编译原理课程实验报告

实验 3：语义分析

姓名	姜思琪	院系	计算机	学号	1160300814
任课教师	辛明影		指导教师	辛明影	
实验地点	格物 208		实验时间	2019.04.28 10.00-11.45	
实验课表现	出勤、表现得分		实验报告得分		实验总分	
	操作结果得分				

一、需求分析

得分

要求：阐述语义分析系统所要完成的功能。

在语法分析器的基础上设计实现类高级语言的语义分析器，基本功能如下：

（1） 能分析以下几类语句，并生成中间代码（三地址指令和四元式形式）：

- 声明语句（变量声明）
- 表达式及赋值语句分支语句：if_then_else
- 循环语句：do_while

（2）具备语义错误处理能力，包括变量或函数重复声明、变量或函数引用前未声明、运算符和运算分量之间的类型不匹配（如整型变量与数组变量相加减）等错误，能准确给出错误所在位置，并采用可行的错误恢复策略。输出的错误提示信息格式如下：

Error at Line [行号]：[说明文字]

（3）系统的输入形式：要求能够通过文件导入测试用例。测试用例要涵盖第（1）条中列出的各种类型的语句，以及第（2）条中列出的各种类型的错误。

（4）系统的输出分为两部分：一部分是打印输出符号表。另一部分是打印输出三地址指令或四元式序列.

除此之外，可以实现一些额外功能，例如自动类型转换，识别其它类型语义错误，如过程返回类型与声明类型不匹配；过程调用时实参与形参数目或类型不匹配；对非数组型变量使用数组访问操作符“[…]”；对普通变量使用过程调用操作符“call”；数组访问操作符“[…]”中出现非整数等

二、文法设计

得分

要求：给出如下语言成分所对应的语义动作

- 声明语句（变量声明
- 表达式及赋值语句
- 分支语句：if_then_else
- 循环语句：do_while

以上的语义动作我定义如下图：

语义规则

产生式	语义规则
0 : D->D D	
1 : D->proc id ; D S	D->proc id ; D S {enter(id.lexeme,proc,int);}
2 : D->T id ;	D->T id ; {enter(id.lexeme,T.type,offset);offset=offset+T.width;}
3 : T->X C	T->X {t=B.type;w=B.width;} C{T.type=C.type;T.width=C.width;}
4 : T->record D	
5 : E->E + E	E->E + E {E.addr=newtemp();gen(E.addr=E1.addr+E2.addr);}
6 : E->E * E	E->E * E {E.addr=newtemp();gen(E.addr=E1.addr*E2.addr);}
7 : E->- E	E->- E {E.addr=newtemp();gen(E.addr=uminus E1.addr);}
8 : E->(E)	E->(E) {E.addr=E1.addr}
9 : E->id	E->id {E.addr=lookup(id.lexeme);if E.addr==null then error}
10 : E->digit	E->digit {E.addr=lookup(digit.lexeme);if E.addr==null then error}
11 : E->L	E->L {E.addr=newtemp(); gen(E.addr=L.array[L.offset];}
12 : F->F , E	{把E.addr添加到q的队尾}
13 : F->E	{将q初始化为只包含E.addr}
14 : P->D P	P->{offset=0}D P
15 : P->S P	P->{S.next=newlabel();}S{label(S.next);} P
16 : P->no	
17 : B->B or B	B->{ B1.true=B.true; B1.false = newlabel();}B1 or {label(B1.false); B2.true=B....
18 : B->B and B	B->{ B1.false=B.false; B1.true = newlabel();}B1 and {label(B1.true); B2.true=B...
19 : B->not B	B->not { B1.true=B.false; B1.false = B.true ;}B1
20 : B->(B)	B->({ B1.true=B.true; B1.false = B.false ;} B1)
21 : B->E1 > E2	B->E1 > E2 {gen (if E1.addr > E2.addr goto B.true);gen(goto B.false);}
22 : B->true	B->true {gen(goto B.true);}
23 : B->false	B->false {gen(goto B.false);}
24 : S->id = E ;	S->id = E ; {p=lookup(id.lexeme);if p== nil then error;S.CODE=E.code||gen(p'=...
25 : S->L = E ;	S->L = E ; {gen(L.array[L.offset]=E.addr);}
26 : S->call id (F) ;	S->call id (F) ; {}
27 : S->if B { S }	S->if {B.true=newlabel();B.false=newlabel();} B { {label(B.true);S1.next=S.nex...
28 : S->if B { S } else { S }	S->if {B.true=newlabel();B.false=newlabel();}B { {label(B.true);S1.next=S.next...
29 : S->while B { S }	S->while{S.begin=newlabel();label(S.begin);B.true=nealabel();B.false=S.next;} ...
30 : S->S S	S->{S1.next=newlabel();}S1 {label(S1.next);S2.next=newlabel();} S2
31 : S->do { S } while B ;	
32 : C->[digit] C	C->[digit] C {C.type=array(num.val,C1.type);C.width=num.val*C1.width;}
33 : C->no	C->no {C.type=t;C.width=4;}
34 : L->id [E]	L->id [E] {L.array=lookup(id.lexeme);if L.array==null then error; L.type=L...
35 : L->L [E]	L->L [E] {L.array = L1L->L [E] {L.array = L1.array;L.type=L1.type.elem;t...
36 : X->int	X->int {X.type=int;X.width=4;}
37 : X->char	X->char {X.type=char;X.width=4;}

具体内容如下：

0 : D->D D

1 : D->proc id ; D S {enter(id.lexeme,proc,int);}

2 : D->T id ; {enter(id.lexeme,T.type,offset);offset=offset+T.width;}

3 : T->X {t=B.type;w=B.width;} C{T.type=C.type;T.width=C.width;}

4 : T->record D

5 : E->E + E {E.addr=newtemp();gen(E.addr=E1.addr+E2.addr);}

6 : E->E * E {E.addr=newtemp();gen(E.addr=E1.addr*E2.addr);}

7 : E->- E {E.addr=newtemp();gen(E.addr=uminus E1.addr);}

8 : E->(E) {E.addr=E1.addr}

9 : E->id {E.addr=lookup(id.lexeme);if E.addr==null then error}

10 : E->digit {E.addr=lookup(digit.lexeme);if E.addr==null then error}

11 : E->L {E.addr=newtemp(); gen(E.addr=L.array[L.offset];}

12 : F->F , E {把 E.addr 添加到 q 的队尾}

13 : F->E {将 q 初始化为只包含 E.addr}

14 : P->{offset=0}D P

15 : P->{S.next=newlabel();}S{label(S.next);} P

16 : P->no

17 : B->{ B1.true=B.true; B1.false = newlabel();}B1 or {label(B1.false); B2.true=B.true; B2.false = B2.false;} B2

