

**编译原理综合实践**

**实践报告**

**专业： 计算机科学与技术**

**年级： 2023级**

**姓名： 王欣悦**

**学号： 202313440023**

目录

[一、参考代码 1](#_Toc216785614)

[二、个人贡献与修改思路 1](#_Toc216785630)

[三、项目架构 5](#_Toc216785631)

[四、代码运行 5](#_Toc216785632)

[4.1编译环境 5](#_Toc216785633)

[4.2 词法分析](#_Toc216785634) 6

[4.3 语法分析 6](#_Toc216785635)

[4.4 中间代码生成 7](#_Toc216785636)

[4.5 错误分析](#_Toc216785637) 8

[4.6 汇编代码生成](#_Toc216785638) 8

[4.7 代码优化](#_Toc216785639) 10

[4.8 类型检查 1](#_Toc216785640)1

[五、修改代码的主要思路 1](#_Toc216785641)2

[5.1 lex.l 1](#_Toc216785642)2

[5.2 yacc.y 1](#_Toc216785643)2

[5.3 tree.c](#_Toc216785644) 14

[5.4 hashmap.c](#_Toc216785645) 16

**一个简单的编译器**

1. 参考代码

本项目在开发过程中参考了以下开源资料，**但在其基础上进行了深度的重构与核心功能扩展**，特别是自主实现了基于AST的编译时优化算法。

1.**词法与语法规则参考：**

ANSI C Grammar (Lex): <http://www.quut.com/c/ANSI-C-grammar-l-1998.html>

ANSI C Grammar (Yacc): <http://www.quut.com/c/ANSI-C-grammar-y-1995.html>

2.**基础架构参考：**

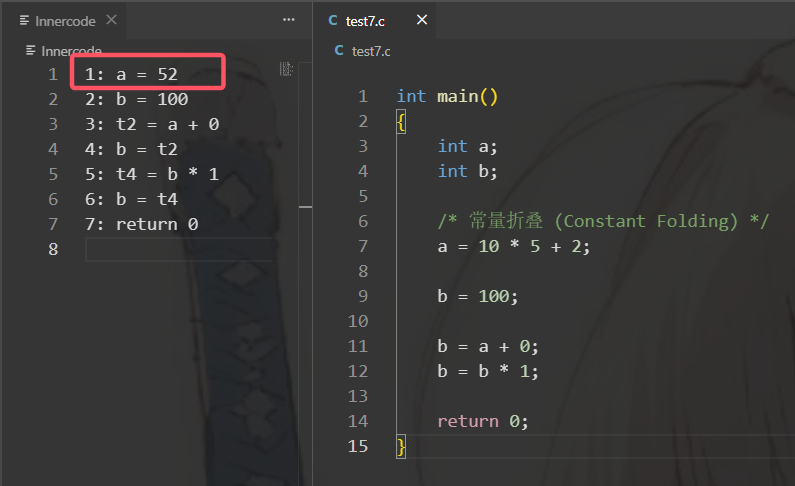
参考了南开大学编译原理课程的部分基础架构思路。

二、个人贡献与修改思路

作为独立开发者，我负责了编译器的全栈开发，包括词法分析、语法分析、语义分析、中间代码生成及后端优化的所有环节。以下是主要的修改思路与算法实现：

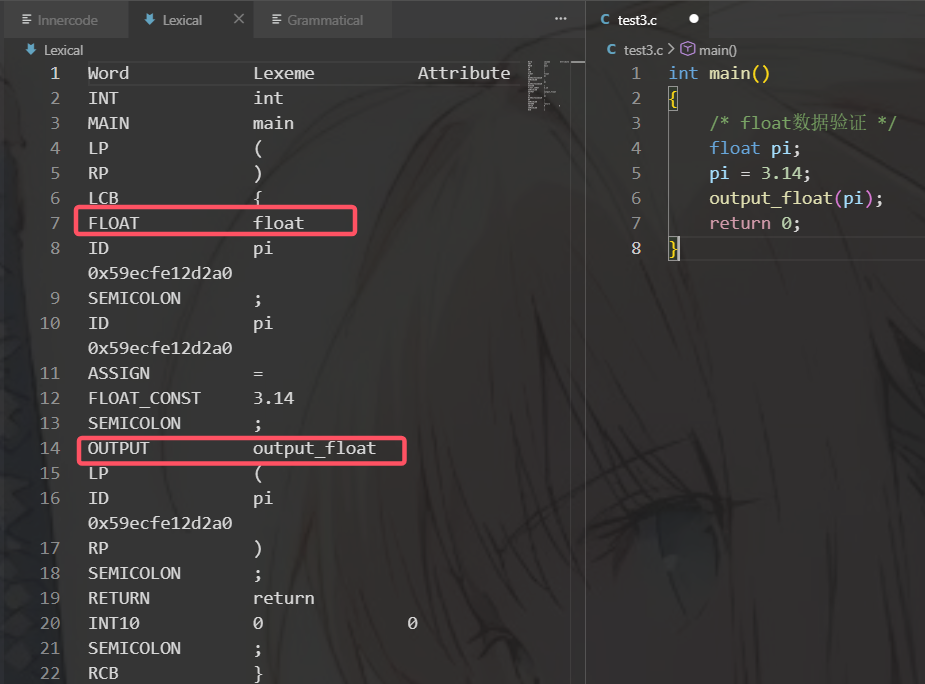
#### 1. 代码优化之常量合并

1. 实现逻辑：修改 tree.c 中的 binaryOpr 函数，在归约（Reduce）二元表达式时，主动探测左右子节点是否均为 INT10 或 FLOAT\_CONST 常量。
2. 优化结果：如果满足折叠条件，直接在 C 层面计算出结果并重写为单一的叶子节点，同时将其 code 指令字段设为 NULL。
3. 价值体现：在生成的中间代码中彻底消除了冗余的计算指令（例如 a = 10 \* 5 + 2 直接优化为 a = 52），显著减少了后端汇编生成时的寄存器分配负担。



