编译原理实验指导书

《编译原理》课程主要讲述编译程序各部件（词法分析、语法分析、语义分析、中间代码生成等）的设计原理和实现技术，为使学生进一步理解掌握课程内容，提高学生运用理论知识解决问题的能力，课程实验以 TEST 语言为源语言，通过实验教学环节构造一个小型编译程序。

1 实验语言

课程实验要求学生对用 TEST 语言编写的源程序进行词法分析、语法分析、语义分析并生成中间代码。

1.1 TEST 语言概述

TEST 语言的程序结构简单，整个程序相当于 C 语言的函数体，即由一对花括号{}括起的语句序列，没有函数，没有数组；声明语句、表达式语句以及控制语句和 C 语言类似。数据类型只有整型，一个声明语句只能声明一个变量；表达式只有算术表达式和布尔表达式两种形式，其中算术表达式为整型变量和整型常数的四则运算组合，布尔表达式为两个算术表达式的比较运算；控制语句只有 if、while 和 for 三种语句；支持复合语句；read 语句和 write 语句实现输入输出；仅支持多行注释/* …… */。

1.2 TEST 语言的词法规则

1、标识符：字母打头，后接任意字母或数字。

2、保留字：标识符的子集，包括 if, else, for, while, int, write, read。

3、无符号整数：由数字组成，但最高位不能为 0，允许一位的 0。

4、分界符：(、)、;、{、}、,

5、运算符：+、-、*、/、=、<、>、>=、<=、!=、==

6、注释符：/*　　*/

1.3 TEST 语言的语法规则

1) <program> → {<declaration_list><statement_list>}

2) <declaration_list> → <declaration_list><declaration_stat> | ε

3) <declaration_stat> → int ID;

4) <statement_list> → <statement_list><statement>| ε

5) <statement> → <if_stat>|<while_stat>|<for_stat>|<read_stat>
|<write_stat>|<compound_stat> |<assignment_stat>|;

6) <if_stat> → if (<bool_expression >) <statement >
| if (<bool_expression>) <statement >else < statement >

7) <while_stat> → while (<bool_expression>) < statement >

8) <for_stat> → for (<assignment_expression>; <bool_expression>;
<assignment_ expression >)<statement>

9) <write_stat> → write < arithmetic_expression >;

10) <read_stat> → read ID;

11) <compound_stat> → {<statement_list>}

12) <assignment_expression> → ID=<arithmetic_expression>

13) <assignment_stat> →<assignment_expression>;

14) <bool_expression>→<arithmetic_expression> > <arithmetic_expression>
|<arithmetic_expression> < <arithmetic_expression>
|<arithmetic_expression> >= <arithmetic_expression>
|<arithmetic_expression> <= <arithmetic_expression>
|<arithmetic_expression> == <arithmetic_expression>
|<arithmetic_expression> != <arithmetic_expression>

15) <arithmetic_expression> → <arithmetic_expression>+<term>
|< arithmetic_expression>-<term>
|< term >