18 : B->{ B1.false=B.false; B1.true = newlabel();}B1 and {label(B1.true); B2.true=B.true; B2.false = B2.false;}B2

19 : B->not { B1.true=B.false; B1.false = B.true ;}B1

20 : B->({ B1.true=B.true; B1.false = B.false ;} B1)

21 : B->E1 > E2 {gen (if E1.addr > E2.addr goto B.true);gen(goto B.false);}

22 : B->true {gen(goto B.true);}

23 : B->false {gen(goto B.false);}

24 : S->id = E ; {p=lookup(id.lexeme);if p== nil then error;S.CODE=E.code||gen(p'='E.addr);}

25 : S->L = E ; {gen(L.array[L.offset]=E.addr);}

26 : S->call id (F) ;{}
27 : S->if {B.true=newlabel();B.false=newlabel();} B { {label(B.true);S1.next=S.next}S1 }
28 : S->if {B.true=newlabel();B.false=newlabel();}B { {label(B.true);S1.next=S.next}S1 {gen(goto S.next);}} else { {label(B.false);S2.next=S.next}S2 }
29 : S->while{S.begin=newlabel();label(S.begin);B.true=nealabel();B.false=S.next;} B {{label(B.true);S1.next=S.begin;} S{gen(goto S.begin);} }
30 : S->{S1.next=newlabel();}S1 {label(S1.next);S2.next=newlabel();} S2
31 : S->do { S } while B ;
32 : C->[digit] C {C.type=array(num.val,C1.type);C.width=num.val*C1.width;}
33 : C->no {C.type=t;C.width=4;}
34 : L->id [E] {L.array=lookup(id.lexeme);if L.array==null then error; L.type=L.array.type.elem; L.offset = newtemp(); gen(L.offset=E.addr*L.type.width);}
35 : L->L [E] {L.array = L1L->L [E] {L.array = L1.array;L.type=L1.type.elem;t=newtemp();gen(t=E.addr*L.type.width); L.offset = newtemp(); gen(L.offset= L1.offset+t);}.array; L.type=L1.type.elem;t=newtemp();
36 : X->int {X.type=int;X.width=4;}
37 : X->char {X.type=char;X.width=4;}