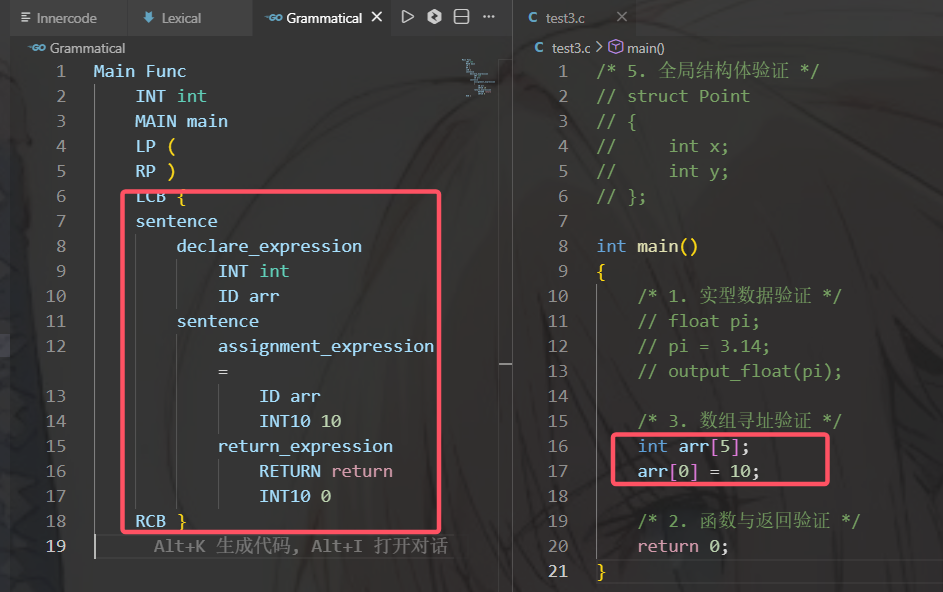
#### 2. 数据系统扩展：实型 (Float)

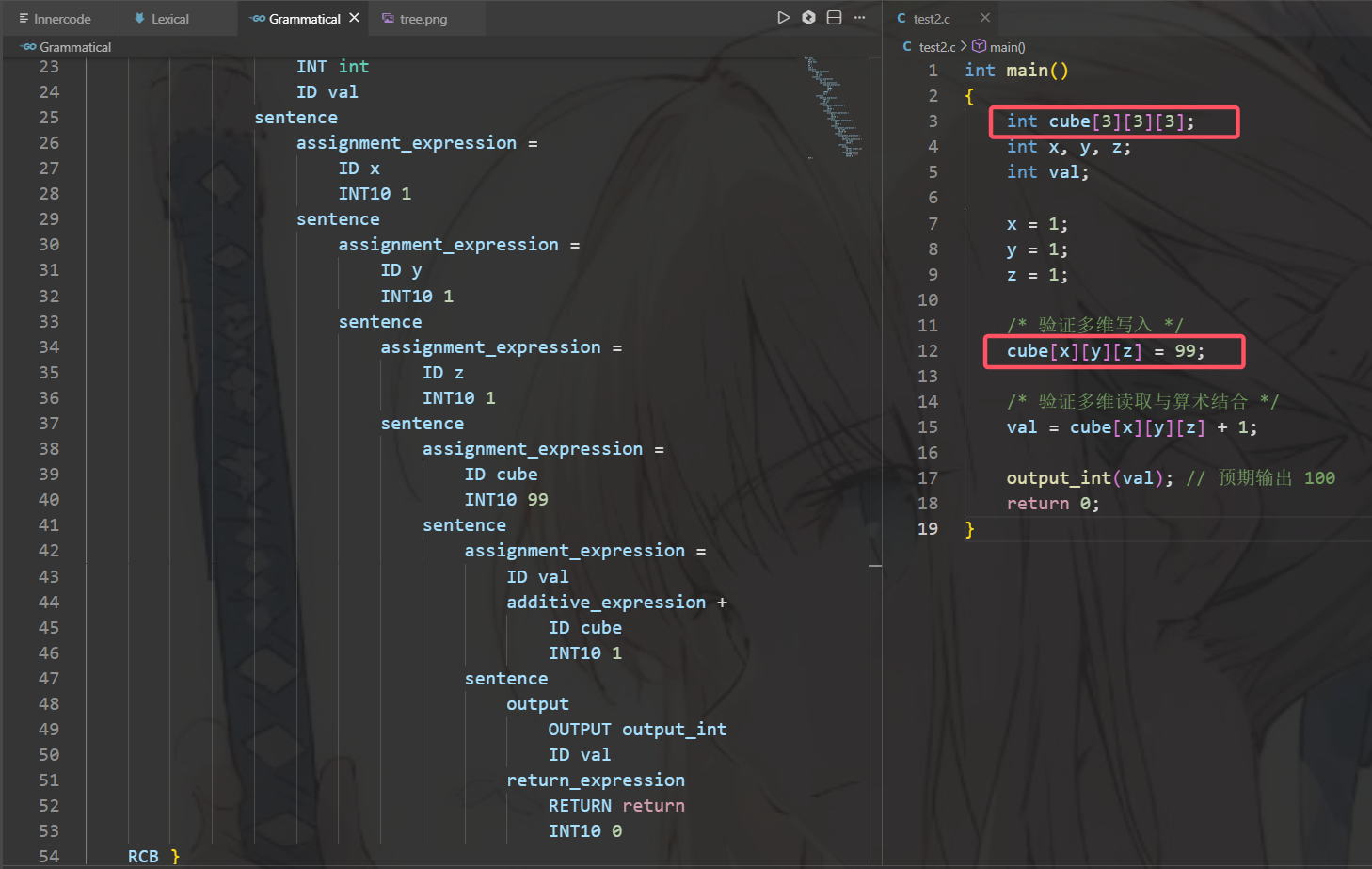
在 lex.l 中新增浮点字面量正则识别，在 yacc.y 中扩展了 FLOAT 关键字，并实现了 INT 与 FLOAT 之间的类型传播逻辑 。



