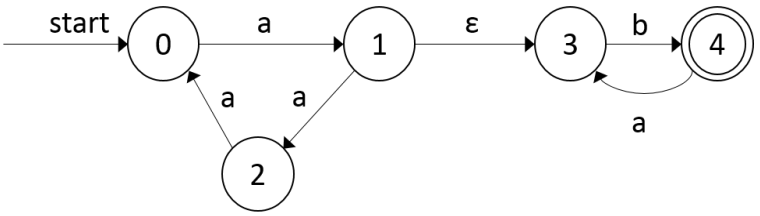
北京大学信息科学技术学院试题

考试科目： 编译技术 **姓名：** _____ **学号：** _____

考试时间： 2019 年 6 月 19 日 **任课教师：** 梁云

一．单项选择题（每小题 2 分，共计 20 分）

1. 下列哪一个正则表达式表示的语言与图中 NFA 能够识别的语言相同？



(A) (aaa)*(ba)+ (B) a(aa)*(ba)* (C) a(aaa)*(ab)* (D) (aaa)*(ab)+

2. 按照文法

E→A*E | A / E | A
A→B+A | B – A| B
B→number | (E)

计算表达式 21 / 7 +（21 - 10）/ 2 + 1 * 2 的结果为：

(A) 7 (B) 10.5 (C) 7/9 (D) 15

3. 如果文法 G 是无二义的，则它的任何句子 α_____。

(A) 最左推导和最右推导对应的语法树必定相同
(B) 最左推导和最右推导对应的语法树可能不同
(C) 最左推导和最右推导必定相同
(D) 可能存在两个不同的最左推导，但它们对应的语法树相同

4. S→bAb | bBa

A→aS | CB
B→b | Bc
C→c | cC
关于该文法，下列说法中不正确的是：
(A) 该文法包含左公因子
(B) 该文法包含左递归
(C) 该文法是 LL(1)的
(D) 该文法能够产生无限多个彼此各不相同的字符串

5. C 语言的 for 语句有如下形式：

for (E1; E2; E3) S
为这个语句的代码生成构造一个 L 属性定义，并使用自底向上的分析方法进行翻译，为此，需要引入若干 Mark 符号，拓展后的产生式以及 Mark 符号的语义动作如下：

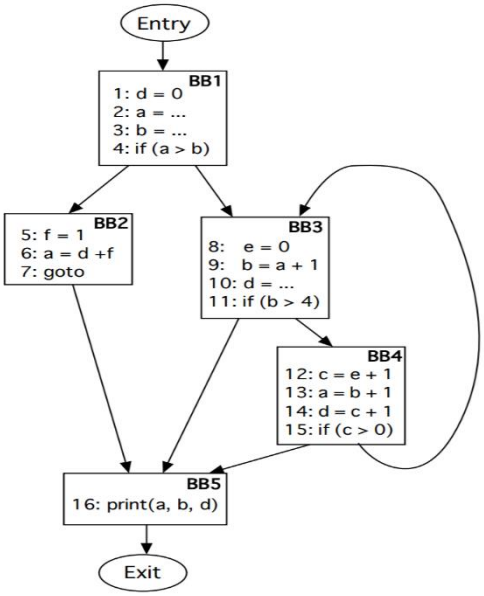
S→for (E1; M1 E2; M2 E3) N S1
M→ε { M.next = nextstmt; }
N→ε { N.nextlist = makelist(nextstmt);
emit('goto _');
N.next = nextstmt; }

