北京交通大學

BEIJING JIAOTONG UNIVERSITY

《编译原理》

实验报告

实验名称：专题五基于SLR(1)语义分析中间代码生成

学　　号：22281188

姓　　名：江家玮

学　　院：计算机科学与技术学院

日　　期：2024 年 12 月 17 日

目录

一、 程序功能描述

1.1 理论知识

语法制导的基本概念，目标代码结构分析的基本方法，赋值语句语法制导生成四元式的基本原理和方法，该过程包括语法分析和语义分析过程。

1.2 实验项目

完成以下描述赋值语句 SLR(1)文法语法制导生成中间代码四元式的过程。

G[S]:S→V=E

E→E+T|E-T|T

T→T*F|T/F|F

F→(E)|i

V→i

1.3 设计说明

终结符号 i 为用户定义的简单变量，即标识符的定义。

1.4 设计要求

（1）构造文法的 SLR(1)分析表，设计语法制导翻译过程，给出每一产生式对应的语义动作；

（2）设计中间代码四元式的结构；

（3）输入串应是词法分析的输出二元式序列，即某赋值语句“专题 1”的输出结果，输出为赋值语句的四元式序列中间文件；

（4）设计两个测试用例（尽可能完备），并给出程序执行结果四元式序列。

（5）考虑根据文法自动构造 SLR（1）分析表，并添加到你的程序中。

二、主要数据结构描述

变量类型	变量名称	说明
string	start	开始符号
set<string>	Vt	终结符号集
set<string>	Vn	非终结符号集
set<string>	VtP	终结符号+#
map<string, set<string>>	first	first 集
map<string, set<string>>	follow	follow 集
struct	struct status	DFA 活前缀
struct	ActionNode	Action 结点
struct	GotoNode	Goto 结点
list<ActionNode>	Action	Action
list<GotoNode>	Goto	Goto
vector<anaNode>	stack	分析栈
vector<string>	leftString	剩余输入串
vector<string>	temp	中转记录输入串
vector<string>	recValue	中转记录真实输入串
vector<string>	recleftValue	剩余真实输入串

三、程序结构描述

3.1 Fisrt 集的计算

先遍历所有产生式，识别出右侧以终结符开头的产生式，例如$A \to \alpha$。在此过程中，必须特别注意不遗漏那些右侧仅由单一终结符组成的产生式，如$A \to a$。需要强调的是，许多研究者在遇到 $A \to \alpha$时，常误以为终结符 a 后必然跟随其他符号，这种假设是不准确的。因此，对于每一个符合条件的产生式，应将终结符 a 添加到 $First(A)$ 集合中。

其次，继续遍历所有产生式，识别出右侧以非终结符开头的产生式，例如 A → Bα。在此步骤中，同样需要注意处理那些右侧仅包含非终结符 B 的产生式，如$A \to B$。对于每一个此类产生式，应将 $FIRST(B)$ 集合中的所有终结符添加到 $FIRST(A)$ 集合中。

3.2 FOLLOW 集的计算

1. 将终结符 # 加入到 FOLLOW(S) 中，其中 S 为文法的起始符号。

2. 遍历所有产生式，识别右侧包含子串 Ba… 的产生式，例如 $A \to \alpha B a \beta$。在此过程中，需特别注意 B 前后可能不存在任何符号的情况。对于每一个符合条件的产生式，应将终结 a 添加到$FOLLOW(B)$中。

3. 遍历所有产生式，识别右侧包含相邻非终结符对 BC 的产生式，例如 $A \to \alpha BCDE$ 。对于任意两个相邻的非终结符(B, C)、(C, D)、(D, E)，应将$FIRST(C)$、$FIRST(D)$ 、$FIRST(E)$ 中除空串以外的所有终结符加入到 $FOLLOW(B)$、$FOLLOW(C)$、 $FOLLOW(D)$ 中。

4. 遍历所有产生式，识别右侧以单一非终结符 B 结尾的产生式，例如 $A \to \alpha B$。此时，应将$FOLLOW(A)$ 集合中的所有终结符添加到 $FOLLOW(B)$ 中。特别需要注意的是，不应遗漏那些右侧仅包含一个非终结符的产生式，如$A \to B$。

5. 遍历所有产生式，识别右侧以相邻非终结符对 BC 结尾的产生式，例如 $A \to \alpha BCDE$。在此情形下，需考虑最后两个非终结符 D 和 E 的关系。若 $FIRST(E)$ 包含空串，则应将$FOLLOW(A)$添加到 $FOLLOW(D)$ 中。此步骤实质上是步骤 4 的一个特殊情况。

