

2024-2025 学年春课程《仓颉语言程序设计》大作业

提交截止日期：2025 年 5 月 18 日 23:59

一、 作业概述

1.1 任务概述

使用仓颉语言开发一个运行在服务端的仓颉代码工具，实现对仓颉程序的简单语法和语义分析，同时使用前端语言设计一个简单的图形界面，在浏览器实现对代码工具的调用。

1.2 作业目标

通过该大作业掌握使用仓颉语言进行综合应用开发的方法，熟悉仓颉语言的并发编程、网络编程、异常处理等能力，学会使用仓颉标准库中的语法解析器分析和修改仓颉源代码，了解网站开发方法和部分编译原理知识。同时，在合作完成大作业的过程中培养团队协作意识、组织能力与表达能力。

二、 作业要求

2.1 任务背景

现代集成开发环境 (IDE) 在代码编辑和分析中扮演着至关重要的角色。它们不仅提供代码高亮、自动补全等基础功能，还通过静态代码分析技术对代码进行深入检查，以发现潜在的错误、提出优化建议或执行用户的优化需求。这些功能的实现很大程度上依赖于抽象语法树 (Abstract Syntax Tree, AST) 这一数据结构。

抽象语法树是源代码的树状表示，它将代码的结构分解为节点和分支，每个节点代表代码中的一个语法结构（如表达式、语句、函数声明等）。通过解析源代码生成 AST，IDE 能够对代码进行多种静态分析，例如：

- **代码格式化：**根据 AST 的结构，IDE 可以自动调整代码的缩进、换行等格式，使其符合编码规范。
- **代码重构：**通过分析 AST，IDE 可以识别代码中的重复模式或冗余代码，并提供重构建议，如变量重命名、函数提取等。
- **错误检测：**AST 可以帮助 IDE 识别代码中的语法错误、类型不匹配等问题，并在编辑器中实时提示。
- **代码优化：**基于 AST，IDE 可以执行常量折叠、死代码消除等优化操作，提升代码的执行效率。

- **文档生成：**通过分析函数和类的 AST 节点，IDE 可以基于 LLM 调用自动生成代码文档，帮助开发者更好地理解代码功能。

仓颉语言作为一种新兴的编程语言，其标准库提供了强大的抽象语法树解析工具 (`std.ast` 包)，使得开发者能够轻松地对仓颉代码进行静态分析。本次大作业的核心任务就是利用仓颉语言的 AST 解析能力，实现一个简单的代码分析工具，模拟现代 IDE 中的部分功能。通过本次作业，你将深入理解抽象语法树在代码分析中的应用，并掌握如何使用仓颉语言进行代码解析、重构和优化。这些技能不仅有助于更好地理解编译原理，还能提升在实际开发中的代码质量与效率。

基于上述背景，本次大作业要求开发一个仓颉代码工具，利用抽象语法树对仓颉代码进行以下静态分析操作：

- **生成代码签名：**提取代码中的函数和类签名，生成简洁的接口文档。
- **变量重命名：**根据指定的作用域，对代码中的变量进行安全的重命名操作，确保代码的语义不变。
- **生成文档：**通过调用大语言模型 (LLM) API，为指定的函数或类生成解释性文档，并以块注释的形式插入到代码中。
- **常量折叠：**对代码中的常量表达式进行计算和折叠，优化代码的执行效率。

这些功能模拟了现代 IDE 中常见的代码分析和优化操作，旨在帮助理解抽象语法树在代码处理中的实际应用。

2.2 功能需求

2.2.1 后端功能需求

1. **功能接口。**创建一个仓颉项目，在源代码根目录 (`src`) 下创建 `CJCodeTool` 类（类名必须使用这个）。该类结构如图1所示。你需要编写代码实现该类中的四个静态函数。每一个静态函数代表一个基础的代码分析/修改功能，它们都接受一个代表单文件代码文本的字符串类型参数 `code`，经过一系列处理之后返回处理结果字符串¹。除了这四种功能以外，你可以至多额外添加一种自定义功能，该功能的复杂度和实现效果会被纳入评分考量。

- (a) `generateCodeSignature` 函数。该函数的功能是生成代码文件的签名。此处的“签名”包含“函数签名”和“类（接口）签名”²，具体到仓颉语言，可以理解为定义接口时使用的无实现成员函数。而类签名则包含了类的声明语句以及它内部成员（包括属性，变量，函数）的签名。一个具体的例子如图2所示。

具体来说，对于变量、属性、函数、类和接口，签名按次序包含以下内容：

- i. 修饰符（如 `public`）。注意：修饰符可能不存在。
- ii. 关键字（如 `func`）。
- iii. 标识符，即命名。
- iv. （对于类和接口）以 `<`：开头的父接口/类序列，使用 `&` 连接。注意：该部分可能不存在。

