

# 合肥工业大学

## 系统软件综合设计报告

### 编译原理分册

设计题目	小型类 C 语言编译器
学生姓名	孙淼
学    号	2018211958
专业班级	计算机科学与技术 18-2 班
指导教师	李宏芒
完成日期	2021.7.13

一、设计目的及设计要求

设计一个完整的小的编译程序模型，在 Java 语言为工具的基础上，实现从 C 语言（部分核心语法）到汇编语言的编译，进一步加深理解和掌握所学知识，完成词法分析，语法分析，语义分析和中间代码生成，最终产生目标汇编代码。

二、开发环境描述

操作系统：Windows 10, Intel(R) Core(TM) i7-8750H CPU @ 2.20GHz

集成开发环境：eclipse-java-2019-06-R-win32-x86\_64

三、设计内容、主要算法描述

对于本部分“设计内容、主要算法描述”，我将分为词法分析、语法分析、中间代码生成、目标代码生成四个主要部分进行介绍（也就是对程序的四个主要功能实现的 Java 类介绍），其中涉及到的其他小类就顺带在大类中介绍。

3.1) 词法分析 LexAnalyse

词法分析的定义是根据词法规则识别出源程序中的各个记号（token），每个记号代表一类单词（lexeme）。源程序中常见的记号可以归为几大类：关键字、标识符、字面量和特殊符号。词法分析器的输入是源程序，输出是识别的记号流。词法分析器的任务是把源文件的字符流转换成记号流。本质上它查看连续的字符然后把它们识别为“单词”。

在我设计的编译器中，词法分析的结果将被存于 wordList 的 ArrayList 中，同时将被存于 tiny Compiler 包下的 output 文件夹下的 wordList.txt 文件中。要对词法分析介绍，就得先介绍其中用到的小类 Word 类和 Error 类。

● Word 类

单词 Word 类包括四个成员变量：整型变量单词序号 id，字符串变量单词的值 value，字符串变量单词类型 type，整型变量单词所在行 line，布尔型变量单词合法标志 flag（但是构造函数时无需构造，默认为 true），如下表所示：

表 1 Word 类成员变量

数据类型	数据名字	数据含义
int	id	单词序号
String	value	单词值

String	type	单词类型
int	line	单词所在行
boolean	flag	单词合法标志

声明中的类型有以下多种：KEY（关键词）、OPERATOR（运算符），INT\_CONST（整形常量）、CHAR\_CONST（字符常量）、BOOL\_CONST（布尔常量）、IDENTIFYER（标志符）、BOUNDARYSIGN（界符）、END（结束符）、UNIDEF（未知类型）；

其中 KEY 类型包括以下符号：void、main、int、char、if、else、while、for、printf、scanf；其中 OPERATOR 类型包括以下符号：+、-、++、--、\*、/、>、<、>=、<=、==、!=、=、&&、||、!、.、?、|、&；其中 BOUNDARYSIGN 类型包括以下符号：（、）、{、}、;、,、;

除构造函数外，单词类 Word 还提供了 5 种成员函数如下表所示：（其中最后两种函数是 isOperator 函数的后续分流函数）

表 2 Word 类成员函数

级别	函数名字	函数功能
1	boolean isKey(String word)	判断单词 word 是否是关键词
1	boolean isOperator(String word)	判断单词 word 是否是运算符
1	boolean isBoundarySign(String word)	判断单词 word 是否是边界符
2	boolean isArOP(String word)	判断 word 是否是算数运算符
2	boolean isBoolOP(String word)	判断 word 是否是逻辑运算符

● Error 类

错误 Error 类有四个成员变量：整型变量错误序号 id，字符串变量 info 错误信息 info，整型变量错误所在行 line，Word 类对象错误的单词 word。其构造函数就是通过其参数构造一个 Error 类对象，如下表所示：

表 3 Error 类成员变量

数据类型	数据名字	数据含义
int	id	错误序号
String	info	错误信息
int	line	错误所在行
Word	word	错误的单词对象

● LexAnalyse 类

介绍完单词类 Word 和错误类 Error，下面就可以介绍 LexAnalyse 类了，其由 ArrayList 单词表 wordList（用于顺序存储词法分析得到的每个单词），ArrayList 错误信息表 errorList，整型变量单词个数 wordCount，整型变量 errorCount 错误个数，布尔类型变量多行注释标志 noteFlag，布尔类型变量词法分词错误标志 lexErrorFlag，如下表所示：

表 4 LexAnalyse 类成员变量

数据类型或结构	名称	含义与作用
ArrayList<Word>	wordlist	单词表，存储词法分析出的单词
ArrayList<Error>	errorList	错误信息表，存储词法分析出的错误信息
int	wordcount	单词个数，统计
int	errorCount	错误个数，统计
boolean	noteFlag	多行注释标志，检测到多行注释时置为 1，排除掉后置为 0
boolean	lexErrorFlag	词法分析出错标志

除构造函数外，本函数还提供以下成员函数，如下表所示：

表 5 LexAnalyse 类成员函数

函数名字	函数作用
boolean isDigit(char ch)	判断单个字符 ch 是否为数字
boolean isInteger(String word)	判断单词对象 word 的值是否为数字
boolean isLetter(char ch)	判断单个字符 ch 是否为字母
boolean isID(String word)	判断单词对象 word 的值是否是标识符
boolean isFail()	判断词法分析是否出错
void analyse(String str, int line)	对 line 行的字符串 str 进行词法分析
ArrayList<Word> lexAnalyse(String str)	对字符串 str，调用 analyse() 函数来进行词法分析，结果存到 wordList
ArrayList<Word> lexAnalyse1(String filePath)	对 filePath 路径内的文件，调用 analyse() 函数来进行词法分析，结果存到 wordList

String outputWordList()	把 wordList 中的词法分析结果写到路径下的/output/文件下，并返回这个文件的路径
-------------------------	---

其中最核心的函数是 void analyse(String str, int line)，也就是对 line 行的字符串 str 进行词法分析的具体函数。限于不能大量引用代码进行分析，我将用图片形式进行介绍，如下图 1 所示：

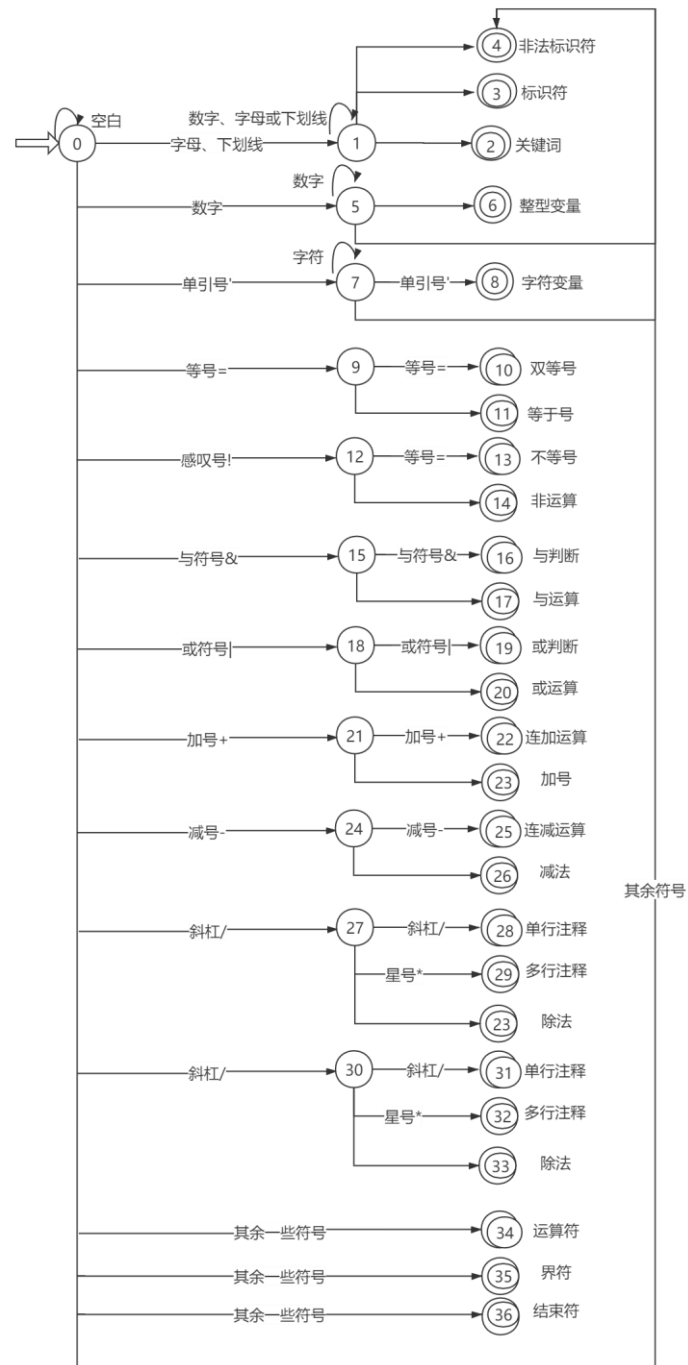


图 1 状态转换图

简单介绍一下上面的状态转换图的思路：

- 1) 首先读取第一个字符，根据第一个字符的判断结果，进行后续操作：
- 2) 如果第一个字符是字母或者下划线，继续判断后面的字符直到出现边界符、运算符、空字符、回车换行符、回车符、制表符才停下，停下后判断读取的 word，是关键字、标识符、或是非法标识符；
- 3) 如果第一个字符是数字，继续判断后面的字符直到出现边界符、运算符、空字符、回车换行符、回车符、制表符才停下，停下后判断读取的 word，是整型常量或是非法标识符；
- 4) 如果第一个字符是单引号'，继续判断后面的字符直到长度超出本行，或是另外一个单引号'，或是某位字符不是 0-255 的 ASCII 码才停下，停下后判断读取的 word，是字符常量或是非法标识符；
- 5) 如果第一个字符是等号=，继续判断紧接着的下一个字符是不是一个=，由此判断运算符=还是运算符==；
- 6) 如果第一个字符是等感叹号!，继续判断紧接着的下一个字符是不是一个=，由此判断运算符!还是运算符!=；
- 7) 如果第一个字符是与符号&，继续判断紧接着的下一个字符是不是一个&，由此判断运算符&还是运算符&&；
- 8) 如果第一个字符是或符号|，继续判断紧接着的下一个字符是不是一个|，由此判断运算符|还是运算符||；
- 9) 如果第一个字符是加号+，继续判断紧接着的下一个字符是不是一个+，由此判断运算符+还是运算符++；
- 10) 如果第一个字符是减号-，继续判断紧接着的下一个字符是不是一个-，由此判断运算符-还是运算符--；
- 11) 如果第一个字符是与符号/，继续判断紧接着的下一个字符是不是一个/，或是一个\*，由此判断是单行注释//，还是多行注释/\*，还是运算符/；
- 12) 如果以上都不是，就通过switch语法判断是否是' '、'\t'、'\r'、'\n'、'['、']'、'('、')'、'{'、'}'、','、'\"'、'. '、';'、'\*'、'% '、'>'、'<'、'? '、'#'，并按照Word中的分类对其进行类型判断，并且存入wordList
- 13) 如果上述都不是，那么算做非法标识符。

### 3.2) 语法分析 Parser

语法分析器根据语法规则识别出记号流中的结构（短语、句子），并构造一棵能够正确反映该结构的语法树。

在我设计的编译器中，语法分析是最困难的一步，结构最复杂。语法分析涉及到的最主要的预备知识和结构如下：

#### ● C 语言的 LL1 语法

为了适当降低难度，我选择的是 LL1 文法，具体文法（没有实现对 C 语言的所有语法，只是子集）如下：

1. 1、文法开始：
2.  $S \rightarrow \text{void main()}\{A\}$
- 3.
4. 2、声明：
5.  $X \rightarrow YZ;$
6.  $Y \rightarrow \text{int} | \text{char} | \text{bool}$
7.  $Z \rightarrow UZ'$
8.  $Z' \rightarrow \epsilon, Z | \$$
9.  $U \rightarrow \text{id}U'$
10.  $U' \rightarrow \epsilon | \$$
- 11.
12. 3、赋值：
13.  $R \rightarrow \text{id} = L;$
- 14.
15. 4、算术运算：
16.  $L \rightarrow TL'$
17.  $L' \rightarrow +L | -L | \$$
18.  $T \rightarrow FT'$
19.  $T' \rightarrow *T | /T | \$$
20.  $F \rightarrow (L)$
21.  $F \rightarrow \text{id} | \text{num}$
22.  $O \rightarrow ++ | -- | \$$
23.  $Q \rightarrow \text{id} 0 | \$$
- 24.
25. 5、布尔运算
26.  $E \rightarrow HE'$
27.  $E' \rightarrow \&\&E | \$$
28.  $H \rightarrow GH'$
29.  $H' \rightarrow | | H$
30.  $H' \rightarrow \$$
31.  $G \rightarrow FDF$
32.  $D \rightarrow < | > | == | !=$

```
33. G->(E)
34. G->!E
35.
36. 5、控制语句
37. B->if (E){A}else{A}
38. B->while(E){A}
39. B->for(YZ;G;Q){A}
40.
41. 6、功能函数
42. B->printf(P);
43. B->scanf(id);
44. P->id|ch|num
45.
46. 7、复合语句
47. A->CA
48. C->X|B|R
49. A->$
```

## ● 四元式类 FourElement

四元式 FourElement 类有五个成员变量：整型变量四元式序号 id，字符串变量操作符 op，字符串变量第一个操作数 arg1，字符串变量第二个操作数 arg2，字符串变量结果 result。其构造函数就是通过其参数构造一个 FourElement 类对象，如下表所示：

● 表 6 FourElement 类成员变量

数据类型	数据名字	数据含义
int	id	四元式序号
String	op	四元式操作符
String	arg1	第一个操作数
String	arg2	第二个操作数
String	result	结果

要进行语法分析，就需要从语法分析器得到的结果 wordList 中不断取出第一个数，与分析栈 analyseStack 之间进行配合分析，必要时将一些数据填入语义栈 semanticStack 和四元式列表 fourElemList。在语法分析这一步，最重要的就是上述四个“数据结构”。大致示意图如下：



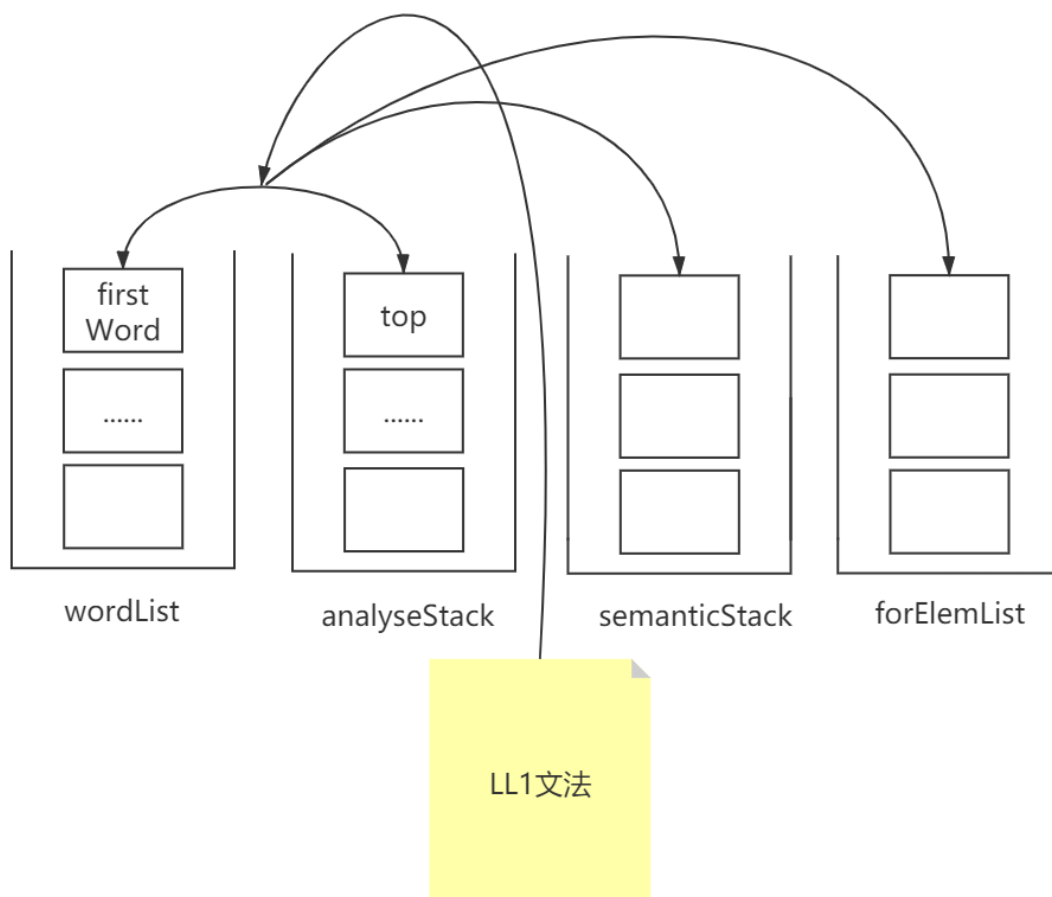


图2 语法分析示意图

LL(1)文法：从文法的开始符，向下推导，推出句子。对文法  $G$  的句子进行确定的自顶向下语法分析的充分必要条件是， $G$  的任意两个具有相同左部的产生式  $A \rightarrow \alpha \mid \beta$  满足下列条件：

- (1) 如果  $\alpha$ 、 $\beta$  均不能推导出  $\epsilon$ ，则  $FIRST(\alpha) \cap FIRST(\beta) = \emptyset$ 。
- (2)  $\alpha$  和  $\beta$  至多有一个能推导出  $\epsilon$ 。
- (3) 如果  $\beta \xRightarrow{*} \epsilon$ ，则  $FIRST(\alpha) \cap FOLLOW(A) = \emptyset$ 。

将满足上述条件的文法称为 LL(1)文法。

### ● 实例分析

由于语法分析比较复杂，为了更好的介绍语法分析，我将对实例中的一段 C 语言代码经过词法分析和语法分析的过程进行介绍，如下：

```
1. void main()
2. {
3.     int sum=0;
4.     for(int i=1;i<11;i++){
```

```

5.    sum=sum+i;
6.    }
7. }
8. #

```

这是我输入的要进行词法分析和语法分析的 C 语言代码段（限制代码段不能超过 10 行，所以不取太多）。

首先是词法分析，根据我们之前介绍的词法分析，编译器中的词法分析器，将会对每一行进行分析，提取出所有的单词 word 存入 wordList，最终结果是 44 个单词（含结束符#），也就是说单词表 wordList 结果如下：

1.	单词序号	单词的值	单词类型	单词所在行	单词是否合法
2.	1	<b>void</b>	关键字	1	<b>true</b>
3.	2	<b>main</b>	关键字	1	<b>true</b>
4.	3	(	界符	1	<b>true</b>
5.	4	)	界符	1	<b>true</b>
6.	5	{	界符	2	<b>true</b>
7.	6	<b>int</b>	关键字	3	<b>true</b>
8.	7	<b>sum</b>	标志符	3	<b>true</b>
9.	8	=	运算符	3	<b>true</b>
10.	9	0	整形常量	3	<b>true</b>
11.	10	;	界符	3	<b>true</b>
12.	11	<b>for</b>	关键字	4	<b>true</b>
13.	12	(	界符	4	<b>true</b>
14.	13	<b>int</b>	关键字	4	<b>true</b>
15.	14	<b>i</b>	标志符	4	<b>true</b>
16.	15	=	运算符	4	<b>true</b>
17.	16	1	整形常量	4	<b>true</b>
18.	17	;	界符	4	<b>true</b>
19.	18	<b>i</b>	标志符	4	<b>true</b>
20.	19	<	运算符	4	<b>true</b>
21.	20	11	整形常量	4	<b>true</b>
22.	21	;	界符	4	<b>true</b>
23.	22	<b>i</b>	标志符	4	<b>true</b>
24.	23	++	运算符	4	<b>true</b>
25.	24	)	界符	4	<b>true</b>
26.	25	{	界符	4	<b>true</b>
27.	26	<b>sum</b>	标志符	5	<b>true</b>
28.	27	=	运算符	5	<b>true</b>
29.	28	<b>sum</b>	标志符	5	<b>true</b>
30.	29	+	运算符	5	<b>true</b>
31.	30	<b>i</b>	标志符	5	<b>true</b>

32. 31	;	界符	5	true
33. 32	}	界符	6	true
34. 33	}	界符	7	true
35. 34	#	结束符	8	true
36. 词法分析通过:				

然后就是语法分析:

首先需要在语法分析栈中放入文法开始符和结束符 S#, 在语义栈中放入#。然后开始语法分析:

### 步骤 1)

S 被读取, 根据程序, 在下一步中, voidmain() {A}# 将被存入语法分析栈;

### 步骤 2)

当前语法分析栈: voidmain() {A}#

wordList 中: voidmain() {intsum=0;for(inti=1;i<11;i++) {sum=sum+i;}}#

语义栈: #

### 步骤 3)

可以看出此时 term.equals(firstWord.value), 所以可以将语法分析栈和 wordList 的栈顶元素相抵消, 并且接下来几个都可以抵消, 直到如下状态:

当前语法分析栈: A}#

wordList 中: intsum=0;for(inti=1;i<11;i++) {sum=sum+i;}}#

语义栈: #

此时根据文法, 应该使用 A->C, 结果如下一步骤

### 步骤 4)

当前语法分析栈: CA}#

wordList: intsum=0;for(inti=1;i<11;i++) {sum=sum+i;}}#

语义栈: #

此后都是按照文法, 判断, 进入不同的路径即可, 直到第一个赋值语句;

### 步骤 5)

由于前面的已经列出的文法的判断帮助, 文法将声明语句帮助我们识别赋值:

1. X->YZ;
2. Y->int|char|bool
3. Z->UZ'

4.  $Z' \rightarrow Z | \$$
5.  $U \rightarrow idU'$
6.  $U' \rightarrow =L | \$$

重点在于报告中要求“基于一遍扫描的语法制导翻译方法。算术表达式、赋值语句、布尔表达式、控制语句等语法单位的翻译模式。”

所以对于之前已经提到的文法，我们需要构造 LL1 属性翻译文法，即在原有 LL1 文法基础上加上动作符号，并给非终结符和终结符加上一定属性，给动作符号加上语义子程序，具体改动如下：

