**西北农林科技大学信息工程学院**

**《编译原理》综合训练实习报告**

|  |  |
| --- | --- |
| **姓名** | 刘睿 |
| **学号** | 2017013115 |
| **专业班级** | 软件工程1701班 |
| **指导教师** | 赵建邦 |
| **实践日期** | 2019年12月16日-2019年12月20日 |

目 录

[一、综合训练目的与要求 1](#_Toc27750803)

[二、 综合训练任务描述 1](#_Toc27750804)

[1.基本功能 1](#_Toc27750805)

[2.扩展功能 1](#_Toc27750806)

[三、 编译系统设计 2](#_Toc27750807)

[1. 总体设计 2](#_Toc27750808)

[2.详细设计 3](#_Toc27750809)

[（1）词法分析 3](#_Toc27750810)

[（2）文法分析和语义分析 3](#_Toc27750811)

[（3）错误检查 8](#_Toc27750812)

[四、调试与测试 9](#_Toc27750813)

[(1) 相关文法的测试 9](#_Toc27750814)

[(2) 错误检查 17](#_Toc27750815)

[五、实习日志 18](#_Toc27750816)

[六、实习总结 18](#_Toc27750817)

# 一、综合训练目的与要求

本综合训练是软件工程专业重要的实践性环节之一，是在学生学习完《编译原理》课程后进行的综合练习。

**本课综合训练的目的：**

1. 巩固和加深学生对编译原理课程基本知识的理解和掌握；

2. 提高抽象思维能力以及加深理解编译系统的原理；

3. 掌握利用JavaCC开发编译系统的方法；

4. 掌握书写设计与实现编译系统说明文档的能力；

5. 提高综合运用算法、程序设计语言、数据结构知识的能力。

**本课程综合训练的要求：**

1. 使用文件读入的方式测试编译程序；
2. 将词法分析、语法分析和语义分析结果分别以文件方式输出；
3. 尽可能实现要求的功能，可以自己决定实现哪些扩展功能；
4. 不要求实现界面，重点实现编译功能。（开发时间有剩余的同学可以实现界面。）

**编译功能要求：**

1. 能够根据单词的构词规则，完成MiniC语言中的单词的解析(词法分析)，如果不符合单词的构词规则，请给出错误信息。如果源语言符合单词的词法规则，输出<单词种别，单词自身值>二元式；
2. 在词法分析的基础上，构造MiniC的递归下降分析文法，利用JavaCC实现递归下降文法。判断源语言是否符合MiniC的语法，如果符合，输出语法树；否则，请给出语法错误信息；
3. 在语法分析的基础上，根据属性文法制导翻译，进行语义分析，输出四元式。如果源语言不符合MiniC的语义，请指出错误信息；
4. 在实习课的基础上，整个编译系统要能够翻译声明语句、赋值语句、布尔表达式，if-else, for, while，do-while，switch-case等语句嵌套的分析与翻译；符号表、语义错误检查：变量重复声明、变量使用时未定义。
5. 扩展内容：可以考虑自增、自减、main()方法调用其它子程序的翻译、数组(二维，多维)翻译、数组下标越界等错误、变量赋值精度错误、函数返回值类型不匹配等；
6. 扩展内容：可以添加功能，如break、continue、return语句、一维指针等。

# 综合训练任务描述

## 1.基本功能

1. 用流的形式读入要分析的MiniC源程序；
2. 能够根据单词的构词规则，完成MiniC语言中的单词的解析(词法分析)，如果不符合单词的构词规则，请给出错误信息。如果源语言符合单词的词法规则，请输出<单词种别，单词自身值>二元式；
3. 在词法分析的基础上，构造MiniC的递归下降分析文法，利用JavaCC实现递归下降分析文法。判断源语言是否符合MiniC的语法，如果符合，输出语法树；否则，请给出语法错误信息；
4. 在语法分析的基础上，根据属性文法制导翻译，进行语义分析，输出四元式。如果源语言不符合MiniC的语义，请指出错误信息；
5. 在平时实习课的基础上，整个编译系统要能够翻译声明语句、算数达式、关系表达式、布尔表达式，if-else, for, while，do-while，赋值等的分析与翻译；
6. 实现符号表，并能够检查变量重复声明、使用时未定义错误。

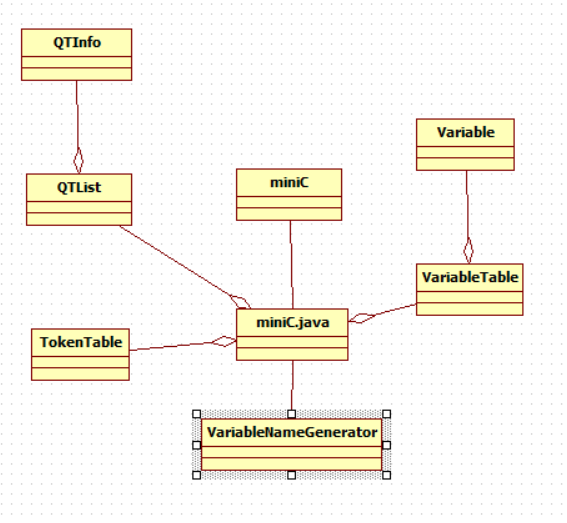
## 2.扩展功能

1. 为编译器添加界面。
2. 函数定义与调用。

# 编译系统设计

## 总体设计

1. 总体设计类图



1. 类图说明

MiniC：显示界面，用户选择要分析的代码文件后，自动调用miniC中的函数进行编译分析，输出词法分析结果、语法分析结果和语义分析结果文件，并显示在界面上。

miniC.java：用于编译miniC代码，定义了词法规则、语法规则，包含QTList、TokenTable、VariableNameGenerator、VariableTable、ConditionValue等辅助类进行语义分析和保存，因此与其关系是组合关系。提供接口函数进行代码分析，因此与MiniC是调用关系。

TokenTable：内含一个ArrayList<Token>类型的变量LEXlist，在miniC进行词法分析时存储、添加、删除或显示Token元素。在显示列表中所有Token元素时，调用Token中自定义的显示方法按实验要求的<单词种别，单词自身值>二元式格式将其存储到qtTable.txt文件中。

QTInfo：保存、修改、显示四元式，用于语义分析。其中元素包括四元式的ID，操作符，操作数，结果。

QTList：内含一个ArrayList<QTInfo>类型的变量QTlist，在miniC进行语义分析时存储、添加、删除或显示四元式。在显示列表中所有的四元式时，将其存储到qtTable.txt文件中。

VariableNameGenerator：用于产生临时变量的类，格式为前缀T+数字序号。

ArrayInfo：内情变量表类。

Variable：变量类，用于保存变量的相关信息，其中包括变量类型、变量名称等相关属性，其中的一个属性是ArrayInfo类型，如果有数组，用于存储数组的信息，如果没有的话其值为null。

VariableTable：变量表，内含一个ArrayList<Variable >类型的变量varTable，即其和Variable的关系是组合关系，可插入、显示Variable，存储程序中出现的变量，还能用来检查变量未定义、变量重复定义错误。

ConditionValue：在语义分析过程中用于存储、回填真链假链的类。

## 2.详细设计

## （1）词法分析

1）定义tokenTable用于存储、添加、删除或显示Token元素。

public static TokenTable tokenTable = new TokenTable();

2）在jjt文件的词法声明部分加入miniC的关键词、逻辑符号、运算符号、关系符号、分隔符等TOKEN定义。

3）在进行语义和语法分析时，用TokenTable的addToken方法保存程序中的Token。

4）在miniC界面上显示列表中所有Token元素时，调用miniC中定义的方法，调用TokenTable中的printTokenTable方法，将程序运行分析后得到的tokenTable按<单词种别，单词自身值>二元式格式存储到qtTable.txt文件中。每个 Token 中int kind 表示单词的 种别， String image 存储了 token 所代表的内容。

## （2）文法分析和语义分析

在jjt文件中进行语法规则的定义和实现。先根据C语言的语法提取MiniC的文法，文法中的每个非终结符对应一个函数，函数调用表示非终结符之间的组成关系。

根据 MiniC 的上下文无关文法，添加语义子程序，实现声明语句、数学表达式、关系表达式、布尔表达式，if-else, for, while，do-while，switch-case，赋值等语句嵌套的分析与翻译；保存并输出分析过程中的四元式，并用符号表保存变量，以检查变量未定义、变量重复定义错误。

以下为文法、语义子程序及其对应的函数实现。

1. 在Start()中调用程序函数Program
2. Program()

<程序> → (void|int) main () { <语句块>\* }

负责匹配程序的主体部分，调用StartKey()分析(void|int) main ()部分，StatementBlocks()分析语句块。

1. StartKey()

<首部> → (void|int) main ()

