**编译原理课程设计任务书**

**——类C编译器设计与实现**

****

**学 号 \_\_\_\_\_\_2053635\_\_\_\_\_\_\_\_**

**姓 名 \_\_\_\_\_\_\_\_任柯睿\_\_\_\_\_\_\_\_\_**

**专 业 \_\_计算机科学与技术\_\_\_\_**

**授课老师 \_\_\_\_\_\_\_卫志华 \_\_\_\_\_\_\_\_**

目 录

[1 需求分析 2](#_Toc16176)

[1.1 程序任务输入及其范围 2](#_Toc5219)

[1.2 输出形式 4](#_Toc4562)

[1.3 程序功能 5](#_Toc13663)

[1.3.1 前端功能 5](#_Toc32350)

[1.3.2 后端功能 5](#_Toc13018)

[1.4 测试数据 6](#_Toc10157)

[1.4.1 正确测试样例 6](#_Toc4177)

[1.4.2 错误测试样例 8](#_Toc18241)

[2 概要分析 12](#_Toc15689)

[2.1 任务的分解 12](#_Toc595)

[2.1 编译器实现任务的分解 12](#_Toc5826)

[2.1 前后端设计实现任务的分解 13](#_Toc25602)

[2.2 数据类型的定义 13](#_Toc32010)

[2.3 主程序流程 14](#_Toc3657)

[2.4 模块间的调用关系 15](#_Toc26995)

[3 详细分析 18](#_Toc32623)

[3.1 词法分析部分 18](#_Toc2685)

[3.2 文法部分 19](#_Toc30720)

[3.3 语法分析部分 22](#_Toc25190)

[3.4 语义分析部分 22](#_Toc11670)

[3.5 优化部分 23](#_Toc844)

[3.6 目标代码生成部分 24](#_Toc6404)

[3.7 可视化设计部分 25](#_Toc7488)

[4 调试分析 28](#_Toc25235)

[4.1 数据测试 28](#_Toc30196)

[4.2 时间复杂度分析 32](#_Toc17782)

[5 用户使用说明 34](#_Toc24306)

[5.1 整体功能说明 34](#_Toc19366)

[5.2 细节功能说明 34](#_Toc9857)

[6 概要分析 40](#_Toc18551)

[6.1 遇到的问题及解决方法 40](#_Toc16801)

[6.2 思考 41](#_Toc13056)

[6.3 对课程的认识 42](#_Toc19131)

[参考文献 43](#_Toc30901)

# 1 需求分析

## 1.1 程序任务输入及其范围

本程序的输入包含以下几项：

1. 类c语言的源程序，源程序需要符合c语言的规范，并且不支持通过include方式引入其他c++库，也不允许使用define语句进行宏定义（否则编译器会将其视为错误输入）。支持的操作包括声明与定义全局变量以及数组、声明函数、调用函数等，同时支持if-else、for、while等基本流程控制语句。

(2) 输入文法内容，该文法类似于c语言语法，其中包括赋值、选择语句，以及对数组判定以及函数调用。在grammar文件中，使用 := 表示推导，其中左边为非终结符，右边为终结符。下面是我设计的类C编译器所支持文法的生成式，可以在项目路径下的grammar.txt中找到，此文件路径在源代码中规定，不支持自定义修改。

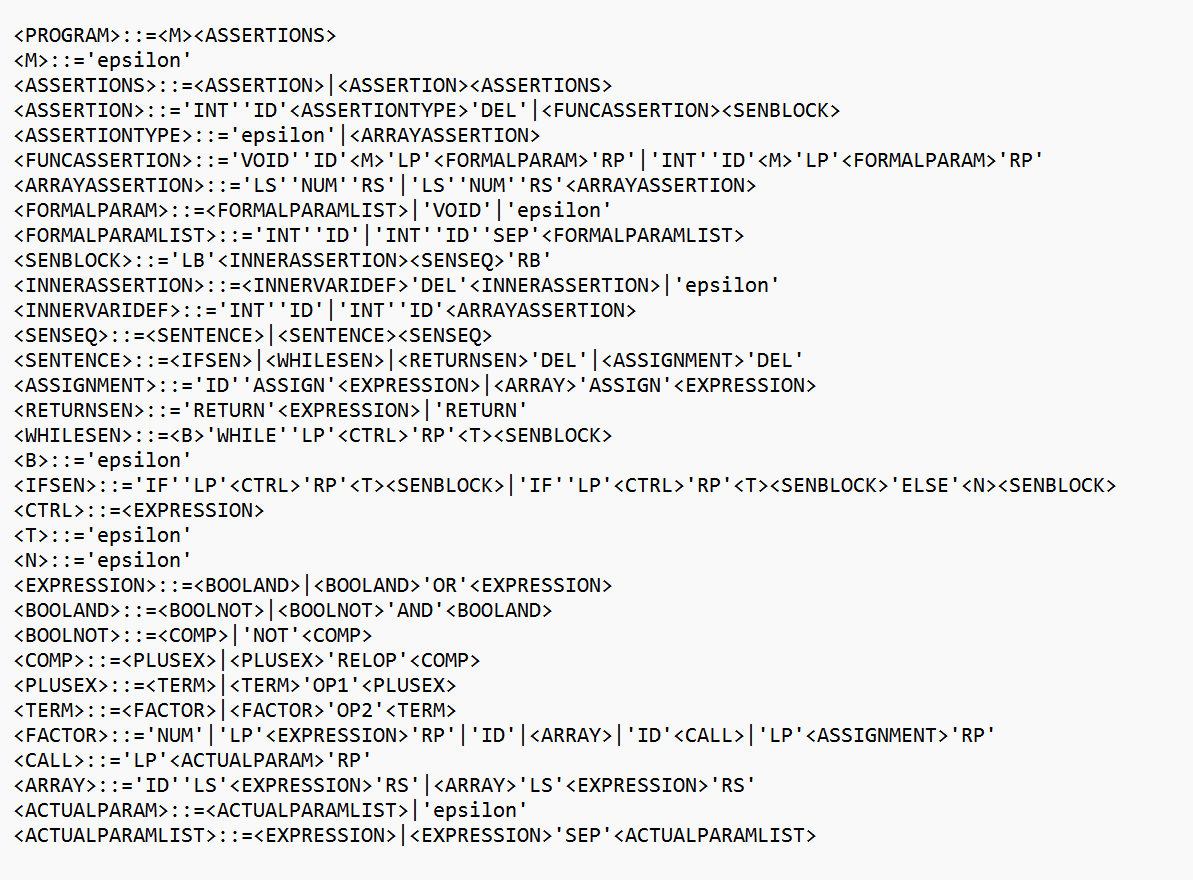


图1-1. 类C编译器文法展示

由于我使用了springboot+myBaits-plus实现后端编译以及数据库功能，同时前端使用vue实现交互，因此与一般QT程序不同，我设计的网页端IDE可以实时修改并保存对源代码的修改，同时所有编写的类C语言程序以及对应.s的汇编代码都会存储到数据库中，实现数据的离线保

存。

在点击前端的编译按钮后，前端使用Fetch API将类C语言代码通过http协议传向后端。后端在接收前端给定输入并从本地获取grammar文件。在接受以上两项输入之后，后端会进行以下五步操作：

1. 词法分析（Lexical Analysis）：词法分析是编译过程的第一步，词法分析器将源代码分解为一个个称为词法单元的最小语义单元。词法单元可以是关键字、标识符、运算符、常量或分隔符等。词法分析器读取源代码字符流，将其转换为词法单元序列，并去除不必要的空格和注释。在完成词法分析后，后端将分析结果传给前端，并在前端的词法分析展示界面显示符号表以及代码的词法单元序列，
2. 语法分析（Syntax Analysis）： 语法分析是编译过程的第二步，语法分析器首先读取grammer文件，读取文法表达式，并在此基础上建立每个元素的FIRST集以及FOLLOW集，之后通过LR(1)的分析方法，构建出LR(1)项目集簇并绘制LR(1)分析表，最后获取词法分析器提供的词法单元，在LR(1)分析表的帮助下，完成推导，计算语法树并保存推导过程。在完成语法分析后，后端将分析结果传给前端，并在前端的语法分析展示界面显示LR(1)项目集簇和推导过程。同时在前端的语法树展示界面绘制出语法树。
3. 语义分析（Semantic Analysis）： 语义分析是编译过程的第三步。它对语法树进行进一步的分析，并检查源代码的语义正确性。语义分析器执行类型检查、符号表管理和语义规则验证等操作。它确保变量和表达式的类型匹配、函数调用正确、变量声明和作用域等语义问题得到处理。在完成语义分析后，可以获取到中间代码，其以四元式的形式存储。
4. 优化（Optimization）： 优化是编译过程的一个重要阶段。在优化阶段，编译器尝试通过修改中间表示形式来改进程序的性能和效率。优化可以针对不同的方面进行，例如时间优化和空间优化。优化技术包括常量折叠、死代码消除、循环展开、代码内联、寄存器分配等。通过优化，编译器可以生成更有效的目标代码，提高程序的执行速度和资源利用率。在完成优化后，后端将分析结果传给前端，并在前端的语义分析和优化界面显示优化前和优化有的中间代码。
5. 目标代码生成（Code Generation）： 目标代码生成是编译过程的最后一步。在这个阶段，编译器将优化后的中间表示形式转换为目标机器的MIPS汇编代码。在完成目标代码生成时，后端将生成的目标代码传给前端，并在前端展示编译执行过程以及最后获得的目标代码。最后，将目标代码保存到被编译的代码同文件夹下的位置。

## 1.2 输出形式

假设输入的源文件符合grammer文件的文法要求，并且在上述的五个阶段没有出现错误，那么在前端页面的右下展示运行日志、词法分析、语法分析、语法树等六个展示界面，其中可以通过点击切换不同页面，展示内容包括符号表、词法单元序列、语法树、中间代码、目标代码等不同阶段的输出；但是如果输入的源文件不符合grammer文件的文法要求，那么后端会在执行编译程序的时候抛出问题，并在前端的运行日志页面输出问题。

## 1.3 程序功能

### 1.3.1 前端功能

前端从后端获取数据并展示到网页上，其主要功能如下（括号中为功能对应快捷键）：

1. 新建文件（CTRL+1）：新建空白文件，并将文件id，文件名，父文件夹等信息传向后端数据库。
2. 新建文件夹（CTRL+2）：新建空文件夹，并将文件夹名，父文件夹等信息传向后端数据库。
3. 保存文件（CTRL+S）：保存选中文件的内容，并将文件id，文件名，文件内容等信息传向后端数据库进行修改。
4. 上传文件（CTRL+U）：从本地上传文件到网页端，并将新上传的文件对应的文件id，文件名，父文件夹等信息传向后端数据库。
5. 下载文件（CTRL+D）：下载选中文件到本地。
6. 剪切内容（CTRL+X）：剪切编辑区域中选中文本到clipboard。
7. 复制内容（CTRL+C）：复制编辑区域中选中文本到clipboard。
8. 粘贴内容（CTRL+V）：从clipboard中获取文本粘贴到编辑区域中对应位置。
9. 搜索内容（CTRL+F）：搜索编译区域中与给定文本相同的行以及对应内容，其中搜索文本两侧置%用于突出。
10. 替换内容（CTRL+R）：搜索编译区域中与给定文本相同的行以及对应内容，并替换为对应替换内容，其中搜索文本两侧置%用于突出。
11. 编译代码（CTRL+G）：编译选中文件，将文件中的内容传给后端，之后后端编译类C代码，并将编译过程中的中间结果传给前端，之后展示在网页右下的展示界面。
12. 帮助文档（CTRL+H）：点击帮助文档后在编辑区显示帮助文档，其中包含网页的功能以及使用注意事项。
13. 修改文件名称：在文件栏区域右键打开弹窗，之后选中修改文件，可以修改选中文件的名称，之后将文件id，文件新名称，文件内容等信息传向后端数据库进行修改。
14. 删除文件名称：在文件栏区域右键打开弹窗，之后选中删除文件，可以删除选中文件，之后将文件id，文件新名称，文件内容等信息传向后端数据库将对应数据删除。
15. TAB键（Tab）：使用Tab快捷键，在编译区域会添加一个缩进。

### 1.3.2 后端功能

1. 获取文件树：在初始化的时候将数据库的所有文件的信息传输到前端，之后在前端就可

以展示出对应的文件树。

1. 获取文件表项数量：获取后端数据库中数据表项的数量。
2. 插入或修改文件表项：从前端获取文件信息，如果后端数据库中不包含文件id，那么就向数据库中插入一条新文件表项，否则在原有表项上进行修改。
3. 删除文件表项：从前端获取文件信息，删除对应文件id的数据库表项。
4. 文件改名：从前端获取文件信息，更新对应文件id的label属性。
5. 保存文件：从前端获取文件信息，更新对应文件id的fileContent属性。
6. 编译：从前端获取文件的文件内容信息，对输入的代码进行词法分析、语法分析、语义分析、优化和目标代码生成，之后将生成的目标代码传回前端；如果在编译的过程中出错则将错误信息传向前端。
7. 保存结果：从前端获取生成的汇编文件名称和汇编代码，将信息插入到数据库中，在被编译代码同目录下新建汇编代码文件 (“./code/1.cpp”编译后的汇编代码会保存到”./code/1.s”)。