16) < term > → < term >*<factor>|< term >/<factor>|< factor >

17) < factor > → (<arithmetic_expression>)|ID|NUM

2 目标机器

本课程实验中间代码生成在一台栈式模拟机进行，本实验指导书称之为TEST语言抽象机，其指令系统各指令及含义描述如下：

LOAD D 将内存单元D中的内容加载到操作数栈。

LOADI 常量 将常量压入操作数栈。

STO D　将操作数栈栈顶单元内容存入内存单元D，且栈顶单元内容保持不变。

ADD　　将次栈顶单元与栈顶单元内容出栈并相加，和置于栈顶。

SUB　　将次栈顶单元减去栈顶单元内容并出栈，差置于栈顶。

MULT　将次栈顶单元与栈顶单元内容出栈并相乘，积置于栈顶。

DIV　　将次栈顶单元与栈顶单元内容出栈并相除，商置于栈顶。

BR　lab　无条件转移到标号 lab。

BRF　lab　检查栈顶单元逻辑值，若为假(0)则转移到标号 lab。

EQ　　将栈顶两单元做等于比较，并将结果 1 或 0 置于栈顶。

NOTEQ　将栈顶两单元做不等于比较，并将结果 1 或 0 置于栈顶。

GT　　次栈顶大于栈顶操作数，则栈顶置 1，否则置 0。

LES　　次栈顶小于栈顶操作数，则栈顶置 1，否则置 0。

GE　　次栈顶大于等于栈顶操作数，则栈顶置 1，否则置 0。

LE　　次栈顶小于等于栈顶操作数，则栈顶置 1，否则置 0。

AND　　将栈顶两单元做逻辑与运算，并将结果 1 或 0 置于栈顶。

OR　　将栈顶两单元做逻辑或运算，并将结果 1 或 0 置于栈顶。

NOT　　将栈顶的逻辑值取反。

IN　　从标准输入设备(键盘)读入一个整型数据，并入操作数栈。

OUT　　将栈顶单元内容出栈，并输出到标准输出设备上(显示器)。

STOP　停止执行。

3 实验题目

课程实验要求对 TEST 语言完成以下实验题目：设计词法分析程序、设计语法分析程序、分析调试语义分析及中间代码生成程序。

3.1 题目一：设计词法分析程序

3.1.1 实验类型

设计型实验，4 学时（2 学时完成 DFA 的设计；2 学时完成程序的编写、调试、测试）

3.1.2 实验目的

通过设计、编写、调试词法分析程序，达到以下目的：

1、 会用正则表达式描述词法规则；

2、 会将正则表达式转化为 NFA；

3、 会将 NFA 确定化并化简为最小的 DFA；

4、 会确定词法分析结果的输出形式；

5、 掌握词法分析程序设计的基本流程。

3.1.3 实验知识

1、词法分析程序的设计流程

（1）用正则表达式描述程序设计语言的词法规则；一个正则表达式对应一条词法规则。

（2）为每个正则表达式构造一个 NFA，用来识别正则表达式描述的单词。

（3）将多个 NFA 合并为一个 NFA。

（4）将 NFA 转换成等价的 DFA。

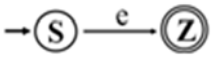
（5）化简 DFA。

（6）确定单词的输出形式。

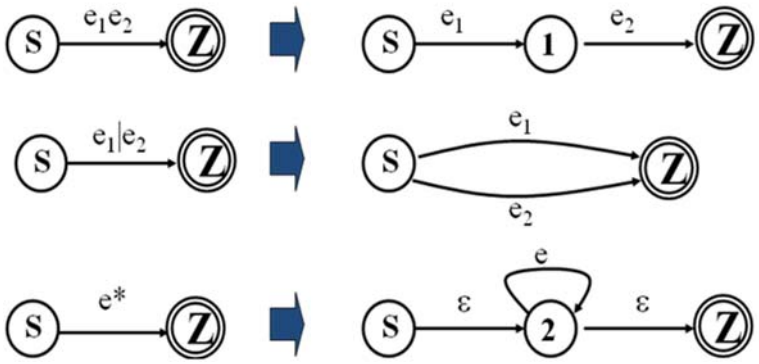
（7）DFA＋单词输出形式⇒构造词法分析程序

2、正则表达式到 NFA 的转换

（1）对正则表达式 e，构造一个广义的 NFA，该 NFA 只有初态和终态，初态到终态的连接弧上为正表达式 e， 如下图所示：



（2）按照一定的分解规则，在 NFA 上将正则表达式 e 进行分解，增加结点，直到所有连接弧上只有单个符号或者 ε 为止，具体规则如下：



3、NFA 确定化为 DFA

（1）画一张具有 n+1 列的矩阵表(n=|∑|)，第一行的分别填上 I，I_a，I_b，I_c，…，其中 a，b，c…∈∑。

（2）令 I= ε -CLOSURE（S_0）。S_0：NFA 的初态； ε -CLOSURE(S_0) = $S_0 \cup S_\varepsilon$，S_ε= {s|从 S_0 出发经过任意条 ε 弧可到达的状态 s}

