编译原理课程实验报告

实验 3：语义分析

姓名	张泽宇	院系	计算机科学与技术学院	学号	1163300620

任课教师	辛明颖		指导教师	辛明颖		
实验地点	格物 208		实验时间	2019.04.28		
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

语义分析的主要任务是收集标识符的属性信息以及语义的检查，从而生成中间代码的过程，本次实验采用的是识别 LR（1）文法的语法制导翻译技术，即一边进行语句的归约一边进行语句的翻译，实现的基本功能如下：

（1） 能分析一下几类语句，并生成相应的中间代码（三地址指令和四元式形式）：

- 声明语句（变量声明）
- 表达式以及赋值语句（包括数组元素的引用和赋值）
- 分支语句：if_then_else
- 循环语句：do_while

（2） 具备语义错误处理能力，包括阿变量或函数重复声明、变量或函数引用前未声明、运算符与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，能准确给出错误的所在位置，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的测试用例。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） 系统的输出分为四部分：一部分是打印输出符号表，第二部分是打印输出的中间代码的三地址指令形式，第三部分是打印输出中间代码的四元式序列的形式，最后一部分是打印语义分析中的错误信息。

（5） 实现的一些额外功能：可以实现算数运算中的自动类型转化，识别其他类型的语义错误，例如：变量在使用之前未进行定义、变量的重复定义等等。

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

P → PROGRAM {offset := 0} ID D ; S

D → D ; D

D → id : T{enter(id.name,T.type,offset);offset:=offset + T.width}

T → integer{T.type := real; T.width := 8}

T → real{T.type := real; T.width := 8}

T→array[num] of T1 {T.type := array(num.val,T1.type);T.width := num.val x T1.width}

- 表达式及赋值语句

S→Left:=E{**if** Left.offset=**null then** /*Left是简单变量**id***/

gencode(Left.addr ':=' E.addr);

else

gencode(Left.addr '[' Left.offset '] ' ':=' E.addr)} /*Left是数组元素*/

$E \rightarrow E_1+E_2$ {E.addr:=newtemp;gencode(E.addr ':='E_1.addr'+'E_2.addr)}

$E \rightarrow (E_1)$ {E.addr:= E_1.addr}

E→Left{**if** Left.offset=**null then** /*Left是简单**id***/

E.addr:= Left.addr

else begin /*Left是数组元素*/

E.addr:=newtemp;

gencode(E.addr ':=' Left.addr ' [' Left.offset ']')

end}

Left→Elist]{ Left.addr:=newtemp;

Left.offset:=newtemp;

gencode(Left.addr ':=' c(Elist.array));

gencode(Left.offset ':=' Elist.addr '*' width(Elist.array))}

Left→**id**{Left.addr:=**id**.addr; Left.offset:=**null**}

$Elist \rightarrow Elist_1$, E{t:=newtemp;m:= $Elist_1$.ndim+1;

gencode(t ':=' $Elist_1$.addr '*' limit($Elist_1$.array, m)); /*计算$e_{m-1} \times n_m$ */

gencode(t ':=' t '+' E.addr); /* 计算+ i_m */

Elist.array:= $Elist_1$.array;

Elist.addr:=t;

Elist.ndim:=m}

Elist→**id**[E {Elist.array:=**id**.addr; Elist.addr:= E.addr; Elist.ndim:=1}

- 分支语句：if_then_else

S→**if** B **then** M_1 S_1 N **else** M_2 S_2{backpatch(B.truelist, M_1.quad);

backpatch(B.falselist, M_2.quad);

S.nextlist := merge(S_1.nextlist, merge(N.nextlist, S_2.nextlist))}

N→ε{N.nextlist := makelist(nextquad); gencode('goto –')}

M→ε{M.quad := nextquad}

S→**if** B **then** M S_1{backpatch(B.truelist, M.quad);

