编译原理课程实验报告

实验 2：语法分析

姓名	张泽宇	院系	计算机科学与技术学院	学号	1163300620

任课教师	辛明影		指导教师	辛明影		
实验地点	格物 208		实验时间	2019.04.21		
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

使用 LR（1）方法实现语法分析。具体要求如下：

（1）能识别以下几类语句：

1.声明语句（变量声明）

2.表达式及赋值语句（简单赋值）

3.分支语句：if_then_else

4.循环语句：do_while

（2）要求编写自动计算 CLOSURE(I)和 GOTO 函数的程序，并自动生成 LR 分析表。

（3）具备简单语法错误处理能力，能准确给出错误所在位置，并采用可行的错误恢复策略。

（4）通过文件导入文法和测试用例

（5）系统的输出分为两部分：一部分是打印输出语法分析器的 LR 分析表。另一部分是打印输出语法分析结果，既输出归约时的产生式序列.

二、文法设计

得分

要求：给出如下语言成分的文法描述。

- 声明语句（变量声明）

```
type_specifier
    : double
    | char
    | int
    | float
    ;

declaration_assign
    : '=' expression
    | '$'
    ;
```

```
declaration_init
    : id declaration_assign
    ;

declaration_init_list
    : ',' declaration_init declaration_init_list
    | '$'
    ;

declaration
    : type_specifier declaration_init declaration_init_list ';'
    ;

function_declaration
    : type_specifier id
    ;

function_declaration_suffix
    : ',' function_declaration function_declaration_suffix
    | '$'
    ;

function_declaration_list
    : function_declaration function_declaration_suffix
    | '$'
    ;

function_definition
```

: function type_specifier id '(' function_declaration_list ')'

;

function_implement

: function_definition compound_statement

;

- 表达式及赋值语句

primary_expression

: id

| number

| '(' expression ')'

;

operator

: '+'

| '-'

| '*'

| '/'

| '%'

| '<'

| '>'

| '=' '='

| '!' '='

;

arithmetic_expression

: operator primary_expression arithmetic_expression

```
        | '$'
        ;

constant_expression
        : primary_expression arithmetic_expression
        ;

assignment_operator
        : '='
        | '+' '='
        | '-' '='
        | '*' '='
        | '/' '='
        | '%' '='
        ;

assignment_expression
        :id assignment_operator expression
        ;
expression_statement
        : assignment_expression_list ';'
        ;
```

- 分支语句：if_then_else

```
jump_statement
        : continue ';'
        | break ';'
        | return expression ';'
        ;
```

selection_statement

: if '(' expression ')' statement else statement

;

- 循环语句：do_while

iteration_statement

: while '(' expression ')' statement

| for '(' declaration expression ';' assignment_expression ')' statement

;