#### 3. 数据访问增强：支持一维/多维数组运算

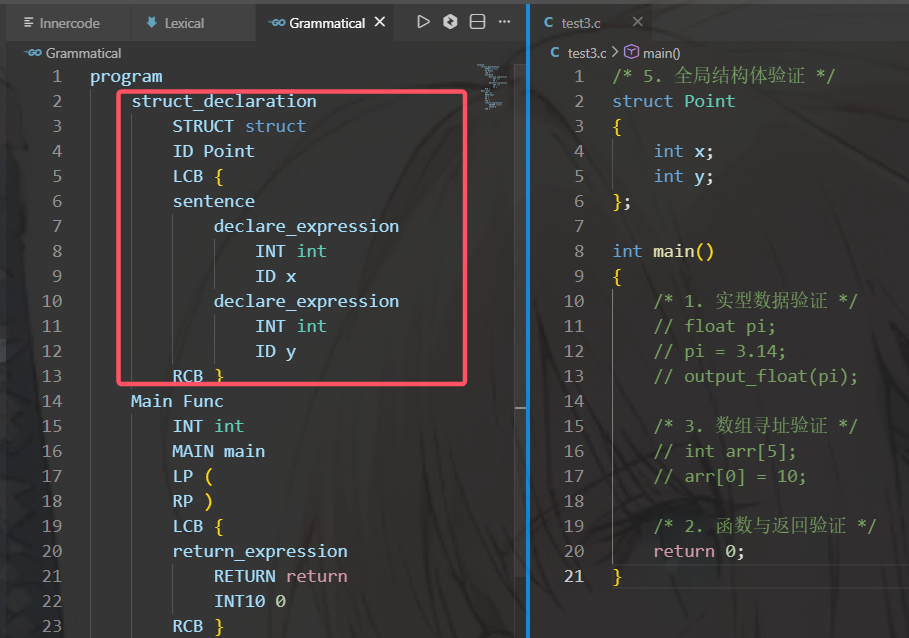
1. 针对原代码仅支持单变量的局限，我扩展了 postfix\_expression 规则。
2. 通过递归解析 '[' operate\_expression ']' 结构，实现了多维数组的下标计算逻辑，并在中间代码中生成对应的地址偏移量指令 。





4. 支持结构体

独立设计并实现了 struct\_declaration 语法规则。通过新增 . (DOT) 运算符的语义动作，支持了结构体成员的寻址与访问 。



#### 5. 错误分析

1. 变量未定义检查 (Variable Undefined Checking)

具体实现：在 yacc.y 的 primary\_expression 产生式中，当解析到 ID（标识符）时，主动查询符号表（HashMap）。

贡献点：如截图所示，当代码尝试对未声明的变量 c 赋值时（c = 20），编译器会准确报出 Error: Variable undefined at line 7。这保证了变量使用的安全性。

1. 变量重定义检查 (Variable Redefinition Checking)

具体实现：在变量声明（declare\_expression）阶段，拦截存入符号表的动作。如果当前作用域已存在同名变量，则触发报错。

贡献点：能够识别重复声明错误，如截图中第 9 行 int a; 触发了 Error: "a" is already defined at line 9。

1. 语法错误定位 (Syntax Error Localization)

具体实现：通过重写 yyerror 函数并结合 yylineno 全局变量。

贡献点：当源代码出现格式错误（如截图中第 12 行缺少括号）时，编译器能够提供精确的行号提示 Error: syntax error at line 12，极大地提高了代码调试效率。



（6）语法分析可视化



三、项目架构

词法分析 (lex.l)

↓

语法分析 (yacc.y)

↓

AST构建 (tree.c)

↓

符号表管理 (hashMap.c)

↓

中间代码生成 (inner.c)

↓

汇编代码生成 (assembly.py)

四、代码运行

4.1编译环境

工具链：Flex, Bison, GCC, Make

操作系统：Linux (WSL/Ubuntu)