- 1.
2. 1、构造 LL1 属性翻译文法
3. 构造 LL1 属性翻译文法即在原有 LL1 文法基础上加上动作符号，并给非终结符和终结符加上一定属性，给动作符号加上语义子程序。
4. 对原有 LL1 文法改进的地方如下：
5. 1、赋值：
6. 产生式                      语义子程序
7.  $R \rightarrow @ASS\_R \ id \ =L@ \ EQ; \ @ASS\{R.VAL=id \text{ 并压入语义栈}\}$
8.  $@EQ\{RES=R.VAL, OP='=', ARG1=L.VAL,$
9.  $\text{new fourElement}(OP, ARG1, /, RES)\}$
10.  $U \rightarrow @ASS\_U idU' \quad \{U.VAL=id \text{ 并压入语义栈}\}$
11.  $U' \rightarrow =L | \$ @EQ\_U' \quad \{RES=U.VAL, OP='=', ARG1=L.VAL, \text{new fourElement}(OP, ARG1, /, RES)\}$
- 12.
13. 2、算术运算：
14. 产生式                      语义子程序
15.  $L \rightarrow TL' @ADD\_SUB \quad \{If(OP \neq null) \ RES = NEWTEMP; L.VAL=RES, \text{并压入语义栈}; \text{New fourElement}(OP, T.VAL, /, L'.VAL, RES),$
16.  $\}$
17.  $L' \rightarrow +L @ADD \quad \{OP=+, ARG2=L.VAL\}$
18.  $L' \rightarrow -L @SUB \quad \{OP=-, ARG2=L.VAL\}$
19.  $L' \rightarrow \$$
20.  $T \rightarrow FT' @DIV\_MUL \quad \{if \ (OP \neq null) \ RES = NEWTEMP; T.VAL=RES;$
21.  $\text{new FourElement}(OP, F.VAL, ARG2, RES)$
22.  $\text{else } ARG1=F.VAL; \}$
23.  $T' \rightarrow /T @DIV \quad \{OP=/, ARG2=T.VAL\}$
24.  $T' \rightarrow *T @MUL \quad \{OP=*, ARG2=T.VAL\}$
25.  $T' \rightarrow \$$
26.  $F \rightarrow (L) @VOL \quad \{F.VAL \rightarrow L.VAL\}$
27.  $F \rightarrow @ASS\_F \ num | id \quad \{F.VAL=num | id\}$
28.  $Q \rightarrow id0 | \$$
29.  $O \rightarrow @SINGLE\_OP ++ | -- \quad \{OP=++ | --\}$

```

30.
31. 3、布尔运算
32. 产生式          语义子程序
33. G->FDF@COMPARE{OP=D.VAL;ARG1=F(1).VAL;ARG2=F(2).VAL,RES=NEWTEMP;
34. New fourElement(OP,F.VAL,ARG2, RES );G.VAL=RES 并压入语义栈}
35. D->@COMPARE_OP<|>|==|!= {D.VAL=<|>|==|!=, 并压入语栈}
36.
37. 4、控制语句
38. 产生式          语义子程序
39. B->if (G)@IF_FJ{A}@IF_BACKPATCH_FJ @IF_RJ else{A}@IF_BACKPATCH_RJ
40. @IF_FJ{OP="FJ";ARG1=G.VAL;RES=if_fj, New fourElement(OP,ARG1,/, RES ),将其插入到四元式列表中第 i 个}
41. @IF_BACKPATCH_FJ{回填前面假出口跳转四元式的跳转序号, BACKPATCH (i,if_fj)}
42. B->while(G)@WHILE_FJ{A}@WHILE_RJ@WHILE_BACKPATCH_FJ      {参照 if else}
43. B->for(YZ;G@FOR_FJ;Q){A@SINGLE}@FOR_RJ@FOR_BACKPATCH_FJ      {参照 if else }
44. @SINGLE {ARG1=id;RES=NEWTEMP;New fourElement(OP,ARG1,/,RES)}
45.
46. 说明:
47. (1)、R.VAL 表示符号 R 的值, VAL 是 R 的一个属性, 其它类似。
48. (2)、NEWTEMP()函数: 每调用一次生成一个临时变量, 依次为 T1, T2, ...Tn。
49. (3)、BACKPATCH (int i,int res):回填函数, 用 res 回填第 i 个四元式的跳转地址。
50. (4)、new fourElement (String OP,String ARG1,String ARG2,String RES ):生成一个四元式
51. (OP,ARG1,ARG2,RES)

```

于是在遇到赋值、算术运算、布尔运算、控制语句，到达可以被属性翻译文法处理时，就遵照改进后的 LL1 属性翻译文法进行，也就是它的优先级更高，步骤如下：

### 步骤 6)

直到各栈等状态如下：

当前语法分析栈：UZ' ;BRA}#

wordList: sum=0;for(int i=1;i<11;i++){sum=sum+i;}}#

语义栈：#

于是根据优先级更高的属性翻译文法，翻译后如下步骤：

### 步骤 7)

当前语法分析栈：@ASS\_UidU' Z' ;BRA}#

wordList: sum=0;for(int i=1;i<11;i++){sum=sum+i;}}#

语义栈: #

这个@ASS\_U 的作用是将要赋值的对象拷入语义栈，然后删除语法分析栈中的@ASS\_U，对 wordList 无影响，运行后各栈状态如下一步骤：

#### 步骤 8)

当前语法分析栈: idU' Z' ;BRA} #

wordList: sum=0;for(int i=1;i<11;i++){sum=sum+i;}}

#语义栈: sum#

可以看到 sum 拷入语义栈

#### 步骤 9)

上一步之后，语法分析栈中剩余的是 id，它的作用是与 wordList 中的标识符相抵消，于是处理后如下一步：

#### 步骤 10)

当前语法分析栈: U' Z' ;BRA} #

余留符号串: =0;for(int i=1;i<11;i++){sum=sum+i;}} #

语义栈: sum#

此时 U' 将由属性翻译文法变为下一步骤所示；

#### 步骤 11)

当前语法分析栈: =L@EQ\_U' Z' ;BRA} #

余留符号串: =0;for(int i=1;i<11;i++){sum=sum+i;}} #

语义栈: sum#

很明显，等号将抵消，但是为保险起见，将语法分析栈中 L 的翻译文法，判断这个等号之前是否存在加减乘除号，如下：

#### 步骤 12)

当前语法分析栈: TL' @ADD\_SUB@EQ\_U' Z' ;BRA} #

余留符号串: 0;for(int i=1;i<11;i++){sum=sum+i;}} #

语义栈: sum#

这里是用于判断有无加减号的，然后下一步：

#### 步骤 13)

当前语法分析栈: FT' @DIV\_MULL' @ADD\_SUB@EQ\_U' Z' ;BRA} #

wordList: 0;for(int i=1;i<11;i++){sum=sum+i;}}#

语义栈:sum#

这里添加上了判断是否有乘除号。

为了存储要赋值给标识符的值，F 将转化为 F->@ASS\_Fnum，如下：

#### 步骤 14)

然后上一步中 wordList 中的 0 将被存入语义栈，num 与 wordList 中的数字 0 抵消，然后到下一步：

#### 步骤 15)

语法分析栈:T' @DIV\_MULL' @ADD\_SUB@EQ\_U' Z' ;BRA}#

wordList: ;for(int i=1;i<11;i++){sum=sum+i;}}#

语义栈:0sum#

此时的状态中，T' 将被用于判断是否有乘除，若无，T' 和@DIV\_MULL 都将失效，然后进入下一步骤：

#### 步骤 16)

语法分析栈:L' @ADD\_SUB@EQ\_U' Z' ;BRA}#

余留符号串: ;for(int i=1;i<11;i++){sum=sum+i;}}#

语义栈:0sum#

这一步中，L' 将被用于判断是否有加减，若无，L' 和@ADD\_SUB 都将失效，然后进入下一步骤：

#### 步骤 17)

语法分析栈:@EQ\_U' Z' ;BRA}#

余留符号串: ;for(int i=1;i<11;i++){sum=sum+i;}}#

语义栈:0sum#

此时就可以确保无误（由于是赋值语句不是判断等语句，所以只需要判断等号前面有无加减乘除即可），下一步将将这个赋值语句生成为四元式。

#### 步骤 18)

当前语法分析栈:@EQ\_U' Z' ;BRA}#

wordList: ;for(int i=1;i<11;i++){sum=sum+i;}}#

语义栈:0sum#

在此状态时，将生成四元式，其中操作符是等号，第一操作数是语义栈中的当前栈顶数 0，并出栈，结果是语义栈中的当前栈顶数 sum。四元式生成完毕后，将被存入四元式集合中。此后，将一直简单判断直到 wordList 栈顶为 for

#### 步骤 19)

此时，由于 wordList 栈顶是 for，

根据文法  $B \rightarrow \text{for}(YZ; G@FOR\_FJ; Q) \{A@SINGLE\} @FOR\_RJ@FOR\_BACKPATCH\_FJ$

将进入如下状态：

语法分析栈:  $\text{for}(YZ; G@FOR\_FJ; Q) \{A@SINGLE\} @FOR\_RJ@FOR\_BACKPATCH\_FJRA\} \#$

余留符号串:  $\text{for}(\text{inti}=1; i<11; i++) \{\text{sum}=\text{sum}+i; \} \#$

语义栈: #

这个针对 for 的属性翻译文法是有讲究的，可以看出

$\text{for}(YZ; G@FOR\_FJ; Q) \{A@SINGLE\} @FOR\_RJ@FOR\_BACKPATCH\_FJRA\}$

YZ 是之前用过的赋值语句，没什么好分析的，我们直接开始分析后面，状态如下：

#### 步骤 20)

步骤当前语法分析栈:  $G@FOR\_FJ; Q) \{A@SINGLE\} @FOR\_RJ@FOR\_BACKPATCH\_FJRA\} \#$

wordList:  $i<11; i++) \{\text{sum}=\text{sum}+i; \} \#$

语义栈: #

根据文法，唯一一种路径只能:  $G \rightarrow FDF@COMPARE$

然后进行下一步：

#### 步骤 21)

分析栈:  $FDF@COMPARE@FOR\_FJ; Q) \{A@SINGLE\} @FOR\_RJ@FOR\_BACKPATCH\_FJRA\} \#$

wordList:  $i<11; i++) \{\text{sum}=\text{sum}+i; \} \#$

语义栈: #

根据文法，由于此时 wordList 的栈顶是 i（标识符），所以下一步：

#### 步骤 22)

栈:  $@ASS\_FidDF@COMPARE@FOR\_FJ; Q) \{A@SINGLE\} @FOR\_RJ@FOR\_BACKPATCH\_FJRA\} \#$

wordList:  $i<11; i++) \{\text{sum}=\text{sum}+i; \} \#$  、

语义栈: #

于是和之前一样，ASS\_F 用于将标识符 i 加入到语义栈，id 用于和 wordList 中



的 i 相抵消, 然后就是 D, D 是一个双目运算符的文法符号, 用于判断==, !=, >, <。所以检测到 wordList 中是<, 将通过文法:  $D \rightarrow @COMPARE\_OP <$ ;

### 步骤 23)

语法分析栈: @COMPARE\_OP<F@COMPARE@FOR\_FJ;Q) {A@SINGLE} @FOR\_RJ  
@FOR\_BACKPATCH\_FJRA}#

余留符号串: <11;i++) {sum=sum+i;}}#

语义栈: i#

此时的@COMPARE\_OP 将用于把<入到语义栈, 然后<相抵消, 之后的 F 将用于读取数字 num1 到生成四元式, 如下一步:

### 步骤 24)

语法分析栈: @COMPARE@FOR\_FJ;Q) {A@SINGLE} @FOR\_RJ@FOR\_BACKPATCH\_FJRA}#  
wordList: ;i++) {sum=sum+i;}}#

语义栈: 11<i#

到这一步时, 语法分析栈中的@COMPARE 将用于生成一个四元式, 其中操作数 2 是语义栈栈顶的 11 并将其出栈, 操作符是语义栈栈顶的<并将其出栈, 操作数 1 是语义栈栈顶的 i, 然后四元式的结果 result 是通过 newTemp() 函数新建的一个字符串 "T"+tempCount (初始为 0++); 然后将这个四元式存入 fourElemList, 然后在语义栈中存入 result。如下一步:

### 步骤 25)

当前语法分析栈: @FOR\_FJ;Q) {A@SINGLE} @FOR\_RJ@FOR\_BACKPATCH\_FJRA}#

余留符号串: ;i++) {sum=sum+i;}}#

语义栈: T1#

此时, 语法分析栈栈顶是@FOR\_FJ, 其作用是将上一步存入语义栈的 T1 拿出生成一个四元式: (FJ, T1, /, 9), 然后将之后的两个;相抵消, 还有重要的作用就是将这一步生成的四元式的序号存入 Stack<Integer> for\_fj 也就是跳转地址栈, 进入下一步:

### 步骤 26)

当前语法分析栈: Q) {A@SINGLE} @FOR\_RJ@FOR\_BACKPATCH\_FJRA}#

wordList: i++) {sum=sum+i;}}#

语义栈: #

这个 Q 也是文法中单独设计的，因为我们知道在 C 语言中，for 语句的一般情况下语法是 for(赋值;判断;递增递减)（没有设计更复杂的），所以 Q 将用于识别这个递增递减部分，如下一步：

#### 步骤 27)

当前语法分析栈: @ASS\_Qid0) {A@SINGLE} @FOR\_RJ@FOR\_BACKPATCH\_FJRA} #

余留符号串: i++) {sum=sum+i;} #

语义栈: #

可以看出，@ASS\_Q 用于读取 i 到语义栈，id 用于和 i 相抵消，0 用于识别递增递减符号，其中 0 生成的 @SINGLE\_OP 用于将 ++ 或是一存入 Stack<String>for\_op，后面在 SINGLE 时，再将其取出这三个 top 识别完成后，状态如下一步（不需要存入四元式，因为 for 循环中的递增不是在这个时候进行，而是在循环完成一轮时进行）：

#### 步骤 28)

当前语法分析栈: {A@SINGLE} @FOR\_RJ@FOR\_BACKPATCH\_FJRA} #

余留符号串: {sum=sum+i;} #

语义栈: i #

A 就不必多说了，是用于读取  $sum = sum + i$ ，唯一需要注意的是这种赋值语句，不论是自身赋值  $a = a + b$ ，还是  $a = b + c$ ，都是只需要两个四元式就可以完成，之前在  $i = 1$  已经使用过了。那么 @SINGLE 在前面的介绍中我们也知道是用于读取之前读到 for\_op 中的递增或是递减符号，如下一步。

#### 步骤 29)

当前语法分析栈: @SINGLE} @FOR\_RJ@FOR\_BACKPATCH\_FJRA} #

余留符号串: }} #

语义栈: i #

这一步就完成了 i++ 的四元式化，下一步就进入到 for 循环的收尾阶段。

#### 步骤 30)

当前语法分析栈: @FOR\_RJ@FOR\_BACKPATCH\_FJRA} #

余留符号串: } #

语义栈: #

这里的@FOR\_RJ 和@FOR\_BACKPATCH\_FJRA 都是为了循环完成后的回溯准备的，思路如下：

```
1. }else if(top.name.equals("@FOR_RJ")){
2.     OP="RJ";
3.     RES=(for_fj.peek()-1)+"";
4.     FourElement fourElem=new FourElement(++fourElemCount,OP,"/", "/", RES);
5.     for_rj.push(fourElemCount);
6.     fourElemList.add(fourElem);
7.     OP=null;
8.     analyseStack.remove(0);
9. }else if(top.name.equals("@FOR_BACKPATCH_FJ")){
10.    backpatch(for_fj.pop(), fourElemCount+1);
11.    analyseStack.remove(0);
12. }
```

可以看出，@FOR\_RJ 是用于生成四元式(RJ, /, /, 3)的，这个 3 就来自于之前跳转地址栈 for\_fj 栈存入的 for 语句判断语句存入的四元式的序号 3，但是并不出栈找这个数字 3，因为后面的@FOR\_BACKPATCH 还要用到。存入了当前四元式的序号到地址栈 Stack<Integer> for\_rj 中，目的是为了退出循环，也是后面的@FOR\_BACKPATCH 还要用到。

步骤 31)

当前语法分析栈:@FOR\_BACKPATCH\_FJRA} #

余留符号串：} #

语义栈: #

根据上面的 12 行代码可知，对参数调用了 backpatch 函数，这个函数定义如下：

```
1. private void backpatch(int i,int res){
2.     FourElement temp=fourElemList.get(i-1);
3.     temp.result=res+"";
4.     fourElemList.set(i-1, temp);
5. }
```

作用很明显，就是将 for 循环的最后一个四元式序号+1（也就是退出 for 循环后的位置）作为参数，然后其传回到 for 循环的判断开始处，更新之前的判断四元式的 result，意即当那个判断为否时，就直接退出 for 循环，来到 for 循

环的下一行（退出循环）。

以上就是本语法分析器的主要作用，本报告以一个 11 行的 C 语言代码（包含赋值语句、判断语句、递减）的词法分析、语法分析、中间代码生成（四元式）的例子来进行功能、设计思想上的介绍。

语法分析中已经生成的中间代码将用于在下一步中进行目标代码（汇编语言）的生成。

### 3.3) 目标代码生成 huibian

中间代码生成器根据语义分析器的输出生成中间代码。中间代码可以有若干种形式，它们的共同特征是与具体机器无关。最常用的一种中间代码是三地址码，它的一种实现方式是四元式，在上一部分的语法分析中，我得到的中间代码结果，也就是四元式，如下：

1. 生成的四元式如下
2. 序号 (OP, ARG1, ARG2, RESULT)
3. 1(=, 0, /, sum)
4. 2(=, 1, /, i)
5. 3(<, i, 11, T1)
6. 4(FJ, T1, /, 9)
7. 5(+, sum, i, T2)
8. 6(=, T2, /, sum)
9. 7(++ , i, /, i)
10. 8(RJ, /, /, 3)

目标代码生成是编译器的最后一个阶段。在生成目标代码时要考虑以下几个问题：计算机的系统结构、指令系统、寄存器的分配以及内存的组织等。编译器生成的目标程序代码可以有多种形式：汇编语言、可重定位二进制代码、内存形式。这一步其实难度不大，有了四元式，想要生成目标代码就是简单的循环判断即可。

在验收的时候。蒋老师对我的工作给出了很高的评价，同时也指出更好的编译器最后一步汇编代码生成，应该是使用单词表，而不是使用逻辑循环来翻译。

如对上述的四元式结果，我们继续实例分析：

#### ● 实例分析

##### 步骤 1)

按照换行符进行分行，从第三行开始处理，通过逗号，进行分词，首先读取第一

个逗号前的符号，如果是=，那么就判断为赋值语句，配合四元式，产生目标汇编代码为：MOV sum, 0

#### 步骤 2)

仍旧是=，赋值语句，目标代码生成为：MOV i, 1

#### 步骤 3)

四元式首个符号为<，比较语句，目标代码生成为：CMP i, 11

#### 步骤 4)

四元式首个符号为F，下一个符号为J，目标代码生成为：JZ 11

（我是按照大二时学习的 8086 汇编代码来设计的，这两行代码的意思也就是比较 i 和 11，如果 11 小于 i，Less (JL) 就跳转到四元式中序号 9 的位置去）

#### 步骤 5)

四元式首个符号为+，运算符，目标代码生成为：MOV AX, i、MOV T2, sum、ADD T2, AX

#### 步骤 6)

四元式首个符号为=，赋值语句，目标代码生成为：MOV sum, T2

#### 步骤 7)

四元式首个符号为++，递增语句，目标代码生成为：INC i

#### 步骤 8)

四元式首个符号为RJ，for 的回溯，目标代码生成为：JMP 3

最后目标代码结果如下：

1. 生成的目标代码如下
2. 序号 (OP, ARG1, ARG2)
3. 1 MOV sum, 0
4. 2 MOV i, 1
5. 3 CMP i, 11
6. 4 JL 11
7. 5 MOV AX, i
8. 6 MOV T2, sum
9. 7 ADD T2, AX
10. 8 MOV sum, T2
11. 9 INC i
12. 10 JMP 3

### 3.4) 演示界面设计

演示代码的设计，由两个类组成：

- 主界面 MainFrame

主界面上设置三个 Panel，布局都是 BorderLayout。

然后在布局 North 加入组件 createUpPane，其中放入一些界面显示文字的控件，最主要的是浏览和确认按钮，其浏览按钮的动作函数是打开文件夹选择文件，确认按钮的动作函数是打开文件内容并且读取，然后显示在 textArea 上。

在布局 Center 加入组件 createCenterPane，其中放入一些界面显示文字的控件，最主要的是一个显示 textArea 控件，用于显示从文件中读取的源代码文件。

在布局 SOUTH 加入组件 createBottomPane，其中放入四个按钮控件，分别唤醒四个子界面 InfoFrame，分别显示词法分析，语法分析，中间代码生成，目标代码生成。

- 子界面 InfoFrame

显示一个有 title 的子界面，将每一步（词法分析、语法分析、中间代码生成、目标代码生成）的存入路径下的结果拿出，显示到子界面的 TextArea 中。

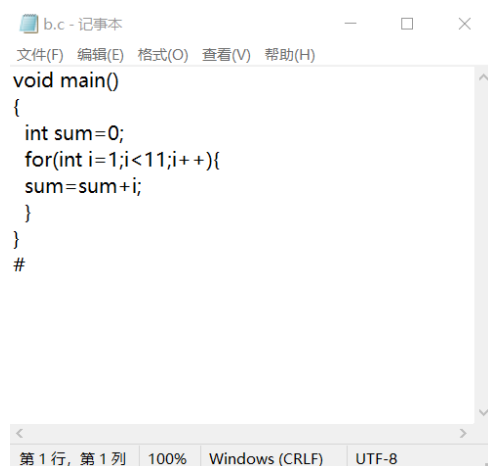
## 四、设计的输入和输出形式

输入内容是一个 C 语言源代码文件 a.c，如下所示：



图 3 源代码在文件夹中的形式

内容如下，其形式是一个完整的 C 语言源代码加上一个结束符#：



```
void main()
{
    int sum=0;
    for(int i=1;i<11;i++){
        sum=sum+i;
    }
}
#
```

图 4 被编译的 C 语言源代码

输出内容是在路径/output/下的三个文件 wordList.txt，用于存储词法分析结

果；LL1.txt，用于存储语法分析结果；FourElement.txt，用于存储四元式分析结果（中间代码）。

词法分析结果 wordList.txt 如下，其形式是五列数据，其内容分别是：单词序号、单词的值、单词类型、单词所在行、单词是否合法。

wordList.txt - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

单词序号	单词的值	单词类型	单词所在行	单词是否合法	
1	void	关键字	1	1	true
2	main	关键字	1	1	true
3	(	界符	1	1	true
4	)	界符	1	1	true
5	{	界符	2	2	true
6	int	关键字	3	3	true
7	sum	标志符	3	3	true
8	=	运算符	3	3	true
9	0	整形常量	3	3	true
10	;	界符	3	3	true
11	for	关键字	4	4	true
12	(	界符	4	4	true
13	int	关键字	4	4	true
14	i	标志符	4	4	true
15	=	运算符	4	4	true
16	1	整形常量	4	4	true
17	;	界符	4	4	true
18	i	标志符	4	4	true
19	<	运算符	4	4	true
20	11	整形常量	4	4	true
21	;	界符	4	4	true
22	i	标志符	4	4	true
23	++	运算符	4	4	true
24	)	界符	4	4	true
25	{	界符	4	4	true
26	sum	标志符	5	5	true
27	=	运算符	5	5	true
28	sum	标志符	5	5	true
29	+	运算符	5	5	true
30	i	标志符	5	5	true
31	;	界符	5	5	true
32	}	界符	6	6	true
33	}	界符	7	7	true
34	#	结束符	8	8	true