则下面哪一个会被用来回填 S1.nextlist：
(A) N.next (B) M1.next
(C) M2.next (D) NULL

6. 下列说法正确的是？

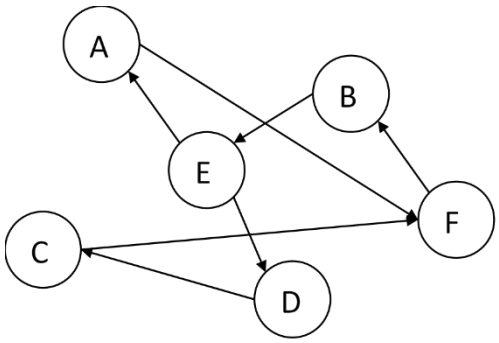
① 如果 L 是一个正则语言，那么$\{\omega\omega^R \mid \omega \in L\}$也是正则语言，其中 ω^R 表示 ω 的逆序串。
② 对任意两个正则表达式 R 和 S，由它们合成的表达式 R(SR)* 和 (RS)*R 是等价的。
③ 如果 L_2, L_1L_2, L_2L_1 都是正则语言，那么 L1 也一定是正则语言。
(A) ②③ (B) ①③ (C) ① (D) ②

7. 对于如下代码片段，在不溢出的前提下，请问至少需要几个寄存器，才能讲图中出现的变量全部染色？



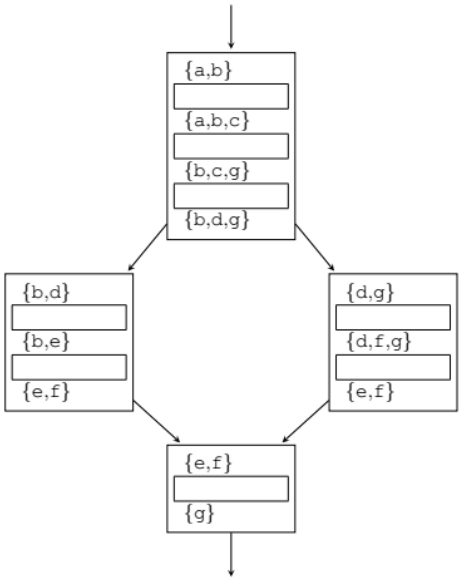
(A) 2
(B) 3
(C) 4
(D) 5

8. 在下列引用关系图中，删除哪个对象会使得 F 被回收?



(A) A (B) B (C) C (D) D

9. 下图是一个函数对应的控制流图，函数中使用了 a，b，c，d，e，f 和 g 这 7 个变量，其中 a、b 是函数参数，g 是返回值。图中每个空框代表一条语句，空框前后的括号内注明了语句执行前后的活跃变量（live variables）。空框里面的语句是下列三种形式之一：x = 1， x = y ， x = y + z ，其中 x，y，z 三个变量各不相同。则该图可能的填写方法有（）种?



(A) 144 (B) 324 (C) 576 (D) 648

10.有如下程序:

```
program main(a);
    var b, i:integer;
    function P(n:integer):integer;
        begin   if n <= 1 then P := 1 else P := P(n - 1) * n   end;
    function Q(n:integer):integer;
        function C(n:integer):integer;
            begin   if n > 3 then C := n – 1 else C := n + 1   end;
        function D(n:integer):integer;
            begin   if n > 27 then D := -1   else D := n + 1   end;
        begin   if n > 7 then Q := D(n)   else Q := C(n)   end;
    begin
        b := a + 1;
```

```
        repeat
            if b < 5 then i := P(b) else i := Q(b); b := i;
        until i < 0;
    end.
```

第 1 次函数调用是 main(2)，问第 11 次调用的函数的访问链指向 ( )

(A) main(2)

(B) P(1)

(C) Q(24)

(D) D(24)

二. 简答题（每小题 6 分，共计 30 分）

1. 请画出一个 DFA，使其接收所有被 4 整除的正整数的三进制表示（不包括 0，可以有前缀 0），如 0011。

2. 将表达式 a * (b[i] + c / b[i - 1])分别翻译为抽象语法树、四元式、间接三元式。

3. 下面为一段三地址代码，a 在程序开始时是活跃的，请回答：

```
（1）   t1 = 0
（2）   n = 1024
（3）   t2 = t1 + 1
（4）   if t2 < n goto (6)
（5）   goto (16)
（6）   t3 = t1 * 4
（7）   t4 = a[t3]
（8）   t5 = t2 * 4
（9）   t6 = a[t5]
（10）  if t4 < t6 goto (12)
（11）  goto (14)
（12）  a[t3] = t6
（13）  a[t5] = t4
（14）  t1 = t1 + 1
（15）  goto (3)
（16）  exit
```

