编译原理课程实验报告

实验 3：语义分析

姓名	朱明彦	院系	计算机学院	学号	1160300314
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	2019/04/27	
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。

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. 额外功能，能实现自动类型转换，将 int 型和 float 型运算的时，将 int 型自动转化为 Float 型；对非数组类型变量进行访问数组引用出错。

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

```
Start -> P

P -> PStart D P | PStart S P | ε

PStart -> ε {{ env = new Env(env); offsetStack.push(offset); offset=0;}}

D -> proc X id ( M ) DM { P } {{pop(tableStack); pop(offset)}} | record id
{ P } | T id A ; {{enter(id.lexeme, T.type, offset);offset = offset +
T.width;}}

DM -> ε {{table = mkTable(top(tableStack)); push(table); push(offset);
offset = 0;}}

A -> = F A | , id A | ε

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;}}

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

X -> int {{X.type = interger; X.width = 4;}} | float {{X.type = float;
X.width = 8;}} | bool | char

C -> [ num ] C {{C.type = C1.type + '[' + num.value + ']'; C.width = num.value
* C1.width;}} | ε {{C.type = t; C.width = w;}}

S -> id = E ; {{S.nextList = null; p = loopUp(id.lexeme); if p == null then
error else gen(p, '=', E.addr);}} | if ( B ) BM S N else BM
S {{backpatch(B.trueList, BM1.instr); backpatch(B.falseList, BM2.instr);
temp = merge(S1.nextList, N.nextList); S.nextList = merge(temp,
S2.nextList); }} | while BM ( B ) BM S {{backpatch(S1.nextList, BM1.instr);
backpatch(B.trueList, BM2.instr); S.nextList = B.falseList; gen('goto',
BM1.instr); }} | call id ( Elist ) ; | return E ; | if ( B ) BM
S {{backpatch(B.trueList, BM.instr); S.nextList = merge(B.falseList,
S1.nextList); }} | L = E ; {{gen(L.array, L.addr, '=', E.addr)}}

N -> ε {{N.nextList = makeList(nextInstr); gen('goto'); }}

L -> L [ E ] {{L.array = L1.array; L.type = L1.type.elem; L.width =
L.type.width; t = new Temp(); L.addr = new Temp(); gen(L.addr, '=', E.addr,
'*', L.width); gen(L.addr, '=', L1.addr, '+', t); }} | id [ E ] {{p =
lookUp(id.lexeme); if p == null then error else L.array = p; L.type = id.type;
L.addr = new Temp(); gen(L.addr, 'addr', E.addr, '*', L.width)}}
```

```
E -> E + G {{E.addr = newTemp(); gen(E.addr, '=', E1.addr, '+', G.addr);}}
| G {{E.addr = G.addr;}}

G -> G * F {{G.addr = newTemp(); gen(G.addr, '=', G1.addr, '*', F.addr);}}
| F {{G.addr = F.addr;}}

F -> ( E ) {{F.addr = E.addr;}}| num {{F.addr = num.value;}}| id {{F.addr
= lookup(id.lexeme); if F.addr == null then error;}}| real {{F.addr =
real.value;}}| string | L {{F.addr = L.array + '[' + L.addr']'}}

B -> B || BM H {{backpatch(B1.falseList, BM.instr); B.trueList =
merge(B1.trueList, H.trueList); B.falstList = H.falstList;}}|
H {{B.trueList = H.trueList; B.falseList = H.falseList;}}

H -> H && BM I {{backpatch(H1.trueList, BM.instr); H.trueList = I.trueList;
H.falseList = merge(H1.falseList, I.falseList);}}| I {{H.trueList =
I.trueList; H.falseList = I.falseList;}}

I -> ! I {{I.trueList = I1.falseList; I.falseList = I1.falseList;}}|
( B ) {{I.trueList = B.trueList; I.falseList = B.falseList;}}| E Relop
E {{I.trueList = makeList(nextInstr); I.falseList = makeList(nextInstr +
1); gen('if', E1.addr, Relop.op, E2.addr, 'goto'); gen('goto');}} |
true {{I.trueList = makeList(nextInstr); gen('goto');}}|
false {{I.falseList = makeList(nextInstr); gen('goto');}}

BM -> ε {{BM.instr = nextInstr}}

Relop -> < | <= | > | >= | == | != {{Relop.op = op}}

Elist -> Elist , E {{Elist.size = Elist1.size + 1;}} | E {{Elist.size = 1;}}
```