(3) 有效项目集规范族的构造

1. 闭包函数$CLOSURE(I)$的定义如下：

- 初始条件：所有属于集合 I 的项目均包含在 $CLOSURE(I)$ 中。
- 扩展条件：若 $A \to \alpha \cdot B\beta$ 属于$CLOSURE(I)$，则每一个形如 $B \to \cdot \gamma$ 的项目也必须包含在 $CLOSURE(I)$ 中。
- 收敛条件：重复应用扩展条件，直至不再有新的项目被加入，即 $CLOSURE(I)$不再扩展。

2. 转换函数 $GO(I,X)$ 的定义：

$$GO(I,X) = CLOSURE(J)$$

其中：

- I 为包含某一项目的状态集合。
- X为文法符号，且 $X \in (V_N \cup V_T)$。
- $J = \{A \to \alpha X \cdot \beta | A \to \alpha \cdot X\beta \in I\}$ 。

通过应用闭包函数和转换函数，可以构造文法 $G\{'\}$ 的 LR(0) 项目集规范族，具体步骤如下：

a) 初始状态的构造：将项目 $S' \to \cdot S$ 作为初态集的核心，然后对其求闭包，得到初态的项目集$CLOSURE(\{S' \to \cdot S\})$ 。

b) 状态的扩展：对初态集及后续构造的各个项目集应用转换函数 $GO(I,X) = CLOSURE(J)$，以求得新的状态集J。

c) 迭代过程：重复步骤 b，直至所有可能的状态均被构造，且不再生成新的状态。

3.3 SLR 分析表的构造

首先，将文法 G 拓广为G'，并基于G'构造 $LR(0)$ 项目集规范族 C 及其状态转换函数GO。随后，依据以下方法构造分析表中的$ACTION$和 $GOTO$ 函数：

1. 移进操作：

✓ 若项目$A \to \alpha \cdot b\beta$属于$I_k$，且 $GO(I_k, b) = I_j$ （其中 b 为终结符），则将$ACTION[k,b]$ 设为“移进至状态 j”，记为“ s_j ”。

2. 规约操作：

✓ 若项目 $A \to \alpha \cdot$ 属于I_k，则对所有 $a \in FOLLOW(A)$，将 ACTION[k, a] 设为“使用产生式 $A \to \alpha$ 进行规约”，记为“ r_j ”，其中假定 $A \to \alpha$ 为文法 G'的第 j 个产生式。

3. 接受操作：

✓ 若项目$S' \to S \cdot$ 属于I_k，则将 ACTION[k, \ #]设为“接受”，记为“acc”。

4. GOTO 操作：

- ✓ 若 $GO(I_k, A) = I_j$（其中 A 为非终结符），则将 GOTO[k, A] 设为 j 。
- ✓ 在构造分析表的过程中，凡是未被上述规则覆盖的空白格应标记为“出错标志”。
- ✓ 分析器的初始状态为包含 $S' \rightarrow \cdot S$ 的项目集状态。
- ✓ SLR 分析中可能遇到的冲突类型主要包括移进-规约冲突和规约-规约冲突。

5. 冲突检测

- ✓ 通过 Conflict 函数检测当前状态的 FIRST 集合和 FOLLOW 集合是否存在冲突。如果检测到冲突，应采取相应措施予以消除；若无冲突，则继续执行后续步骤。

3.4 四元式的生成

根据语法分析过程进行语法制导翻译，具体步骤如下：

1. 栈的设立：设立三个栈，分别为状态栈、符号栈以及存放四元式变量的栈。

2. 规约过程：

- 当出现规约操作 $V \rightarrow \alpha$ 时，将 α 压入相应的栈中。
- 对于特定的产生式规则，例如：
- $A \rightarrow V = E$
- $E \rightarrow E + T$ 或 $E \rightarrow E - T$
- $T \rightarrow T * F$ 或 $T \rightarrow T / F$