S.nextlist := merge(B.falselist, S_1.nextlist)}

- 循环语句：do_while

S→**while** M_1 B **do** M_2 S_1{backpatch(S_1.nextlist, M_1.quad);

backpatch(B.truelist,M_2.quad);S.nextlist:=B.falselist; gencode('goto'M_1.quad)}

S→**begin** L **end**{S.nextlist:=L.nextlist}

S→A{S.nextlist := **nil**}

L→L_1;MS{backpatch(L_1.nextlist, M.quad); L.nextlist := S.nextlist}

L→S{L.nextlist := S.nextlist}

三、系统设计

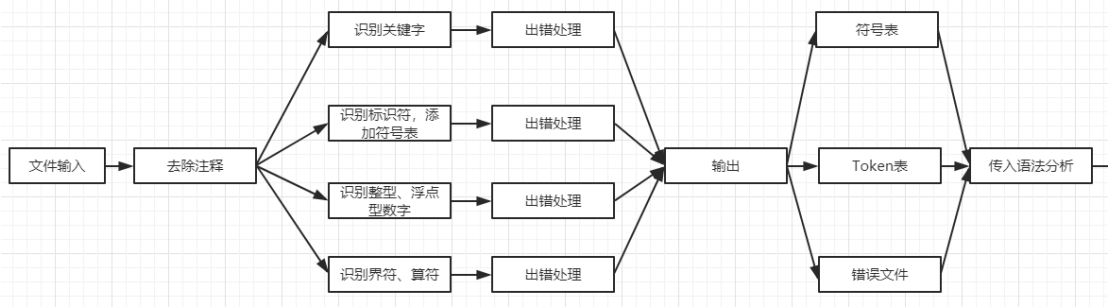
得分

（1）系统概要设计：

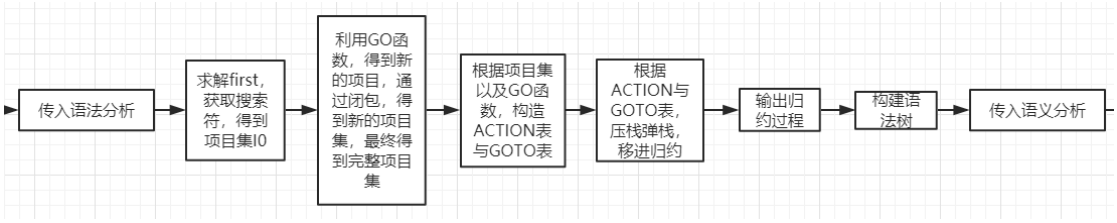
从根遍历语法树，遍历树节点的每个子节点，查找此节点存在的语义片段，如果存在，执行该语义片段，判断此子节点是否为叶节点，如果不是，遍历此节点，重复上述操作，从而实现回填的效果(作为一个树节点，遍历其所有子节点后，才会实现完成该产生式的语义动作)。遍历完成后，得到四元式序列，完成符号表的填充，语义分析结束。