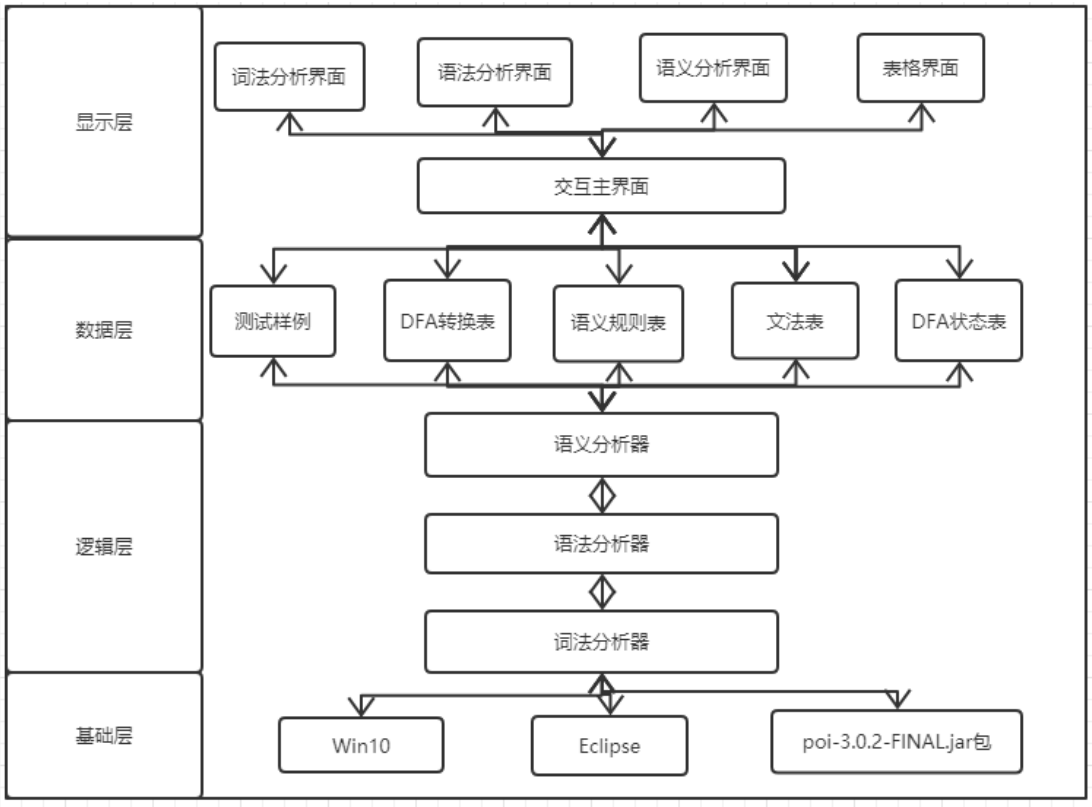
三、系统设计

得分

要求：分为系统概要设计和系统详细设计。

（1）系统概要设计：给出必要的系统宏观层面设计图，如系统框架图、数据流图、功能模块结构图等以及相应的文字说明。

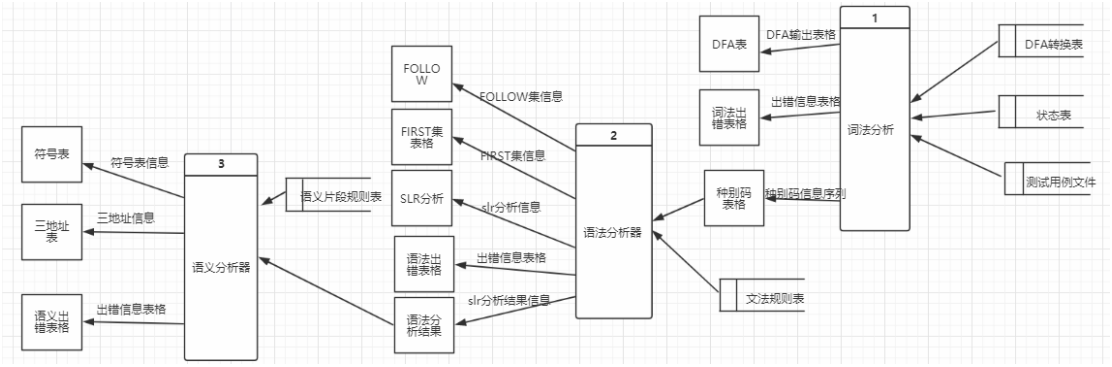
1. 系统框架图



本次实验主要可以分为四个层次：基础层是硬件环境，本次实验是在 win10 系统下

Eclipse 编辑器中 Java 语言实现的，利用 poi-3.0.2-FINAL.jar 完成的。逻辑层表明本实验利用词法分析器、语法分析器和语义分析器进行本次实验。数据层表明本次实验的输入文件有 DFA 状态表（finish.xls）、 DFA 转换表（DFA.xls）、文发表（fuzhi.xls）、语义规则表。本实验设计了一个交互界面，来可视化、简便化实验结果，即显示层的功能。

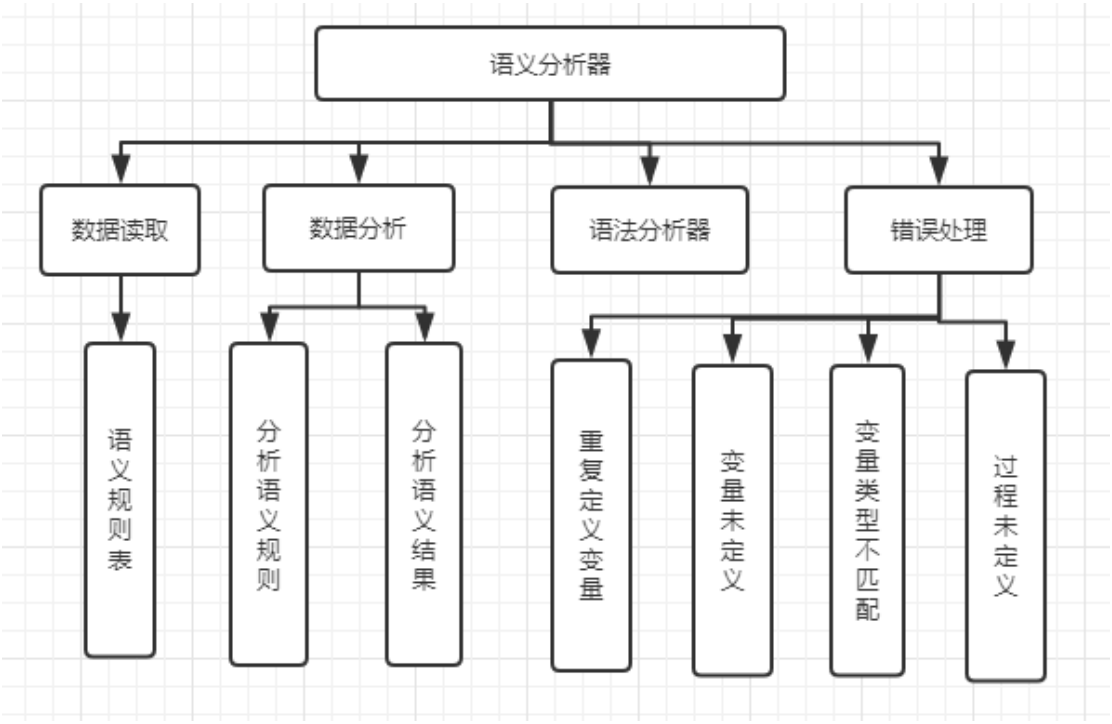
2. 数据流图



通过导入测试文件、DFA 转换表、DFA 状态表，利用 poi-3.0.2-FINAL.jar 实现 词法分析器，并将词法分析输出结果与文法表结合，构成语法分析器；利用语法分析结果和语义规则表实现语义分析器，然后一一输出所要求的结果，显示出分析结果、各种表格和出错表格等信息。