负责匹配程序开头部分的关键字。

1. StatementBlocks()

〈语句〉 →〈顺序语句〉|〈条件语句〉|〈循环语句〉

负责匹配语句块中的以上三种类型的语句。

1. SeqStatement ()

〈顺序语句〉→ (〈声明语句〉| 〈赋值语句〉) ”;” | 〈switch-case语句〉

负责匹配顺序语句，即声明变量的语句或给变量赋值的语句或switch-case语句。

1. DeclareStatement ()

〈声明语句〉→ <变量类型> ID(,ID)\*

负责匹配声明变量的语句。变量类型是指int|float|char|double|bool中的一种，用DeclarKey()来匹配。ID则为标识符变量< IDENTIFIER >；

为了能够检查变量重复声明类型的错误，将ID(,ID)\*部分封装成一个函数DefineId，返回值是boolean类型，功能是将新定义的变量插入到变量表中去，且插入前会检测这个Token的image是否已经存在与变量表中。如果已存在，返回false并输出报错信息；如果不存在，返回true。

1. AssignStatement ()

〈赋值语句〉→ ID =〈表达式〉

负责为变量赋值的语句，表达式用Expression()匹配。

为了能够检查变量不存在类型的错误，将ID部分封装成一个函数Identifier，返回值是Token，功能是匹配标识符< IDENTIFIER >，并遍历、更新变量表，检测这个Token的image是否存在于变量表中。如果已存在，返回匹配的标识符Token；如果不存在，输出报错信息。

添加语义子程序时，先在第一个{}中定义变量：String first;String s;Token op;first是被赋值的变量名，s是赋予的值，op是操作符‘=’；

然后在()中为变量赋值，first = Identifier() op = "=" s = Expression()；

第二个{}中生成一条QTInfo类型的四元式信息，并将其加入到QtTable中。QTInfo qt = new QTInfo(op.image, s, "\_", first); qtList.addQTInfo(qt);

1. SwitchCase ()

〈switch-case语句〉→ switch (ID)<case语句>

〈case语句〉→(case < INTEGER\_LITERAL >:〈语句块〉break;)\*default:〈语句块〉break;

负责匹配switch-case语句。

1. MultiplicativeExpression()与AdditiveExpression()

负责数学运算，分别是乘除运算语句和加减运算语句。

添加语义子程序时，乘除运算语句先在第一个{}中定义变量：String first;String middle;Token op，String newTemp;;first是第一个要运算的变量名，middle是第二个要运算的变量名，op是操作符‘\*’‘/’‘%’，newTemp是计算结果和返回值；

为first和middle调用UnaryExpression()赋值后，在第二个{}中先为结果变量newTemp赋值为variableNameGenerator.genVariableName()，即新产生一个变量名，再将其加入QTTable，first也赋值为newTemp，因为该部分可能循环被调用。

加减运算语句与乘除运算语句不同之处在于first、middle的赋值来源于MultiplicativeExpression()的值，因为乘除等运算比加减优先级高。

1. 布尔表达式或与非：Logic()、LogicAnd()、LogicNot()、Condition

<逻辑> →<与逻辑>（<||> <与逻辑>）\*

<与逻辑> →<非逻辑>（<&&> <非逻辑>）\*

<非逻辑> →<条件>（<!> <条件>）\*

<条件> →<表达式>（<关系符> <表达式>）\*

由于产生式的优先级是优先级越高，其位置越往下，在布尔表达式中优先级从高到低依次是关系符、非、与、或，因此产生了如上的文法。各个相应函数中语义子程序的添加过程如下：

Logic():

逻辑或。调用LogicAnd()得到chain，如果接下来匹配了<OR>符号，则说明chain中的假链出口已找到，将四元式序号加一并将其回填给chain的假链；

调用LogicAnd()得到下一个条件的链chain2，然后声明一个空链reschain,先将chain2的假链复制到reschain，然后将chain的真链复制到chain2后面，然后将chain2的真链复制给reschain。也就是把假链替换成当前条件返回的假链，真链后添加上当前条件的真链。

LogicAnd():

逻辑与。调用LogicNot()得到chain，如果接下来匹配了<AND>符号，则说明chain中的真链出口已找到，将四元式序号加一并将其回填给chain的真链；

调用LogicNot()得到下一个条件的链chain2，然后声明一个空链reschain,先将chain2的真链复制到reschain，然后将chain的假链复制到chain2后面，然后将chain2的假链复制给reschain。也就是把真链替换成当前条件返回的真链，假链后添加上当前条件的假链。

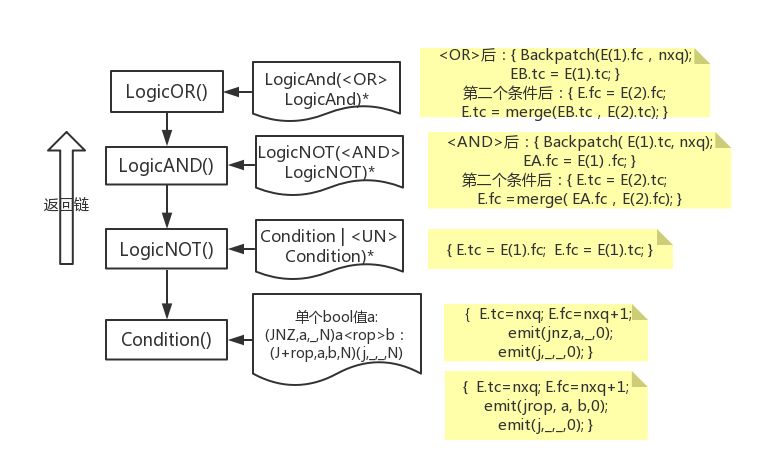
LogicNot():

逻辑非。调用condition()得到根据当前条件被填入四元式后的chain，如果有符号‘！’，则要将从condition得到的值的真假链置换一下，即在chain1 = Condition()之后，chain.setTrueChain(chain1.getFalseChain());chain.setFalseChain(chain1.getTrueChain())。

其中，condition()函数用来给某个条件的链填入真链和假链的四元式。先在第一个{}中定义变量：String first;String middle;String rop;ConditionValue chain = new ConditionValue(); rop是比较符，first = Expression()获取第一个变量，rop = RelationChar()获取符号值，middle = Expression()获取第二个变量；

然后添加四元式：QTInfo qt1 = new QTInfo("J" + rop, first, middle, "N");QTInfo qt2 = new QTInfo("J", "\_", "\_", "N");如果未获取关系符，则QTInfo qt1为"JNZ", first, "\_", "N"。

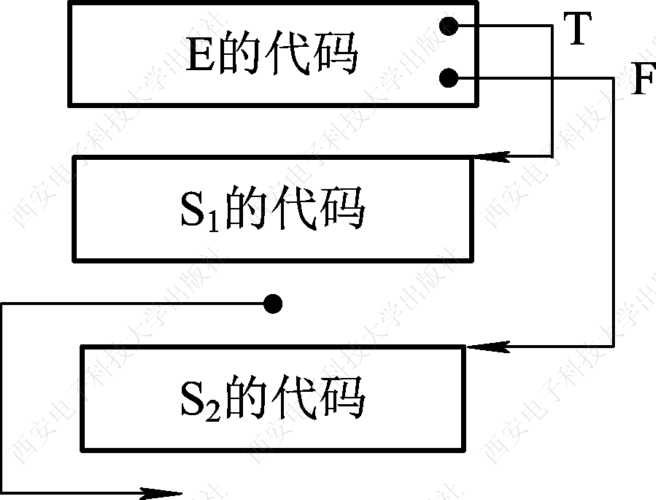
布尔表达式的调用关系和逻辑结构如下：



1. If-else语句CondStatement()

〈条件语句〉→ if〈条件〉〈语句块〉（else 〈语句块〉）？

负责if-else语句的翻译。下图为if-else语句真假出口的示意图，按照该图对由逻辑产生的真假链进行回填，引用自编译原理4.5节ppt课件：



定义语义子程序时，先在第一个{}中定义变量：用于回填的链ConditionValue chain、temp和qt语句序号int i；

在第二个{}中调用Logic函数为chain赋值，得到if括号内条件的真链和假链，并已在函数内将其回填了一部分；

在if后面的语句由于找到真链出口，故在其后{}中i = QTInfo.innerIdSeqen + 1，调用chain.backpatchTrueChain(i)回填真链；

在if语句块结束后，由于需要跳过整个else语句块，所以加入跳转语句后，为了回填，再将其加入一个临时链，待else语句块结束之后进行回填；

在if结束后的语句和else开始前是假链出口，的{}中chain.backpatchFalseChain(i);回填假链。

1. 循环语句LoopStatement()、While()、Do\_While()、 For\_Loop()