1) 为这段代码划分基本块并标出每个基本块入口和出口处的活跃变量。(4 分)

2) 画出各个变量的干涉图（interference graph）（2 分）

4. 张量可以看作是多维数组（如矩阵是二维张量），张量积算即是张量间的数学运算。这里给出一种可以表达张量计算的极简文法：

Stmt → For id = 0; id < N; id = id + 1 Stmt | V = Expr
Expr → Expr + F | Expr – F | F
F → F * P | F / P | P
P → V | N
V → id | V[Expr]
N → Number
id → Variable

使用该文法来表示矩阵乘法的例子是：

```
For i = 0; i < 1024; i = i + 1
  For j = 0; j < 1024; j = j + 1
    For k = 0; k < 1024; k = k + 1
      C[i][j] = C[i][j] + A[i][k] * B[k][j]
```

为了生成高效的代码，需要进行 split 操作，所谓 split 操作即是将原有循环分割为两个循环，一个称为外层循环(outer)，另一个称为内层循环(inner)，两个新的循环的大小乘积等于原循环大小，如对上述矩阵乘法例子的 split 结果如下：

```
For (int i.outer = 0; i.outer < 32; i.outer = i.outer + 1)
 For (int i.inner = 0; i.inner < 32; i.inner = i.inner + 1)
  For (int j.outer = 0; j.outer < 32; j.outer = j.outer + 1)
   For (int j.inner = 0; j.inner < 32; j.inner = j.inner + 1)
    For (int k.outer = 0; k.outer < 32; k.outer = k.outer + 1)
```

```
For (int k.inner = 0; k.inner < 32; k.inner = k.inner + 1)
    C[i.outer * 32 + i.inner][j.outer * 32 + j.inner] =
        C[i.outer * 32 + i.inner][j.outer * 32 + j.inner] +
        A[i.outer * 32 + i.inner][k.outer * 32 + k.inner] *
        B[k.outer * 32 + k.inner][j.outer * 32 + j.inner]
```

这里给出部分残缺的 SDT 用来完成上述 split 操作并生成代码。请补全 SDT 中的空缺（即_____处）实现上述代码生成工作

(**提示**：SDT 使用的数据结构 replace_dict 是一个 Map，用来替换访存下标，可以参考上述矩阵乘法 split 的例子进行理解)。(每空 1 分)

(1) **Stmt → For id = 0 id < N; id = id + 1**

{ id.extent = N.val

outer = new Variable(name=id.name + ".outer" , extent=

①_____);

inner = new Variable(name= id.name + ".inner", extent=32);

Stmt1.replace_dict[id.name] =②_____;

emit("For (int " || outer.name || " = 0; " outer.name || " < " || outer.extent ||

"; " || outer.name || "=" || outer.name || " + 1)");

emit(③_____) }

Stmt1

(2) **Stmt →**

{ V.replace_dict =④ _____}

V =

{ emit("="); Expr.replace_dict = Stmt.replace_dict }

Expr

(3) **Expr →**

{ Expr1.replace_dict = Expr.replace_dict }

Expr1 +

{ emit("+"); F.replace_dict = Expr.replace_dict }

F

(4) **Expr →**

{ Expr1.replace_dict = Expr.replace_dict }

Expr1 –

{ emit("-"); F.replace_dict = Expr.replace_dict }

F

(5) **Expr →**{ F.replace_dict = Expr.replace_dict } **F**

(6) **F →**

{ F1.replace_dict = F.replace_dict }
F1 *
{ emit(" * "); P.replace_dict = F.replace_dict }
P

(7) **F →**
{ F1.replace_dict = F.replace_dict }
F1 /
{ emit("/"); P.replace_dict = F.replace_dict }
P