（3）把 I 填入矩阵表的 I 列

（4）计算 I_a，I_b，I_c，…，并填入相应的列。

I_a= ε -CLOSURE(J_a)，J_a = {s|从 I 的所有状态出发经过一条 a 弧可到达的状态 s}

（5）若 J∈{I_a，I_b，I_c，…}未在 I 列出现，则令 I=J。重复 3-5 直到所有的 J 均在 I 列中出现。

（6）把矩阵表中各子集作为状态，并重新命名。

（7）确定终态和初态：初态：I 列的第一个元素；终态：含有原 NFA 任一终态的子集。

（8）画出相应的 DFA

4、化简 DFA

将 DFA M= （Q，∑，δ，q_0，F）最小化的算法如下：

（1）将 Q 划分成终态集合 S_1和非终态集合 S_2：Q= $S_1 \cup S_2$

（2）对各状态集合进行划分。设第 m 次划分将 Q 分成 n 个状态集合，即 Q= $S_1 \cup S_2 \cup \cdots \cup S_n$。 对状态集合 S_i(i=1,2,…,n)中的各个状态逐一检查，设状态 q_1和 q_2是状态集合 S_i 中的两个状态，对于**所有输入符号** a(a∈Σ)，有 δ(q_1,a)= S′ 、δ(q_2,a)=S″，若状态 S′ 和 S″ 属于同一状态集合，则将状态 q_1和 q_2放在同一状态集合中；否则，将状态 q_1和 q_2放到不同的两个集合中。

（3）重复第 2 步，直到所有状态集合都不能划分。此时，每个状态集合中的状态都是等价的。

（4）将每个状态集合中的等价状态合并成一个等价代表状态。将状态函数中的所有状态用相应的等价代表状态表示。

（5） 删除多余状态。

3.1.4 实验内容

1、根据 TEST 语言的词法规则，分别写出每条词法规则对应的正则表达式；

2、将每一个正则表达式转换为 NFA；

3、将多个 NFA 合并后进行确定化并化简；

4、根据化简后的 DFA 画出流程图；

5、参阅在实验语言部分给出的 TEST 语言语法规则，确定单词分类、单词输出方案；

6、编写词法分析程序；

7、对下面的 TEST 语言源程序进行词法分析，将合法单词存入 lex.txt，并报告词法错误及其位置。**注：不能修改以下源程序。**

```
{
/*This a test program.*/
int abc;
int 123;
int A$@;
int i;
int n;
int b,c;
int 2a;
int a2;
read n;
n = 012345;
for (i=1;i<=n; i= i+1)
{
abc=abc+i;
}
if(i!=n) n = n+i;
if (!n) b = b+c;
/*The loop ended
write abc;
}
```

3.1.5 实验报告要求

1、实验设计

（1）DFA 设计过程：给出每条词法规则对应的正则表达式、由正则表达式构造的 NFA、合并后的 NFA、确定化后的 DFA、化简后的 DFA。

（2）词法分析结果的输出形式：给出单词类别。

2、实验过程

（1）简述完成实验的步骤。

（2）实验调试记录：实验过程中遇到的问题，对该问题进行描述，分析其产生的原因，提出解决方案，给出最后的解决结果。

3、实验结果

（1）给出词法分析程序识别的单词。

（2）给出词法分析程序识别出的词法错误。

4、讨论与分析

（1）通过实验对词法分析相关知识点的理解。

（2）回答实验思考部分提出的问题。

3.1.6 实验思考

1、设计的词法分析程序是否满足最长匹配原则？如果满足请给出实现方案。如果不满足请给出改进方案。

7、给出单词分类方案，并说明理由。

3、构建词法分析程序一般过程是怎样的？

3.2 题目二：设计语法分析程序

3.2.1 实验类型

设计型实验，6 学时(2 学时完成文法改造； 2 学时完成程序编码；2 学时完成程序测试)

