清华大学本科生考试试题专用纸

考试课程 《编译原理》 （样题） 06 级 07 级试题合并而成

（以 07 级试题为基础，第九题改为 06 级试题，附加了 06 级的第一题）

学号：_____ 姓名：_____

一. （07级）

1.

对于右图中的 Decaf/Mind 程序，根据第二阶段实验建立符号表的过程，当处理到图中所标出的 for 语句时，当前作用域栈中含哪些开作用域？它们对应的符号表中分别包含哪些符号？

```
class Computer {
   int cpu;
   void Crash(int numTimes) {
      int i;
      for (i = 0; i < numTimes; i = i + 1)
         Print("sad\n");
   }
}

class Mac extends Computer {
   int mouse;
   void Crash(int numTimes) {
      Print("ack!");
   }
}

class Main {
   static void main() {
      class Mac powerbook;
      powerbook = new Mac();
      powerbook.Crash(2);
   }
}
```

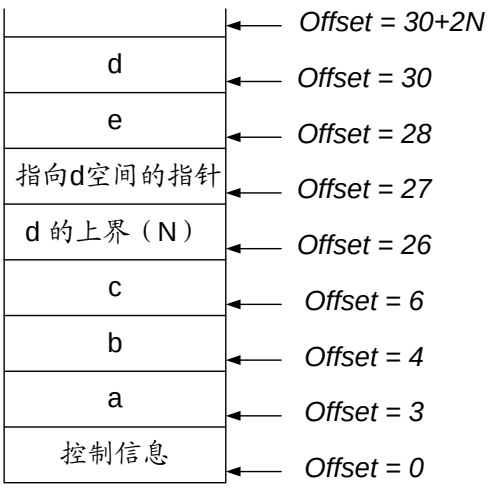
2. 对于上图（第1小题图）中的 Decaf/Mind 程序，根据课程实验所采取的运行时存储组织方式，当 powerbook 所指向的对象创建后，其对象存储空间中含有哪些内容？class Mac 的 vtable 中包含哪些内容？

3.

若按照某种运行时存储组织方式，如下函数 p 被激活时的过程活动记录如右图所示。其中 d 是动态数组。

```
static int N;

void p( int a)   {
    float b;
    float c[10];
    float d[N];
    float e;
    …
}
```



试指出函数 p 中访问 d[i]（$0 \le i < N$）时相对于活动记录基址的 *Offset* 值如何计算？

若将数组 c 的声明改为 float c[N]，那么你将如何修改函数 p 活动记录的组织（说明一种可行方案的思路即可）？

一. （06级）

1. PL0编译器的符号表采用一个全局的单符号表栈结构。对于下图左边的PL0程序，当PL0编译器在处理到第 L 行时符号表栈中的符号有哪些？

```
(1)  var a,b;
(2)  procedure  p ;
(3)       var x;
(4)        procedure  r ;
(5)            var x, y;
(6)            begin
(7)                a := 0;
(8)                if   a < b  then call q;
 .                 ……
 .            end;
 .       begin
 .            call r ;
 .            ……
 .       end ;
 .   procedure  q ;
 .       var x;
 .       begin
(L)            if   a < b  then call p ;
 .             ……
 .       end ;
 .   begin
 .       a := 1;
 .       b := 2;
 .       call q;
 .       ……
 .   end .
```

25	x	
24	?	RA
23		DL
22		SL
21		
20	?	RA
19		DL
18		SL
17	y	
16	x	
15	?	RA
14	9	DL
13	9	SL
12	x	
11	?	RA
10	5	DL
9	0	SL
8	x	
7	?	RA
6	0	DL
5	0	SL
4	b	
3	a	
2	?	RA
1	0	DL
0	0	SL

2. 对于PL0的栈式动态存储分配（过程活动记录中的控制信息包括静态链 SL，动态链 DL，以及返回地址 RA），上图左边的 PL0 程序执行到过程 p 被第二次激活时，运行栈的当前状态如上图右边所示（栈顶指向单元26）。试补齐该运行状态下，单元18、19、21、22、及 23 中的内容。