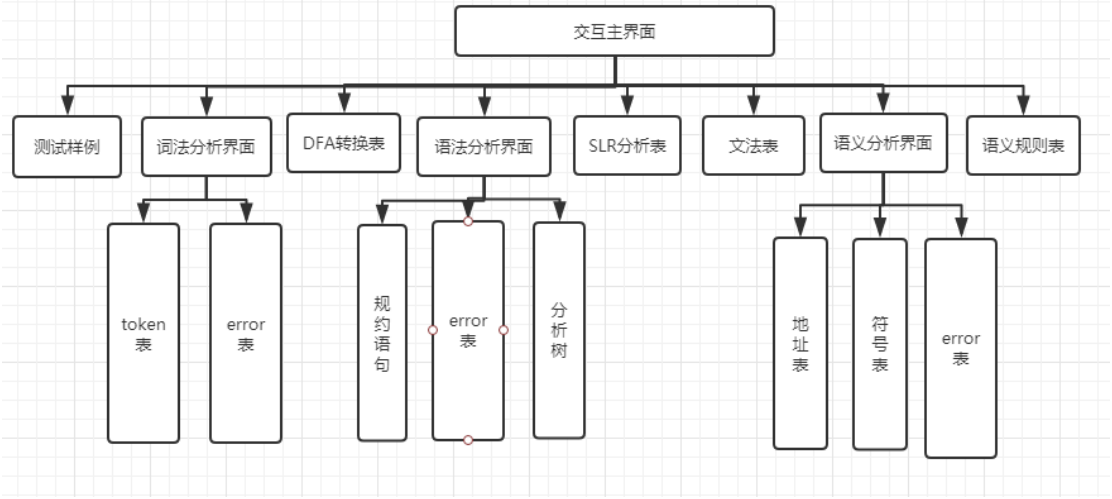
3. 功能模块结构图

语义分析模块：



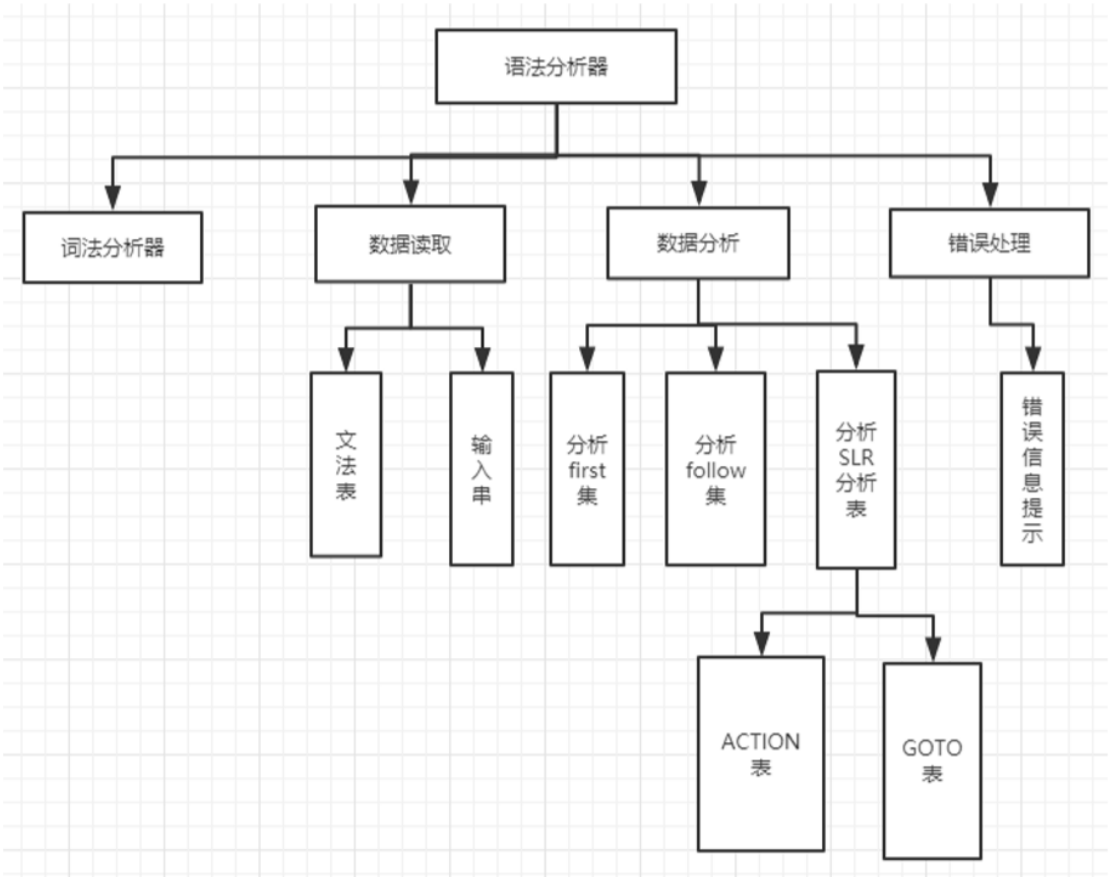
语义分析器是本次实验的重点。通过输入语义规则表和利用语法分析结果，采用自顶向下的递归方法，经行语义分析，从而输出结果。

显示模块：



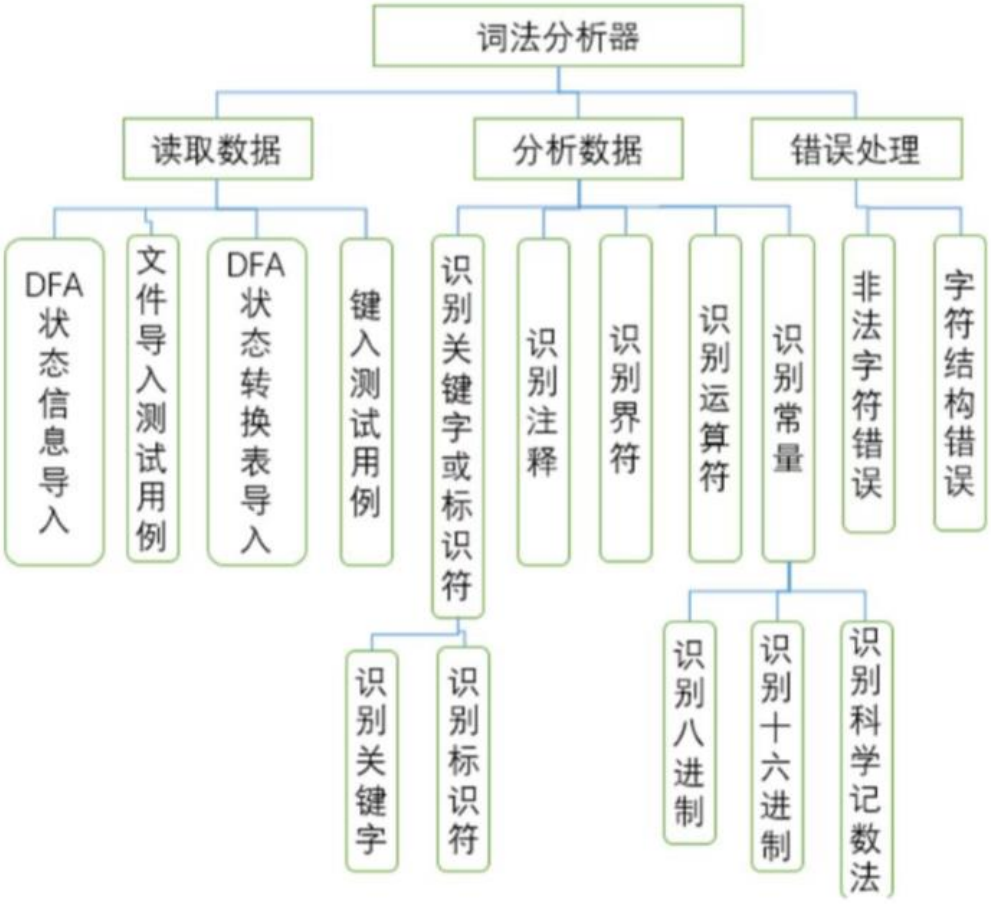
这个模块主要功能是可视化交互过程和结果。输出界面一共分四种：词法分析的界面、 语法分析的界面、语义分析界面和各种表格。在输入框输入测试样例，点击测试，选择要看的内容， 本次实验主要看语义分析界面和 语义规则表。