3.2.2 实验目的

采用递归下降分析方法设计 TEST 语言的语法分析程序，达到以下目的：

1、会判别一个文法是否是 LL(1)文法；

2、会对上下文无关文法进行改造，使之满足确定的自顶向下分析要求。

3、掌握递归下降分析程序的编写方法。

4、会采用超前读单词方法对不能改造的产生式进行特殊处理。

3.2.3 实验知识

1、递归下降语法分析程序的设计流程

（1）改造文法，消除左递归、提取左因子。

（2）计算产生式右部符号串的 FIRST 集及非终结符的 FOLLOW 集。

（3）检查是不是 LL(1) 文法，若不是 LL(1)文法，对不满足条件的产生式进行特殊处理。

（4）根据产生式设计相应的程序。

2、LL(1)文法

满足以下三个条件的文法称为 LL(1)文法。

（1）文法不含左递归。

（2）文法的任何一个非终结符 A 的各个产生式的右部符号串的 FIRST 集两两不相交，即：

若有 $A\rightarrow\alpha_1|\alpha_2|\ldots|\alpha_n$ ，则 $FIRST(\alpha_i)\cap FIRST(\alpha_j)=\Phi \quad (i\neq j,\ 1\leq i,j\leq n)$

（3)文法的任何一个非终结符 A,若有 $A\rightarrow\alpha_1|\alpha_2|\ldots|\alpha_n$ 且存在 α_i，有 $\varepsilon\in FIRST(\alpha_i)$，则 $FIRST(\alpha_j)\cap FOLLOW(A)=\Phi \quad (i\neq j,\ j=1,2,\ldots,n)$

3、文法改造

（1）消除左递归

A、对有左递归的产生式：$A\rightarrow A\alpha_1|A\alpha_2|\ldots|A\alpha_m|\beta_1|\beta_2|\ldots|\beta_n$，可以引入新非终结符，改写为右递归形式：

$A \rightarrow \beta_1 A' | \beta_2 A' | \ldots | \beta_n A'$

$A' \rightarrow \alpha_1 A' | \alpha_2 A' | \ldots | \alpha_m A' | \varepsilon$

B、对有左递归的产生式：$A \rightarrow A\alpha_1 | A\alpha_2 | \ldots | A\alpha_m | \beta_1 | \beta_2 | \ldots | \beta_n$，可采用 EBNF 的元符号“{}”，改写为：

$A \rightarrow \beta_1 | \beta_2 | \ldots | \beta_n \{\alpha_1 | \alpha_2 | \ldots | \alpha_m\}$

注：该方式仅限于递归下降分析使用。

（2）提取左公因子

对有左公因子的产生式：$A \rightarrow \alpha\beta_1 | \alpha\beta_2 \ldots\ldots | \alpha\beta_n | \gamma_1 | \gamma_2 \ldots\ldots | \gamma_m$，可改写为：

$A \rightarrow \alpha A' | \gamma_1 | \gamma_2 \ldots\ldots | \gamma_m$

$A' \rightarrow \beta_1 | \beta_2 \ldots\ldots | \beta_n$

4、递归下降分析程序的基本架构

（1）命名约定：变量 ch 为当前符号；函数 GetNextSymbol 读输入串下一个单词符号；

（2）非终结符函数结构：若非终结符具有 $U \rightarrow u_1 | u_2 | \ldots | u_n$ 的产生式，其函数处理架构如下：

```
U( )
{ //ch 为当前符号
   if(ch∈FIRST(u1))          处理 u1 的程序部分;
   else if(ch∈FIRST(u2))     处理 u2 的程序部分;
   …
  else if(ε∈FIRST(U) && ch∈FOLLOW(U)) return;
       else
      error();
 }
```

（3）对于每个右部 $u_1 \rightarrow x_1 x_2 \ldots x_n$ 的处理架构如下：

处理 x_1 的程序；

处理 x_2 的程序；

…… ……

处理 x_n 的程序；

注：如果右部为空，则不处理，直接 return。

（4）对于右部中的每个符号 x_i，如果 x_i 为终结符号，则

if(x_i == 当前符号)

{

ch = GetNextSymbol();