词法分析通过：

图 5 词法分析结果文件

语法分析结果 LL1.txt 如下，其形式是：步骤及序号、当前语法分析栈的内容（对



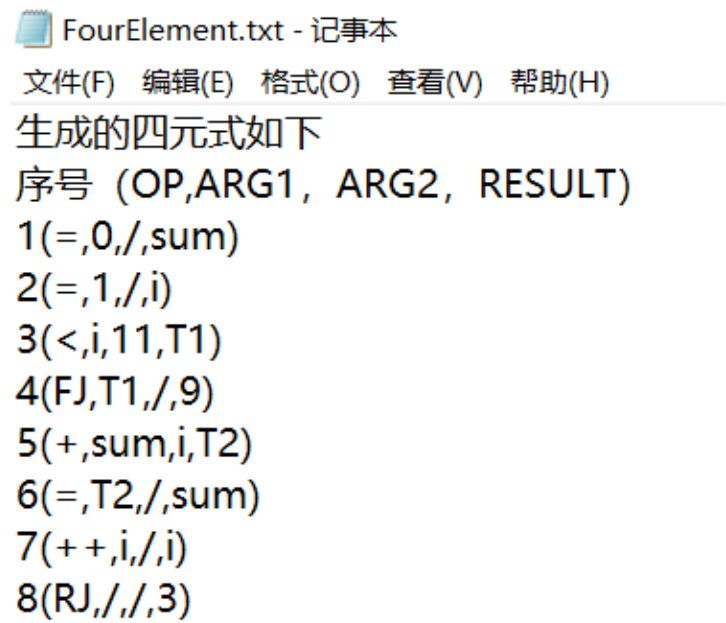
LL1.txt - 记事本

[illegible]

图 6 语法分析结果文件



四元式分析结果 FourElement.txt 如下，其形式是：四元式序号、四元式操作符、四元式操作数 1、四元式操作数 2、四元式结果。



```
生成的四元式如下
序号 (OP,ARG1, ARG2, RESULT)
1(=,0,/,sum)
2(=,1,/,i)
3(<,i,11,T1)
4(FJ,T1,/,9)
5(+,sum,i,T2)
6(=,T2,/,sum)
7(+ +,i,/,i)
8(RJ,/,/,3)
```

图 7 四元式分析结果文件

五、程序运行的结果

为了充分展示设计到的 C 语言语法,此处我们选择一个相对较为复杂,包含赋值,循环,判断,递增,四则运算等的的源代码作为输入,也就是 a.c,作为我的程序运行测试的样例。运行程序,初始界面如下:



图 8 程序运行初始界面

点击“浏览...”按钮，可以选择要编译的 C 语言文件，界面如下：

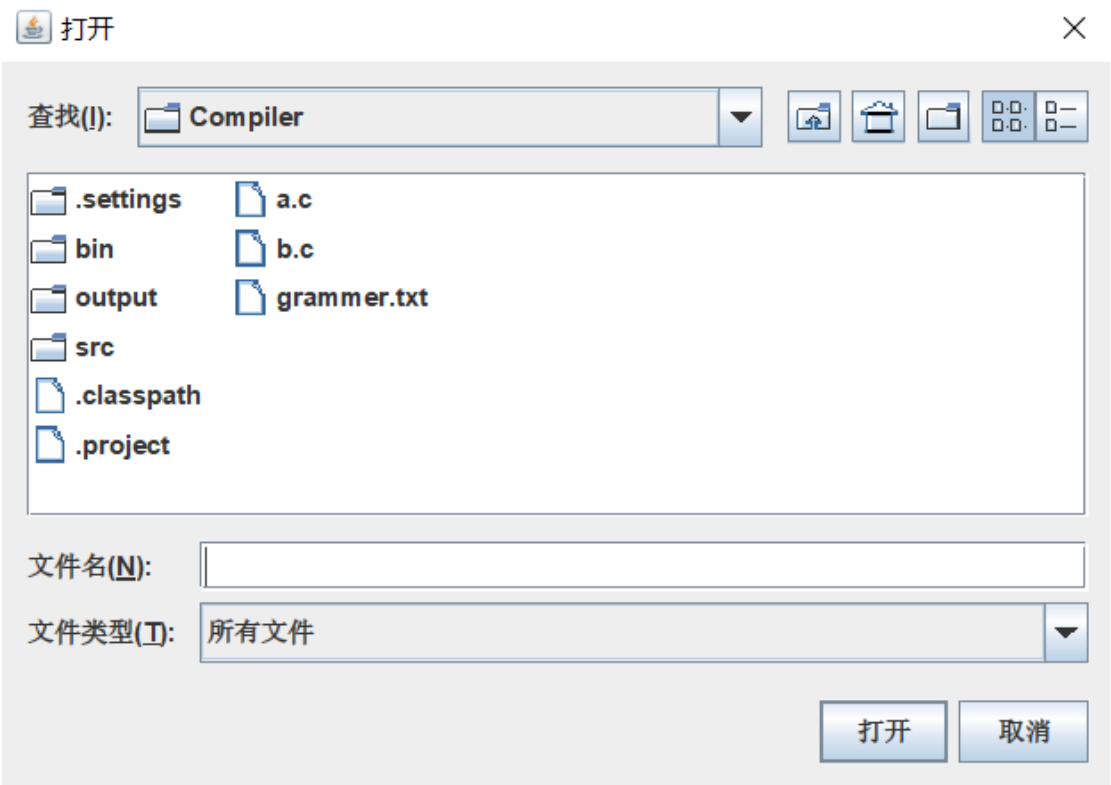


图 9 打开文件夹选择.c 文件功能界面

选择好之后源代码文件 b.c 之后，该文件路径将显示在 text 中。



图 10 选择好.c 文件后界面

然后点击按钮“确定”，就可以将文件显示在下面的 textArea 中，如下所示：



图 11 源程序展示界面

然后点击“词法分析”按钮，就会弹出子界面，里面是词法分析的结果：

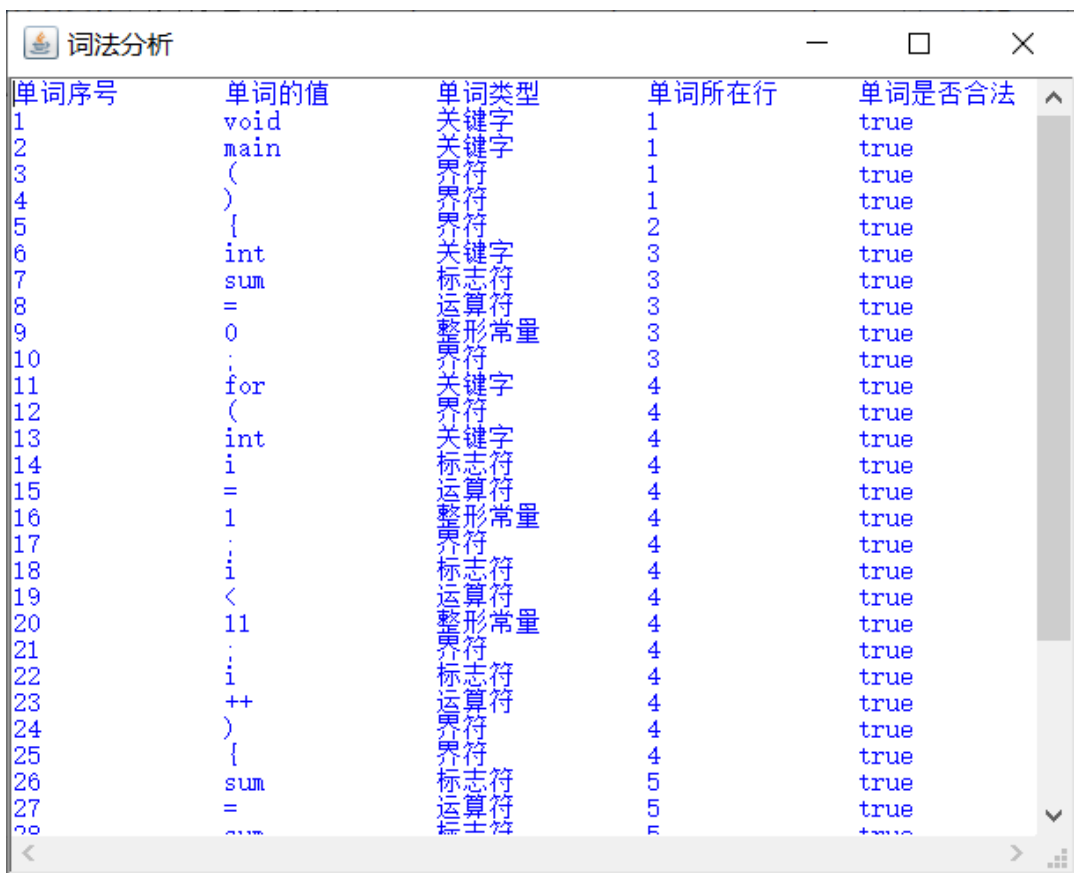


图 12 词法分析结果界面

然后点击“语法分析”按钮，就会弹出子界面，里面是语法分析的结果：



图 13 语法分析结果界面

然后点击“中间代码生成”按钮，就会弹出子界面，里面是四元式分析结果：

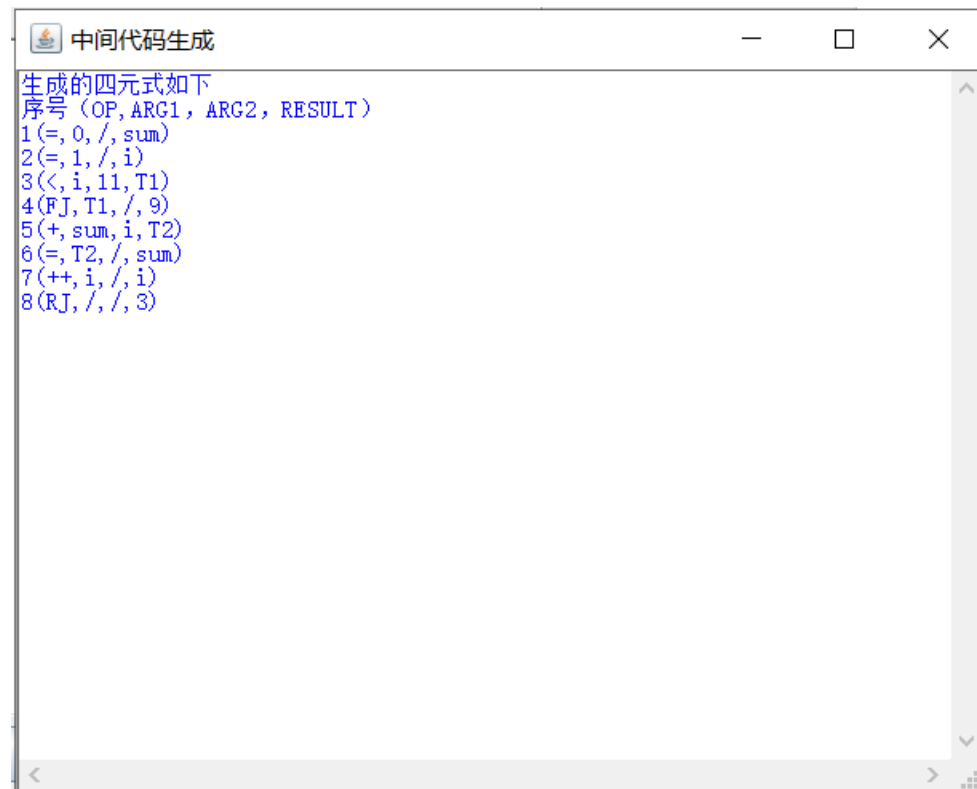


图 14 中间代码四元式分析结果界面

然后点击目标代码生成，就会弹出子界面，里面是目标代码（汇编语言）：

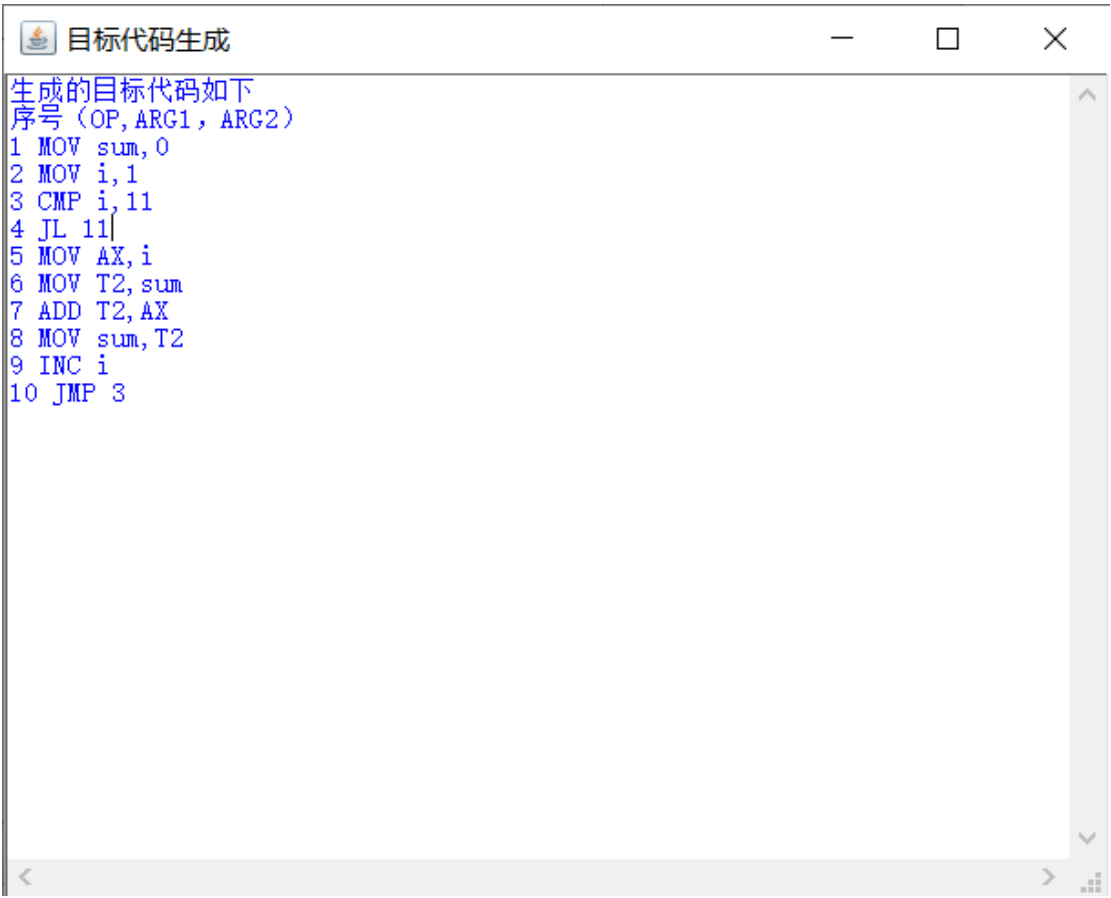


图 15 目标代码生成结果界面

至此，所有界面都已展示完毕，下面安装课设要求，拷贝结果的打印输出：

源文件 a.c:

```
1. void main()  
2. {  
3.     int sum=0;  
4.     for(int i=1;i<11;i++){  
5.         sum=sum+i;  
6.     }  
7. }  
8. #
```

词法分析结果:

1.	单词序号	单词的值	单词类型	单词所在行	单词是否合法
2.	1	void	关键字	1	true
3.	2	main	关键字	1	true
4.	3	(	界符	1	true
5.	4	)	界符	1	true

6.	5	{	界符	2	true
7.	6	int	关键字	3	true
8.	7	sum	标志符	3	true
9.	8	=	运算符	3	true
10.	9	0	整形常量	3	true
11.	10	;	界符	3	true
12.	11	for	关键字	4	true
13.	12	(	界符	4	true
14.	13	int	关键字	4	true
15.	14	i	标志符	4	true
16.	15	=	运算符	4	true
17.	16	1	整形常量	4	true
18.	17	;	界符	4	true
19.	18	i	标志符	4	true
20.	19	<	运算符	4	true
21.	20	11	整形常量	4	true
22.	21	;	界符	4	true
23.	22	i	标志符	4	true
24.	23	++	运算符	4	true
25.	24	)	界符	4	true
26.	25	{	界符	4	true
27.	26	sum	标志符	5	true
28.	27	=	运算符	5	true
29.	28	sum	标志符	5	true
30.	29	+	运算符	5	true
31.	30	i	标志符	5	true
32.	31	;	界符	5	true
33.	32	}	界符	6	true
34.	33	}	界符	7	true
35.	34	#	结束符	8	true
36. 词法分析通过:					

语法分析结果:

- 步骤 0 当前语法分析栈:voidmain(){A}# 余留符号串:  
voidmain(){intsum=0;for(inti=1;i<11;i++){sum=sum+i;}}# 语义栈:#
- 步骤 1 当前语法分析栈:main(){A}# 余留符号串:  
main(){intsum=0;for(inti=1;i<11;i++){sum=sum+i;}}# 语义栈:#
- 步骤 2 当前语法分析栈:(){A}# 余留符号串:  
(){intsum=0;for(inti=1;i<11;i++){sum=sum+i;}}# 语义栈:#
- 步骤 3 当前语法分析栈:){A}# 余留符号串:  
){intsum=0;for(inti=1;i<11;i++){sum=sum+i;}}# 语义栈:#
- 步骤 4 当前语法分析栈:{A}# 余留符号串:  
{intsum=0;for(inti=1;i<11;i++){sum=sum+i;}}# 语义栈:#

6. 步骤 5 当前语法分析栈:A)# 余留符号串:  
intsum=0;for(inti=1;i<11;i++){sum=sum+i;}}# 语义栈:#
7. 步骤 6 当前语法分析栈:CA)# 余留符号串:  
intsum=0;for(inti=1;i<11;i++){sum=sum+i;}}# 语义栈:#
8. 步骤 7 当前语法分析栈:XBRA)# 余留符号串:  
intsum=0;for(inti=1;i<11;i++){sum=sum+i;}}# 语义栈:#
9. 步骤 8 当前语法分析栈:YZ;BRA)# 余留符号串:  
intsum=0;for(inti=1;i<11;i++){sum=sum+i;}}# 语义栈:#
10. 步骤 9 当前语法分析栈:intZ;BRA)# 余留符号串:  
intsum=0;for(inti=1;i<11;i++){sum=sum+i;}}# 语义栈:#
11. 步骤 10 当前语法分析栈:Z;BRA)# 余留符号串:  
sum=0;for(inti=1;i<11;i++){sum=sum+i;}}# 语义栈:#
12. 步骤 11 当前语法分析栈:UZ';BRA)# 余留符号串:  
sum=0;for(inti=1;i<11;i++){sum=sum+i;}}# 语义栈:#
13. 步骤 12 当前语法分析栈:@ASS\_UidU'Z';BRA)# 余留符号串:  
sum=0;for(inti=1;i<11;i++){sum=sum+i;}}# 语义栈:#
14. 步骤 13 当前语法分析栈:idU'Z';BRA)# 余留符号串:  
sum=0;for(inti=1;i<11;i++){sum=sum+i;}}# 语义栈:sum#
15. 步骤 14 当前语法分析栈:U'Z';BRA)# 余留符号串:  
=0;for(inti=1;i<11;i++){sum=sum+i;}}# 语义栈:sum#
16. 步骤 15 当前语法分析栈:=L@EQ\_U'Z';BRA)# 余留符号串:  
=0;for(inti=1;i<11;i++){sum=sum+i;}}# 语义栈:sum#
17. 步骤 16 当前语法分析栈:L@EQ\_U'Z';BRA)# 余留符号串:  
0;for(inti=1;i<11;i++){sum=sum+i;}}# 语义栈:sum#
18. 步骤 17 当前语法分析栈:TL'@ADD\_SUB@EQ\_U'Z';BRA)# 余留符号串:  
0;for(inti=1;i<11;i++){sum=sum+i;}}# 语义栈:sum#
19. 步骤 18 当前语法分析栈:FT'@DIV\_MULL'@ADD\_SUB@EQ\_U'Z';BRA)# 余留符号串:  
0;for(inti=1;i<11;i++){sum=sum+i;}}# 语义栈:sum#
20. 步骤 19 当前语法分析栈:@ASS\_FnumT'@DIV\_MULL'@ADD\_SUB@EQ\_U'Z';BRA)# 余留符号串:  
0;for(inti=1;i<11;i++){sum=sum+i;}}# 语义栈:sum#
21. 步骤 20 当前语法分析栈:numT'@DIV\_MULL'@ADD\_SUB@EQ\_U'Z';BRA)# 余留符号串:  
0;for(inti=1;i<11;i++){sum=sum+i;}}# 语义栈:0sum#
22. 步骤 21 当前语法分析栈:T'@DIV\_MULL'@ADD\_SUB@EQ\_U'Z';BRA)# 余留符号串:  
;for(inti=1;i<11;i++){sum=sum+i;}}# 语义栈:0sum#
23. 步骤 22 当前语法分析栈:@DIV\_MULL'@ADD\_SUB@EQ\_U'Z';BRA)# 余留符号串:  
;for(inti=1;i<11;i++){sum=sum+i;}}# 语义栈:0sum#
24. 步骤 23 当前语法分析栈:L'@ADD\_SUB@EQ\_U'Z';BRA)# 余留符号串:  
;for(inti=1;i<11;i++){sum=sum+i;}}# 语义栈:0sum#
25. 步骤 24 当前语法分析栈:@ADD\_SUB@EQ\_U'Z';BRA)# 余留符号串:  
;for(inti=1;i<11;i++){sum=sum+i;}}# 语义栈:0sum#
26. 步骤 25 当前语法分析栈:@EQ\_U'Z';BRA)# 余留符号串:  
;for(inti=1;i<11;i++){sum=sum+i;}}# 语义栈:0sum#
27. 步骤 26 当前语法分析栈:Z';BRA)# 余留符号串:  
;for(inti=1;i<11;i++){sum=sum+i;}}# 语义栈:#

