**姓名：曾泇睷 学号：1820221053**

1、回答问题：

根据程序1，试分析：

1.1全局变量、局部变量、静态变量的特点是什么

**全局变量位于静态数据区中。局部变量保存在动态数据区的栈中，只有在所在函数被调用时才动态地为变量分配存储单元。静态变量属于静态存储方式，其存储空间为内存中的静态数据区（在静态存储区内分配存储单元），该区域中的数据在整个程序的运行期间一直占用这些存储空间（在程序整个运行期间都不释放），也可以认为是其内存地址不变，直到整个程序运行结束。**

根据程序2，试分析：

2.1 根据盘子数的增长，hanoi函数被调用的次数是如何变化的？

**2n-1**

2.2 设盘子的移动次数为H(n）。汉诺塔问题的[递归](https://lexue.bit.edu.cn/mod/programming/view.php?id=400801" \o "递归)表达式：

H⑴ = 1

H(n) = 2\*H(n-1）+1 (n>1）

那么就能得到H(n）的一般式：

H(n) = 2^n - 1 (n>0)

根据一般式，可以不使用[递归](https://lexue.bit.edu.cn/mod/programming/view.php?id=400801" \o "递归)，就能得到盘子的移动次数。

请根据这一现象，分析[递归](https://lexue.bit.edu.cn/mod/programming/view.php?id=400801)方法的优缺点。

**好处：代码简洁清晰，可读性更好。**

**坏处：由于递归需要系统推栈，所以空间消耗要比非递归代码要大得多**

2.3 请比较[递归](https://lexue.bit.edu.cn/mod/programming/view.php?id=400801" \o "递归)与迭代两种方法，包括控制结构，终止测试，计算代价，程序直观度。

**控制结构：迭代用重复结构，而递归用选择结构。**

**终止测试：迭代在循环条件失败时终止，递归在遇到基本情况时终止。**

**如果递归永远无法回到基本情况，则发生无穷递归。递归函数通过调用自身来完成任务，而在每次调用自身时减少任务量，而迭代是循环的一种形式，这种循环不是用户输入控制，每次迭代步骤必须将剩余的任务减少。**

根据程序3，试分析：

3.1   函数swap能否实现对main函数中的a，b值的交换呢？为什么？

**不能，因为main传递进swap的参数只在swap内部进行转换，并不会影响main函数的值。**

3.2  如果把Swap中的a和b分别改成x和y，会发生什么变化吗？如果将它们换成别的名称呢？函数的形参名需要和main函数中的实参名保持一致吗？

**不会发生任何变化，别的名字也一样；不需要保持一致。**

3.3   请你编写一个新的swap函数，要求使用传递变量地址的方法，实现对main函数中a，b值的交换。

**void Swap(int \*a, int \*b)**

**{**

**int temp;**

**temp = \*a;**

**\*a = \*b;**

**\*b = temp;**

**printf("In Swap():a = %d, b = %d\n", \*a, \*b);**

**}**

表1        程序1的变量说明

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Printf语句的序号 | 变量X的序号 | 变量X的值 | 变量x的类型 | 变量x的作用域 |
| A | 2 | 5 | int | 块作用域 |
| B | 3 | 7 | int | 局部作用域 |
| C1 | 4 | 25 | int | 局部作用域 |
| C2 | 4 | 26 | int | 局部作用域 |
| D1 | 5 | 50 | int | 局部作用域 |
| D2 | 5 | 51 | int | 局部作用域 |
| E1 | 1 | 1 | int | 全局作用域 |
| E2 | 1 | 10 | int | 全局作用域 |
| F1 | 4 | 25 | int | 局部作用域 |
| F2 | 4 | 26 | int | 局部作用域 |
| G1 | 5 | 51 | int | 局部作用域 |
| G2 | 5 | 52 | int | 局部作用域 |
| H1 | 1 | 10 | int | 全局作用域 |
| H2 | 1 | 100 | int | 全局作用域 |
| I | 2 | 5 | int | 块作用域 |

表2    程序2的空缺语句

|  |  |
| --- | --- |
| 序号 | 语句 |
| 【1】 | void hanoi(int n, char A ,char B ,char C) |
| 【2】 | void move(char a,char b) |
| 【3】 | count++ |
| 【4】 | hanoi(n-1,A,C,B) |
| 【5】 | hanoi(n-1,B,A,C) |

表3  程序2的运行情况分析

|  |  |
| --- | --- |
| 项数number | Hanoi函数的调用次数 |
| 0 | 0 |
| 1 | 1 |
| 2 | 3 |
| 3 | 7 |
| 4 | 15 |
| 5 | 31 |
| 6 | 63 |
| 7 | 127 |
| 10 | 1023 |
| 15 | 32767 |

表4   程序3的运行分析

|  |  |  |  |
| --- | --- | --- | --- |
| 步骤 |  | 变量a的值 | 变量b的值 |
| 1 | main函数（调用swap前） | 5 | 7 |
| 2 | swap函数（return语句之前） | 7 | 5 |
| 3 | main函数（调用swap后） | 5 | 7 |