系统框架图:由于框架过大，分为三部分展示

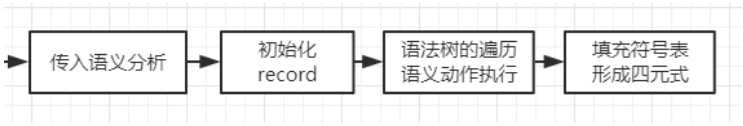
第一部分展示如下：



第二部分展示如下：

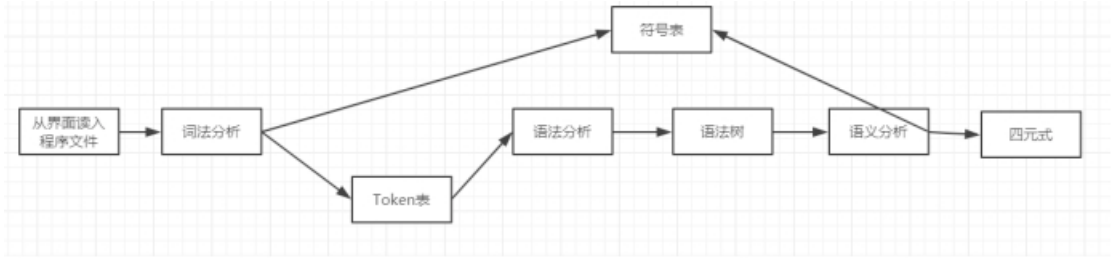


第三部分展示如下：



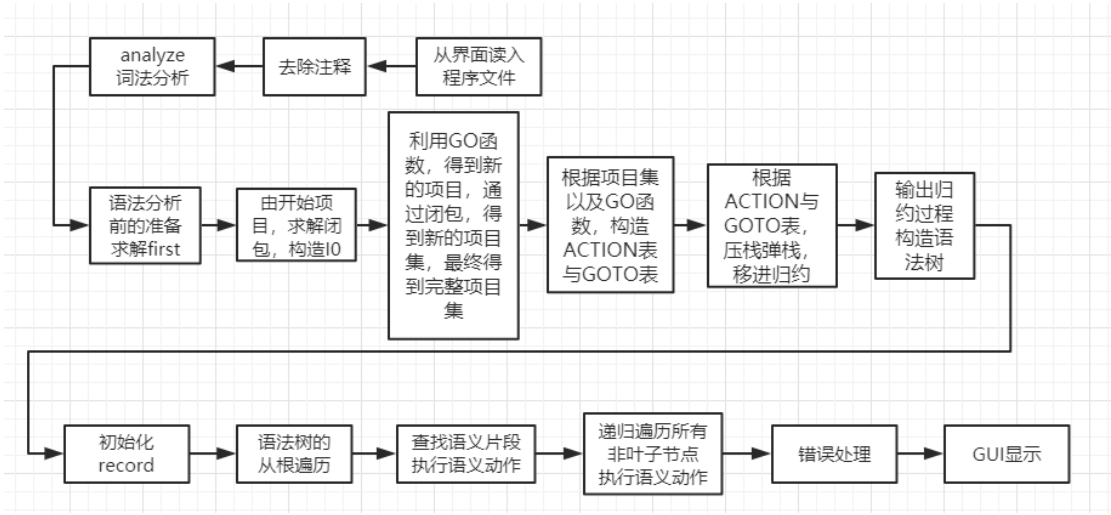
从界面读入文件后，经过词法分析得到 Token 表与符号表，经过语法分析后，得到语法树，经过语义分析后，重填符号表，得到四元式组。

数据流图



功能模块图

读入文件后，去除注释，求解 first 集，由开始项目构造 I0，根据 GO 函数与闭包函数得到所有项目集，构造 ACTION 表与 GOTO 表，将词法分析后得到的结果构建成字符串，根据 ACTION 表与 GOTO 表得到规约过程与规约产生式根据得到的规约产生式构造语法树，对得到的语法树进行遍历，对每个树节点执行对应的语义片段，得到最终的四元式序列与符号表，并在 GUI 上显示。



（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

Token：

属性：String type(种别码)

属性：String string(字符)
属性：String error(错误判断)
属性：int location(整型数值或标识符在符号表中的位置)
属性：double location1(浮点型数值)
(LRFormula)项目：
属性：String before 产生式的左部
属性：String[] next 产生式的右部
属性：int flag 圆点所在的位置
属性：String tags 项目的搜索符
(grammerTable)产生式表
属性：String name 产生式的左部
属性：String[] value 产生式的右部
(ACTIONTable)ACTION 表与 GOTO 表
属性：int state 状态编号
属性：String inout 输入符号
属性：String[] action 采取的操作(S 移入 R 归约 G GOTO 表 E 出错)
LRTree 语法树的节点
属性：String name：树节点的名字
属性：String[] childId：此节点的子节点表
Addr 地址表:
属性：String addr：地址标识符
属性：int num：对应地址
SymbolTable 符号表
属性：String character：字符
属性：String type：类型
属性：String offset：偏移量
FourAddr 四元式：
属性：String op：操作符
属性：String param1：参数 1
属性：String param2：参数 2
属性：String result：结果
grammerSemantic 语法语义对应表：
属性：String grammerNum：产生式编号
属性：String ruleLoc：语义片段所在位置
属性：String ruleNum：语义片段编号
treeSem 语义树：
属性：String treenodenum：树节点编号
属性：String treenodename：树节点名字
属性：String property：属性
属性：String value：值

✓ 主要功能函数说明

semantic(grammerTable[]grammerTable,ArrayList<LRTree>lrTreeArray, ArrayList<treeSem> treeSemlist, grammerSemantic[] grammersemanticlist,ArrayList<String> addrList,

ArrayList<String> addrResult, ArrayList<Addr> addr,ArrayList<FourAddr>fourAddr, ArrayList<String>param,int num 执行语义操作