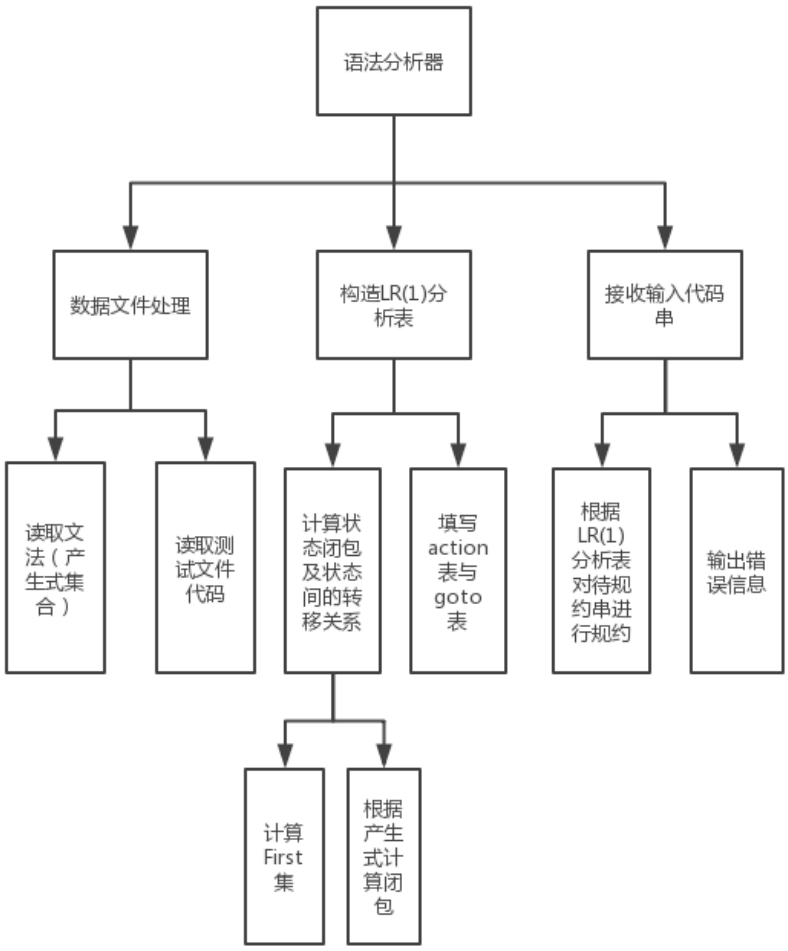
三、系统设计

得分

（1）语法分析器的实现流程

语法部分分析主要完成如下工作：

1. 根据文法对定义来构造 LR(1)分析表
2. 根据构造的 LR(1)表，对由终端输入的代码进行接收，发生错误后需要输出错误提示。



（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

1. class LRItem

介绍：用于表示 LR 分析表构造过程中的**项目。**

属性：String left，项目的左部

List<String> right，项目的右部

Int status，表示该项目的**识别状态，**也就是圆点的位置。

Set<String> search_symbols，表示该项目的搜索符

函数：仅包含一些基本的函数，包括构造函数，各属性的 getter()方法、setter()方法以及 hashCode()、equals()和 toString()函数。

2. class LRItemSet

介绍：用于表示 LR 分析法中的**有效项目集**。

属性：

Set<Item> decideItems，表示该项目集的决定项目，通过这些项目可以确定一个有效项目集。

List<Item> items，该项目集包含的所有项目，通过对决定项目求项目集闭包**(cluster)**可得。

函数：仅包含一些基本的函数，包括构造函数，getter()、setter()函数以及 hashCode()和 equals()，要注意该类的 hashCode()与 equals()编写时，仅与**决定项目(decideItems)**有关系。

3. class Production

介绍：用于表示产生式。

属性：左部：String left

右部：List<String> right

函数：仅包含基本的函数，hashCode()、equals()等。

4. Map<Integer, Map<String, String>> LRTable

功能：用于**存储 LR 分析表**的数据结构，以三元组的形式表示。

介绍：第一个 Integer 表示状态的编号，内层 Map 中第一个 String 表示输入符号，第二个 String 表示在该状态下遇到该符号时，要进行的操作（移入、归约）。

5. List<ItemSet> itemSets

功能：用于存储 LR 分析表自动生成的**所有有效项目集**

介绍：自动生成 LR 分析表之前首先要计算文法所有的有效项目集，然后根据项目集可以生成 LR 分析表。

6. Map<String, List<String>> firsts

功能：用于存储所有非终结符的 first 集。

7. List<Production> productions

功能：用于存储 LR 文法中所有的产生式。

✓ 主要功能函数说明

1. List<String> first(String input)

函数功能：给定一个输入符号，求该符号在文法中的 first 集。

输入：输入符号，可能是终结符也可能是非终结符

输出：该符号的 first 集。

执行过程：

1. 如果输入符号是终结符，则返回只有输入符号的列表。

2. 如果输入符号是非终结符，遍历所有左部为该符号的产生式，递归地求其首字符的 first 集并加入输入符号 first 集中，如果遇到空产生式，直接将空符号加入 first 集。注意有左递归的情况需要判断。

2. void follow()

