编译原理与设计实验报告

姓名：卜梦煜 学号：1120192419 班级：07111905

# 1. 实验名称

中间代码生成实验

# 2. 实验目的

（1）了解编译器中间代码表示形式和方法；

（2）掌握中间代码生成的相关技术和方法，设计并实现针对某种中间代码的编译器模块；

（3）掌握编译器从前端到后端各个模块的工作原理，中间代码生成模块与其他模块之间的交互过程。

# 3. 实验内容

以BIT-MiniCC的语义分析阶段的抽象语法树为输入，针对不同的语句类型，将其翻译为中间代码序列。选择四元式为中间代码格式，对具体AST结点四元式的定义见“实验过程与步骤”。

# 4. 实验环境

Intellij IDEA Community 2021.3.2

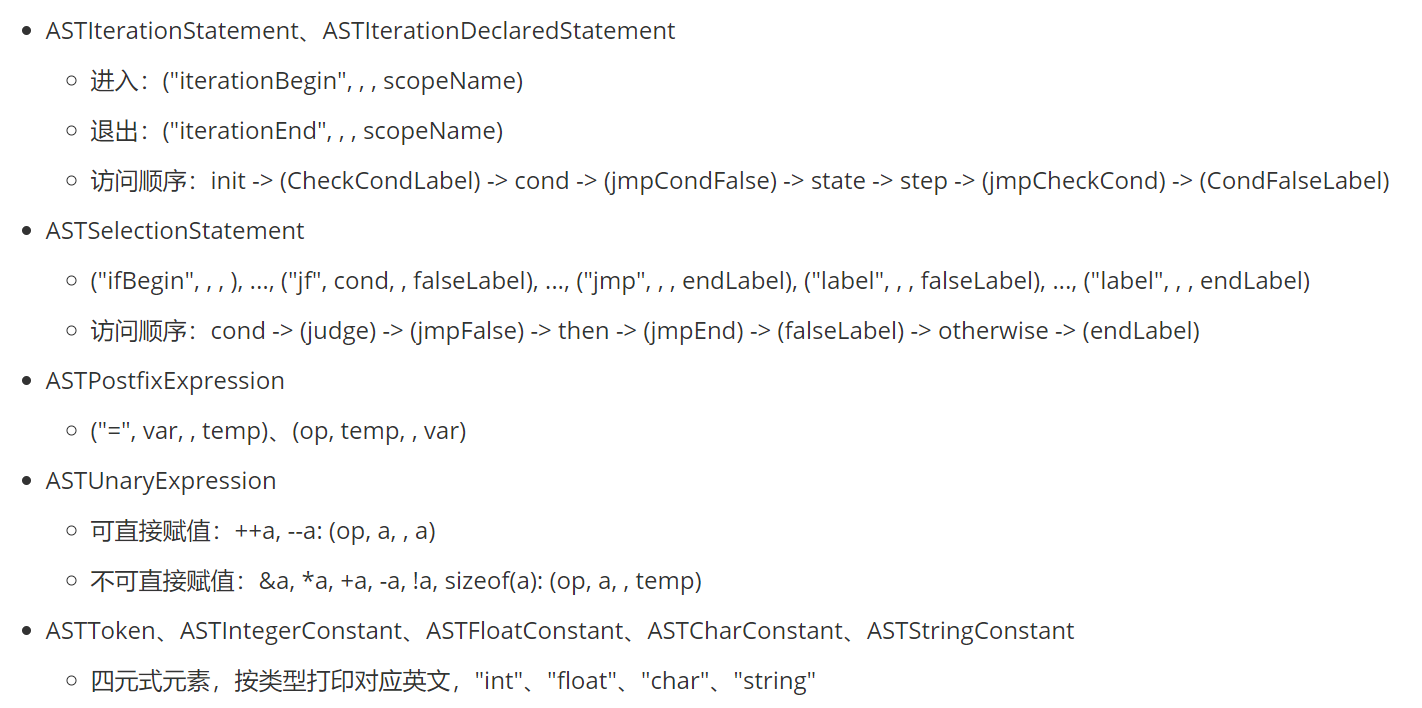
# 5. 实验过程与步骤

本实验首先针对每个AST结点定义四元式结构，之后在BITMiniCC的Visitor模式中编写结点访问方法与四元式生成方法，最后配置BITMiniCC并运行测试用例，验证四元式结构是否合理。