## 1.4 测试数据

测试数据分别包含多份符合语法规范并带有函数调用和数组的源代码，以及多份包含各类语法错误（包含基本的语法错误以及较为高级的语义错误）的的源代码。

### 1.4.1 正确测试样例

(1) right1.cpp：不包含数组，最简单的测试样例。

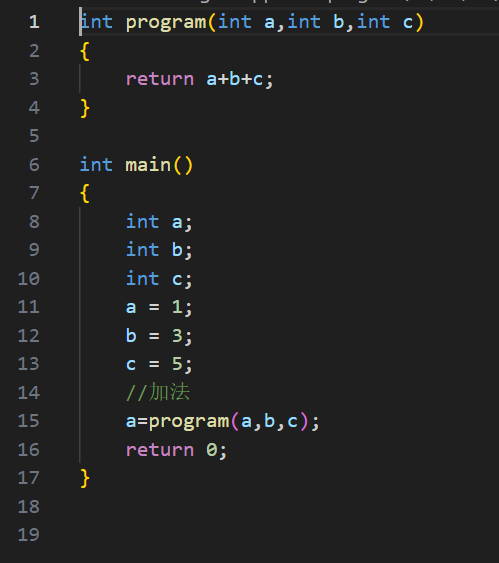


图1-2. right1.cpp测试样例



图1-3. right1.cpp前端输出结果

(2) right2.cpp：包含数组，函数调用，较困难的测试样例。

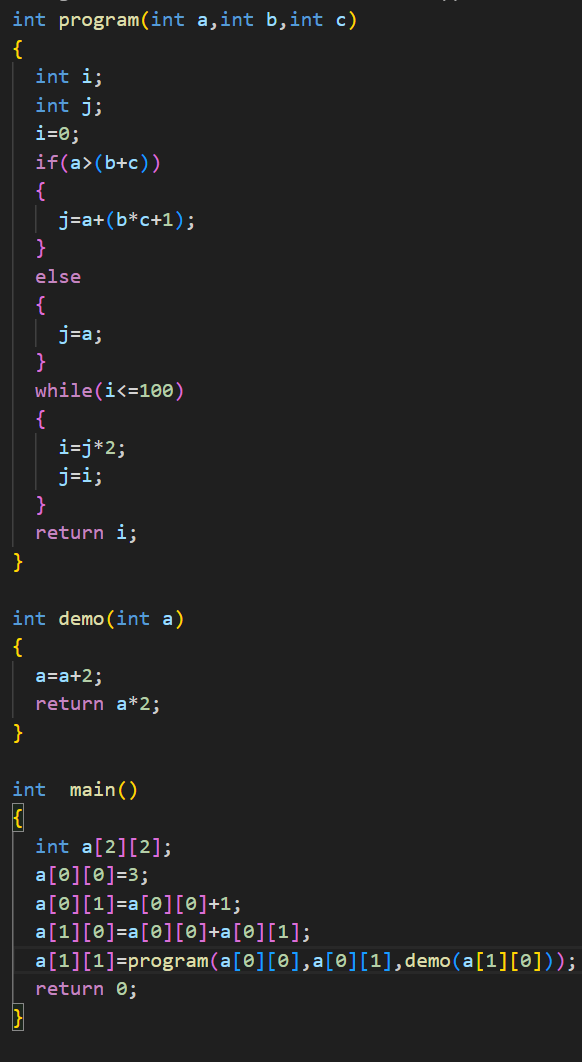


图1-4. right2.cpp测试样例



图1-5. right2.cpp前端输出结果

### 1.4.2 错误测试样例

(1) wrong1.cpp：错误源程序（注释错误）

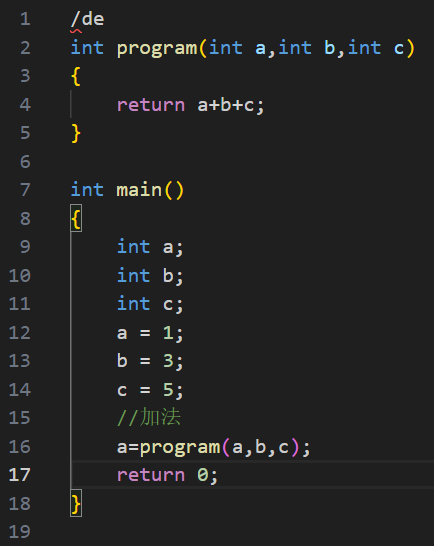


图1-6. wrong1.cpp测试样例

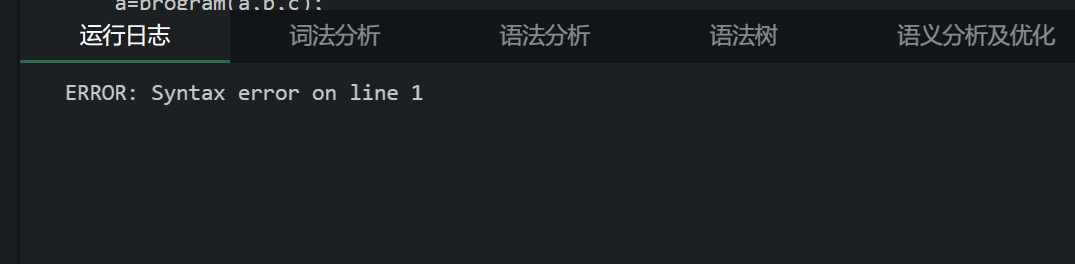


图1-7. wrong1.cpp前端输出结果

1. wrong2.cpp：错误源程序（不符合文法生成式）

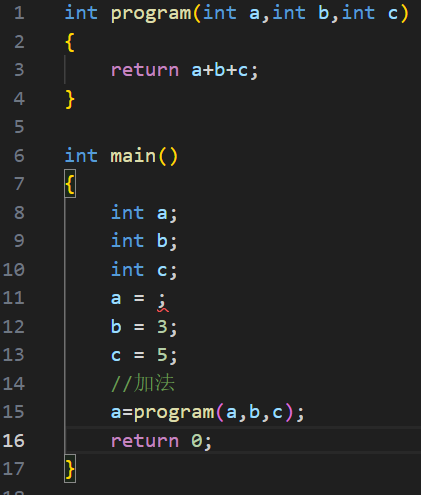


图1-8. wrong2.cpp测试样例

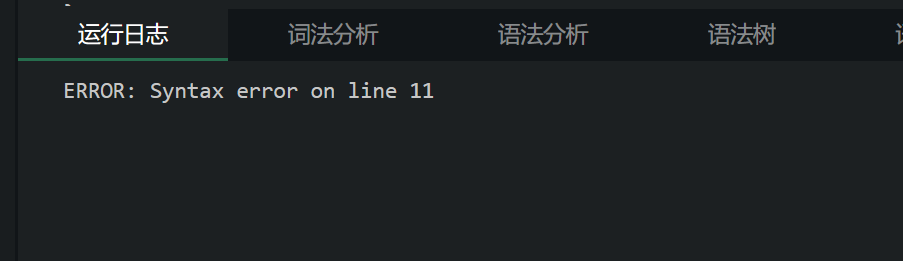


图1-9. wrong2.cpp前端输出结果

1. wrong3.cpp：错误源程序（函数无返回）

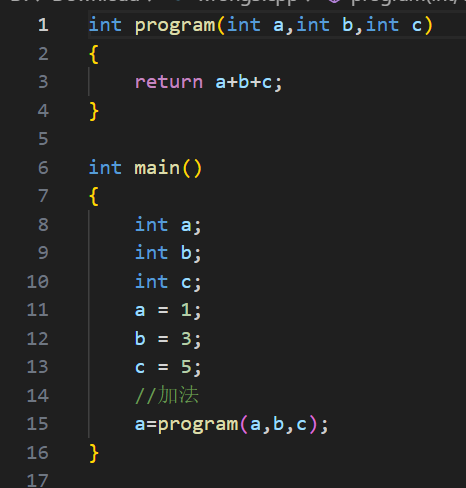


图1-10. wrong3.cpp测试样例

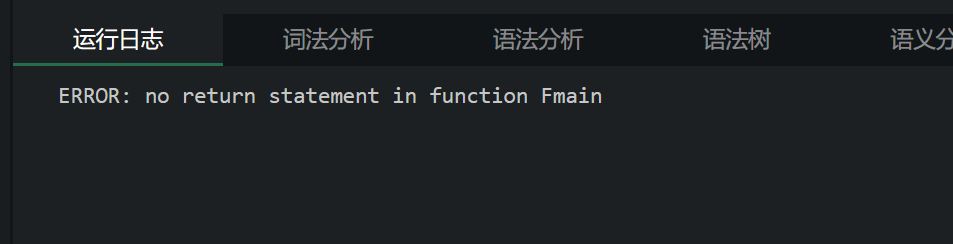


图1-11. wrong3.cpp前端输出结果

1. wrong4.cpp：错误源程序（变量在申明之前调用）

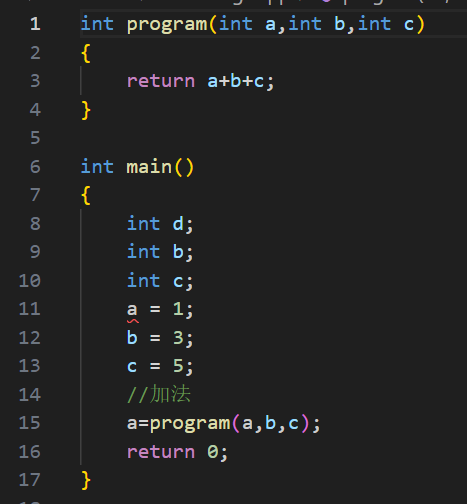


图1-12. wrong4.cpp测试样例



图1-13. wrong4.cpp前端输出结果

1. wrong5.cpp：错误源程序（函数调用与声明时形参不对应）

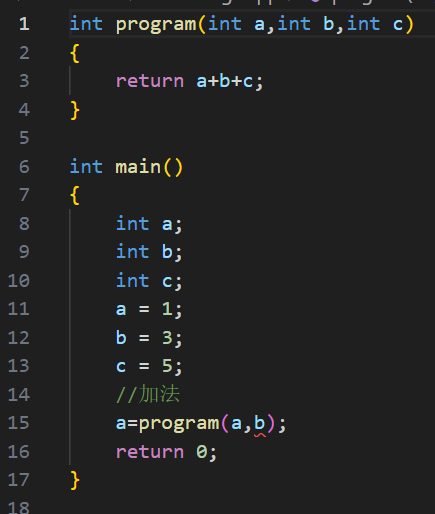


图1-14. wrong5.cpp测试样例

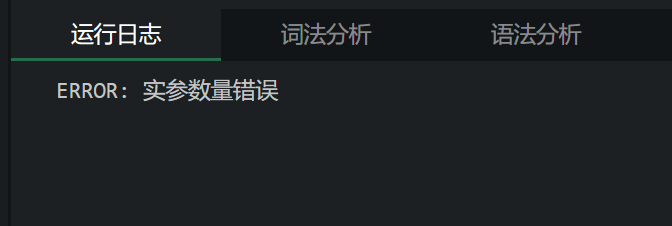


图1-15. wrong5.cpp前端输出结果

# 2 概要分析

## 2.1 任务的分解

本次类C编译器的设计，根据编译器的实现可以分解为词法和语法分析、语义分析、优化和目标代码生成；根据前后端设计可以分解为，后端编译功能和文件功能实现、后端数据库连接、前端网页设计和前后端接口连接。

## 2.1 编译器实现任务的分解

针对词法分析任务，要完成的目标是：

1. 依据给定的词法规则和正则表达式，将输入的源程序分解为一个个词法单元。
2. 对于非法的输入，即不符合词法规则的部分，抛出错误提示。

针对语法分析任务，要完成的目标是：

1. 基于给定的文法，生成文法的FIRST集合和FOLLOW集合。
2. 根据文法生成LR(1)规范簇，并构建文法对应的DFA。
3. 使用Action表和Goto表对词法单元序列进行分析，生成语法树。
4. 检测语法错误，并抛出错误提示。

针对语义分析任务，要完成的目标是：

1. 对语法树进行语义分析，验证源代码的语义正确性。
2. 执行类型检查，确保变量和表达式的类型匹配。
3. 进行符号表管理，记录变量的声明和作用域。
4. 进行语义规则验证，例如函数调用的正确性。
5. 生成中间代码所需的四元式。
6. 检测语法错误，并抛出错误提示。

针对优化任务，要完成的目标是：

1. 对生成的中间代码进行优化，以提高程序的性能和效率。
2. 使用优化技术对中间表示形式进行修改，例如常量折叠、循环展开等。
3. 生成优化后的中间代码。
4. 检测语法错误，并抛出错误提示。

针对目标代码生成任务，要完成的目标是：

1. 将优化后的中间代码转换为目标机器的机器代码或汇编代码。
2. 执行寄存器分配、指令选择等操作。
3. 生成目标代码。
4. 检测语法错误，并抛出错误提示。

## 2.1 前后端设计实现任务的分解

针对后端编译功能和文件功能任务，要完成的目标是：

1. 使用JAVA语义实现类C编译器，实现后端编译功能，接收用户上传的源代码字符串，并进行编译过程的各个阶段。
2. 实现文件功能，包括文件上传、文件保存、文件读取和文件删除等操作。
3. 将用户上传的源代码传递给编译器，并接收编译器各个阶段生成的结果。

针对后端数据库连接任务，要完成的目标是：

1. 使用mysql建立数据库system\_tree，之后通过设置MyBaits-plus配置后端服务器和数据库之间的连接参数（如数据库的地址、用户名、密码等），实现后端与数据库的连接和数据交互功能。
2. 在springboot项目中创建对应的Controller、Servic和Mapper，将用户上传的数据存储到数据库中，并能够从数据库中检索和修改数据。

针对前端网页设计和前后端接口连接任务，要完成的目标是：

(1)前端通过vue实现前后端分离设计，其中通过css来调整页面格式，js来实现各项细节函数，实现前端的动态效果。

(2)设计和实现具有良好用户体验的前端网页界面，包括文件树、编辑区域、编辑结果展示区域和操作选项区域。

(3)通过Fetch API实现从前端端口到后端端口的访问，实现get、post等多种方式的传输。

## 2.2 数据类型的定义

(1) 词法分析部分：

LexicalAnalysis：语法分析器类

(2) 语法分析部分：

Gramar：文法类

Closure：文法闭包类

Canonical：文法生成式类

ActionItem：ACTION表项类

TreeNode：语法树类

SyntaxAnalysis：语法分析器类

(3) 语义分析部分：

TASitem：中间代码四元式类

SymbolTable：符号表类

SymbolTableItem: 符号表项类

SemanticAnalysis：语义分析类

(4) 优化部分：

BlockItem：中间代码分块类

DAGitem：DAG项类

OptimizerAnalysis：优化类

(5) 目标代码生成部分：

ObjectCodeGenerator：目标代码生成类

MassageTableItem：待用活跃信息表项类

AnalysisHistoryItem：寄存器分配类

(6) 前后端设计部分：

CompileInput：编译输入类  
 ResultSave：目标代码保存类

SystemNode：前端文件树类

SystemTree：后端文件树类

SyntaxTreeNode：语法树展示类

## 2.3 主程序流程

本类C编译器项目基于前端vue，后端springboot，数据库mysql搭建。在整个主程序(编译程序)执行过程中，首先前端在选中某个文件的前提下，点击页面上方的编译按钮，之后将文件的文件内容信息通过Fetch API传给后端端口8081，之后后端对输入的类C代码进行词法分析、语法分析、语义分析、优化和目标代码生成，在这五个阶段中，程序会将编译过程的中间结果保存到类变量中，最后将生成的目标代码传回前端。如果在编译的过程中出错则将错误信息传向前端。

在编译完成后，前端向后端多次通过Fetch API发送请求获取中间结果，之后前端将获取到的结果展示到网页的展示部分，同时绘制出语法树。到此，完整的编译过程完成。

## 2.4 模块间的调用关系

1. 总体架构

首先调用SyntaxAnalysis模块的getinput函数获取前端传的代码后，之后调用SyntaxAnalysis模块生成中间代码，在获取中间代码后再通过OptimizerAnalysis模块对中间代码进行优化，最后通过ObjectCodeGenerator模块在中间代码的四元式基础上生成对应的目标代码。其中SyntaxAnalysis模块包含词法分析模块、文法模块、语法分析模块和语义分析模块。