<循环语句> →<while语句>|<for语句>|<do-while语句>

<while语句> →<while><逻辑> <语句块>

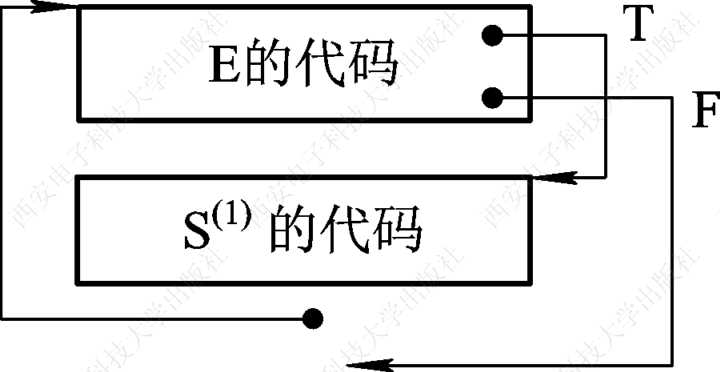
<for语句> → <for>(<赋值语句>;<逻辑>;<赋值语句>)<语句块>

<do-while语句> →<do><语句块><while><逻辑>

循环语句中有三种循环，for循环、do-while循环、while循环，三种循环产生四元式的时候因为其文法结构不同，所以四元式的回填位置也不同。语义子程序设计如下：

While():

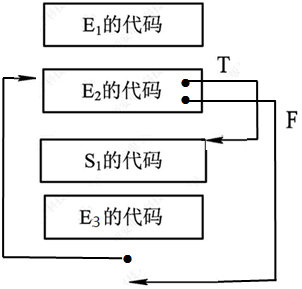
下图为while语句真假出口的示意图，按照该图对由逻辑产生的真假链进行回填，引用自编译原理4.5节ppt课件：



先记录当前的QTTable大小，即四元式语句编号tag，以便每轮循环结束后跳转；在左右括号中的条件部分调用chain = Logic()，在开始循环之前的{}进行chain中真链的回填，执行完while花括号内的StateBlocks后，添加一个跳回while开头的J,\_,\_,tag语句，然后到了chain的假链出口，回填chain的假链。

For\_Loop():

下图为for语句真假出口的示意图，按照该图对由逻辑产生的真假链进行回填，引用自编译原理4.5节ppt课件：



在声明语句、赋值语句之后，回填真链，并记录当前的四元式编号tag以便每轮循环结束后跳转；执行完花括号内的StateBlocks后，添加一个跳回花括号内语句开头的J,\_,\_,tag语句，然后到了chain的假链出口，回填chain的假链。

Do\_While() :

do标识符后面为真出口，进行真链回填；记录当前的四元式编号tag以便每轮循环结束后跳转；在{StateBlocks()}(后，调用Logic()获取经过逻辑判断后的链，回填其假链。

1. 函数定义和调用语句function()、callfunc ()

<函数> →<变量类型> ID(<参数><,参数>\*) {〈语句块〉(return ID;)?}

<调用语句> →ID(ID(,ID)\*);

实际实现的时候，调用语句被合并到赋值语句中了，因为赋值语句和调用语句开头都是要匹配变量标识符，会产生冲突警告。所以实际上的结构为<赋值或调用语句> →ID((ID(,ID)\*);)?( =〈表达式〉)?。

## （3）错误检查

利用符号表VariableTable存储程序中声明的变量，该类中有canInsert(String name)函数，如果该名字已存在该符号表中，就返回false，代表不能声明该变量；否则返回true，代表能声明该变量，或不能进行调用。

1. 检查重复定义变量错误DefineId()

该函数用于匹配接收变量标识符，检查变量重复定义错误，并将通过检查的变量加入到符号表中。该函数在有变量定义的时候使用。伪代码如下：

boolean DefineId() :

{}

{

id = < IDENTIFIER >

{

if (变量表中不存在该变量)

{

在变量表中加入该变量；

返回true；

}

将错误的行、列、变量名存入文件error.txt；

返回false；

}

}

1. 检查变量未定义错误Identifier()

该函数用于匹配接收变量标识符，检查变量未定义错误。该函数在有变量被调用的时候使用。伪代码如下：

Token Identifier () :

{}

{

id = < IDENTIFIER >

{

if (变量表中存在该变量)

{

返回该变量；

}

将错误的行、列、变量名存入文件error.txt；

返回该变量；

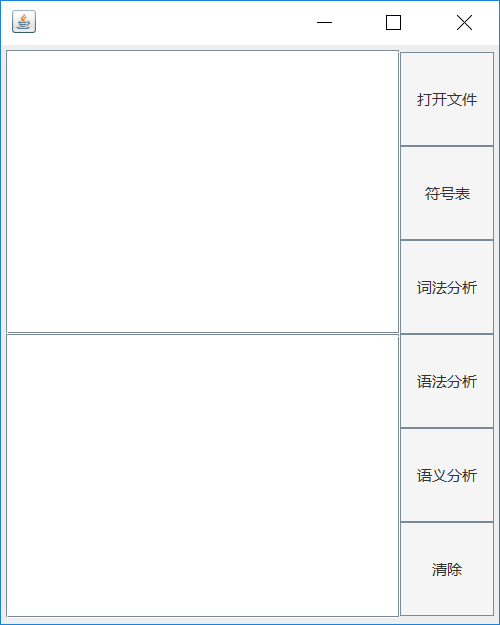
}

}

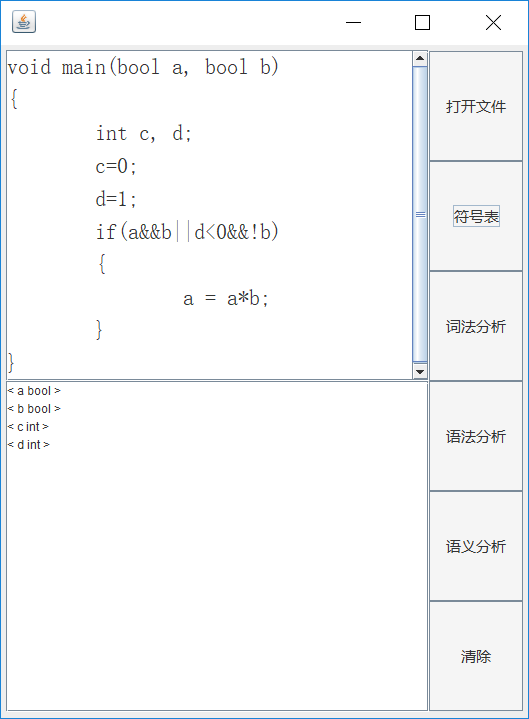
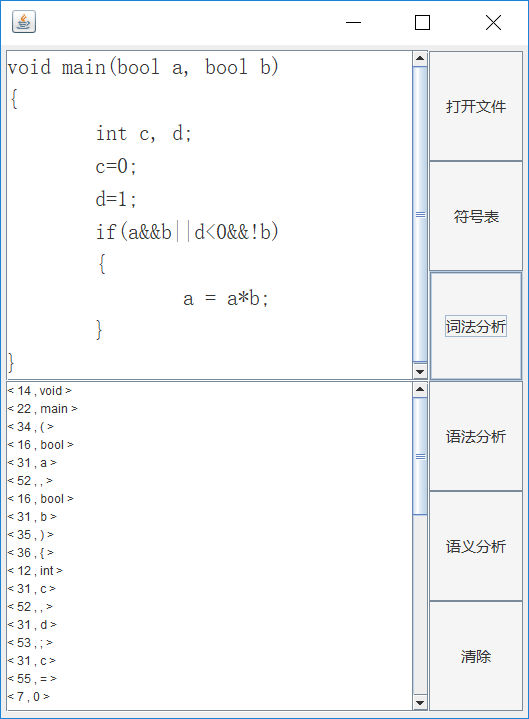
# 四、调试与测试

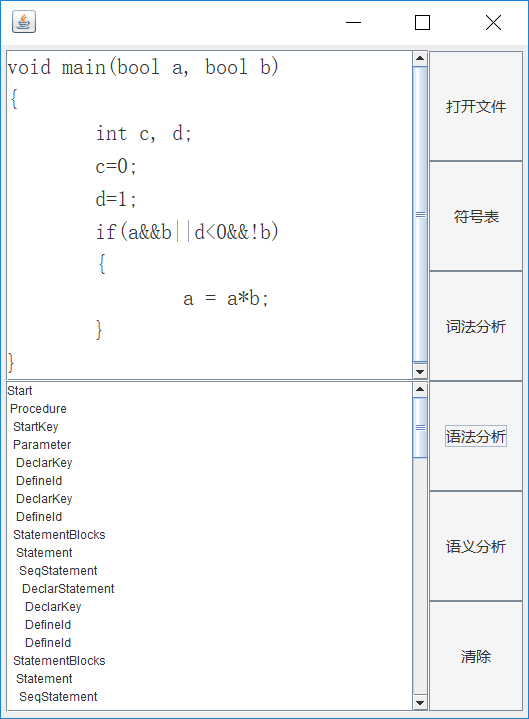
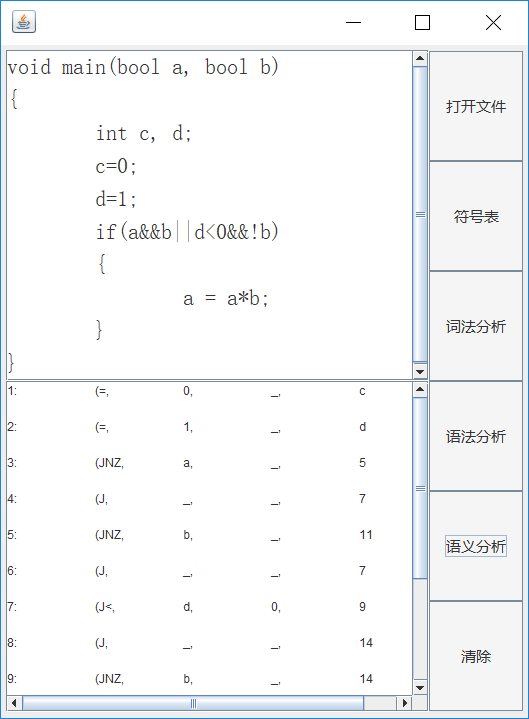
## 相关文法的测试