}

else 出错处理

如果 x_i 为非终结符号，直接调用相应的过程 x_i( )。

（5）语法分析程序的主程序

假设开始符号对应的函数为 S( )，则主程序为

```
{
    /*首先读入一个符号，以满足约定*/
     ch = GetNextSymbol();
    S( );
}
```

3.2.4 实验内容

1、消除 1.3 节 TEST 语言的文法中的左递归和左公因子。

2、判别改造后的文法是否满足 LL(1)文法的条件，对不能满足条件的非终结符采用特殊处理方法。

3、编写递归下降语法分析程序。

4、使用下述的 TEST 语言代码测试编写的递归下降分析程序（先使用实验一的词法分析程序进行词法分析）。

```
{
  int a;
  int i;
  int 2b;
  int c
  for (i=1; i <= 10 i=i+1)
  {
  ;
  a=a+i
  b=b*i;
  {
```

```
c=a+b;
}
while(x<=)
{
C=a+b+(x*y;

if (a>b)
{
read a;
}
else
{
write b+5;
}
write c
}
```

3.2.5. 实验报告要求

1、实验设计

（1）给出对哪些产生式进行了改造及其改造结果。

（2）给出改造后所有产生式右部符号串的 FIRST 集和非终结符的 FOLLOW 集。

（3）给出不能满足递归下降分析的产生式及采取的特殊处理方案

2、实验过程

（1）简述完成实验的步骤。

（2）实验调试记录：实验过程中遇到的问题，对该问题进行描述，分析其产生的原因，提出解决方案，给出最后的解决结果。

3、实验结果

采用发现一个语法就停止分析的方式，给出测试结果。

4、讨论与分析

（1）通过实验对词法分析相关知识点的理解。

（2）回答实验思考部分提出的问题

3.2.6 实验思考

1、TEST 语言语法规则中哪些不满足无回溯的递归下降分析条件？应该如何处理？

2、所有文法都可以改写为满足递归下降分析条件的文法吗？如果不能，请给出一个反例。

3、改写文法有什么弊端？

3.3 题目三：分析调试语义分析及中间代码生成程序

3.3.1 实验类型

验证型实验。 6 学时(2 学时分析理解语义分析及中间代码生成程序；2 学时分析理解抽象机模拟程序；2 学时调试、测试前面两个程序)

3.3.2 实验目的

通过分析调试 TEST 语言的语义分析和中间代码生成程序，到达以下目的：

（1）理解对语法制导翻译思想。

（2）掌握符号表在语义分析和中间代码生成阶段的作用

（3）理解属性翻译文法的构造方法，阐释每个产生式上的符号属性及语义动作的含义。

3.3.3 实验知识

1、语法制导翻译

语法制导翻译的基本思想：在产生式的不同位置，根据语义分析和中间代码生成需要，配上多个语义子程序以描述产生式所对应的翻译工作。这些工作包括：操作符号表、报告语义错误、生成中间代码、修改编译程序某些工作变量的值等。在语法分析过程中，使用产生式进行推导或归约时，同时执行就调用产生式上的语义子程序，以完成相应的翻译任务。

2、属性翻译文法

属性翻译文法可用三元组表示：A=(G，V，F)，其中：

G：一个上下文无关文法，描述程序语言的语法规则，属性翻译文法的基础。

V：有穷的属性集，每个属性隶属于文法 G 的某个符号，这些属性代表与文法符号相关的语义信息，如：类型、地址、值、代码等。属性与变量一样，可以进行计算和传递。属性有两类：

综合属性：归约型属性，用于“自下而上”传递信息。

继承属性：推导型属性，用于“自上而下”传递信息。

F：属性的计算规则，也称为语义规则。语义规则与一个产生式相联，只引用该产生式左端或右端的终结符或非终结符相关联的属性。

3、TEST 语言的属性翻译文法

（1）约定：TEST 语言的语法规则中所有符号具有 3 个继承属性：

vartablep：指向符号表最后一个记录的下一个位置，即第一个空白记录位置；

datap：指向可分配的地址空间首部（可分配地址空间是一个连续的区域），初始值为 0；

labelp：指向控制语句转向的语句。

在 TEST 语言的属性翻译文法中省略此三个属性以保持简洁。

程序实现时将代表 vartablep,datap,lablep 这 3 个继承属性的变量定义成同名的整型变量，并设为全局变量，初值均为 0。

```
int vartablep=0, datap=0, labelp=0;
```