语法分析模块：



语法分析器是实验 2 的重点，在此不做赘述。通过利用词法分析器的结果和文发表，生成 first 集、 follow 集和 SLR 分析表。我在本次实验设计了可以识别 3 种语法错误，分别是括号数目不 匹配（如 while （true）） ）、缺少运算符（如 i=i 2）、缺少运算分量（如 i = +2），会输出行 号和错误类型，遇到缺少的错误会进行输入栈的压入，遇到数目不匹配会弹出并继续识别。

词法分析模块：



词法分析器为实验一的主要内容，这里不做赘述，只简要概括。在此模块实现词法分析，识别关键字、标 识符、界符、注释、常量（十六进制、八进制、科学记数法）等，同时识 别非法字符输入、结构错误、提示注释未封闭等，通过可视化模块，将结果显示。

（2）系统详细设计：对如下工作进行展开描述

✓ 核心数据结构的设计

沿用实验二的代码，增添的结构如下：

1. AddrNum.java
   地址与语句的映射表。在生成三地址指令回填的时候，用 num 代替 addr 即可，这样简化了回填的步骤。
   String addr:地址名称；Int num :对应的语句
2. CharTable.java
   存储符号表。
   String chara; 符号名称；String type;符号类型；String offset;偏移量
3. grammerSemanticloca.java
   含语义规则的产生式。
   int grammerNum;产生式编号；int rulelLoc;语义规则片段所在位置；int ruleNum：对应的语义片段的序号
4. grammerTable.java
   读入文法表。
   String name ： 文法前面的非终结符；String[] value：产生式
5. ReeSemanticRecord.java

执行语义片段动作。

int treeNodeNum：树节点编号；String treeNodeName：结点名字；String property：属性；String value：值

6. FourAddr.java

便于显示四元式。

String op; String param1; String param2; String toaddr;

✓ 主要功能函数说明

1. void semanticTest(grammerTable[] grammerTable, ArrayList<SLRTree> slrTreeArray, ArrayList<RreeSemanticRecord> treeSemanticRecord, grammerSemanticLoca[] grammersemanticLoca, ArrayList<String> addrList, ArrayList<String> addrResult, ArrayList<AddrNum> addrNum, ArrayList<CharTable> charTable, ArrayList<FourAddr> fourAddr, ArrayList<String> param,int num)

自底向上遍历分析树，找到当前结点的产生式，遍历此产生式的右部，分别调用record()函数执行相应语义片段，若当前结点非叶节点则递归调用函数semanticTest()。输入依次为文法表、语法分析树、树节点表、含语义片段的文法、地址表、输出结果、地址与序号对应表、符号表。

2. void record(grammerSemanticLoca[] grammersemanticLoca,ArrayList<SLRTree> slrTreeArray, ArrayList<RreeSemanticRecord> treeSemanticRecord, ArrayList<String> addrList, ArrayList<String> addrResult,ArrayList<AddrNum> addrNum, ArrayList<CharTable> charTable, ArrayList<FourAddr> fourAddr, ArrayList<String> param, int treeNodeNum, int grammerNum,int loc)

根据 isExitSemanticRule()函数判断是否有语义片段执行，若有则执行，调用 semanticRule 类的函数，从而执行语义片段。

3. String findValue(ArrayList<RreeSemanticRecord> treeSemanticRecord, int treeNodeNum, String property)

返回树节点的值（孩子结点）。输入为树节点表、节点编号、结点属性。

4. ArrayList<String> change(ArrayList<String> addrResult, ArrayList<AddrNum> addrNum)

5. void initRecord(ArrayList<RreeSemanticRecord> treeSemanticRecord, ArrayList<SLRTree> slrTreeArray)

存储树节点，将语法分析生成的树节点存到结构 treeSemanticRecord 中。输入为语义分析树和语法分析书。

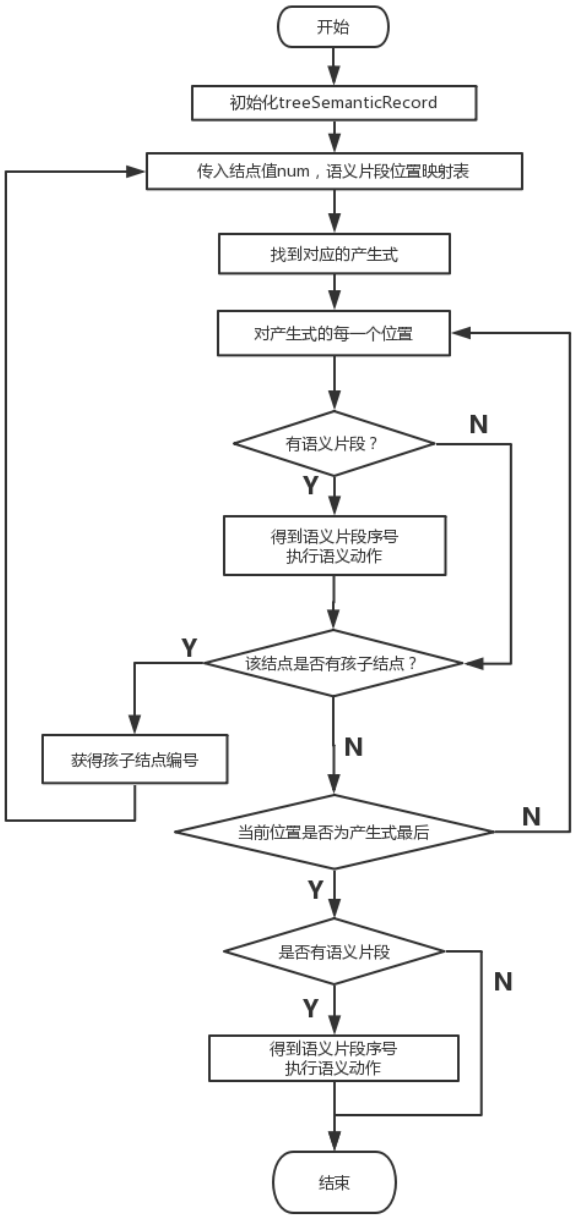
6. int isExitSemanticRule(int grammerNum, int loc, grammerSemanticLoca[] grammersemanticLoca)

获得当前产生式当前位置是否有语义规则片段，有则返回片段号，无则返回-1。输入为产生式编号、位置、含语义片段的文法表。

7.