参数介绍：

grammerTable[]grammerTable 文法表

ArrayList<LRTree>lrTreeArray 语法树序列

ArrayList<treeSem> treeSemlist 语义树序列(相对于语法树节点添加属性和值)

grammerSemantic[] grammersemanticlist 语义片段对应表

ArrayList<Addr> addr 地址表

ArrayList<FourAddr>fourAddr 四元式表

int num 树节点标号

实现方式：

对于传入的某个树节点，遍历它所有的子节点，查找此节点存在的语义片段，如果存在，执行该语义片段，判断此子节点是否为叶节点，如果不是，将此子节点所在编号作为参数传入 semantic，递归调用此函数，从而实现回填的效果(作为一个树节点，遍历其所有子节点后，才会实现完成该产生式的语义动作)。

语义操作：

普通变量声明语句：

T->X(1) C(2)

1. 在 1 处执行语义片段时，存储 X.type 与 X.width(在规约出 X->int 或 X->char 时，X.type 与 X.width 已知)在 t、w 中
2. 在规约 C->时，将 t 与 w 的值传给 C
3. 在 2 处执行语义片段时，将 C.type 与 C.width 传给 T

数组变量声明语句：

T->X(1) C(2)

1. 在 1 处执行语义片段时，存储 X.type 与 X.width(在规约出 X->int 或 X->char 时，X.type 与 X.width 已知)在 t、w 中
2. 在规约 C->时，将 t 与 w 的值传给 C
3. 在规约 C->[digit] C 时，将 array[digit.lexme，C.type]作为 C.type，将 digit.lexme*C.width 作为 C.width
4. 在 2 处执行语义片段时，将 C.type 与 C.width 传给 T

D->T id ;

在符号表中添加 id 对应的标识符，类型，offset，更新 offset，如果符号表中已经存在该标识符，报错(重复声明)

赋值语句：以 E->E+E 为例

获取树节点的第一与第三个子节点(参数)，查找语义树表，获取这两个子节点对应的值，同时查询符号表获取这两个子节点对应的类型，如果不是 int，返回错误，并将这两个子节点传入四元式的参数中，获取树节点的第二子节点，作为操作符传入四元式的 op 中，完成四元式的构建，加入四元式表中。

普通变量赋值语句：

S->id = E

获取树节点的第二与第四个子节点(参数)，查找语义树表，获取这两个子节点对应的值，将这两个子节点传入四元式的参数中，将=作为操作符传入四元式的 op 中，完成四元式的构建，加入四元式表中。在符号表中查找这两个子节点，如果没有找到或者这两个子节点的类型不一致，返回错误。

数组赋值

L->id [E]

L->L1 [E]

S->L = E ;

通过获取 E 的数值，并在符号表中查找对应类型的大小，得到对应 offset，L.offset=E.addr*L.type.width，t=E.addr*L.type.width 赋值过程与普通变量赋值相同。

分支语句：if_then_else

S->if B(1){S(2)}else{S(3)}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 }

在语义片段 1 位置，新建 B.true 与 B.false 的接口，在语义片段 2 位置填充 B.true 的跳转位置，设定此处 S1.next 为 S.next，在语义片段 3 位置填充 B.false 的跳转位置，设定此处 S2.next 为 S.next

S->S S S->{S1.next=newlabel();}S1 {label(S1.next);S2.next=newlabel();} S2

在此处设定和填充 S.next

循环语句

S->while (1)B{ (2) S(3) }

S->while{S.begin=newlabel();label(S.begin);B.true=newlabel(); B.false=S.next;} B {{label(B.true);S1.next=S.begin;} S{gen(goto S.begin);} }

在语义片段 1 位置,新建 S.begin 与 B.true 的接口，并填充 S.begin 的接口，设定 B.false 的接口为 S.next，在语义片段 2 位置，填充 B.true 的接口，设定此处 S1.next 为 S.next，在语义片段 3 位置，设定此处四元式的 result 为 S.begin。