显然文法的开始符号具有这 3 个继承属性的初始值。

（2）声明语句

< declaration_stat > →int ID$_{\uparrow n}$@name-def$_{\downarrow n}$;

其中属性 n 表示标识符名。语义动作含义如下

@name-def$_{\downarrow n}$：查询符号表，若变量 n 未定义则填入符号表，并分配内存；若已定义则报告变量重复定义。

（3）因子

<factor>→(<arithmetic_expression >)|ID$_{\uparrow n}$@LOOK$_{\downarrow n\uparrow d}$@LOAD$_{\downarrow d}$|NUM$_{\uparrow i}$@LOADI$_{\downarrow i}$

其中属性 n 表示标识符名；d 表示该变量的内存地址；i 表示常量值。

其中各语义动作含义如下：

@LOOK$_{\downarrow n\uparrow d}$：根据变量名 n 查符号表，给出变量地址 d; 若没有则变量没定义。

@LOAD$_{\downarrow d}$：输出指令代码 LOAD d。

@LOAD$_{\downarrow i}$：输出指令代码 LOADI i。

（4）项

< term > → < term >*<factor>|< term >/<factor>|< factor >

采用 EBNF 表示法改为：

< term >→<factor>{(*< factor > | /< factor >}

添加语义动作得到：

< term >→<factor>{(*< factor >@MULT | /< factor >@DIV)}

各语义动作含义如下：

@MULT：输出指令代码 MULT。

@DIV：输出指令代码 DIV。

（5）<arithmetic_expression> → <arithmetic_expression>+<term>

|< arithmetic_expression>-<term>|< term >

采用 EBNF 表示法改为：

<arithmetic_expression> →<term>{(+< term > |-<term>}

添加语义动作得到：

<arithmetic_expression> →<term>{(+< term >@ADD |-<term>@SUB)}

各语义动作含义如下：

@ADD：输出指令代码 ADD。

@SUB：输出指令代码 SUB。

（6）布尔表达式

<bool_expression>→<arithmetic_expression> > <arithmetic_expression>@GT

|<arithmetic_expression> < <arithmetic_expression>@LES

|<arithmetic_expression> >= <arithmetic_expression>@GE

|<arithmetic_expression> <= <arithmetic_expression>@LE

|<arithmetic_expression> == <arithmetic_expression>@EQ

|<arithmetic_expression> != <arithmetic_expression>@NOTEQ

各语义动作含义如下：

@GT：输出指令代码 GT。

@LES：输出指令代码 LES。

@GE：输出指令代码 GE

@LE：输出指令代码 LE

@EQ：输出指令代码 EQ

@NOTEQ：输出指令代码 NOTEQ

（7）赋值语句

< assignment_expression >→ID↑n@LOOK↓n↑d=<arithmetic_expression >@STO↓d @POP

<assignment_stat> →<assignment_expression>;

其中属性 n 表示标识符名；d 表示该变量的内存地址。

@LOOK↓n↑d:根据变量名 n 查符号表，给出变量地址 d; 若没有则变量没定义。

@STO↓d：输出指令代码 STO d

@POP：输出指令代码 POP

（8）if 语句

<if_stat> → if (<bool_expression >) <statement > | if (<bool_expression>)
<statement >else < statement >

采用 EBNF 表示法改为：

<if_stat>→if(<bool_expression>)@BRF$_{\uparrow label1}$<statement> @BR$_{\uparrow label2}$
@SETlabel$_{\downarrow label1}$[else<statement>]@SETlabel$_{\downarrow label2}$

各语义动作含义如下：

@BRF$_{\uparrow label1}$ ：输出 BRF label1

@BR$_{\uparrow label2}$：输出 BR label2

@SETlabel$_{\downarrow label1}$：设置标号 label1

@SETlabel$_{\downarrow label2}$：设置标号 label2

（9）while 语句

<while_stat>→ while @SETlabel$_{\uparrow label1}$ (<bool_expression >) @BRF$_{\uparrow label2}$
< statement >@BR$_{\downarrow label1}$ @SETlabel$_{\downarrow label2}$

