（若发现问题，请及时告知）

A2 是选自 Lecture08 文档中的题目

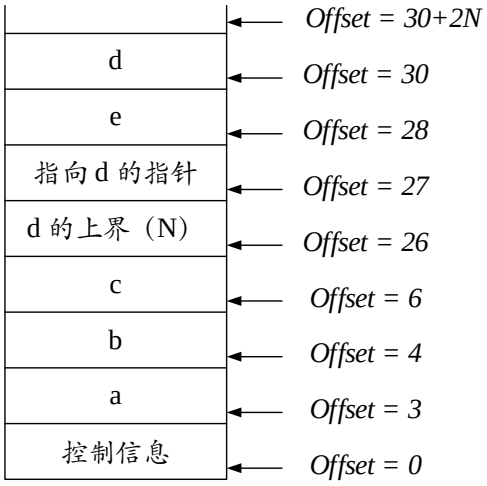
A4，A5，A7 是选自 Lecture09 文档中的题目

A1.

若按照某种运行时组织方式，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 ≤ i < N）时相对于活动记录基址的 *Offset* 值如何计算？*Offset* 为 26 和 27 的单元用于存放数组 d 的内情向量，按你的理解，后者（*Offset* 为 27 的单元）的内容应该存放什么？ 若数组 c 也是动态数组，并采用与 d 相同的存储方式，则存放“指向 d 的指针”单元的 *Offset* 值是多少？内容是什么？

参考解答：

函数 p 中访问 d[i]（0 ≤ i < N）时相对于活动记录基址的 *Offset* 值可通过 30+2i 来计算。

Offset 为 27 的单元，内容可以为30，指向 d 在栈中的起始位置（*Offset*）。

若数组 c 也是动态数组，并采用与 d 相同的存储方式，则存放“指向 d 的指针”单元的 *Offset* 值是29，内容不确定。

A2. 下图左边是某简单语言的一段代码。语言中不包含数据类型的声明，所有变量的类型默认为整型（假设占用一个存储单元）。语句块的括号为‘begin’和‘end’组合；赋值号为 ‘:=’。每一个过程声明对应一个静态作用域。该语言支持嵌套的过程声明，但只能定义无参过程，且没有返回值。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。程序的执行遵循静态作用域规则。下图左边的 PL/0 程序执行到过程 p 被第二次激活时，运行栈的当前状态如下图右半部分所示（栈顶指向单元 26），其中变量的名字用于代表相应的内容。

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

1. 试补齐该运行状态下，单元18、19、21、22、及 23 中的内容。

2. 若采用 Display 表来代替静态链。假设采用只在活动记录保存一个Display 表项的方法，且该表项占居图中SL的位置。试指出当前运行状态下 Display 表的内容，以及各活动记录中所保存的Display 表项的内容（即图中所有SL位置的新内容）

3. 若采取动态作用域规则，该程序的执行效果与之前有何不同？

参考解答：

1. 试补齐该运行状态下，单元18、19、21、22、及 23 中的内容。

单元18中的内容：0；

单元19中的内容：13；

单元21中的内容：q中x的内容；

单元22中的内容：0；

单元23中的内容：18。

2.

当前 Display 表的内容：

D[0] = 0
D[1] = 22
D[2] = 13

各活动记录中所保存的Display 表项的内容：

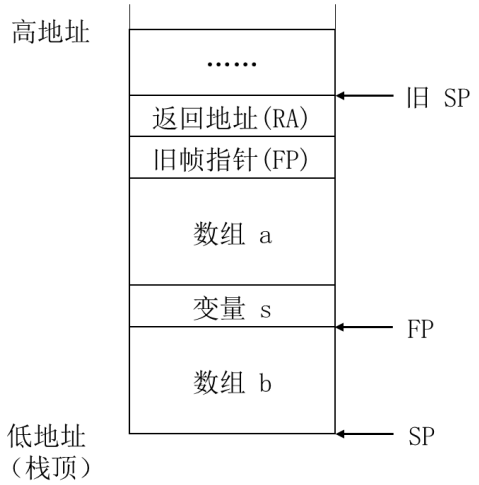
单元0中的内容： _ 无效
单元5中的内容： _ 无效
单元9中的内容：5
单元13中的内容： _ 无效
单元18中的内容： 9
单元22中的内容：18