28. 步骤 27 当前语法分析栈:;BRA}# 余留符号串: ;for(inti=1;i<11;i++){sum=sum+i;}}# 语义栈: #
29. 步骤 28 当前语法分析栈: BRA}# 余留符号串: for(inti=1;i<11;i++){sum=sum+i;}}# 语义栈: #
30. 步骤 29 当前语法分析  
栈: for(YZ;G@FOR\_FJ;Q){A@SINGLE}@FOR\_RJ@FOR\_BACKPATCH\_FJRA}# 余留符号串: for(inti=1;i<11;i++){sum=sum+i;}}# 语义栈: #
31. 步骤 30 当前语法分析  
栈: (YZ;G@FOR\_FJ;Q){A@SINGLE}@FOR\_RJ@FOR\_BACKPATCH\_FJRA}# 余留符号串: (inti=1;i<11;i++){sum=sum+i;}}# 语义栈: #
32. 步骤 31 当前语法分析  
栈: YZ;G@FOR\_FJ;Q){A@SINGLE}@FOR\_RJ@FOR\_BACKPATCH\_FJRA}# 余留符号串: inti=1;i<11;i++){sum=sum+i;}}# 语义栈: #
33. 步骤 32 当前语法分析  
栈: intZ;G@FOR\_FJ;Q){A@SINGLE}@FOR\_RJ@FOR\_BACKPATCH\_FJRA}# 余留符号串: inti=1;i<11;i++){sum=sum+i;}}# 语义栈: #
34. 步骤 33 当前语法分析  
栈: Z;G@FOR\_FJ;Q){A@SINGLE}@FOR\_RJ@FOR\_BACKPATCH\_FJRA}# 余留符号串: i=1;i<11;i++){sum=sum+i;}}# 语义栈: #
35. 步骤 34 当前语法分析  
栈: UZ';G@FOR\_FJ;Q){A@SINGLE}@FOR\_RJ@FOR\_BACKPATCH\_FJRA}# 余留符号串: i=1;i<11;i++){sum=sum+i;}}# 语义栈: #
36. 步骤 35 当前语法分析  
栈: @ASS\_UidU'Z';G@FOR\_FJ;Q){A@SINGLE}@FOR\_RJ@FOR\_BACKPATCH\_FJRA}# 余留符号串: i=1;i<11;i++){sum=sum+i;}}# 语义栈: #
37. 步骤 36 当前语法分析  
栈: idU'Z';G@FOR\_FJ;Q){A@SINGLE}@FOR\_RJ@FOR\_BACKPATCH\_FJRA}# 余留符号串: i=1;i<11;i++){sum=sum+i;}}# 语义栈: i#
38. 步骤 37 当前语法分析  
栈: U'Z';G@FOR\_FJ;Q){A@SINGLE}@FOR\_RJ@FOR\_BACKPATCH\_FJRA}# 余留符号串: =1;i<11;i++){sum=sum+i;}}# 语义栈: i#
39. 步骤 38 当前语法分析  
栈: =L@EQ\_U'Z';G@FOR\_FJ;Q){A@SINGLE}@FOR\_RJ@FOR\_BACKPATCH\_FJRA}# 余留符号串: =1;i<11;i++){sum=sum+i;}}# 语义栈: i#
40. 步骤 39 当前语法分析  
栈: L@EQ\_U'Z';G@FOR\_FJ;Q){A@SINGLE}@FOR\_RJ@FOR\_BACKPATCH\_FJRA}# 余留符号串: 1;i<11;i++){sum=sum+i;}}# 语义栈: i#
41. 步骤 40 当前语法分析  
栈: TL'@ADD\_SUB@EQ\_U'Z';G@FOR\_FJ;Q){A@SINGLE}@FOR\_RJ@FOR\_BACKPATCH\_FJRA}# 余留符号串: 1;i<11;i++){sum=sum+i;}}# 语义栈: i#
42. 步骤 41 当前语法分析  
栈: FT'@DIV\_MULL'@ADD\_SUB@EQ\_U'Z';G@FOR\_FJ;Q){A@SINGLE}@FOR\_RJ@FOR\_BACKPATCH\_FJRA}# 余留符号串: 1;i<11;i++){sum=sum+i;}}# 语义栈: i#



43. 步骤 42 当前语法分析  
栈:@ASS\_FnumT'@DIV\_MULL'@ADD\_SUB@EQ\_U'Z';G@FOR\_FJ;Q){A@SINGLE}@FOR\_RJ@FOR\_BACKPATCH\_FJRA}# 余留符号串: 1;i<11;i++){sum=sum+i;}# 语义栈:i#
44. 步骤 43 当前语法分析  
栈:numT'@DIV\_MULL'@ADD\_SUB@EQ\_U'Z';G@FOR\_FJ;Q){A@SINGLE}@FOR\_RJ@FOR\_BACKPATCH\_FJRA}# 余留符号串: 1;i<11;i++){sum=sum+i;}# 语义栈:1i#
45. 步骤 44 当前语法分析  
栈:T'@DIV\_MULL'@ADD\_SUB@EQ\_U'Z';G@FOR\_FJ;Q){A@SINGLE}@FOR\_RJ@FOR\_BACKPATCH\_FJRA}# 余留符号串: ;i<11;i++){sum=sum+i;}# 语义栈:1i#
46. 步骤 45 当前语法分析  
栈:@DIV\_MULL'@ADD\_SUB@EQ\_U'Z';G@FOR\_FJ;Q){A@SINGLE}@FOR\_RJ@FOR\_BACKPATCH\_FJRA}# 余留符号串: ;i<11;i++){sum=sum+i;}# 语义栈:1i#
47. 步骤 46 当前语法分析  
栈:L'@ADD\_SUB@EQ\_U'Z';G@FOR\_FJ;Q){A@SINGLE}@FOR\_RJ@FOR\_BACKPATCH\_FJRA}# 余留符号串: ;i<11;i++){sum=sum+i;}# 语义栈:1i#
48. 步骤 47 当前语法分析  
栈:@ADD\_SUB@EQ\_U'Z';G@FOR\_FJ;Q){A@SINGLE}@FOR\_RJ@FOR\_BACKPATCH\_FJRA}# 余留符号串: ;i<11;i++){sum=sum+i;}# 语义栈:1i#
49. 步骤 48 当前语法分析  
栈:@EQ\_U'Z';G@FOR\_FJ;Q){A@SINGLE}@FOR\_RJ@FOR\_BACKPATCH\_FJRA}# 余留符号串: ;i<11;i++){sum=sum+i;}# 语义栈:1i#
50. 步骤 49 当前语法分析  
栈:Z';G@FOR\_FJ;Q){A@SINGLE}@FOR\_RJ@FOR\_BACKPATCH\_FJRA}# 余留符号串: ;i<11;i++){sum=sum+i;}# 语义栈:#
51. 步骤 50 当前语法分析  
栈:;G@FOR\_FJ;Q){A@SINGLE}@FOR\_RJ@FOR\_BACKPATCH\_FJRA}# 余留符号串: ;i<11;i++){sum=sum+i;}# 语义栈:#
52. 步骤 51 当前语法分析  
栈:G@FOR\_FJ;Q){A@SINGLE}@FOR\_RJ@FOR\_BACKPATCH\_FJRA}# 余留符号串: i<11;i++){sum=sum+i;}# 语义栈:#
53. 步骤 52 当前语法分析  
栈:FDF@COMPARE@FOR\_FJ;Q){A@SINGLE}@FOR\_RJ@FOR\_BACKPATCH\_FJRA}# 余留符号串: i<11;i++){sum=sum+i;}# 语义栈:#
54. 步骤 53 当前语法分析  
栈:@ASS\_FidDF@COMPARE@FOR\_FJ;Q){A@SINGLE}@FOR\_RJ@FOR\_BACKPATCH\_FJRA}# 余留符号串: i<11;i++){sum=sum+i;}# 语义栈:#
55. 步骤 54 当前语法分析  
栈:idDF@COMPARE@FOR\_FJ;Q){A@SINGLE}@FOR\_RJ@FOR\_BACKPATCH\_FJRA}# 余留符号串: i<11;i++){sum=sum+i;}# 语义栈:i#
56. 步骤 55 当前语法分析  
栈:DF@COMPARE@FOR\_FJ;Q){A@SINGLE}@FOR\_RJ@FOR\_BACKPATCH\_FJRA}# 余留符号串: <11;i++){sum=sum+i;}# 语义栈:i#

57. 步骤 56 当前语法分析  
栈:@COMPARE\_OP<F@COMPARE@FOR\_FJ;Q){A@SINGLE}@FOR\_RJ@FOR\_BACKPATCH\_FJRA}#  
余留符号串: <11;i++){sum=sum+i;}}# 语义栈:i#
58. 步骤 57 当前语法分析  
栈:<F@COMPARE@FOR\_FJ;Q){A@SINGLE}@FOR\_RJ@FOR\_BACKPATCH\_FJRA}# 余留符号串:  
<11;i++){sum=sum+i;}}# 语义栈:<i#
59. 步骤 58 当前语法分析  
栈:F@COMPARE@FOR\_FJ;Q){A@SINGLE}@FOR\_RJ@FOR\_BACKPATCH\_FJRA}# 余留符号串:  
11;i++){sum=sum+i;}}# 语义栈:<i#
60. 步骤 59 当前语法分析  
栈:@ASS\_Fnum@COMPARE@FOR\_FJ;Q){A@SINGLE}@FOR\_RJ@FOR\_BACKPATCH\_FJRA}# 余留符号串:  
11;i++){sum=sum+i;}}# 语义栈:<i#
61. 步骤 60 当前语法分析  
栈:num@COMPARE@FOR\_FJ;Q){A@SINGLE}@FOR\_RJ@FOR\_BACKPATCH\_FJRA}# 余留符号串:  
11;i++){sum=sum+i;}}# 语义栈:11<i#
62. 步骤 61 当前语法分析  
栈:@COMPARE@FOR\_FJ;Q){A@SINGLE}@FOR\_RJ@FOR\_BACKPATCH\_FJRA}# 余留符号串:  
;i++){sum=sum+i;}}# 语义栈:11<i#
63. 步骤 62 当前语法分析  
栈:@FOR\_FJ;Q){A@SINGLE}@FOR\_RJ@FOR\_BACKPATCH\_FJRA}# 余留符号串:  
;i++){sum=sum+i;}}# 语义栈:T1#
64. 步骤 63 当前语法分析栈:;Q){A@SINGLE}@FOR\_RJ@FOR\_BACKPATCH\_FJRA}# 余留符号串:  
;i++){sum=sum+i;}}# 语义栈:#
65. 步骤 64 当前语法分析栈:Q){A@SINGLE}@FOR\_RJ@FOR\_BACKPATCH\_FJRA}# 余留符号串:  
i++){sum=sum+i;}}# 语义栈:#
66. 步骤 65 当前语法分析  
栈:@ASS\_QidO){A@SINGLE}@FOR\_RJ@FOR\_BACKPATCH\_FJRA}# 余留符号串:  
i++){sum=sum+i;}}# 语义栈:#
67. 步骤 66 当前语法分析栈:ido){A@SINGLE}@FOR\_RJ@FOR\_BACKPATCH\_FJRA}# 余留符号串:  
i++){sum=sum+i;}}# 语义栈:i#
68. 步骤 67 当前语法分析栈:O){A@SINGLE}@FOR\_RJ@FOR\_BACKPATCH\_FJRA}# 余留符号串:  
++){sum=sum+i;}}# 语义栈:i#
69. 步骤 68 当前语法分析  
栈:@SINGLE\_OP++){A@SINGLE}@FOR\_RJ@FOR\_BACKPATCH\_FJRA}# 余留符号串:  
++){sum=sum+i;}}# 语义栈:i#
70. 步骤 69 当前语法分析栈:++){A@SINGLE}@FOR\_RJ@FOR\_BACKPATCH\_FJRA}# 余留符号串:  
++){sum=sum+i;}}# 语义栈:i#
71. 步骤 70 当前语法分析栈:){A@SINGLE}@FOR\_RJ@FOR\_BACKPATCH\_FJRA}# 余留符号串:  
){sum=sum+i;}}# 语义栈:i#
72. 步骤 71 当前语法分析栈:{A@SINGLE}@FOR\_RJ@FOR\_BACKPATCH\_FJRA}# 余留符号串:  
{sum=sum+i;}}# 语义栈:i#
73. 步骤 72 当前语法分析栈:A@SINGLE}@FOR\_RJ@FOR\_BACKPATCH\_FJRA}# 余留符号串:  
sum=sum+i;}}# 语义栈:i#

74. 步骤 73 当前语法分析栈:CA@SINGLE}@FOR\_RJ@FOR\_BACKPATCH\_FJRA}# 余留符号串:  
sum=sum+i;}}# 语义栈:i#
75. 步骤 74 当前语法分析栈:XBRA@SINGLE}@FOR\_RJ@FOR\_BACKPATCH\_FJRA}# 余留符号  
串: sum=sum+i;}}# 语义栈:i#
76. 步骤 75 当前语法分析栈:BRA@SINGLE}@FOR\_RJ@FOR\_BACKPATCH\_FJRA}# 余留符号串:  
sum=sum+i;}}# 语义栈:i#
77. 步骤 76 当前语法分析栈:RA@SINGLE}@FOR\_RJ@FOR\_BACKPATCH\_FJRA}# 余留符号串:  
sum=sum+i;}}# 语义栈:i#
78. 步骤 77 当前语法分析  
栈:@ASS\_Rid=L@EQ;A@SINGLE}@FOR\_RJ@FOR\_BACKPATCH\_FJRA}# 余留符号串:  
sum=sum+i;}}# 语义栈:i#
79. 步骤 78 当前语法分析栈:id=L@EQ;A@SINGLE}@FOR\_RJ@FOR\_BACKPATCH\_FJRA}# 余留  
符号串: sum=sum+i;}}# 语义栈:sumi#
80. 步骤 79 当前语法分析栈:=L@EQ;A@SINGLE}@FOR\_RJ@FOR\_BACKPATCH\_FJRA}# 余留符号  
串: =sum+i;}}# 语义栈:sumi#
81. 步骤 80 当前语法分析栈:L@EQ;A@SINGLE}@FOR\_RJ@FOR\_BACKPATCH\_FJRA}# 余留符号  
串: sum+i;}}# 语义栈:sumi#
82. 步骤 81 当前语法分析  
栈:TL'@ADD\_SUB@EQ;A@SINGLE}@FOR\_RJ@FOR\_BACKPATCH\_FJRA}# 余留符号串:  
sum+i;}}# 语义栈:sumi#
83. 步骤 82 当前语法分析  
栈:FT'@DIV\_MULL'@ADD\_SUB@EQ;A@SINGLE}@FOR\_RJ@FOR\_BACKPATCH\_FJRA}# 余留符号  
串: sum+i;}}# 语义栈:sumi#
84. 步骤 83 当前语法分析  
栈:@ASS\_FidT'@DIV\_MULL'@ADD\_SUB@EQ;A@SINGLE}@FOR\_RJ@FOR\_BACKPATCH\_FJRA}# 余  
留符号串: sum+i;}}# 语义栈:sumi#
85. 步骤 84 当前语法分析  
栈:idT'@DIV\_MULL'@ADD\_SUB@EQ;A@SINGLE}@FOR\_RJ@FOR\_BACKPATCH\_FJRA}# 余留符号  
串: sum+i;}}# 语义栈:sumsumi#
86. 步骤 85 当前语法分析  
栈:T'@DIV\_MULL'@ADD\_SUB@EQ;A@SINGLE}@FOR\_RJ@FOR\_BACKPATCH\_FJRA}# 余留符号  
串: +i;}}# 语义栈:sumsumi#
87. 步骤 86 当前语法分析  
栈:@DIV\_MULL'@ADD\_SUB@EQ;A@SINGLE}@FOR\_RJ@FOR\_BACKPATCH\_FJRA}# 余留符号串:  
+i;}}# 语义栈:sumsumi#
88. 步骤 87 当前语法分析  
栈:L'@ADD\_SUB@EQ;A@SINGLE}@FOR\_RJ@FOR\_BACKPATCH\_FJRA}# 余留符号串:+i;}}# 语  
义栈:sumsumi#
89. 步骤 88 当前语法分析  
栈:+L@ADD@ADD\_SUB@EQ;A@SINGLE}@FOR\_RJ@FOR\_BACKPATCH\_FJRA}# 余留符号串:  
+i;}}# 语义栈:sumsumi#
90. 步骤 89 当前语法分析  
栈:L@ADD@ADD\_SUB@EQ;A@SINGLE}@FOR\_RJ@FOR\_BACKPATCH\_FJRA}# 余留符号串:i;}}# 语  
义栈:sumsumi#

91. 步骤 90	当前语法分析	栈:TL'@ADD_SUB@ADD@ADD_SUB@EQ;A@SINGLE}@FOR_RJ@FOR_BACKPATCH_FJRA}# 余留符号串: i;}}# 语义栈:sumsumi#
92. 步骤 91	当前语法分析	栈:FT' <b>@DIV_MULL'</b> @ADD_SUB@ADD@ADD_SUB@EQ;A@SINGLE}@FOR_RJ@FOR_BACKPATCH_FJRA}# 余留符号串: i;}}# 语义栈:sumsumi#
93. 步骤 92	当前语法分析	栈:@ASS_FidT' <b>@DIV_MULL'</b> @ADD_SUB@ADD@ADD_SUB@EQ;A@SINGLE}@FOR_RJ@FOR_BACKPATCH_FJRA}# 余留符号串: i;}}# 语义栈:sumsumi#
94. 步骤 93	当前语法分析	栈:idt' <b>@DIV_MULL'</b> @ADD_SUB@ADD@ADD_SUB@EQ;A@SINGLE}@FOR_RJ@FOR_BACKPATCH_FJRA}# 余留符号串: i;}}# 语义栈:isumsumi#
95. 步骤 94	当前语法分析	栈:T' <b>@DIV_MULL'</b> @ADD_SUB@ADD@ADD_SUB@EQ;A@SINGLE}@FOR_RJ@FOR_BACKPATCH_FJRA}# 余留符号串: ;}}# 语义栈:isumsumi#
96. 步骤 95	当前语法分析	栈:@DIV_MULL'@ADD_SUB@ADD@ADD_SUB@EQ;A@SINGLE}@FOR_RJ@FOR_BACKPATCH_FJRA}# 余留符号串: ;}}# 语义栈:isumsumi#
97. 步骤 96	当前语法分析	栈:L'@ADD_SUB@ADD@ADD_SUB@EQ;A@SINGLE}@FOR_RJ@FOR_BACKPATCH_FJRA}# 余留符号串: ;}}# 语义栈:isumsumi#
98. 步骤 97	当前语法分析	栈:@ADD_SUB@ADD@ADD_SUB@EQ;A@SINGLE}@FOR_RJ@FOR_BACKPATCH_FJRA}# 余留符号串: ;}}# 语义栈:isumsumi#
99. 步骤 98	当前语法分析	栈:@ADD@ADD_SUB@EQ;A@SINGLE}@FOR_RJ@FOR_BACKPATCH_FJRA}# 余留符号串: ;}}# 语义栈:isumsumi#
100. 步骤 99	当前语法分析	栈:@ADD_SUB@EQ;A@SINGLE}@FOR_RJ@FOR_BACKPATCH_FJRA}# 余留符号串: ;}}# 语义栈:isumsumi#
101. 步骤 100	当前语法分析	栈:@EQ;A@SINGLE}@FOR_RJ@FOR_BACKPATCH_FJRA}# 余留符号串: ;}}# 语义栈:T2sumi#
102. 步骤 101	当前语法分析	栈:;A@SINGLE}@FOR_RJ@FOR_BACKPATCH_FJRA}# 余留符号串: ;}}# 语义栈:i#
103. 步骤 102	当前语法分析	栈:A@SINGLE}@FOR_RJ@FOR_BACKPATCH_FJRA}# 余留符号串: }}# 语义栈:i#
104. 步骤 103	当前语法分析	栈:@SINGLE}@FOR_RJ@FOR_BACKPATCH_FJRA}# 余留符号串: }}# 语义栈:i#
105. 步骤 104	当前语法分析	栈:@FOR_RJ@FOR_BACKPATCH_FJRA}# 余留符号串: }}# 语义栈:#
106. 步骤 105	当前语法分析	栈:@FOR_RJ@FOR_BACKPATCH_FJRA}# 余留符号串: }# 语义栈:#
107. 步骤 106	当前语法分析	栈:@FOR_BACKPATCH_FJRA}# 余留符号串: }# 语义栈:#
108. 步骤 107	当前语法分析	栈:RA}# 余留符号串: }# 语义栈:#

```

109. 步骤 108    当前语法分析栈:A}# 余留符号串: }#    语义栈:#
110. 步骤 109    当前语法分析栈:}# 余留符号串: }#    语义栈:#
111. 步骤 110    当前语法分析栈:# 余留符号串: # 语义栈:#
112. 步骤 111
113. 当前语法分析栈:    余留符号串:    语义栈:#
114. 语法分析通过:

```

中间代码生成结果:

```

1. 生成的四元式如下
2. 序号 (OP,ARG1, ARG2, RESULT)
3. 1(=,0,/,sum)
4. 2(=,1,/,i)
5. 3(<,i,11,T1)
6. 4(FJ,T1,/,9)
7. 5(+,sum,i,T2)
8. 6(=,T2,/,sum)
9. 7(++ ,i,/,i)
10. 8(RJ,/,/,3)
11.

