编译课系统程实验报告

实验 3：语义分析

姓名	杨富祥 王天一 白镇北	院系	计算机科学与技术	学号	1171800323 1170300220 1170301005
任课教师	陈鄞		指导教师	文荟俨	
实验地点	软件学院三楼		实验时间	2020-05-10　13: 00 – 15: 30	
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					

组内分工情况说明：

• 杨富祥：
翻译方案设计，核心数据结构设计，声明、表达式及赋值、分支语句翻译，报告撰写。

• 王天一：
错误处理，三地址指令转为四元式，循环、过程调用语句翻译，报告撰写。

• 白镇北：
语义分析信息输出 GUI。

小组群内讨论的截图：

一、需求分析	得分	

要求：阐述语义分析系统所要完成的功能。

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（包括变量声明、数组声明、记录声明和过程声明）
   - 表达式及赋值语句（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
   - 过程调用语句
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和四元式序列。
5. 额外功能，能实现自动类型转换，int 型与 float 型运算时，int 自动转换为 float 型；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（包括变量声明、数组声明、记录声明和过程声明）
- 表达式及赋值语句（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
- 过程调用语句

```
1. Program -> P
2. P -> PM D P | S P | ε
3. PM -> ε
   {offset = 0;}
4. D -> T id ;                          // 变量声明，数组声明
   {enter(id.lexeme, T.type, offset);
   offset = offset + T.width;}
   | struct id DM1 { P }                // 记录声明
   | proc X id DM2 ( M ) { P }          // 过程声明
5. DM1 -> ε
   {type = ‘record’;
   enterrecord(id.lexeme, type, offset);}
6. DM2 -> ε
   {type = ‘proc’;
   enterproc(id.lexeme, type, offset);}
7. T -> X TM C
   {T.type = C.type; T.width = C.width;}
8. TM -> ε
   {t = X.type; w = X.width;}
9. X -> int
   {X.type = int; X.width = 4;}
```

```
    | float
    {X.type = float; X.width = 8;}
    | char
    {X.type = char; X.width = 1;}
10. C -> [ num ] C
    {C.type = array(num.val, C1.type);
    C.width = num.val * C1.width;}
    | ε
    {C.type = t; C.width = w;}
11. M -> M , X id
    {enter(id.lexeme, X.type, offset);
    offset = offset + X.width;
    M.size = M1.size + 1;}
    | X id
    {enter(id.lexeme, X.type, offset);
    offset = offset + X.width;
    M.size = 1;}
12. S -> L = E ;                        // 数组元素引用或赋值
    {gen(L.array '[' L.offset ']' '=' E.addr);}
    | id = E ;
    {p = lookup(id.lexeme);
    if p == null then error
    else gen(p '=' E.addr);}
    | if ( B ) BM then S N else BM S        // 分支语句
    {backpatch(B.truelist, BM1.quad);
    backbatch(B.falselist, BM2.quad);
    temp = merge(S1.nextlist, N.nextlist);
    S.nextlist = merge(temp, S2.nextlist);}
    | while BM ( B ) do BM S                // 循环语句
    {backpatch(S1.nextlist, BM1.quad);
    Backpatch(B.truelist, BM2.quad);
    S.nextlist = B.falselist;
    gen('goto' BM1.quad);}
    | call id ( Elist ) ;                   // 过程调用语句
    {n = 0;
    for q 中的每个 t
    do {gen('param' t);
    n = n + 1;}
```

```
    gen('call' id.addr ',' n);}
    | return E ;
    {gen('return' E.addr);}
13. BM -> ε
    {BM.quad = nextquad;}
14. N -> ε
    {N.nextlist = makelist(nextquad);
    gen('goto');}
15. L -> 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);}
    | id [ E ]
    {p = lookup(id.lexeme);
    if p == null then error
    else L.array = p;
    L.type = p.type.elem;
    L.offset = newtemp();
    gen(L.offset '=' E.addr '*' L.type.width);}
16. E -> E + G
    {E.addr = newtemp();
    gen(E.addr '=' E1.addr '+' G.addr);}
    | G
    {E.addr = G.addr;}
17. G -> G * F
    {G.addr = newtemp;
    gen(G.addr '=' G1.addr '*' F.addr);}
    | F
    {G.addr = F.addr;}
18. F -> ( E )
    {F.addr = E.addr;}
    | num
    {F.addr = num.val;}
    | id
    {F.addr = looup(id.lexeme);
```

```
    if F.addr == null then error;}
    | real
    {F.addr = real.val;}
    | character
    {F.addr = character.val;}
    | L
    {F.addr = L.array '[' L.offset ']';}
19. B -> B || BM H
    {B.truelist = merge(B1.truelist, H.truelist);
    B.falselist = H.falselist;
    backpatch(B1.falselist, BM.quad);}
    | H
    {B.truelist = H.truelist;
    B.falselist = H.falselist;}
20. H -> H && BM I
    {H.truelist = I.truelist;
    H.falselist = merge(H1.falselist, I.falselist);
    backpatch(H1.truelist, BM.quad);}
    | I
    {H.truelist = I.truelist;
    H.falselist = I.falselist;}
21. I -> ! I
    {I.truelist = I1.falselist;
    I.falselist = I1.truelist;}
    | ( B )
    {I.truelist = B.truelist;
    I.falselist = B.falselist;}
    | E relop E
    {I.truelist = makelist(nextquad);
    I.falselist = mekelist(nextquad + 1);
    gen('if' E1.addr relop E2.addr 'goto');
    gen('goto')}
    | true
    {I.truelist = mekelist(nextquad);
    gen('goto')}
    | false
    {I.falselist = makelist(nextquad);
    gen('goto')}
```

```
22. relop -> < | > | <= | >= | != | ==
23. Elist -> Elist , E
    {将 E.addr 添加到 q 的队尾; }
    | E
    {将 q 初始化为只包含 E.addr; }
```