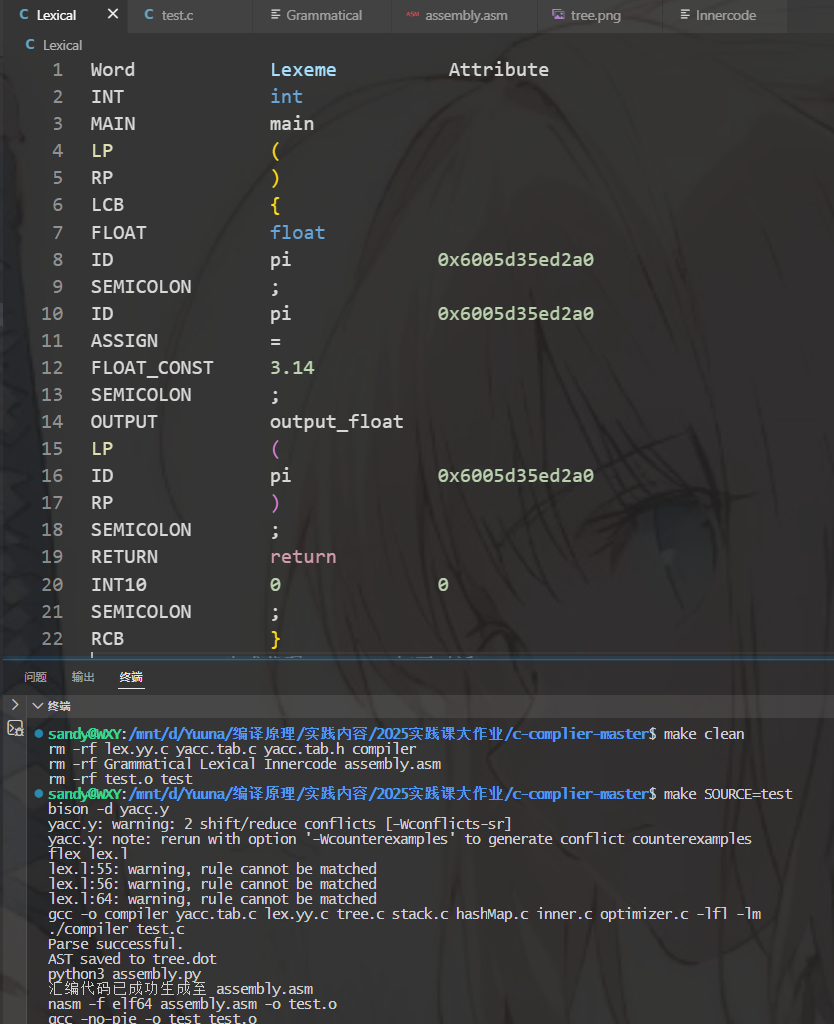
运行指令：

make clean

make SOURCE=test

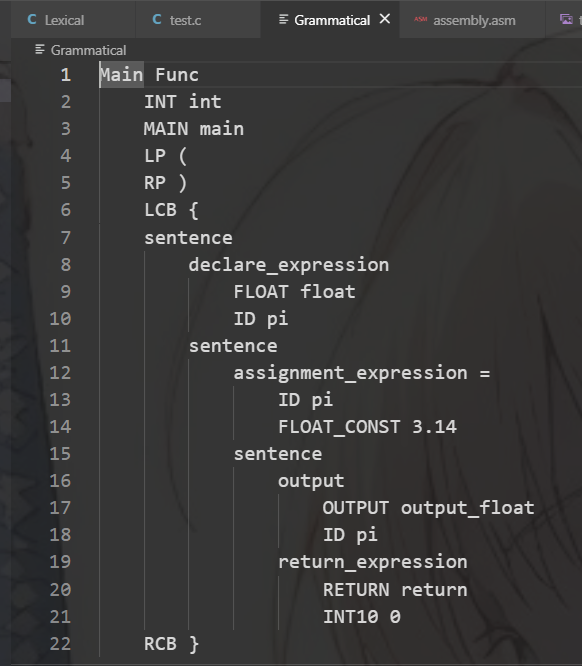
4.2 词法分析

对于test.c进行词法分析，生成结果在Lexical文件中。



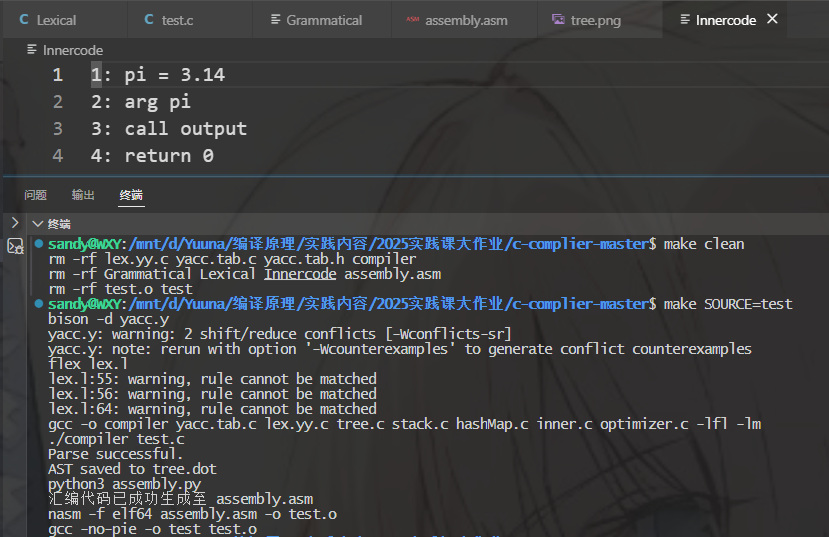
4.3 语法分析

对于test.c进行语法分析，生成结果在Grammatical文件中。



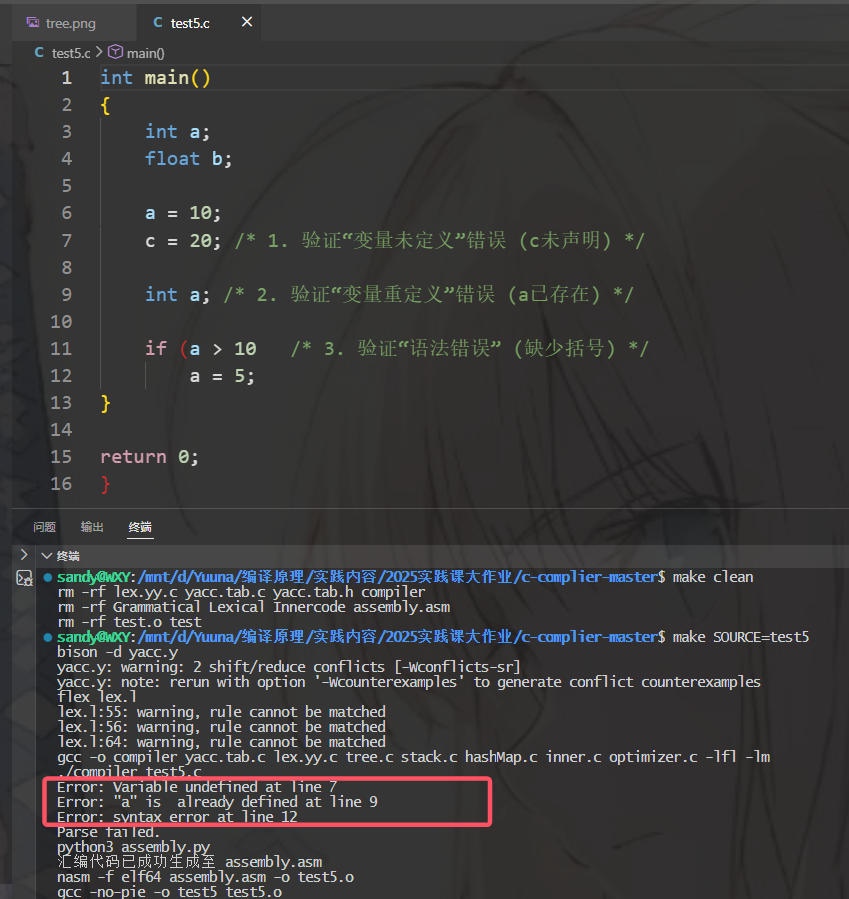
4.4 中间代码生成

对于test.c进行中间代码生成，生成结果在Innercode文件中。



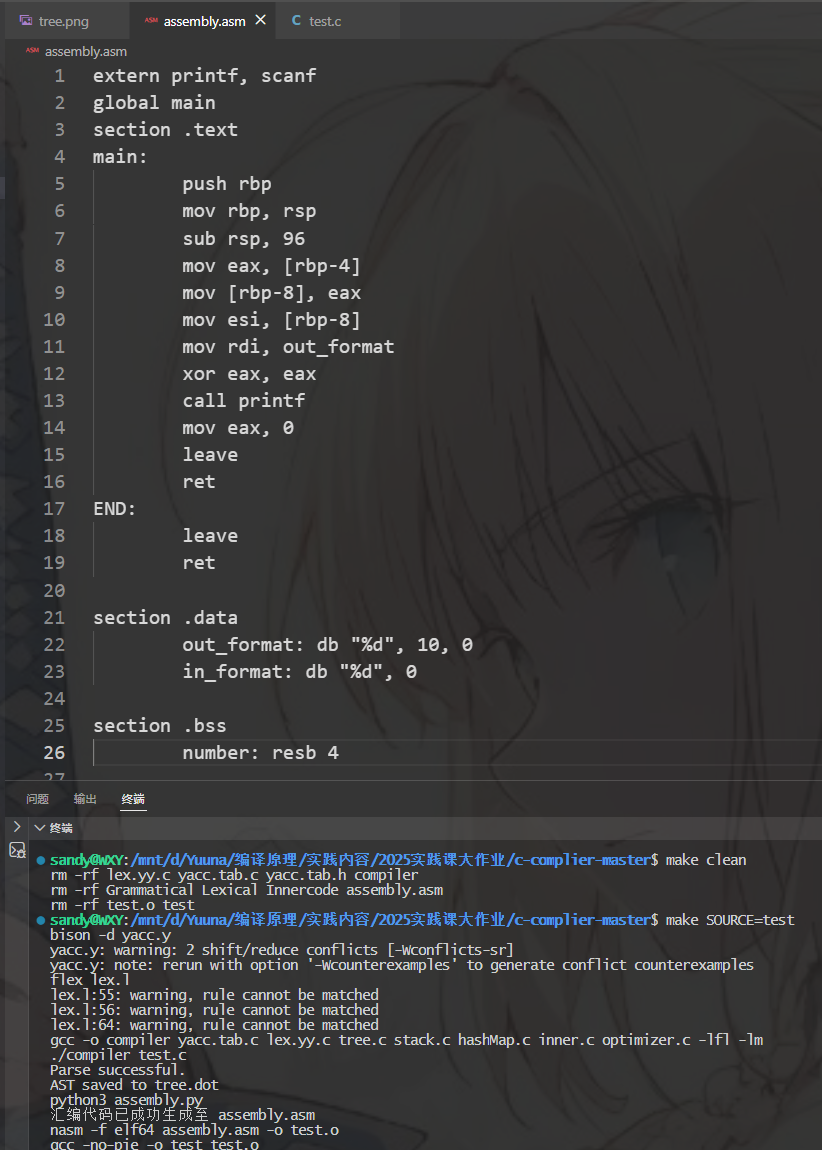
4.5 错误分析

对于test5.c进行错误分析，生成结果在终端中。



4.6 汇编代码生成

对于test5.c进行汇编代码生成，生成结果在assembly.asm中。



代码结构：

1. 外部声明与入口