三、系统设计	得分	

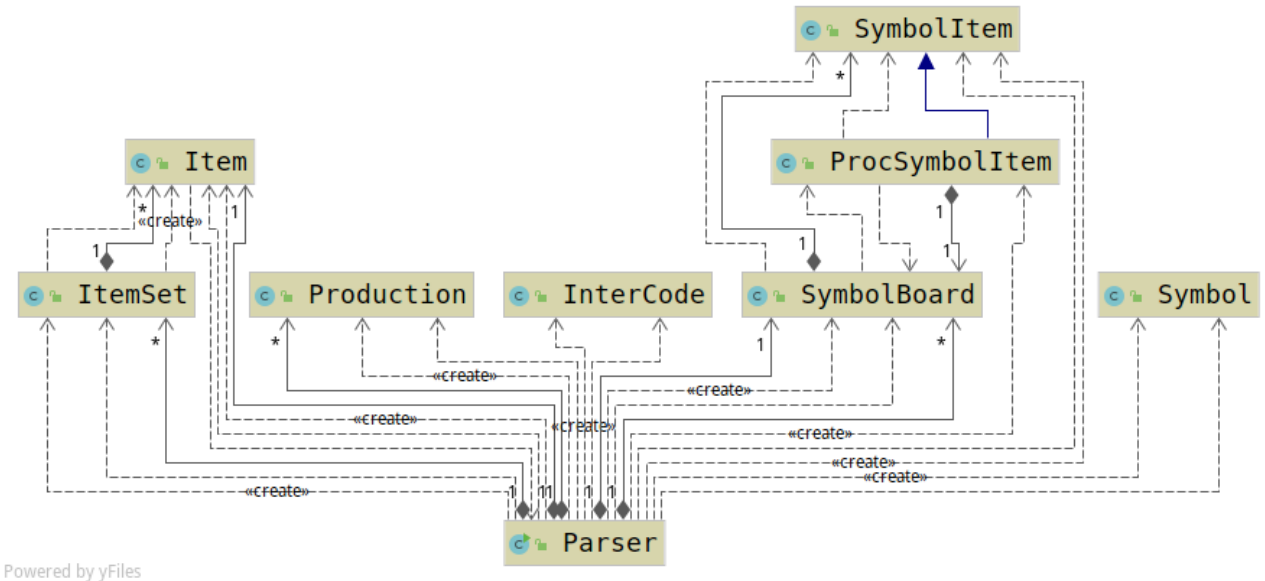
要求：分为系统概要设计和系统详细设计。

（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）系统概要设计



语义分析器的 UML 图如上所示。

其中，parser 为语法分析器的主类，类似之前的语法分析；SymbolBoard 为符号表类，其中主要的符号表条目为 SymbolItem，为一般标识符的符号表条目；

另一种符号表条目为 ProcSymbolItem，其为函数声明的符号表条目，在其中记录着函数用于记录局部变量的符号表；InterCode 为中间代码类，用于语义动作中生成局部代码的操作。