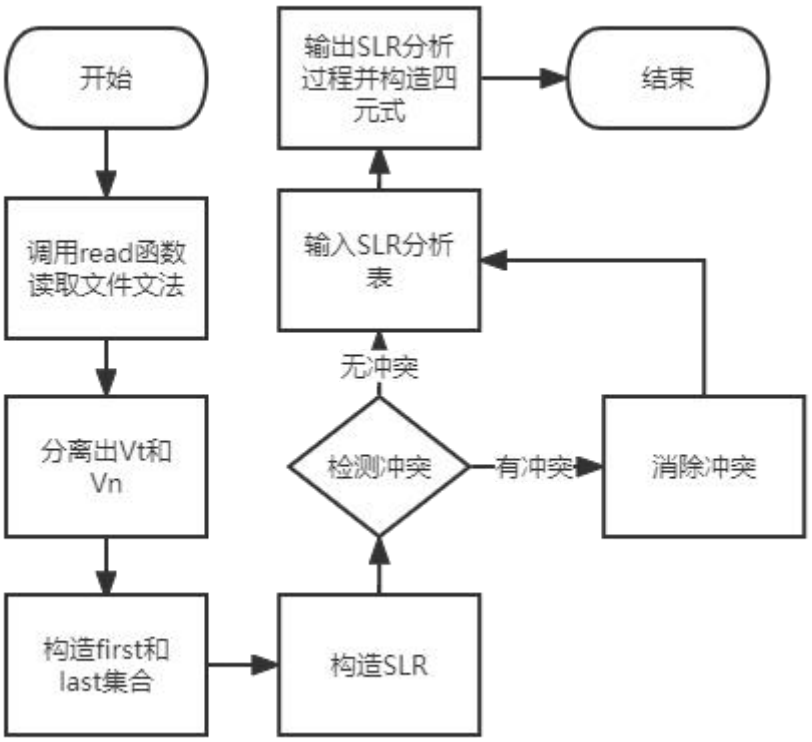
使用 $newTemp$ 函数生成新的中间变量，根据语义规则生成相应的四元式。

3. 变量的管理：

- 在规约过程中，将对应栈中的变量弹出，并将新生成的变量压入四元式变量栈中。

通过上述步骤，可以生成反映程序中间代码逻辑的四元式，从而为后续的代码生成和优化提供基础。

3.5 程序流程图如下：



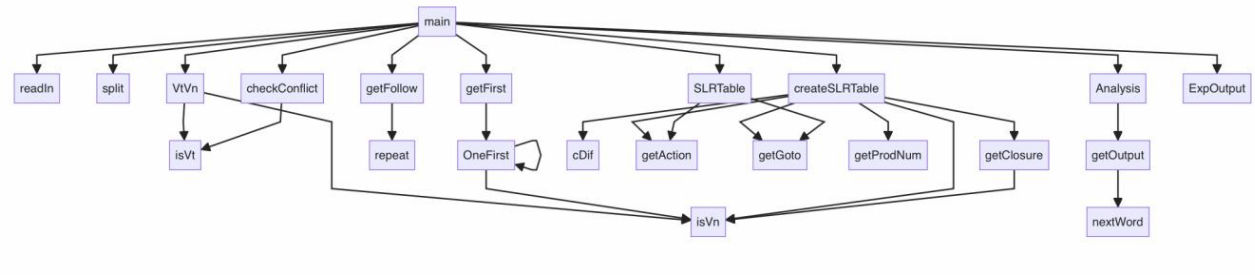
3.6 函数定义以及函数之间的调用关系

3.6.1 函数定义

函数名称	函数功能描述
split()	分解文法规则，将每条产生式拆分为左部和右部，并根据
isNumber()	判断字符串是否为数字，返回布尔值。
readIn()	计算所有非终结符的 FIRSTVT 集合，用于构建优先关系表，确定终结符之间的优先级。
VtVn()	根据读取的文法规则区分终结符和非终结符，并将 # 加入扩展终结符集合。
OneFirst()	递归计算单个非终结符的 FIRST 集，处理终结符和非终结符的 FIRST 规则，并存储结果。
getFirst()	计算所有非终结符的 FIRST 集，并输出结果。
getFollow()	计算所有非终结符的 FOLLOW 集，按照规则逐步更新，并输出结果。

cDif()	比较两个状态是否相等，用于判定 DFA 状态是否重复。
getClosure()	计算一个状态的闭包，处理项集中的点 · 后的符号，递归生成新状态项。
createSLRTable()	创建 SLR(1) 分析表，包含 ACTION 和 GOTO 表的生成，同时输出项集族。