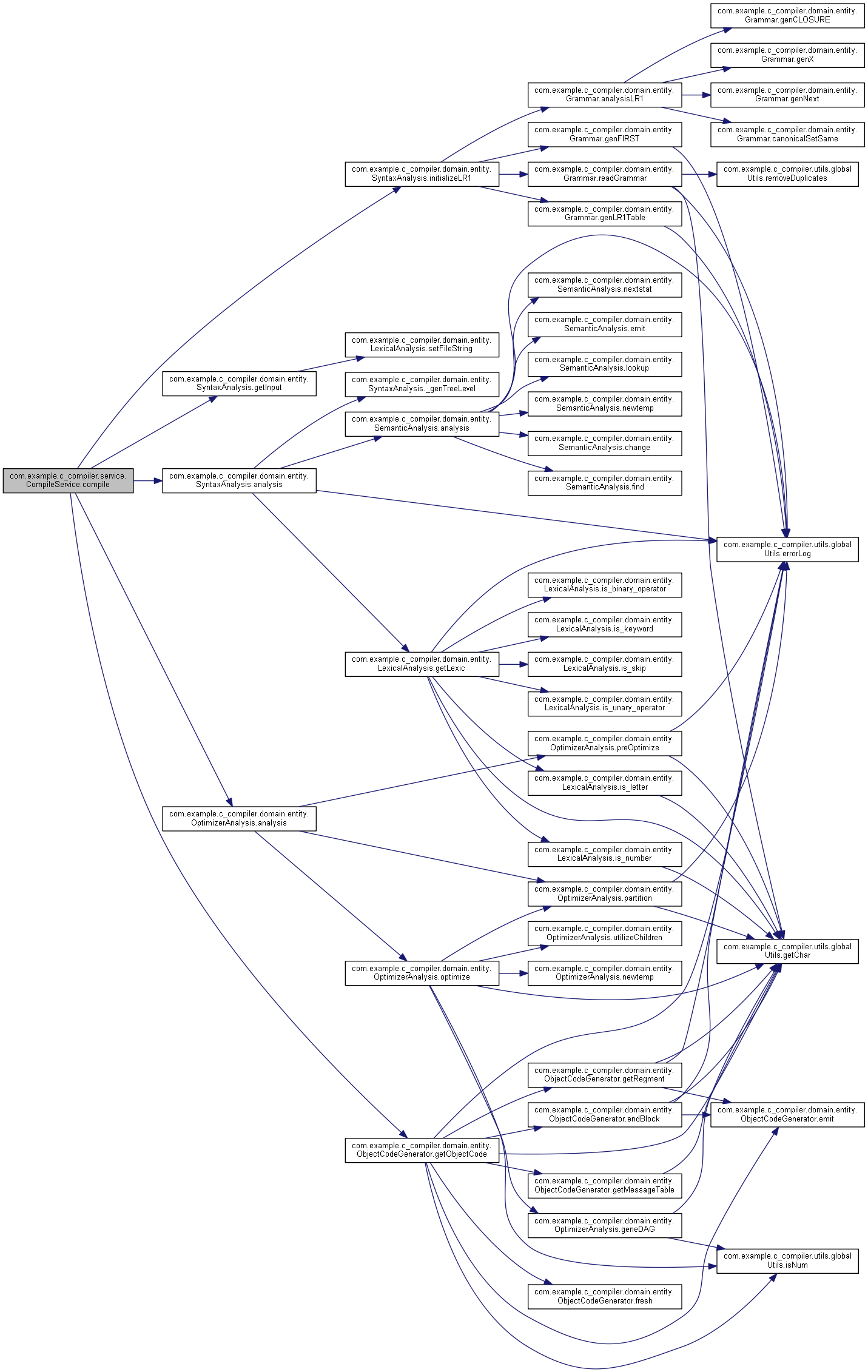


图2-1. 总体架构图

1. 词法分析模块

在本项目中，词法分析并不是独立的，而是附属于语法分析模块的。在语义分析中的analysis函数中通过getLexic函数获取一个经过词法分析的，之后将词法单元进行语法分析和语义分析，实现一次编译而不是多次编译。

1. 文法模块

在本项目中，文法模块并不是独立的，而是附属于语法分析模块的。在词法分析模块将输入的类C代码转为Token序列前，首先会在文法模块读入grammar文件时进行文法导入、First集的计算、LR1分析表的建立等，因此我们才能在后续的分析中对Token序列进行推导并基于此生成语法树。

1. 语法分析模块

在本项目中，语法分析是在文法对如后进行的，在获取到LR(1)分析表之后，语法分析模块通过栈的方式实现推导，并得到对应的Token序列，其中每个Token对应一个产生式。在推导的同时，在此过程中会完成语法树的建立。

1. 语义分析模块

为实现一遍遍历，在本项目中语义分析模块集成在语法分析之后，在语法分析经过推导获得Token序列后，对每个Token对应生成式，并根据生成式添加对应的四元式作为中间代码。下图为Syntaxanalysis模块的调用关系。

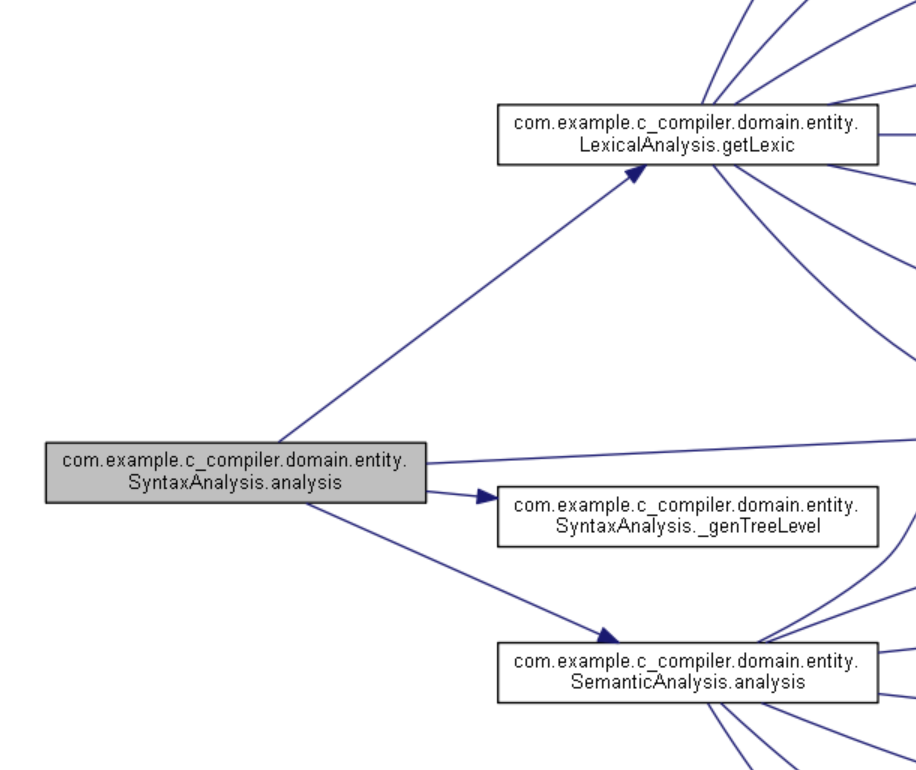


图2-2. SyntaxAnalysis模块调用词法分析、语法分析和语义分析模块

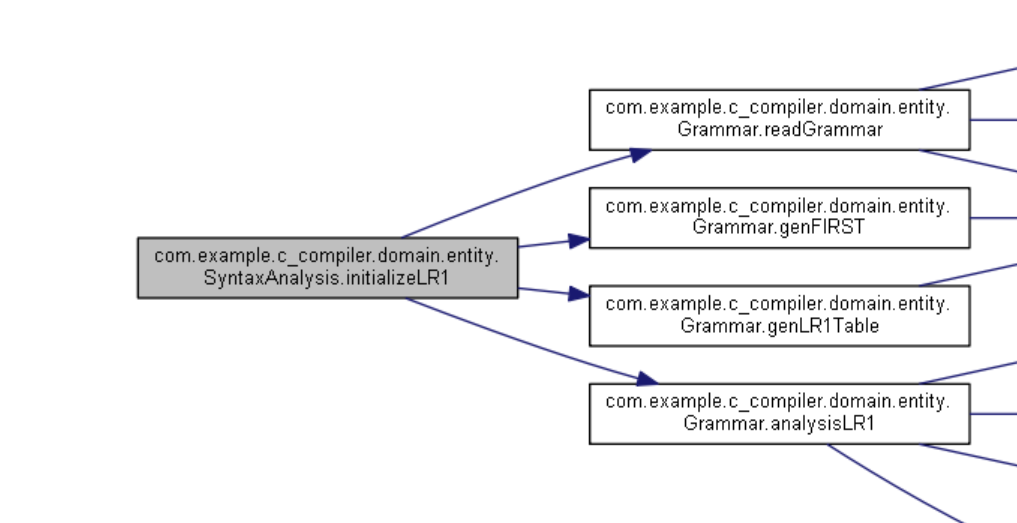


图2-3. SyntaxAnalysis模块调用Grammar模块

1. 优化模块

在优化模块中，对输入的四元式进行划分块，之后进行多次优化，直到优化前后代码相同结束。该模块没有嵌套的子模块，相对其他模块是独立的。

1. 目标代码生成模块

在目标代码生成模块中，对输入的中间代码进行对照的目标代码的生成，在这个过程中，最重要的是寄存器的分配以及在堆栈中的存储问题。与优化模块类似，该模块没有嵌套的子模块，相对其他模块是独立的。

1. 可视化模块

在可视化模块中，主体实现在前端对UI的设计以及对交互逻辑的设计，后端主要是作为前端和数据库之间连接的桥梁，前端有四个模块，分别是Menu模块、Explorer模块、Editor模块和Compile Result模块。在实现过程中，由于位置以及功能的相似行，后两个模块都集成在Editor模块中。整体来说，可视化部分的模块分析较为复杂，考虑到本项目以编译器为主体，因此不对这四个模块具体展开。

# 3 详细分析

## 3.1 词法分析部分

在读入从前端传送的字符串后，词法分析器通过getLexic()函数从输入字符串中获取词法单元，实现的具体细节是通过设计state对当前字符及下个字符进行处理（这里是因为考虑到超过一位的操作数带来的影响）。

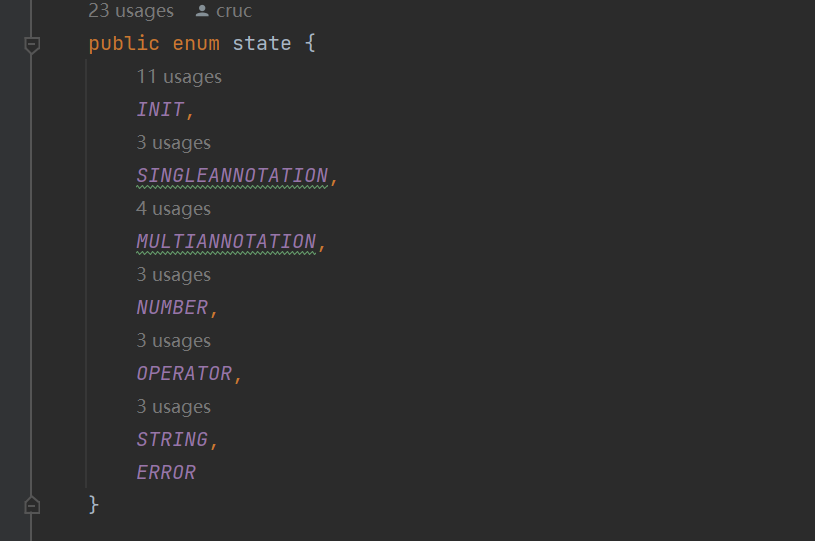


图3-1. globalUtils类中定义词法分析状态

其中state被定义在gloablUtil类中，这几个状态对应这数字、字符串、操作数和注释以及初始和错误状态，针对这些状态，我可以向语法分析说明当前词法单元的属性，因此可以方便后续分析的进行，下图为getLexic函数的调用关系。

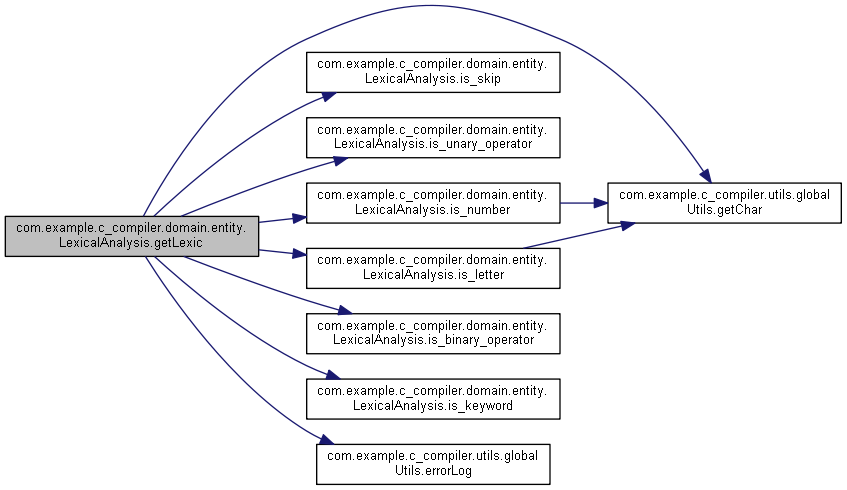


图3-2. getLexic函数的调用关系

## 3.2 文法部分

整个编译过程中的文法导入发生在SyntaxAnalysis类的initializeLR1函数，这个函数的调用关系如下。

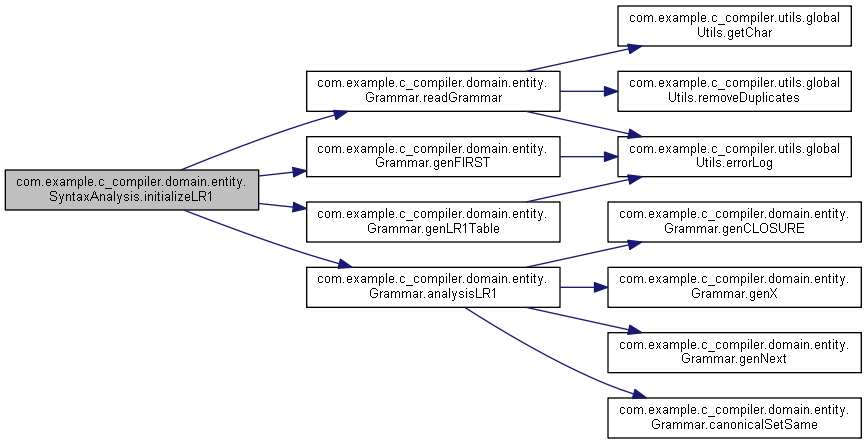


图3-3. initializeLR1的调用关系

这四步分别是读入Grammar文件、生成Fisrt集合、生成LR(1)项目簇以及生成LR(1)分析表，下面针对这四个函数及其子函数对文法的读入的详细过程进行分析。

首先是读入Grammar文件过程中，在将产生式左侧存入VN，将产生式右侧对应存入VT，如果在读入文件时遇到错误会抛出错误，其函数调用图如下。

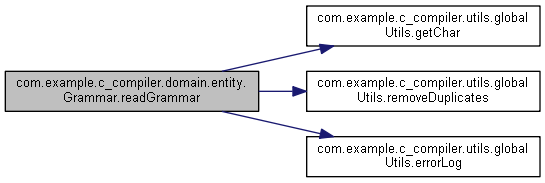


图3-4. readGrammer的调用关系

之后是生成First集合，在genFisrt函数中，针对VT和VN，将VN中所有选择的终结符或未终结符的First集合放入VT中对应元素的Fisrt集合中，直到First集合不再发生变化，如果在计算First集合时遇到错误会抛出错误，其函数调用图如下所示。