3. 若采取动态作用域规则，该程序的执行效果与之前有何不同？

在第二次执行到语句 L 时，若是静态作用域规则，则 a=1, b=2，因此会再次调用 P; 若是动态作用域规则，则 a=3, b=2，因此不会调用 P。

A3. 对于以下 C 语言函数片段，其运行时的活动记录按右图方式组织：

```
static int N, L;
int func() {
    int a[100];
    int s = 3;
    int b[N];
    register int i;
    for (i = 0; i < L; i ++) {
        b[i] = i * i;
    }
    a[10] = b[5];
    return s;
}
```



其中 i 是寄存器变量，保存在寄存器 T0 中，不保存在栈上。a 和 s 的大小是确定的，可直接确定其在栈内的偏移。而 b 是一个动态数组，编译器并不能确定将需要多少存储空间。本题的活动记录中的控制信息包括返回地址（RA）和旧帧指针（FP）。本题的活动记录中，首先按照定义顺序依次存储变量与静态数组，之后按照定义顺序存储动态数组。本题中不在活动记录中保存内情向量和指向动态数组的指针。如右图所示，本题的活动记录中首先保存 RA 与 FP，之后按照定义顺序保存静态数组 a 和变量 s，最后保存动态数组 b。其中用 FP 保存分配动态数组（即 b）之前的栈顶指针（SP）。本题的活动记录中数组中的元素根据编号由低地址向高地址存储在栈中，例如 b[0]存储在 SP 所指位置，即 b[0]=M[SP]。假设全局变量 N 存放的内存位置为 0x4000。一个 int 类型变

量占 4 个字节。RA 与 FP 各占 4 个字节。

回答以下问题：

1. 该函数的栈帧（过程活动记录）总大小为_____字节（写成关于 N 的表达式）。

2. 在进入函数时，需要为所有**确定大小**的变量（数组 a 和变量 s）和旧 FP、RA 分配栈空间，之后再为 b 动态分配空间。生成的 RISC-V 目标代码如下（如果不熟悉汇编指令，右侧给出了详细的注释帮助你理解）。请补全其中的偏移量（使用十进制整数填写）。

```
func_prologue:
addi sp, sp, ____①____   # SP <- SP + ??? （分配栈帧）
sw   ra, ____②____(sp)   # M[SP + offset_RA] <- RA （保存旧 RA）
sw   fp, ____③____(sp)   # M[SP + offset_FP] <- FP （保存旧 FP）
mv   fp, sp              # FP <- SP （设置新 FP）
lw   a0, 0x4000(zero)    # A0 <- M[0x4000] （装入 N）
slli a0, a0, 2           # A0 <- A0 * 4 （装入 N）
sub  sp, sp, a0          # SP <- SP - A0 （为 b 分配空间）
```

3. 函数中 a[10]=b[5]; 这条语句对应着如下的目标代码。请补全其中的偏移量（使用十进制整数填写）。

```
lw   a0, ____④____(sp)# 装入 b[5]
sw   a0, ____⑤____(fp)# M[FP + offset_a_10] <- A0（a[10] = b[5]）
```

4. 源程序中存在一个缓冲区溢出漏洞，如果 L 的值大于 N 时会导致 b 数组访问越界，此时可能会覆盖掉栈上的一些数据。覆盖数据可能会导致函数返回错误的值，更严重的是，如果被覆盖的数据恰好是返回地址，函数返回时就会跳转到错误的地址。精心构造该地址可使得程序执行流程发生更改，导致执行恶意代码等严重后果。假设 N 是固定的常数，试问 L 取值大于等于多少时会覆盖函数的返回值？取值大于等于多少时会覆盖函数的返回地址？