3.6.2 函数之间的调用关系



四、程序测试

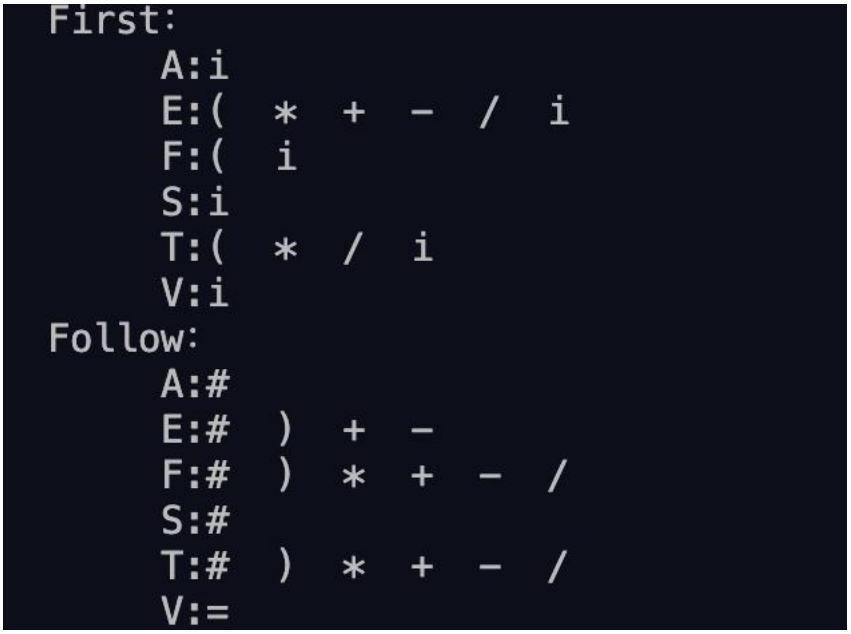
4.1 用例 1 测试

测试用例:

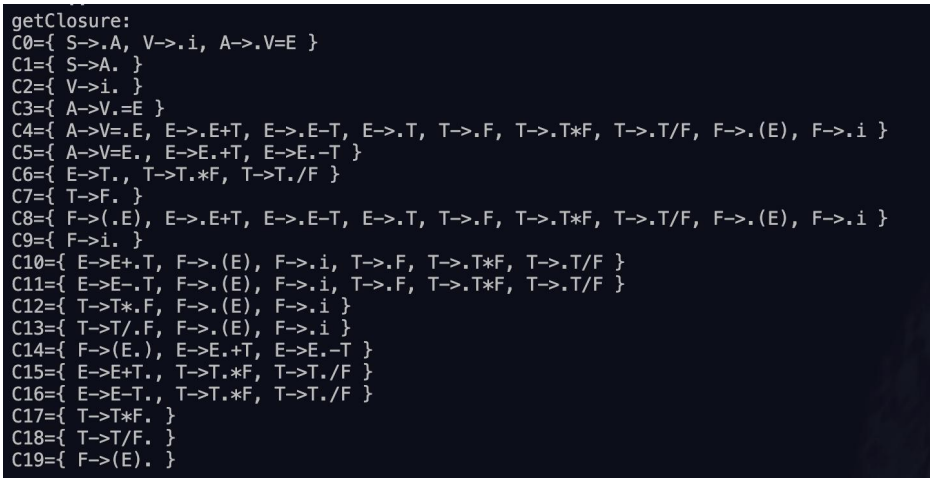
(identifier,p)
(=,)
(identifier,x)
(+,)
((,)
(identifier,x)
(/,)
((,)
(identifier,y)
(+,)
(identifier,z)
(),)
((,)
(*,)
((,)
(identifier,z)
(/,)
(identifier,y)
(*,)

(identifier,y) **(),)**

（1）生成的 First 和 Follow 集如下



（2）生成的有效项目集规范族如下



（3）冲突检测结果如下：

（4）SLR(1)分析表构造如下

SLR(1)分析表如下

	Action								GOTO				
I	|(| |)	|*	|+	|-	|/	|=	|i	|#	|A	|E	|F	|T	|V
|			|s2		|1				|3					
1									|acc					
2								|r10						
3								|s4						
4	|s8								|s9			|5	|7	|6	
5				|s10	|s11					|r0					
6		|r3	|s12	|r3	|r3	|s13				|r3					
7		|r7	|r7	|r7	|r7	|r7				|r7					
8	|s8								|s9			|14	|7	|6	
9		|r5	|r5	|r5	|r5	|r5				|r5					
10	|s8								|s9				|7	|15	
11	|s8								|s9				|7	|16	
12	|s8								|s9				|17		
13	|s8								|s9				|18		
14		|s19		|s10	|s11										
15		|r1	|s12	|r1	|r1	|s13				|r1					
16		|r2	|s12	|r2	|r2	|s13				|r2					
17		|r8	|r8	|r8	|r8	|r8				|r8					
18		|r9	|r9	|r9	|r9	|r9				|r9					
19		|r4	|r4	|r4	|r4	|r4				|r4					

