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。

对AST结点的四元式：

- ASTArrayDeclarator
  - ("arr", type, , arrName)，其中type为复合变量类型，如int[10][20]
- ASTVariableDeclarator
  - ("var", type, , varName)
  - 有初始化：("=", tempValue, , varName)，tempValue按照ASTBinaryExpression计算得出
- ASTFunctionDeclarator
  - ("func", returnType, argNum, funcName)
- ASTParamsDeclarator
  - ("param", type, , argName)
- ASTArrayAccess
  - a[idx(1)]...[idx(n)]要先计算相对位置再取值
  - eg: ("*", idx(1), size(2), temp1), ("+", temp1, idx(2), temp1), ("=[]", temp1, a, temp2)
- ASTBinaryExpression
  - 赋值操作：("=", arg1, , res)
  - 双目运算操作：计算双目运算结果存到中间变量，最后赋值 (op, arg1, , res)
- ASTBreakStatement、ASTContinueStatement
- ASTGotoStatement
  - ("jmp", , label)
- ASTLabeledStatement
  - ("label", , , labelValue)
- ASTReturnStatement
  - 有返回值：("ret", , , returnValue)，无返回值：("ret", , ,)
- ASTFunctionCall
  - 函数调用：无返回值时("call", functionName, ,)，有返回值时("call", functionname, , tempReturnValue)
  - 参数：("arg", functionName, , arg)
- ASTIterationStatement、ASTIterationDeclaredStatement
  - 进入：("iterationBegin", , , scopeName)
  - 退出：("iterationEnd", , , scopeName)
  - 访问顺序：init -> (CheckCondLabel) -> cond -> (jmpCondFalse) -> state -> step -> (jmpCheckCond) -> (CondFalseLabel)
- ASTSelectionStatement
  - ("ifBegin", , ,), ..., ("jf", cond, , falseLabel), ..., ("jmp", , , endLabel), ("label", , , falseLabel), ..., ("label", , , endLabel)
  - 访问顺序：cond -> (judge) -> (jmpFalse) -> then -> (jmpEnd) -> (falseLabel) -> otherwise -> (endLabel)
- ASTPostfixExpression
  - ("=", var, , temp)、(op, temp, , var)
- ASTUnaryExpression
  - 可直接赋值：++a, --a: (op, a, , a)
  - 不可直接赋值：&a, *a, +a, -a, !a, sizeof(a): (op, a, , temp)
- ASTToken、ASTIntegerConstant、ASTFloatConstant、ASTCharConstant、ASTStringConstant
  - 四元式元素，按类型打印对应英文，"int"、"float"、"char"、"string"

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：

```
// 符号表
public SymbolTable globalSymbolTable;
public SymbolTable localSymbolTable;
public SymbolTable functionTable;

Map<String, ASTNode> labelTable;

// 循环控制
public Integer iterationId;
public Integer jmpLabelId;
```

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：

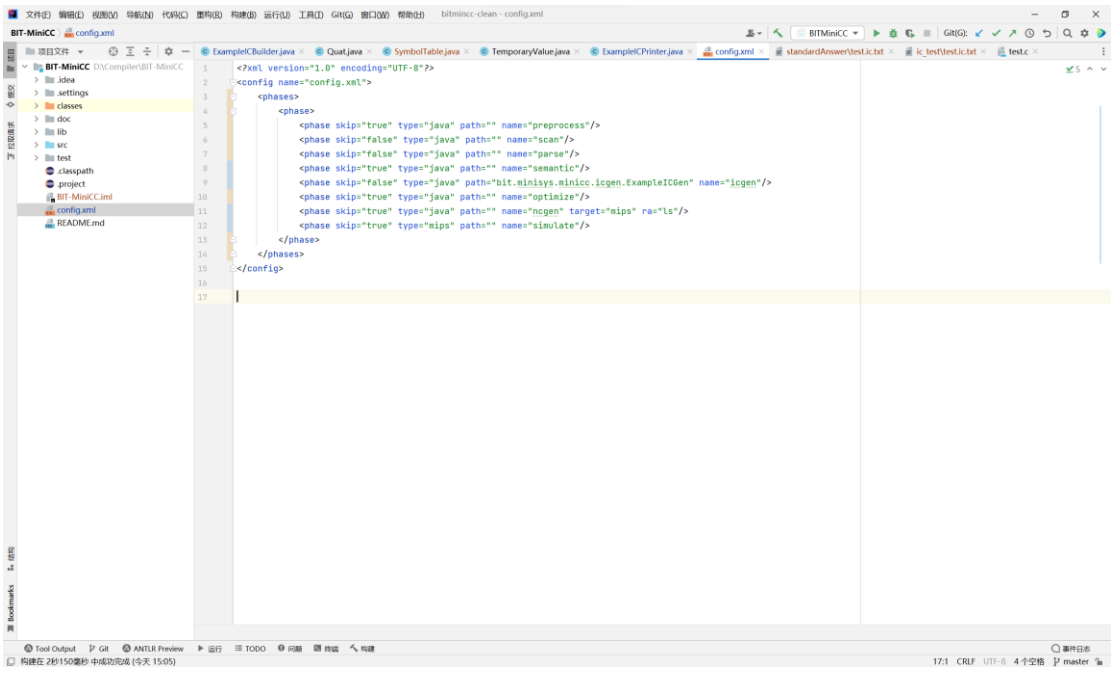
```
else if (declarator instanceof ASTArrayDeclarator) {
  ASTDeclarator arrayDeclarator = ((ASTArrayDeclarator) declarator).declarator;
  ASTExpression expr = ((ASTArrayDeclarator) declarator).expr;

  LinkedList<Integer> arrayLimit = new LinkedList<~>();

  // 取出数组各维大小，并加入变量类型中，新变量类型实例 int[10][20]
  while (true) {
      int limit = ((ASTIntegerConstant) expr).value;
      arrayLimit.addFirst(limit);

      if (arrayDeclarator instanceof ASTArrayDeclarator) {
          expr = ((ASTArrayDeclarator) arrayDeclarator).expr;
          arrayDeclarator = ((ASTArrayDeclarator) arrayDeclarator).declarator;
      } else {
          break;
      }
  }

  // 生成复合数组变量类型
  for (int limit : arrayLimit) {
      typeSpecifiers += "[" + limit + "]";
  }
  DescriptionLabel compoundTypeLabel = new DescriptionLabel(typeSpecifiers);

  if (declaration.scope == this.globalSymbolTable) {
      this.globalSymbolTable.addVariable(name, typeSpecifiers, arrayLimit);
  } else {
      this.localSymbolTable.addVariable(name, typeSpecifiers, arrayLimit);
  }

  Quat quat = new Quat( op: "arr", arrayDeclarator, compoundTypeLabel,  opnd2: null);
  quats.add(quat);
```