函数功能：用于求所有非终结符的 Follow 集，该函数在 SLR 表的自动生成过程中需要用到。

执行过程：

1. 首先初始化所有非终结符的 Follow 集。
2. 将'#'加入文法开始符号的 follow 集中
3. 对于该文法所有的产生式，重复执行下面的步骤，直到所有非终结符的 Follow 集不发生变化。
4. 对于产生式 A -> α B β，Follow(B) = Follow(B) ∪ First(β) – {空串}
5. 对于产生式 A -> α B 或 A -> α B β， β 可以推出空串，那么

Follow(B) = Follow(B) ∪ Follow(A)

3. List<Item> cluster(ItemSet)

函数功能：求对于同一活前缀有效的项目的集合。SLR 文法与 LR 文法该函数的实现过程略有不同，对于 SLR 文法，遍历一遍项目的列表直到列表不再扩大就可以停止；对于 LR 文法，需要重复遍历所有的项目，直到某一次遍历结束前后项目集不发生变化为止。

输入：项目集(需要包含决定项目)

输出：与该项目集中项目对同一活前缀有效的项目集集合。

执行过程：

1. C ：= I
2. 重复执行下面的过程，直到 C 不发生变化。
3. 对 C 中的每个项目[A -> α .B β , a], 对每一个 B -> γ 产生式和 First(β a)中的每个终结符，构成新的项目，如果新项目不在 C 中，则将其加入。
4. return C

4. ItemSet goTo<ItemSet itemset, String input)

函数功能：确定项目集遇到一个输入符号后的**转移项目集**。

输入：ItemSet itemset，当前项目集

String input，输入符号

输出：当前项目集关于输入符号的后继项目集。

执行过程：

1. 遍历当前项目集的所有项目，处理除了归约项目之外的其他项目[A -> α .X β]，将所有 X 相同的项目的圆点后移一位，[A -> α X. β]加入新的项目集中。

2. 对于每个新的项目集，求其项目集闭包后返回。

5. void calculeta_ItemSets()

函数功能：构造文法**所有的项目集规范族**

执行过程：

1. 初始化。将文法的开始产生式加入项目集，求其项目集闭包，然后加入项目集规范族 C。
2. 重复执行下面的过程，直到项目集规范族 C 不发生变化

3. 对于每个项目集，求该项目集关于每个符号的 goTo()后继项目集，如果后继项目集不在 C 当中，则将该项目集加入。

6. void construct_table()

函数功能：根据文法的所有项目集规范族，**自动生成 LR 分析表**。对于 SLR 分析表，不同的地方仅在于归约项目的处理。

执行过程：

1. 调用 calculate_ItemSets()，生成文法的项目集规范族。
2. 对于每个项目集，考虑其中的所有项目 I_k，根据项目的类型进行选择进行下面的操作。
3. 对于移进项目 [A -> α .a β , b]，如果 goTo(I_k, a)=I_j，填入 action[k, a] = Sj;
4. 对于待约项目，如果 goTo(I_k, B)=I_j，填入 goto[k, B] = j;
5. 对于归约项目，归约所使用的产生式编号为 j，对于该项目的所有搜索符，填入 action[k, a]=rj
6. 对于接受项目，填入 action[k, #] = acc.

7. void analyze()

函数功能：语法分析的**主控程序**。

执行过程：

1. 初始化状态栈，向里面压入 0；初始化符号栈，向里面压入#
2. 考虑当前状态栈顶的状态和输入符号，查找 LR 分析表，根据内容选择进行下面的操作：
3. 如果是 **Si**，表明当前需要移入，将当前输入符号和状态分别压入状态栈和符号栈。
4. 如果是 **rj**，表明当前需要归约，用下标所示的产生式进行归约，规约过程中，状态栈和符号栈需要弹出的个数相同，然后将产生式的左部压入栈中。然后根据当前两个栈顶的内容查看 goto 表，将该表中的内容压入状态栈。
5. 如果查表结果为空，说明发生了错误，进入错误处理程序。下面介绍一下**错误处理策略**。