解答：

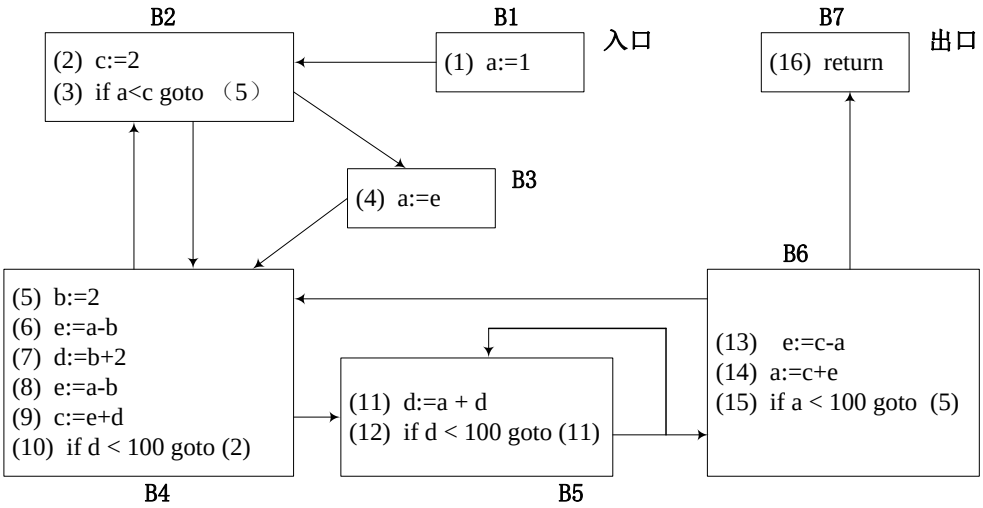
(1) 4N+412

(2) ① -412 ② 408 ③ 404

(3) ④ 20 ⑤ 44

(4) L>=N+1　　L>=N+102

A4.　下图是包含 7 个基本块的流图，其中 B1 为入口基本块，B7 为出口基本块：



1. 指出在该流图中，基本块 B4 的支配结点（基本块）集合，始于 B4 的回边，以及基于该回边的自然循环中包含哪些基本块？

2. 采用迭代求解数据流方程的方法对活跃变量信息进行分析。假设B7的 LiveOut 信息为空集 ∅，迭代结束时的结果在下图所示表中给出。试填充该表的内容。

	LiveUse	DEF	LiveIn	LiveOut
B1				
B2				
B3				
B4				
B5				
B6				
B7				∅

3. 对于该流图，根据采用迭代求解数据流方程对到达-定值（reaching definitions）数据流信息进行分析的方法。假设 B1 的 IN 信息为空集 ∅，迭代结束时的结果在下图所示表中给出。试填充该表的内容。

	GEN	KILL	IN	OUT
B1			∅	
B2				
B3				
B4				

B5				
B6				
B7				

4. 指出该流图范围内，变量 a 在（11）的 UD 链。

5. 指出该流图范围内，变量 c 在（2）的 DU 链。

参考解答：

1. 基本块 B4 的支配结点（基本块）集合：{ B_1, B_2, B4};

   始于 B4 的回边 B4→B2；

   基于该回边的自然循环中包含基本块：B2, B3, B4, B5, B6

2. 求解结果如下：

	LiveUse	**DEF**	**LiveIn**	**LiveOut**
B1	∅	{a}	{e}	{a,e}
B2	{a}	{c}	{a,e}	{a,e}
B3	{e}	{a}	{e}	{a}
B4	{a}	{b,c,d,e}	{a}	{ a,c,d,e}
B5	{a,d}	∅	{ a,c,d}	{ a,c,d}
B6	{a,c}	{e}	{a,c}	{a}
B7	∅	∅	∅	∅

3. 求解结果如下：

	GEN	**KILL**	**IN**	**OUT**
B1	{1}	∅	∅	{1}
B2	{2}	{9}	{1, 4, 5, 7, 8, 9, 14}	{1, 4, 5, 7, 8, 14, 2}
B3	{4}	{1, 14}	{1, 4, 5, 7, 8, 14, 2}	{4, 5, 7, 8, 2}
B4	{5, 7, 8, 9}	{2, 11,13}	{1, 4, 5, 7, 8, 2,9,11,13,14}	{1, 4, 5, 7, 8, 9, 14}
B5	{11}	{7}	{1, 4, 5, 7, 8, 9, 11, 14}	{1, 4, 5, 8, 9, 11, 14}

B6	{13, 14}	{1,4,6,8}	{1, 4, 5, 8, 9, 11, 14}	{ 5, 9, 11, 13, 14}
B7	∅	∅	{ 5, 9, 11, 13, 14}	{ 5, 9, 11, 13, 14}

4. 该流图范围内，变量 a 在（11）的 UD 链{（1），（4），（14）}.

5. 该流图范围内，变量 c 在 （2） 的 DU 链{（3）}.