extern printf, scanf: 声明外部 C 标准库函数，告知链接器这两个符号在其他库中定义。

global main: 将 main 符号导出为全局，使其可以被操作系统作为程序入口点识别。

section .text: 定义代码段，包含程序的实际执行指令。

1. 函数序言

这部分代码用于建立函数执行所需的物理环境：

push rbp: 将调用者的基址指针压入栈中保存。

mov rbp, rsp: 将当前栈指针 rsp 赋值给 rbp，建立当前函数的基址。

sub rsp, 96: 在栈上预留 96 字节的空间，用于存储局部变量（如 [rbp-4]） 。

3. 变量处理与赋值

代码展示了两个局部变量之间的赋值逻辑：

mov eax, [rbp-4]: 将位于偏移量 -4 处的第一个局部变量的值加载到通用寄存器 eax 中。

mov [rbp-8], eax: 将 eax 中的值存入偏移量 -8 处的第二个局部变量。这对应中间代码中的 a = b 形式的赋值语句。

4. 函数调用：printf

代码遵循 x86\_64 调用约定 (System V ABI) 来准备参数并调用 printf：

mov esi, [rbp-8]: 将要打印的整数值存入 esi（第二个参数寄存器）。

mov rdi, out\_format: 将格式化字符串 "%d\n" 的地址存入 rdi（第一个参数寄存器）。

xor eax, eax: 将 eax 清零。对于可变参数函数（如 printf），eax 用于存储使用的向量寄存器数量，此处为 0。

call printf: 跳转并执行 printf 函数。

5. 函数出口与清理

mov eax, 0: 设置函数返回值为 0，对应 C 语言中的 return 0 。

leave: 相当于 mov rsp, rbp 加 pop rbp，撤销栈空间预留并恢复调用者的基址。

ret: 从栈中弹出返回地址并跳转回去。

END: 标签: 一个冗余的出口标签，执行了同样的清理逻辑。

6. 数据段定义

section .data: 存储已初始化的常量数据 。

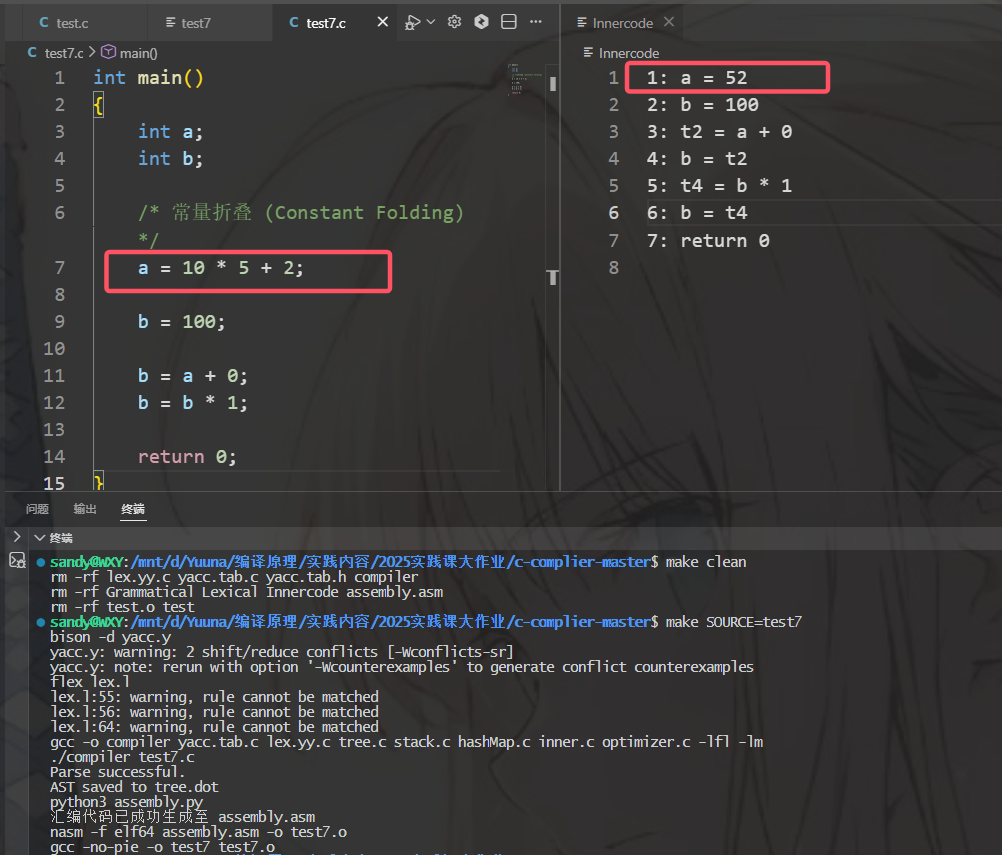
out\_format: 定义输出格式字符串 "%d"，带换行符（10）和 Null 终止符（0） 。