8. void error_handle()

函数功能：语法分析的**错误处理程序**。

方法介绍：当查询 action 表为空时，执行该程序。这里采用的是**恐慌模式**的错误处理策略。当发现错误时，语法分析器开始抛弃输入符号，每次抛弃一个符号，直到发现某个指定的**同步记号**位置。同步符号在本实验中采用的是 }，而 } 对应的非终结符为 **D**，这种方法相当于确定从 D 推导出的串中包含错误，这个串一部分已经被处理，并在栈顶形成了一个状态序列，所以要先这些**状态序列弹出（对应执行过程的第一步）**，这个串的剩余部分还没有处理，所以要**试图跳过这个串的剩余部分，找到一个可以合法的跟在 D 后的符号（对应执行过程的第二步）**。这种方法通常会跳过大量的输入符号，而不检查其中是否有其他错误。

执行过程：

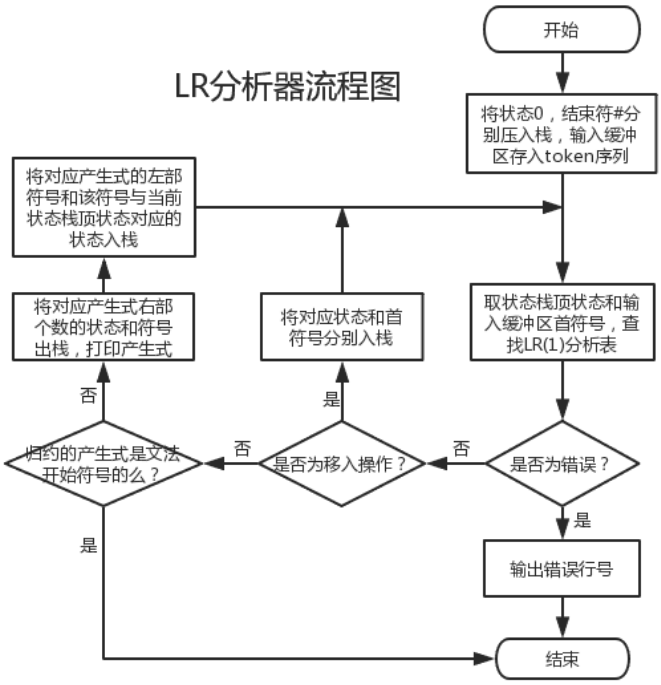
1. 首先从栈顶开始退栈，直到发现 D 上具有的转移状态；
2. 然后抛弃输入符号，直到一个可以合法地跟在 D 后的符号 a，也就是 a∈Follow(D)，然后就可以恢复正常分析。

错误信息的提示：错误信息采用的格式如下图所示：

```
Error at Line(24): [ Missing variable ]
Error at Line(30): [ Missing ';' ]
Error at Line(33): [ Missing '{' ]
```

