1. (1) 查看函数 g 和 f 的反汇编代码，分别给出函数 g 和 f 中数组 a，b 在栈上的分布，在下图中给出 a[0]-a[9]以及 b[0]、b[1]位置。

函数 g

| old rbp |
|---|
| %fs(28) |
| a[9] |
| a[8] |
| a[7] |
| a[6] |
| a[5] |
| a[4] |
| a[3] |
| a[2] |
| a[1] |
| a[0] |

函数 f

| old rbp |
|---|
| %fs(28) |
| b[1] |
| b[0] |
| |
| |
| |
| |
| |
| |
| |
| |

(2) 运行程序，程序的输入为 9 位学号，观察输出。请详细解释为什么 b[0]和 b[1]是这两个值。说明使用未初始化的程序局部变量的危害。

```
river@ubuntu:~/Desktop/workspace/lab04$ ./array_init
input student id:
191220154
4 -48
```

函数 f 未对 b[M]进行初始化，f 中 b[0]与 b[1]在栈中分配的地址恰好为 g 中（init 中）a[8]与 a[9]的地址，则 b[0] = a[8] = '4' – 48 = 4，b[1] = '\0' – 48 = -48。

程序中的局部变量被分配在栈中，栈中的空间可能被其它的函数使用过，已经有一些别的数据，而这些数据对新的函数而言可能是没有任何意义的"垃圾值"，运行结果难以预期。

2. 1）将数组地址计算扩展到三维，给出 A[i][j][k]地址的表达式。
(A 的定义为 int A[R][S][T]，sizeof(int)=4，起始地址设为 addr(A))

A[i][j][k]的地址为 addr(A) + (i * S * T + j * T + k) * 4

2) 使用命令 gdb ./3_d_array 启动 gdb 调试。在 store_ele 函数入口设置断点，以自己的 9 位学号为输入，运行程序。在 store_ele 函数中单步执行，并打印出每步汇编指令执行后寄存器 eax、ecx、edx 的值，根据实验结果填写每条指令运行后的结果。

| | %eax | %ecx | %edx |
|---|---|---|---|
| 3 | 0x7 (7) | 0x0 (0) | 0x16 (22) |
| 4 | 0x7 (7) | 0x160 (352) | 0x16 (22) |
| 5 | 0x7 (7) | 0x160 (352) | 0x7 (7) |
| 6 | 0xe (14) | 0x160 (352) | 0x7 (7) |
| 7 | 0xe (14) | 0x160 (352) | 0xe (14) |
| 8 | 0x70 (112) | 0x160 (352) | 0xe (14) |
| 9 | 0x62 (98) | 0x160 (352) | 0xe (14) |
| 10 | 0x62 (98) | 0x160 (352) | 0xfa40 (64064) |
| 11 | 0x62 (98) | 0x160 (352) | 0xfaa2 (64162) |
| 12 | 0x0 (0) | 0x160 (352) | 0xfaa2 (64162) |
| 13 | 0x0 (0) | 0x160 (352) | 0xfaa2 (64162) |
| 14 | 0xb65c9ba (191220154) | 0x160 (352) | 0xfaa2 (64162) |
| 15 | 0xb65c9ba (191220154) | 0x160 (352) | 0xfaa2 (64162) |
| 16 | 0x5c6c0 (378560) | 0x160 (352) | 0xfaa2 (64162) |

3）根据以上内容确定 R、S、T 的取值

i = 352，j = 7，k = 0，地址偏移量为(352 * 182 + 7 * 14 + 0) * 4
即 S * T = 182，T = 14，又 sizeof(A) = R * S * T * 4 = 378560
则有 R = 520，S = 13，T = 14。

3. int recursion(int x) {
    if(x <= 2)
        return 1;
    else
        return recursion(x - 1) + recursion (x - 2);
}

4. 1）确定下列字节的偏移量。

e1.p 0
e1.x 8
y 0
y[0] 0
y[1] 4
y[2] 8
next 16

2）下面的过程（省略一些表达式）是对链表进行操作，链表是以上述结构作为元素的。现有 proc 函数主体的汇编码，查看汇编代码，并根据汇编代码补全 proc 函数中缺失的表达式，并保存为 proc.c。（不需要进行强制类型转换）

```
void proc(struct ele *up) {
    up->next->y[0] = *(up->e1.p) + up->e1.x;
}
```

3）有以下 main 函数，该 main 函数中声明了一数组和一链表并打印了每个元素的地址，查看地址，并解释产生原因，体会数组与链表分别使用静态内存和动态内存的差异。

```
river@ubuntu:~/Desktop/workspace/lab04$ ./proc
array address:
488b5db0        488b5dc8        488b5de0

list address:
415043a0        41504320        415042c0        41504280        41504260
```

数组使用静态内存，每个元素占据内存空间为 24 字节，地址从小到大连续分布。
链表使用动态内存，每个元素占据内存空间仍为 24 字节，但地址的分布不连续。