各语义动作含义如下：

@SETlabel$_{\downarrow label1}$：设置标号 label1

@BRF$_{\uparrow label2}$ ：输出 BRF label2

@BR$_{\uparrow label1}$：输出 BR label1

@SETlabel$_{\downarrow label2}$：设置标号 label2

（10）for 语句

<for_stat>→for (<assignment_expression >;
@SETlabel$_{\uparrow label1}$< bool _expression > ; @BRF$_{\uparrow label2}$@BR$_{\uparrow label3}$
@SETlabel$_{\uparrow label4}$ < assignment_expression >@BR$_{\downarrow label1}$)
@SETlabel$_{\downarrow label3}$ <statement>@BR$_{\downarrow label4}$@SETlabel$_{\downarrow label2}$

各语义动作含义如下：

@SETlabel$_{\downarrow label1}$:设置标号 label1

@BRF$_{\uparrow label2}$:输出 BRF label2

@BR$_{\uparrow label3}$:输出 BR label3

@SETlabel$_{\downarrow label4}$:设置标号 label4

@BR$_{\uparrow label1}$：输出 BR label1

@SETlabel$_{\downarrow label3}$：设置标号 label3

@BR$_{\uparrow label4}$：输出 BR label4

@SETlabel$_{\downarrow label2}$：设置标号 label2

（11）write 语句

<write_stat>→write < arithmetic_expression >;@ OUT@POP

各语义动作如下：

@OUT：输出指令代码 OUT。

@POP：输出指令代码 POP。

（12）read 语句。

<read_stat>→read ID$_{\uparrow n}$ LOOK$_{\downarrow n\uparrow d}$ @IN @STO$_{\downarrow d}$@POP;

其中属性 n 表示标识符名；d 表示变量内存地址。

@LOOK$_{\downarrow n\uparrow d}$: 根据变量名 n 查符号表，给出变量地址 d; 若没有则变量没定义。

@IN：输指令代码 IN。

@STO$_{\downarrow d}$：输出指令代码 STO d。

@POP：输出指令代码 POP。

3.3.4 实验内容

1、根据 TEST 语言的属性翻译文法，分析、修改、调试 TEST 语言的语义分析及中间代码生成程序。

2、熟悉本实验指导书第二部分的 TEST 语言抽象机的指令系统

3、分析调试 TEST 抽象机模拟器程序

4、通过前 3 步工作后，用 TEST 语言编写程序实现以下功能：

从键盘输入 1-10 并计算输出其乘积与和。

5、将第 4 步的 TEST 源程序编译得到中间代码并在抽象机上运行，能得到正确的输出结果。

3.3.5 实验报告要求

1、实验知识

（1）简述语法制导翻译的基本思想

（2）给出 TEST 语言的语法制导翻译方案

2、实验过程

（1）对语义分析程序的调试（修改了哪些代码，为什么修改，怎么修改的）

（2）在语义分析程序中怎么检查变量是否赋初值的？给出完整的方案

（3）如何调试抽象机代码的？（修改了哪些代码，为什么修改，怎么修改的）

3、实验结果

（1）用 TEST 语言编写的源程序代码。

（2）生成的中间代码。

（3）中间代码在抽象机上的运行结果。

（4）反映对变量否赋初值的测试（可以用 TEST 语言编写代码，并给出测试结果）

4、讨论与分析

（1）通过实验对课程知识点的理解；

（2）回答实验指导书的实验思考提出的问题。

3.3.6 实验思考

1、实验给出的语义分析及中间代码生成程序中的符号表管理方案存在什么问题？提出改进方案。

2、阐释非终结符<declaration_stat>、<factor>的产生式上所添加的动作含义。

3、抽象机模拟程序如何进行指令计数？

4、如果要检查变量在参与运算时是否有值，应当如何处理？给出解决方案。