IMG_256

图3-5. genFisrt的调用关系

之后是建立LR(1)项目集簇，下图代码是核心代码，在genCLOUSE函数会更新簇，之后判定簇是否需要继续更新，直到不再更新后将其加入到collection中。

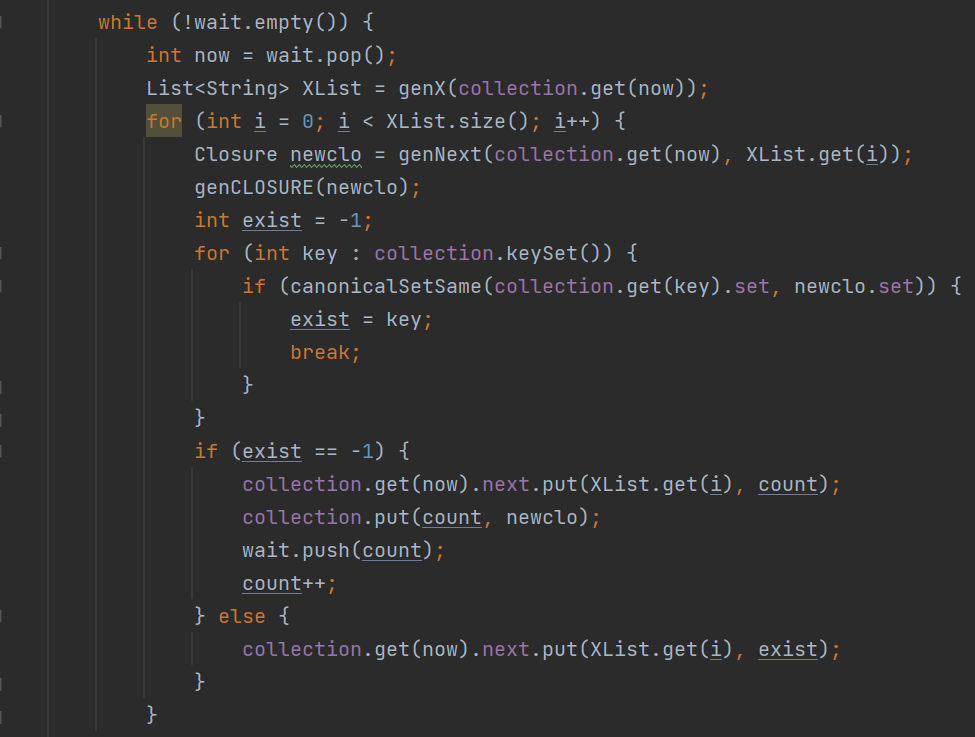


图3-6. analysisLR1函数核心代码

下图是 analysisLR1函数的调用关系。

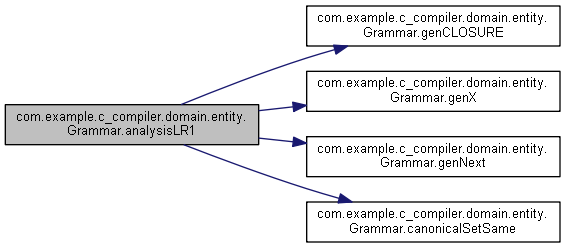


图3-7. analysisLR1函数调用关系

最后是构造LR(1)分析表，对构建好的项目集簇进行遍历，通过分析项目集簇的跳转关系对action表和goto表进行复制，其核心代码如下图所示

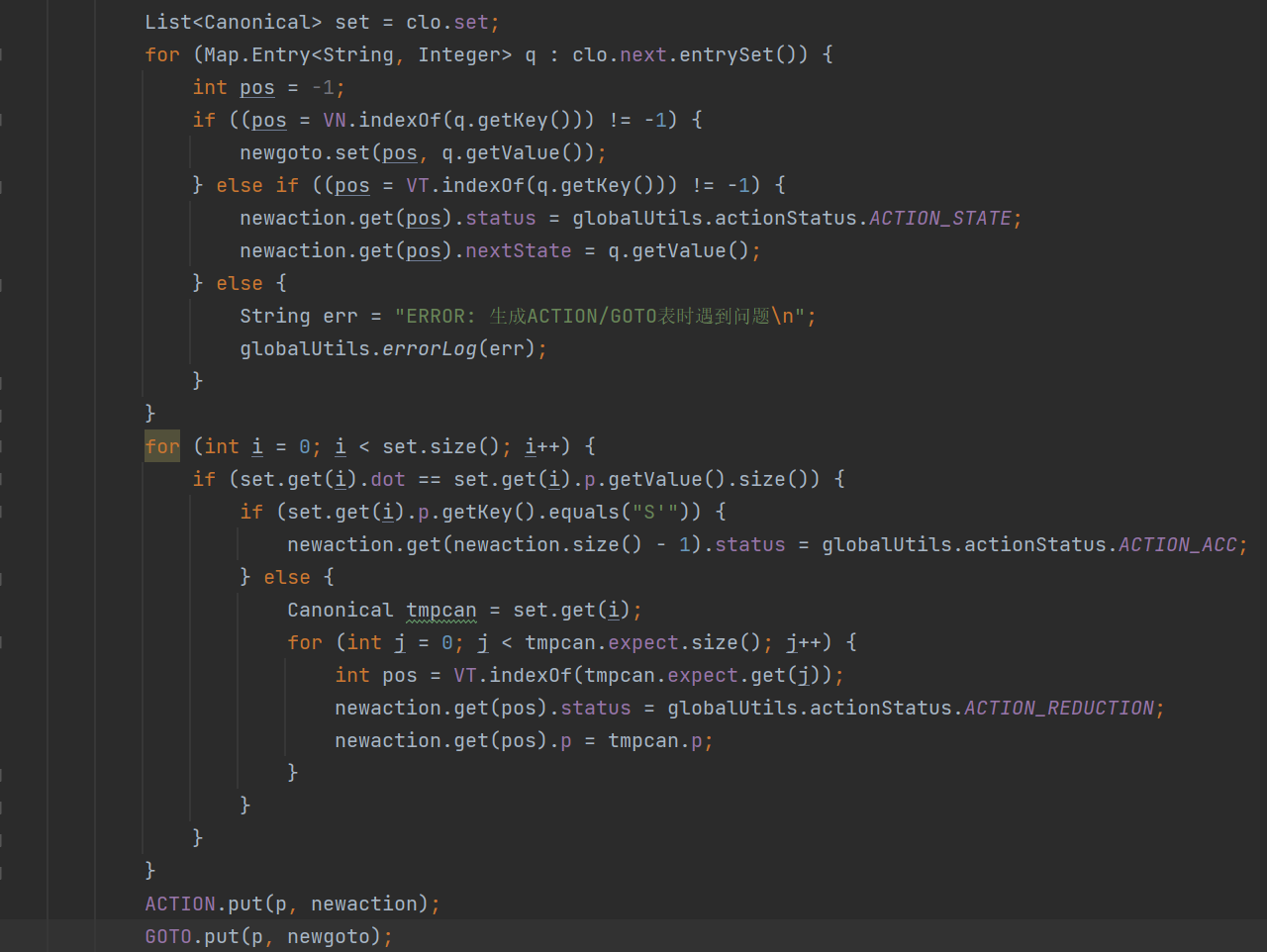


图3-8. action表和goto表的构造方法

下图是 genLR1Table函数的调用关系。

IMG_256

图3-9. genLR1的调用关系

## 3.3 语法分析部分

语法分析的核心代码是analysis函数，这段代码连接了语法分析和语义分析，其通过状态的转换来实语法分析，其状态定义如下。

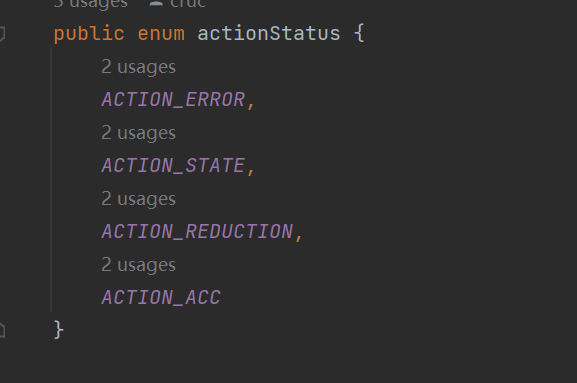


图3-10. globalUtils类中定义语法分析状态

如果是 ACTION\_ACC，表示分析成功，生成语法分析树，并结束分析过程。

如果是 ACTION\_ERROR，如果当前词法单元不是 epsilon（空符号），则保存下一个词法单元，将当前词法单元设置为 epsilon，并继续下一次循环。否则，输出语法错误信息。

如果是 ACTION\_STATE，表示需要移入动作。创建一个新的树节点（tp），将当前词法单元设置为节点内容，并将节点推入栈中。更新状态和符号，并获取下一个词法单元。

如果是 ACTION\_REDUCTION，表示需要归约动作。获取归约产生式，并根据产生式对栈中的节点进行归约操作。将归约后的节点设置为新的树根节点（reductionTreeRoot），并将其推入栈中。更新状态和符号，以及相关的字符串列表（strstate、strsymbol、input）。

语法分析部分的analyse函数的调用关系如下。

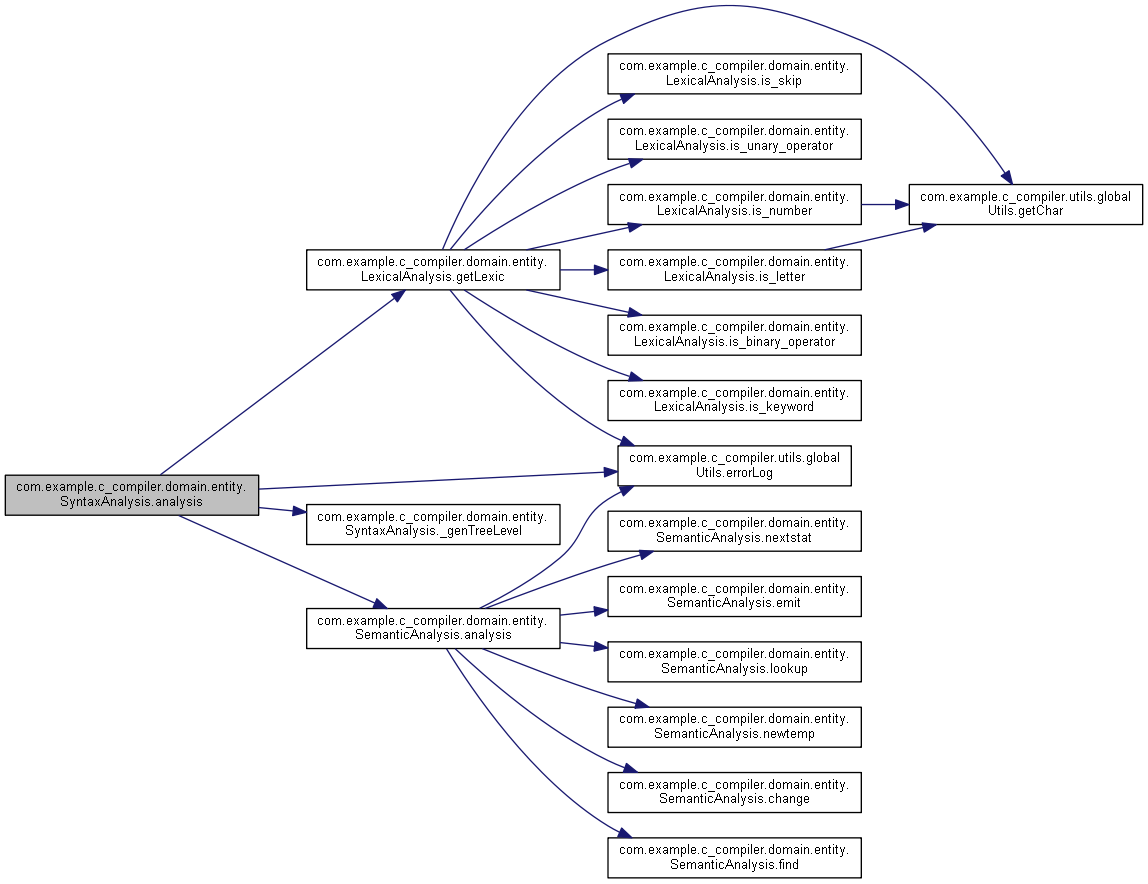
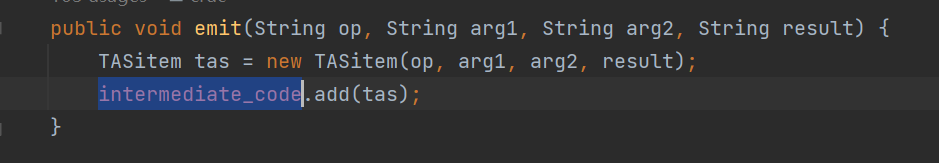


图3-11. analysis函数语法分析部分的函数调用

## 3.4 语义分析部分

在完成语法分析后，每次向语义分析中analysis函数传参Token以及nameTable。在语法分析中首先将Token与文法中生成式进行对应，并向intermediate\_code中添加四元式，添加函数为emit函数，其定义如下。

图3-12. 语义分析部分的emit函数

语义分析部分analysis函数的函数调用关系如下图所示。

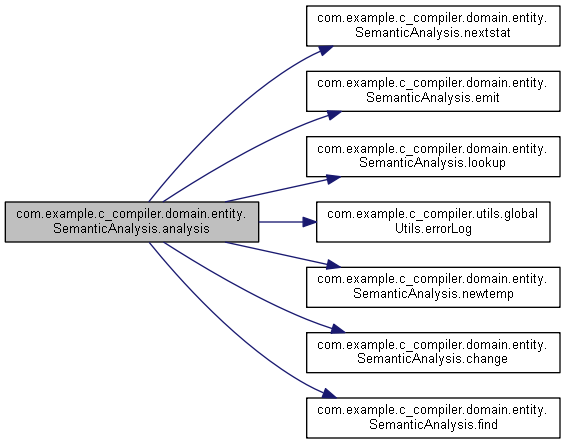


图3-13. analysis函数调用关系

## 3.5 优化部分

在得到四元式格式的中间代码后，首先调用preOptimize函数对中间代码中的跳转操作添加标签，细节如下：如果操作符是 "jal"（跳转到函数），则检查 label\_map 中是否包含 e.result 对应的标签。如果不包含，则将该标签添加到 label\_map 中，并将 e.result 更新为相应的标签；如果操作符的第一个字符是 'j'（跳转指令，如条件跳转），同样检查 label\_map 中是否包含 e.result 对应的标签。如果不包含，则将该标签添加到 label\_map 中，并将 e.result 更新为相应的标签。

preOptimize函数的调用关系如下图所示。



图3-14. preOptimize函数调用关系

在调用preOptimze函数后，优化部分会调用partition函数，这个函数实现了对中间代码的分块操作。它遍历中间代码列表 intermediate\_code 的每个元素，并根据特定条件判断是否开始或结束一个代码块，其函数调用关系如下。

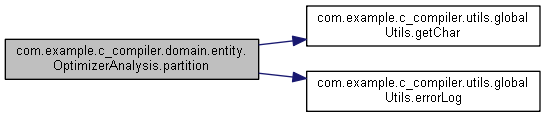


图3-15. partition函数调用关系

在完成对中间代码的分块后，会循环对中间代码进行优化，直到优化前代码和优化后代码相同。在optimize函数中，会遍历block group中每一个块，之后对每个块建立DAG。