```

目标代码生成结果:

```

1. 生成的目标代码如下
2. 序号 (OP,ARG1, ARG2)
3. 1 MOV sum,0
4. 2 MOV i,1
5. 3 CMP i,11
6. 4 JL 11
7. 5 MOV AX,i
8. 6 MOV T2,sum
9. 7 ADD T2,AX
10. 8 MOV sum,T2
11. 9 INC i
12. 10 JMP 3

```

以上就是所有结果的展示。

## 六、总结与体会

从知道编译器开始,就一直期望自己能动手设计一个简单编译器,但是学习编译原理的时候是大二,由于一直没有刻意去锻炼自己的代码能力,所以当时是心有余而力不足的状态,直到现在大三末,借着这次课设的“东风”,我顺水推舟地选择了“设计一个小的编译器”的题目。

设计一个编译器，难度不在于实现，在于设计，怎么去构思，怎么去把课程内学习到的知识运用进去，用什么样的结构，用什么样的算法，一步错的话就是步步错，我花了很长的时间去设计编译器的结构，最后在网易云课堂的一门在线课程中复习了相关的预备知识和大致的结构设计，其中对编译器的设计的描述是下面这样的：

编译器（Compiler）是一种将由一种语言编写的程序转换为另一种编程语言的可执行程序。现代软件对于编译器的需求远甚从前，究其原因很简单：作为中间层，编译器是构建更高层抽象的基础设施。编译器意欲将人类可阅读的高阶代码，翻译为机器能运行的低阶代码。现代编译器的主要工作流程为：源代码（source code）→ 预处理器（preprocessor）→ 编译器（compiler）→ 汇编程序（assembler）→ 目标代码（object code）→ 链接器（Linker）→ 可执行文件（executables）。其中，编译器位于一个最重要的位置：将源码转为汇编。

经过复杂的准备和设计，最终简单实现了一个编译器，我对编译器有了新的理解，但是新的问题出现，如果一个编译器是用来将一种语言翻译成另外一种语言的，那么第一个语言如何翻译？答案如下：“编译器自举一般都是编译器开发的一个里程碑事件，Pascal 编译器，其第一版编译器是用 Fortran 写的，而这也是常见的编译器自举过程的几乎必走的一条路，即最开始使用 X 语言（如 Fortran）实现 Y 语言（如 Pascal）的编译器，即解决鸡与蛋的问题，我们先要使用其它语言构建出我们的第一版编译器，之后成熟以后，就可以完全使用已经生成好的编译器来编译出我们的新编译器。”

整个实验做完，收获不在代码，而对于对编译器的理解加深，对编译原理这门课的宝贵实践经验。

## 七、源程序清单

按照要求：“如附源程序数量较大可以另外增加一文件单独存放源程序”，文件夹中另有附录存储以下源码。

Word. java

```
1. package compiler;
2.
3. import java.util.ArrayList;
```

```

4.
5. import compiler.LexAnalyse;
6.
7. /**
8.  * 单词类
9.  *
10.  * @author Administrator 1、单词序号 2、单词的值 3、单词类型 4、单词所在行 5、单
    词是否合法
11.  */
12. public class Word {
13.     public final static String KEY = "关键字";
14.     public final static String OPERATOR = "运算符";
15.     public final static String INT_CONST = "整形常量";
16.     public final static String CHAR_CONST = "字符常量";
17.     public final static String BOOL_CONST = "布尔常量";
18.     public final static String IDENTIFIER = "标识符";
19.     public final static String BOUNDARYSIGN = "界符";
20.     public final static String END = "结束符";
21.     public final static String UNIDEF = "未知类型";
22.     public static ArrayList<String> key = new ArrayList<String>();// 关键字集
    合
23.     public static ArrayList<String> boundarySign = new ArrayList<String>();//
    / 界符集合
24.     public static ArrayList<String> operator = new ArrayList<String>();// 运
    算符集合
25.     static {
26.         Word.operator.add("+");
27.         Word.operator.add("-");
28.         Word.operator.add("++");
29.         Word.operator.add("--");
30.         Word.operator.add("*");
31.         Word.operator.add("/");
32.         Word.operator.add(">");
33.         Word.operator.add("<");
34.         Word.operator.add(">=");
35.         Word.operator.add("<=");
36.         Word.operator.add("==");
37.         Word.operator.add("!=");
38.         Word.operator.add("=");
39.         Word.operator.add("&&");
40.         Word.operator.add("||");
41.         Word.operator.add("!");
42.         Word.operator.add(".");
43.         Word.operator.add("?");

```

```
44.         Word.operator.add("|");
45.         Word.operator.add("&");
46.         Word.boundarySign.add("(");
47.         Word.boundarySign.add(")");
48.         Word.boundarySign.add("{");
49.         Word.boundarySign.add("}");
50.         Word.boundarySign.add(";");
51.         Word.boundarySign.add(",");
52.         Word.key.add("void");
53.         Word.key.add("main");
54.         Word.key.add("int");
55.         Word.key.add("char");
56.         Word.key.add("if");
57.         Word.key.add("else");
58.         Word.key.add("while");
59.         Word.key.add("for");
60.         Word.key.add("printf");
61.         Word.key.add("scanf");
62.     }
63.     int id;// 单词序号
64.     String value;// 单词的值
65.     String type;// 单词类型
66.     int line;// 单词所在行
67.     boolean flag = true;//单词是否合法
68.
69.     public Word() {
70.
71.     }
72.
73.     public Word(int id, String value, String type, int line) {
74.         this.id = id;
75.         this.value = value;
76.         this.type = type;
77.         this.line = line;
78.     }
79.
80.     public static boolean isKey(String word) {
81.         return key.contains(word);
82.     }
83.
84.     public static boolean isOperator(String word) {
85.         return operator.contains(word);
86.     }
87.
```



```

88.     public static boolean isBoundarySign(String word) {
89.         return boundarySign.contains(word);
90.     }
91.
92.     public static boolean isArOP(String word) { // 判断单词是否为算术运算符
93.         if ((word.equals("+") || word.equals("-") || word.equals("*") || word
94.             .equals("/")))
95.             return true;
96.         else
97.             return false;
98.     }
99.
100.    public static boolean isBoolOP(String word) { // 判断单词是否为布尔运算符
101.        if ((word.equals(">") || word.equals("<") || word.equals("==")
102.            || word.equals("!=") || word.equals("!") || word.equals("&&")
103.            || word
104.                .equals("||")))
105.            return true;
106.        else
107.            return false;
108.    }

```

## Error.java

```

1. package compiler;
2.
3. public class Error {
4.     int id ; // 错误序号;
5.     String info; // 错误信息;
6.     int line ; // 错误所在行
7.     Word word; // 错误的单词
8.     public Error(){
9.
10.    }
11.
12.    public Error(int id, String info, int line, Word word){
13.        this.id=id;
14.        this.info=info;
15.        this.line=line;
16.        this.word=word;
17.    }

```

```
18. }
```

## LexAnalyse.java

```
1. package compiler;
2.
3. import java.io.BufferedInputStream;
4. import java.io.BufferedOutputStream;
5. import java.io.BufferedReader;
6. import java.io.File;
7. import java.io.FileInputStream;
8. import java.io.FileOutputStream;
9. import java.io.IOException;
10. import java.io.InputStreamReader;
11. import java.io.OutputStreamWriter;
12. import java.io.PrintWriter;
13. import java.util.ArrayList;
14.
15. /**
16.  * 词法分析器
17.  *
18.  *
19.  */
20. public class LexAnalyse {
21.
22.     ArrayList<Word> wordList = new ArrayList<Word>();//单词表
23.     ArrayList<Error> errorList = new ArrayList<Error>();//错误信息列表
24.     int wordCount = 0;// 统计单词个数
25.     int errorCount = 0;// 统计错误个数
26.     boolean noteFlag = false;// 多行注释标志
27.     boolean lexErrorFlag = false;// 词法分析出错标志
28.     public LexAnalyse() {
29.
30.     }
31.
32.     public LexAnalyse(String str) {
33.         lexAnalyse(str);
34.     }
35.
36.     /*以下是方法
37.     1) 为了确定初始位 便于继续 while
38.     2) 判断 while 得到的 word 属于 Word 类中的哪一种 type
39.     */
40.     /**
```

```
41.     * 单个字符 判断是否为数字
42.     *
43.     * @param
44.     * @return
45.     */
46.     private static boolean isDigit(char ch) {
47.         boolean flag = false;
48.         if ('0' <= ch && ch <= '9')
49.             flag = true;
50.         return flag;
51.     }
52.
53.     /**
54.     * 字符串 判断是否为数字 int
55.     *
56.     * @param string
57.     * @return
58.     */
59.     private static boolean isInteger(String word) {
60.         int i;
61.         boolean flag = false;
62.         for (i = 0; i < word.length(); i++) {
63.             if (Character.isDigit(word.charAt(i))) {
64.                 continue;
65.             } else {
66.                 break;
67.             }
68.         }
69.         if (i == word.length()) {
70.             flag = true;
71.         }
72.         return flag;
73.     }
74.
75.     /**
76.     * 字符串 判断是否都是\char\
77.     * @param
78.     * @return
79.     */
80.     private static boolean isChar(String word) {
81.         boolean flag = false;
82.         int i = 0;
83.         char temp = word.charAt(i);
84.         if (temp == '\\') {
```

```

85.         for (i = 1; i < word.length(); i++) {
86.             temp = word.charAt(i);
87.             if (0 <= temp && temp <= 255)
88.                 continue;
89.             else
90.                 break;
91.         }
92.         if (i + 1 == word.length() && word.charAt(i) == '\\')
93.             flag = true;
94.     } else
95.         return flag;
96.
97.     return flag;
98. }
99.
100. /**
101.  * 字符 判断是否为字母
102.  *
103.  * @param ch
104.  * @return
105.  */
106. private static boolean isLetter(char ch) {
107.     boolean flag = false;
108.     if (('a' <= ch && ch <= 'z') || ('A' <= ch && ch <= 'Z'))
109.         flag = true;
110.     return flag;
111. }
112.
113. /**
114.  * 字符串 判断是否为合法标识符 ID
115.  *
116.  * @param
117.  * @return
118.  */
119. private static boolean isID(String word) {
120.     boolean flag = false;
121.     int i = 0;
122.     if (Word.isKey(word))
123.         return flag;
124.     char temp = word.charAt(i);
125.     if (isLetter(temp) || temp == '_') { //字母或是_开头
126.         for (i = 1; i < word.length(); i++) {
127.             temp = word.charAt(i);

```

```

128.             if (isLetter(temp) || temp == '_' || isDigit(temp))//只含有
                字母、_、数字
129.                 continue;
130.             else
131.                 break;
132.         }
133.         if (i >= word.length())
134.             flag = true;
135.     } else
136.         return flag;
137.
138.     return flag;
139. }
140.
141. /**
142.  * 判断词法分析是否出错了
143.  *
144.  */
145. public boolean isFail() {
146.     return lexErrorFlag;
147. }
148.
149. public void analyse(String str, int line) { //str 都是处理掉空白符号的 按行
    处理
150.     int beginIndex;
151.     int endIndex;
152.     int index = 0;
153.     int length = str.length();
154.     Word word = null;
155.     Error error;
156.     // boolean flag=false;
157.     char temp;
158.     while (index < length) { //
159.         temp = str.charAt(index);
160.         if (!noteFlag) { //不是多行注释
161.             if (isLetter(temp) || temp == '_') { //如果是字母或者_开头, 就
                判断是不是标志符
162.                 beginIndex = index; //初始为 0
163.                 index++;
164.                 // temp=str.charAt(index);
165.                 while ((index < length) //while 循环 直到出现&&中的各种情
                    况

```

```

166.                && (!Word.isBoundarySign(str.substring(index, index + 1)))//看开头除外的后面的 从 beginIndex 开始取, 到 endIndex 结束, 从 0 开始数, 其中不包括 endIndex 位置的字符
167.                && (!Word.isOperator(str.substring(index, index + 1)))
168.                && (str.charAt(index) != ' ')
169.                && (str.charAt(index) != '\t')
170.                && (str.charAt(index) != '\r')
171.                && (str.charAt(index) != '\n'))
172.            {
173.                index++;
174.            }
175.            endIndex = index;
176.            word = new Word();
177.            wordCount++;
178.            word.id = wordCount;
179.            word.line = line;//按行处理
180.            word.value = str.substring(beginIndex, endIndex);//把刚刚 while 得到的词存入 wordList
181.            if (Word.isKey(word.value)) {
182.                word.type = Word.KEY;
183.            } else if (isID(word.value)) {
184.                word.type = Word.IDENTIFIER;
185.            } else { //如果上述都不是, 那就是一个非法标识符
186.                word.type = Word.UNIDEF;
187.                word.flag = false;
188.                errorCount++;
189.                error = new Error(errorCount, "非法标识符", word.line, word);
190.                errorList.add(error);
191.                lexErrorFlag = true;
192.            }
193.            index--; //
194.
195.        } else if (isDigit(temp)) { //如果初始位是数字 判断是不是 int 常数
196.            beginIndex = index;
197.            index++;
198.            while ((index < length)
199.                && (!Word.isBoundarySign(str.substring(index, index + 1)))
200.                && (!Word.isOperator(str.substring(index, index + 1)))
201.                && (str.charAt(index) != ' '))

```

```

202.                && (str.charAt(index) != '\t')
203.                && (str.charAt(index) != '\r')
204.                && (str.charAt(index) != '\n'))
205.            {
206.                index++;
207.            }
208.            endIndex = index;
209.            word = new Word();
210.            wordCount++; //lexAnalyse 中的全局变量
211.            word.id = wordCount;
212.            word.line = line;
213.            word.value = str.substring(beginIndex, endIndex);
214.            if (isInteger(word.value)) {
215.                word.type = Word.INT_CONST;
216.            } else {
217.                word.type = Word.UNIDEF;
218.                word.flag = false; //应该是用来标识不是错误符
219.                errorCount++;
220.                error = new Error(errorCount, "非法标识符", word.line, word);
221.                errorList.add(error);
222.                lexErrorFlag = true;
223.            }
224.            index--;
225.
226.        } else if (String.valueOf(str.charAt(index)).equals("'")) {
227.            //当第一个是字符是'时 字符常量
228.            //flag=true;
229.            beginIndex = index;
230.            index++;
231.            temp = str.charAt(index);
232.            while (index < length && (0 <= temp && temp <= 255)) {
233.                if (String.valueOf(str.charAt(index)).equals("'")) /
234.                //直到读到'
235.                break;
236.                index++;
237.                // temp=str.charAt(index);
238.            }
239.            if (index < length) {
240.                endIndex = index;
241.                word = new Word();
242.                wordCount++;
243.                word.id = wordCount;

```

```
242.         word.line = line;
243.         word.value = str.substring(beginIndex, endIndex);
244.         word.type = Word.CHAR_CONST;
245.         // flag=true;
246.         // word.flag=flag;
247.         index--;
248.     } else {
249.         endIndex = index;
250.         word = new Word();
251.         wordCount++;
252.         word.id = wordCount;
253.         word.line = line;
254.         word.value = str.substring(beginIndex, endIndex);
255.         word.type = Word.UNIDEF;
256.         word.flag = false;
257.         errorCount++;
258.         error = new Error(errorCount, "非法标识符", word.line, word);
259.         errorList.add(error);
260.         lexErrorFlag = true;
261.         index--;
262.     }
263. } else if (temp == '=') { //当第一个字符是 = 时 就往后再来一个 看是 = 还是 ==
264.     beginIndex = index;
265.     index++;
266.     if (index < length && str.charAt(index) == '=') {
267.         endIndex = index + 1;
268.         word = new Word();
269.         wordCount++;
270.         word.id = wordCount;
271.         word.line = line;
272.         word.value = str.substring(beginIndex, endIndex);
273.         word.type = Word.OPERATOR;
274.     } else {
275.         // endIndex=index;
276.         word = new Word();
277.         wordCount++;
278.         word.id = wordCount;
279.         word.line = line;
280.         word.value = str.substring(index - 1, index);
281.         word.type = Word.OPERATOR;
282.         index--;
283.     }
```



```
284.         } else if (temp == '!') { //当第一个字符是 ! 就往后再来一个看  
           是 ! 还是 !=  
285.             beginIndex = index;  
286.             index++;  
287.             if (index < length && str.charAt(index) == '=') {  
288.                 endIndex = index + 1;  
289.                 word = new Word();  
290.                 wordCount++;  
291.                 word.id = wordCount;  
292.                 word.line = line;  
293.                 word.value = str.substring(beginIndex, endIndex);  
294.                 word.type = Word.OPERATOR;  
295.                 index++;  
296.             } else {  
297.                 // endIndex=index;  
298.                 word = new Word();  
299.                 wordCount++;  
300.                 word.id = wordCount;  
301.                 word.line = line;  
302.                 word.value = str.substring(index - 1, index);  
303.                 word.type = Word.OPERATOR;  
304.                 index--;  
305.             }  
306.         } else if (temp == '&') { //当第一个字符是 & 就往后再来一个看  
           是 & 还是 &&  
307.             beginIndex = index;  
308.             index++;  
309.             if (index < length && str.charAt(index) == '&') {  
310.                 endIndex = index + 1;  
311.                 word = new Word();  
312.                 wordCount++;  
313.                 word.id = wordCount;  
314.                 word.line = line;  
315.                 word.value = str.substring(beginIndex, endIndex);  
316.                 word.type = Word.OPERATOR;  
317.             } else {  
318.                 // endIndex=index;  
319.                 word = new Word();  
320.                 wordCount++;  
321.                 word.id = wordCount;  
322.                 word.line = line;  
323.                 word.value = str.substring(index - 1, index);  
324.                 word.type = Word.OPERATOR;  
325.                 index--;
```

```

326.         }
327.     } else if (temp == '|') { //当第一个字符是 | 就往后再来一个看
        是 | 还是 ||
328.         beginIndex = index;
329.         index++;
330.         if (index < length && str.charAt(index) == '|') {
331.             endIndex = index + 1;
332.             word = new Word();
333.             wordCount++;
334.             word.id = wordCount;
335.             word.line = line;
336.             word.value = str.substring(beginIndex, endIndex);
337.             word.type = Word.OPERATOR;
338.         } else {
339.             // endIndex=index;
340.             word = new Word();
341.             wordCount++;
342.             word.id = wordCount;
343.             word.line = line;
344.             word.value = str.substring(index - 1, index);
345.             word.type = Word.OPERATOR;
346.             index--;
347.         }
348.     } else if (temp == '+') { //当第一个字符是 + 就往后再来一个看
        是 + 还是 ++
349.         beginIndex = index;
350.         index++;
351.         if (index < length && str.charAt(index) == '+') {
352.             endIndex = index + 1;
353.             word = new Word();
354.             wordCount++;
355.             word.id = wordCount;
356.             word.line = line;
357.             word.value = str.substring(beginIndex, endIndex);
358.             word.type = Word.OPERATOR;
359.
360.         } else {
361.             // endIndex=index;
362.             word = new Word();
363.             wordCount++;
364.             word.id = wordCount;
365.             word.line = line;
366.             word.value = str.substring(index - 1, index);
367.             word.type = Word.OPERATOR;

```

```

368.             index--;
369.         }
370.     } else if (temp == '-') { //当第一个字符是 - 就往后再来一个看
        是 - 还是 --
371.         beginIndex = index;
372.         index++;
373.         if (index < length && str.charAt(index) == '-') {
374.             endIndex = index + 1;
375.             word = new Word();
376.             wordCount++;
377.             word.id = wordCount;
378.             word.line = line;
379.             word.value = str.substring(beginIndex, endIndex);
380.             word.type = Word.OPERATOR;
381.         } else {
382.             // endIndex=index;
383.             word = new Word();
384.             wordCount++;
385.             word.id = wordCount;
386.             word.line = line;
387.             word.value = str.substring(index - 1, index);
388.             word.type = Word.OPERATOR;
389.             index--;
390.         }
391.     } else if (temp == '/') { //当第一个字符是 / 就往后再来一个看
        是 / 还是 /* 还是//
392.         index++;
393.         if (index < length && str.charAt(index) == '/')
394.             break; //单行注释就不加了 直接跳出这一行 因为 单行注释只
                能在这行的最后出现 反正也不需要编译
395.         /*
396.         * { index++; while(str.charAt(index)!='\n'){ index++;
        } }
397.         */
398.         else if (index < length && str.charAt(index) == '*') { /
            /如果是多行注释
399.             noteFlag = true;
400.         } else { //只是单纯的除法
401.             word = new Word();
402.             wordCount++;
403.             word.id = wordCount;
404.             word.line = line;
405.             word.value = str.substring(index - 1, index);
406.             word.type = Word.OPERATOR;

```

```
407.         }
408.         index--;
409.     } else { // 不是标识符、数字常量、字符串常量
410.
411.         switch (temp) {
412.             case ' ':
413.             case '\t':
414.             case '\r':
415.             case '\n':
416.                 word = null;
417.                 break; // 过滤空白字符
418.             case '[':
419.             case ']':
420.             case '(':
421.             case ')':
422.             case '{':
423.             case '}':
424.             case ',':
425.             case '"':
426.             case '.':
427.             case ';':
428.                 // case '+':
429.                 // case '-':
430.             case '*':
431.                 // case '/':
432.             case '%':
433.             case '>':
434.             case '<':
435.             case '?':
436.             case '#':
437.                 word = new Word();
438.                 word.id = ++wordCount;
439.                 word.line = line;
440.                 word.value = String.valueOf(temp);
441.                 if (Word.isOperator(word.value))
442.                     word.type = Word.OPERATOR;
443.                 else if (Word.isBoundarySign(word.value))
444.                     word.type = Word.BOUNDARYSIGN;
445.                 else
446.                     word.type = Word.END;
447.                 break;
448.             default: //没考虑到的非法标识符
449.                 word = new Word();
450.                 wordCount++;
```

```

451.                word.id = wordCount;
452.                word.line = line;
453.                word.value = String.valueOf(temp);
454.                word.type = Word.UNIDEF;
455.                word.flag = false;
456.                errorCount++;
457.                error = new Error(errorCount, "非法标识符
    ", word.line, word);
458.                errorList.add(error);
459.                lexErrorFlag = true;
460.            }
461.        }
462.    } else { //如果前面发现了/* 也就是多行注释 那么通过 indexOf 找到多行注
        释的结尾 把注释跳过
463.        int i = str.indexOf("*/");
464.        if (i != -1) {
465.            noteFlag = false;
466.            index = i + 2;
467.            continue;
468.        } else
469.            break;
470.    }
471.    if (word == null) {
472.        index++;
473.        continue;
474.    }
475.    wordList.add(word);
476.    index++;
477.    }
478.    }
479.
480.    public ArrayList<Word> lexAnalyse(String str) {
481.        String buffer[];
482.        buffer = str.split("\n");
483.        int line = 1;
484.        for (int i = 0; i < buffer.length; i++) {
485.            analyse(buffer[i].trim(), line);
486.            line++;
487.        }
488.        if (!wordList.get(wordList.size() - 1).type.equals(Word.END)) {
489.            Word word = new Word(++wordCount, "#", Word.END, line++);
490.            wordList.add(word);
491.        }
492.        return wordList;

```

```

493.     }
494.
495.     public ArrayList<Word> lexAnalyse1(String filePath) throws IOException
    {
496.         FileInputStream fis = new FileInputStream(filePath);
497.         BufferedInputStream bis = new BufferedInputStream(fis);
498.         InputStreamReader isr = new InputStreamReader(bis, "utf-8");
499.         BufferedReader inbr = new BufferedReader(isr);
500.         String str = "";
501.         int line = 1;
502.         while ((str = inbr.readLine()) != null) {
503.             // System.out.println(str);
504.             analyse(str.trim(), line);
505.             line++;
506.         }
507.         inbr.close();
508.         if (!wordList.get(wordList.size() - 1).type.equals(Word.END)) {
509.             Word word = new Word(++wordCount, "#", Word.END, line++);
510.             wordList.add(word);
511.         }
512.         return wordList;
513.     }
514.
515.     public String outputWordList() throws IOException { //处理之后, 返回一个文件
        名 供 InfoFrame 使用
516.         File file = new File("./output/");
517.         if (!file.exists()) {
518.             file.mkdirs();
519.             file.createNewFile(); // 如果这个文件不存在就创建它
520.         }
521.         String path = file.getAbsolutePath();
522.         FileOutputStream fos = new FileOutputStream(path + "/wordList.txt")
        ;
523.         BufferedOutputStream bos = new BufferedOutputStream(fos);
524.         OutputStreamWriter osw1 = new OutputStreamWriter(bos, "utf-8");
525.         PrintWriter pw1 = new PrintWriter(osw1);
526.         pw1.println("单词序号\t单词的值\t单词类型\t单词所在行\t单词是否合法
        "); //mainframe 显示的只是 compiler 里面的, 难道 mainframe 与 backup 无关?
527.         Word word;
528.         for (int i = 0; i < wordList.size(); i++) { //数据取自 wordList
529.             word = wordList.get(i);
530.             if (isInteger(word.value)) { //调整一下位置 使之对齐
531.                 pw1.println(word.id + "\t\t" + word.value + "\t\t" + word.t
                ype + "\t" + word.line + "\t\t" + word.flag);
            }
        }
    }

```

```

532.         }
533.         else {
534.             pw1.println(word.id + "\t\t" + word.value + "\t\t" + word.type
+ "\t" + "\t" + word.line + "\t\t" + word.flag);
535.         }
536.     }
537.     if (lexErrorFlag) {
538.         Error error;
539.         pw1.println("错误信息如下: ");
540.
541.         pw1.println("错误序号\t 错误信息\t 错误所在行 \t 错误单词");
542.         for (int i = 0; i < errorList.size(); i++) {
543.             error = errorList.get(i);
544.             pw1.println(error.id + "\t" + error.info + "\t\t" + error.l
ine
+ "\t" + error.word.value);
545.         }
546.     } else {
547.         pw1.println("词法分析通过: ");
548.     }
549.     pw1.close();
550.     return path + "/wordList.txt";
551. }
552.
553.
554.
555.
556.     public static void main(String[] args) throws IOException {
557.         LexAnalyse lex = new LexAnalyse();
558.         lex.lexAnalyse1("b.c");
559.         lex.outputWordList();
560.     }
561. }

```

### AnalyseNode. java

```

1. package compiler;
2.
3. import java.util.ArrayList;
4.
5. /**
6.  * 分析栈节点类
7.  * String type;//节点类型
8.  * String name;//节点名
9.  * Object value;//节点值