A5. 试对以下基本块：

B:=3
D:=A+C
E := A*C
F := D+E
G := B*F
H := A+C
I := A*C
J := H+I
K := B*5
L := K+J
M := L

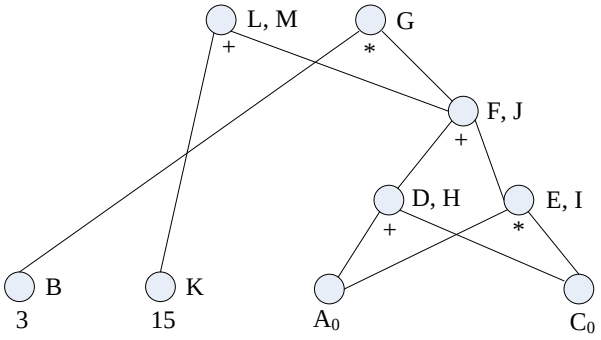
应用DAG对它们进行优化，并就以下两种情况分别写出优化后的TAC语句序列：

1. 假设只有G、L、M在基本块后面还要被引用；

2. 假设只有L在基本块后面还要被引用。

参考解答：

DAG：



1. 假设只有G、L、M在基本块后面还要被引用；

D:=A+C
E := A*C
F := D+E
G := 3*F

L := 15+F

M := L

2. 假设只有L在基本块后面还要被引用。

D:=A+C

E := A*C

F := D+E

L := 15+F

（注：因选取的变量不同，形式上可以有所不同）

A6. 给定如下文法 $G[S]$：

（1）$S \to P$

（2）$P \to P\ P\ \wedge$

（3）$P \to P\ P\ \vee$

（4）$P \to P\ \neg$

（5）$P \to \underline{id}$

其中，$\wedge$、$\vee$、$\neg$ 分别代表命题逻辑与、或、非等运算符单词，$\underline{id}$ 代表标识符单词。如下是以 $G[S]$ 为基础文法的一个 S 翻译模式：

（1）$S \to P$ { *print* (*P*.*s*) }

（2）$P \to P_1\ P_2\ \wedge$ { $P.s := f(P_1.s, P_2.s)$ }

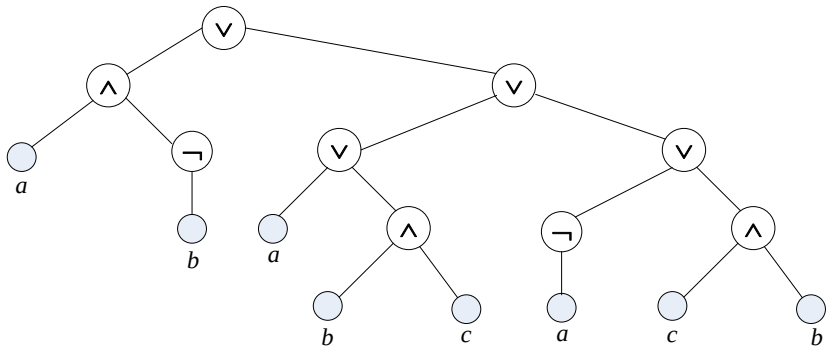
（3）$P \to P_1\ P_2\ \vee$ { $P.s := f(P_1.s, P_2.s)$ }

（4）$P \to P_1\ \neg$ { $P.s := g(P_1.s)$ }

（5）$P \to \underline{id}$ { $P.s := 1$ }

其中，*print* 为显示函数，f 和 g 为其他语义函数。

1. 文法 $G[S]$ 可用于识别后缀形式（逆波兰式）的命题表达式。例如，输入串 $a\ b\ \neg\ \wedge\ a\ b\ c\ \wedge\ \vee\ a\ \neg\ c\ b\ \wedge\ \vee\ \vee\ \vee$ 对应于中缀式 $(a \wedge \neg\ b) \vee (\ (\ a \vee (b \wedge c)) \vee (\ \neg\ a \vee (\ c \wedge b\ )\ )$，以下是该命题表达式对应的表达式树：



如果上述 S 翻译模式中 *print* (*P*.*s*) 的含义是显示由 P 所识别后缀命题表达式

所对应的表达式树根节点对应的Ershov 数（ Ershov number），请给出语义函数 f 和 g 的具体定义。

2. 假设在一个简单的基于寄存器的机器 M 上进行表达式求值，除了load/store指令用于寄存器值的装入和保存外，其余操作均由下列格式的指令完成：

    OP　reg0, reg1, reg2

    OP　reg0, reg1

    其中，reg0, reg1, reg2处可以是任意的寄存器，OP 为运算符。运行这些指令时，对reg1和reg2的值做二元运算，或者对reg1的值做一元运算，结果存入reg0。对于load/store指令，假设其格式为：

