI）或基于 Web 的表达式交互平台，将进一步增强系统的可视化表达与交互体验。

从编译原理教学角度看，该项目为理解编译前端各阶段处理流程提供了清晰、完整的工程实践模型。通过面向对象的模块划分、异常机制设计和统一测试驱动开发，实验不仅实现了对理论知识的落地应用，也锻炼了系统性问题建模与解决能力，为进一步理解语言处理系统（如解释器、编译器、静态分析器等）的构建方法奠定了坚实基础。该实验具备良好的教学推广价值及工程演进潜力。

参考文献

[1] 沐言科技 李兴华. Java 编程 从入门到实践 [M]. 第 1 版. 安徽: 中国水利水电出版社, 2021.

[2] Aho A V, SLam M, Sethi R, etal. 编译原理 [M]. 赵建华 等译. 第 2 版. 北京: 机械工业出版社, 2009.

[3] LI W J. Lecture05: LL Parsing[PPT][Z]. Sun Yat-sen University, 2022.

[4] LI W J. Lecture06: OPP[PPT][Z]. Sun Yat-sen University, 2022.