编译原理课程实验报告

实验 3：语义分析

姓名	席凯迪	院系	软件学院	学号	161110507
任课教师	韩希先		指导教师	韩希先	
实验地点	研究院中 507		实验时间	2018.11.10	
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、实验目的

1. 巩固对语义分析的基本功能和原理的认识
2. 能够基于语法指导翻译的知识进行语义分析
3. 掌握类高级语言中基本语句所对应的语义动作
4. 理解并处理语义分析中的异常和错误

掌握语法指导翻译的基本功能并将其实现，使得程序能够给出各语句的语义动作，并尽可能处理符号表中的信息。

二、实验内容

1.文法描述和对应的语义动作

S->E print(E.val)
E->E+T E.val=E1.val+T.val
E->T E.val=T.val
T->T*F T.val=T1.val*F.val
T->F T.val=F.val
F->(E) F.val=E.val
F->d F.val=d.lexval

2. 语义动作具体加入所涉及的数据结构及其物理实现

数据结构：一维 String 数组

物理实现：

```
private String []semantic= {"print(E.val)","E.val=E1.val+T.val","E.val=T.val",
"T.val=T1.val*F.val","T.val=F.val","F.val=E.val","F.val=d.lexval"};
```

3. 语义属性的分析、设计和实现

语义属性是根据文法得出的，由于只含有赋值运算和算术运算，所以父节点的属性值都是由子节点得属性值得到的，根据属性定义，此文法只有综合属性。非终结符得值都由非终结符加点加 val 组成，此文法中表示为 E.val, T.val, F.val, 此文法中的终结符 d 的属性值表示为 d.lexval。

4.语义分析的实现：

进行语义分析前的准备工作：

```
public void Get_Run() {
    String getstr="d*(d+d)+(d+d)#";          //语法分析
    String numstr="1*(2+3)+(4+5)#";
    Stack<Integer> stack = new Stack<Integer>(); //状态栈
    Stack<Character> stackSymbol = new Stack<Character>();    //符号栈
    Stack<String> stacknum=new Stack<String>();      //数字栈，用来存储进行运算的数字
    stack.push(0);
    stackSymbol.push('#');
    stacknum.push("#");
    System.out.printf("%-20s ","状态栈");       //输出表头
    System.out.print("                                        ");
    System.out.printf("%-20s ","符号栈");
    System.out.print("                                        ");
    System.out.printf("%-20s ","剩余字符串");
    System.out.print("                                        ");
    System.out.printf("%-20s ","动作");
    System.out.print("                                        ");
    System.out.printf("%-20s ","语义动作");
    System.out.print("                                        ");
    System.out.printf("%-20s ","数字栈");
    System.out.print("                                  ");
    System.out.printf("%-20s ","中间代码生成");
    System.out.println();
    int i=0,j,k,t;
    while(true) {
        int s=stack.peek();                    //peek方法，查看栈顶对象而不移除它
        if(ACTION[s][IsInVt(getstr.charAt(i))].charAt(0)=='s') {
            System.out.printf("%-20s ",Print_Stack(stack));       //输出内容
            System.out.print("          ");
            System.out.printf("%-20s ",Print_stackSymbol(stackSymbol));//输出状态栈
            System.out.print("          ");
            System.out.printf("%-20s ",Print_Buffer(getstr,i));


            System.out.print("          ");
            System.out.printf("%-20s ",Print_Buffer(getstr,i));
            System.out.print("          ");
            System.out.printf("%-20s ","移入");
            System.out.print("                                        ");            //对应语义动作那一列
            System.out.printf("%-20s ","移入");
            System.out.print("                                        ");
            System.out.printf("%-20s ",Print_Stacknum(stacknum));          //将对应数字移入
            System.out.println();
            stackSymbol.push(getstr.charAt(i));//将一个字符压入栈中
            stacknum.push(String.valueOf(numstr.charAt(i)));            //char  转成String
            stack.push(Integer.parseInt(ACTION[s][IsInVt(getstr.charAt(i))].substring(1,
                    ACTION[s][IsInVt(getstr.charAt(i))].length())));//状态入栈
            i++;
        } else {
            if(ACTION[s][IsInVt(getstr.charAt(i))].charAt(0)=='r') {
                System.out.printf("%-20s ",Print_Stack(stack));
                System.out.print("          ");
                System.out.printf("%-20s ",Print_stackSymbol(stackSymbol));
                System.out.print("          ");
                System.out.printf("%-20s ",Print_Buffer(getstr,i));
                System.out.print("          ");
                j=Integer.parseInt(ACTION[s][IsInVt(getstr.charAt(i))].substring(1,
                        ACTION[s][IsInVt(getstr.charAt(i))].length()));
                System.out.printf("%-20s ","根据"+wenfa.get(j)+"归约");
                System.out.print("          ");
                System.out.printf("%-20s ","执行"+semantic[j]+"动作");            //语义动作
                System.out.print("                                        ");
                System.out.printf("%-20s ",Print_Stacknum(stacknum));               //数字栈
```