```

```

10. */
11. public class AnalyseNode {
12.     public final static String NONTERMINAL="非终结符";
13.     public final static String TERMINAL="终结符";
14.     public final static String ACTIONSIGN="动作符";
15.     public final static String END="结束符";
16.     static ArrayList<String>nonterminal=new ArrayList<String>();//非终结符集
    合
17.     static ArrayList<String>actionSign=new ArrayList<String>();//动作符集合
18.     static{
19.         //N:S,B,A,C,,X,R,Z,Z',U,U',E,E',H,H',G,M,D,L,L',T,T',F,O,P,Q
20.         nonterminal.add("S");
21.         nonterminal.add("A");
22.         nonterminal.add("B");
23.         nonterminal.add("C");
24.         nonterminal.add("D");
25.         nonterminal.add("E");
26.         nonterminal.add("F");
27.         nonterminal.add("G");
28.         nonterminal.add("H");
29.         nonterminal.add("L");
30.         nonterminal.add("M");
31.         nonterminal.add("O");
32.         nonterminal.add("P");
33.         nonterminal.add("Q");
34.         nonterminal.add("X");
35.         nonterminal.add("Y");
36.         nonterminal.add("Z");
37.         nonterminal.add("R");
38.         nonterminal.add("U");
39.         nonterminal.add("Z'");
40.         nonterminal.add("U'");
41.         nonterminal.add("E'");
42.         nonterminal.add("H'");
43.         nonterminal.add("L'");
44.         nonterminal.add("T");
45.         nonterminal.add("T'");
46.         actionSign.add("@ADD_SUB");
47.         actionSign.add("@ADD");
48.         actionSign.add("@SUB");
49.         actionSign.add("@DIV_MUL");
50.         actionSign.add("@DIV");
51.         actionSign.add("@MUL");
52.         actionSign.add("@SINGLE");

```



```

53.         actionSign.add("@SINGTLE_OP");
54.         actionSign.add("@ASS_R");
55.         actionSign.add("@ASS_Q");
56.         actionSign.add("@ASS_F");
57.         actionSign.add("@ASS_U");
58.         actionSign.add("@TRAN_LF");
59.         actionSign.add("@EQ");
60.         actionSign.add("@EQ_U'");
61.         actionSign.add("@COMPARE");
62.         actionSign.add("@COMPARE_OP");
63.         actionSign.add("@IF_FJ");
64.         actionSign.add("@IF_BACKPATCH_FJ");
65.         actionSign.add("@IF_RJ");
66.         actionSign.add("@IF_BACKPATCH_RJ");
67.         actionSign.add("@WHILE_FJ");
68.         actionSign.add("@WHILE_BACKPATCH_FJ");
69.         actionSign.add("@IF_RJ");
70.         actionSign.add("@FOR_FJ");
71.         actionSign.add("@FOR_RJ");
72.         actionSign.add("@FOR_BACKPATCH_FJ");
73.     }
74.
75.     String type;//节点类型
76.     String name;//节点名
77.     String value;//节点值
78.
79.     public static boolean isNonterm(AnalyseNode node){
80.         return nonterminal.contains(node.name);
81.     }
82.     public static boolean isTerm(AnalyseNode node){
83.         return Word.isKey(node.name) || Word.isOperator(node.name) || Word.isBoundarySign(node.name)
84.             || node.name.equals("id") || node.name.equals("num") || node.name.equals("ch");
85.     }
86.     public static boolean isActionSign(AnalyseNode node){
87.         return actionSign.contains(node.name);
88.     }
89.     public AnalyseNode(){
90.
91.     }
92.     public AnalyseNode(String type,String name,String value){
93.         this.type=type;
94.         this.name=name;

```

```
95.         this.value=value;
96.     }
97.
98. }
```

### FourElement.java

```
1. package compiler;
2.
3. public class FourElement {
4.     int id;//四元式序号，为编程方便
5.     String op;//操作符
6.     String arg1;//第一个操作数
7.     String arg2;//第二个操作数
8.     Object result;//结果
9.     public FourElement(){
10.
11. }
12. public FourElement(int id,String op,String arg1,String arg2,String result){
13.
14.     this.id=id;
15.     this.op=op;
16.     this.arg1=arg1;
17.     this.arg2=arg2;
18.     this.result=result;
19. }
```

### Parser.java

```
1. package compiler;
2.
3. import java.io.BufferedOutputStream;
4. import java.io.File;
5. import java.io.FileOutputStream;
6. import java.io.IOException;
7. import java.io.OutputStreamWriter;
8. import java.io.PrintWriter;
9. import java.util.ArrayList;
10. import java.util.Stack;
11. /**
12.  * 语法分析器
13.  * @author xxz
14.  *
```

```

15. */
16. public class Parser {
17.
18.     /**
19.      * @param args
20.      */
21.
22.     private LexAnalyse lexAnalyse ;//词法分析器
23.     ArrayList<Word>wordList=new ArrayList<Word>();//单词表
24.     Stack<AnalyseNode>analyseStack=new Stack<AnalyseNode>();//分析栈
25.     Stack<String>semanticStack=new Stack<String>();//语义栈
26.     ArrayList<FourElement>fourElemList=new ArrayList<FourElement>();//四元式
        列表
27.     ArrayList<Error>errorList=new ArrayList<Error>();//错误信息列表
28.     StringBuffer bf;//分析栈缓冲流
29.     int errorCount=0;//统计错误个数
30.     boolean graErrorFlag=false;//语法分析出错标志
31.     int tempCount=0;//用于生成临时变量
32.     int fourElemCount=0;//统计四元式个数
33.     AnalyseNode S,B,A,C,X,Y,R,Z,Z1,U,U1,E,E1,H,H1,G,M,D,L,L1,T,T1,F,O,P,Q;
34.     AnalyseNode ADD_SUB,DIV_MUL,ADD,SUB,DIV,MUL,ASS_F,ASS_R,ASS_Q,ASS_U,TRAN
        _LF;
35.     AnalyseNode SINGLE,SINGLE_OP,EQ,EQ_U1,COMPARE,COMPARE_OP,IF_FJ,IF_RJ,IF_
        BACKPATCH_FJ,IF_BACKPATCH_RJ;
36.     AnalyseNode WHILE_FJ,WHILE_RJ,WHILE_BACKPATCH_FJ,FOR_FJ,FOR_RJ,FOR_BACKP
        ATCH_FJ;
37.     AnalyseNode top;//当前栈顶元素
38.     Word firstWord;//待分析单词
39.     String OP=null;
40.     String ARG1,ARG2,RES;
41.     Error error;
42.     //int if_fj,if_rj,while_fj,while_rj,for_fj,for_rj;
43.     Stack<Integer>if_fj,if_rj,while_fj,while_rj,for_fj,for_rj;//if while for
        跳转地址栈
44.     Stack<String>for_op=new Stack<String>();
45.
46. public Parser(){
47.
48.     }
49. public Parser(LexAnalyse lexAnalyse){
50.     this.lexAnalyse=lexAnalyse;
51.     this.wordList=lexAnalyse.wordList;
52.     init();
53.     }

```

```

54. private String newTemp(){
55.     tempCount++;
56.     return "T"+tempCount;
57. }
58. public void init(){
59.     S=new AnalyseNode(AnalyseNode.NONTERMINAL, "S", null);
60.     A=new AnalyseNode(AnalyseNode.NONTERMINAL, "A", null);
61.     B=new AnalyseNode(AnalyseNode.NONTERMINAL, "B", null);
62.     C=new AnalyseNode(AnalyseNode.NONTERMINAL, "C", null);
63.     X=new AnalyseNode(AnalyseNode.NONTERMINAL, "X", null);
64.     Y=new AnalyseNode(AnalyseNode.NONTERMINAL, "Y", null);
65.     Z=new AnalyseNode(AnalyseNode.NONTERMINAL, "Z", null);
66.     Z1=new AnalyseNode(AnalyseNode.NONTERMINAL, "Z'", null);
67.     U=new AnalyseNode(AnalyseNode.NONTERMINAL, "U", null);
68.     U1=new AnalyseNode(AnalyseNode.NONTERMINAL, "U'", null);
69.     E=new AnalyseNode(AnalyseNode.NONTERMINAL, "E", null);
70.     E1=new AnalyseNode(AnalyseNode.NONTERMINAL, "E'", null);
71.     H=new AnalyseNode(AnalyseNode.NONTERMINAL, "H", null);
72.     H1=new AnalyseNode(AnalyseNode.NONTERMINAL, "H'", null);
73.     G=new AnalyseNode(AnalyseNode.NONTERMINAL, "G", null);
74.     F=new AnalyseNode(AnalyseNode.NONTERMINAL, "F", null);
75.     D=new AnalyseNode(AnalyseNode.NONTERMINAL, "D", null);
76.     L=new AnalyseNode(AnalyseNode.NONTERMINAL, "L", null);
77.     L1=new AnalyseNode(AnalyseNode.NONTERMINAL, "L'", null);
78.     T=new AnalyseNode(AnalyseNode.NONTERMINAL, "T", null);
79.     T1=new AnalyseNode(AnalyseNode.NONTERMINAL, "T'", null);
80.     O=new AnalyseNode(AnalyseNode.NONTERMINAL, "O", null);
81.     P=new AnalyseNode(AnalyseNode.NONTERMINAL, "P", null);
82.     Q=new AnalyseNode(AnalyseNode.NONTERMINAL, "Q", null);
83.     R=new AnalyseNode(AnalyseNode.NONTERMINAL, "R", null);
84.     ADD_SUB=new AnalyseNode(AnalyseNode.ACTIONSIGN, "@ADD_SUB", null);
85.     ADD=new AnalyseNode(AnalyseNode.ACTIONSIGN, "@ADD", null);
86.     SUB=new AnalyseNode(AnalyseNode.ACTIONSIGN, "@SUB", null);
87.     DIV_MUL=new AnalyseNode(AnalyseNode.ACTIONSIGN, "@DIV_MUL", null);
88.     DIV=new AnalyseNode(AnalyseNode.ACTIONSIGN, "@DIV", null);
89.     MUL=new AnalyseNode(AnalyseNode.ACTIONSIGN, "@MUL", null);
90.     ASS_F=new AnalyseNode(AnalyseNode.ACTIONSIGN, "@ASS_F", null);
91.     ASS_R=new AnalyseNode(AnalyseNode.ACTIONSIGN, "@ASS_R", null);
92.     ASS_Q=new AnalyseNode(AnalyseNode.ACTIONSIGN, "@ASS_Q", null);
93.     ASS_U=new AnalyseNode(AnalyseNode.ACTIONSIGN, "@ASS_U", null);
94.     TRAN_LF=new AnalyseNode(AnalyseNode.ACTIONSIGN, "@TRAN_LF", null);
95.     SINGLE=new AnalyseNode(AnalyseNode.ACTIONSIGN, "@SINGLE", null);
96.     SINGLE_OP=new AnalyseNode(AnalyseNode.ACTIONSIGN, "@SINGLE_OP", null);
97.     EQ=new AnalyseNode(AnalyseNode.ACTIONSIGN, "@EQ", null);

```

```

98.     EQ_U1=new AnalyseNode(AnalyseNode.ACTIONSIGN, "@EQ_U'", null);
99.     COMPARE=new AnalyseNode(AnalyseNode.ACTIONSIGN, "@COMPARE", null);
100.     COMPARE_OP=new AnalyseNode(AnalyseNode.ACTIONSIGN, "@COMPARE_OP", null)
        ;
101.     IF_FJ=new AnalyseNode(AnalyseNode.ACTIONSIGN, "@IF_FJ", null);
102. IF_RJ=new AnalyseNode(AnalyseNode.ACTIONSIGN, "@IF_RJ", null);
103. IF_BACKPATCH_FJ=new AnalyseNode(AnalyseNode.ACTIONSIGN, "@IF_BACKPATCH_FJ",
        null);
104. IF_BACKPATCH_RJ=new AnalyseNode(AnalyseNode.ACTIONSIGN, "@IF_BACKPATCH_RJ",
        null);
105. WHILE_FJ=new AnalyseNode(AnalyseNode.ACTIONSIGN, "@WHILE_FJ", null);
106. WHILE_RJ=new AnalyseNode(AnalyseNode.ACTIONSIGN, "@WHILE_RJ", null);
107. WHILE_BACKPATCH_FJ=new AnalyseNode(AnalyseNode.ACTIONSIGN, "@WHILE_BACKPATC
        H_FJ", null);
108. FOR_FJ=new AnalyseNode(AnalyseNode.ACTIONSIGN, "@FOR_FJ", null);
109. FOR_RJ=new AnalyseNode(AnalyseNode.ACTIONSIGN, "@FOR_RJ", null);
110. FOR_BACKPATCH_FJ=new AnalyseNode(AnalyseNode.ACTIONSIGN, "@FOR_BACKPATCH_FJ
        ", null);
111. if_fj=new Stack<Integer>();
112. if_rj=new Stack<Integer>();
113. while_fj=new Stack<Integer>();
114. while_rj=new Stack<Integer>();
115. for_fj=new Stack<Integer>();
116. for_rj=new Stack<Integer>();
117.
118. }
119. public void grammerAnalyse(){//LL1 分析方法进行语法分析
120.     if(lexAnalyse.isFail())javax.swing.JOptionPane.showMessageDialog(null,
        "词法分析未通过，不能进行语法分析");
121.     bf=new StringBuffer();
122.     int gcount=0;
123.     error=null;
124.     analyseStack.add(0,S);
125.     analyseStack.add(1,new AnalyseNode(AnalyseNode.END, "#", null));
126.     semanticStack.add("#");
127.     while(!analyseStack.empty() && !wordList.isEmpty()){
128.         //System.out.println(fourElemCount);
129.         bf.append('\n');
130.         bf.append("步骤"+gcount+"\t");
131.         if(gcount++>10000){//一万步就算了
132.             graErrorFlag=true;
133.             break;
134.         }
135.         /*

```

```

136.      * 此方法接受整数数据类型的强制参数索引。
137.      * 它指定要从堆栈中获取的元素的位置或索引。
138.      * Java.util.Stack.get()方法用于从堆栈中获取或检索特定索引处的元素。
139.      */
140.      top=analyseStack.get(0);//当前分析栈栈底元素（不是栈顶吧
141.      firstWord=wordList.get(0);//待分析单词
142.      if(firstWord.value.equals("#") && top.name.equals("#")){//正常结束
143.          bf.append("\n");
144.          analyseStack.remove(0);
145.          wordList.remove(0);
146.      }
147.      else if(top.name.equals("#")){//如果 top == # 而 first != # 则出错
148.          analyseStack.remove(0);
149.          graErrorFlag=true;
150.          break;
151.
152.      }
153.      //以 AnalyseStack 中的 top 结点为触媒
154.      //!firstWord.value.equals("#") && !top.name.equals("#")
155.      else if(AnalyseNode.isTerm(top)){//语法分析栈 top 为终结符时的处理
156.          termOP(top.name);
157.      }else if(AnalyseNode.isNonterm(top)){//语法分析栈 top 为非终结符时的处
        理
158.          nonTermOP(top.name);
159.      }else if(top.type.equals(AnalyseNode.ACTIONSIGN)){//语法分析栈 top 是
        动作符号时的处理
160.          actionSignOP();//top.name
161.      }
162.
163.      bf.append("当前语法分析栈:");
164.      for(int i=0;i<analyseStack.size();i++){
165.          bf.append(analyseStack.get(i).name);
166.          //System.out.println(analyseStack.get(i).name);
167.      }
168.      bf.append("\t").append("余留符号串: ");
169.      for(int j=0;j<wordList.size();j++){
170.          bf.append(wordList.get(j).value);
171.      }
172.      bf.append("\t").append("语义栈:");
173.      for(int k=semanticStack.size()-1;k>=0;k--){
174.          bf.append(semanticStack.get(k));
175.      }
176.  }
177. }

```

```

178.
179. private void termOP(String term){
180.     if(firstWord.type.equals(Word.INT_CONST)
181.         ||firstWord.type.equals(Word.CHAR_CONST)
182.         ||term.equals(firstWord.value)//相等也行
183.         ||(term.equals("id")&&firstWord.type.equals(Word.IDENTIFIER)))

184.     {
185.         analyseStack.remove(0);
186.         wordList.remove(0);
187.     }
188.     else{
189.         errorCount++;
190.         analyseStack.remove(0);
191.         wordList.remove(0);
192.         error=new Error(errorCount,"语法错误",firstWord.line,firstWord);
193.         errorList.add(error);
194.         graErrorFlag=true;
195.     }
196.
197. }
198. private void nonTermOP(String nonTerm){
199.     if(nonTerm.equals("Z"))nonTerm="1";
200.     if(nonTerm.equals("U"))nonTerm="2";
201.     if(nonTerm.equals("E"))nonTerm="3";
202.     if(nonTerm.equals("H"))nonTerm="4";
203.     if(nonTerm.equals("L"))nonTerm="5";
204.     if(nonTerm.equals("T"))nonTerm="6";
205.     switch(nonTerm.charAt(0)){//栈顶为非终结符处理
206.         //N:S,B,A,C,,X,R,Z,Z',U,U',E,E',H,H',G,M,D,L,L',T,T',F,O,P,Q
207.         case 'S':
208.             if(firstWord.value.equals("void")){
209.                 analyseStack.remove(0);
210.                 analyseStack.add(0,new AnalyseNode(AnalyseNode.TERMINAL, "void", null));
211.                 analyseStack.add(1,new AnalyseNode(AnalyseNode.TERMINAL, "main", null));
212.                 analyseStack.add(2,new AnalyseNode(AnalyseNode.TERMINAL, "(", null));
213.                 analyseStack.add(3,new AnalyseNode(AnalyseNode.TERMINAL, ")", null));
214.                 analyseStack.add(4,new AnalyseNode(AnalyseNode.TERMINAL, "{", null));
215.                 analyseStack.add(5,A);

```

```
216.         analyseStack.add(6,new AnalyseNode(AnalyseNode.TERMINAL, "}", null)
217.     );
218.     }else{//第一步就出错了
219.         errorCount++;
220.         analyseStack.remove(0);
221.         wordList.remove(0);
222.         error=new Error(errorCount,"主函数没有返回值
223.             ",firstWord.line,firstWord);
224.         errorList.add(error);
225.         graErrorFlag=true;
226.     }
227.     break;
228.     case 'A':
229.         if(firstWord.value.equals("int")||firstWord.value.equals("char")//A
230.             ->CA
231.             ||firstWord.value.equals("bool")){
232.             analyseStack.remove(0);
233.             analyseStack.add(0,C);
234.             analyseStack.add(1,A);
235.         }else if(firstWord.value.equals("printf")){
236.             analyseStack.remove(0);
237.             analyseStack.add(0,C);
238.             analyseStack.add(1,A);
239.         }
240.         else if(firstWord.value.equals("scanf")){
241.             analyseStack.remove(0);
242.             analyseStack.add(0,C);
243.             analyseStack.add(1,A);
244.         }
245.         else if(firstWord.value.equals("if")){
246.             analyseStack.remove(0);
247.             analyseStack.add(0,C);
248.             analyseStack.add(1,A);
249.         }
250.         else if(firstWord.value.equals("while")){
251.             analyseStack.remove(0);
252.             analyseStack.add(0,C);
253.             analyseStack.add(1,A);
254.         }
255.         else if(firstWord.value.equals("for")){
256.             analyseStack.remove(0);
257.             analyseStack.add(0,C);
258.             analyseStack.add(1,A);
```



```

257.         }
258.         else if(firstWord.type.equals(Word.IDENTIFIER)){
259.             analyseStack.remove(0);
260.             analyseStack.add(0,C);
261.             analyseStack.add(1,A);
262.         }else{//A->$
263.             analyseStack.remove(0);
264.         }
265.         break;
266.
267.     case 'B':
268.     if(firstWord.value.equals("printf")){
269.         analyseStack.remove(0);
270.         analyseStack.add(0,new AnalyseNode(AnalyseNode.TERMINAL, "print
271. f", null));
272.         analyseStack.add(1,new AnalyseNode(AnalyseNode.TERMINAL, "(", n
273. ull));
274.         analyseStack.add(2,P);
275.         analyseStack.add(3,new AnalyseNode(AnalyseNode.TERMINAL, ")", n
276. ull));
277.         analyseStack.add(4,A);
278.         analyseStack.add(5,new AnalyseNode(AnalyseNode.TERMINAL, ";", n
279. ull));
280.     }
281.     else if(firstWord.value.equals("scanf")){
282.         analyseStack.remove(0);
283.         analyseStack.add(0,new AnalyseNode(AnalyseNode.TERMINAL, "scanf
284. ", null));
285.         analyseStack.add(1,new AnalyseNode(AnalyseNode.TERMINAL, "(", n
286. ull));
287.         analyseStack.add(2,new AnalyseNode(AnalyseNode.TERMINAL, "id",
288. null));
289.         analyseStack.add(3,new AnalyseNode(AnalyseNode.TERMINAL, ")", n
290. ull));
291.         analyseStack.add(4,A);
292.         analyseStack.add(5,new AnalyseNode(AnalyseNode.TERMINAL, ";", n
293. ull));
294.     }
295.     else if(firstWord.value.equals("if")){
296.         analyseStack.remove(0);
297.         analyseStack.add(0,new AnalyseNode(AnalyseNode.TERMINAL, "if",
298. null));

```

```

290.         analyseStack.add(1,new AnalyseNode(AnalyseNode.TERMINAL, "(", null));
291.         analyseStack.add(2,G);
292.         analyseStack.add(3,new AnalyseNode(AnalyseNode.TERMINAL, ")", null));
293.         analyseStack.add(4,IF_FJ);
294.         analyseStack.add(5,new AnalyseNode(AnalyseNode.TERMINAL, "{", null));
295.         analyseStack.add(6,A);
296.         analyseStack.add(7,new AnalyseNode(AnalyseNode.TERMINAL, "}", null));
297.         analyseStack.add(8,IF_BACKPATCH_FJ);
298.         analyseStack.add(9,IF_RJ);
299.         analyseStack.add(10,new AnalyseNode(AnalyseNode.TERMINAL, "else", null));
300.         analyseStack.add(11,new AnalyseNode(AnalyseNode.TERMINAL, "{", null));
301.         analyseStack.add(12,A);
302.         analyseStack.add(13,new AnalyseNode(AnalyseNode.TERMINAL, "}", null));
303.         analyseStack.add(14,IF_BACKPATCH_RJ);
304.     }
305.     else if(firstWord.value.equals("while")){
306.         analyseStack.remove(0);
307.         analyseStack.add(0,new AnalyseNode(AnalyseNode.TERMINAL, "while", null));
308.         analyseStack.add(1,new AnalyseNode(AnalyseNode.TERMINAL, "(", null));
309.         analyseStack.add(2,G);
310.         analyseStack.add(3,new AnalyseNode(AnalyseNode.TERMINAL, ")", null));
311.         analyseStack.add(4,WHILE_FJ);
312.         analyseStack.add(5,new AnalyseNode(AnalyseNode.TERMINAL, "{", null));
313.         analyseStack.add(6,A);
314.         analyseStack.add(7,new AnalyseNode(AnalyseNode.TERMINAL, "}", null));
315.         analyseStack.add(8,WHILE_RJ);
316.         analyseStack.add(9,WHILE_BACKPATCH_FJ);
317.     }
318.     else if(firstWord.value.equals("for")){
319.         analyseStack.remove(0);
320.         analyseStack.add(0,new AnalyseNode(AnalyseNode.TERMINAL, "for", null));