    LD　reg, mem　　/* 取内存或立即数 mem 的值到寄存器 reg */

    ST　reg, mem　　/* 存寄存器 reg 的值到内存量 mem */

    我们假设M机器指令中，逻辑运算 ∧、∨、¬ 分别用助记符 AND、OR、NOT 表示。

    试说明，为上一小题图中所示的表达式树生成机器 M 指令序列时，需要寄存器数目的最小值 n =？假设这些寄存器分别用助记符R_0，R_1，...，和R_{n-1} 表示，试采用课程中所介绍的方法生成该命题表达式的目标代码（仅含指令AND、OR、NOT、LD和ST，以及仅用寄存器 R_0，R_1，...，和R_{n-1}）。（给出算法执行结果即可，不必进行目标代码优化）

参考解答：

1. （等效结果亦可，如更换自变量、右端重写为结果等价的表达式）

$$f(P_1.s, P_2.s) = \text{if } P_1.s > P_2.s \text{ then } P_1.s$$
$$\textbf{else if } P_2.s > P_1.s \text{ then } P_2.s$$
$$\textbf{else } P_1.s + 1$$

$$g(P_1.s) = P_1.s$$

2. n =3。

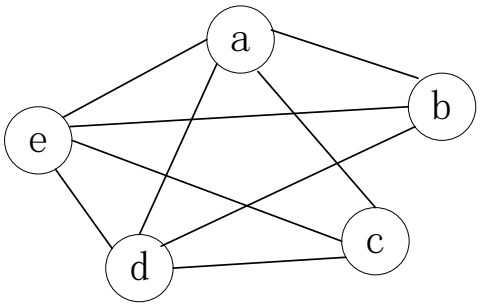
假设这些寄存器分别用 R_0，R_1，和R_2 表示，生成该命题表达式的目标代码如下：（等效的代码，或经过某些优化，均可）

```
LD      R0, b
LD      R1, c
AND     R0, R0, R1
LD      R1, a
OR      R0, R1, R0
LD      R1, c
LD      R2, b
AND     R1, R1, R2
LD      R2, a
NOT     R2, R2
```

```
OR      R1, R2, R1
OR      R0, R1, R0
LD      R1, a
LD      R2, b
NOT     R2, R2
AND     R1, R1, R2
OR      R0, R1, R0
```

A7. 对于题A4图中的流图，给出相应的寄存器相干图。按照课程所介绍的基本的图着色全局寄存器分配算法，若要保证图着色过程中不会出现将寄存器泄漏到内存中的情形，那么可供分配的物理寄存器的最小数目分别是多少？。

参考解答：对于题 A4 图中的流图，寄存器相干图为

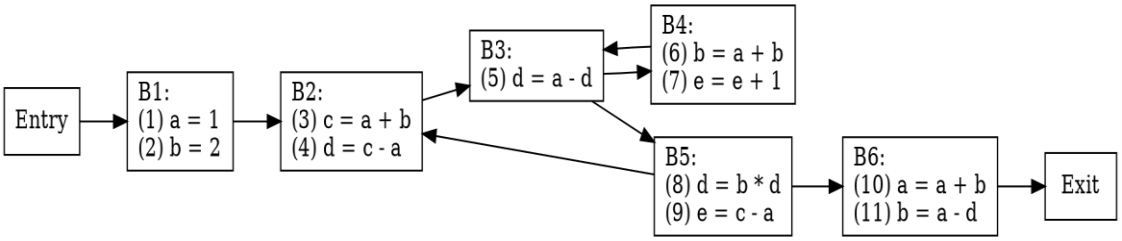


若要保证图着色过程中不会出现将寄存器泄漏到内存中的情形，那么可供分配的物理寄存器的最小数目是4。

A8.

数据流分析在编译器中后端的应用非常广泛，除了课堂上讲到的两种数据流分析（活跃变量分析和到达-定值分析）之外还有很多，它们的具体定义和功能不尽相同，但是大致过程都是对数据流方程的求解，且求解方式都是类似的。一般来说，只要给出数据流方程，以及迭代求解方程的初始值，就可以采用统一的迭代算法来求解方程。下面的问题中，我们都不会给出求解方程的迭代算法，请大家自己根据课上所学知识进行类比。

在下面的两个小题中都使用这个数据流图来进行计算。流图中的 ***Entry*** 和 ***Exit*** 是两个虚拟的基本块，***Entry*** 是程序的唯一入口，***Exit*** 是程序的唯一出口；除此以外，你可以认为它们就是两个不包含任何指令的普通基本块。流图省略了基本块间的跳转语句，它们是否存在对本题的作答无影响。



1.