编译器界面如图2所示。左上方框中可以输入测试代码，或通过打开文件按钮显示代码；左下方框中通过点选不同按钮显示不同的分析结果，展示词法、语法、语义分析的结果。每完成一次测试用例，需要点选清除按钮，然后进行下一次测试

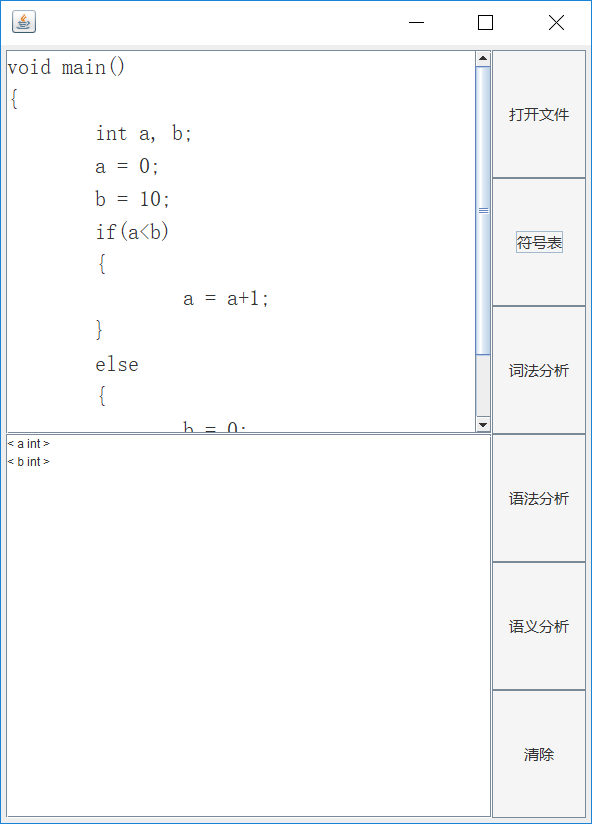
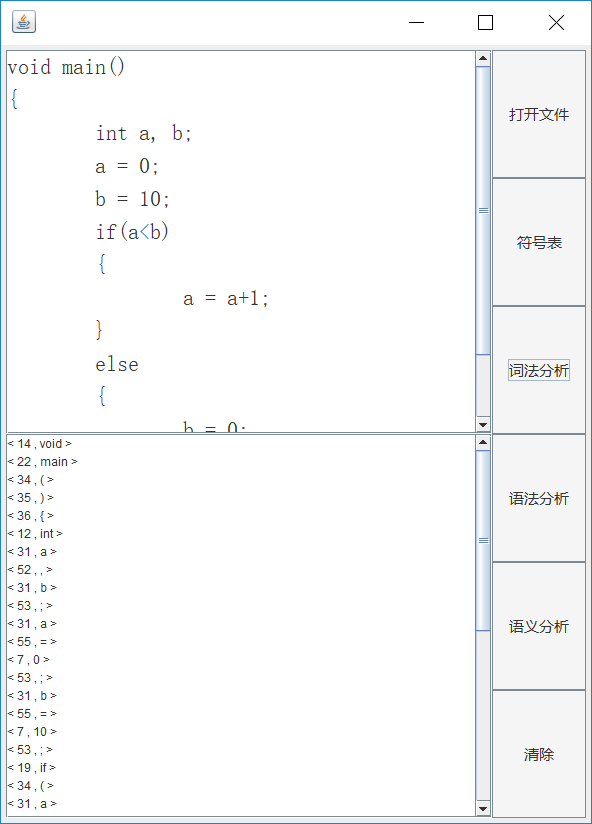


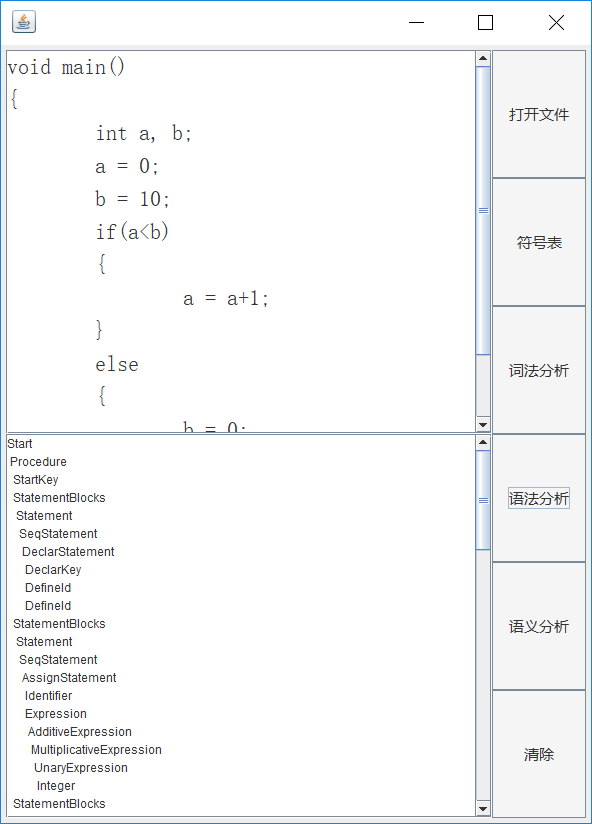
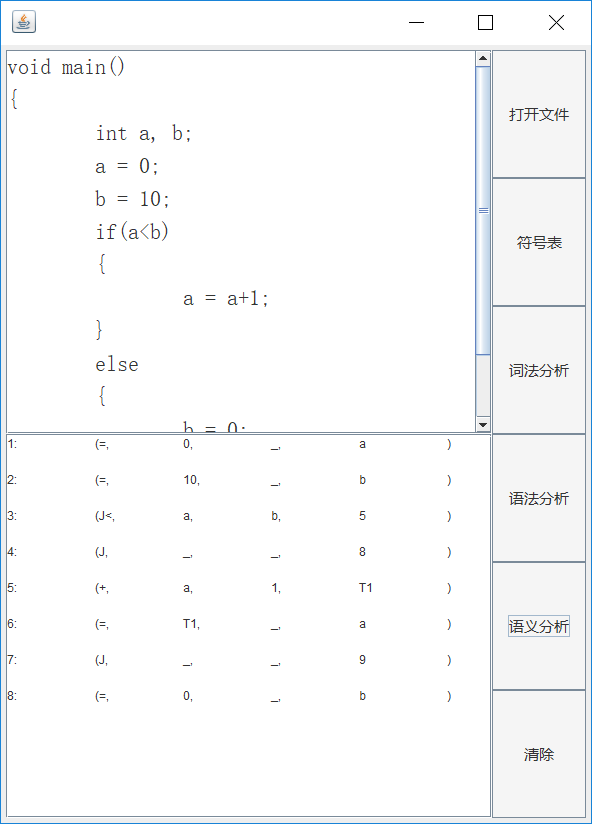
1.布尔表达式用例测试