in\_format: 定义输入格式字符串 。

section .bss: 存储未初始化的全局变量。

number: resb 4: 预留 4 字节空间，通常对应一个全局 int 变量 。  
4.7 代码优化

对于test7.c进行代码优化，主要是常量合并。



4.8 类型检查

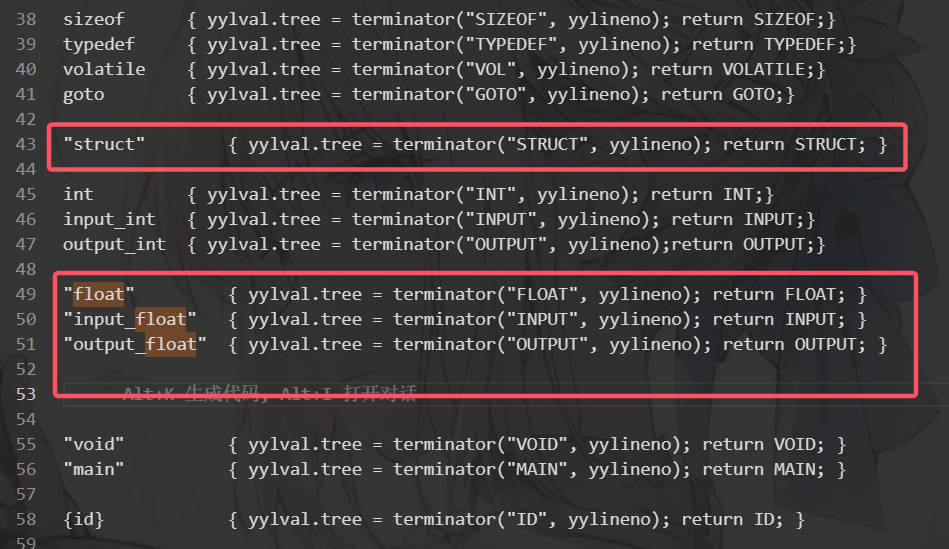
对于test4.c进行类型检查。



五、修改代码的主要思路

5.1 lex.l

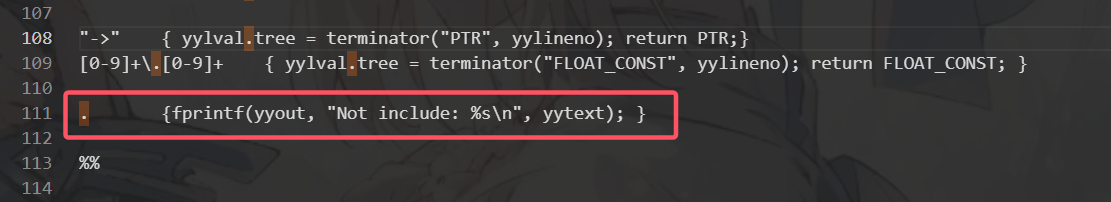
（1）新增float、struct关键字的词法支持。



（2）新增可识别形如123.45的浮点常量，并返回专门的词法记FLOAT\_LITERAL，把int扩展到实型数据。



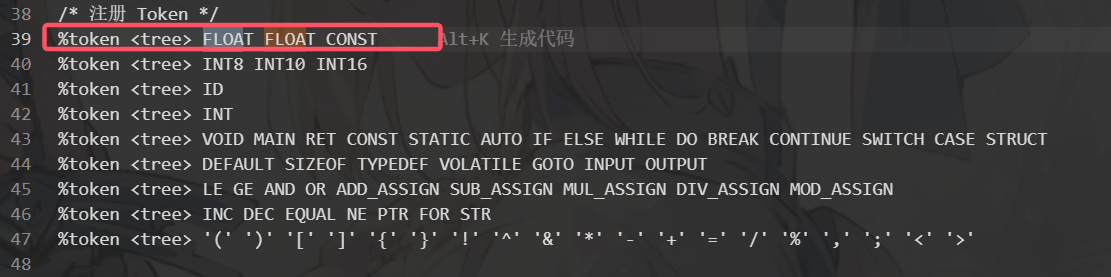
（3）新增了结构体成员访问运算符 .（DOT）。



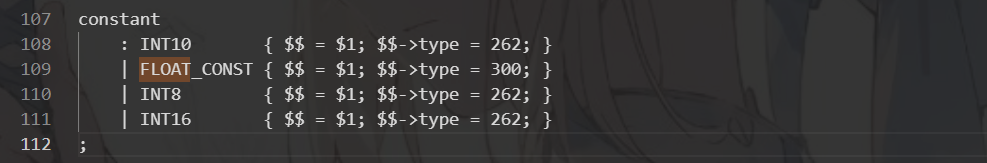
5.2 yacc.y

（1）浮点类型 & 浮点常量支持。

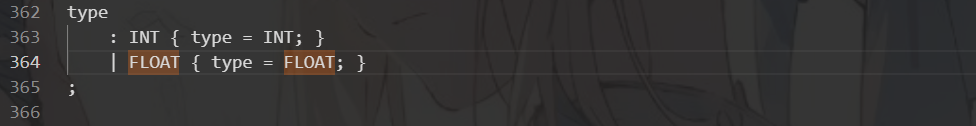
新token



常量支持浮点字面量