¹代码文本可在北航云盘获取

²函数签名的定义可以参考维基百科

```

class CJCodeTool {
    static func generateCodeSignature(code: String): String {
        """
        Your implementation
        """
        return ""
    }
    static func refactorVariable(code: String, path: String, oldName: String, newName: String): String {
        """
        Your implementation
        """
        return ""
    }
    static func generateDocument(code: String, path: String): String {
        """
        Your implementation
        """
        return ""
    }
    static func foldConstant(code: String): String {
        """
        Your implementation
        """
        return ""
    }
}

```

图 1: 后端功能接口

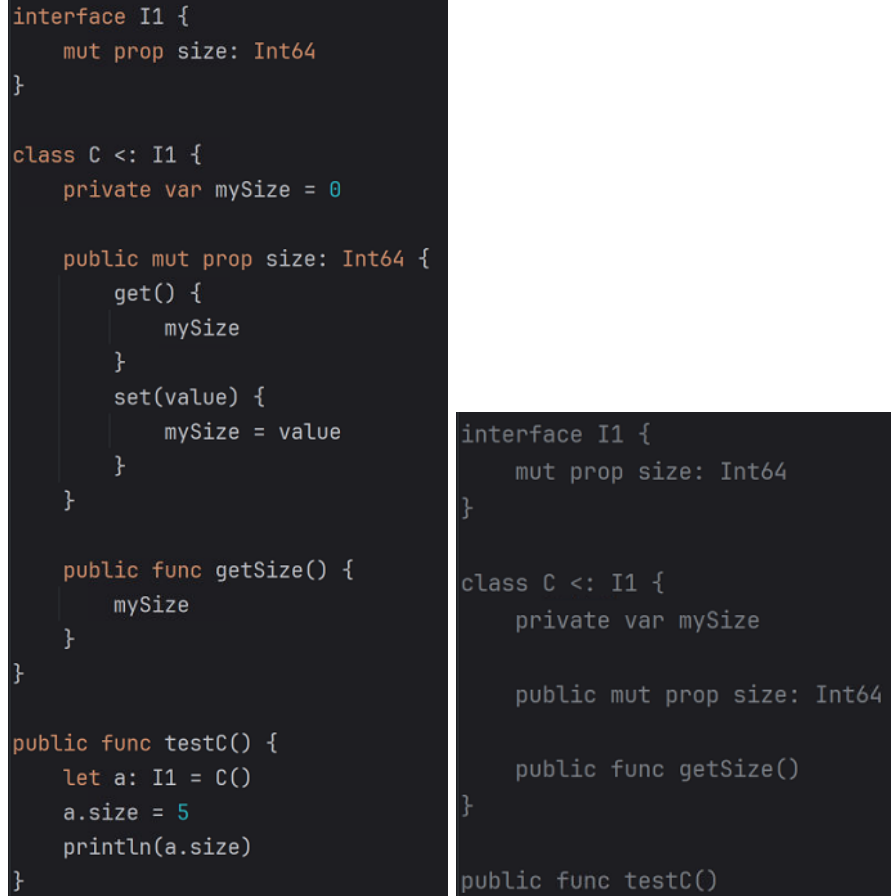
- v. (对于函数) 参数列表。该部分和源代码保持一致即可。
- vi. (对于变量、属性和函数) 以冒号开头的返回类型。**如果源代码中未显式声明返回类型, 则签名中同样不用包含。**
- vii. (对于类和接口) 内部声明的签名。

数据限制:

- 文件中的一级声明 (即最外层的声明) 只包含函数 (FuncDecl 类)、类 (ClassDecl 类) 和接口 (InterfaceDecl 类) 声明; 类和接口内部只包含变量 (VarDecl 类)、属性 (PropDecl 类) 和函数 (FuncDecl 类) 声明; 函数内部只包含变量声明; 不将形参声明 (FuncParam 类) 视为声明, 也就是不必将其纳入是否满足数据限制的考虑。
- 不存在泛型类、接口和函数声明。
- 不存在操作符重载函数声明。

输出要求:

- 按顺序输出文件中所有函数、类和接口的签名 (即图2右图的格式)。**对于不符合数据限制的输入, 返回字符串 'ILLEGAL INPUT'。**类和接口中的函数签名无需额外单独输出。
- 单次缩进为四个空格, 每一个签名之间有一个空行。
- 词法单元之间的空格规范不做要求 (但要保证语法正确性)。



```

interface I1 {
    mut prop size: Int64
}

class C <: I1 {
    private var mySize = 0

    public mut prop size: Int64 {
        get() {
            mySize
        }
        set(value) {
            mySize = value
        }
    }

    public func getSize() {
        mySize
    }
}

public func testC() {
    let a: I1 = C()
    a.size = 5
    println(a.size)
}