（5）SLR 分析过程及产生的四元式如下图

表达式： identifier=identifier+(identifier/(identifier+identifier)(*(identifier/identifier*identifier)

SLR(1)分析过程如下

|状态栈	|符号栈	|剩余串	|操作	|四元式
|s0	|#	|i=i+(i/(i+i)(*(i/i*i)#	|s2	
|s0s2	|#i	|=i+(i/(i+i)(*(i/i*i)#	|r10	
|s0s3	|#V	|=i+(i/(i+i)(*(i/i*i)#	|s4	
|s0s3s4	|#V=	|i+(i/(i+i)(*(i/i*i)#	|s9	
|s0s3s4s9	|#V=i	|+(i/(i+i)(*(i/i*i)#	|r5	
|s0s3s4s7	|#V=F	|+(i/(i+i)(*(i/i*i)#	|r7	
|s0s3s4s6	|#V=T	|+(i/(i+i)(*(i/i*i)#	|r3	
|s0s3s4s5	|#V=E	|+(i/(i+i)(*(i/i*i)#	|s10	
|s0s3s4s5s10	|#V=E+	|(i/(i+i)(*(i/i*i)#	|s8	
|s0s3s4s5s10s8	|#V=E+(	|i/(i+i)(*(i/i*i)#	|s9	
|s0s3s4s5s10s8s9	|#V=E+(i	|/(i+i)(*(i/i*i)#	|r5	
|s0s3s4s5s10s8s7	|#V=E+(F	|/(i+i)(*(i/i*i)#	|r7	
|s0s3s4s5s10s8s6	|#V=E+(T	|/(i+i)(*(i/i*i)#	|s13	
|s0s3s4s5s10s8s6s13	|#V=E+(T/	|(i+i)(*(i/i*i)#	|s8	
|s0s3s4s5s10s8s6s13s8	|#V=E+(T/(	|i+i)(*(i/i*i)#	|s9	
|s0s3s4s5s10s8s6s13s8s9	|#V=E+(T/(i	|+i)(*(i/i*i)#	|r5	
|s0s3s4s5s10s8s6s13s8s7	|#V=E+(T/(F	|+i)(*(i/i*i)#	|r7	
|s0s3s4s5s10s8s6s13s8s6	|#V=E+(T/(T	|+i)(*(i/i*i)#	|r3	
|s0s3s4s5s10s8s6s13s8s14	|#V=E+(T/(E	|+i)(*(i/i*i)#	|s10	
|s0s3s4s5s10s8s6s13s8s14s10	|#V=E+(T/(E+	|i)(*(i/i*i)#	|s9	
|s0s3s4s5s10s8s6s13s8s14s10s9	|#V=E+(T/(E+i	|)(*(i/i*i)#	|r5	
|s0s3s4s5s10s8s6s13s8s14s10s7	|#V=E+(T/(E+F	|)(*(i/i*i)#	|r7	
|s0s3s4s5s10s8s6s13s8s14s10s15	|#V=E+(T/(E+T	|)(*(i/i*i)#	|r1	(+,identifier,identifier,T1)
|s0s3s4s5s10s8s6s13s8s14	|#V=E+(T/(E	|)(*(i/i*i)#	|s19	
|s0s3s4s5s10s8s6s13s8s14s19	|#V=E+(T/(E)	|(*(i/i*i)#	|Error!	