```

```

321.         analyseStack.add(1,new AnalyseNode(AnalyseNode.TERMINAL, "(", null));
322.         analyseStack.add(2,Y);
323.         analyseStack.add(3,Z);
324.         analyseStack.add(4,new AnalyseNode(AnalyseNode.TERMINAL, ";", null));
325.         analyseStack.add(5,G);
326.         analyseStack.add(6,FOR_FJ);
327.         analyseStack.add(7,new AnalyseNode(AnalyseNode.TERMINAL, ";", null));
328.         analyseStack.add(8,Q);
329.         analyseStack.add(9,new AnalyseNode(AnalyseNode.TERMINAL, ")", null));
330.         analyseStack.add(10,new AnalyseNode(AnalyseNode.TERMINAL, "{", null));
331.         analyseStack.add(11,A);
332.         analyseStack.add(12,SINGLE);
333.         analyseStack.add(13,new AnalyseNode(AnalyseNode.TERMINAL, "}", null));
334.         analyseStack.add(14,FOR_RJ);
335.         analyseStack.add(15,FOR_BACKPATCH_FJ);
336.     }
337.     else{
338.         analyseStack.remove(0);
339.     }
340.     break;
341. case 'C': //C->X|B|R
342.
343.         analyseStack.remove(0);
344.         analyseStack.add(0,X);
345.         analyseStack.add(1,B);
346.         analyseStack.add(2,R);
347.         break;
348. case 'X':
349.         if(firstWord.value.equals("int") || firstWord.value.equals("char") || firstWord.value.equals("bool")){
350.             analyseStack.remove(0);
351.             analyseStack.add(0,Y);
352.             analyseStack.add(1,Z);
353.             analyseStack.add(2,new AnalyseNode(AnalyseNode.TERMINAL, ";", null));
354.         }else{
355.             analyseStack.remove(0);
356.         }

```

```

357.         break;
358.     case 'Y':
359.         if(firstWord.value.equals("int")){
360.             analyseStack.remove(0);
361.             analyseStack.add(0,new AnalyseNode(AnalyseNode.TERMINAL, "int",
null));
362.         }
363.         else if(firstWord.value.equals("char")){
364.             analyseStack.remove(0);
365.             analyseStack.add(0,new AnalyseNode(AnalyseNode.TERMINAL, "char"
, null));
366.         }
367.         else if(firstWord.value.equals("bool")){
368.             analyseStack.remove(0);
369.             analyseStack.add(0,new AnalyseNode(AnalyseNode.TERMINAL, "bool"
, null));
370.         }
371.         else{
372.             errorCount++;
373.             analyseStack.remove(0);
374.             wordList.remove(0);
375.             error=new Error(errorCount,"非法数据类型
",firstWord.line,firstWord);
376.             errorList.add(error);
377.             graErrorFlag=true;
378.         }
379.         break;
380.     case 'Z':
381.         if(firstWord.type.equals(Word.IDENTIFIER)){
382.             analyseStack.remove(0);
383.             analyseStack.add(0,U);
384.             analyseStack.add(1,Z1);//Z1 即为一个 analyseNode(z')
385.         }
386.         else{
387.             errorCount++;
388.             analyseStack.remove(0);
389.             wordList.remove(0);
390.             error=new Error(errorCount,"非法标识符
",firstWord.line,firstWord);
391.             errorList.add(error);
392.             graErrorFlag=true;
393.         }
394.         break;
395.     case '1'://z'

```

```

396.         if(firstWord.value.equals(",")){
397.             analyseStack.remove(0);
398.             analyseStack.add(0,new AnalyseNode(AnalyseNode.TERMINAL, ",", n
null));
399.             analyseStack.add(1,Z);
400.         }
401.         else{
402.             analyseStack.remove(0);
403.         }
404.         break;
405.     case 'U':
406.         if(firstWord.type.equals(Word.IDENTIFIER)){
407.             analyseStack.remove(0);
408.             analyseStack.add(0,ASS_U); //actionSignOP
409.             analyseStack.add(1,new AnalyseNode(AnalyseNode.TERMINAL, "id",
null)); //助于下一步删除 wordList 中的顶值 (已经拷贝到 semanticStack 中)
410.             analyseStack.add(2,U1);
411.         }
412.         else{
413.             errorCount++;
414.             analyseStack.remove(0);
415.             wordList.remove(0);
416.             error=new Error(errorCount,"非法标识符
",firstWord.line,firstWord);
417.             errorList.add(error);
418.             graErrorFlag=true;
419.         }
420.         break;
421.     case '2': //U'
422.         if(firstWord.value.equals("=")){
423.             analyseStack.remove(0);
424.             analyseStack.add(0,new AnalyseNode(AnalyseNode.TERMINAL, "=", n
null));
425.             analyseStack.add(1,L);
426.             analyseStack.add(2,EQ_U1);
427.         }
428.         else{//有 = 但是没赋值 就走这条路
429.             analyseStack.remove(0);
430.         }
431.         break;
432.     case 'R':
433.         if(firstWord.type.equals(Word.IDENTIFIER)){
434.             analyseStack.remove(0);

```

```

435.         analyseStack.add(0,new AnalyseNode(AnalyseNode.ACTIONSIGN, "@AS
S_R", null));
436.         analyseStack.add(1,new AnalyseNode(AnalyseNode.TERMINAL, "id",
null));
437.         analyseStack.add(2,new AnalyseNode(AnalyseNode.TERMINAL, "=", n
ull));
438.         analyseStack.add(3,L);
439.         analyseStack.add(4,EQ);
440.         analyseStack.add(5,new AnalyseNode(AnalyseNode.TERMINAL, ";", n
ull));
441.     }
442.     else{
443.         analyseStack.remove(0);
444.     }
445.     break;
446.     case 'P':
447.         if(firstWord.type.equals(Word.IDENTIFIER)){
448.             analyseStack.remove(0);
449.             analyseStack.add(0,new AnalyseNode(AnalyseNode.TERMINAL, "id",
null));
450.         }else if(firstWord.type.equals(Word.INT_CONST)){
451.             analyseStack.remove(0);
452.             analyseStack.add(0,new AnalyseNode(AnalyseNode.TERMINAL, "num",
null));
453.         }else if(firstWord.type.equals(Word.CHAR_CONST)){
454.             analyseStack.remove(0);
455.             analyseStack.add(0,new AnalyseNode(AnalyseNode.TERMINAL, "ch",
null));
456.         }
457.         else{
458.             errorCount++;
459.             analyseStack.remove(0);
460.             wordList.remove(0);
461.             error=new Error(errorCount,"不能输出的数据类型
",firstWord.line,firstWord);
462.             errorList.add(error);
463.             graErrorFlag=true;
464.         }
465.         break;
466.     case 'E':
467.         if(firstWord.type.equals(Word.IDENTIFIER)){
468.             analyseStack.remove(0);
469.             analyseStack.add(0,H);
470.             analyseStack.add(1,E1);

```

```

471.         }else if(firstWord.type.equals(Word.INT_CONST)){
472.             analyseStack.remove(0);
473.             analyseStack.add(0,H);
474.             analyseStack.add(1,E1);
475.         }else if(firstWord.value.equals("(")){
476.             analyseStack.remove(0);
477.             analyseStack.add(0,H);
478.             analyseStack.add(1,E1);
479.         }
480.         else{
481.             errorCount++;
482.             analyseStack.remove(0);
483.             wordList.remove(0);
484.             error=new Error(errorCount,"不能进行算术运算的数据类型",firstWord.line,firstWord);
485.             errorList.add(error);
486.             graErrorFlag=true;
487.         }
488.         break;
489.     case '3': //E'
490.         if(firstWord.value.equals("&&")){
491.             analyseStack.remove(0);
492.             analyseStack.add(0,new AnalyseNode(AnalyseNode.TERMINAL, "&&", null));
493.             analyseStack.add(1,E);
494.         }else {
495.             analyseStack.remove(0);
496.         }
497.         break;
498.     case 'H':
499.         if(firstWord.type.equals(Word.IDENTIFIER)){
500.             analyseStack.remove(0);
501.             analyseStack.add(0,G);
502.             analyseStack.add(1,H1);
503.         }else if(firstWord.type.equals(Word.INT_CONST)){
504.             analyseStack.remove(0);
505.             analyseStack.add(0,G);
506.             analyseStack.add(1,H1);
507.         }else if(firstWord.value.equals("(")){
508.             analyseStack.remove(0);
509.             analyseStack.add(0,G);
510.             analyseStack.add(1,H1);
511.         }
512.         else{

```

```

513.         errorCount++;
514.         analyseStack.remove(0);
515.         wordList.remove(0);
516.         error=new Error(errorCount,"不能进行算术运算的数据类型
",firstWord.line,firstWord);
517.         errorList.add(error);
518.         graErrorFlag=true;
519.     }
520.     break;
521.     case '4'://H'
522.         if(firstWord.value.equals("||")){
523.             analyseStack.remove(0);
524.             analyseStack.add(0,new AnalyseNode(AnalyseNode.TERMINAL, "||",
null));
525.             analyseStack.add(1,E);
526.         }else {
527.             analyseStack.remove(0);
528.         }
529.         break;
530.     case 'D':
531.         if(firstWord.value.equals("==")){
532.             analyseStack.remove(0);
533.             analyseStack.add(0,COMPARE_OP);
534.             analyseStack.add(1,new AnalyseNode(AnalyseNode.TERMINAL, "==",
null));
535.         }else if(firstWord.value.equals("!=")){
536.             analyseStack.remove(0);
537.             analyseStack.add(0,COMPARE_OP);
538.             analyseStack.add(1,new AnalyseNode(AnalyseNode.TERMINAL, "!=",
null));
539.
540.         }else if(firstWord.value.equals(">")){
541.             analyseStack.remove(0);
542.             analyseStack.add(0,COMPARE_OP);
543.             analyseStack.add(1,new AnalyseNode(AnalyseNode.TERMINAL, ">", n
ull));
544.         }else if(firstWord.value.equals("<")){
545.             analyseStack.remove(0);
546.             analyseStack.add(0,COMPARE_OP);
547.             analyseStack.add(1,new AnalyseNode(AnalyseNode.TERMINAL, "<", n
ull));
548.         }
549.         else{
550.             errorCount++;

```



```

551.         analyseStack.remove(0);
552.         wordList.remove(0);
553.         error=new Error(errorCount,"非法运算符
",firstWord.line,firstWord);
554.         errorList.add(error);
555.         graErrorFlag=true;
556.     }
557.     break;
558.     case 'G':
559.         if(firstWord.type.equals(Word.IDENTIFIER)){
560.             analyseStack.remove(0);
561.             analyseStack.add(0,F);
562.             analyseStack.add(1,D);
563.             analyseStack.add(2,F);
564.             analyseStack.add(3,COMPARE);
565.         }else if(firstWord.type.equals(Word.INT_CONST)){
566.             analyseStack.remove(0);
567.             analyseStack.add(0,F);
568.             analyseStack.add(1,D);
569.             analyseStack.add(2,F);
570.             analyseStack.add(3,COMPARE);
571.         }
572.         else if(firstWord.value.equals("(")){
573.             analyseStack.remove(0);
574.             analyseStack.add(0,new AnalyseNode(AnalyseNode.TERMINAL, "(", n
ull));
575.             analyseStack.add(1,E);
576.             analyseStack.add(2,new AnalyseNode(AnalyseNode.TERMINAL, ")", n
ull));
577.         }
578.         else if(firstWord.value.equals("!")){
579.             analyseStack.remove(0);
580.             analyseStack.add(0,new AnalyseNode(AnalyseNode.TERMINAL, "!", n
ull));
581.             analyseStack.add(1,E);
582.         }
583.         else{
584.             errorCount++;
585.             analyseStack.remove(0);
586.             wordList.remove(0);
587.             error=new Error(errorCount,"不能进行算术运算的数据类型或括号不匹配
",firstWord.line,firstWord);
588.             errorList.add(error);
589.             graErrorFlag=true;

```

```

590.     }
591.     break;
592.     case 'L':
593.         if(firstWord.type.equals(Word.IDENTIFIER)){
594.             analyseStack.remove(0);
595.             analyseStack.add(0,T);
596.             analyseStack.add(1,L1);
597.             analyseStack.add(2,ADD_SUB);
598.         }else if(firstWord.type.equals(Word.INT_CONST)){
599.             analyseStack.remove(0);
600.             analyseStack.add(0,T);
601.             analyseStack.add(1,L1);
602.             analyseStack.add(2,ADD_SUB);
603.         }
604.         else if(firstWord.value.equals("(")){
605.             analyseStack.remove(0);
606.             analyseStack.add(0,T);
607.             analyseStack.add(1,L1);
608.             analyseStack.add(2,ADD_SUB);
609.         }
610.         else{
611.             errorCount++;
612.             analyseStack.remove(0);
613.             wordList.remove(0);
614.             error=new Error(errorCount,"不能进行算术运算的数据类型或括号不匹配",firstWord.line,firstWord);
615.             errorList.add(error);
616.             graErrorFlag=true;
617.         }
618.         break;
619.     case '5'://1'
620.         if(firstWord.value.equals("+")){
621.             analyseStack.remove(0);
622.             analyseStack.add(0,new AnalyseNode(AnalyseNode.TERMINAL, "+", null));
623.             analyseStack.add(1,L);
624.             analyseStack.add(2,ADD);
625.         }
626.         else if(firstWord.value.equals("-")){
627.             analyseStack.remove(0);
628.             analyseStack.add(0,new AnalyseNode(AnalyseNode.TERMINAL, "-", null));
629.             analyseStack.add(1,L);
630.             analyseStack.add(2,SUB);

```

```

631.         }else {
632.             analyseStack.remove(0);
633.         }
634.         break;
635.
636.     case 'T':
637.         if(firstWord.type.equals(Word.IDENTIFIER)){
638.             analyseStack.remove(0);
639.             analyseStack.add(0,F);
640.             analyseStack.add(1,T1);
641.             analyseStack.add(2,DIV_MUL);
642.         }else if(firstWord.type.equals(Word.INT_CONST)){
643.             analyseStack.remove(0);
644.             analyseStack.add(0,F);
645.             analyseStack.add(1,T1);
646.             analyseStack.add(2,DIV_MUL);
647.         }
648.         else if(firstWord.value.equals("(")){
649.             analyseStack.remove(0);
650.             analyseStack.add(0,F);
651.             analyseStack.add(1,T1);
652.             analyseStack.add(2,DIV_MUL);
653.         }
654.         else{
655.             errorCount++;
656.             analyseStack.remove(0);
657.             wordList.remove(0);
658.             error=new Error(errorCount,"不能进行算术运算的数据类型",firstWord.line,firstWord);
659.             errorList.add(error);
660.             graErrorFlag=true;
661.         }
662.         break;
663.     case '6'://T'
664.         if(firstWord.value.equals("*")){
665.             analyseStack.remove(0);
666.             analyseStack.add(0,new AnalyzeNode(AnalyzeNode.TERMINAL, "*", null));
667.             analyseStack.add(1,T);
668.             analyseStack.add(2,MUL);
669.         }
670.         else if(firstWord.value.equals("/")){
671.             analyseStack.remove(0);

```

```
672.         analyseStack.add(0,new AnalyseNode(AnalyseNode.TERMINAL, "/", null));
673.         analyseStack.add(1,T);
674.         analyseStack.add(2,DIV);
675.     }else {
676.         analyseStack.remove(0);
677.     }
678.     break;
679. case 'F':
680.     if(firstWord.type.equals(Word.IDENTIFIER)){
681.         analyseStack.remove(0);
682.         analyseStack.add(0,ASS_F);
683.         analyseStack.add(1,new AnalyseNode(AnalyseNode.TERMINAL, "id", null));
684.     }else if(firstWord.type.equals(Word.INT_CONST)){
685.         analyseStack.remove(0);
686.         analyseStack.add(0,ASS_F);
687.         analyseStack.add(1,new AnalyseNode(AnalyseNode.TERMINAL, "num", null));
688.     }else if(firstWord.value.equals("(")){
689.         analyseStack.add(0,new AnalyseNode(AnalyseNode.TERMINAL, "(", null));
690.         analyseStack.add(1,L);
691.         analyseStack.add(2,new AnalyseNode(AnalyseNode.TERMINAL, ")", null));
692.         analyseStack.add(3,TRAN_LF);
693.     }
694.     else{
695.         //errorCount++;
696.         analyseStack.remove(0);
697.         // wordList.remove(0);
698.         //error=new Error(errorCount,"不能进行算术运算的数据类型",firstWord.line,firstWord);
699.         //errorList.add(error);
700.         //graErrorFlag=true;
701.     }
702.     break;
703. case 'O':
704.     if(firstWord.value.equals("++")){
705.         analyseStack.remove(0);
706.         analyseStack.add(0,new AnalyseNode(AnalyseNode.ACTIONSIGN, "@SINGLE_OP", null));
707.         analyseStack.add(1,new AnalyseNode(AnalyseNode.TERMINAL, "++", null));
```

```

708.         }else if(firstWord.value.equals("--")){
709.             analyseStack.remove(0);
710.             analyseStack.add(0,new AnalyseNode(AnalyseNode.ACTIONSIGN, "@SI
    NGLE_OP", null));
711.             analyseStack.add(1,new AnalyseNode(AnalyseNode.TERMINAL, "--",
    null));
712.
713.         }else {
714.             analyseStack.remove(0);
715.         }
716.         break;
717.     case 'Q'://Q
718.         if(firstWord.type.equals(Word.IDENTIFIER)){
719.             analyseStack.remove(0);
720.             analyseStack.add(0,new AnalyseNode(AnalyseNode.ACTIONSIGN, "@AS
    S_Q", null));
721.             analyseStack.add(1,new AnalyseNode(AnalyseNode.TERMINAL, "id",
    null));
722.             analyseStack.add(2,new AnalyseNode(AnalyseNode.TERMINAL, "0", n
    ull));
723.         }else {
724.             analyseStack.remove(0);
725.         }
726.         break;
727.
728.     }
729. }
730.
731. private void actionSignOP(){
732.     if(top.name.equals("@ADD_SUB")){
733.         if(OP!=null&&(OP.equals("+")||OP.equals("-"))){
734.             ARG2=semanticStack.pop();
735.             ARG1=semanticStack.pop();
736.             RES=newTemp();
737.             //fourElemCount++;
738.             FourElement fourElem=new FourElement(++fourElemCount,OP,ARG1,AR
    G2,RES);
739.             fourElemList.add(fourElem);
740.             L.value=RES;
741.             semanticStack.push(L.value);
742.             OP=null;
743.         }
744.         analyseStack.remove(0);
745.

```

```

746.     }else if(top.name.equals("@ADD")){
747.         OP="+";
748.         analyseStack.remove(0);
749.     }else if(top.name.equals("@SUB")){
750.         OP="-";
751.         analyseStack.remove(0);
752.     }else if(top.name.equals("@DIV_MUL")){//确保不是*= /=
753.         if(OP!=null&&(OP.equals("*")||OP.equals("/"))){
754.             ARG2=semanticStack.pop();
755.             ARG1=semanticStack.pop();
756.             RES=newTemp();
757.             //fourElemCount++;
758.             FourElement fourElem=new FourElement(++fourElemCount,OP,ARG1,ARG2,RES);
759.             fourElemList.add(fourElem);
760.             T.value=RES;
761.             semanticStack.push(T.value);
762.             OP=null;
763.         }
764.         analyseStack.remove(0);
765.     }
766.     else if(top.name.equals("@DIV")){
767.         OP="/";
768.         analyseStack.remove(0);
769.     }
770.     else if(top.name.equals("@MUL")){
771.         OP="*";
772.         analyseStack.remove(0);
773.     }else if(top.name.equals("@TRAN_LF")){
774.         F.value=L.value;
775.         //semanticStack.push(F.value);
776.         analyseStack.remove(0);
777.     }else if(top.name.equals("@ASS_F")){
778.         F.value=firstWord.value;
779.         semanticStack.push(F.value);
780.         analyseStack.remove(0);
781.     }else if(top.name.equals("@ASS_R")){
782.         R.value=firstWord.value;
783.         semanticStack.push(R.value);
784.         analyseStack.remove(0);
785.     }else if(top.name.equals("@ASS_Q")){
786.         Q.value=firstWord.value;
787.         semanticStack.push(Q.value);
788.         analyseStack.remove(0);

```

```

789.     }
790.     else if(top.name.equals("@ASS_U")){
791.         U.value=firstWord.value;
792.         semanticStack.push(U.value);
793.         analyseStack.remove(0);
794.     }else if(top.name.equals("@SINGLE")){
795.         if(for_op.peek()!=null){
796.             ARG1=semanticStack.pop();
797.             RES=ARG1;
798.             //fourElemCount++;
799.             FourElement fourElem=new FourElement(++fourElemCount,for_op.pop
            (),ARG1,"/",RES);
800.             fourElemList.add(fourElem);
801.         }
802.         analyseStack.remove(0);
803.     }else if(top.name.equals("@SINGLE_OP")){
804.         for_op.push(firstWord.value);
805.         analyseStack.remove(0);
806.     }else if(top.name.equals("@EQ")){
807.         OP="=";
808.         ARG1=semanticStack.pop();
809.         RES=semanticStack.pop();
810.         //fourElemCount++;
811.         FourElement fourElem=new FourElement(++fourElemCount,OP,ARG1,"/",RE
            S);
812.         fourElemList.add(fourElem);
813.         OP=null;
814.         analyseStack.remove(0);
815.     }
816.     else if(top.name.equals("@EQ_U')){
817.         OP="=";
818.         ARG1=semanticStack.pop();
819.         RES=semanticStack.pop();
820.         //fourElemCount++;
821.         FourElement fourElem=new FourElement(++fourElemCount,OP,ARG1,"/",RE
            S);// /代表 = 的另外一个操作数缺失 即为单目运算
822.         fourElemList.add(fourElem);
823.         OP=null;
824.         analyseStack.remove(0);
825.     }else if(top.name.equals("@COMPARE")){
826.         ARG2=semanticStack.pop();
827.         OP=semanticStack.pop();
828.         ARG1=semanticStack.pop();
829.         RES=newTemp();