3. 针对静态作用域规则和动态作用域规则，分别指出上图左边的 PL0 程序执行到过程 q 被第二次激活时，第 L 行代码中所访问的变量 a 的值各是多少？

二.（07级）

1. 设有如下无二义文法G[*S*]：

$$S \to A\,b \mid A\,B\,c$$
$$A \to a\,A \mid a$$
$$B \to b$$

试给出句型 $a\,A\,b\,c$ 的所有短语、直接短语和句柄；即填充下表：

句型	短语	直接短语	句柄
a A b c			

2. 给定文法 *G*[*S*]：

$S \to AB$
$A \to \varepsilon \mid aAbB$
$B \to cA$

下表中已给出针对文法 G[E] 各产生式右部文法符号串的 First 集合，以及各产生式左部非终结符的 Follow 集合。试求出各产生式的预测集合PS（即完成下表），并指出文法 *G*[*S*] 是否 *LL*(1) 文法？

G 中的规则 *r*	*First* (*rhs*(*r*))	*Follow* (*lhs*(*r*))	*PS* (*r*)
$S \to AB$	*a*，*c*	#	
$A \to \varepsilon$	ε	*b*,*c*,#	
$A \to aAbB$	*a*	*b*,*c*,#	
$B \to cA$	*c*	*b*,*c*,#	

表中的 *rhs(r)* 表示产生式 *r* 右部的文法符号串，*lhs(r)* 表示产生式 *r* 左部的非终结符。

三（07级）

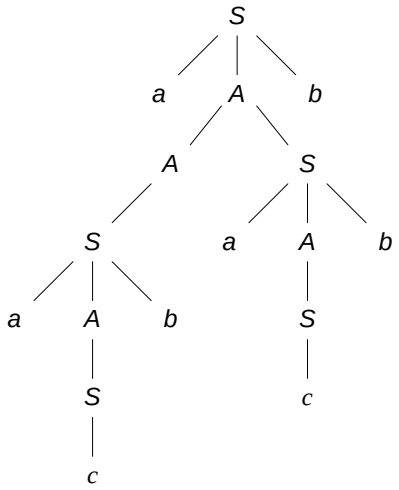
1. 给定 SLR(1) 文法 G[S]：

$S \to a\ A\ b$
$S \to c$
$A \to A\ S$
$A \to S$

如下是基于G[S]的一个S 属性文法/翻译模式：

$S \to a\ A\ b$ { $S.x := A.x + 1$ } // := 为赋值号
$S \to c$ { $S.x := 0$ }
$A \to A_1\ S$ { $A.x := A_1.x + S.x$ }
$A \to S$ { $A.x := S.x$ }

(1) 输入串 *aacbacbb* 的一棵语法分析树如下图所示。试针对该语法分析树进行标注（即得到一棵带标注的语法分析树，标出分析树中各结点所对应文法符号的属性值，本题中的终结符不对应属性值）。



(2) 如果在 LR 分析过程中根据该翻译模式进行自下而上翻译，试写出在按每个产生式归约时语义处理的一个代码片断（设语义栈由向量 v 表示，归约前栈顶位置为 top，终结符不对应语义值，而每个非终结符的综合属性都只对应一个语义值，可用 v[i].*x* 表示；不用考虑对 top 的维护）。

2. 变换如下翻译模式，使嵌在产生式中间的语义动作集中仅含复写规则，并使得在自底向上的语法分析过程中，文法符号的所有继承属性均可以通过归约前已出现在分析栈中的确定的综合属性进行访问：

$D \rightarrow D_1$; T { $L.type := T.type$; $L.offset := D_1.width$; $L.width := T.width$ } L
{$D.width := D_1.width + L.num \times T.width$ }
$D \rightarrow T$ { $L.type := T.type$; $L.offset := 0$; $L.width := T.width$ } L
{$D.width := L.num \times T.width$ }
$T \rightarrow$ integer { $T.type := int$; $T.width := 4$ }
$T \rightarrow$ real { $T.type := real$; $T.width := 8$ }
$L \rightarrow$ { $L_1.type := L.type$; $L_1.offset := L.offset$; $L_1.width := L.width$; } L_1 , id
{ *enter* (id.*name*, $L.type$, $L.offset + L_1.num \times L.width$) ; $L.num := L_1.num + 1$ }
$L \rightarrow$ id { *enter* (id.*name*, $L.type$, $L.offset$) ; $L.num := 1$}