（6）最终将中间代码输入到文件中如下图

```
(+,identifier,identifier,T1)
```

4.2 用例 2 测试

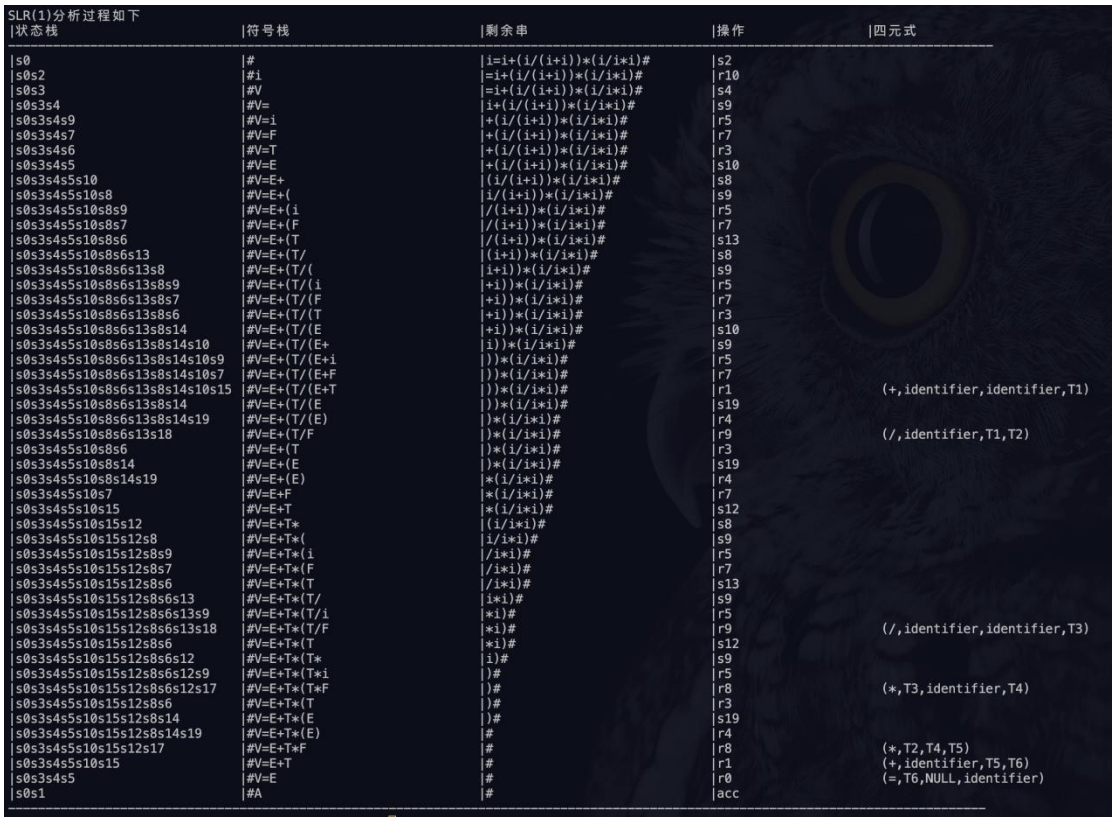
测试用例:

(identifier,p)
(=,)
(identifier,x)
(+,)
((,)
(identifier,x)
(/,)
((,)
(identifier,y)
(+,)

(identifier,z)
(),)
(),)
(*,)
((,)
(identifier,z)
(/,)
(identifier,y)
(*,)
(identifier,y)
(),)