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。

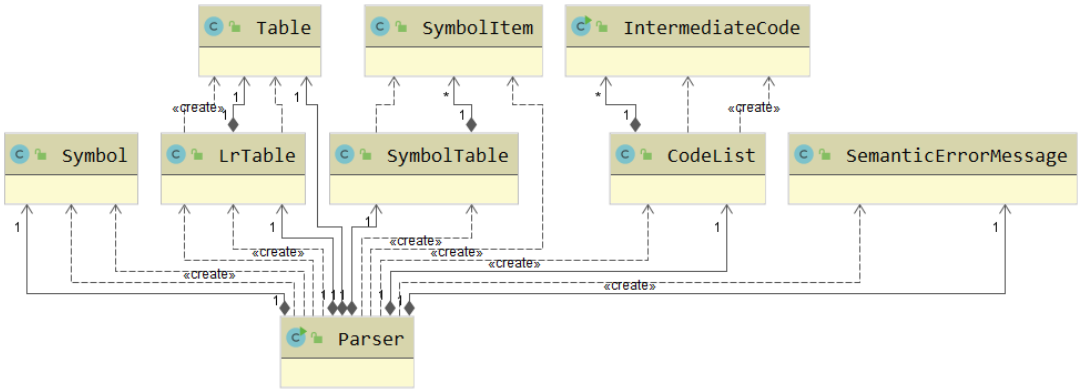
（2）系统详细设计：对如下工作进行展开描述

✓ 核心数据结构的设计

✓ 主要功能函数说明

✓ 程序核心部分的程序流程图

（1） UML 图如下所示：



Symbol 代表文法符号栈中元素。

Table 与 LrTable 构建 LR(1)分析表。

SymbolItem 代表符号表表项，SymbolTable 为符号表。

IntermediateCode 代表一条中间代码，CodeList 为中间代码集合。

SemanticErrorMessag 表示语义分析错误信息。

Parser 为语法分析器主类，在内部规约动作时执行语义动作。

（2） 系统详细设计

• 核心数据结构

语法分析阶段维护两个栈：状态栈和符号栈，符号栈中元素就是 String 类型的文法符号。而进入语义分析阶段，需要扩展语法分析栈。我们要为文法符号增加若干属性，如文法符号

“id”，需要为其增加“lexeme”、“lineNum”等属性。故使用 Symbol 这个数据结构作为符号栈中元素，Symbol 中属性字段为：

```
public class Symbol {
    private final String name;
    private final Map<String, String> attributes = new HashMap<>();
    private final Map<String, List<Integer>> listMap = new HashMap<>();
```

其中 name 是文法符号名，attributes 为属性的类型与值的映射关系，listMap 代表控制语句翻译方案中涉及的“nextlist”、“truelist”、“falselist”等概念。