对于DAG的保留节点，如果节点的 arg1 和 arg2 不为空且不在 wait\_variable 列表中，将它们添加到 wait\_variable 列表中。

对于非保留节点，根据节点的子节点情况进行处理：如果节点没有三个子节点，则根据节点的标签和 wait\_variable 进行筛选，并更新节点的标签和是否为有效节点的状态。然后调用 utilizeChildren方法，更新子节点的使用状态；如果节点不是叶子节点且标签为空，则添加一个新的临时标签到节点的标签列表中；如果节点有三个子节点，则将节点标记为有效节点，并调用 utilizeChildren方法，更新子节点的使用状态。

最后遍历 DAG 中的每个节点，根据节点的类型将代码项添加到 optimized\_code 中，并将其中连续的加法操作合并为同一个。Optimize函数的调用关系如下图所示。



图3-16. optimize函数调用关系

## 3.6 目标代码生成部分

在得到优化后的代码后，目标代码生成部分会调用getObjectCode函数生成目标代码。

首先，通过调用emit()方法生成一些初始的目标代码指令，包括.data、.text等。这些指令用于设置数据段和代码段的基本信息。然后，通过循环遍历block\_group列表中的每个块，获取每个块的消息表MessageTable。

对于每个消息表中的项目，根据操作符和操作数生成目标代码指令。根据操作数的类型，可能需要执行一些寄存器分配和数据加载的操作。在生成目标代码指令时，会考虑一些特殊的操作符，如函数调用、条件跳转等，对应的指令会相应地生成。在处理完每个项目后，会将生成的目标代码指令存储在new\_code列表中。getObjectCode函数的调用关系如下图所示。

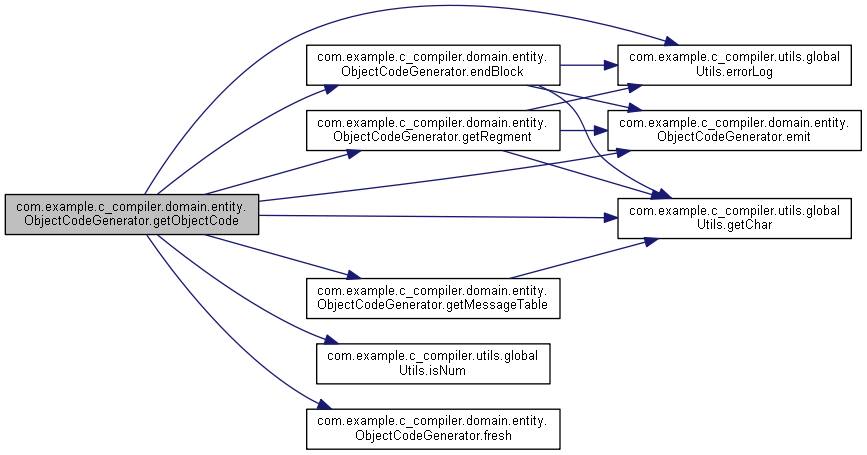


图3-17. getObjectCode函数调用关系

## 3.7 可视化设计部分

可视化部分包括前端和后端，由于其整体公告较为复杂，因此我仅选取与编译以及文件相关的功能进行分析。首先是语法树的展示，在获取reductionTreeRoot作为树根后，getSyntaxTree函数调用dfs函数对子节点进行深度优先搜索，并将结果以json的方式返回到前端。dfs代码如下图所示。

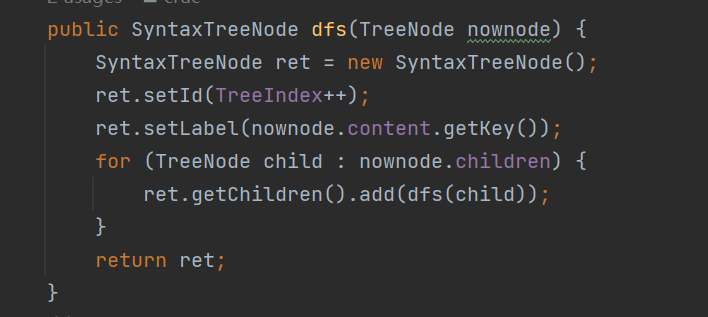


图3-18. dfs获取语法树信息

getSyntaxTree函数的函数调用关系如下图所示。

IMG_256

图3-19. dfs获取语法树信息

前端代码如下图所示。



图3-20. 语法树对应前端代码



图3-21. 语法树结果

资源管理系统的文件树实现的具体代码如下所示，在数据库中文件以文件id、文件名、父节点id、文件类型和文件内容存储，其中文件类型包括文件和文件夹。但是为方便调用，在前端文件以文件id、文件名、文件子节点、文件类型和文件内容存储，因此我需要在后端将两种格式进