## 5.1 定义AST结点的四元式结构

四元式基础格式为(op, arg1, arg2, result)，针对具体AST结点设计四元式结构如下。



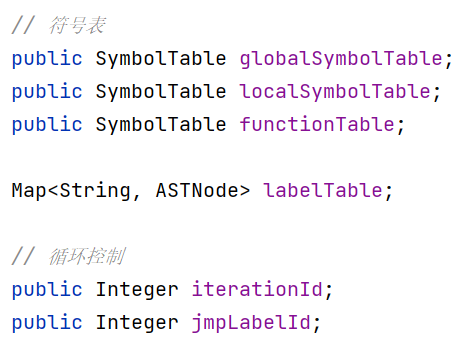


## 5.2 Visitor模式下编写代码

中间代码生成部分需要编写的代码包括AST结点属性访问，函数表、变量表、标签表、伪寄存器值等信息维护，四元式的生成等。此外还需编写四元式的打印方法。

（1）中间代码生成器

中间代码生成过程中需维护的信息包括全局变量表、局部变量表、函数表、标签表、循环作用域编号、跳转标签编号。定义的信息维护变量如下：



对每个AST结点，需编写的代码包括访问AST结点属性结点，维护信息，生成四元式。以ASTArrayDeclarator为例，编写代码如下：

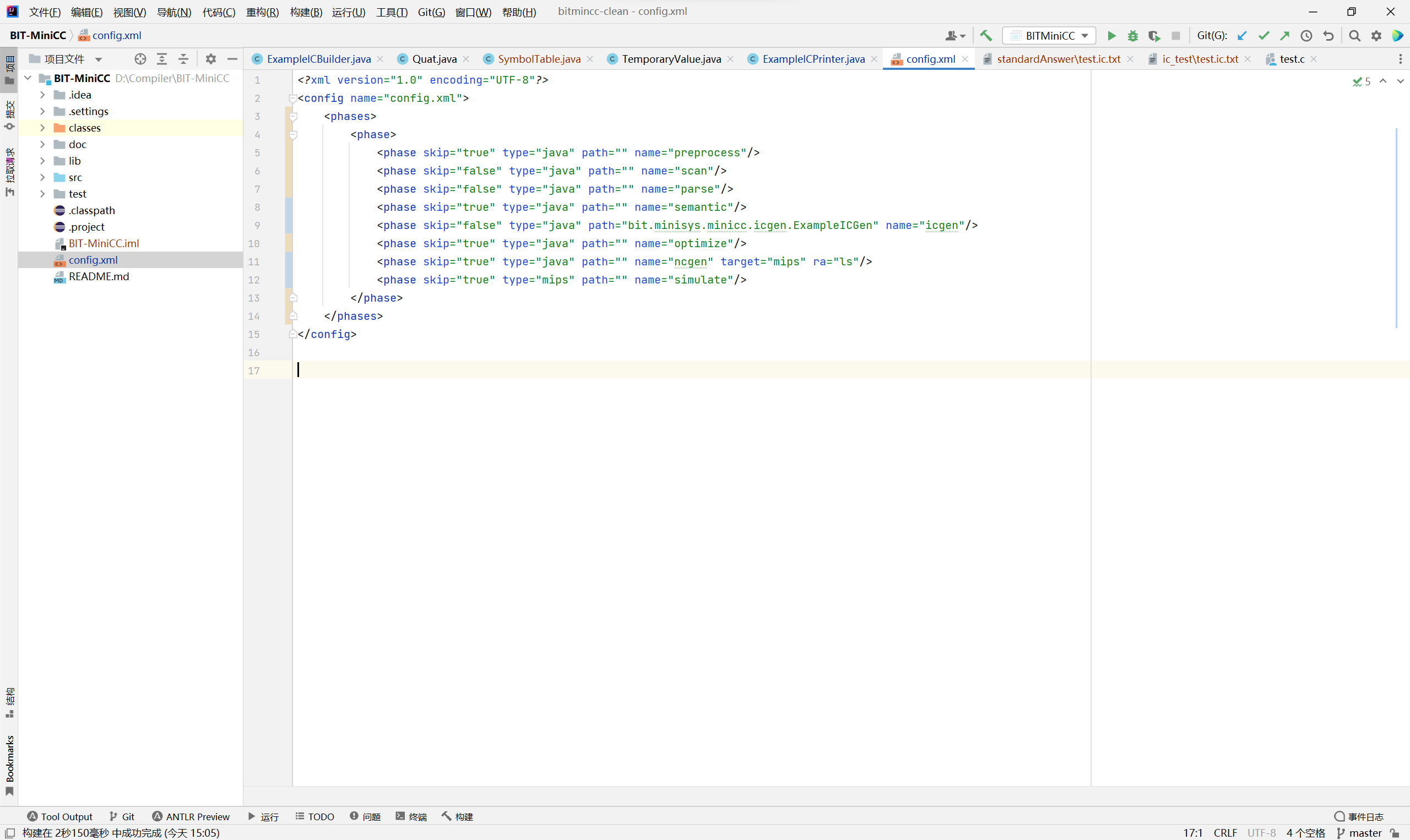