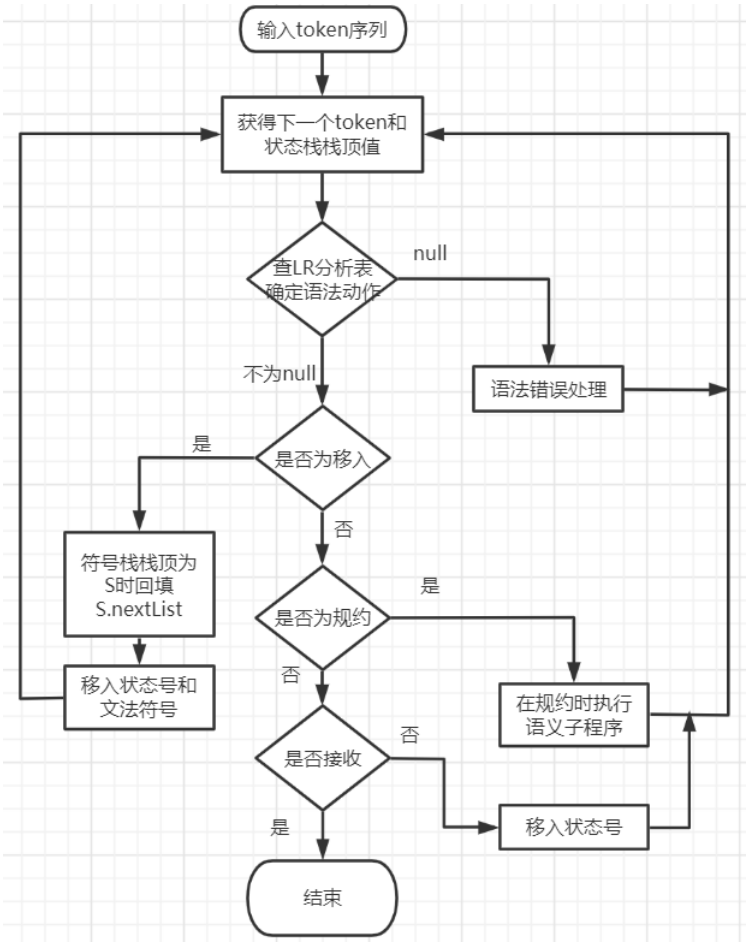
符号表 SymbolTable 类中使用 Map 记录标识符和其对应的符号表条目。

• 主要函数说明

在 Parser 类中，
handle()函数模拟自动机，对两个栈进行操作。
reduce()函数在规约的时候执行相关语义动作，逐步构建符号表、生成中间代码并记录错误信息。

在 IntermediateCode 类中，将三地址指令转变为四元式序列。

• 核心部分流程图



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；

（2） 针对一测试程序输出其语义分析结果；

（3） 输出针对此测试程序经过语义分析后的符号表；

（4） 输出针对此测试程序对应的语义错误报告；

（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。

（1）遇到的问题

①、翻译方案设计比较困难，由于是 LR 分析法，要给出 S-属性定义，从 L-属性定义到 S-属性定义的转换比较麻烦。本文设计的翻译方案并不完全是 S-属性定义，但在工程上能够实现，可以通过使用一些全局变量、直接查找栈中元素对应属性等手段来解决。

②、在错误处理中，为了比较和记录错误信息，我们往需要的文法的终结符和非终结符中，添加了翻译方案以外的一些属性，包括每一条的行号，每一个非终结符或者终结符的类型。

（2）测试用例：

```
struct student{
    int id ;
    char name ;
}

int [2][3] array;
array[0][1] = 2;
int temp;
temp = array[0][1];

int a;
a = 0xf;
int b;  int c;  int d;
int e;  int f;
int x;  int y;  int z;

while (a < b)
    do
    if (c < 5) then
        while (x > y)
        do
            z = x + 1;
    else
        x = y;
```

```
a = 1 ;
if(a < b || c < d && e < f) then
    x = a;
else
    x = b;
x = 1 ;
proc int getSum(int elem1, int elem2){
        int sum;
        sum = elem1 + elem2;
        return sum;
}

call getSum(a,b);
```

（3）符号表：

```
符号表：
id-type-offset-lineNum
<student, record, 0, 1>
<id, int, 0, 2>
<name, char, 4, 3>
<array, array(2, array(3, int)), 5, 6>
<temp, int, 29, 8>
<a, int, 33, 11>
<b, int, 37, 13>
<c, int, 41, 13>
<d, int, 45, 13>
<e, int, 49, 14>
<f, int, 53, 14>
<x, int, 57, 15>
<y, int, 61, 15>
<z, int, 65, 15>
<getSum, proc, 69, 32>
<elem1, int, 69, 32>
<elem2, int, 73, 32>
<sum, int, 77, 33>
```