函数调用

D->proc id ; D S

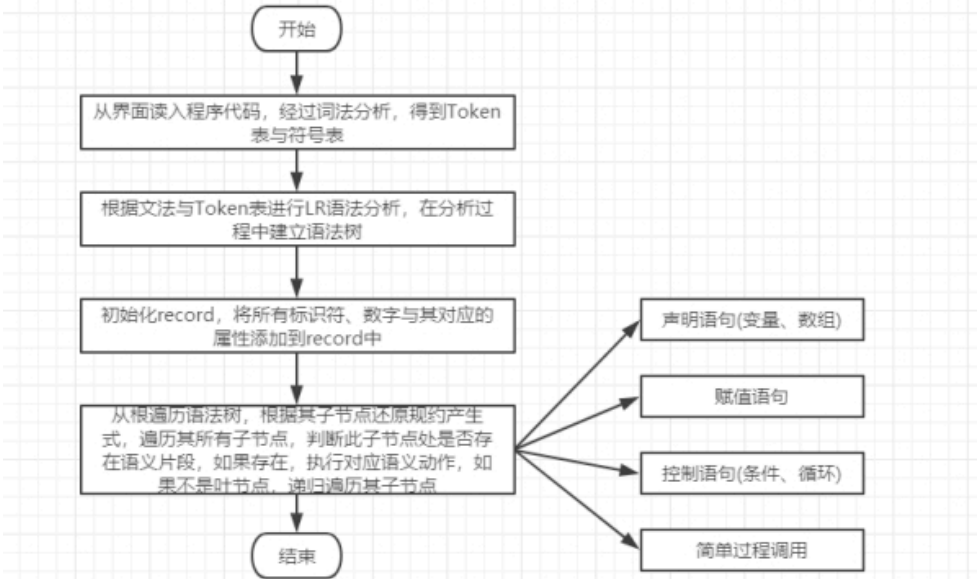
F->F , E

F->E

S->call id (F) ;

在 F->F , E 处添加参数表，在 S->call id (F) ;处添加进四元式表，D->proc id ; D S 遍历符号表，如果过程名在符号表中已经存在，报错(过程名重复定义)，S->call id (F) ;遍历符号表，如果过程名在符号表中不存在，报错(过程名不一致)

✓ 程序核心部分的程序流程图



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；

- 在语法制导翻译的过程中，符号表中的变量偏移量没有统一标准。解决办法：设置了一个记录变量偏移量的全局静态变量 offset。
- 在进行变量的算数运算时，没能记录临时变量的类型。解决办法：为每个临时变量都生成对应的类型，并将临时变量同样加入到符号表当中。
- 在算数运算过程中，如果出现了强制的类型变量的转换，要按照转换后的变量类型来分配相应的地址。
- 语法制导翻译的回填技术，要主要处理插入的空串所带来的语义动作。需要在前面生成语法分析表的时候考虑空串的情况。

  在进行有关变量定义的语义错误检查时，要查询所建立好的符号表来进行相关错误的识别。

（2）语义分析输出的中间代码

- 输入的源程序（更多测试用例见源程序）

  Test1：

```
program sumary sum:integer;a:integer;
    begin
        begin
            a:=0
        end;
        if a<6 then
        begin
                while a<5 do
            begin
                sum:=sum+a;
                a:=a+1
            end
        end;
        begin
            a:=10
        end
    end
```

Test2：

```
program sumary arr:array [5] of integer;sum:integer;a:integer;
    begin
        begin
            a:=1;
            arr[a]:=2;
            arr[1]:=7;
            arr[2]:=4;
            arr[3]:=3;
            arr[4]:=2;
            a:=0
        end;
        while a<5 do
        begin
            sum:=sum+arr[a];
            a:=a+1
        end;
        begin
            a:=10
        end
    end
```

- 生成的中间代码
  （三地址码）和（四元式）

Test1：

三元组：

```
0: t0:=0
1: a:=t0
2: t1:=6
3: if a < t1 goto 5
4: goto 14
5: t2:=5
6: if a < t2 goto 8
7: goto 14
8: t3:=sum+a
9: sum:=t3
10: t4:=1
11: t5:=a+t4
12: a:=t5
13: goto 6
14: t6:=10
15: a:=t6
```

四元式：

```
0: := , 0 , null , t0
1: := , t0 , null , a
2: := , 6 , null , t1
3: < , a , t1 , 5
4: null , null , null , 14
5: := , 5 , null , t2
6: < , a , t2 , 8
7: null , null , null , 14
8: + , a , sum , t3
9: := , t3 , null , sum
10: := , 1 , null , t4
11: + , t4 , a , t5
12: := , t5 , null , a
13: null , null , null , 6
14: := , 10 , null , t6
15: := , t6 , null , a
```