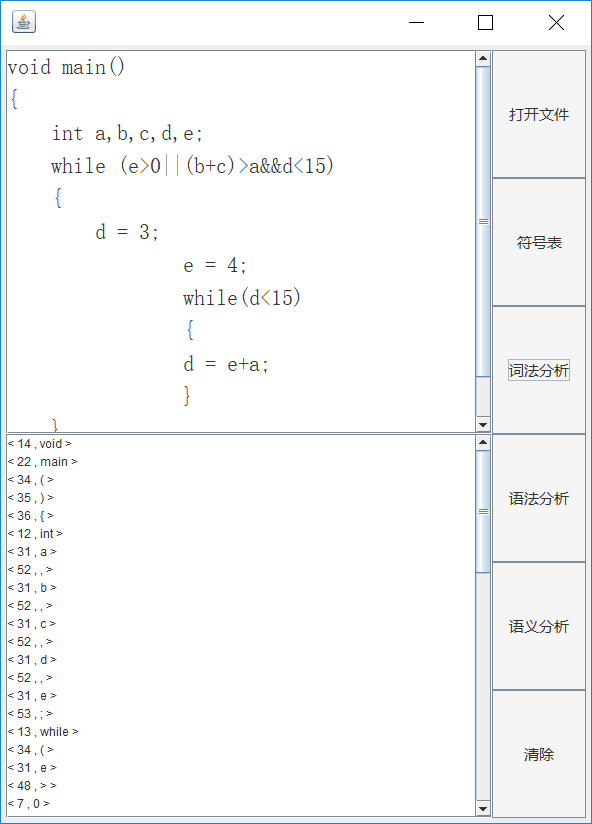
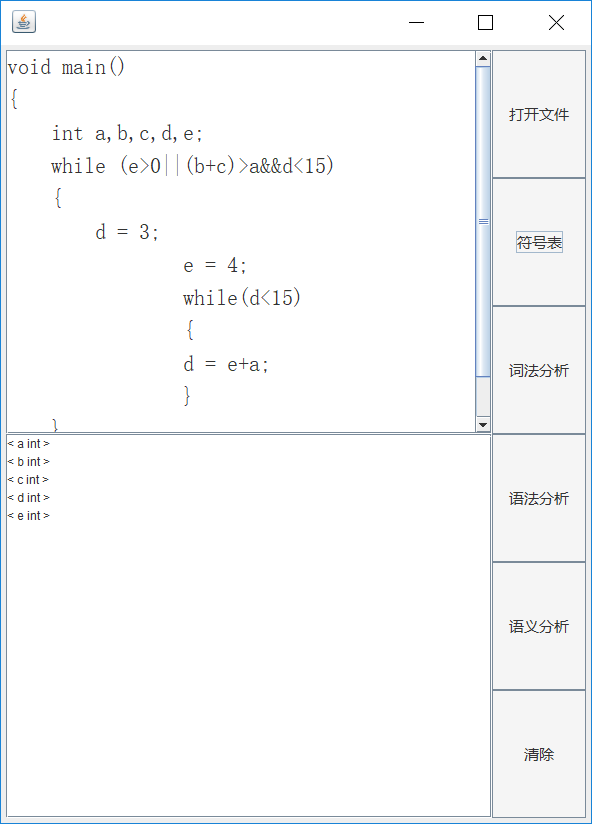
 

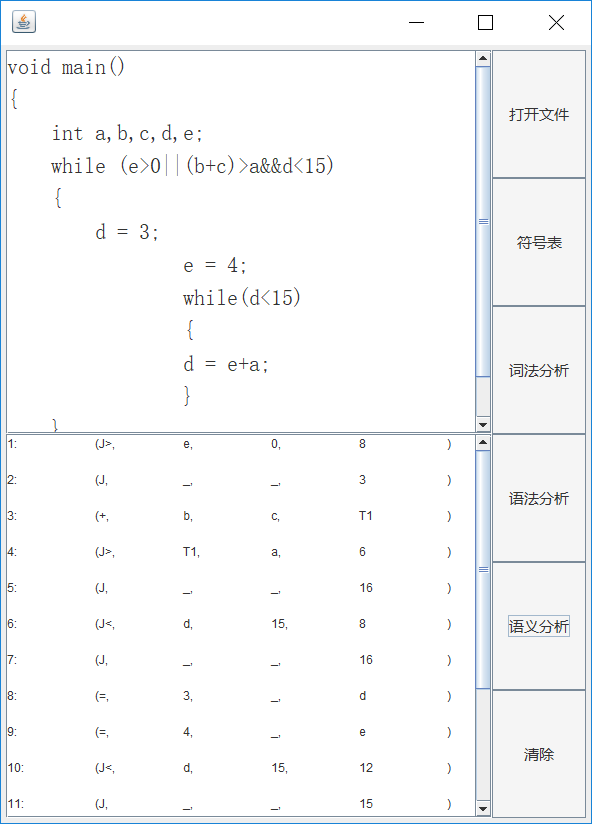
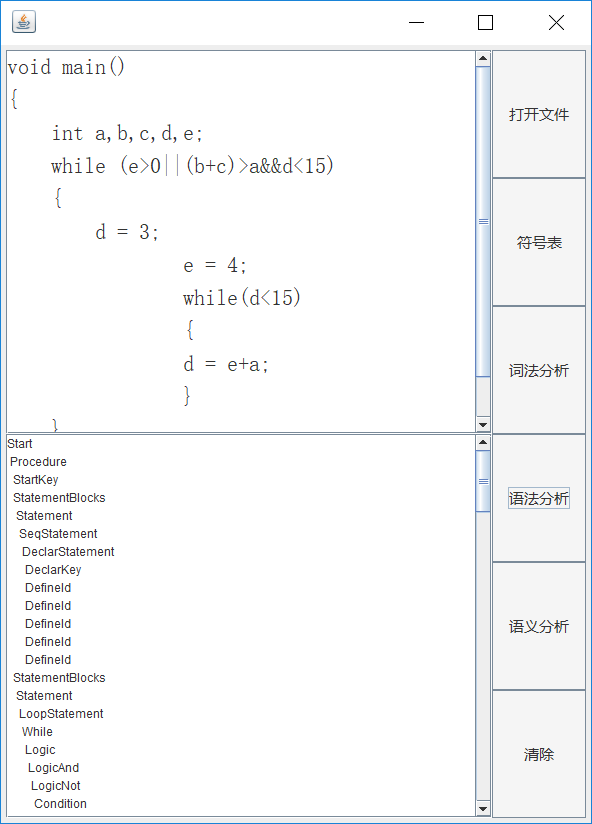
2.If-else语句测试

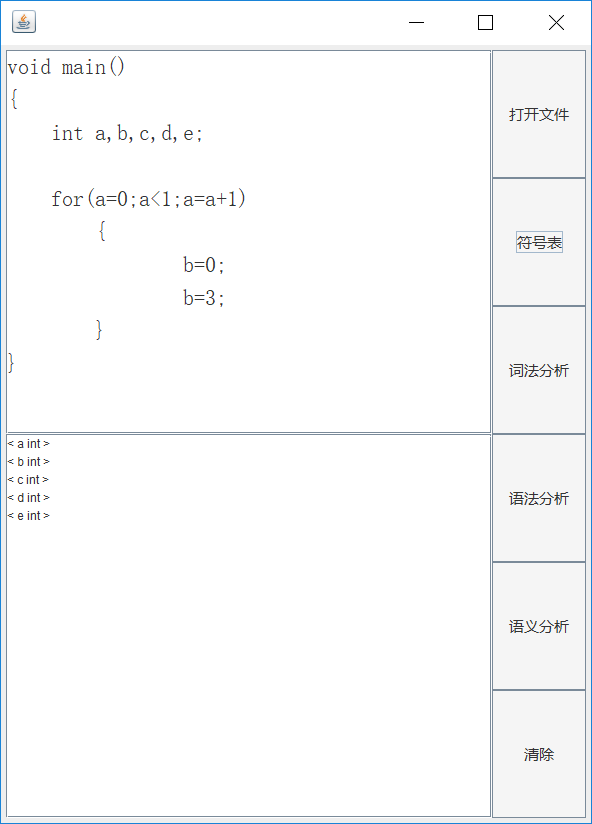
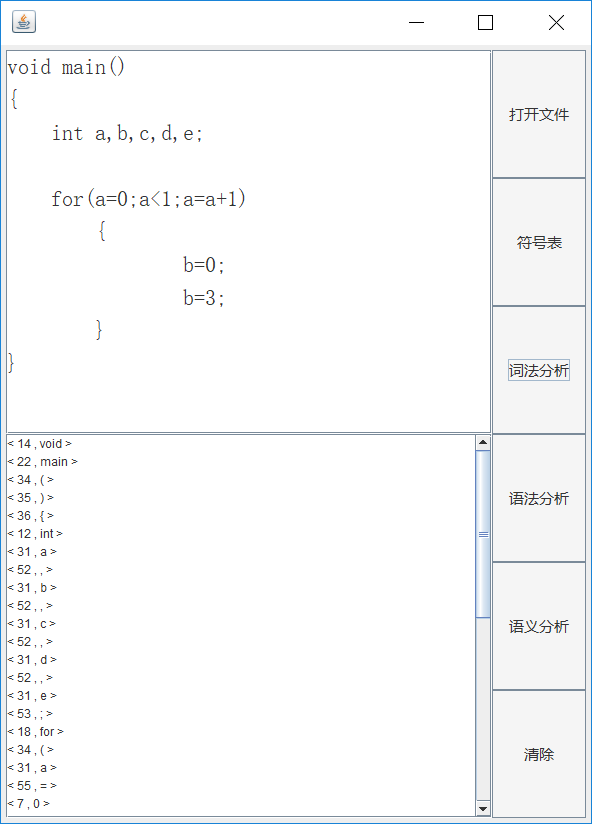
 