行转换。代码如下所示。

****

图3-22. 文件类型转换

数据库中数据保存形式如下图所示。

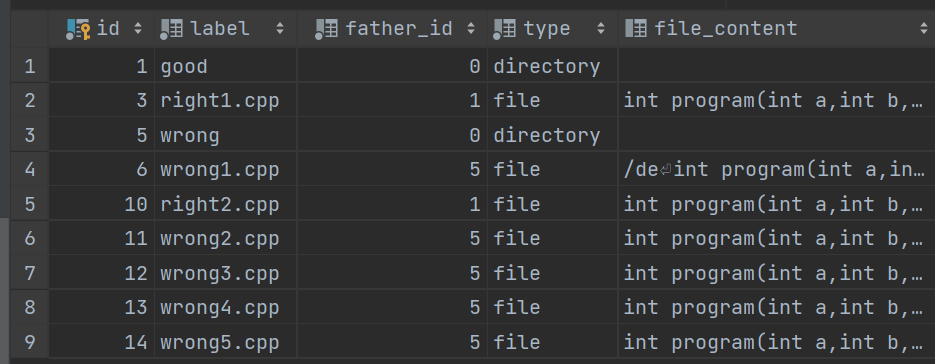


图3-23. system\_tree数据库

前端资源管理器部分文件树展示如下，其与上方数据库是对应的。

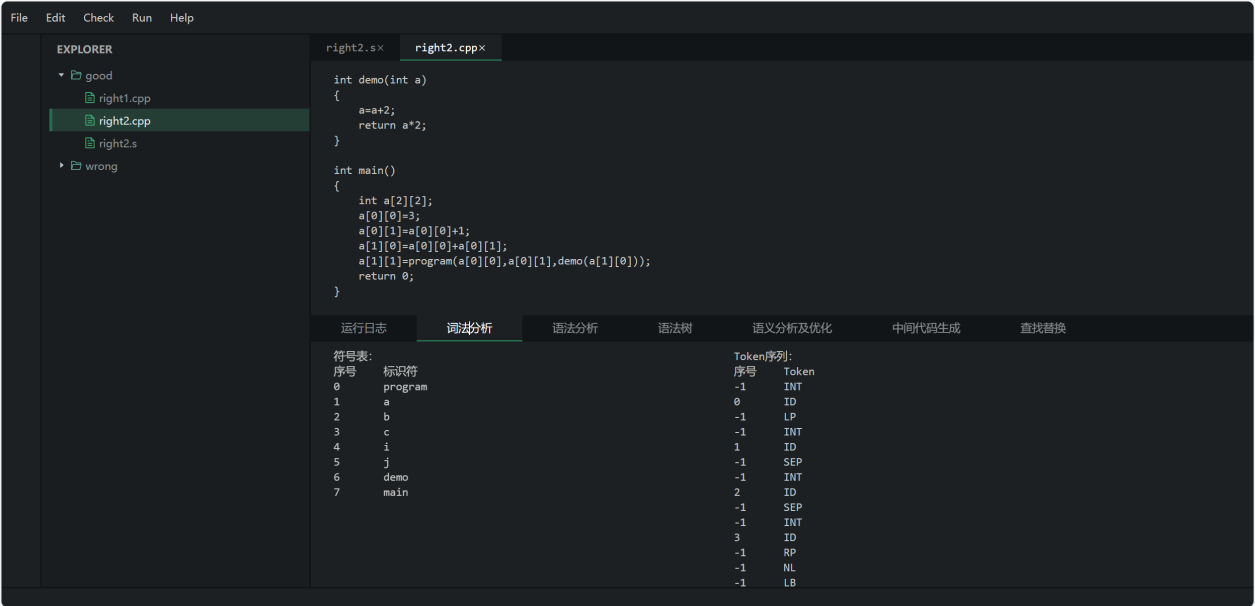


图3-24. 前端Explorer展示

# 4 调试分析

## 4.1 数据测试

1. 正确的测试样例，测试结果如下：

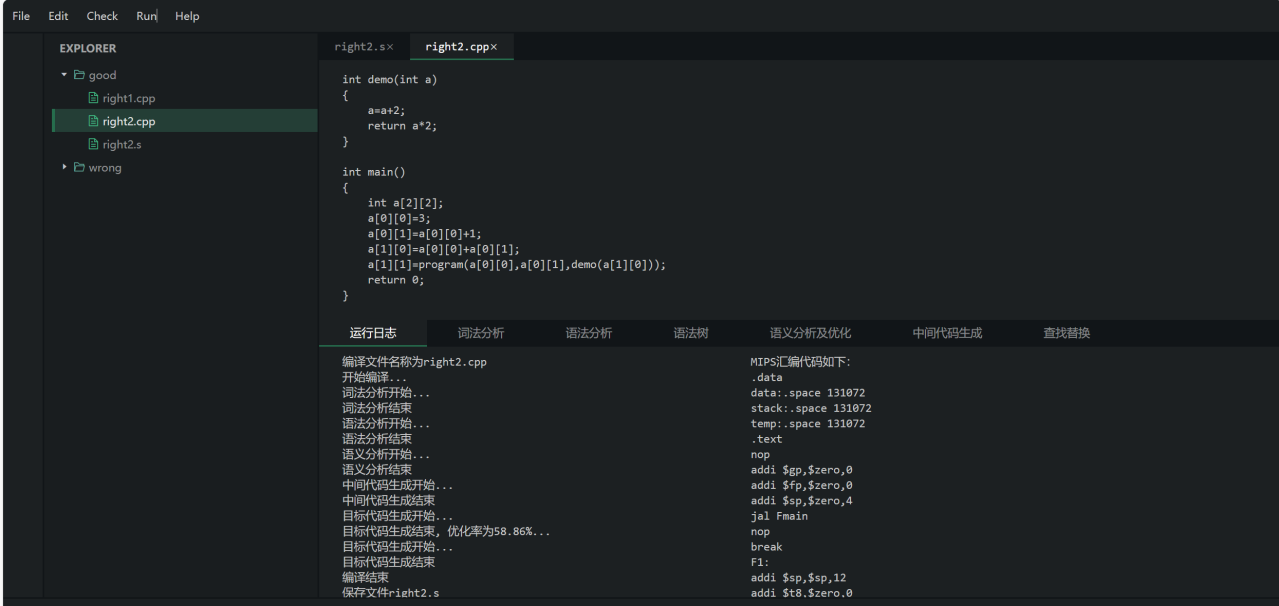


图4-1. 正确样例运行日志图

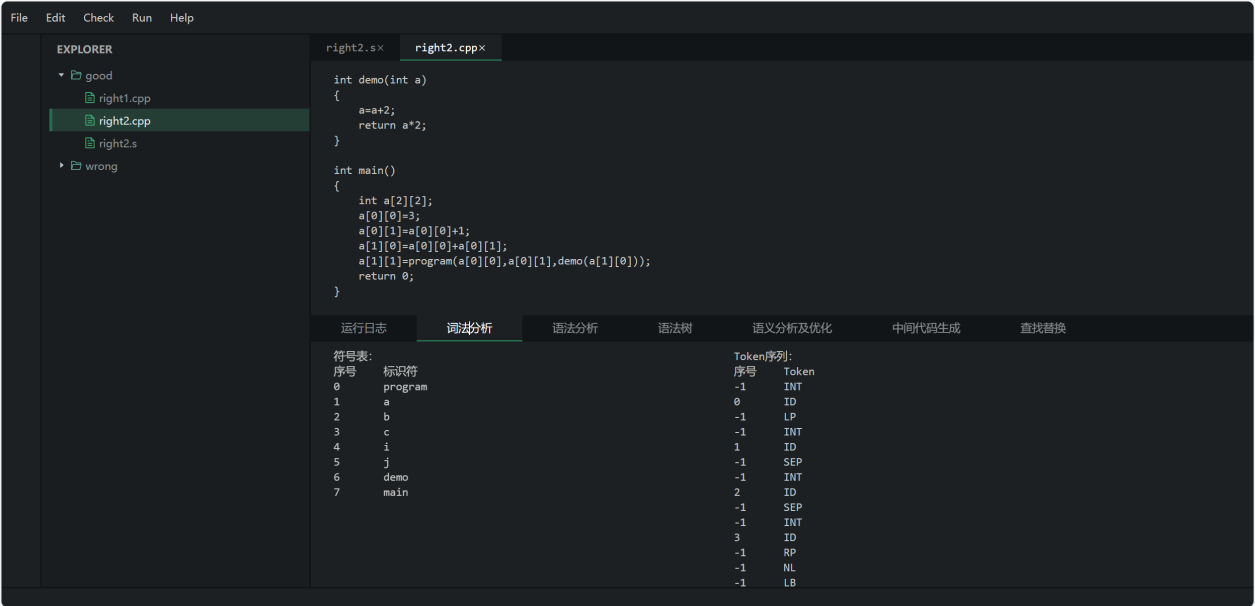


图4-2. 正确样例词法分析图

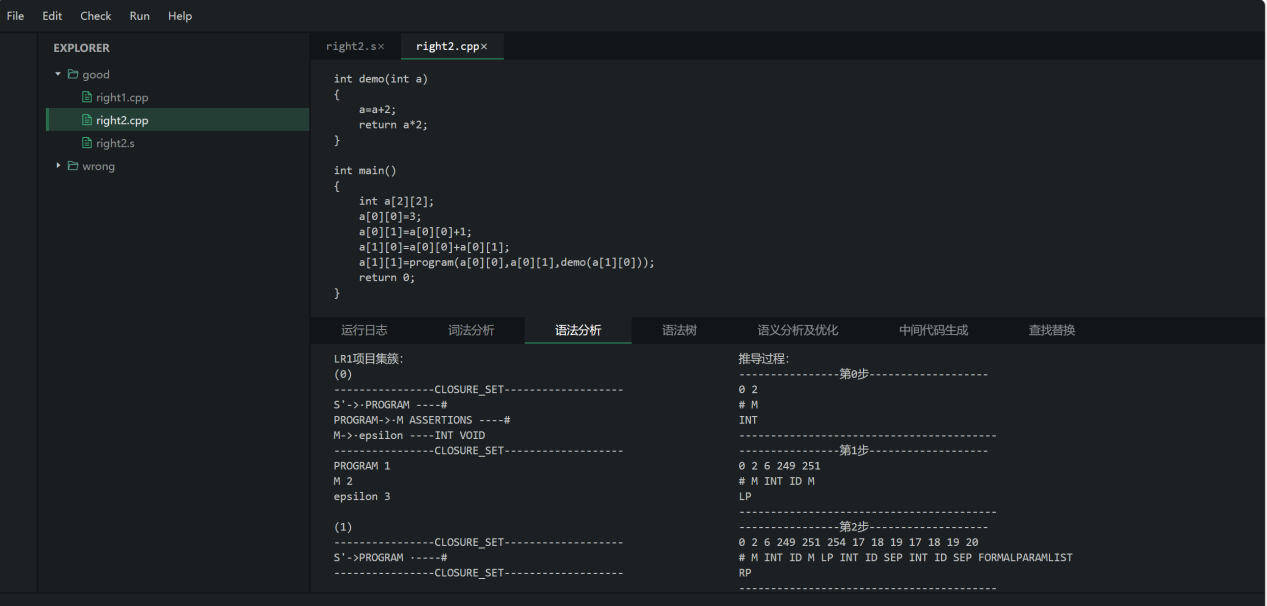


图4-3. 正确样例语法分析图



图4-4. 正确样例语法树

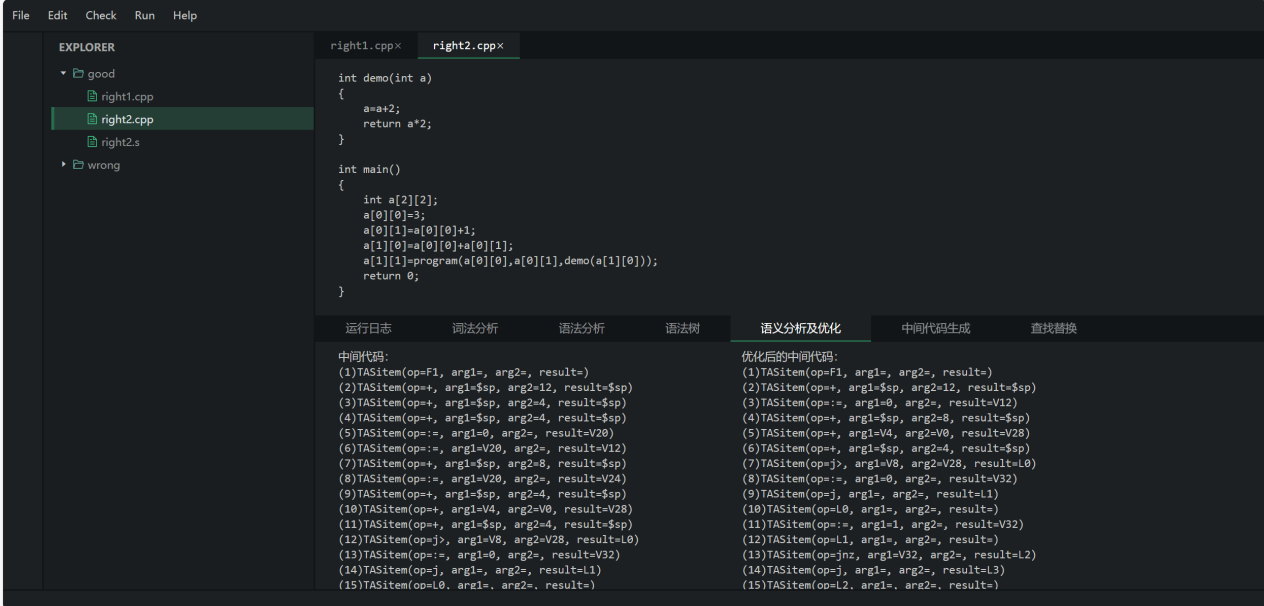


图4-5. 正确样例语义分析及优化图

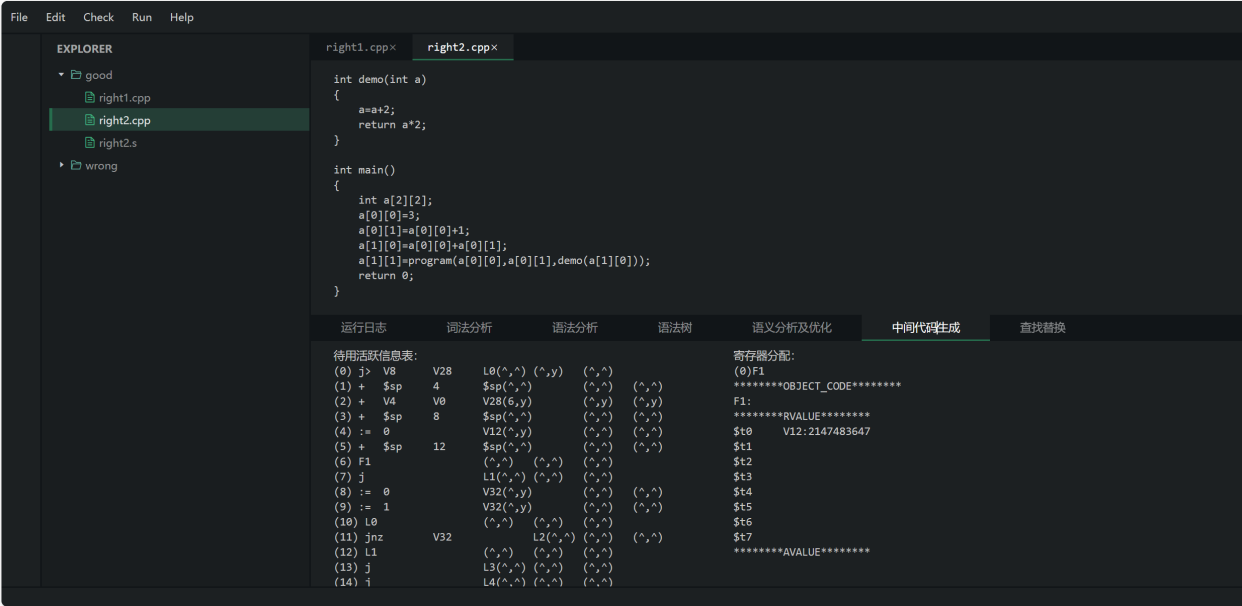


图4-6. 中间代码生成图

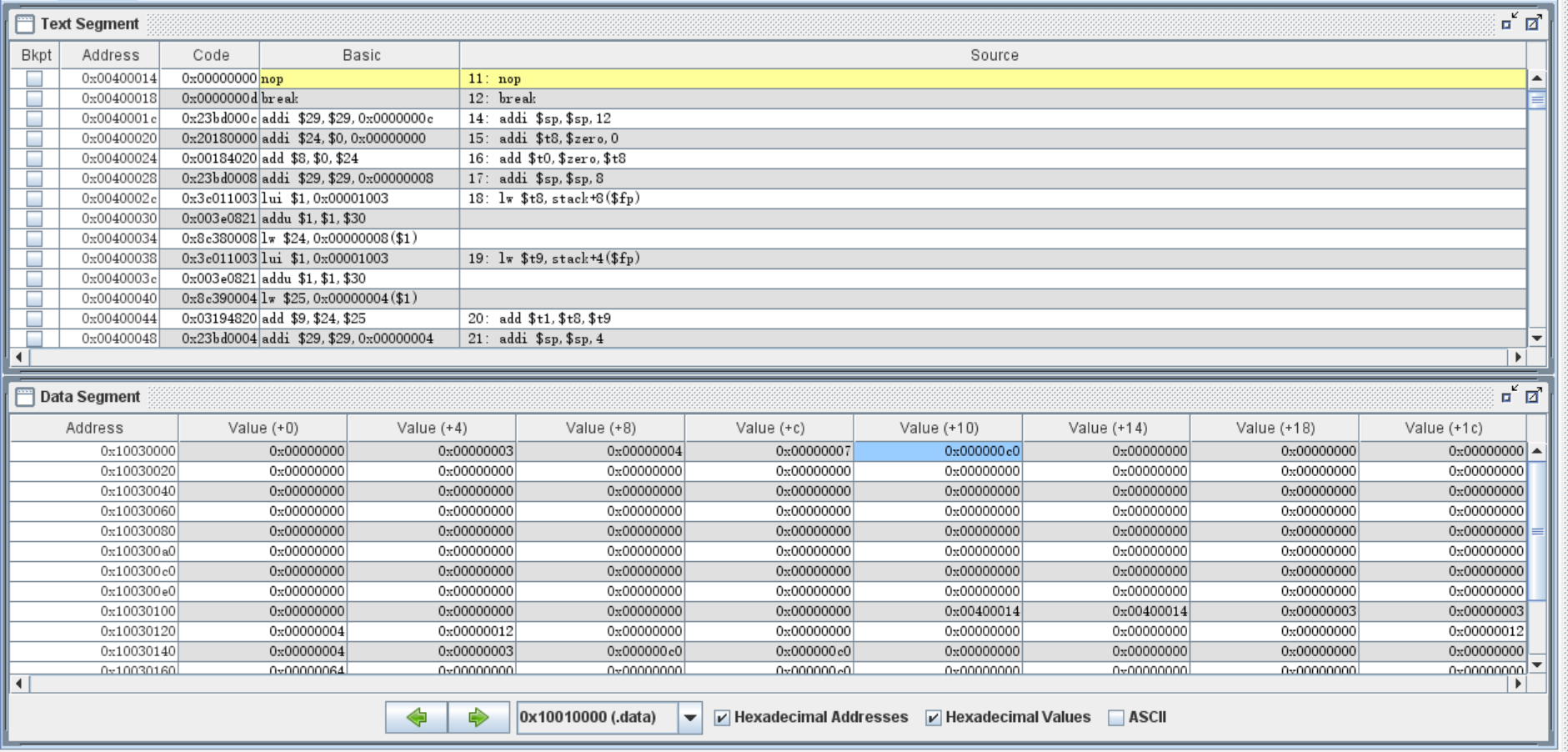


图4-7. Mars汇编运行图

1. 错误源程序（注释错误），测试结果如下：



图4-8. 注释错误报错图

1. 错误源程序（不符合文法生成式），测试结果如下：

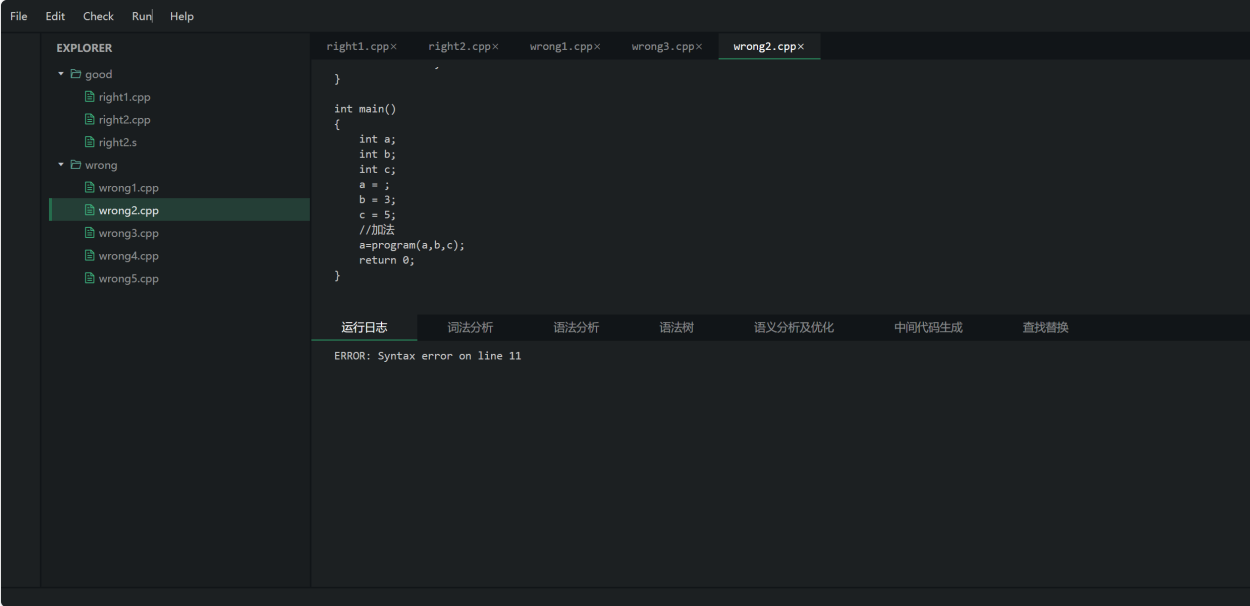


图4-9. 不符合文法生成式报错图

1. 错误源程序（函数返回值缺失），测试结果如下：

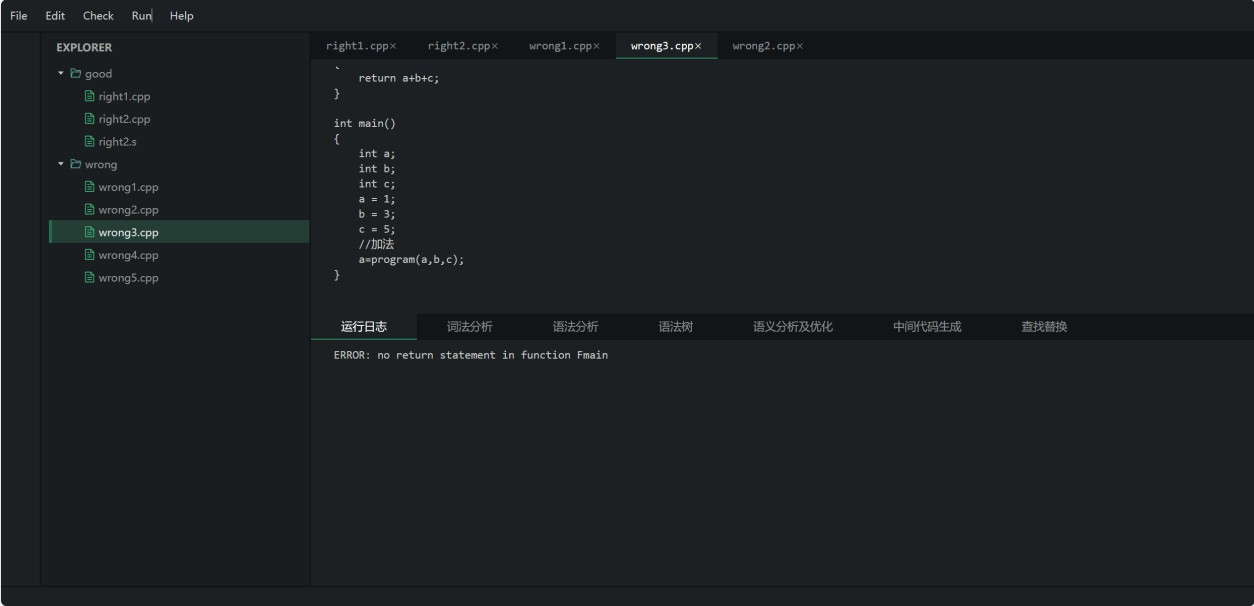


图4-10. 函数返回值缺失报错图

1. 错误源程序（变量在申明之前调用），测试结果如下：



图4-11. 变量在申明之前调用报错图

1. 错误源程序（函数调用与申明时形参不对应），测试结果如下：



图4-12. 函数调用与申明时形参不对应报错图

## 4.2 时间复杂度分析

1. 词法分析部分：

在分析词法分析部分的时间复杂度的时候，主要考虑getLexic()函数的时间复杂度，在这个函数中，每次调用返回一个词法单元，因此在完整的对源代码进行词法分析即将源代码进行一次遍历，因此词法分析部分的时间复杂度是O(n)。