(8) **F →** { P.replace_dict = F.replace_dict } **P**

(9) **P →** { V.replace_dict = P.replace_dict } **V**

(10) **P →N** { emit(N.val) }

(11) **V → id**
{ if id.name in V.replace_dict
then emit(⑤_____)
else emit(⑥_____) }

(12) **V →**
{ V1.replace_dict = V.replace_dict }
V1 [
{ emit("["); Expr.replace_dict = V.replace_dict }
Expr]
{ emit("]") }

(13) **N → Number** { N.val = Number.lexval }

(14) **id →Variable** { id.name = Variable.name }

5. Python 语言具有一些独特的特点，比如通过缩进识别控制块的嵌套、不需要对变量进行定义（自动推测出变量类型）。当变量的类型发生变化时，Python 会以这个新的类型来重新定义该变量。而在 C 语言中，我们可以在变量类型发生变化时通过重新定义一个新的变量，并在后续的使用中操作这个新的变量，从而达到和 Python 中类似的效果。

如果 Python 代码中仅有 if 控制语句，且规定 if 语句块内的对变量的重新定义（即某变量在进入 if 语句块后发生了类型变化，需要重新定义）不会对 if 语句块之外对该变量的使用造成影响，那么 Python 代码可以被翻译为等价的 C 代码。

如下的两段代码等价：

```
# Python 代码段：
a = 1
i = false
if i:
    if a > 0:
        a = false
        i = a and false
    a = 2 + a
i = 2
```

```
// C 代码段：
int a0 = 1;
bool i0 = false;
if (i0) {
    if (a0 > 0) {
        bool a1 = false;
        i0 = a1 && false;
    }
    a0 = 2 + a0;
}
int i1 = 2;
```

某简易 Python 代码由下列文法生成：

Code	→ Code Line | **ε**
Line	→ Tabs Command |***n***
Command	→ ***if*** Bool ***:*** | Variable = Rvalue
Rvalue	→ Variable | Bool | Int
Bool	→ Variable | Bool ***and*** Bool | Int > Int | ***true*** | ***false***
Int	→ Variable | Int + Int | ***constant***
Variable	→ ***word***
Tabs	→ Tabs ***tab*** | **ε**

本题讨论的数据类型有 int 型和 bool 型，并规定 int 类型和 bool 类型的变量不兼容，即两种数据类型之间不能进行转换。本题讨论的运算有加法运算（+，

以两个 int 型数据为操作数，结果类型为 int 型）、与运算（Python 中为 and，C 中为&&，以两个 bool 型数据为操作数，结果为 bool 型）、大于运算（>，以两个 int 型数据为操作数，结果为 int 型）、赋值运算（=，如果等于号两边的变量类型不同，那么等于号左边的变量将会被重新定义，其类型与等于号右边的变量/值类型相同）。本题讨论的控制语句仅有 if 语句（Python 中为 if cond: ，C 中为 if(cond){}，其中 cond 必须为 bool 型）。

请在上述文法的基础上，设计一个语法制导定义（SDD），能够对由这个文法生成的 Python 代码进行合法性检查，并将其翻译成等价的 C 代码。需要进行的合法性检查有：tab 数嵌套（tab 嵌套数最初为 0，当且仅当在遇到 if 语句时语句块内 tab 嵌套数加 1）、对 Variable 的类型检查（加法、与、大于运算的操作数的类型是否和运算相符）。（提示：当检查到 Python 代码中存在错误时，只需直接输出“error”并返回即可；所生成的 C 代码不需要考虑缩进换行等格式问题；constant 和 word 的值可以直接通过其 val 属性获得； SDD 中允许定义所需要的数据结构，如数组、集合、堆、栈、队列等，及其相应的操作）