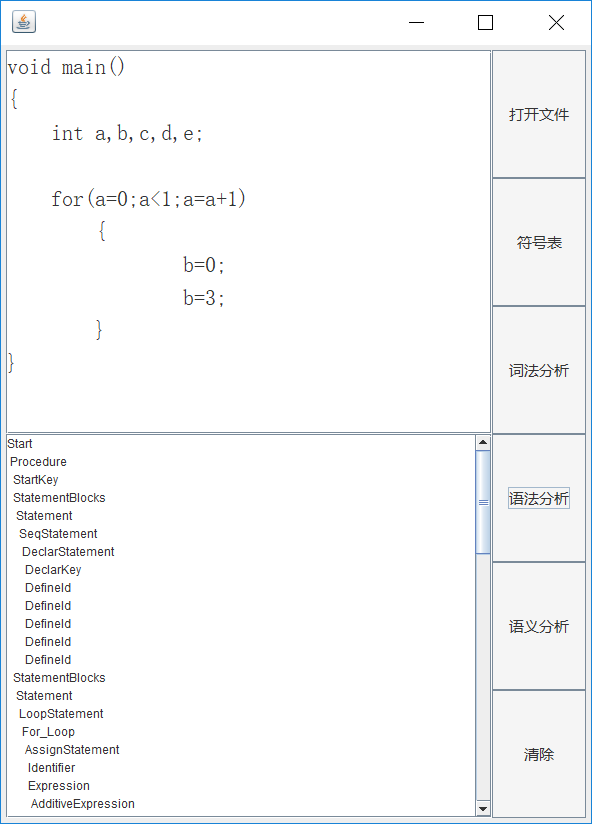
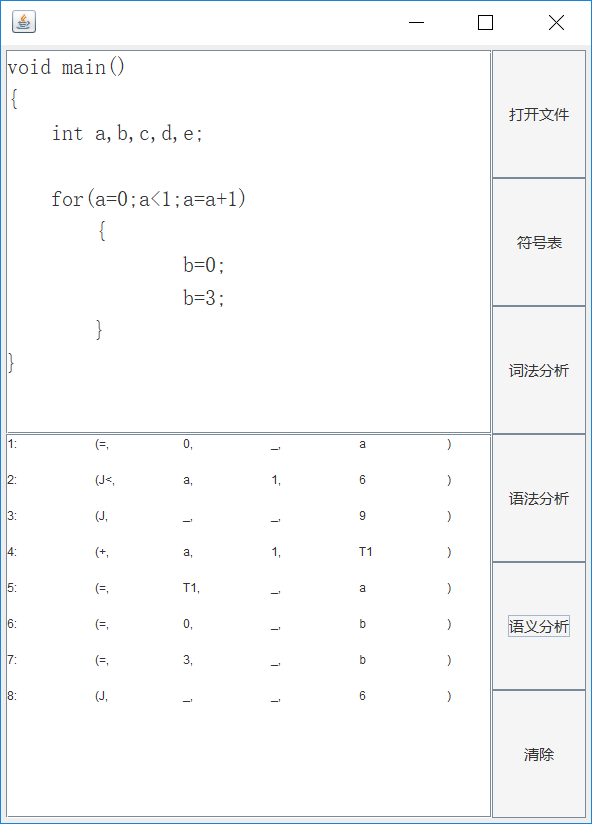
3.While语句测试



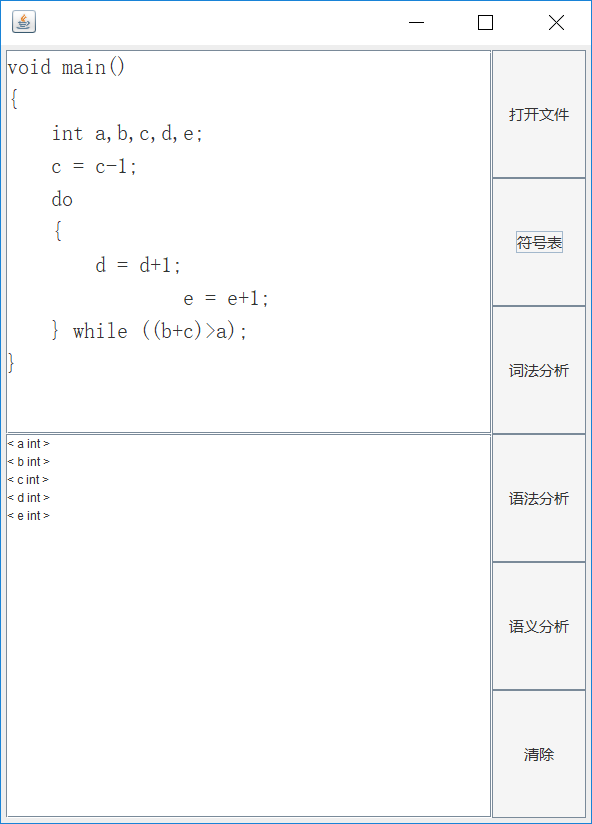
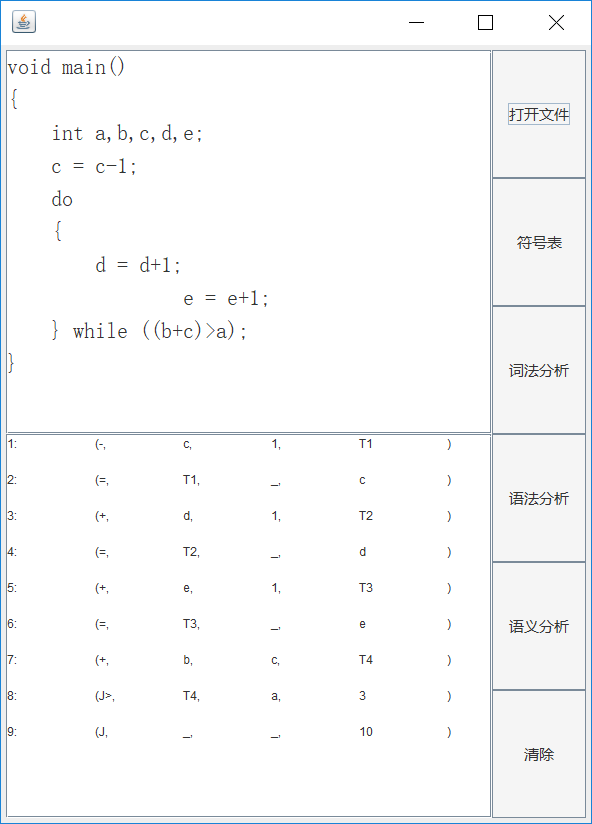


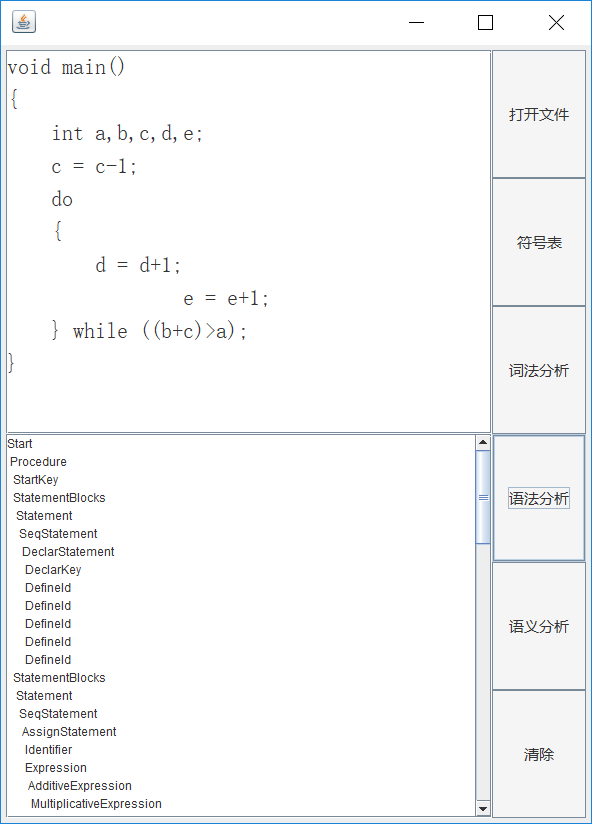
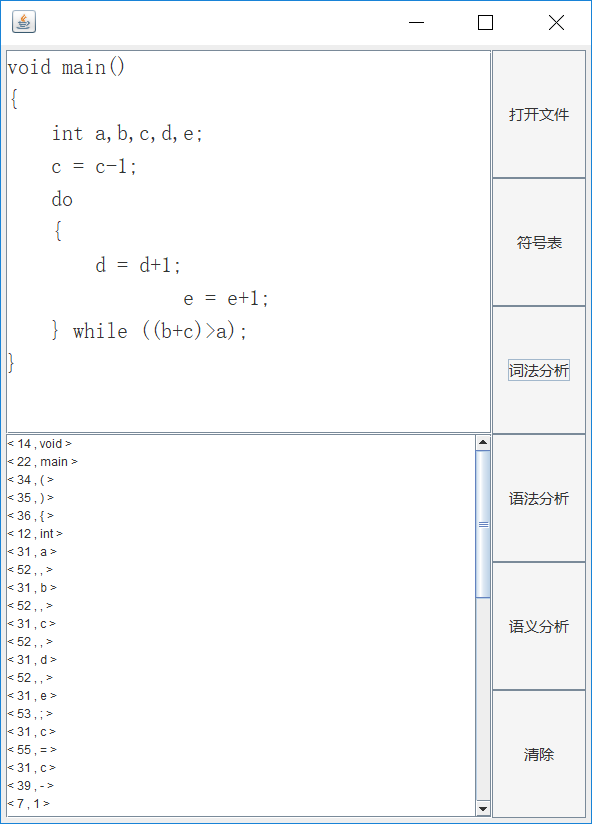
4.For语句测试