1. 语法分析部分：

针对语法分析部分，主要分析语法分析部分的analysis函数，其对输入的词法单元进行处理，因此在遍历词法单元序列的时间复杂度为O(n)，而且在遍历过程中，语法分析会设计到在List中删除某位置的元素，由于其存储特性，因此需要O(m)的时间复杂度来计算，综上所述，语法分析的时间复杂度为O(nm)。

1. 文法部分：

针对LR(1)分析表的建立过程，有以下四步：首先是从grammar文件中读入数据，在这个阶段，时间复杂度为O(n)，因为只涉及对文件的遍历；之后是First集的建立，其包括三重循环，包括对产生式的遍历、对产生式右推导的所有第一个符号进行遍历、以及对符号的First集的处理，所以时间复杂度为O(nm2)；之后是LR(1)项目集簇的建立，在这里包含多重循环，一是wait栈的push和pop，二是对Xlist的遍历，因此时间复杂度为O(n2m)；最后是LR(1)分析表的建立，包括对VN，VT的遍历以及对set的二重遍历，因此时间复杂度为O(n(n+m2))。

1. 中间代码生成部分：

本次实验的中间代码生成部分的目标是生成给定代码对应的四元式。而实现四元式的生成部分的程序逻辑主要是在扩展语法分析过程，通过翻译模式来进行相关四元式的发送，回填等操作做。针对每个四元式，其有且仅有一条发送语句与之对应，同时四元式的四个域中的信息里，其中操作说明在发送时就已经确定，而操作数2域中的内容有时在发送时确定，有时需要回填等方法来进行确定，上述两种情况下，填入四元式的时间复杂度上界为O(4q)=O(q)其中q为最终生成四元式的条数。那么在进行翻译的过程中，对于每一个文法生成式的规约，具有有限个翻译后的说明语句与之对应。针对任何一个翻译模式，设其针对单个文法生成式的规约具有最多c个说明语句与之对应。最坏情况下每个说明语句均发送四元式，那么此时q=cn。由于四元式生成是语法分析过程的副产物，因此最终本次实验下生成中间代码的时间复杂度为O(n2mc)。

1. 可视化部分：

由于可视化部分设计前后端的连接，因此很多代码难以分析复杂度。此处仅对语法分析树中节点布局规划部分的时间复杂度进行分析。针对语法分析树的可视化，首先应用了广度优先搜索的方式构建整棵语法分析树，同时进行树中节点的布局，若语法分析树中共有个节点，则进行节点布局的时间复杂度为。