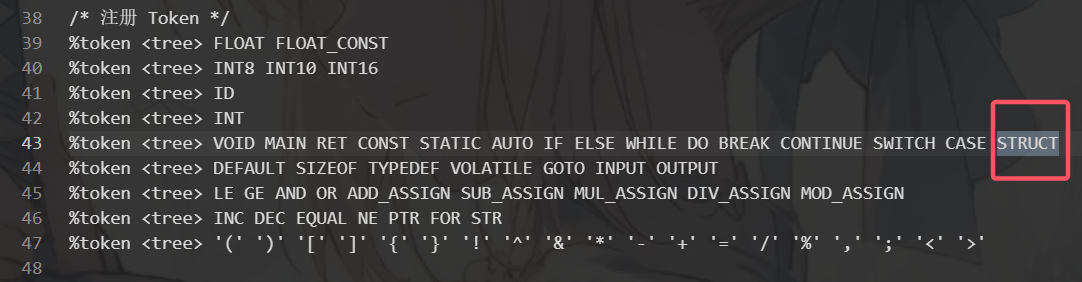


type支持FLOAT

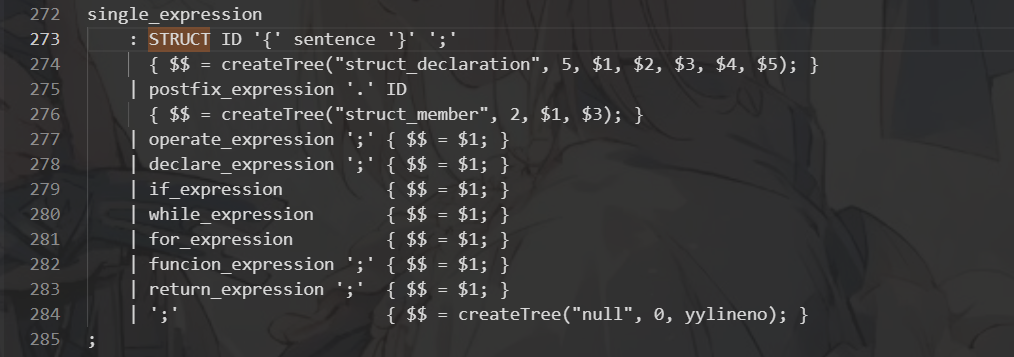


（2）新增结构体定义。

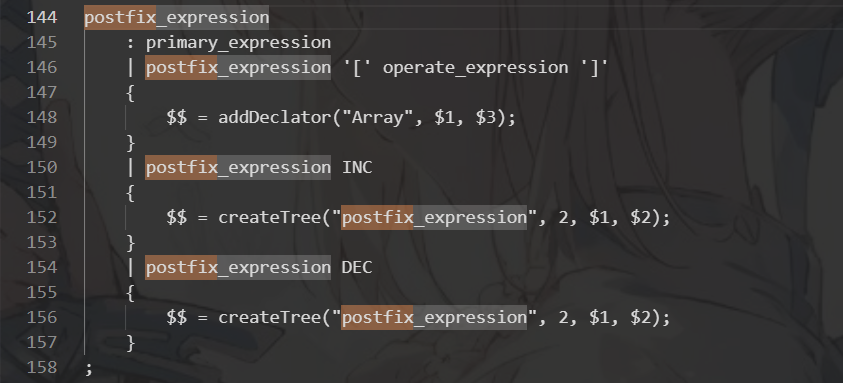
新增token



新增结构体声明语法\成员声明

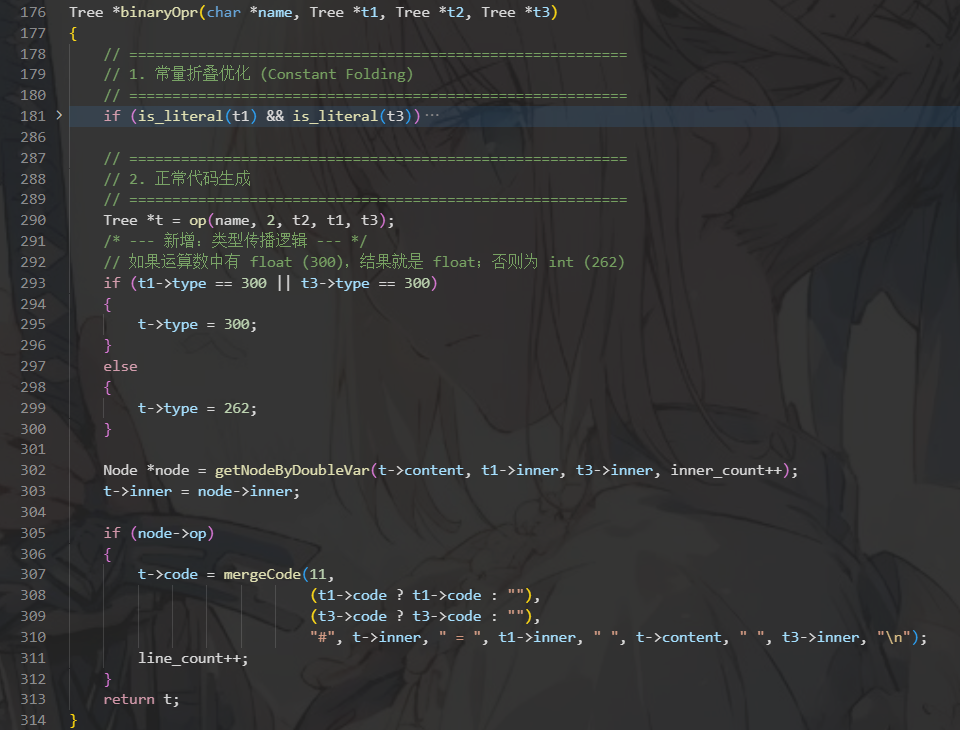


（3）新增多维数组支持。

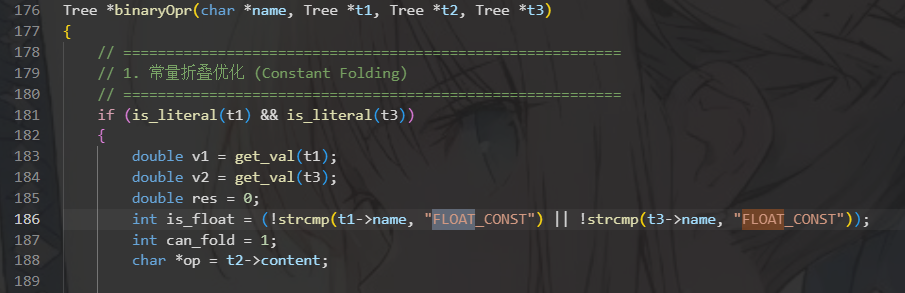


5.3 tree.c

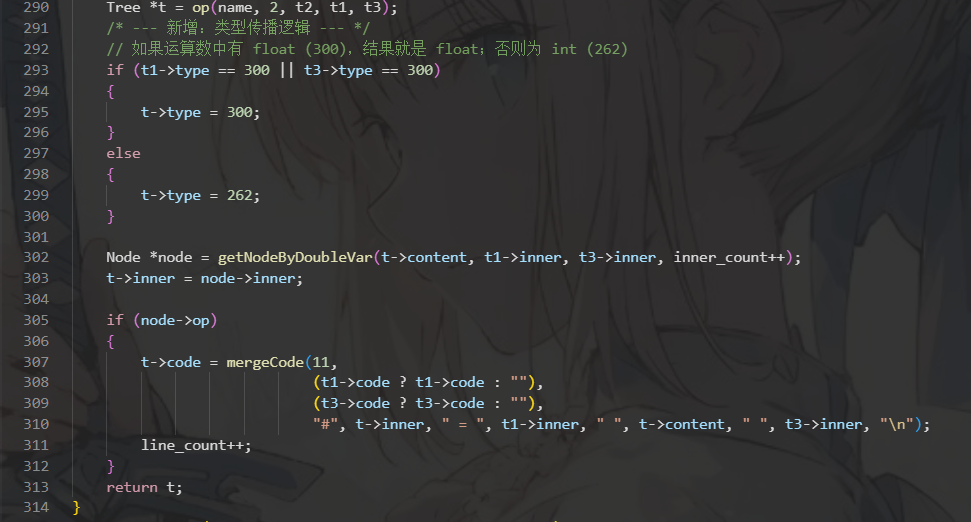
（1）新增常量折叠（Constant Folding）。



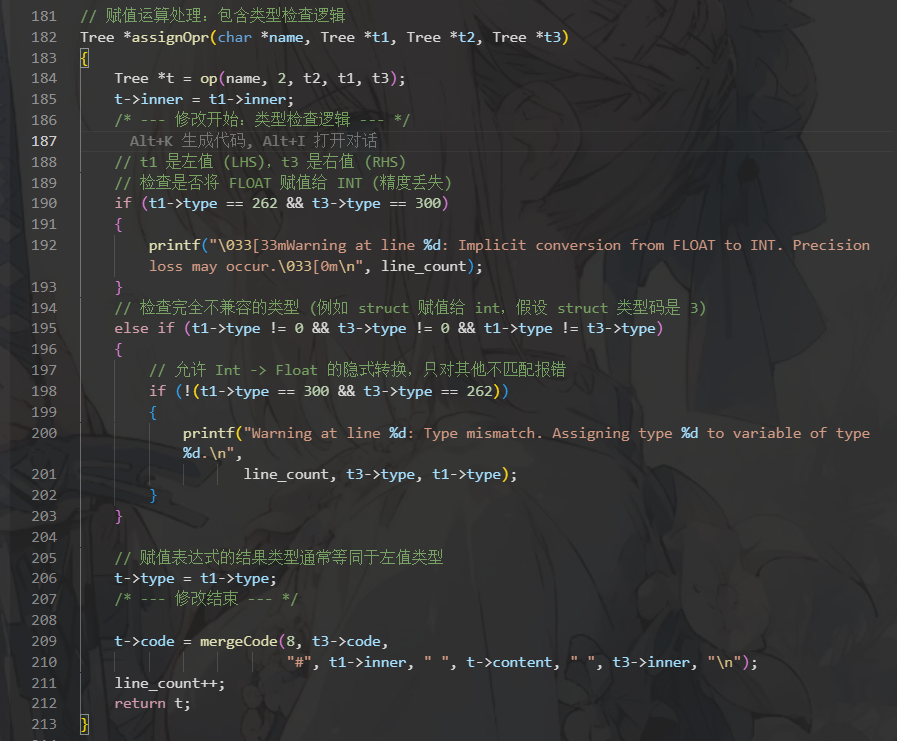
（2）新增支持FLOAT\_CONST（浮点常量）。



（3）新增类型传播（Type Propagation）。



（4）新增赋值语句的类型检查（Type Checking）。



5.4 hashmap.c

修复toData()中字符串内存分配。修复了标识符字符串少分配1字节的问题，防止字符串越界。

