编译原理课程实验报告

实验 2：语法分析

姓名	姚敏敏		院系	软件学院		学号	170720231	
任课教师	韩希先			指导教师	韩希先			
实验地点	研究院中 517			实验时间	2019 年 10 月 26 日			
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

要求：需分析本次实验的基本目的，并综述你是如何实现这些目的的？

1. 巩固对语法分析的基本功能和原理的认识。

自底向上的语法分析的基本功能，是从给定的输入符号串出发，试图自底向上地为其建立一棵语法分析树。自底向上的语法分析的原理是从输入串出发，反复利用产生式进行归约，如果最后能得到文法的开始符号，则输入串是相应文法的一个句子，否则输入串有语法错误。在分析过程中，寻找当前句型最左的和某个产生式的右部相匹配的子串，用该产生式的左部符号代替该子串。如果每步都能正确选择子串，就可以得到输入串的最左归约过程，即规范归约过程。

2. 通过对语法分析表的自动生成加深语法分析表的认识。

我们可以通过对闭包的求解、*first* 集的求解、*follow* 集的求解、*action* 表的求解、*goto* 表的求解，从而自动生成语法分析表。

3. 理解并处理语法分析中的异常和错误。

语法分析中的异常和错误主要是移进归约冲突和归约归约冲突，通过建立包含出错处理的优先矩阵来进行处理。

二、实验内容

要求：对如下工作进行展开描述

1. 给出如下语言成分的文法描述

- 函数定义(或过程定义)
  - P → prog id (input, output) D ; S
- 变量说明
  - P → prog id (input, output) D ; S
  - $D \rightarrow D$; D | *List* : T | proc id D ; S
  - *List* → $List_1$, id | id
  - $T \rightarrow$ integer | real | array C of T_1 | ↑T_1 | record D
  - $C \rightarrow$ [num] C | ε
- 赋值
  - S → id := E
  - $E \rightarrow E_1 + E_2$
  - $E \rightarrow -E_1$

- $E \rightarrow (E_1)$
- $E \rightarrow id$

- 布尔表达式
  - $B \rightarrow B_1 \text{ or } B_2 \mid B_1 \text{ and } B_2 \mid \text{not } B_1 \mid (B_1) \mid E_1 \text{ relop } E_2 \mid \text{true} \mid \text{false}$
- 循环
  - $S \rightarrow \text{while B do S1}$
  - $S \rightarrow \text{begin Slist end}$
  - $\text{Slist} \rightarrow \text{Slist; S} \mid \text{S}$
- 分支
  - $S \rightarrow \text{if B then S1} \mid \text{if B then S1 else S2}$

2. 语法分析程序的总体结构及物理实现

首先，加载语言，求解源程序的 first 集和 follow 集，初始化 SLR 分析表，初始化 action 表和 goto 表，然后初始化语法分析程序的状态栈 stateStack 和数值栈 valueStack，然后读入经过词法分析的源程序，然后开始循环对源程序进行语法分析，并且打印 action 表，goto 表，状态栈 stateStack。

3. 语法分析表及其数据结构和查找算法

- 语法分析表主要采用二维 Slr1Node 类型的数组 slr1 来存储，通过一个静态类 Slr1Node 来存储结点的类型 type 和对应的 action，goto 的数值 num。
- 语法分析表的查找算法：算法的主要思想是，利用已有的状态栈和分析栈，首先根据当前读入的状态决定是移进还是规约，如果是移进，就继续移进状态栈，如果是规约，再依据从状态栈栈顶 pop 出的状态进行跳转，直至分析结束。

4. 语法分析表的生成算法

算法的大致步骤如下：

输入：文法 $G=(V, T, P, S)$的拓广文法 G';

输出：G'的 $LR(0)$分析表，即 *action* 表和 *goto* 表;

步骤：

1. 令 I_0= CLOSURE($\{S' \rightarrow .S\}$)，构造 G'的 $LR(0)$项目集规范族 $C=\{ I_0,I_1,\ldots,I_n\}$
2. 让 I_i 对应状态 i，I_0 对应状态 0，0 为初始状态。
3. for k=0 to n do begin

   ⑴ if $A \rightarrow \alpha.a\beta \in I_k$ & $a \in T$ & GO(I_k, a)=I_j then *action*[k,a]:=Sj;

