1 函数局部存储的管理

在每次递归调用中，数组 a 的内存地址减 0x60，在除了第一次 main 函数的调用中，数组 a 对应元素的值均相同（除在本次调用中被赋值的元素）。在每次调用函数时，都会进行初始化，由于函数相同，初始化过程相同，导致内存区中存储了相同的数据。在定义数组 a 时未进行初始化，程序不会清除内存区的数据，因此在本次调用中未被赋值的元素均相同。

2 二维数组的内存管理与系统限制

程序失败最小的 a 为 1024。32 位系统每个进程的地址空间有限，数组过大可能会导致超出进程地址空间限制导致段错误。在现代编译器中，编译器会进行程序优化、数据分段、虚拟内存管理等方法，避免过大数组的出现。

通过修改/etc/security/limits.conf 文件，在最后添加 <domain> soft stack <value> 和 <domain> hard stack <value>，将 value 设置为大于 8192 的值，修改栈的大小，然后重启虚拟机。

3 数组的访问方式

(a+i)-a 和 (b+i)-b 理应打印第 i 个元素相对于数组第 0 个元素的偏移地址，但实际程序输出的是 i 的值本身。

程序会返回储存在地址 c+200 的元素，以 short int 解析，并输出。可能会导致程序异常访问不应访问的内存地址，导致数据程序出错。

通过递归调用函数 fill，每次填充二维数组的一圈，最终将二维数组根据要求填充完整。