（4）中间代码：

```
中间代码：
0: t1 = 0 * 12
1: t2 = 1 * 4
2: t3 = t1 + t2
3: array [ t3 ] = 2
4: t4 = 0 * 12
5: t5 = 1 * 4
6: t6 = t4 + t5
7: temp = array[t6]
8: a = 15
9: if a < b goto 11
10: goto 21
11: if c < 5 goto 13
12: goto 19
13: if x > y goto 15
14: goto 9
15: t7 = x + 1
16: z = t7
17: goto 13
18: goto 9
19: x = y
20: goto 9
21: a = 1
22: if a < b goto 28
23: goto 24
24: if c < d goto 26
25: goto 30
26: if e < f goto 28
27: goto 30
28: x = a
29: goto 31
30: x = b
31: x = 1
32: t8 = elem1 + elem2
33: sum = t8
34: return sum
35: param a
36: param b
37: call getSum , 2
```

（5）四元式序列

```
四元式序列:
0: (*, 0, 12, t1)
1: (*, 1, 4, t2)
2: (+, t1, t2, t3)
3: ([]=, 2, array, t3)
4: (*, 0, 12, t4)
5: (*, 1, 4, t5)
6: (+, t4, t5, t6)
7: (=[], array, t6, temp)
8: (=, 15, _, a)
9: (j<, a, b, 11)
10: (j, _, _, 21)
11: (j<, c, 5, 13)
12: (j, _, _, 19)
13: (j>, x, y, 15)
14: (j, _, _, 9)
15: (+, x, 1, t7)
16: (=, t7, _, z)
17: (j, _, _, 13)
18: (j, _, _, 9)
19: (=, y, _, x)
20: (j, _, _, 9)
21: (=, 1, _, a)
22: (j<, a, b, 28)
23: (j, _, _, 24)
24: (j<, c, d, 26)
25: (j, _, _, 30)
26: (j<, e, f, 28)
27: (j, _, _, 30)
28: (=, a, _, x)
29: (j, _, _, 31)
30: (=, b, _, x)
31: (=, 1, _, x)
32: (+, elem1, elem2, t8)
33: (=, t8, _, sum)
34: (return, sum, _, _)
35: (param, a, _, _)
36: (param, b, _, _)
37: (call, getSum, 2, _)
```

（6）错误测试用例：

```
struct student{
    int id ;
    char name ;
}
struct student{
}

int [2][3] array;
array[0][1.88] = 2;
int temp;
temp = array[0][1];

int a;
int b ;
a = 5 * 'c' ;
a[2] = 6 ;
g = 5;
a = 0xf;
int b;
int c;
int d;
int x;
int y;
int z;

while (a < b)
do if (c < 5) then
   while (x > y) do z = x + 1;
else
   x = y;
```

```
a = 1;
if (a > b) then
    x = a;
else
    x = b;

proc int getSum(int elem1, int elem2){
        int sum;
        sum = elem1 + elem2;
        return sum;
}

int e;
int f;

if(a < b || c < d && e < f) then
    x = a;
else
    x = b;
a = 5;

proc int getSum(int elem3, int elem4){
}

call getsum(a,b) ;
call f( a ,b) ;
```

（7）错误信息报告：

语义分析错误信息：

Error at line[5]: 重复的记录声明student

Error at line[9]: 数组下标不是整数: 1.88

Error at line[15]: 运算符与运算分量不匹配: 5 * 'c'