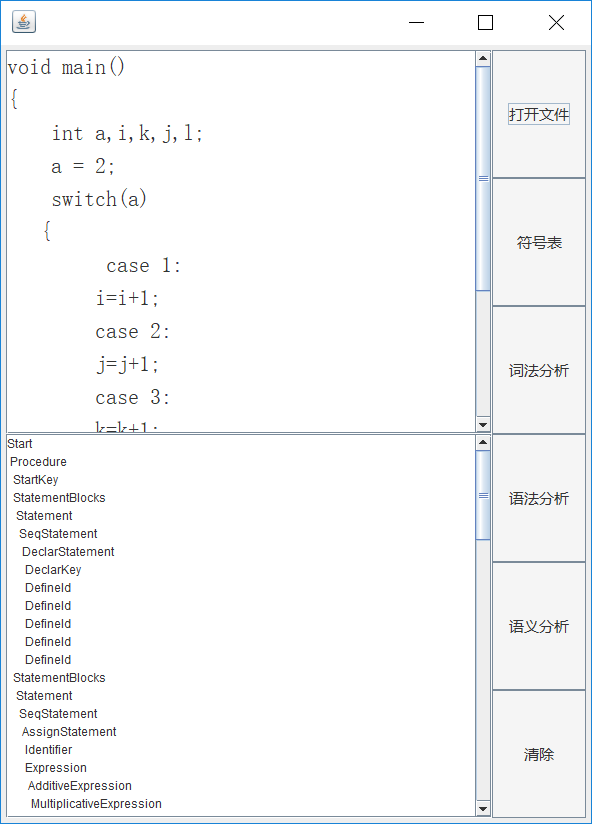
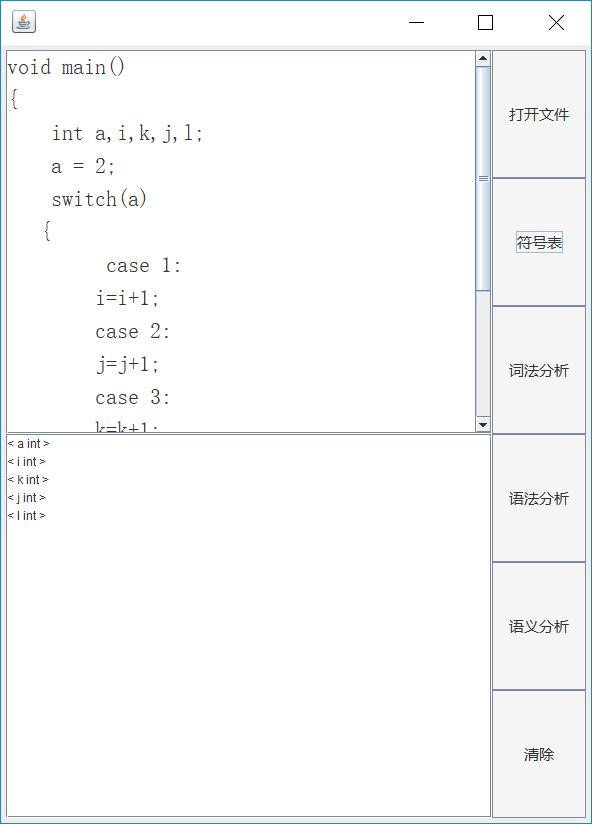
 

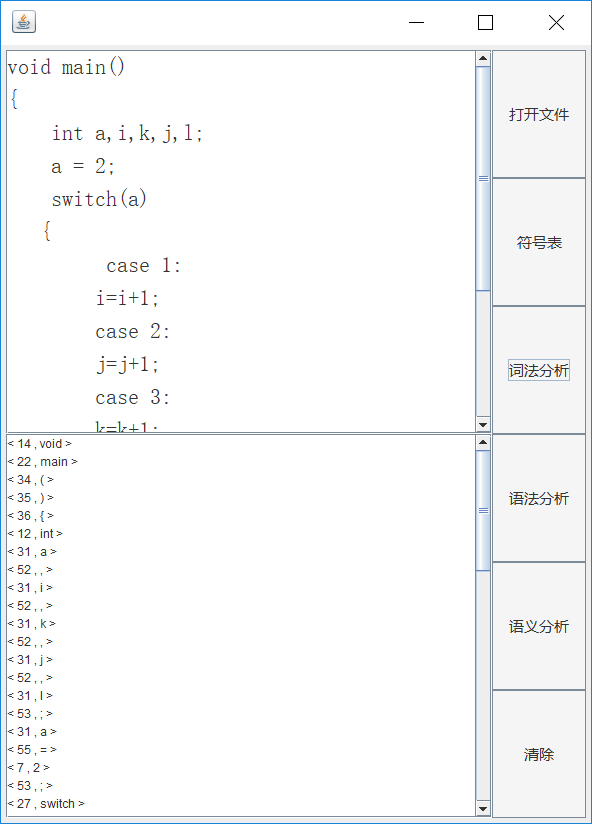
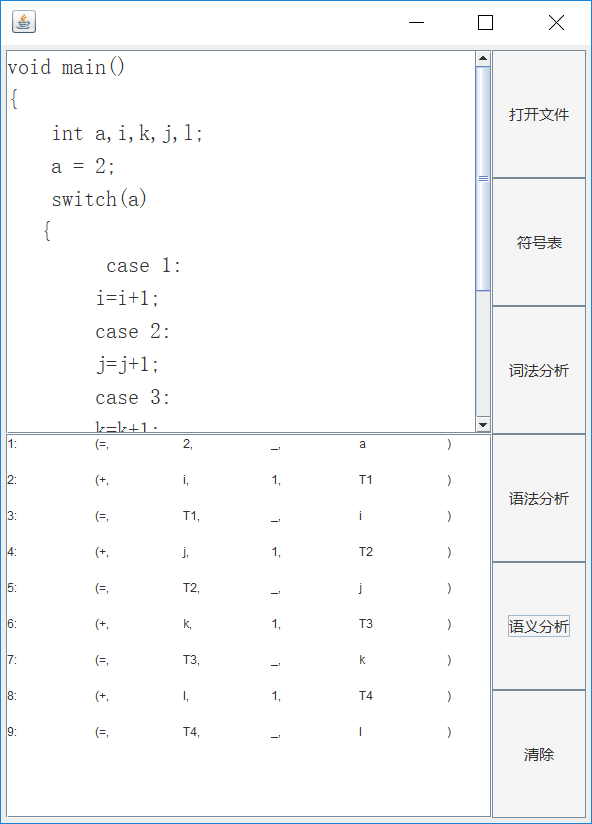
5.Do-while语句测试



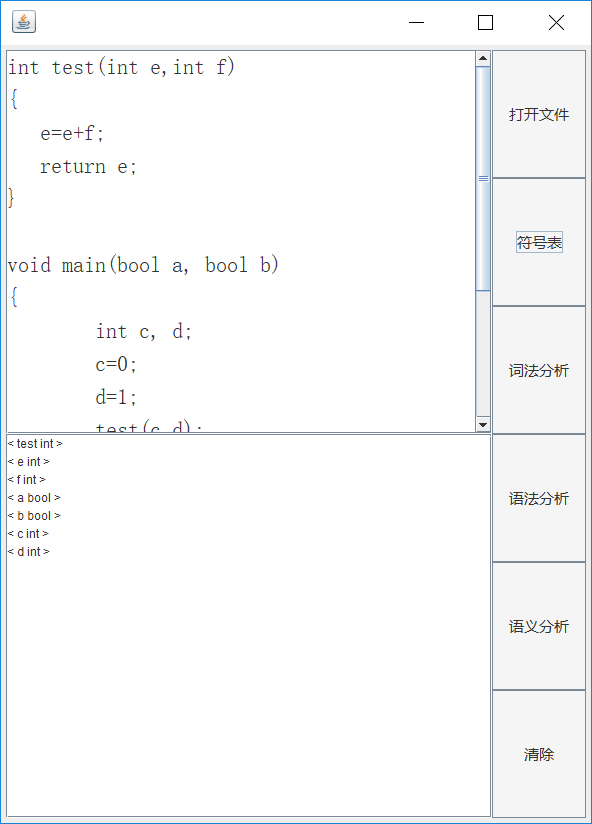
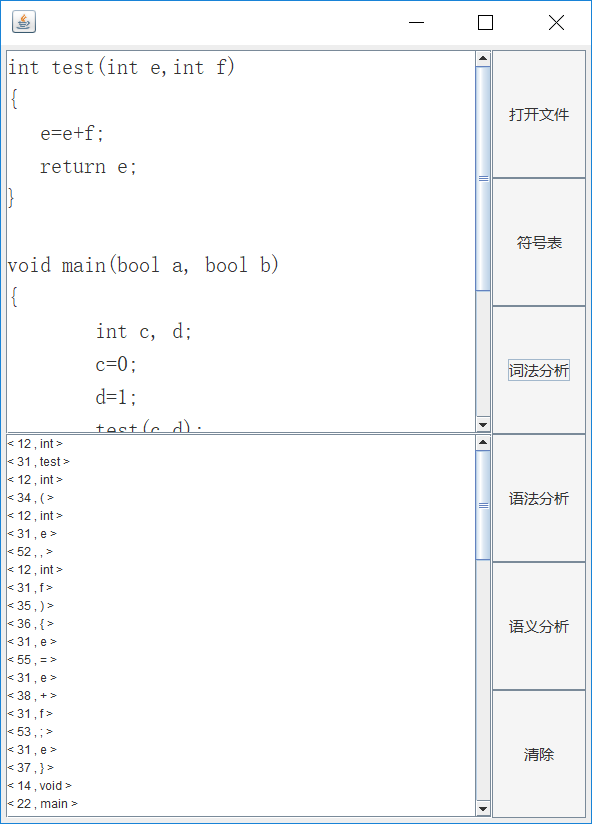