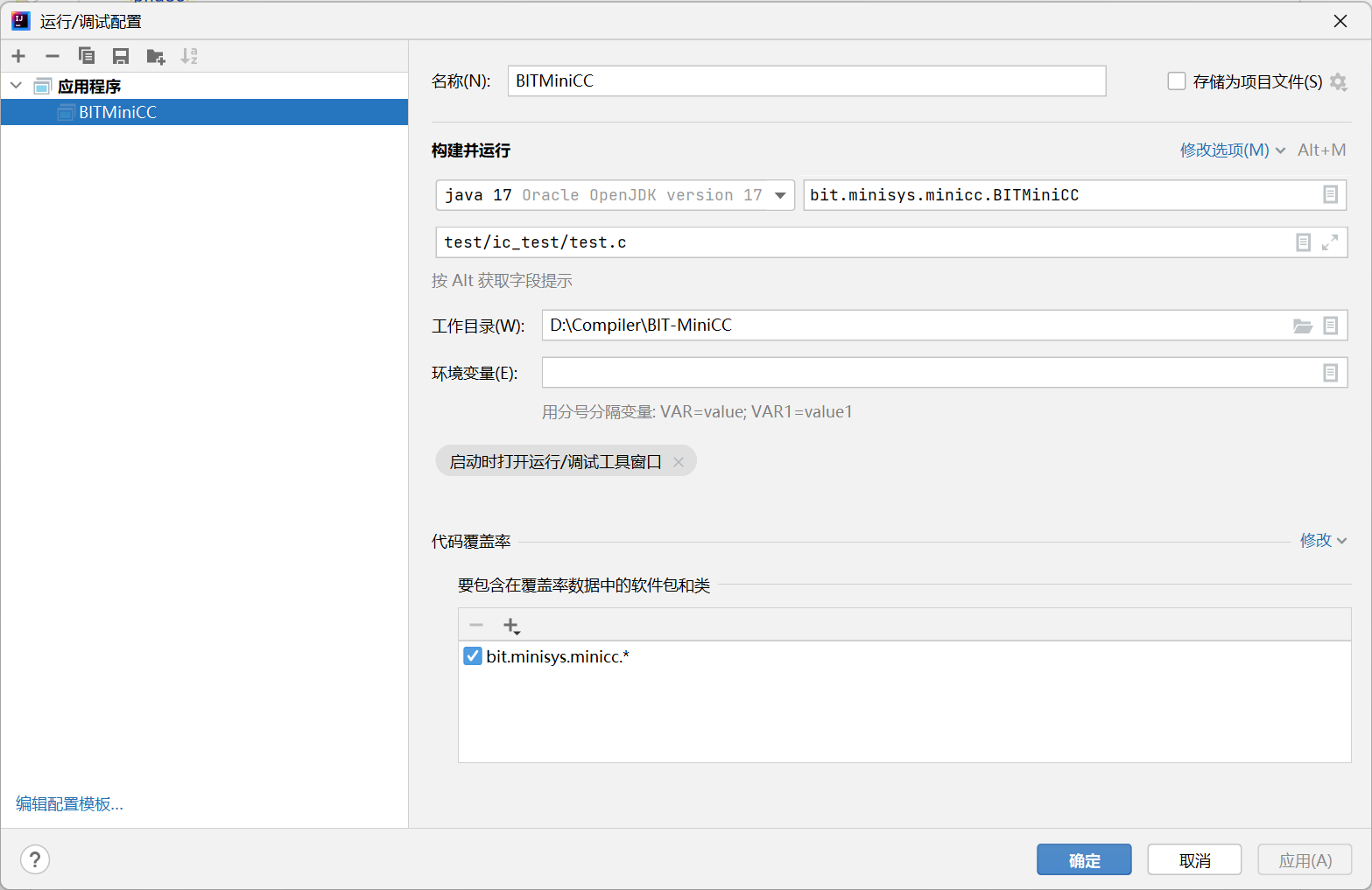
（2）打印四元式

打印四元式是从四元式变量中依次读取，需要编写的代码是每个AST结点的变量名，根据AST结点类型特判即可。

## 5.3 配置BITMiniCC，运行测试用例

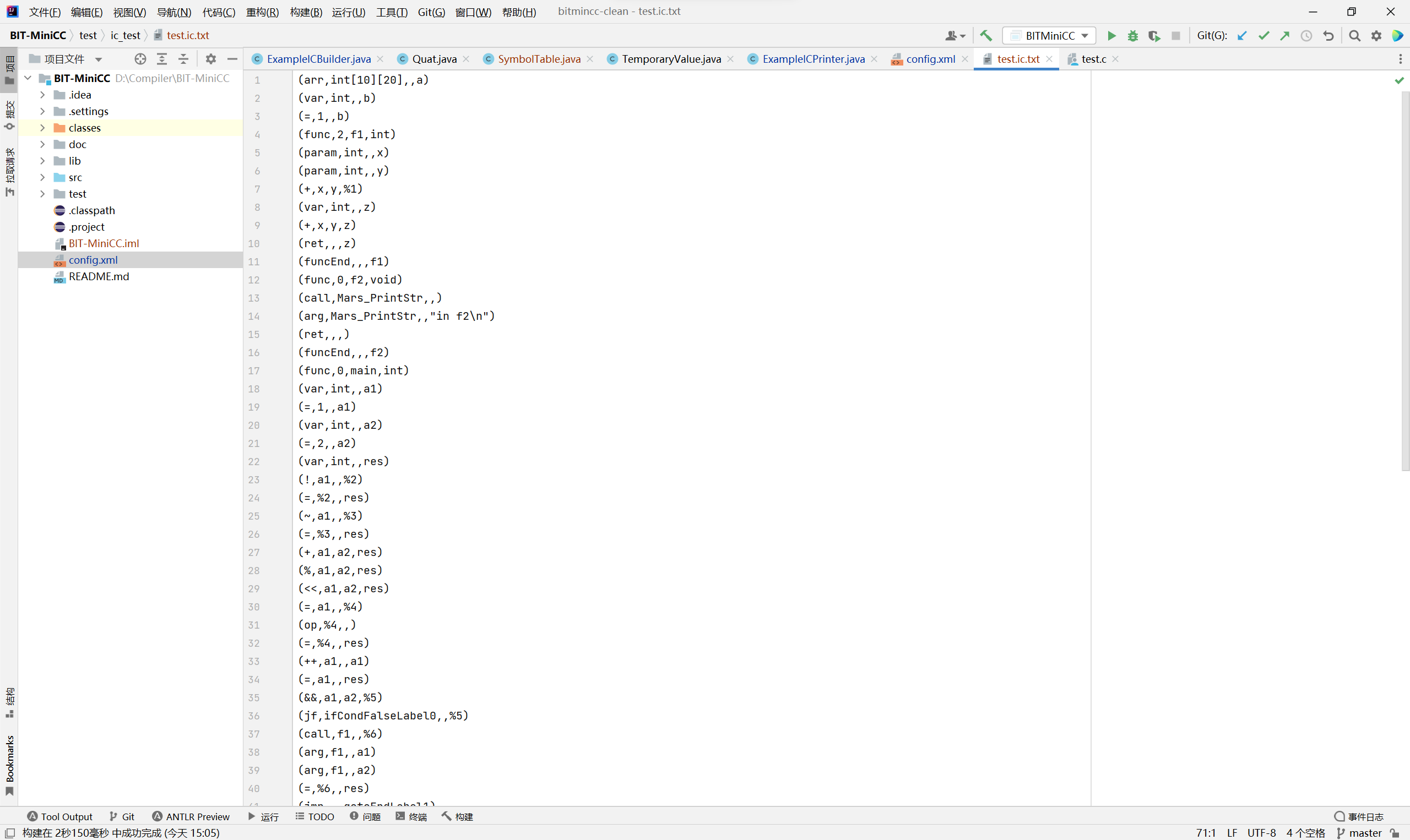
配置BITMiniCC配置文件，使用内置的词法分析器、语法分析器，自己编写的中间代码生成工具，并将测试用例路径写入运行配置。配置文件具体信息如下：





# 6. 实验结果与分析

BITMiniCC自带的中间代码生成测试文件test.c文件的运行结果如图。



分析可知，编写的中间代码生成器可以按照设计的四元式格式，正确生成各种AST结点的四元式。

部分c语句与生成的对应四元式举例如下：

|  |  |  |
| --- | --- | --- |
| **语句类型** | **语句** | **生成的四元式** |
| 变量定义 |  |  |
| 函数定义 |  |  |
| 赋值语句 |  |  |
| if语句 |  |  |
| for语句 |  |  |

# 6. 个人心得体会

本次实验是一个灵活性非常高的实验，中间代码格式的选择、具体AST结点的四元式定义都是可以自行设计和定义的，但不同的设计可能在编写代码难度上有所体现。我刚开始并没有很好的格式选择思路，参考了MAPPLE和LLVM的中间代码格式后，感觉有部分中间代码难以理解，因此最后选择参考两种中间代码，采用四元式的形式编写。我的实验收获如下。

（1）掌握了中间代码生成方法，能够使用四元式对C语言不同结构语句编写中间代码。

（2）对BITMiniCC的Visitor模式的理解进一步加深，能熟练掌握BITMiniCC各个模块的工作原理，以及中间代码生成模块与其他模块的交互过程。