关于不同种类的错误信息的提示，是根据从栈顶弹出的第一个状态确定的，因为第一个状态可以表明当前语法分析执行到了哪一步，是从哪一步开始出现的问题。

✓ 程序核心部分的程序流程图



四、系统实现及结果分析

得分

（1） 系统实现过程中遇到的问题；

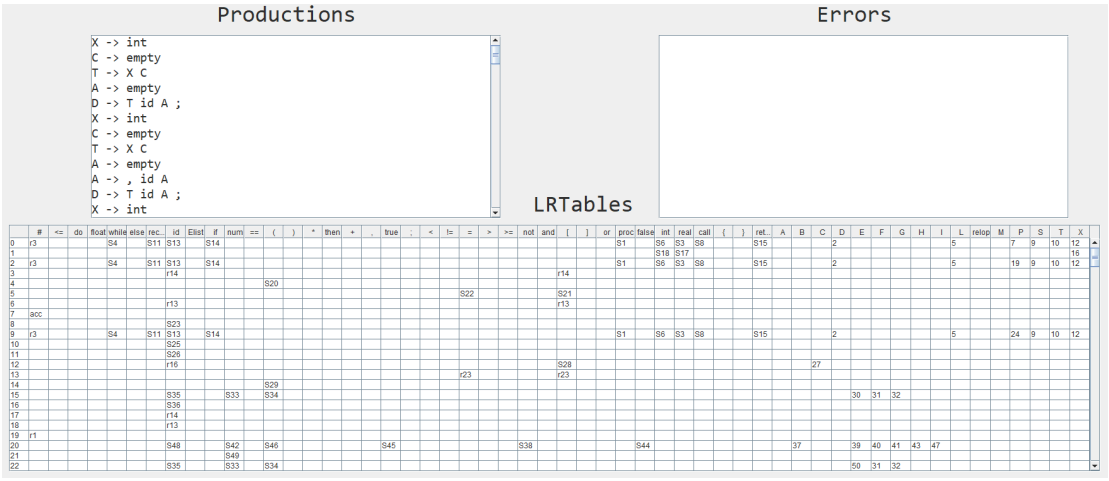
1. 在求解 First 集的时候遇到的问题：需要判断是否有左递归的情况，否则会一直递归导致栈满。

2. 在求解 LR 项目集闭包时遇到的问题：不能只通过一次循环就获得项目集闭包，因为新加入的项目可能会对之前的项目的搜索符有影响，所以需要一直循环，直到某次循环前后项目集的内容不发生变化为止。如果是 SLR 的话不会出现这样的问题，因为没有搜索符，后加入的项目不会影响到之前的项目。

3. 代码实现上出现的一个问题：Java 的深拷贝与浅拷贝，比如说在求项目集时，需要判断某次循环前后是否发生变化，这个时候如果浅拷贝的话肯定是不发生变化的，得不到正确的结果。

（2） 针对一测试程序输出其句法分析结果；

语法分析结果包括四个部分，产生式序列、错误信息以及符号表可以在程序的可视化界面中展示出来，如下图所示：



第四部分为执行过程中，状态栈、符号栈以及剩余输入符号的内容：

```
, 123, 166, 197, 104, 104, 113, 32]
, 123, 166, 197, 104, 104, 113]
, 123, 166, 197, 104, 104, 113, 159]
, 123, 166, 197, 104, 104, 113, 159, 59]
, 123, 166, 197, 104, 104, 113, 159, 59, 34]
, 123, 166, 197, 104, 104, 113, 159, 59]
, 123, 166, 197, 104, 104, 113, 159, 59, 31]
, 123, 166, 197, 104, 104, 113, 159, 59]
, 123, 166, 197, 104, 104, 113, 159, 59, 116]
, 123, 166, 197, 104, 104, 113]
, 123, 166, 197, 104, 104, 113, 159]
, 123, 166, 197, 104, 104, 113, 159, 190]
, 123, 166, 197, 104, 104]
, 123, 166, 197, 104, 104, 109]
, 123, 166, 197, 104, 104, 109]
, 123, 166, 197, 104, 104, 109, 155]
, 123, 166, 197, 104, 104]
, 123, 166, 197, 104, 104, 150]
, 123, 166, 197, 104]
, 123, 166, 197, 104, 150]
, 123, 166, 197]
, 123, 166, 197, 216]
, 123, 166, 197, 216, 229]
```

```
[#, D, D, proc, X, id, (, M, ), {, D, D, return, G]
[#, D, D, proc, X, id, (, M, ), {, D, D, return, E]
[#, D, D, proc, X, id, (, M, ), {, D, D, return, E]
[#, D, D, proc, X, id, (, M, ), {, D, D, return, E, +]
[#, D, D, proc, X, id, (, M, ), {, D, D, return, E, +, id]
[#, D, D, proc, X, id, (, M, ), {, D, D, return, E, +, F]
[#, D, D, proc, X, id, (, M, ), {, D, D, return, E, +, F]
[#, D, D, proc, X, id, (, M, ), {, D, D, return, E, +, G]
[#, D, D, proc, X, id, (, M, ), {, D, D, return, E, +, G]
[#, D, D, proc, X, id, (, M, ), {, D, D, return, E]
[#, D, D, proc, X, id, (, M, ), {, D, D, return, E]
[#, D, D, proc, X, id, (, M, ), {, D, D, return, E, ;]
[#, D, D, proc, X, id, (, M, ), {, D, D, S]
[#, D, D, proc, X, id, (, M, ), {, D, D, S]
[#, D, D, proc, X, id, (, M, ), {, D, D, S, P]
[#, D, D, proc, X, id, (, M, ), {, D, D, S, P]
[#, D, D, proc, X, id, (, M, ), {, D, D, P]
[#, D, D, proc, X, id, (, M, ), {, D, D, P]
[#, D, D, proc, X, id, (, M, ), {, D, P]
[#, D, D, proc, X, id, (, M, ), {, D, P]
[#, D, D, proc, X, id, (, M, ), {, P]
[#, D, D, proc, X, id, (, M, ), {, P, }]
[#, D, D, D]
[#, D, D, D]
[#, D, D, D, P]
[#, D, D, D, P]
[#, D, D, P]
```