核心代码：

```
  switch(j)
  {
  case 0:
     System.out.printf("%-30s ","计算的最终结果是"+Print_Stacknum(stacknum));
     break;
  case 1:
          E1_val=Integer.parseInt(stacknum.pop());      //弹出栈顶的两个值进行运算
          stacknum.pop();     //弹出+
          T_val=Integer.parseInt(stacknum.pop());
          E_val = E1_val + T_val;
          String str = String.valueOf(E_val);    //int转成string
          stacknum.push(str);    //int 转成 char    存入数字栈中；将运算结果存入数字栈中
          System.out.printf("%-20s "," E_val = E1_val + T_val; 操作"+
          " E_val = "+E_val);
          break;
  case 2:
     T_val=Integer.parseInt(stacknum.peek()); ;     //取出栈顶元素
     E_val=T_val;  //赋值
     System.out.printf("%-20s ",Print_Stacknum(stacknum)+
             "  E_val =T_val 操作"+" E_val =" + E_val);
      break;
  case 3:
          F_val=Integer.parseInt(stacknum.pop()); ;      //弹出栈顶的两个值进行运算
          stacknum.pop();     //弹出*
          T1_val=Integer.parseInt(stacknum.pop());
          T_val=T1_val*F_val;
          String str1 = String.valueOf(T_val);    //int转成string
          stacknum.push(str1);    //int 转成 char    存入数字栈中；将运算结果存入数字栈中
          System.out.printf("%-20s "," T_val=T1_val*F_val操作，T_val="+T_val);
           break;
  case 4:
     F_val=Integer.parseInt(stacknum.peek());  //取出栈顶元素
     T_val=F_val;  //赋值
     System.out.printf("%-20s ","T_val=F_val; 操作"+" T_val =" + T_val);
     break;
  case 5:

     stacknum.pop();    //弹出（括号
     E_val=Integer.parseInt(stacknum.pop());    //取出栈顶元素
     F_val=E_val;  //赋值
     stacknum.pop();    //弹出右括号

      String str3 = String.valueOf(F_val);    //int转成string
      stacknum.push(str3);    //int 转成 char    存入数字栈中；将运算结果存入数字栈中
     System.out.printf("%-20s ","F_val=E_val 操作"+" F_val =" + F_val);
     break;
  case 6:
     d=Integer.parseInt(stacknum.peek());      //取出栈顶元素
     F_val=d;  //赋值
     System.out.printf("%-20s ","F_val=d操作"+" F_val =" + F_val);
     break;
  }
 k=Get_Number(j);//判断出栈个数
 while(k>0) {
     stack.pop();
     stackSymbol.pop();
     k--;
 }
 stackSymbol.push(wenfa.get(j).charAt(0));//每次规约后都会有goto
 t=stack.peek();
 stack.push(GOTO[t][IsInTotal(wenfa.get(j).charAt(0))-terminal.length()]);
 System.out.println();
```

输出结果：

```
    } else {
        if(ACTION[s][IsInVt(getstr.charAt(i))].equals("acc")) {
            System.out.printf("%-20s",Print_Stack(stack));
            System.out.print("          ");
            System.out.printf("%-20s",Print_stackSymbol(stackSymbol));
            System.out.print("          ");
            System.out.printf("%-20s",Print_Buffer(getstr,i));
            System.out.print("                ");
            System.out.print("接受");       //根据wenfa.get(0)规约，此时变成接受
            System.out.print("                                    ");
            System.out.print("执行print(E.val)动作");
            System.out.print("                                        ");
            System.out.printf("%-20s",Print_Stacknum(stacknum));
            System.out.println();
            break;
        } else {
            System.out.println("ERROR");
            break;
        }
    }
}
```

5.符号表的自动构造：

SLR(1)分析表:

	+	*	(	)	d	#	S	E	T	F
0			s4		s5			a1	a2	a3
1	s6					acc				
2	r2	s7		r2		r2				
3	r4	r4		r4		r4				
4			s4		s5			a8	a2	a3
5	r6	r6		r6		r6				
6			s4		s5				a9	a3
7			s4		s5					a10
8	s6			s11						
9	r1	s7		r1		r1				
10	r3	r3		r3		r3				
11	r5	r5		r5		r5				

