1.

(1)

函数 g

old ebp
%gs(14)
a[9]
a[8]
a[7]
a[6]
a[5]
a[4]
a[3]
a[2]
a[1]
a[0]

函数 f

old ebp
%gs(14)
b[1]
b[0]

(2)

```
noob@noob-virtual-machine:~/workspace/lab04$ ./array_init
input student id :
161130118
8   -48
```

我们观察到该程序 main 函数中，先调用了函数 g，函数 g 结束后再调用了函数 f，所以函数 g 和函数 f 应该先后共享了一部分栈帧，进而可推测是该结果的出现是因为函数 f 使用了函数 g 分配的局部变量导致的.

通过进一步观察，可得到在函数 g 中调用的 init 函数里通过一个字符串 temp[10]接收我们输入的学号，其末尾会自动添加一个字符串结束符('\0')，该符号在 ascii 编码中被编码为十进制的 0.

在后续初始化数组 a 的过程中，数组 a 中的元素满足:

$$a[i] = temp[i] - '0'$$

而'0'的 ascii 编码十进制值为 48，于是数组 a 里的数据如下:

a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]	a[9]
1	6	1	1	3	0	1	1	8	-48

函数 g 结束后，该栈帧上分配的内容并未消失，在接下来函数 f 的调用中，由于函数 f 未对数组 b 进行初始化而 b[1]和 a[9], b[0]和 a[8]先后共享了同一位置，所以在打印数组 b 的值时，实际打印的是之前函数 g 分配给 a[8]和 a[9]的值.

危害: 可导致数据泄露，如本例. 有些时候也会导致程序得不到预期结果.

2.

(1)

$$A[i][j][k] = addr(A) + (i * S * T + j * T + k) * 4$$

(2)

	%eax	%ecx	%edx
3	4	6	16
4	4	51	16
5	4	51	4
6	8	51	4
7	8	51	8
8	64	51	8
9	56	51	8
10	56	51	9282
11	56	51	9338
12	6	51	9338
13	6	51	9344
14	161130118	51	9344
15	161130118	51	9344
16	378560	51	9344

(3)

R = 520, S = 13, T = 14.

4.

(1)

0 4 0 0 4 8 12.

(3)

```
noob@noob-virtual-machine:~/workspace/lab04$ ./diff
array address:
bfad48cc        bfad48dc        bfad48ec

list address:
8ad20c8 8ad2080 8ad2048 8ad2020 8ad2008
```

数组使用的是栈区的静态内存，而链表使用的是堆区的动态内存.

差异: 静态内存连续, 动态内存不连续.