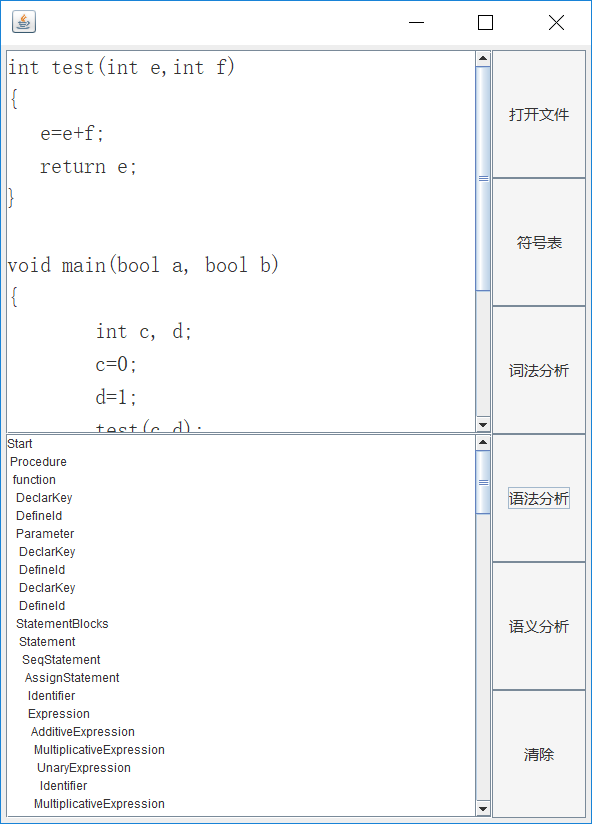
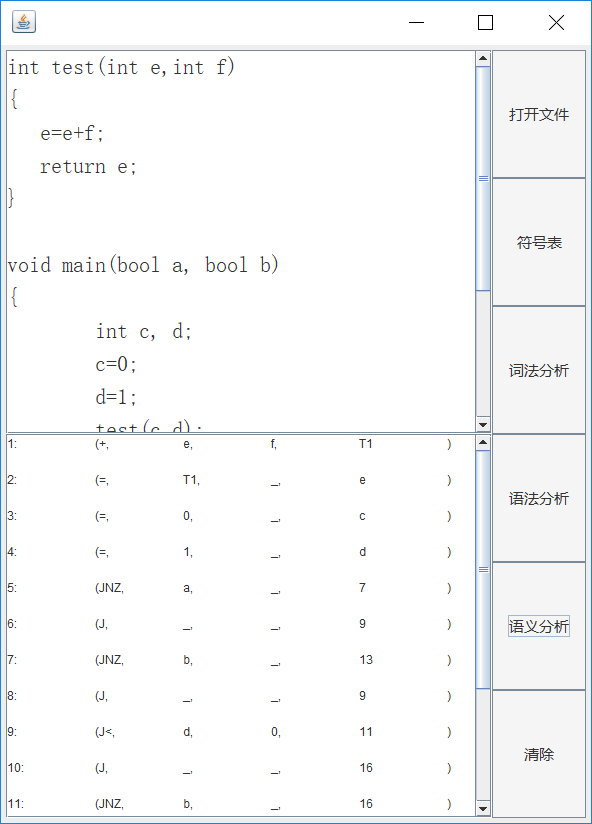
6.Switch-case语句测试

7.函数声明与函数调用

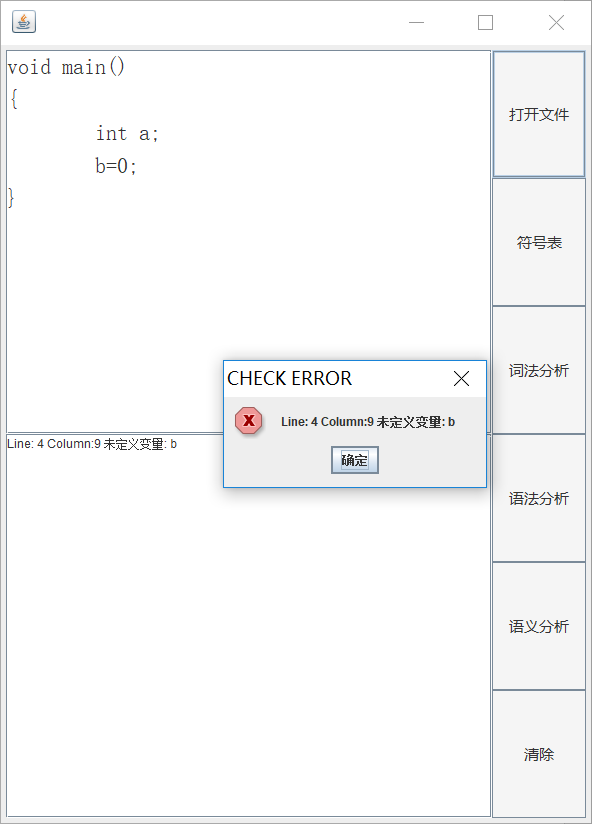
 

## 错误检查

1.变量重复定义错误



2．变量未定义错误



# 五、实习日志

12月16日

写实施计划书，分析题目，确定在以前实习的基础上要添加的功能；根据要新加的功能如布尔表达式、switch-case语句，在Token中新增加符号，如||、&&等，以便程序支持词法分析。

上午把之前实习一和实习二的代码复制过来调通了；下午再之前实习基础上加入了for语句、do\_while语句和switch\_case语句的语法。此处的条件还只是简单的两两比较或单个字符。For循环和do-while的语义子程序分析方式比较简单，参考课本上的解释后，真出口和假出口位置比较好确定，for循环真出口是在语句块开头位置，假出口是在语句块结尾之后，且需要在假出口前添加一个跳转语句到循环判断前。在相应位置回填真链假链即可。

12月17日

今天将昨天的for语句和switch\_case语句完善了，并且将实习三的内容完成。经过一天的学习和调试后终于可以输出各种语法的四元式了。

12月18日

要求的必要功能添加完毕后，开始尝试添加扩展功能的第一项，即考虑多个函数及函数调用翻译。为了实现函数定义功能，我添加了一个function()，识别是按照数据类型、标识符、左括号、参数、右括号、{、语句块、}顺序，这部分经测试没什么问题。但在写函数调用功能callfunc的时候遇到了问题：我定义了函数调用语句后，把它也作为可能出现的语句之一，以|的方式连接在了StatementBlocks中，但是会出现conflict警告和冲突。

12月19日

在之前的基础上加入了界面。为了方便显示，将结果输出为文件，点选相应按钮显示在屏幕上。界面调通之后开始制作ppt。

12月20日

上午进行答辩，通过老师的提问和同学的作品展示，了解了一些别人的实现逻辑，学习到了很多。下午写实习报告，整理实习材料。

# 六、实习总结

本次实习的工作量很大，所以需要高效率的进行编程，而且一些语法的细节部分也没来得及完善，通用性不够，以后有时间会继续完善和学习。

这次实习也让我加深了对课本内容的理解，并进行实践。新加的功能中布尔表达式难度较大，需要仔细研读和理解课本中的逻辑才能实现。

这次实习还让我深刻体会到了整体设计的重要性，如果整体架构不好，就会出现各种冲突错误。在编程过程中，我最常遇到的就是冲突警告，往往是由于可选择的分支中有分支的开头匹配相同的元素，如果不解决这个冲突，则冲突的语句可能都没法在该被调用的时候被调用，比如我曾让函数调用和变量赋值出现冲突，通过合并它们的变量标识符识别部分才解决。

感谢老师们一个星期以来的陪伴和指导。