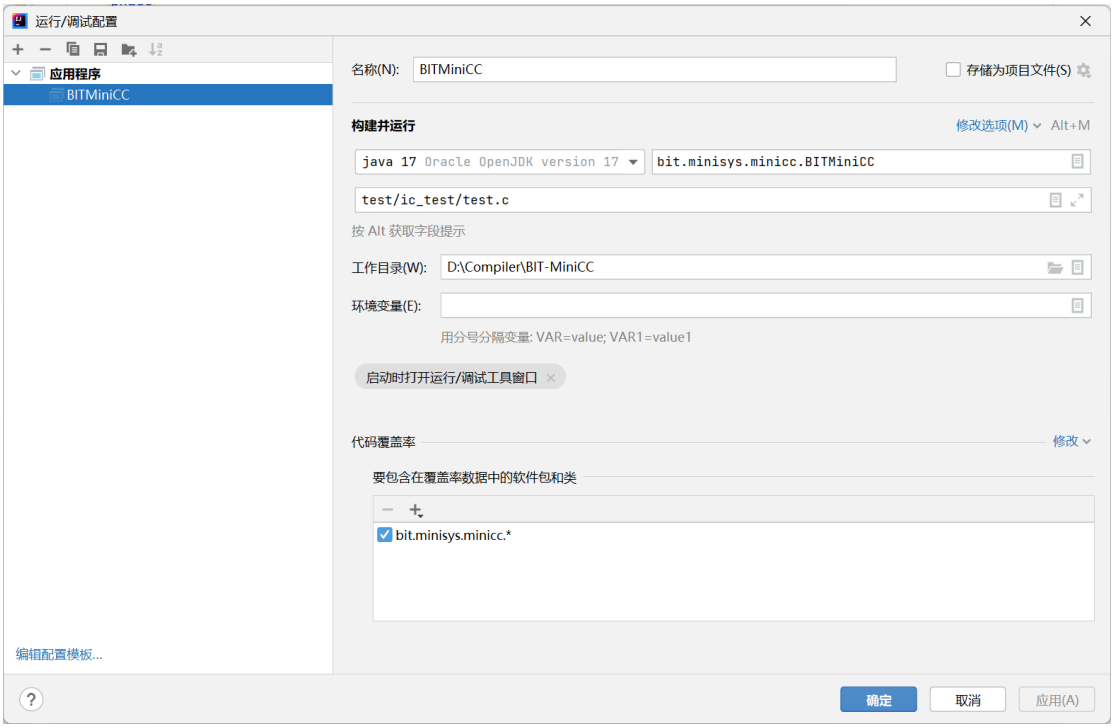
（2）打印四元式

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

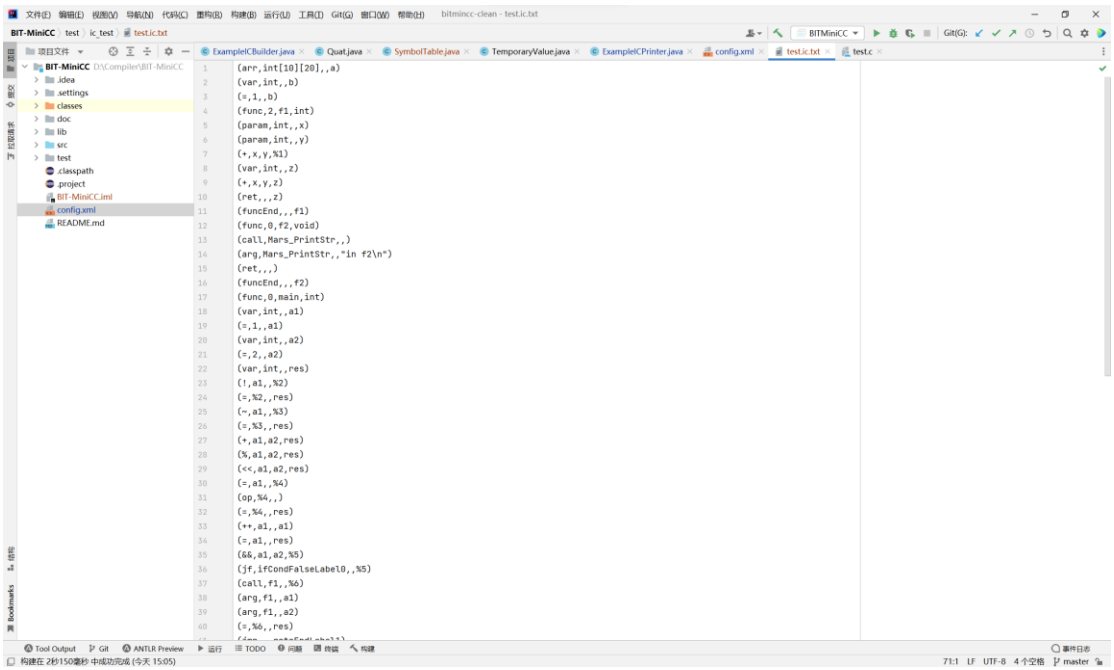
配置 BITMiniCC 配置文件，使用内置的词法分析器、语法分析器，自己编写的中间代码生成工具，并将测试用例路径写入运行配置。配置文件具体信息如下：





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。



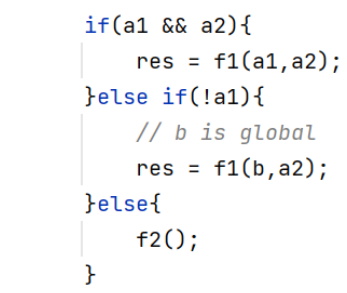
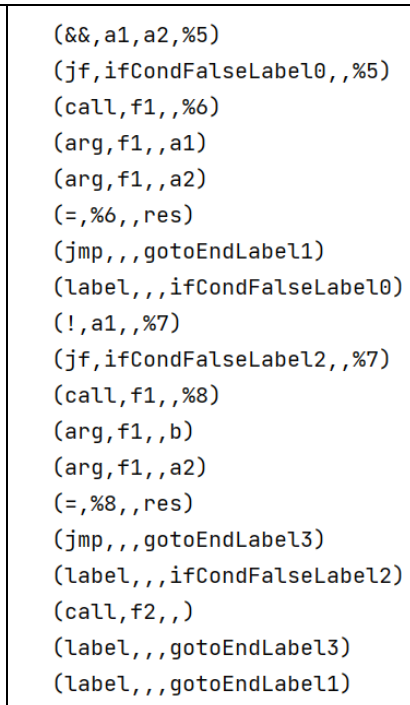
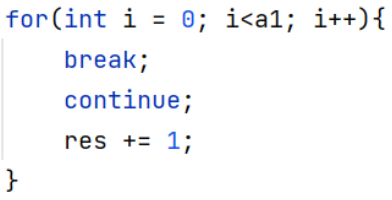
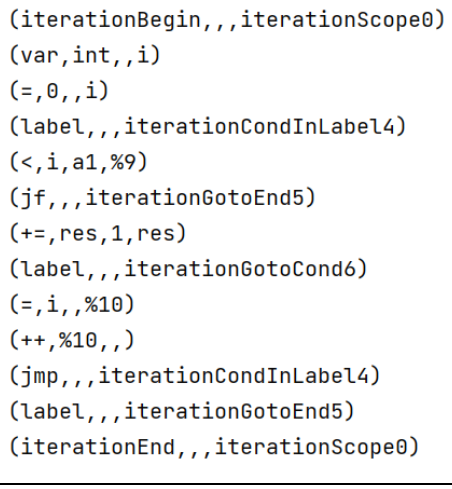
分析可知，编写的中间代码生成器可以按照设计的四元式格式，正确生成各种 AST 结点的四元式。

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	`int a[10][20];` `int b = 1;`	(arr,int[10][20],,a) (var,int,,b) (=,1,,b)

函数定义	`void f2(){`
	`  Mars_PrintStr("in f2\n");`
	`  return;`
	`}`
		(func,0,f2,void)
		(call,Mars_PrintStr,,)
		(arg,Mars_PrintStr,,"in f2\n")
		(ret,,,)

赋值语句	`res = !a1;`	(!,a1,,%2) (=,%2,,res)

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。