四 （07级）如下是以 LL(1) 文法 G[S] 作为基础文法设计的翻译模式，试针对该翻译模式构造一个自上而下的递归下降（预测）翻译程序：

$S \rightarrow F$ { $P.i := F.val$ } P { *print*($P.s$) } // := 为赋值号
$P \rightarrow + F$ { $P_1.i := P.i + F.val$ } P_1 { $P.s := P_1.s$ }
$P \rightarrow \varepsilon$ { $P.s := P.i$ }
$F \rightarrow a$ { $F.val := 1$ }

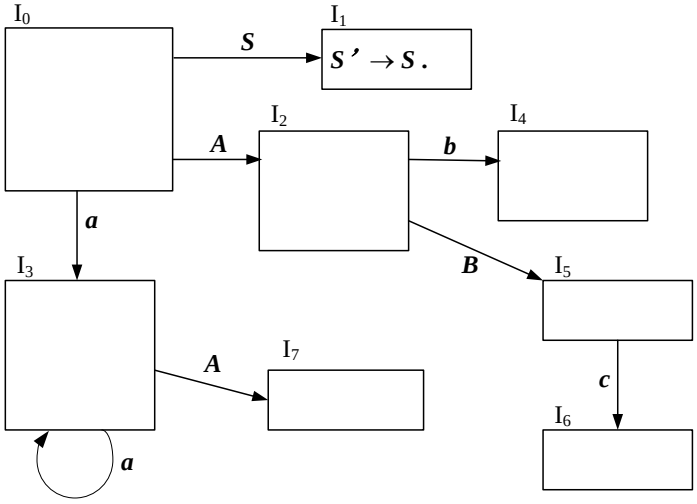
（可以直接使用讲稿中的 MatchToken 函数）

五 （07级）

给定文法G(S)：

$S \rightarrow A\ b$
$S \rightarrow A\ B\ c$
$A \rightarrow a\ A$
$A \rightarrow a$
$B \rightarrow b$

1. 下图表示该文法的LR(0)自动机，部分状态所对应的项目集未给出，试补齐之（即分别给出状态 I_0，I_2，I_3 ，I_4，I_5 ，I_6 和 I_7 对应的项目集。



2. 指出该LR(0)自动机中冲突的状态（并指出是哪种类型的冲突），以说明该文法不是LR(0) 文法。

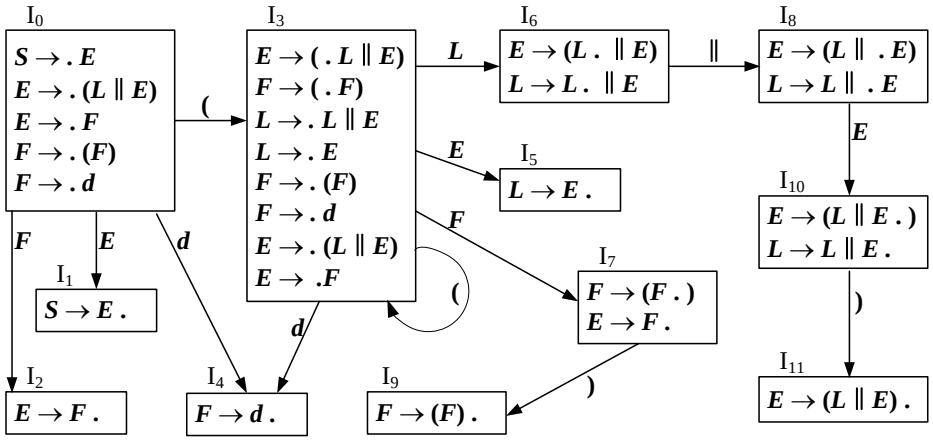
3. 说明该文法是 SLR(1) 文法。

六（07级）

给定如下文法 G(E)：

(1) $E \to (L \parallel E)$
(2) $E \to F$
(3) $L \to L \parallel E$
(4) $L \to E$
(5) $F \to (F)$
(6) $F \to d$

为文法 $G[E]$ 增加产生式 $S \to E$，得到增广文法$G'[S]$，并构造相应得 LR（0） 有限状态机如下图所示：



1. 该 LR（0）有限状态机中存在两个冲突的状态，试指出它们，并分别说明这两个状态是否可用SLR(1)分析方法解决？
2. 根据课程的讲解，对于不可用SLR(1)分析方法解决的冲突状态，实际上可以根据句柄实际所期望的后跟符号来解决这一冲突。试通过分析该 LR（0）有限状态机，指出相应句柄实际所期望的后跟符号，并说明你的结论是通过观察哪几个状态得到的？