   ⑵ if $A \rightarrow \alpha.B\beta \in I_k$&$B \in V$&GO($I_k$, B)=I_j then *goto*[k,B]:=j;

   ⑶ if $A \rightarrow \alpha. \in I_k$ & $A \rightarrow \alpha$ 为 G 的第 j 个产生式 then

   for $\forall a \in$FOLLOW(A) do *action*[k,a]:=r_j;

   ⑷ if $S' \rightarrow S. \in I_k$ then *action*[k,#]:=acc end;
4. 上述⑴到⑷步未填入信息的表项均置为 error。

5. 错误处理

如果 SLR1 分析表中产生移进规约冲突，就报 SLR1 Conflict error 错误，然后程序退出。

三、实验结果

针对测试程序输出其语法分析结果

- 当我们的输入是 i+i*i#的时候，我们可以打印出此时的状态栈和分析栈中的内容，具体内容和动作如下所示。

```
i+i*i#
步骤    状态栈   符号栈   输入串   动作
1       0        #        i+i*i#   s5
2       05       #i       +i*i#    r6
3                                  goto3
4       03       #F       +i*i#    r4
5                                  goto2
6       02       #T       +i*i#    r2
7                                  goto1
8       01       #E       +i*i#    s7
9       017      #E+      i*i#     s5
10      0175     #E+i     *i#      r6
11                                 goto3
12      0173     #E+F     *i#      r4
13                                 goto9
14      0179     #E+T     *i#      s6
15      01796    #E+T*    i#       s5
16      017965   #E+T*i    #        r6
17                                 goto8
18      017968   #E+T*F    #        r3
19                                 goto9
20      0179     #E+T     #        r1
21                                 goto1
22      01       #E       #        接受!
```

- 我们输入以下的测试程序：

```
int main()
{
    float x;
    x=60;
    int y;
    y=0;
    if(x>=0&&x<60)
    {
        x=60;
    }
    else
    {
        x=100;
    }
    while(x>60)
    {
        y=y+1;
    }
}
$
```

- 测试程序的 *first* 集如图所示

```
================================
First P:    int char float void
First F:    int char float void
First T':   int char float void
First A:    int char float
First D:    int char float
First T:    int char float
First S:    int char ID float { if while
First L:    int char ID float { ) if while
First E:    CHAR - INT REAL ID (
First C:    CHAR INT REAL
First B:    CHAR - INT ! REAL ID (
First R:    < <= > >= == !=
First M:    int char ID float ) { if while
First W:    while
First EL:   else
First OR:   ||
First AND:  &&
```

- 测试程序的 *follow* 集如图所示

```
Follow P:   $
Follow F:   $
Follow T':  ID
Follow A:   )
Follow D:   ; ) ,
Follow T:   ID
Follow S:   int char ID $ else float { } if while
Follow L:   int char ID float { } if while
Follow E:   + * char < ID <= > >= == != ; float ) if int && || { while
Follow C:   + * char < ID <= > >= == != ; float ) if int && || { while
Follow B:   int char ID && || float ) { if while
Follow R:   CHAR - INT REAL ID (
Follow M:   int char ID float { if while
Follow W:   (
Follow EL:  int char ID float { if while
Follow OR:  CHAR - INT ! REAL ID (
Follow AND: CHAR - INT ! REAL ID (
```