✓ 程序核心部分的程序流程图

语义分析流程：



四、系统实现及结果分析	得分	

要求：对如下内容展开描述。

（1） 系统实现过程中遇到的问题；

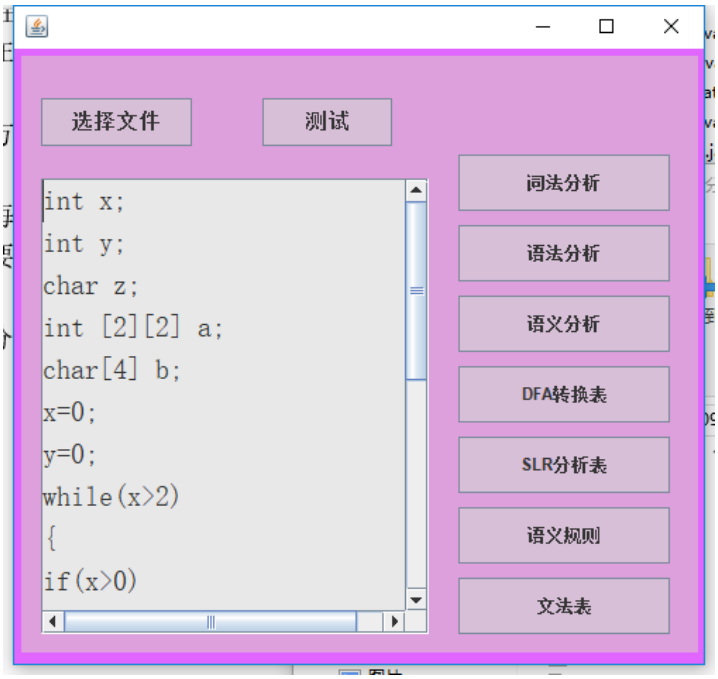
1. 文法表和语义规则无法一一对应。
   为了解决此问题，我将文法表打印出，语义规则表打印出。然后写代码和输入文件一一对应，避免出错。
2. 语法分析中，将语法分析的错误根据错误类型在树上添加或删除结点以保证构造一个正确的树。然后就可以传给语义分析进行实验。
3. 用自底向上的分析方法需要将予以片段放在最后，

（2） 针对一测试程序输出其语义分析结果；

输入文本具体样例见输入文件语义正确用例.txt 和语义错误用例.txt。

代码运行结果图如下：

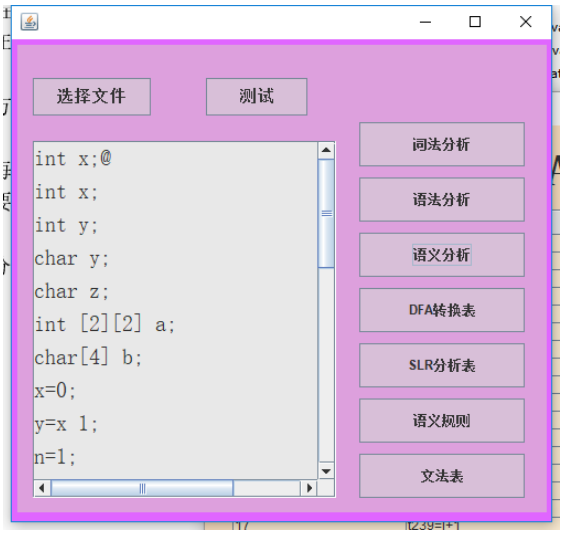
正确示例结果图：



Address TABLE

序号	三地址	四元式
1	x=0	< = , 0 , - , x >
2	y=0	< = , 0 , - , y >
3	if x > 2 goto 5	< j> , x , 2 , L5 >
4	goto 16	< j , - , - , L1 >
5	if x > 0 goto 7	< j> , x , 0 , L6 >
6	goto 13	< j , - , - , L7 >
7	t168= x*16	< * , x , 16 , t168 >
8	t169=y*8	< * , y , 8 , t169 >
9	t170=t168+t169	< + , t168 , t169 , t170 >
10	t173=x+y	< + , x , y , t173 >
11	a[t170] = t173	< = , t173 , - , a[t170] >
12	goto 3	< j , - , - , L4 >
13	t176= 0*16	< * , 0 , 16 , t176 >
14	b[t176] = z	< = , z , - , b[t176] >
15	goto 3	< j , - , - , L4 >
16	t193=i+1	< + , i , 1 , t193 >
17	i=t193	< = , t193 , - , i >
18	param x	< param , x , - , - >
19	param y	< param , y , - , - >
20	call demo , 2	< call , demo , 2 , - >

错误示例结果图：



Address TABLE

序号	三地址	四元式
1	x=0	< = , 0 , - , x >
2	t195=x+1	< + , x , 1 , t195 >
3	y=t195	< = , t195 , - , y >
4	if x > 2 goto 6	< = , 1 , - , n >
5	goto 17	< = , x , - , z >
6	if x > 0 goto 8	< j> , x , 2 , L7 >
7	goto 14	< j , - , - , L1 >
8	t213= x*16	< j> , x , 0 , L8 >
9	t214=y*8	< j , - , - , L9 >
10	t215=t213+t214	< * , x , 16 , t213 >
11	t219=x+y	< * , y , 8 , t214 >
12	a[t215] = t219	< + , t213 , t214 , t215 >
13	goto 4	< + , x , y , t219 >
14	t222= 0*16	< = , t219 , - , a[t215] >
15	b[t222] = z	< j , - , - , L6 >
16	goto 4	< * , 0 , 16 , t222 >
17	t239=i+1	< = , z , - , b[t222] >
18	i=t239	< j , - , - , L6 >
19	error 5 demo2	< + , i , 1 , t239 >

（3） 输出针对此测试程序经过语义分析后的符号表；

正确事例的符号表：

Symbol TABLE

序号	符号	偏移
x	int	0
y	int	4
z	char	8
a	array(2,array(2,int))	12
b	array(4,char)	28
i	int	44
j	int	48
demo	proc	int

错误事例的符号表：

Symbol TABLE

序号	符号	偏移
x	int	0
x	int	4
y	int	8
y	char	12
z	char	16
a	array(2,array(2,int))	20
b	array(4,char)	36
i	int	52
j	int	56
x	proc	int