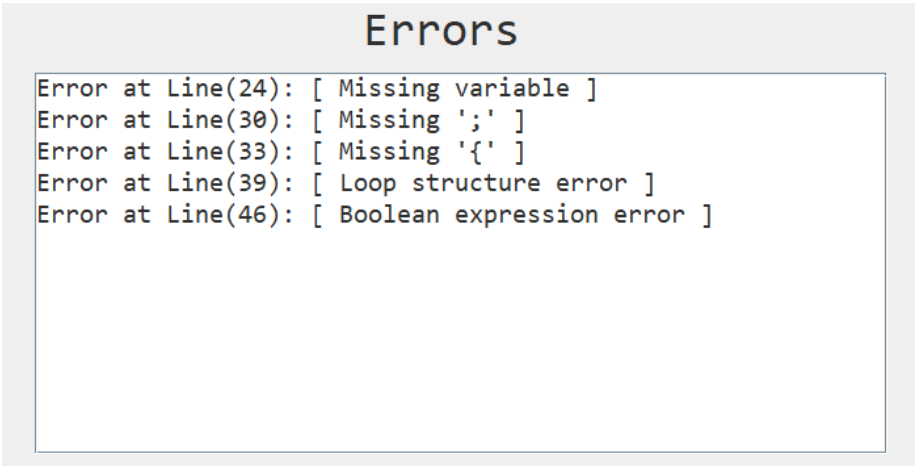
```
GOTO[113, G]
r25(E -> G )
GOTO[113, E]
S59
S34
r30(F -> id )
GOTO[59, F] =
r27(G -> F )
GOTO[59, G] =
r24(E -> E +
GOTO[113, E]
S190
r21(S -> retu
GOTO[104, S]
r3(P -> empty
GOTO[109, P]
r2(P -> S P )
GOTO[104, P]
r1(P -> D P )
GOTO[104, P]
r1(P -> D P )
GOTO[197, P]
S229
r4(D -> proc
GOTO[2, D] =
r3(P -> empty
GOTO[2, P] =
r1(P -> D P )
```

语法分析正确测试样例结果分析：

1. 分析表由 251 个状态组成，由于实验中使用的文法是 LR1 的，所以不会出现一个表项中有两个的情况，且该分析表可以用来正确地执行语法分析。

2. 产生式序列：结果显示出归约时所使用的产生式序列。

（3） 输出针对此测试程序对应的语法错误报告；



（4） 对实验结果进行分析。

由于实验中使用的**恐慌模式**的错误处理方法，所以在出错位置向后，到同步符号之前，如果这段程序中有其他的语法错误，是检查不到的，这就是恐慌模式的缺点，所以这种方法适用于语法错误较少的程序段。错误信息包括发生语法错误的位

置以及错误的原因。
指导教师评语： 日期：