Test2：

三元组：

```
0: t7:=1
1: a:=t7
2: t8:=arr
3: t9:=a
4: t10:=2
5: t8[t9]:=t10
6: t11:=1
7: t8:=arr
8: t12:=t11
9: t13:=7
10: t8[t12]:=t13
11: t14:=2
12: t8:=arr
13: t15:=t14
14: t16:=4
15: t8[t15]:=t16
16: t17:=3
17: t8:=arr
18: t18:=t17
19: t19:=3
20: t8[t18]:=t19
21: t20:=4
22: t8:=arr
23: t21:=t20
24: t22:=2
25: t8[t21]:=t22
26: t23:=0
27: a:=t23
28: t24:=5
29: if a < t24 goto 31
30: goto 40
31: t8:=arr
32: t25:=a
33: t26:=t8[t25]
34: t27:=sum+t26
35: sum:=t27
36: t28:=1
37: t29:=a+t28
38: a:=t29
39: goto 29
```

四元式：

```
0: := , 1 , null , t7
1: := , t7 , null , a
2: := , a , null , t9
3: := , a , null , t9
4: := , 2 , null , t10
5: []= , t10 , t9 , t8
6: := , 1 , null , t11
7: := , t11 , null , t12
8: := , t11 , null , t12
9: := , 7 , null , t13
10: []= , t13 , t12 , t8
11: := , 2 , null , t14
12: := , t14 , null , t15
13: := , t14 , null , t15
14: := , 4 , null , t16
15: []= , t16 , t15 , t8
16: := , 3 , null , t17
17: := , t17 , null , t18
18: := , t17 , null , t18
19: := , 3 , null , t19
20: []= , t19 , t18 , t8
21: := , 4 , null , t20
22: := , t20 , null , t21
23: := , t20 , null , t21
24: := , 2 , null , t22
25: []= , t22 , t21 , t8
26: := , 0 , null , t23
27: := , t23 , null , a
28: := , 5 , null , t24
29: < , a , t24 , 31
30: null , null , null , 40
31: := , a , null , t25
32: := , a , null , t25
33: =[] , t8 , t25 , t26
34: + , t26 , sum , t27
35: := , t27 , null , sum
36: := , 1 , null , t28
37: + , t28 , a , t29
38: := , t29 , null , a
39: null , null , null , 29
```

```
40: t30:=10
41: a:=t30
```

```
40: := , 10 , null , t30
41: := , t30 , null , a
```

（3）语义分析之后的符号表

Test1：

```
t4      INTEGER 24
a       INTEGER 4
t5      INTEGER 28
t6      INTEGER 32
sum     INTEGER 0
t0      INTEGER 8
t1      INTEGER 12
t2      INTEGER 16
t3      INTEGER 20
```

Test2：

```
t4      INTEGER 44
t5      INTEGER 48
t6      INTEGER 56
t7      INTEGER 60
t8      INTEGER 64
t9      INTEGER 72
sum     INTEGER 20
t10     INTEGER 76
t12     INTEGER 88
t11     INTEGER 80
t14     INTEGER 96
t13     INTEGER 92
t16     INTEGER 108
t15     INTEGER 104
t18     INTEGER 116
t17     INTEGER 112
t19     INTEGER 124
arr     ARRAY(5,INTEGER)        0
a       INTEGER 24
t21     INTEGER 132
t20     INTEGER 128
t23     INTEGER 140
t22     INTEGER 136
t0      INTEGER 28
t1      INTEGER 32
t2      INTEGER 32
t3      INTEGER 40
```

（4）语义错误报告

- 输入的有误文件：

Error1：

```
1  program sumary sum:integer;a:integer;a:integer;
2      begin
3          begin
4              a:=0
5          end;
6          while a<5 do
7          begin
8              sum:=sum+a;
9              a:=a+1
10         end
11     end
```

Error2：

```
1  program sumary sum:integer;
2      begin
3          begin
4              a:=0
5          end;
6          while a<5 do
7          begin
8              sum:=sum+a;
9              a:=a+1
10         end
11     end
```

- 错误信息的输出：

Error1：

```
Error at Line [1] :变量定义重复!
```

Error2：

```
Error at Line [4] :变量使用前未定义!
```

（5）实验结果的分析

通过观察两个两个错误文件，第一个文件的第一行，出现定义了两个 integer 类型的变量 a，因此出现了变量定义重复的错误；观察第二个错误文件第四行为变量 a 赋上了初值，但是在赋初值之前并没有定义变量 a，因此出现了变量使用前未定义的错误。而对于上面的正确的测试用例，不难发现，三地址码和四元式都是正确的，对于回填的语句标号也没有问题。

指导教师评语：

日期：