- 测试程序的 SLR 分析表如下图所示，由于分析表过于庞大，所以只截取一小部分作为展示。

```
================================
VT: auto    break   case    char    const   continue    default do  double  else    enum    extern  float   for goto    if  int long    register    return  shc
0:  e0  e0  e0  r41 e0  e0  e0  e0  e0  e0  e0  r41 e0  e0  r41 r41 e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  r41 e0  e0  e0  e0  e0  e0
1:  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0
2:  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  s21 e0  e0  e0  e0  e0
3:  e0  e0  e0  s74 e0  e0  e0  e0  e0  e0  e0  s79 e0  e0  e0  s80 e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  s1  e0  e0  e0  e0  e0  e0  e0
4:  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  s47 e0  s51 e0  e0  e0
5:  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0
6:  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0
7:  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0
8:  e0  e0  e0  r22 e0  e0  e0  e0  e0  e0  e0  r22 e0  e0  r22 r22 e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  r22 r22 e0  r22 e0  r22 r22 r2
9:  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0
10: e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0
11: e0  e0  e0  r31 e0  e0  e0  e0  e0  e0  e0  r31 e0  e0  r31 r31 e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  r31 e0  e0  e0  e0  e0  e0
12: e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0
13: e0  e0  e0  r11 e0  e0  e0  e0  e0  r11 e0  e0  r11 e0  e0  r11 r11 e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  r11 e0  e0  e0  e0  e0  e0
14: e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  r36 e0  e0  e0  e0
15: e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  r38 e0  e0  e0  e0
16: e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  r35 e0  e0  e0  e0
17: e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  r37 e0  e0  e0  e0
18: e0  e0  e0  r32 e0  e0  e0  e0  e0  e0  e0  r32 e0  e0  r32 r32 e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  r32 s47 e0  s51 e0  e0  e0
19: e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  r34 e0  e0  e0  e0
20: e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  r39 e0  e0  e0  e0
21: e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  s21 e0  e0  e0  e0
22: e0  e0  e0  r40 e0  e0  e0  e0  e0  e0  e0  r40 e0  e0  r40 r40 e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  r40 e0  e0  e0  e0  e0  e0
23: e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0  e0
```

四、实验中遇到的问题总结

一、 实验过程中遇到的问题如何解决的？

由于第一个实验词法分析是用 C++进行开发的，而语法分析这个程序是用 Java 进行开发的，这样就涉及到词法分析的结果的传递的问题。限于目前的知识面，暂时无法做到两个不同语言的程序之间直接传数据。最后，我想到了一个解决办法，就是把之前语法分析的结果写入到 txt 格式的文件中，然后语法分析程序再从此文件中读取，虽然这种办法在程序的健壮性、安全性、完整性等多个方面都大打折扣，但是却十分简便易行。如果还想改进这个办法，最好还是将之前

写的 C++程序用 Java 重新写一遍。但是考虑到时间紧迫，所以暂时没有完成。

二、 思考题的思考与分析

思考题 1：给出在生成语法分析表时所遇到的困难，以及是如何处理的？

生成语法分析表时的困难主要是实现求解 *first* 集的求解、*follow* 集的求解、*action* 表的求解、*goto* 表的求解。这几个算法都是比较困难的，在实现他们过程中，首先认真地阅读了书上的相关章节，然后到网络上找了很多相关资料，历时两周才写完构造以上几个表和语法分析表的算法。

思考题 2：思考还可以什么形式来给出语法分析的结果？

可以通过构建语法树的形式来给出语法分析的结果，不过构建出一棵树的成本还是有些高，如果不好好优化的话，时间复杂度还是有些高。

还可以通过绘制识别拓展文法全部活前缀的 DFA，从图中看出各个状态的转换，然后构造语法分析表，通过绘制有限确定状态机来显示语法分析的结果。

思考题 3：如果在语法分析中遇到了语法错误，是应该中断语法分析呢，还是应该进行适当处理后继续语法分析，你是怎么处理的？

如果是产生了移进规约冲突，或者是规约规约冲突，我们不必中断语法分析，可以对程序采取紧急方式恢复策略，发现错误时跳过一些输入符号，直到出现下一个语法成分包含的第一个符号为止，在实现的时候，通常用某个期望的同步记号作为相应的标记，这种错误处理方法的效果依赖于属于 *follow* 集合的同步记号的选择。

如果是其他未知的语法错误，事先没有准备的，就可以中断语法分析，因为通常在这种情况下是很难处理的，而且这也是很少出现的。

五、实验体会

通过语法分析的实验，对 LR(0)分析法和 SLR(1)分析法都有了更加深刻的了解。LR(0)分析法不需要向前查看输入符号，只需要根据当前的栈顶状态就可以确定下一步所应采取的动作。但是，SLR(1)分析法让分析器向前查看一个输入符号，以便确定面对当前输入符号时是否进行规约，以及按照哪个产生式进行规约。只需要在 LR(0)分析法的算法中更改一小部分代码即可。

在使用 Java 编写语法分析程序的时候，对于状态栈和分析栈中实现，我们采用了 Java 中自带的 Stack 类来对状态栈和分析栈进行操作。这样子极大得节省了代码的编写时间，提高了工作效率。

指导教师评语：

日期：