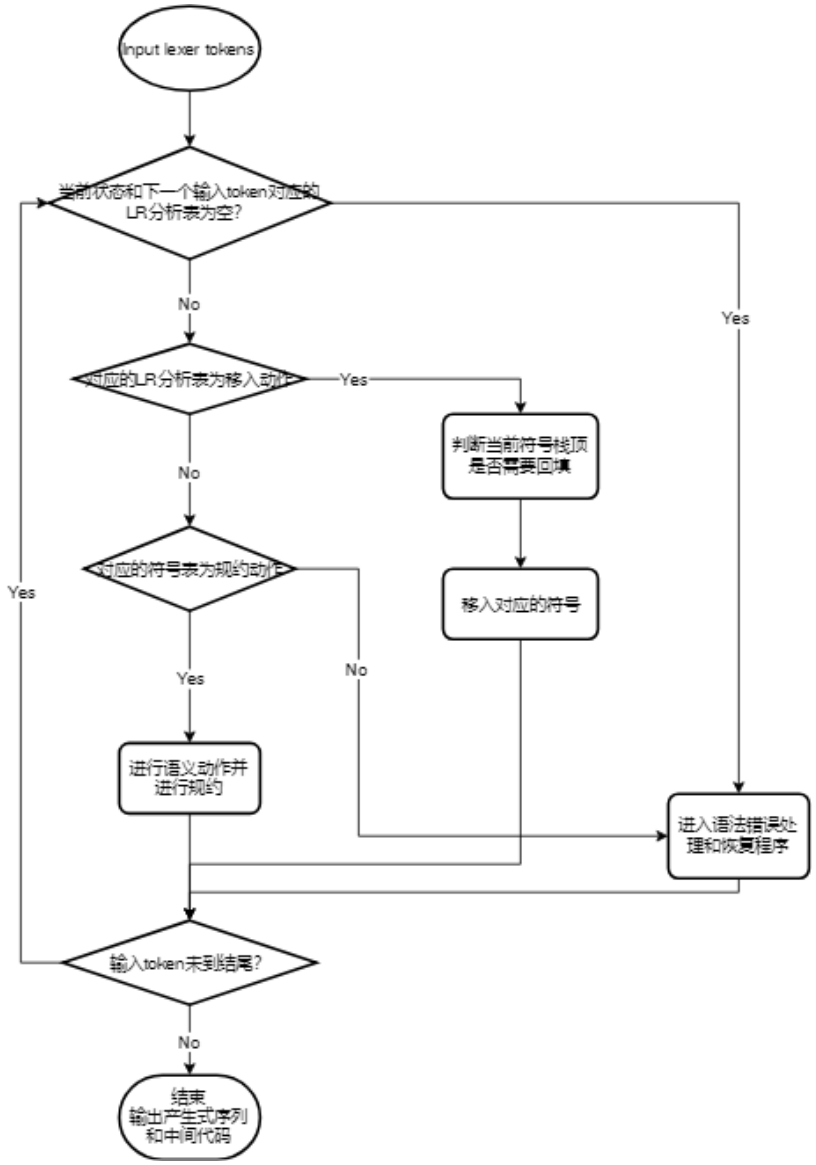
（2）系统详细设计

a. 核心数据结构，在符号表中使用 Map 记录标识符与其对应的符号表条目；在 Parser 中使用栈记录调用关系，将 offset 和之前的符号表在声明函数时进行压栈处理。

b. 主要函数说明，在语义分析部分，由于使用的是仅有 S 属性的自底向上翻译方案，所以在规约之后进行语义动作即可，基于此所有的语义动作都定义在 reduce 函数中进行。

reduce 函数的功能为进行 LR(1)语法分析，规约出产生式，并在有相应语义动作的规约之后进行，并进行语法的错误处理和恢复，以及语义的错误提示。

c. reduce 函数流程图如下



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) 嵌套符号表的记录，对于子过程而言，其可以在压栈时简单处理即可记录父过程的符号表；而对于子过程的符号表，则需要在父过程的符号表中显式记录子过程的名字和符号表。因此，修改了最初的 `SymbolItem` 设计，新增了 `ProcSymbolItem`。

(2) 语义分析结果

测试使用的源代码如下：

```
int a;
a = 1 + 2;
int b;
int c;
c = 10;
b = c + 1;
int d;
d = c * 2;
int e;
e = 0;
int x;
int y;
y = 999;
int z;
z = 100;
while (a < b)
    if (c < d) x = y + z; else x = a + b;

a = b + c * (d + e);

if(a > b)
    c = d;
```

```
proc float function(float i){
    i = i + 1;
    return i;
}

int [2][3] list;
int c;
int i;
int j;
int d;
float h;
d = c + list[i][j];
list[i][j] = c;

d = h * c;

c[1][2] = d;
proc int function(int a, int c){
    a = c + 10;
    int d;
    return a;
}
```

```
0 : t1 = 1 + 2
1 : a = t1
2 : c = 10
3 : t2 = c + 1
4 : b = t2
5 : t3 = c * 2
6 : d = t3
7 : e = 0
8 : y = 999
9 : z = 100
10 : if a < b goto 12
11 : goto 20
12 : if c < d goto 14
13 : goto 17
14 : t4 = y + z
15 : x = t4
16 : goto 10
17 : t5 = a + b
18 : x = t5
19 : goto 10
20 : t6 = d + e
21 : t7 = c * t6
22 : t8 = b + t7
23 : a = t8
24 : if a > b goto 26
25 : goto 27
26 : c = d
27 : t9 = i + 1
28 : i = t9
29 : t10 = i * 12
30 : t11 = j * 4
31 : t12 = t10 + t11
32 : t13 = c + list[t12]
33 : d = t13
34 : t14 = i * 12
35 : t15 = j * 4
36 : t16 = t14 + t15
37 : list [ t16 ] = c
38 : t17 = h * (float) c
39 : d = t17
40 : t18 = c + 10
41 : a = t18
```

（3）符号表

```
<a, int, 1, 0>
<b, int, 3, 4>
<c, int, 4, 8>
<d, int, 7, 12>
<e, int, 9, 16>
<h, float, 34, 64>
<i, int, 31, 56>
<j, int, 32, 60>
<list, int[2][3], 29, 32>
<function, proc, 41, 72>
function Table:
{
<a, int, 41, 0>
<c, int, 41, 4>
<d, int, 43, 8>
}
<x, int, 11, 20>
<y, int, 12, 24>
<z, int, 14, 28>
```

(4) 错误报告

Error at line[30], c is defined early.

Error at line[33], d is defined early.

Error at line[40], c type error

Error at line[41], proc defined again

（5）实验结果分析，源代码中包含 while 循环、if-else 分支、if 分支、函数声明以及数组元素的使用。并设置了若干错误，变量的重复声明、函数的重复声明以及对非数组元素的下标引用等错误，以及在 int 和 float 类型之间的运算。

最终的中间代码中，也实现了相应的代码回填。符号表中实现了对嵌套的符号表的访问和打印。

指导教师评语：

日期：