First 集、Follow 集、SLR 分析表的构造都同 5.1。

测试结果如下图，



SLR(1)分析过程如下

状态栈	符号栈	剩余串	操作	四元式
s0	#	i=i+(i/(i+i))*(i/i*i)#	s2	
s0s2	#i	=i+(i/(i+i))*(i/i*i)#	r10	
s0s3	#V	=i+(i/(i+i))*(i/i*i)#	s4	
s0s3s4	#V=	i+(i/(i+i))*(i/i*i)#	s9	
s0s3s4s9	#V=i	+(i/(i+i))*(i/i*i)#	r5	
s0s3s4s7	#V=F	+(i/(i+i))*(i/i*i)#	r7	
s0s3s4s6	#V=T	+(i/(i+i))*(i/i*i)#	r3	
s0s3s4s5	#V=E	+(i/(i+i))*(i/i*i)#	s10	
s0s3s4s5s10	#V=E+	(i/(i+i))*(i/i*i)#	s8	
s0s3s4s5s10s8	#V=E+(	i/(i+i))*(i/i*i)#	s9	
s0s3s4s5s10s8s9	#V=E+(i	/(i+i))*(i/i*i)#	r5	
s0s3s4s5s10s8s7	#V=E+(F	/(i+i))*(i/i*i)#	r7	
s0s3s4s5s10s8s6	#V=E+(T	/(i+i))*(i/i*i)#	s13	
s0s3s4s5s10s8s6s13	#V=E+(T/	(i+i))*(i/i*i)#	s8	
s0s3s4s5s10s8s6s13s8	#V=E+(T/(	i+i))*(i/i*i)#	s9	
s0s3s4s5s10s8s6s13s8s9	#V=E+(T/(i	+i))*(i/i*i)#	r5	
s0s3s4s5s10s8s6s13s8s7	#V=E+(T/(F	+i))*(i/i*i)#	r7	
s0s3s4s5s10s8s6s13s8s6	#V=E+(T/(T	+i))*(i/i*i)#	r3	
s0s3s4s5s10s8s6s13s8s14	#V=E+(T/(E	+i))*(i/i*i)#	s10	
s0s3s4s5s10s8s6s13s8s14s10	#V=E+(T/(E+	i))*(i/i*i)#	s9	
s0s3s4s5s10s8s6s13s8s14s10s9	#V=E+(T/(E+i	))*(i/i*i)#	r5	
s0s3s4s5s10s8s6s13s8s14s10s7	#V=E+(T/(E+F	))*(i/i*i)#	r7	
s0s3s4s5s10s8s6s13s8s14s10s15	#V=E+(T/(E+T	))*(i/i*i)#	r1	(+,identifier,identifier,T1)
s0s3s4s5s10s8s6s13s8s14	#V=E+(T/(E	))*(i/i*i)#	s19	
s0s3s4s5s10s8s6s13s8s14s19	#V=E+(T/(E)	)*(i/i*i)#	r4	
s0s3s4s5s10s8s6s13s18	#V=E+(T/F	)*(i/i*i)#	r9	(/,identifier,T1,T2)
s0s3s4s5s10s8s6	#V=E+(T	)*(i/i*i)#	r3	
s0s3s4s5s10s8s14	#V=E+(E	)*(i/i*i)#	s19	
s0s3s4s5s10s8s14s19	#V=E+(E)	*(i/i*i)#	r4	
s0s3s4s5s10s7	#V=E+F	*(i/i*i)#	r7	
s0s3s4s5s10s15	#V=E+T	*(i/i*i)#	s12	
s0s3s4s5s10s15s12	#V=E+T*	(i/i*i)#	s8	
s0s3s4s5s10s15s12s8	#V=E+T*(	i/i*i)#	s9	
s0s3s4s5s10s15s12s8s9	#V=E+T*(i	/i*i)#	r5	
s0s3s4s5s10s15s12s8s7	#V=E+T*(F	/i*i)#	r7	
s0s3s4s5s10s15s12s8s6	#V=E+T*(T	/i*i)#	s13	
s0s3s4s5s10s15s12s8s6s13	#V=E+T*(T/	i*i)#	s9	
s0s3s4s5s10s15s12s8s6s13s9	#V=E+T*(T/i	*i)#	r5	
s0s3s4s5s10s15s12s8s6s13s18	#V=E+T*(T/F	*i)#	r9	(/,identifier,identifier,T3)
s0s3s4s5s10s15s12s8s6	#V=E+T*(T	*i)#	s12	
s0s3s4s5s10s15s12s8s6s12	#V=E+T*(T*	i)#	s9	
s0s3s4s5s10s15s12s8s6s12s9	#V=E+T*(T*i	)#	r5	
s0s3s4s5s10s15s12s8s6s12s17	#V=E+T*(T*F	)#	r8	(*,T3,identifier,T4)
s0s3s4s5s10s15s12s8s6	#V=E+T*(T	)#	r3	
s0s3s4s5s10s15s12s8s14	#V=E+T*(E	)#	s19	
s0s3s4s5s10s15s12s8s14s19	#V=E+T*(E)	#	r4	
s0s3s4s5s10s15s12s17	#V=E+T*F	#	r8	(*,T2,T4,T5)
s0s3s4s5s10s15	#V=E+T	#	r1	(+,identifier,T5,T6)
s0s3s4s5	#V=E	#	r0	(=,T6,NULL,identifier)
s0s1	#A	#	acc	

五、研究性教学实验感受

通过完成《编译原理》中“SLR(1)分析法的语法制导翻译及中间代码生成”的实验，我深刻体会到理论与实践结合的重要性。这次实验涵盖了从文法的 FIRST 和 FOLLOW 集计算到 SLR 分析表的构造，再到中间代码的生成整个过程，让我对编译器设计的核心环节有了更直观的认识。从理论层面上，我不仅理解了

SLR(1)文法的分析方法，还进一步掌握了语法制导翻译的原理以及四元式中间代码的生成方式。在实践过程中，通过动手设计数据结构并实现算法逻辑，我真正体会到了编译原理从抽象到具体的转化过程。

实验的完成让我学会了如何将复杂问题逐步分解并实现。从初始的分析表设计，到冲突检测与优化，再到语义动作的设置和四元式生成，我逐步完善了实验的各个环节。在此过程中，我不仅增强了编程能力，还培养了面对问题时的逻辑推理与创新思维。此外，测试用例的设计和运行结果的分析也让我意识到程序健壮性和完备性的关键作用。在不断调试、优化代码的过程中，我更加体会到持续改进的重要性

实验还带给我一种从理论实现到实际结果的成就感。当程序运行并生成正确的四元式中间代码时，我切实感受到自己对编译原理的理解在逐步深化。整个实验不仅让我在技术上有所提升，也让我更深刻地认识到编译器设计中的严谨性和创造性，这为我后续的学习和研究奠定了坚实的基础。