七 （07级）

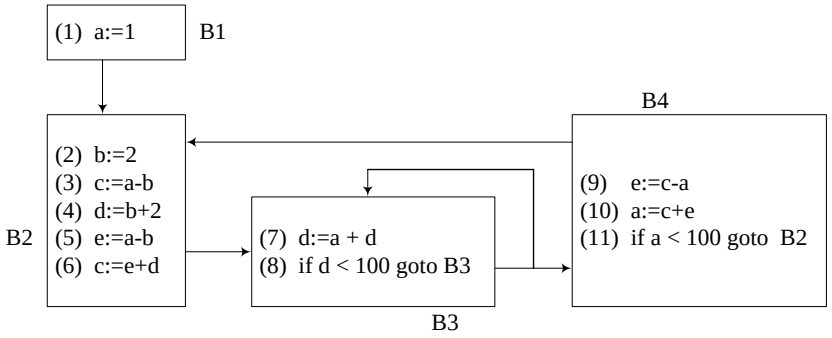
给定文法G(S)：

$$S \rightarrow S\,S \mid a\,S\,b \mid \varepsilon$$

在该文法的 LR(1) 自动机中，存在有冲突的 LR(1) 项目集（状态）。

1. 给出这些项目集（状态）中任意一个存在移进/归约冲突的项目集（状态）。
2. 给出这些项目集（状态）中任意一个存在归约/归约冲突的项目集（状态）。

八 （07级）



1. 对于上图所给出的流图，为基本块 B2 构造 DAG 图。
2. 对于上图所给出的流图，根据采用迭代求解数据流方程对活跃变量信息进行分析的方法，求出 B2 出口处、B2 入口处以及 B4 入口处的活跃变量集合，即 LiveOut(B2)，LiveIn(B2) 和 LiveIn (B4) 的值。假设迭代开始时 LiveOut(B4)= ϕ。
3. 利用 LiveOut(B2) 的结果，指出在语句（4）和语句（5）之间的程序点，哪些变量是活跃的。
4. 求该流图范围内，d 在定值点 (4) 的 DU 链。

九 以下是语法制导生成TAC语句的一个S-属性文法（翻译模式）： **（06级）**

$S \rightarrow$ *if E then M* S_1
{ *backpatch(E.truelist,M.gotostm) ;*
S.nextlist := merge(E.falselist, S_1*.nextlist)* }

$S \rightarrow$ *if E then* M_1 S_1 *N else* M_2 S_2
{ *backpatch(E.truelist,* M_1*.gotostm) ;*
backpatch(E.falselist, M_2*.gotostm) ;*
S.nextlist := merge(S_1*.nextlist, merge(N.nextlist,* S_2*.nextlist))* }

$S \rightarrow$ *while* M_1 E *then* M_2 S_1

{ *backpatch*(S_1.*nextlist*, M_1.*gotostm*) ;
backpatch(*E.truelist*, M_2.*gotostm*) ;
S.nextlist := *E.falselist*;
emit(‘*goto* ’ , M_1.*gotostm*) }

$S \rightarrow S_1$; M S_2

{ *backpatch*(S_1.*nextlist*, *M.gotostm*) ; *S.nextlist* := S_2.*nextlist* }

$S \rightarrow S'$ { *S.nextlist* := *S'.nextlist*) }

$S' \rightarrow$ *id* := *A*

{ *S'.nextlist* := ” ;
emit (*id* .*place* ‘:=’*A* . *place*) }

$E \rightarrow E_1$ or M E_2

{ *backpatch*(E_1.*falselist*,*M.gotostm*) ;
E.truelist := *merge*(E_1.*truelist*, E_2.*truelist*) ;
E.falselist := E_2.*falselist* }

$E \rightarrow E_1$ *and* M E_2

{ *backpatch*(E_1.*truelist*,*M.gotostm*) ;
E.falselist := *merge*(E_1.*falselist*, E_2 .*falselist*) ;
E.truelist := E_2 .*truelist* }