Error at line[16]: 非数组类型变量使用了数组操作: a

Error at line[17]: 未经声明就使用的变量: g

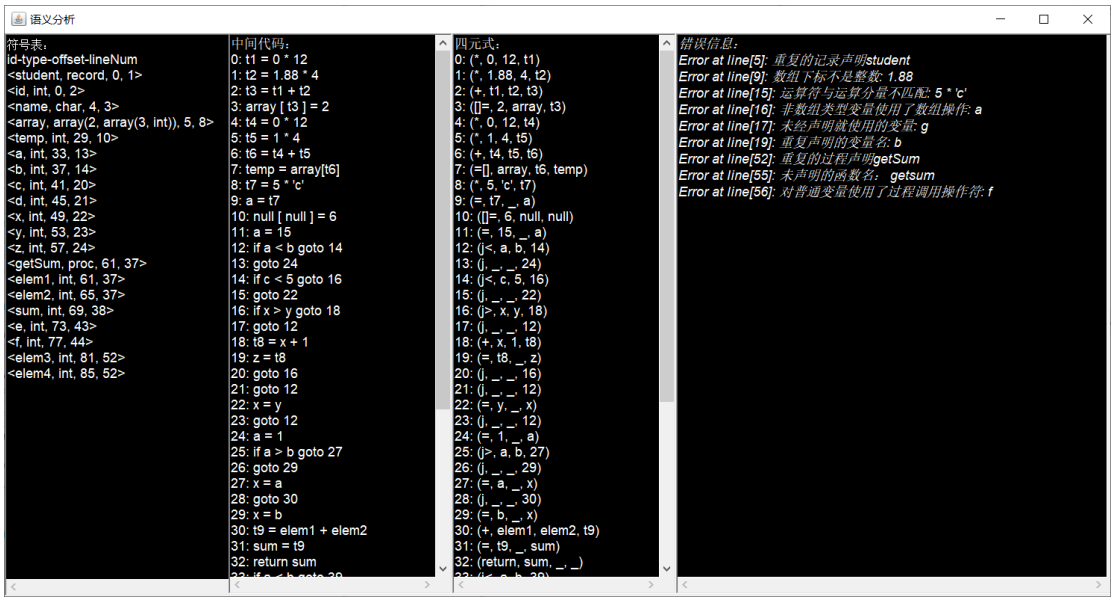
Error at line[19]: 重复声明的变量名: b

Error at line[52]: 重复的过程声明getSum

Error at line[55]: 未声明的函数名： getsum

Error at line[56]: 对普通变量使用了过程调用操作符: f

（8）程序运行界面：


语义分析
符号表：
id-type-offset-lineNum
<student, record, 0, 1>
<id, int, 0, 2>
<name, char, 4, 3>
<array, array(2, array(3, int)), 5, 8>
<temp, int, 29, 10>
<a, int, 33, 13>
<b, int, 37, 14>
<c, int, 41, 20>
<d, int, 45, 21>
<x, int, 49, 22>
<y, int, 53, 23>
<z, int, 57, 24>
<getSum, proc, 61, 37>
<elem1, int, 61, 37>
<elem2, int, 65, 37>
<sum, int, 69, 38>
<e, int, 73, 43>
<f, int, 77, 44>
<elem3, int, 81, 52>
<elem4, int, 85, 52>
中间代码：
0: t1 = 0 * 12
1: t2 = 1.88 * 4
2: t3 = t1 + t2
3: array [t3] = 2
4: t4 = 0 * 12
5: t5 = 1 * 4
6: t6 = t4 + t5
7: temp = array[t6]
8: t7 = 5 * 'c'
9: a = t7
10: null [null] = 6
11: a = 15
12: if a < b goto 14
13: goto 24
14: if c < 5 goto 16
15: goto 22
16: if x > y goto 18
17: goto 12
18: t8 = x + 1
19: z = t8
20: goto 16
21: goto 12
22: x = y
23: goto 12
24: a = 1
25: if a > b goto 27
26: goto 29
27: x = a
28: goto 30
29: x = b
30: t9 = elem1 + elem2
31: sum = t9
32: return sum
四元式：
0: (*, 0, 12, t1)
1: (*, 1.88, 4, t2)
2: (+, t1, t2, t3)
3: ([]=, 2, array, t3)
4: (*, 0, 12, t4)
5: (*, 1, 4, t5)
6: (+, t4, t5, t6)
7: (=[], array, t6, temp)
8: (*, 5, 'c', t7)
9: (=, t7, _, a)
10: ([]=, 6, null, null)
11: (=, 15, _, a)
12: (j<, a, b, 14)
13: (j, _, _, 24)
14: (j<, c, 5, 16)
15: (j, _, _, 22)
16: (j>, x, y, 18)
17: (j, _, _, 12)
18: (+, x, 1, t8)
19: (=, t8, _, z)
20: (j, _, _, 16)
21: (j, _, _, 12)
22: (=, y, _, x)
23: (j, _, _, 12)
24: (=, 1, _, a)
25: (j>, a, b, 27)
26: (j, _, _, 29)
27: (=, a, _, x)
28: (j, _, _, 30)
29: (=, b, _, x)
30: (+, elem1, elem2, t9)
31: (=, t9, _, sum)
32: (return, sum, _, _)
错误信息：
Error at line[5]: 重复的记录声明student
Error at line[9]: 数组下标不是整数: 1.88
Error at line[15]: 运算符与运算分量不匹配: 5 * 'c'
Error at line[16]: 非数组类型变量使用了数组操作: a
Error at line[17]: 未经声明就使用的变量: g
Error at line[19]: 重复声明的变量名: b
Error at line[52]: 重复的过程声明getSum
Error at line[55]: 未声明的函数名： getsum
Error at line[56]: 对普通变量使用了过程调用操作符: f

指导教师评语：

日期：