```

```

interface I1 {
    mut prop size: Int64
}

class C <: I1 {
    private var mySize

    public mut prop size: Int64

    public func getSize()
}

public func testC()

```

图 2: 文件签名示例。左图是原文件内容，右图是提取的签名

- (b) `refactorVariable` 函数。该函数的功能是通过 `path` 参数指定代码中的一个类或者函数，然后将其作用域内的某个标识符 (`oldName`) 全部替换为另一个标识符 (`newName`)。对于函数，作用域是**函数签名和函数体这个整体**，对于类也完全类似。

输入规范:

- `path`: 形如 `'' 函数名 ''`、`'' 类名 ''`、`'' 类名. 函数名 ''` 的格式。第一种是最外层的函数（不属于任何类的函数）。
- 不存在内层变量覆盖、重载等影响重构作用域和路径搜索的特殊情况。
- 标识符都是合法的**普通标识符**，并且不会和内置类型标识符重名。

数据限制:

- 使用仓颉 `std.ast` 包解析之后的语法树只存在这些类型的节点: `AssignExpr`, `BinaryExpr`, `Block`, `Body`, `ClassDecl`, `Decl`, `Expr`, `ForInExpr`, `FuncDecl`, `FuncParam`, `IfExpr`, `ImportContent`, `ImportList`, `LitConstExpr`, `Modifier`, `Node`, `PackageHeader`, `ParenExpr`, `Pattern`, `PrimitiveType`, `RangeExpr`, `RefExpr`, `RefType`, `ReturnExpr`, `Token`, `Tokens`, `TypeNode`, `UnaryExpr`, `VarDecl`, `VarPattern`。

- `path` 只存在以下三种情况：'' 函数名''、'' 类名''、'' 类名. 函数名''。当且仅当 `path` 在源代码中不存在时，视为不满足数据限制。

输出要求：

- 输出重构后的完整代码。对于不符合数据限制的输入，返回字符串''ILLEGAL INPUT''。
 - 词法单元之间的空格和换行规范不做要求（但要保证语法正确性）。
- (c) `generateDocument` 函数。该函数的功能是通过 `path` 参数指定代码中的一个类或者函数，然后为其生成解释文档（以块注释的形式展示），用于解释该类或者函数的功能。一个例子如图3所示。

```
class C <: I1 {
    /*
    this is a class
    */
    private var mySize = 0

    public mut prop size: Int64 {
        get() {
            mySize
        }
        set(value) {
            mySize = value
        }
    }

    public func getSize() {
        /*
        this is a function
        */
        mySize
    }
}
```

图 3: 文档示例

输入规范：

- `path`：形如'' 函数名''、'' 类名''、'' 类名. 函数名'' 的格式。第一种是最外层的函数（不属于任何类的函数）。

数据限制：

- `path` 只存在以下三种情况：'' 函数名''、'' 类名''、'' 类名. 函数名''。当且仅当 `path` 在源代码中不存在时，视为不满足数据限制。

输出要求：

- 输出生成文档后的完整代码。对于不符合数据限制的输入，返回字符串''ILLEGAL INPUT''。
- 除了插入的文档所在的行，其他行的内容与格式应当与源文件完全相同。

- (d) `foldConstant` 函数。该函数的功能是将代码中可直接计算的常量进行折叠³，不考虑常量传播。例如，如果代码中存在一句 `'let a = 3 + 2'`，则折叠之后这句代码就变为 `'let a = 5'`。

输入规范：

- 代码中的常量运算一定合法。
- 不存在不同字面量类型之间的类型转换。
- 不存在计算溢出情况。

数据限制：

- 使用仓颉 `std.ast` 包解析之后的语法树只存在这些类型的节点：AssignExpr, BinaryExpr, Block, Body, ClassDecl, Decl, Expr, ForInExpr, FuncDecl, FuncParam, IfExpr, ImportContent, ImportList, LitConstExpr, Modifier, Node, PackageHeader, ParenExpr, Pattern, PrimitiveType, RangeExpr, RefExpr, RefType, ReturnExpr, Token, Tokens, TypeNode, UnaryExpr, VarDecl, VarPattern（和上面一样）。
- 代码中的字面量只包含整数类型字面量和浮点数类型字面量，且字面量没有前后缀。换言之，字面量的类型只包含 `Float64` 和 `Int64` 两种。
- 常量之间的运算只包含加/减/乘/除/模和取负六种运算。