三、实验结果

根据规定好的句子 d*(d+d)+(d+d)和表达式 1*(2+3)+(4+5)，对应的分析和语义动作如下图所示：

```
根据F->d归约          执行F.val=d.lexval动作          #1              F_val=d操作 F_val =1
根据T->F归约          执行T.val=F.val动作           #1             T_val=F_val; 操作 T_val =1
移入                        移入                  #1
移入                        移入                  #1*
移入                        移入                  #1*(
根据F->d归约          执行F.val=d.lexval动作          #1*(2           F_val=d操作 F_val =2
根据T->F归约          执行T.val=F.val动作           #1*(2          T_val=F_val; 操作 T_val =2
根据E->T归约          执行E.val=T.val动作           #1*(2          #1*(2  E_val =T_val 操作 E_val =2
移入                        移入                  #1*(2
移入                        移入                  #1*(2+
 根据F->d归约          执行F.val=d.lexval动作          #1*(2+3          F_val=d操作 F_val =3
 根据T->F归约          执行T.val=F.val动作           #1*(2+3         T_val=F_val; 操作 T_val =3
 根据E->E+T归约         执行E.val=E1.val+T.val动作          #1*(2+3          E_val = E1_val + T_val; 操作 E_val = 5
移入                        移入                  #1*(5
根据F->(E)归约        执行F.val=E.val动作           #1*(5)          F_val=E_val 操作 F_val =5
根据T->T*F归约        执行T.val=T1.val*F.val动作          #1*5          T_val=T1_val*F_val操作，T_val=5
根据E->T归约          执行E.val=T.val动作           #5             #5  E_val =T_val 操作 E_val =5
移入                        移入                  #5
移入                        移入                  #5+
移入                        移入                  #5+(
根据F->d归约          执行F.val=d.lexval动作          #5+(4           F_val=d操作 F_val =4
根据T->F归约          执行T.val=F.val动作           #5+(4          T_val=F_val; 操作 T_val =4
根据E->T归约          执行E.val=T.val动作           #5+(4          #5+(4  E_val =T_val 操作 E_val =4
移入                        移入                  #5+(4
移入                        移入                  #5+(4+
 根据F->d归约          执行F.val=d.lexval动作          #5+(4+5          F_val=d操作 F_val =5
 根据T->F归约          执行T.val=F.val动作           #5+(4+5         T_val=F_val; 操作 T_val =5
 根据E->E+T归约         执行E.val=E1.val+T.val动作          #5+(4+5          E_val = E1_val + T_val; 操作 E_val = 9
移入                        移入                  #5+(9
根据F->(E)归约        执行F.val=E.val动作           #5+(9)          F_val=E_val 操作 F_val =9
根据T->F归约          执行T.val=F.val动作           #5+9           T_val=F_val; 操作 T_val =9
根据E->E+T归约         执行E.val=E1.val+T.val动作          #5+9          E_val = E1_val + T_val; 操作 E_val = 14
   接受               执行print(E.val)动作          #14
```

四、实验中遇到的问题总结

一、遇到的问题

1.根据文法如何设计相应的语义属性和语义动作。

首先根据文法的描述，由于只含有赋值运算和算术运算，所以父节点的属性值都是由子节点的属性值得到的，根据属性定义，此文法只有综合属性，那么相应的语义动作就是赋值操作和算术运算操作。

2.如何在语法分析的同时进行语义分析

可以再建立一个分析栈用来存储语义分析的结果，根据语法分析的结果进行移入或运算操作。

二、思考题

思考题 1：你编写的程序能否和语义动作完全无关，即无论什么样的语义动作，都不需要修改你编写的程序，说明要达到完全无关需满足什么样的条件？

不是完全无关。要达到程序和语义动作完全无关需要在规定文法的同时规定好相应的语义动作，不能写在程序中。

思考题 2：如果你采用的自顶向下的语法分析，你是如何处理综合属性的，如果你采用的是自底向上的语法分析，你是如何处理继承属性的？

先计算出所有的综合属性的值，计算继承属性的值得时候，利用父节点和兄弟节点的综合属性的值和易经的出的继承属性的值来运算该节点继承属性的值。

思考题 3：你产生的结果(四元组)还需经过什么样的处理后就可以等价于汇编程序了？

中间代码优化，目标代码生成。

五、实验体会

此次实验语义分析是建立在语法分析的基础之上进行的，在进行语法分析的同时进行了语义分析，由于上次语法分析程序已经实现，所以这次实现较为简单，只需要在原来的基础上增加规定的语义动作和一个存放语义动作的栈即可。通过这次实验，我体会到语法和语义的紧密连接，只有在学好语法的基础上才能更好地进行语义分析。

指导教师评语：

日期：