三．LR 文法（10 分）

有增广文法如下：

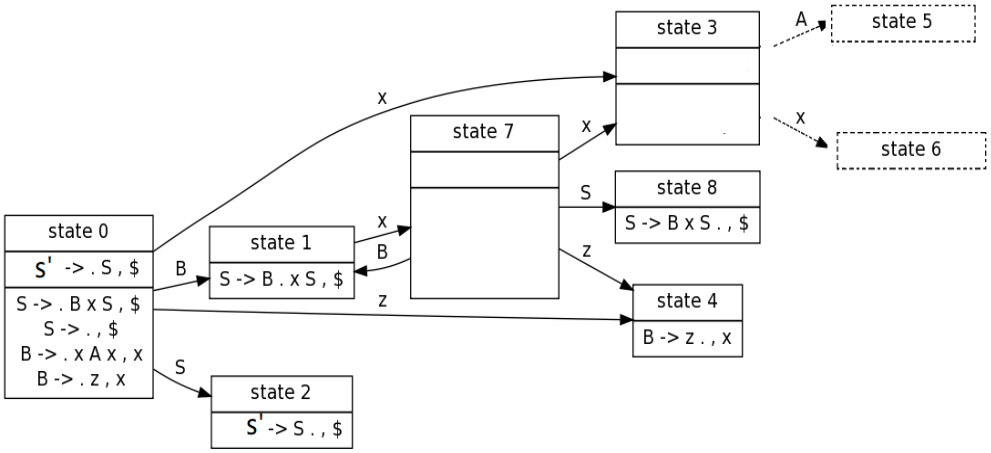
S'→S

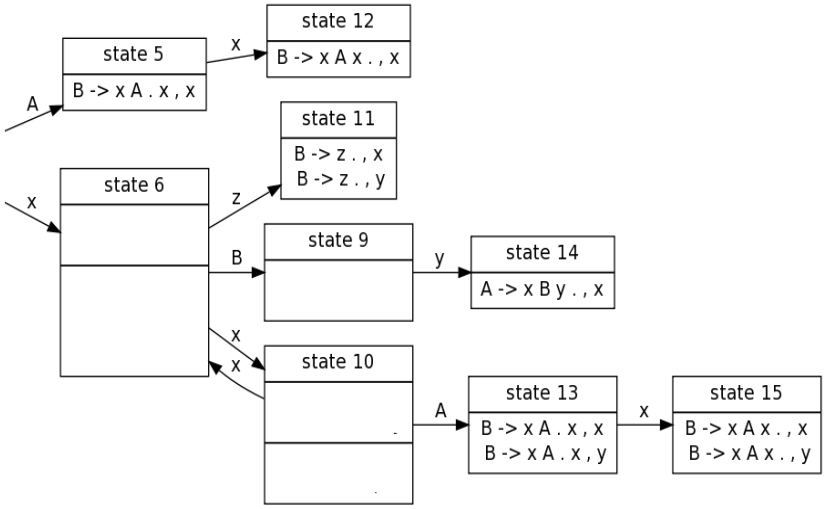
S→ B x S | ε

A→ x B y | x B

B → x A x | z

对于这个文法，有 LR(1)自动机如下（部分内容被挖去）。





1） 补全 LR(1)自动机。答题时标明对应 state。（5 分）

2） 该文法是 SLR 的么？该文法是 LR(0)的么？请分别回答并称述理由。（5 分）

四．SDD（10 分）

有如下 SDD：

产生式	语义规则
T → B C	T.t = C.t;　T.width = C.width; C.b = B.t;　C.w = B.width;
B → **int**	B.t = integer;　B.width = 4;
B → **float**	B.t = float;　　B.width = 8;
C → [**num**]C_1	C_1.b = C.b　C_1.w = C.w C.t = array(num.val, C_1.t); C.width = num.val × C_1.width;
C →　ε	C.t = C.b;　　C.width = C.w;