$E \rightarrow$ not E_1

{ *E.truelist* := E_1.*falselist* ; *E.falselist* := E_1.*truelist* }

$E \rightarrow (E_1)$

{ *E.truelist* := E_1.*truelist* ; *E.falselist* := E_1.*falselist* }

$E \rightarrow$ $\underline{id}_1$ rop $\underline{id}_2$

{ *E.truelist* := *makelist* (*nextstm*);
E.falselist := *makelist* (*nextstm*+1);
emit (‘if‘ $\underline{id_1}$.*place* rop.op $\underline{id_2}$.*place* ‘goto _’);
emit (‘goto _’) }

$A \rightarrow A_1 + A_2$

{ *A* .*place* := *newtemp*;
emit(*A* .*place* ’:=’ A_1 . *place* + A_2 . *place*) }

$A \rightarrow A_1 * A_2$

{ *A* .*place* := *newtemp*;
emit(*A* .*place* ’:=’ A_1 . *place* * A_2 . *place*) }

$A \rightarrow$ *id* { *A* .*place* := *newtemp*;
emit (*A* .*place* := *id* .*place*) }

$A \rightarrow$ *const* { *A* .*place* := *newtemp*;
emit (*A* .*place* := *const* .*val* }

..... /*这里略去关于算术表达式更多的部分*/

$M \rightarrow \varepsilon$

{ *M.gotostm* := *nextstm* }

$N \rightarrow \varepsilon$

{ *N.nextlist* := *makelist*(*nextstm*); *emit*(‘*goto* _’) }

其中，所用到的语义函数与讲稿中一致，列举如下：

makelist(i)：创建只有一个结点 i 的表，对应存放目标*TAC* 语句数组的一个下标

merge(p_1,p_2)：连接两个链表 p_1 和 p_2 ，返回结果链表

backpatch(p,i)：将链表 p 中每个元素所指向的跳转语句的标号置为 i

nextstm：下一条*TAC* 语句的地址

emit (…)：输出一条*TAC* 语句，并使 *nextstm* 加1

newtemp：返回一个未使用过的名字

属性 *E .truelist , E .falselist , S .nextlist, S'.nextlist, A .place, M .gotostm, N .nextlist* 以及所涉及到的 TAC 语句与讲稿中一致，

另外，要注意到本题中使用的文法非终结符含义：*S*（所有语句），*S'*（赋值语句），*E*（布尔表达式），*A*（ 算术表达式）。

（此外，假设在语法制导处理过程中遇到的二义性问题可以按照某种原则处理比如规定优先级，else 匹配之前最近的 if，运算的结合性，等等，这里不必考虑基础文法的二义性。）

1. 若在基础文法中增加产生式

$$E \to \Delta (E, E , E)$$

其中“Δ”代表一个三元逻辑运算符。逻辑运算 $\Delta (E_1 , E_2 , E_3)$ 的语义可由下表定义：

E_1	E_2	E_3	$\Delta (E_1 , E_2 , E_3)$
false	false	false	false
false	false	true	false
false	true	false	true
false	true	true	false
true	false	false	false
true	false	true	false
true	true	false	false
true	true	true	false

试参考上述布尔表达式的处理方法，给出相应的语义处理部分（必要时增加文法符号，类似上述属性文法中的符号 *M* 和 *N*），不改变 S-属性文法（翻译模式）的特征。

2. 若在基础文法中增加关于循环卫士语句的下列产生式

$$S \to do\ G\ od$$

$$G \to E : S \square G$$

$$G \rightarrow E : S$$

试参考上述控制语句的处理方法，给出相应的的语义处理部分（必要时增加文法符号，类似上述属性文法中的符号 M 和 N， 以及增加新的属性），不改变 S-属性文法（翻译模式）的特征。

注：循环卫士语句的一般形式如

$$do\ E_1 : S_1 \square E_2 : S_2 \square \ldots \square E_n : S_n\ od$$

我们将其语义解释为：

(1) 依次判断布尔表达式 E_1 ，E_2 ，…，E_n 的计算结果。

(2) 若计算结果为 true 的第一个表达式为 E_k（$1 \le k \le n$），则执行语句 S_k；转（1）。

(3) 若 E_1 ，E_2 ，…，E_n 的计算结果均为 false ，则跳出循环。