课堂上定义了数据流图中的”支配节点“的概念。为了计算数据流图中每个基本块的支配

节点集，我们可以应用数据流分析的方法。定义数据流方程如下：

$$Out(B) = In(B) \cup \{B\}$$
$$In(B) = \cap\ Out(B'),\quad B' \in pred(B)$$

其中 $pred(B)$ 为 B 的前驱基本块的集合。

方程的初值为：

$$In(B) = \emptyset$$
$$Out(B) = U,\quad B \neq Entry$$
$$Out(Entry) = \{Entry\}$$

其中 $\emptyset$ 为空集；U 为全集，即所有基本块组成的集合。

试填空：

（a）这个数据流分析属于 _____ 数据流分析。 （填：前向/后向）

（b）$In(B_4)$= _____，$Out(B_5)$ = _____。

（c）每个基本块的支配节点集就是它的 _____ 集合。（填：In/Out）。

2.

实际编译器中，一种常见且重要的优化手段是部分冗余消除(PRE，Partial Redundancy Elimination)。为了实现 PRE，需要使用多种数据流分析，其中一种称为预期执行的表达式分析（Anticipated Expression Analysis）。

对于一个程序点，如果从这一点处开始的**所有路径**都会计算某个表达式的值，并且从这一点到表达式的计算点的路径上，这个表达式的所有操作数都没有被重新定值，则说这个表达式在这一点处被预期执行。预期执行的表达式分析就是希望找出每个程序点处预期执行的表达式的集合。

定义数据流方程如下：

$$In(B) = Gen(B) \cup (\ Out(B) - Kill(B)\ )$$
$$Out(B) = \cap\ In(B'),\quad B' \in succ(B)$$

其中 $succ(B)$ 为 B 的后继基本块的集合。

方程的初值为：

$$Out(B) = \emptyset$$
$$In(B) = U,\quad B \neq Exit$$
$$In(Exit) = \emptyset$$

其中 $\emptyset$ 为空集；U 为全集，即程序中出现过的所有表达式的集合。在本题中，

$$U = \{\ a + b,\ c - a,\ a - d,\ e + 1,\ b * d\ \}$$

方程中涉及到的四个集合的定义如下：

- $Gen(B)$：B 中计算过，且从计算点到 B 的入口都没有操作数被重新定值的表达式的集合；对于一条语句 $x = x + y$，应该把它看作先计算 $x + y$ ，再对 x 定值

- ***Kill(B)***：以 B 中定值的任一变量作为操作数，且在程序中出现过的表达式的集合
- ***In(B)***：B 入口处预期执行的表达式的集合
- ***Out(B)***：B 出口处预期执行的表达式的集合

试填空：

（*a*）这个数据流分析属于 _____ 数据流分析。 （填：前向/后向）

（*b*）***Gen(B_2)*** = _____，***Kill(B_3)*** = _____，***In(B_4)*** = _____，***Out(B_1)*** = _____。

（*c*）在 PRE 中，预期执行的表达式分析可以用来确定哪些位置可以放置一个表达式。可以放置的条件是：这个表达式在这一点处被预期执行（否则可能会导致原程序中没有执行过的计算在优化后的程序中执行了）。据此，判断 ***B_3*** 中 (5) 后面是否可以放置表达式 ***a + b***：____，***B_5*** 中 (8) 和 (9) 之间是否可以放置表达式 ***a - d***：____ 。（填: 是/否）

参考解答：

1. （*a*）这个数据流分析属于__前向__ 数据流分析。（填：前向/后向）

（*b*）$In(B_4)$ =_$\{Entry, B_1, B_2, B_3\}$_，$Out(B_5)$ =_$\{Entry, B_1, B_2, B_3, B_5\}$_。

（*c*）每个基本块的支配节点集就是它的_Out_集合（填In/Out）。

2. （a）这个数据流分析属于_后向_（填前向/后向）数据流分析。

（*b*）$Gen(B_2)$ =_$a+b$_，$Kill(B_3)$ =_$a-d, b*d$_，

$In(B_4)$ =_$a+b, c-a, a-d, e+1$_，$Out(B_1)$ =_$a+b$_。

（*c*） 在 PRE 中，预期执行的表达式分析可以用来确定哪些位置可以放置一个表达式。可以放置的条件是：这个表达式在这一点处被预期执行（否则可能会导致原程序中没有执行过的计算在优化后的程序中执行了）。据此，判断B_3中(5)后面处是否可以放置表达式$a+b$：_是_，B_5中(8)(9)之间是否可以放置表达式$a-d$：_否_。（填：是/否）。