输出要求：

- 输出重构后的完整代码。对于不符合数据限制的输入，返回字符串 `'ILLEGAL INPUT'`。
- 常量必须被完全折叠，但对于类似于两个常量之间间隔一个变量的情况（这种情况下，二元表达式中存在一个操作符是变量），则不继续向上折叠。
- 整数类型字面量被折叠之后仍然是整数类型字面量，浮点数类型字面量被折叠之后仍然是浮点数类型字面量。浮点数类型字面量的小数保留位数不做特殊处理。
- 词法单元之间的空格和换行规范不做要求（但要保证语法正确性）。
- 不考虑常量传播。

2. **HTTP 服务器**。使用仓颉语言的网络编程接口自行设计一个 HTTP 服务器，根据请求路径和报文数据调用相应的功能函数，然后将返回值添加到响应报文中进行响应。

2.2.2 前端功能需求

自行设计并用前端编程语言（html, css, js 等）或前端框架（vue, react 等）编写一个网页，实现对后端各功能接口的调用，页面操作逻辑形式不限。一种简单的实现思路是：网页仅包含一个文本输入框和一个提交按钮，将调用的功能接口和除 `code` 以外参数写在第一行，剩余行输入待处理的代码。点击按钮之后，将文本框内容提交至后端进行处理，最后将返回内容以 html 段落的形式展示出来。本大作业重点在于仓颉语言的综合运用，故无需在前端耗费过多工作量。

³常量折叠的定义可以参考维基百科

2.3 实现方式限制

- 对于后端的所有代码处理函数，请使用仓颉的 `std.ast` 标准库进行处理。正则表达式或序列处理很难完全实现预期功能。
- 对于 `generateDocument` 函数中的文档生成，必须调用 LLM API 来完成，建议使用DeepSeek⁴。对于文档的插入，由于仓颉语法树中不存在注释类型节点，建议通过语法树获取类/函数签名所在行列数之后直接使用字符串方法进行插入。
- 请将主函数单独放在源代码根目录（src）下的 `main.cj` 文件中。

2.4 组队要求

- 人数要求：2-3 人一组
- 组队登记：腾讯文档

三、 提交与展示要求

3.1 展示要求

- 展示时间：最后一周（第 12 周）周一课上。
- 展示方式：PPT 课上展示（所有组员都要发言）。
- 展示内容：至少应包含以下内容：
 1. 各功能（包含自定义功能，若有）实现方法
 2. 测试的各种输入情况（无需展示具体测试输入）
 3. 测试结果（需包含网页截图）

3.2 提交要求

- 提交内容：
 1. 代码文件：将源代码根目录（src）打包为 zip 格式，命名为**小组号 _ 大作业代码.zip**。
 2. PPT：命名为**小组号 _ 大作业 PPT.pptx**。
 3. 大作业报告：对大作业的完成进行简要的总结。该报告至少应包含以下内容：
 - (a) 组内分工及工作量比例（以百分比计）
 - (b) 各功能（包含自定义功能，若有）实现方法
 - (c) 测试的各种输入情况（需包含具体测试输入）
 - (d) 测试结果（需包含网页截图）
 - (e) 总结与体会

⁴代码实现可参考GitCode（需要注册后查看）

该文件命名为**小组号 _ 大作业报告.pdf**。

然后将代码压缩包与报告一起打包为**小组号 _ 大作业.zip**。

- **提交方式：**上传到北航云盘。
- **截止时间：**2025 年 5 月 18 日 23:59

四、 其他事项

4.1 重要提示

1. 可以使用 `cangjieLex` 函数将 `String` 转换为 `Tokens` 类型。
2. 由于目前版本的仓颉标准库在 Windows 平台上存在缺陷，在 HTTP 服务器的处理函数中直接调用 `std.ast` 库中的函数时会报 `NoneValueException` 错误，导致代码解析逻辑无法正常执行。为了规避这个缺陷，可以将代码解析类（`CJCodeTool`）中的函数使用一个子线程执行，然后再通过获取 `future.get()` 函数获取子线程的返回值。

4.2 参考资料

- **仓颉语言官方文档：**华为开发者文档或仓颉语言文档。
- **仓颉语言学习资源。**
- **抽象语法树：**维基百科。

4.3 问题解答

- 若对该文档本身有任何疑问，可联系助教咨询。
- 若在编写或运行后端仓颉代码的过程中遇到技术问题，并且查阅资料无法解决，可联系助教咨询。
- 其他问题请自行使用参考资料、搜索引擎以及大模型等工具解决。