（4） 输出针对此测试程序对应的语义错误报告；

Error TABLE

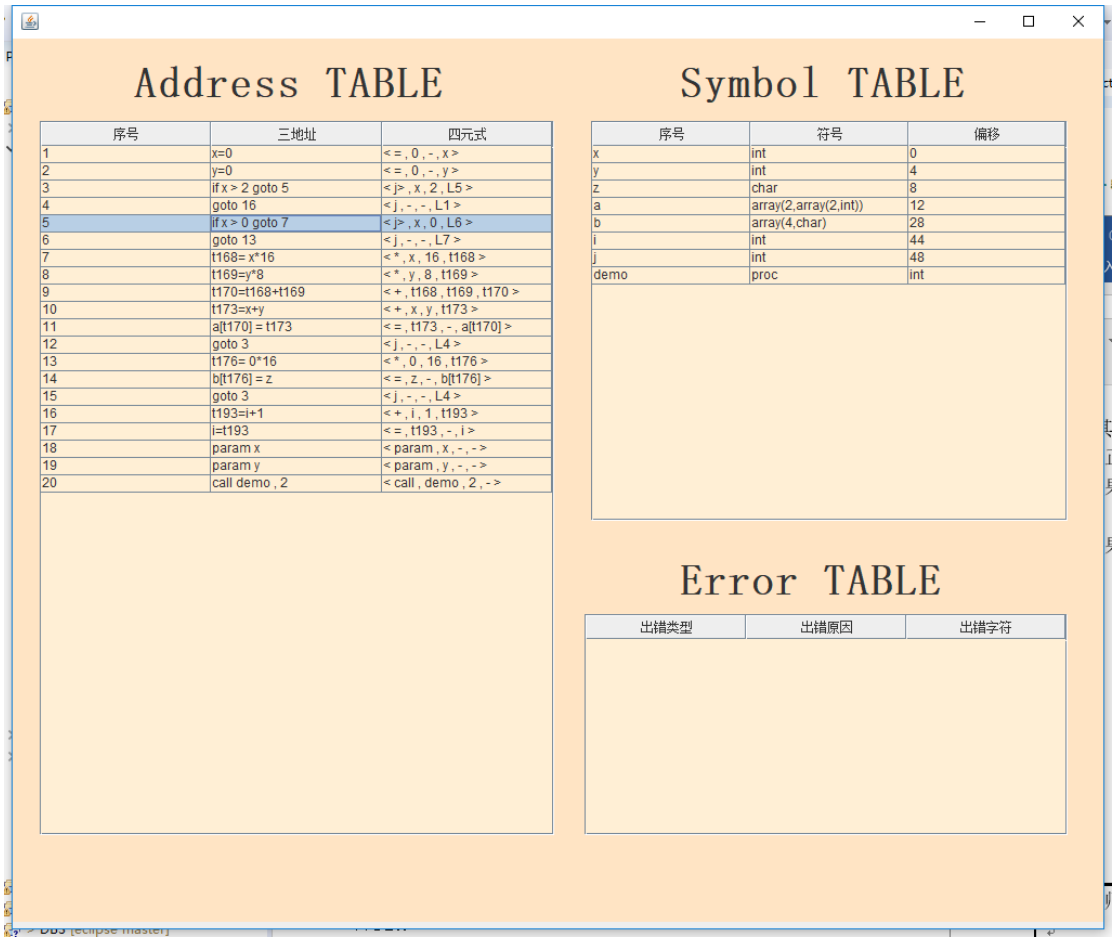
出错类型	出错原因	出错字符
2	变量重复定义	x
2	变量重复定义	y
1	变量未定义	n
3	变量不匹配	z(char) x(int)
4	过程名定义冲突	x
5	过程名未定义	demo2