# 5 用户使用说明

## 5.1 整体功能说明

下图是我使用vue设计的前端UI，其中包括最上方的菜单，左侧的资源管理器，右侧的编辑区以及右下方的编译结果展示区。

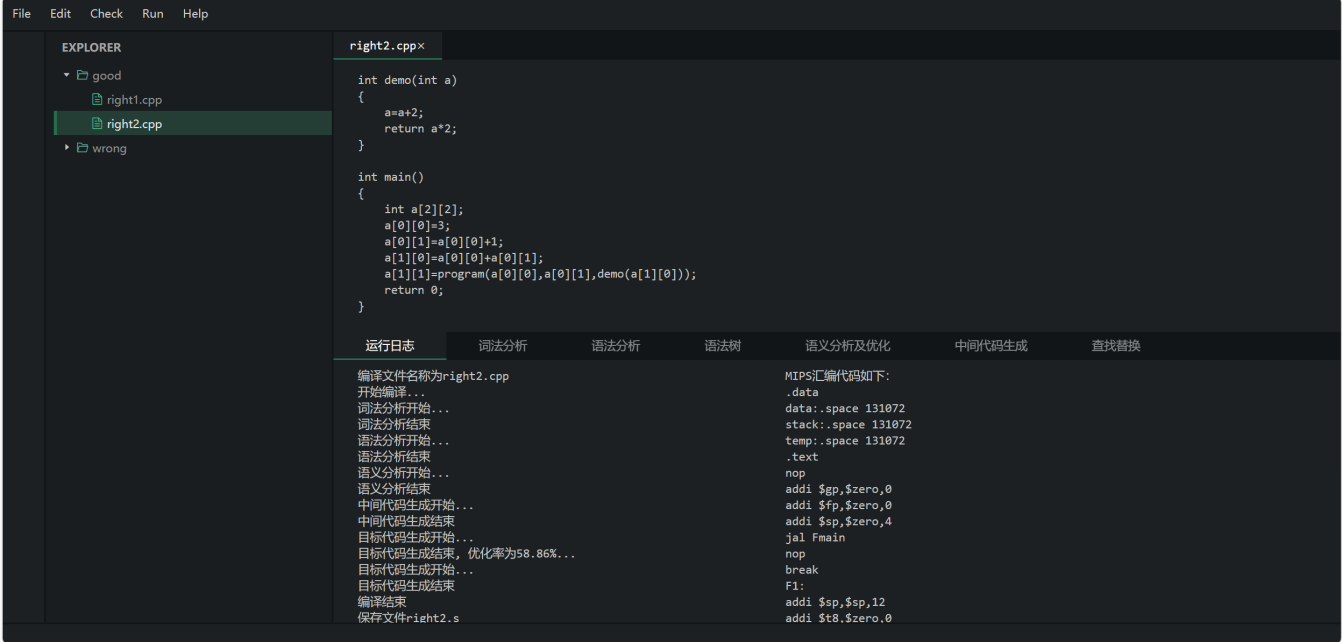


图5-1. 整体页面设计

## 5.2 细节功能说明

1、菜单（Menu）部分

菜单的一级选择项有五项，分别是File、Edit、Check、Run和Help。

1. File：新建文件，新建文件夹，保存文件，上传文件，下载文件

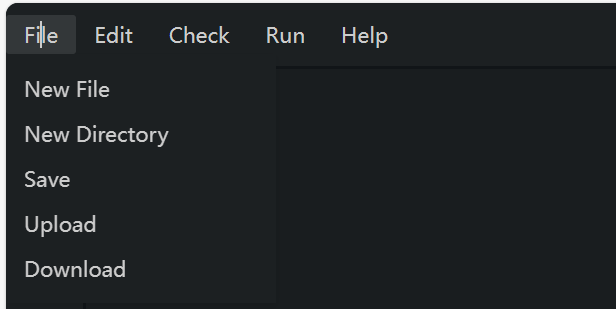


图5-2. 菜单中File子选择栏

1. Edit：剪切，复制，粘贴

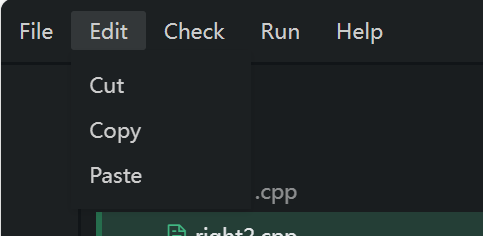


图5-3. 菜单中Edit子选择栏

1. Check：查找内容，替换内容

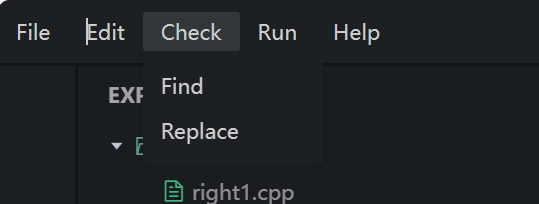


图5-4. 菜单中Check子选择栏

1. Run：编译，保存目标代码

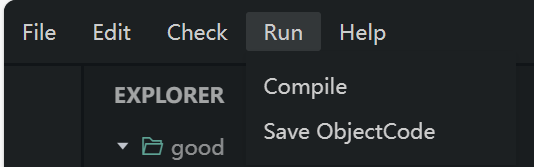


图5-5. 菜单中Run子选择栏

1. Help：打开帮助文档

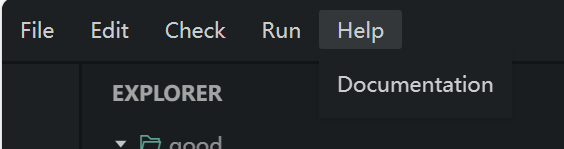


图5-6. 菜单中Help子选择栏

2、资源管理器（Explorer）部分

资源管理器部分是用于管理文件和文件夹之间的管理，主要功能包括文件关系展示和文件相关的一些功能。

1. Explorer：点击文件或文件夹即选中对应文件或文件夹，可以对其下文件或文件夹进行操作。

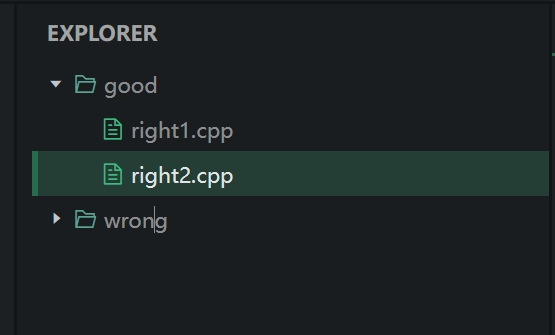


图5-7. 资源管理器部分

1. 小操作栏：在Explorer部分右键打开小操作栏，可以选择新建、删除和改名文件或文件夹。

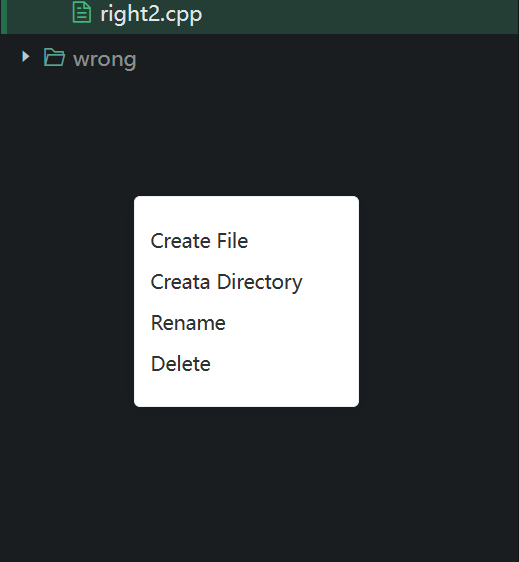


图5-8. 资源管理器中小操作栏

3、编辑区（Editor）部分

编辑器是用于用户编辑代码的部分，主要功能包括切换多个文件和使用tab、剪切、复制和粘贴进行编辑。

1. 文件栏：展示打开的文件，其中只有一个文件可以被编辑，同时如果文件被修改后，会在右侧添加星号。

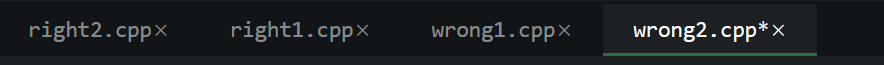


图5-8. 编辑区的文件栏

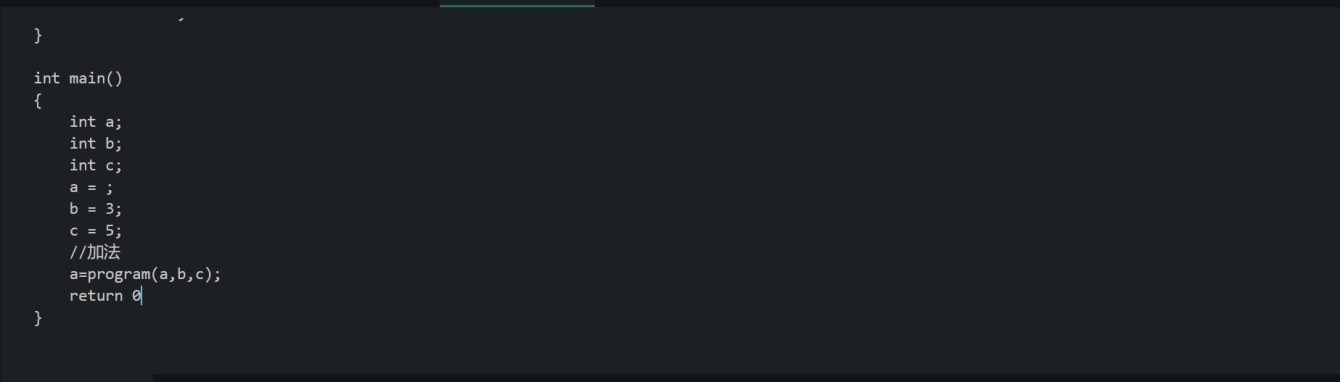
1. 编辑框：对打开的文件进行编辑，支持tab、跨文件剪切、复制和粘贴。

图5-9. 编辑区的编辑框

4、编译展示部分

编译展示部分用户展示编译的中间结果以及运行日志的部分。

1. 运行日志：左侧展示运行流程及错误，右侧展示编译得到的汇编代码。



图5-10. 编译展示部分的运行日志

1. 词法分析：左侧展示符号表，右侧展示词法单元序列。



图5-11. 编译展示部分的词法分析

1. 语法分析：左侧展示LR1项目集簇，右侧展示推导过程。



图5-12. 编译展示部分的语法分析

1. 语法树：展示语法树。

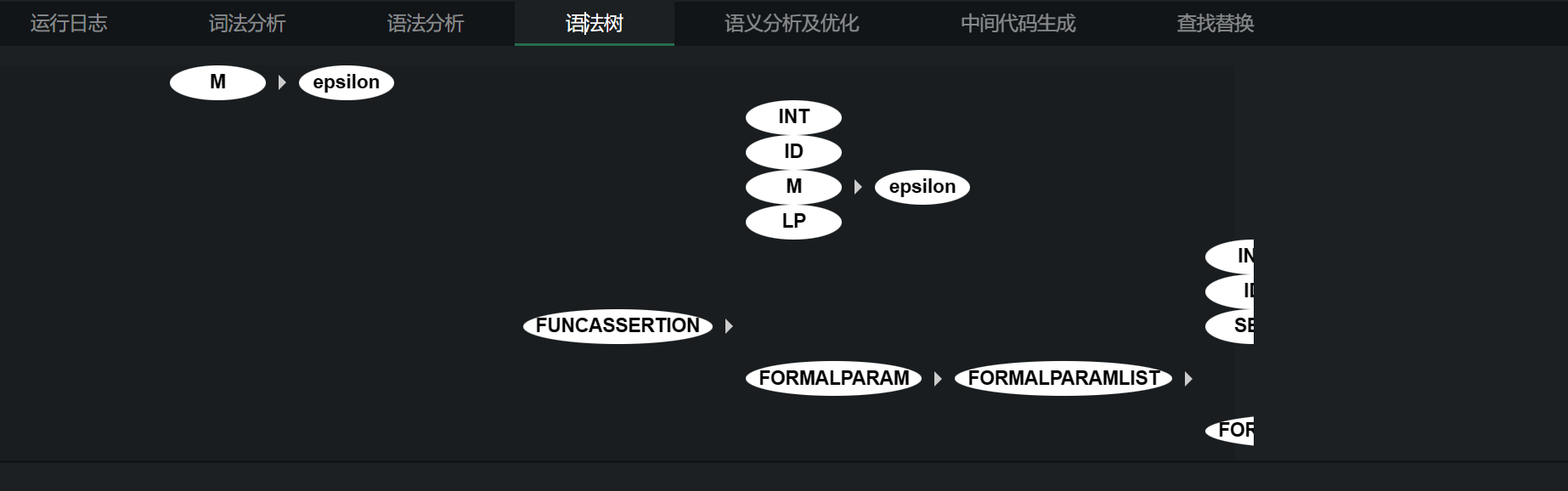


图5-13. 编译展示部分的语法树

1. 语义分析及优化：左侧展示中间代码，右侧展示优化前的中间代码。



图5-13. 编译展示部分的语法树

1. 中间代码生成



图5-13. 编译展示部分的语法树

1. 查找替换：查找当前文件中所有值为”int”的字符串并替换为”var”。

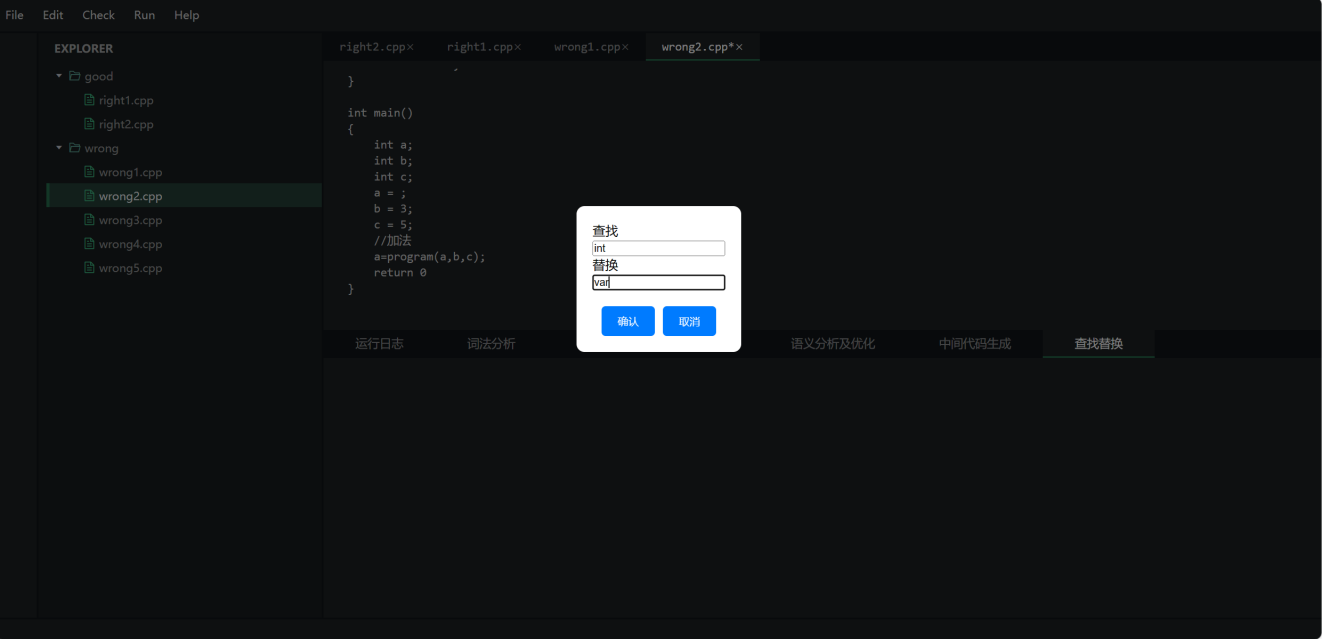


图5-14. 查找替换弹窗



图5-15. 查找结果

# 6 概要分析

## 6.1 遇到的问题及解决方法

1. 问题1：输出的寄存器分配表中RVALUE的每个$tx都为空。

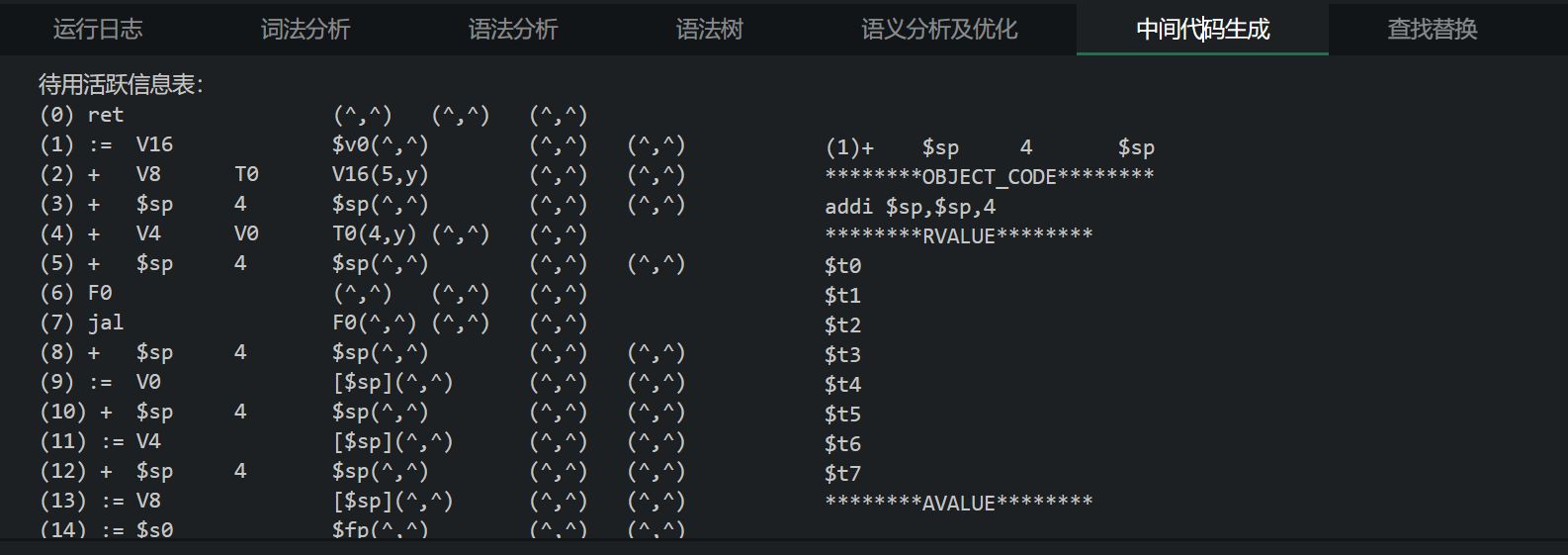


图6-1. 问题1出错截图

解决方法：出现这个问题的原因是java的语法特点，在进行赋值的时候，默认是进行浅拷贝，也就是类似与引用，例如a=b，如果修改b的值，a的值也会随之改变。因此在生成目标代码的最后，我将函数中tmp变量clear之后，RVALUE中的历史全会被clear，因此输出的结果为每个$tx都为空。

之所以出现这个问题，是因为在AnalysisHistoryItem类中，全参数初始化我原本使用@AllArgsConstructor中默认全参构造，但是其对每个类变量的赋值都是引用，因此会出现上述问题。在修改为new新的变量后，这里并不是引用，而是深拷贝，就不会出现上述的问题了。

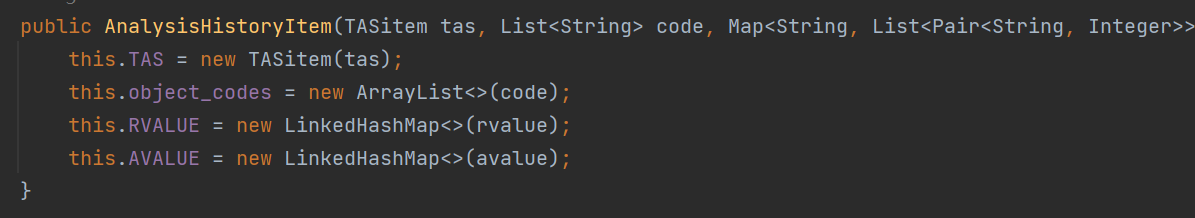


图6-2. 问题1解决方法

1. 问题2：可视化过程中文件树无法初始化，无法显示文件名以及标识。

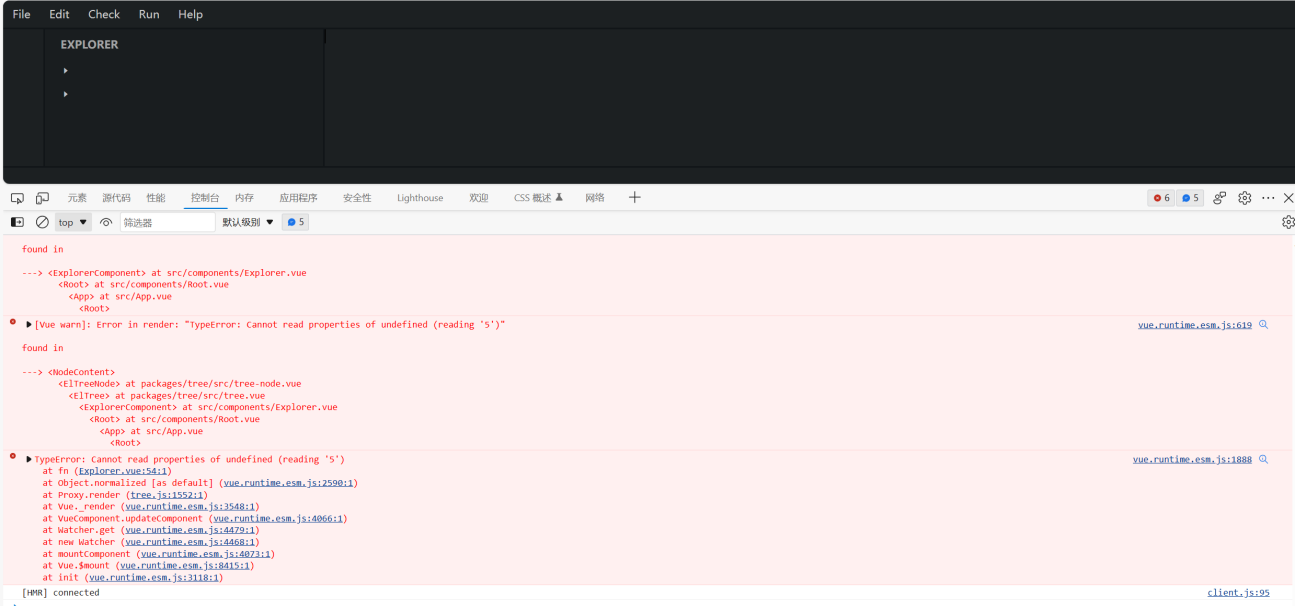


图6-3. 问题2出错截图

解决方法：出现这个问题的原因是vue在初始渲染界面时，我需要的文件树是需要从后端通过Fetch API来获取的，这会出现时间差。也就是说在网页初始化完成时，我还没获取到文件树的值，因此就无法显示文件树。解决问题的方法有两种，一是在初始化变量的时候不使用函数赋值初始化，而是直接赋值初始化，第二种方法是实时监视文件数信息对应的变量，在它第一次发生变化的时候，对文件树重新赋值并刷新界面。出于工作量的考量，我选择了前者，也可以很好的解决问题。

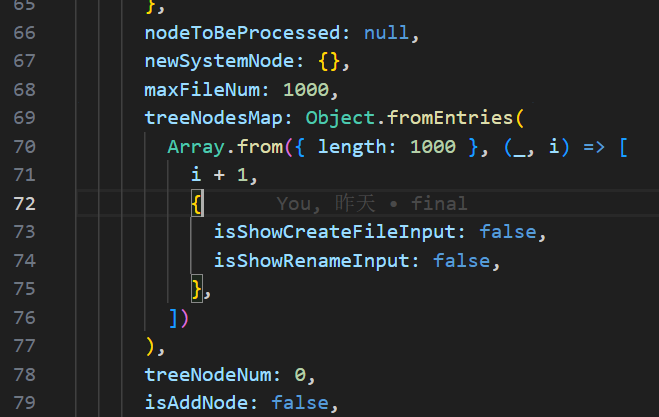


图6-4. 问题2解决方法

## 6.2 思考

在完成本次类C编译器项目中，不仅涉及编译功能的涉及，还要考虑前后端的交互设计和UI设计。在编译功能方面，考虑到java与C语言联合编译的困难性，我将上学期的词法分析、语法分析和语义分析重新用java实现了一遍，将上学期的多次编译修改为一次编译，将这三个阶段进行了有机的结合，使得这三个部分不是分离的，而是相辅相成的，极大提高了代码的效率和逻辑性。之后在此基础上，我完成了优化部分的编写，将原本冗余的四元式通过分块后对代码的分析，成功简化了程序，同时对于最后的目标代码生成有很大的帮助。最后在四元式的基础上完成了目标代码生成，并在Mars上对生成的代码进行了测试并得到了正确的结果。

同时，在进行前后端UI设计和交互设计时，我即考虑了编译后各个阶段的展示设计，同时考虑了用户在使用程序时，可能需要在写代码的时候添加tab键、剪切复制粘贴来提高写代码的效率，同时提供从本地上传文件到网页，避免在线写代码带来的困难。另外，以往用QT编写的程序无法实现文件的离线存储，针对这一点，我设计了对应的文件数据库，实现了文件的增删查改。仿照vscode页面的UI，我设计出以暗色调为主的UI设计，同时墨绿色的穿插使用提到了页面颜色的丰富度，使得整体设计较为出色。

## 6.3 对课程的认识

通过本次类C语言编译器项目的编写，我成功地在之前完成的中间代码生成器的基础上，使用java语言重构C++代码并添加剩余编译部分，并添加了网页端的交互设计。在这其中，我通过编写代码，实际地完成各类算法的实现，在调试错误的同时有效地补充了理论学习中对实践技术思考的缺乏，同时一方面进一步巩固了理论知识，另一方面也在功能完善的过程中，提高了代码编写能力以及错误调试能力。更为重要的是，在进行后续功能的实现与完善的过程中，我得以从另一个视角审视之前的代码架构以及模块设计，进而更好地发现模块组织以及接口设计等方面的不合理之处，并加以改进，最终实现了编译阶段完整，并添加前后端的类C编译器。

最重要的是，通过这一次项目，我体验了从设计文法，到逐步完成编译的每一步骤，逐渐领会编译原理的核心思想，在面对问题的时候冷静解决，完成可视化的设计。在增强对编译原理知识的认识的同时增强了我的java语言编程以及springboot+vue框架的使用，这门课对我的帮助意义重大，我从中收获颇多，感谢卫老师和各位助教学长对我的帮助。

## 参考文献

[1]万峰松,杨得胜,郭星,吴建国.类C编译器设计初探[J].计算技术与自动化,2008(03):133-136.

[2]王博俊,张宇.自己动手写编译器、链接器[M].清华大学出版社,2015-1

[3]Keith Cooper.Engineering a Compiler[M].人民邮电出版社,2012-12

[4]Christopher W. Fraser.A Retargetable C Compiler [M].电子工业出版社,2015-1