请先说明 SDD 中出现的各个属性是综合属性还是继承属性。（3 分）

然后画出 int [5][6] 的注释语法分析树，并用虚线标注出求值顺序。（7 分）

五．中间代码分析（10 分）

现有如下结构体：

```
struct {
   double a;
   int b;
} simple;
```

int 和 double 的长度分别为 4、8 字节。有如下代码和对应的三地址码：

int i;
double g[10];
simple s;
s.a = 1;
s.b = 10;
for (i = 0; i < s.b; i++) {
  if (i < 5 || i > 0) {
  g[i] = s.a;
  }
  else {
  while (g[i] < 100) {
  g[i] += s.a;
  }
  }
}
	1) s[0] = 1
	2) s[8] = 10
	3) i = 0
	L0:
	4) if i < 5 goto L1
	5) if i > 0 goto L1
	6) goto L2
	L1:
	7) t1 = s[0]
	8) t2 = i * 8
	9) g[t2] = t1
	L2:
	10) t3 = i * 8
	11) t4 = g[t3]
	12) if t4 >= 100 goto L3
	13) t5 = s[0]
	14) t6 = i * 8
	15) t7 = g[t6]
	16) t8 = t5 + t7
	17) g[t6] = t8
	18) goto L2
	L3:
	19) t9 = s[8]
	20) if i >= t9 goto L4
	21) t10 = i + 1
	22) i = t10
	23) goto L0
	L4:
	24)

1） 请更正这段三地址码中翻译的 2 处错误。（提示：2 处错误均和控制流相关；每改正一处错误，最多只允许增加/删除/修改两条语句。）（2 分）

2） 在(1)的三地址码中，如果在编号为 x 的语句和编号为 x+1 的语句之间增加了 y 条语句，则将这 y 条语句的编号逐条命名为 x-1, x-2, …, x-y. 其余的语句编号不变，请为(1)更正后的三地址码构造出控制流图（CFG），流图中的基本块只需注明其所包含的语句编号即可。（5 分）

3） 请用基本块表示出该控制流图中的所有循环；设 S 为 if-else 语句体，请用三地址代码的编号表示出 S 的 nextlist。（3 分）

六．SDT（10 分）

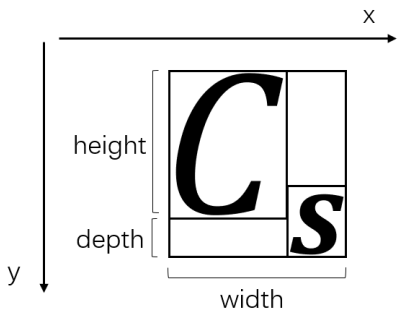
某排版系统中表达式文法为：

$$\mathrm{B} \rightarrow \mathrm{B_1B_2} | \mathrm{B_1subB_2} | (B_1) | \mathbf{text}$$

这里的 B 代表文本框，sub 和 text 都是终结符号，分别代表“下标运算”和“单个字母或符号”。这四个产生式的含义分别为：

(1) 并列组成的文本框
(2) 右侧有下标的文本框
(3) 用小括号分组的文本框
(4) 文本串

在使用该文法排版时，可以理解为从小文本框组合出大文本框的过程，规定分析过程是左结合的，sub 运算优先级高于连接运算。每个文本框具有属性 width，height，font_size，depth，height 和 dpeth 都是相对于一条基准线的，而不是指文本框整体的高度和深度。参考下图理解各个属性的含义。