```

```

830.         //fourElemCount++;
831.         FourElement fourElem=new FourElement(++fourElemCount,OP,ARG1,ARG2,RES);
832.         fourElemList.add(fourElem);
833.         G.value=RES;
834.         semanticStack.push(G.value);
835.         OP=null;
836.         analyseStack.remove(0);
837.     }else if(top.name.equals("@COMPARE_OP")){
838.         D.value=firstWord.value;
839.         semanticStack.push(D.value);
840.         analyseStack.remove(0);
841.     }else if(top.name.equals("@IF_FJ")){
842.         OP="FJ";
843.         ARG1=semanticStack.pop();
844.         FourElement fourElem=new FourElement(++fourElemCount,OP,ARG1,"/",RES);
845.         if_fj.push(fourElemCount);
846.         fourElemList.add(fourElem);
847.         OP=null;
848.         analyseStack.remove(0);
849.     }else if(top.name.equals("@IF_BACKPATCH_FJ")){
850.         backpatch(if_fj.pop(), fourElemCount+2);
851.         analyseStack.remove(0);
852.     }else if(top.name.equals("@IF_RJ")){
853.         OP="RJ";
854.         FourElement fourElem=new FourElement(++fourElemCount,OP,"/", "/", "/");
855.         if_rj.push(fourElemCount);
856.         fourElemList.add(fourElem);
857.         OP=null;
858.         analyseStack.remove(0);
859.     }else if(top.name.equals("@IF_BACKPATCH_RJ")){
860.         backpatch(if_rj.pop(), fourElemCount+1);
861.         analyseStack.remove(0);
862.     }else if(top.name.equals("@WHILE_FJ")){
863.         OP="FJ";
864.         ARG1=semanticStack.pop();
865.         FourElement fourElem=new FourElement(++fourElemCount,OP,ARG1,"/", "/");
866.         while_fj.push(fourElemCount);
867.         fourElemList.add(fourElem);
868.         OP=null;
869.         analyseStack.remove(0);

```



```

870.     }else if(top.name.equals("@WHILE_RJ")){
871.         OP="RJ";
872.         RES=(while_fj.peek()-1)+"";
873.         FourElement fourElem=new FourElement(++fourElemCount,OP,"/", "/", RES
            );
874.         for_rj.push(fourElemCount);
875.         fourElemList.add(fourElem);
876.         OP=null;
877.         analyseStack.remove(0);
878.     }else if(top.name.equals("@WHILE_BACKPATCH_FJ")){
879.         backpatch(while_fj.pop(), fourElemCount+1);
880.         analyseStack.remove(0);
881.     }else if(top.name.equals("@FOR_FJ")){
882.         OP="FJ";
883.         ARG1=semanticStack.pop();
884.         FourElement fourElem=new FourElement(++fourElemCount,OP,ARG1,"/", "/"
            );
885.         for_fj.push(fourElemCount);
886.         fourElemList.add(fourElem);
887.         OP=null;
888.         analyseStack.remove(0);//除去 Ti
889.     }else if(top.name.equals("@FOR_RJ")){
890.         OP="RJ";
891.         RES=(for_fj.peek()-1)+"";
892.         FourElement fourElem=new FourElement(++fourElemCount,OP,"/", "/", RES
            );
893.         for_rj.push(fourElemCount);
894.         fourElemList.add(fourElem);
895.         OP=null;
896.         analyseStack.remove(0);
897.     }else if(top.name.equals("@FOR_BACKPATCH_FJ")){
898.         backpatch(for_fj.pop(), fourElemCount+1);
899.         analyseStack.remove(0);
900.     }
901.
902.
903. }
904. private void backpatch(int i,int res){
905.     FourElement temp=fourElemList.get(i-1);
906.     temp.result=res+"";
907.     fourElemList.set(i-1, temp);
908. }
909. public String outputLL1() throws IOException{
910.     //grammerAnalyse();

```

```

911.     File file=new File("./output/");
912.     if(!file.exists()){
913.         file.mkdirs();
914.         file.createNewFile();//如果这个文件不存在就创建它
915.     }
916.     String path=file.getAbsolutePath();
917.     FileOutputStream fos=new FileOutputStream(path+"/LL1.txt");
918.     BufferedOutputStream bos=new BufferedOutputStream(fos);
919.     OutputStreamWriter osw1=new OutputStreamWriter(bos,"utf-8");
920.     PrintWriter pw1=new PrintWriter(osw1);
921.     pw1.println(bf.toString());
922.     bf.delete(0, bf.length());
923.     if(greErrorFlag){
924.         Error error;
925.         pw1.println("错误信息如下: ");
926.
927.         pw1.println("错误序号\t 错误信息\t 错误所在行 \t 错误单词");
928.         for(int i=0;i<errorList.size();i++){
929.             error=errorList.get(i);
930.             pw1.println(error.id+"\t"+error.info+"\t\t"+error.line+"\t"+error.w
ord.value);
931.         }
932.     }else {
933.         pw1.println("语法分析通过: ");
934.     }
935.     pw1.close();
936.     return path+"/LL1.txt";
937. }
938. public String outputFourElem() throws IOException{
939.
940.     File file=new File("./output/");
941.     if(!file.exists()){
942.         file.mkdirs();
943.         file.createNewFile();//如果这个文件不存在就创建它
944.     }
945.     String path=file.getAbsolutePath();
946.     FileOutputStream fos=new FileOutputStream(path+"/FourElement.txt");
947.     BufferedOutputStream bos=new BufferedOutputStream(fos);
948.     OutputStreamWriter osw1=new OutputStreamWriter(bos,"utf-8");
949.     PrintWriter pw1=new PrintWriter(osw1);
950.     pw1.println("生成的四元式如下");
951.     pw1.println("序号 (OP, ARG1, ARG2, RESULT) ");
952.     FourElement temp;
953.     for(int i=0;i<fourElemList.size();i++){

```

```

954.         temp=fourElemList.get(i);
955.         pw1.println(temp.id+"("+temp.op+", "+temp.arg1+", "+temp.arg2+", "+tem
           p.result+")");
956.     }
957.     pw1.close();
958.
959.     return path+"/FourElement.txt";
960. }
961.     public static void main(String[] args) {
962.         // TODO Auto-generated method stub
963.
964.     }
965.
966. }

```

## huibian. java

```

1. package compiler;
2.
3. import java.awt.BorderLayout;
4. import java.awt.Color;
5. import java.awt.Container;
6. import java.awt.Dimension;
7. import java.awt.Toolkit;
8. import java.io.BufferedInputStream;
9. import java.io.BufferedReader;
10. import java.io.FileInputStream;
11. import java.io.IOException;
12. import java.io.InputStreamReader;
13.
14. import javax.swing.JFrame;
15. import javax.swing.JPanel;
16. import javax.swing.JTextArea;
17. import java.awt.*;
18.
19. public class huibian extends JFrame {
20.
21.     //private static final long serialVersionUID = 8766059377195109228L;
22.     private static String title;
23.     private static String fileName;
24.     private static JTextArea text;
25.
26.     public huibian() {
27.         init();

```

```

28.
29.     }
30. public huibian(String title,String fileName){//huibian inf = new huibian("目
    标代码生成", fourElementPath);
31.     this.title=title;
32.     this.fileName=fileName;
33.     init();
34.     this.setTitle(title);
35.     try {
36.         String str[] =readFile(fileName).split("\n");
37.         for(int i = 2; i < str.length; i++)
38.         {
39.             String temp[] = str[i].split(",");
40.             if(temp[0].charAt(temp[0].length() -1) == '='){//=
41.
42.
43.                 String temp1 = temp[0].substring(0,temp[0].length() - 2) + "
MOV " + temp[3].substring(0,temp[3].length() - 1) + "," + temp[1] + "\n";
44.                 //System.out.println(temp1);
45.                 text.append(temp1);
46.             }
47.             //++
48.             else if(temp[0].charAt(temp[0].length() -1) == '+' && temp[0].ch
                arAt(temp[0].length() -2) == '+'){
49.                 String temp1 = temp[0].substring(0,temp[0].length() - 3) + "
INC " + temp[1] + "\n";
50.                 //System.out.println(temp1);
51.                 text.append(temp1);
52.             }
53.             //+
54.             else if(temp[0].charAt(temp[0].length() -1) == '+'){
55.                 String temp1 = temp[0].substring(0,temp[0].length() - 2) + "
ADD " + temp[3].substring(0,temp[3].length() - 1) + "," + temp[1] + "\n";
56.                 //System.out.println(temp1);
57.                 text.append(temp1);
58.             }
59.             //-
60.             else if(temp[0].charAt(temp[0].length() -1) == '-'){
61.                 String temp1 = temp[0].substring(0,temp[0].length() - 2) + "
SUB " + temp[3].substring(0,temp[3].length() - 1) + "," + temp[1] + "\n";
62.                 //System.out.println(temp1);
63.                 text.append(temp1);
64.             }
65.             /*

```

```

66.         else if(temp[0].charAt(temp[0].length() -1) == '*'){
67.             String temp1 = temp[0].substring(0,temp[0].length() - 2) + "
        MUL " + temp[3].substring(0,temp[3].length() - 1) + "," + temp[1] + "\n";
68.             //System.out.println(temp1);
69.             text.append(temp1);
70.         }
71.         ///
72.         else if(temp[0].charAt(temp[0].length() -1) == '/'){
73.             String temp1 = temp[0].substring(0,temp[0].length() - 2) + "
        DIV " + temp[3].substring(0,temp[3].length() - 1) + "," + temp[1] + "\n";
74.             //System.out.println(temp1);
75.             text.append(temp1);
76.         }
77.         //RJ
78.         else if(temp[0].charAt(temp[0].length() -1) == 'J' && temp[0].ch
        arAt(temp[0].length() -2) == 'R'){
79.             String temp1 = temp[0].substring(0,temp[0].length() - 3) + "
        JMP " + temp[3].substring(0,temp[3].length() - 1) + "\n";
80.             //System.out.println(temp1);
81.             text.append(temp1);
82.         }
83.         //FJ
84.         else if(temp[0].charAt(temp[0].length() -1) == 'J' && temp[0].ch
        arAt(temp[0].length() -2) == 'F'){
85.             String temp1 = temp[0].substring(0,temp[0].length() - 3) + "
        JZ " + temp[3].substring(0,temp[3].length() - 1) + "\n";
86.             //System.out.println(temp1);
87.             text.append(temp1);
88.         }
89.         ///>
90.         else if(temp[0].charAt(temp[0].length() -1) == '>'){
91.             String temp1 = temp[0].substring(0,temp[0].length() - 2) + "
        JG " + temp[3].substring(0,temp[3].length() - 1) + "\n";
92.             //System.out.println(temp1);
93.             text.append(temp1);
94.         }
95.         ///<
96.         else if(temp[0].charAt(temp[0].length() -1) == '<'){
97.             String temp1 = temp[0].substring(0,temp[0].length() - 2) + "
        JL " + temp[3].substring(0,temp[3].length() - 1) + "\n";
98.             //System.out.println(temp1);
99.             text.append(temp1);
100.        }

```

```

101.         }
102.
103.     } catch (IOException e) {
104.
105.         e.printStackTrace();
106.     }
107. }
108.     private void init() {
109.         Toolkit toolkit = Toolkit.getDefaultToolkit();
110.         Dimension screen = toolkit.getScreenSize();
111.         setSize(500, 400);
112.         super.setLocation(screen.width / 2 - this.getWidth() / 2, screen.he
            ight
113.                               / 2 - this.getHeight() / 2);
114.         setContentPane(createContentPane());
115.     }
116.
117.     private Container createContentPane() {
118.         JPanel pane = new JPanel(new BorderLayout());
119.         text = new TextArea();
120.         //msg.setBackground(Color.green);
121.         text.setForeground(Color.BLUE);
122.         pane.add(BorderLayout.CENTER, text);
123.         return pane;
124.     }
125.     private String readFile(String filename)
126.     throws IOException{
127.         StringBuilder sbr = new StringBuilder();
128.         String str;
129.         FileInputStream fis = new FileInputStream(filename);
130.         BufferedInputStream bis = new BufferedInputStream(fis);
131.         InputStreamReader isr = new InputStreamReader(bis, "UTF-8");
132.         BufferedReader in=new BufferedReader(isr);
133.         while((str=in.readLine())!=null){
134.             sbr.append(str).append('\n');
135.         }
136.         in.close();
137.         //text.setText(sbr.toString());
138.         return sbr.toString();
139.     }
140.     public static String getTitl() {
141.         return title;
142.     }
143.

```

```

144.     public static void setTitl(String title) {
145.         huibian.title = title;
146.     }
147.
148.     public static String getFileName() {
149.         return fileName;
150.     }
151.
152.     public static void setFileName(String fileName) {
153.         huibian.fileName = fileName;
154.     }
155.
156.     public static TextArea getText() {
157.         return text;
158.     }
159.
160.     public static void setText(TextArea jText) {
161.         huibian.text = jText;
162.     }
163.     public static void main(String[] args) {
164.         // TODO Auto-generated method stub
165.         huibian inf=new huibian("测试","test.txt");
166. inf.setVisible(true);
167.     }
168.
169. }

```

### MainFrame.java

```

1. package gui;
2.
3. import java.awt.*;
4. import java.awt.event.ActionEvent;
5. import java.awt.event.ActionListener;
6. import java.io.BufferedInputStream;
7. import java.io.BufferedReader;
8. import java.io.File;
9. import java.io.FileInputStream;
10. import java.io.FileNotFoundException;
11. import java.io.IOException;
12. import java.io.InputStreamReader;
13.
14. import javax.swing.*;
15. import compiler.*;

```

```

16.
17. public class MainFrame extends JFrame {
18.
19.     TextArea sourceFile;//用来显示源文件的文本框
20.     String sourcePath;// 源文件路径
21.     String LL1Path;
22.     String wordListPath;
23.     String fourElementPath;
24.     LexAnalyse lexAnalyse;
25.
26.     Parser parser;
27.     public MainFrame() {
28.         this.init();
29.     }
30.
31.     public void init() {
32.
33.         Toolkit toolkit = Toolkit.getDefaultToolkit();
34.         Dimension screen = toolkit.getScreenSize();
35.         setTitle("C 语言小型编译器");
36.         setSize(750, 480);
37.         super.setResizable(false);
38.         super.setLocation(screen.width / 2 - this.getWidth() / 2, screen.height / 2 - this.getHeight() / 2);
39.         this.setContentPane(this.createContentPane());
40.     }
41.
42.     private JPanel createContentPane() {
43.         JPanel p = new JPanel(new BorderLayout());
44.         p.add(BorderLayout.NORTH, createUpPane());
45.         p.add(BorderLayout.CENTER, createCenterPane());
46.         p.add(BorderLayout.SOUTH, createBottomPane());
47.         // p.setBorder(new EmptyBorder(8,8,8,8));
48.         return p;
49.     }
50.
51.     private Component createUpPane() {
52.         JPanel p = new JPanel(new FlowLayout());
53.         final FilePanel fp = new FilePanel("选择待分析文件");
54.         JButton button = new JButton("确定");
55.         button.addActionListener(new ActionListener() {
56.
57.             @Override
58.             public void actionPerformed(ActionEvent e) {

```



```

59.         String text;
60.         try {
61.             sourcePath = fp.getFileName();
62.             text = readFile(sourcePath);
63.             sourceFile.setText(text);
64.
65.         } catch (IOException e1) {
66.             e1.printStackTrace();
67.         }
68.
69.     }
70. });
71. p.add(fp);
72. //p.add(fp1);
73. p.add(button);
74. return p;
75. }
76.
77. private Component createCenterPane() {
78.     JPanel p = new JPanel(new BorderLayout());
79.     JLabel label = new JLabel("源文件如下: ");
80.     sourceFile = new TextArea();
81.     sourceFile.setText("");
82.     p.add(BorderLayout.NORTH, label);
83.     p.add(BorderLayout.CENTER, sourceFile);
84.     return p;
85. }
86.
87. private Component creatBottomPane() {
88.     JPanel p = new JPanel(new FlowLayout());
89.     JButton bt1 = new JButton("词法分析");
90.     JButton bt2 = new JButton("语法分析");
91.     JButton bt4 = new JButton("中间代码生成");
92.     JButton bt5 = new JButton("目标代码生成");
93.     bt1.addActionListener(new ActionListener() {
94.
95.         @Override
96.         public void actionPerformed(ActionEvent e) {
97.             try {
98.                 lexAnalyse=new LexAnalyse(sourceFile.getText());
99.                 wordListPath = lexAnalyse.outputWordList();
100.            } catch (IOException e1) {
101.                // TODO Auto-generated catch block
102.                e1.printStackTrace();

```

```

103.         }
104.         InfoFrame inf = new InfoFrame("词法分析", wordListPath);
105.
106.         inf.setVisible(true);
107.     }
108. });
109. bt2.addActionListener(new ActionListener() {
110.
111.     @Override
112.     public void actionPerformed(ActionEvent e) {
113.         lexAnalyse=new LexAnalyse(sourseFile.getText());
114.         parser=new Parser(lexAnalyse);
115.         try {
116.             parser.grammerAnalyse();
117.             LL1Path= parser.outputLL1();
118.         } catch (IOException e1) {
119.             // TODO Auto-generated catch block
120.             e1.printStackTrace();
121.         }
122.         InfoFrame inf = new InfoFrame("语法分析", LL1Path);
123.         inf.setVisible(true);
124.     }
125. });
126.
127. bt4.addActionListener(new ActionListener() {
128.
129.     @Override
130.     public void actionPerformed(ActionEvent e) {
131.         try {
132.             lexAnalyse=new LexAnalyse(sourseFile.getText());
133.             parser=new Parser(lexAnalyse);
134.             parser.grammerAnalyse();
135.             fourElementPath=parser.outputFourElem();
136.         } catch (IOException e1) {
137.             // TODO Auto-generated catch block
138.             e1.printStackTrace();
139.         }
140.         InfoFrame inf = new InfoFrame("中间代码生成
141. ", fourElementPath);
142.         inf.setVisible(true);
143.     }
144. });
145. bt5.addActionListener(new ActionListener() {

```

```
146.         @Override
147.         public void actionPerformed(ActionEvent e) {
148.             try {
149.                 lexAnalyse=new LexAnalyse(sourceFile.getText());
150.                 parser=new Parser(lexAnalyse);
151.                 parser.grammarAnalyse();
152.                 fourElementPath=parser.outputFourElem();
153.             } catch (IOException e1) {
154.                 // TODO Auto-generated catch block
155.                 e1.printStackTrace();
156.             }
157.             huibian inf = new huibian("目标代码生成
", fourElementPath);
158.             inf.setVisible(true);
159.         }
160.     });
161.
162.     p.add(bt1);
163.     p.add(bt2);
164.     //p.add(bt3);
165.     p.add(bt4);
166.     p.add(bt5);
167.     return p;
168. }
169.
170. public static String readFile(String fileName) throws IOException {
171.     StringBuilder sbr = new StringBuilder();
172.     String str;
173.     FileInputStream fis = new FileInputStream(fileName);
174.     BufferedInputStream bis = new BufferedInputStream(fis);
175.     InputStreamReader isr = new InputStreamReader(bis, "UTF-8");
176.     BufferedReader in = new BufferedReader(isr);
177.     while ((str = in.readLine()) != null) {
178.         sbr.append(str).append('\n');
179.     }
180.     in.close();
181.     return sbr.toString();
182. }
183.
184. public static void main(String[] args) {
185.     // TODO Auto-generated method stub
186.     MainFrame mf = new MainFrame();
187.     //TinyCompiler tinyCompiler = new TinyCompiler();
188.     //mf.tinyCompiler = tinyCompiler;
```

```

189.         mf.setVisible(true);
190.     }
191. }
192.
193. class FilePanel extends JPanel {
194.     FilePanel(String str) {
195.         JLabel label = new JLabel(str);
196.         JTextField fileText = new JTextField(35);
197.         JButton chooseButton = new JButton("浏览...");
198.         this.add(label);
199.         this.add(fileText);
200.         this.add(chooseButton);
201.         clickAction ca = new clickAction(this);
202.         chooseButton.addActionListener(ca);
203.     }
204.
205.     public String getFileName() {
206.         JTextField jtf = (JTextField) this.getComponent(1);
207.         return jtf.getText();
208.     }
209.
210.     // 按钮响应函数
211.     private class clickAction implements ActionListener {
212.         private Component cmpt;
213.
214.         clickAction(Component c) {
215.             cmpt = c;
216.         }
217.
218.         public void actionPerformed(ActionEvent event) {
219.             JFileChooser chooser = new JFileChooser();
220.             chooser.setCurrentDirectory(new File("."));
221.             int ret = chooser.showOpenDialog(cmpt);
222.             if (ret == JFileChooser.APPROVE_OPTION) {
223.                 JPanel jp = (JPanel) cmpt;
224.                 JTextField jtf = (JTextField) jp.getComponent(1); //获取组
件
225.                 jtf.setText(chooser.getSelectedFile().getPath());
226.             }
227.         }
228.     }
229. }

```

InfoFrame. java

```
1. package gui;
2.
3. import java.awt.BorderLayout;
4. import java.awt.Color;
5. import java.awt.Container;
6. import java.awt.Dimension;
7. import java.awt.Toolkit;
8. import java.io.BufferedInputStream;
9. import java.io.BufferedReader;
10. import java.io.FileInputStream;
11. import java.io.IOException;
12. import java.io.InputStreamReader;
13.
14. import javax.swing.JFrame;
15. import javax.swing.JPanel;
16. import javax.swing.JTextArea;
17. import java.awt.*;
18.
19. public class InfoFrame extends JFrame {
20.
21.     private static final long serialVersionUID = 8766059377195109228L;
22.     private static String title;
23.     private static String fileName;
24.
25.     private static JTextArea text;
26.
27.     public InfoFrame() {
28.         init();
29.
30.     }
31.     public InfoFrame(String title,String fileName){
32.         this.title=title;
33.         this.fileName=fileName;
34.         init();
35.         this.setTitle(title);
36.         try {
37.             readFile(fileName);
38.         } catch (IOException e) {
39.
40.             e.printStackTrace();
41.         }
42.     }
43.     private void init() {
44.         Toolkit toolkit = Toolkit.getDefaultToolkit();
```

```

45.         Dimension screen = toolkit.getScreenSize();
46.         setSize(500, 400);
47.         super.setLocation(screen.width / 2 - this.getWidth() / 2, screen.hei
            ght / 2 - this.getHeight() / 2);
48.         setContentPane(createContentPane());
49.     }
50.
51.     private Container createContentPane() {
52.         JPanel pane = new JPanel(new BorderLayout());
53.         text = new TextArea();
54.         //msg.setBackground(Color.blue);
55.         text.setForeground(Color.BLUE);
56.         pane.add(BorderLayout.CENTER, text);
57.         return pane;
58.     }
59.
60.     private String readFile(String filename)
61.     throws IOException{
62.         StringBuilder sbr = new StringBuilder();
63.         String str;
64.         FileInputStream fis = new FileInputStream(filename);
65.         BufferedInputStream bis = new BufferedInputStream(fis);
66.         InputStreamReader isr = new InputStreamReader(bis, "UTF-8");
67.         BufferedReader in=new BufferedReader(isr);
68.         while((str=in.readLine())!=null){
69.             sbr.append(str).append('\n');
70.         }
71.         in.close();
72.         text.setText(sbr.toString());
73.         return sbr.toString();
74.     }
75.     public static String getTitl() {
76.         return title;
77.     }
78.
79.     public static void setTitl(String title) {
80.         InfoFrame.title = title;
81.     }
82.
83.     public static String getFileName() {
84.         return fileName;
85.     }
86.
87.     public static void setFileName(String fileName) {

```

```
88.         InfoFrame.fileName = fileName;
89.     }
90.
91.     public static TextArea getText() {
92.         return text;
93.     }
94.
95.     public static void setText(TextArea jText) {
96.         InfoFrame.text = jText;
97.     }
98.     public static void main(String[] args) {
99.         // TODO Auto-generated method stub
100. InfoFrame inf=new InfoFrame("测试","test.txt");
101. inf.setVisible(true);
102.     }
103.
104. }
```