六、附件（源代码列表）

```
lab4/
├── in.txt
├── lexia.txt
├── out.txt
├── lab5.cpp
├── lab5
└── main.h
```

lab5.cpp

```
#include <iostream>
#include <cstring>
#include <cctype>
#include <cstdio>
#include <vector>
#include <set>
#include <map>
#include <list>
#include <fstream>
#include <iomanip>
#include <algorithm>
```

```
#include<string>
#include<iterator>
#include <sstream>
#pragma warning(disable:4996)
using namespace std;
set<string> prod; // 产生式集合
map<string, set<string> > splProd; // 分解的产生式集合
string start; // 开始符号
set<string> Vt; // 终结符集合
set<string> Vn; // 非终结符集合
set<string> VtPlus; // 终结符集合 + #
map<string, set<string> > first; // first 集
map<string, set<string> > follow; // follow 集

string ptr; // 当前词法单元对应的字符
string RecValue; // 当前词法单元对应的实际值

struct Item {
	int finish = 0;
	int pos = 3; // 圆点位置
	string prod;
};
struct status {
	int number;
	list<Item> Itemlist;
};  // 状态
list<status> DFA;
Item firstItem;   // 初始项目 I0

struct ActionNode {
	int I;   // 当前状态
	string Vt;   // 终结符
	string act;   // r||s
	int nextI_P;   // 下一状态/规约产生式编号
};
struct GotoNode {
	int I;   // 当前状态
	string Vn;   // 非终结符
	int next_I;   // 转移状态
};
list<ActionNode> ACTION;
list<GotoNode> GOTO;

struct varX {
```

```
    string X; // 符号
    string value; // 符号的值/Temp
};  // 符号
struct anaNode {
    int S;   // 状态
    varX X;   // 符号
};
vector<anaNode> stack; // 分析栈(符号栈和状态栈)
vector<string> leftString; // 剩余输入串
vector<string> temp; // 临时记录输入串
vector<string> recValue; // 临时记录实际输入串
vector<string> recleftValue; // 剩余实际输入串

struct quaterExp {
    string op;
    string arg1;
    string arg2;
    string result;
};
list<quaterExp> Exp; // 四元式
list<int> quaterI; // 可产生四元式的状态
int recTemp = 0; // 记录 result 编号
```

main.h

```
#include <iostream>
#include <cstring>
#include <cctype>
#include <cstdio>
#include <vector>
#include <set>
#include <map>
#include <list>
#include <fstream>
#include <iomanip>
#include <algorithm>
#include<string>
#include<iterator>
#include <sstream>
#pragma warning(disable:4996)
using namespace std;
set<string> prod; // 产生式集合
map<string, set<string> > splProd; // 分解的产生式集合
```

```
string start; // 开始符号
set<string> Vt; // 终结符集合
set<string> Vn; // 非终结符集合
set<string> VtPlus; // 终结符集合 + #
map<string, set<string> > first; // first 集
map<string, set<string> > follow; // follow 集

string ptr; // 当前词法单元对应的字符
string RecValue; // 当前词法单元对应的实际值

struct Item {
    int finish = 0;
    int pos = 3; // 圆点位置
    string prod;
};
struct status {
    int number;
    list<Item> Itemlist;
};  // 状态
list<status> DFA;
Item firstItem;   // 初始项目 I0

struct ActionNode {
    int I;   // 当前状态
    string Vt;   // 终结符
    string act;   // r|s
    int nextI_P;   // 下一状态/规约产生式编号
};
struct GotoNode {
    int I;   // 当前状态
    string Vn;   // 非终结符
    int next_I;   // 转移状态
};
list<ActionNode> ACTION;
list<GotoNode> GOTO;

struct varX {
    string X; // 符号
    string value; // 符号的值/Temp
};  // 符号
struct anaNode {
    int S;   // 状态
    varX X;   // 符号
};
```

```
vector<anaNode> stack; // 分析栈(符号栈和状态栈)
vector<string> leftString; // 剩余输入串
vector<string> temp; // 临时记录输入串
vector<string> recValue; // 临时记录实际输入串
vector<string> recleftValue; // 剩余实际输入串

struct quaterExp {
	string op;
	string arg1;
	string arg2;
	string result;
};
list<quaterExp> Exp; // 四元式
list<int> quaterI; // 可产生四元式的状态
int recTemp = 0; // 记录 result 编号
```