font_size 指方框中字体大小，没有下标时的字体大小为 10 点。计算这些属性的规则如下：

$\mathrm{S} \rightarrow B$	$\mathrm{B.font\_size} = 10$
$\mathrm{B} \rightarrow B_1 B_2$	$\mathrm{B}_1.font\_size = B.font\_size$ $\mathrm{B}_2.font\_size = B.font\_size$ $\mathrm{B.height}=\max(\mathrm{B}_1.height, B_2.height)$ $\mathrm{B.width}=\mathrm{B}_1.width + B_2.width$ $\mathrm{B.depth}=\max(\mathrm{B}_1.depth, B_2.depth)$
$\mathrm{B} \rightarrow B_1 \mathbf{sub} B_2$	$\mathrm{B}_1.font\_size = B.font\_size$ $\mathrm{B}_2.font\_size = 0.6 \times B.font\_size$ $\mathrm{B.height}=\max(\mathrm{B}_1.height, B_2.height - 0.3 \times B.font\_size)$ $\mathrm{B.width}=\mathrm{B}_1.width + B_2.width$ $\mathrm{B.depth}=\max(\mathrm{B}_1.depth, B_2.depth + 0.3 \times B.font\_size)$
$\mathrm{B} \rightarrow (B_1)$	$\mathrm{B}_1.font\_size = B.font\_size$ $\mathrm{B.height} = \mathrm{B}_1.height$ $\mathrm{B.width} = \mathrm{B}_1.width$ $\mathrm{B.depth} = \mathrm{B}_1.depth$
$\mathrm{B} \rightarrow \mathbf{text}$	$\mathrm{B.height} = \mathrm{getHeight(B.font\_size, text.lexval)}$ $\mathrm{B.width} = \mathrm{getWidth(B.font\_size, text.lexval)}$ $\mathrm{B.depth} = \mathrm{getDepth(B.font\_size, text.lexval)}$

请回答以下问题：

1) font_size, height, width, depth 中哪些是综合属性，哪些是继承属性？（2 分）

2) 为了实现排版，需要确定每个文本框的位置，以纸张的左上角为原点，用一个二维坐标来表示每个文本框左上角的位置便可以实现定位，一旦确定了文本框位置，使用系统中给出的 placeBox 接口便可以放置好文本框，该函数的声明如下：

void placeText (int xpos, int ypos, int height, int width, Box & parent);

参数分别为横坐标，纵坐标，该文本框高度，宽度和父文本框。注意这里接收的坐标都是相对于父文本框的。
此外，为了实际绘出文本，对于最后一个生成式，需要额外调用接口 drawText 绘制文本，该接口声明为：

void drawText (string textval, Box & current);

参数分别为文本值（text.lexval）和所在文本框。所有的文本框都是 Box 类型的, S 也可以看作是 Box 类型。

请设计并写出一个 SDT 完成以下任务对各个文本框的坐标计算，实际放置以及文本绘制。注意坐标计算方法（参考题图）以及求值次序，以及不要混淆 height 属性和文本框高度。为了简洁，本题不考虑换行。（8 分）

七．代码优化（10 分）

循环是一种基本的控制语句，程序运行的大部分时间常常花在少数的循环代码块当中，因此对循环的优化是编译优化的一大重点。在循环中可能存在所谓的“循环不变表达式”，该表达式的值在每次循环中都不发生变化，这时可以在不影响循环语句块的语义的前提下，将该表达式的计算从循环内部移动到循环外部，避免在循环内部进行重复计算。这种对循环进行的基本优化称为“循环不变表达式外移”（loop invariant expression motion）。

1） 循环可以由基本块表出，我们将仅包含该循环的基本块的控制流图称为这个循环的循环控制流图。请设计一个数据流分析算法来找到一个循环控制流图中的所有循环不变表达式，只需说明你所采用的数据流算法的方向（前向、后向），状态的表示（用什么数据结构标识和判断一个表达式是否为循环不变表达式，如何初始化该数据结构），状态转移函数（经过一条语句后状态应该如何改变），交汇操作（两个状态如何合并）。（6 分）

2） 请找出下列代码中的循环不变表达式，并指出这些循环不变表达式外移到的位置。（不需要说明步骤）（2 分）

```
int a = 1, b = 2, c = 3, d = 4, e = 5, f = 6;
int array[100];
for (int i = 0; i < 100; i++) {
   for (int j = 0; j < 100; j++) {
if (j < 50) {
   a = i / 2;
   b = j * 2;
   array[j] = a + b;
}
else {
   c = b + 3;
   d = d * 2;
   e = f / 2;
   array[j] = c + d + e;
}
   }
}
```

3）将循环不变表达式外移，是否总是能够减少程序运行的实际指令数、提高程序性能？请简要解释。（2 分）