（5） 对实验结果进行分析。

注：其中的测试样例需先用已编写的词法分析程序进行处理。

结果均正确，整体界面效果如下：

语义分析界面：

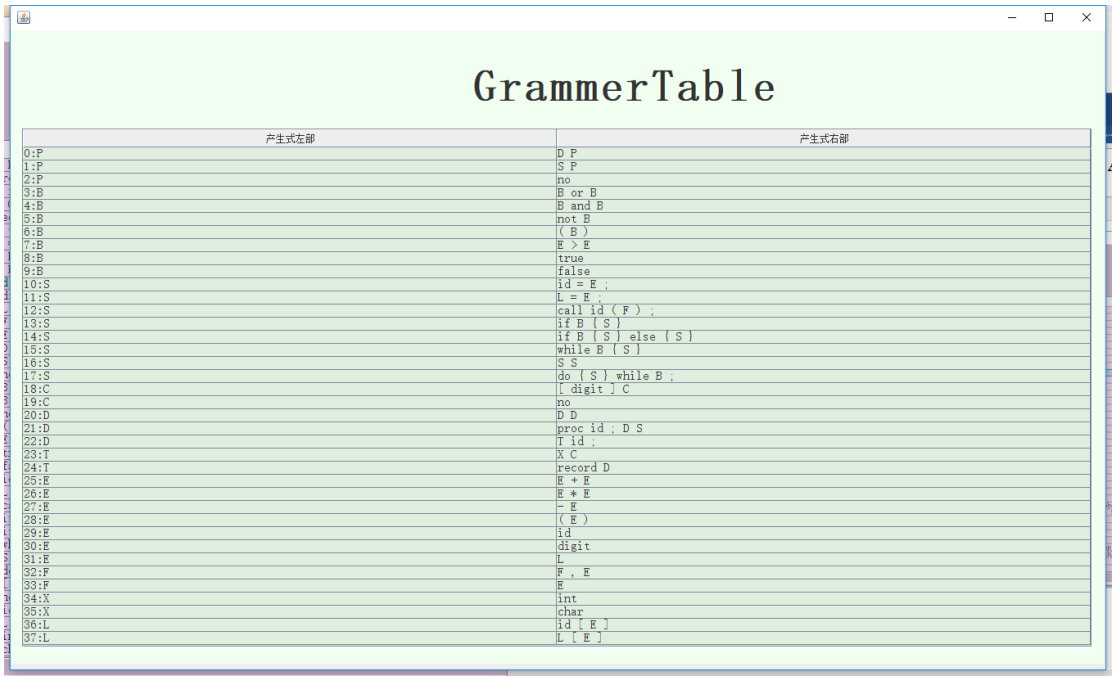


语义规则表：

语义规则

产生式	语义规则
0 : D->D D	
1 : D->proc id ; D S	D->proc id ; D S {enter(id.lexeme,proc,int);}
2 : D->T id ;	D->T id ; {enter(id.lexeme,T.type,offset);offset=offset+T.width;}
3 : T->X C	T->X {t=B.type;w=B.width;} C{T.type=C.type;T.width=C.width;}
4 : T->record D	
5 : E->E + E	E->E + E {E.addr=newtemp();gen(E.addr=E1.addr+E2.addr);}
6 : E->E * E	E->E * E {E.addr=newtemp();gen(E.addr=E1.addr*E2.addr);}
7 : E->- E	E->- E {E.addr=newtemp();gen(E.addr=uminus E1.addr);}
8 : E->(E)	E->(E) {E.addr=E1.addr}
9 : E->id	E->id {E.addr=lookup(id.lexeme);if E.addr==null then error}
10 : E->digit	E->digit {E.addr=lookup(digit.lexeme);if E.addr==null then error}
11 : E->L	E->L {E.addr=newtemp() gen(E.addr=L.array[L.offset];}
12 : F->F , E	{把E.addr添加到q的队尾}
13 : F->E	{将q初始化为只包含E.addr}
14 : P->D P	P->{offset=0}D P
15 : P->S P	P->{S.next=newlabel();}S{label(S.next);} P
16 : P->no	
17 : B->B or B	B->{ B1.true=B.true; B1.false = newlabel();}B1 or {label(B1.false); B2.true=B.true; B2.false = B2.false;} B2
18 : B->B and B	B->{ B1.false=B.false; B1.true = newlabel();}B1 and {label(B1.true); B2.true=B.true; B2.false = B2.false;}B2
19 : B->not B	B->not { B1.true=B.false; B1.false = B.true ;}B1
20 : B->(B)	B->({ B1.true=B.true; B1.false = B.false ;} B1)
21 : B->E > E	B->E1 > E2 {gen (if E1.addr > E2.addr goto B.true);gen(goto B.false);}
22 : B->true	B->true {gen(goto B.true);}
23 : B->false	B->false {gen(goto B.false);}
24 : S->id = E ;	S->id = E ; {p=lookup(id.lexeme);if p== nil then error;S.CODE=E.code||gen(p' =' E.addr);}
25 : S->L = E ;	S->L = E ; {gen(L.array[L.offset]=E.addr);}
26 : S->call id (F) ;	S->call id (F) ;{}
27 : S->if B { S }	S->if {B.true=newlabel();B.false=newlabel();} B { {label(B.true);S1.next=S.next}S1 }
28 : S->if B { S } else { S }	S->if {B.true=newlabel();B.false=newlabel();}B { {label(B.true);S1.next=S.next}S1 {gen(goto S.next);}} else { {label(B.false...
29 : S->while B { S }	S->while{S.begin=newlabel();label(S.begin);B.true=nealabel();B.false=S.next;} B {{label(B.true);S1.next=S.begin;} S{gen(goto...
30 : S->S S	S->{S1.next=newlabel();}S1 {label(S1.next);S2.next=newlabel();} S2
31 : S->do { S } while B ;	
32 : C->[digit] C	C->[digit] C {C.type=array(num.val,C1.type);C.width=num.val*C1.width;}
33 : C->no	C->no {C.type=t;C.width=4;}
34 : L->id [E]	L->id [E] {L.array=lookup(id.lexeme);if L.array==null then error; L.type=L.array.type.elem; L.offset = newtemp(); gen(L....
35 : L->L [E]	L->L [E] {L.array = L1L->L [E] {L.array = L1.array;L.type=L1.type.elem;t=newtemp();gen(t=E.addr*L.type.width); L.offse...
36 : X->int	X->int {X.type=int;X.width=4;}
37 : X->char	X->char {X.type=char;X.width=4;}

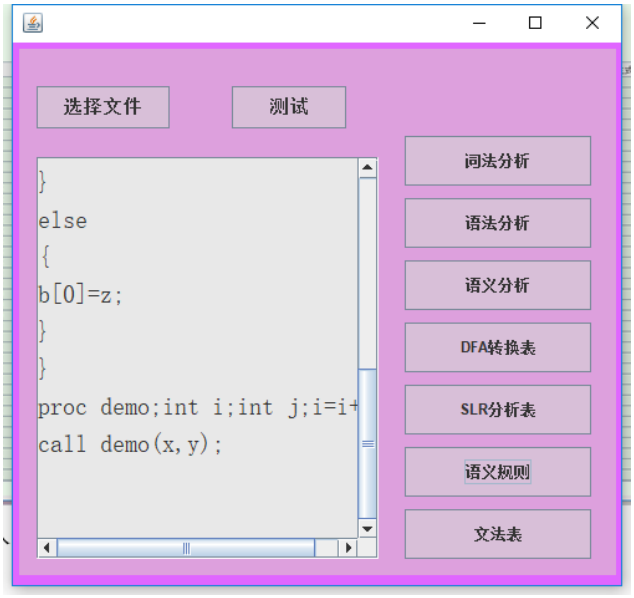
文法表：



GrammerTable

产生式左部	产生式右部
0:P	D P
1:P	S P
2:P	no
3:B	B or B
4:B	B and B
5:B	not B
6:B	(B)
7:B	E > E
8:B	true
9:B	false
10:S	id = E ;
11:S	L = E ;
12:S	call id (F) ;
13:S	if B { S }
14:S	if B { S } else { S }
15:S	while B { S }
16:S	S S
17:S	do { S } while B ;
18:C	[digit] C
19:C	no
20:D	D D
21:D	proc id ; D S
22:D	T id ;
23:T	X C
24:T	record D
25:E	E + E
26:E	E * E
27:E	- E
28:E	(E)
29:E	id
30:E	digit
31:E	L
32:F	F , E
33:F	E
34:X	int
35:X	char
36:L	id [E]
37:L	L [E]

输入界面：


选择文件
测试
}
else
{
b[0]=z;
}
}
proc demo;int i;int j;i=i+
call demo(x,y);
词法分析
语法分析
语义分析
DFA转换表
SLR分析表
语义规则
文法表

指导教师评语：

日期：