

程序设计专题



主讲教师：张引

助教：田沈晶，熊海辉

浙江大学计算学院，2018年3月6日

程序设计专题

● 教学目的

- 进一步提高学生应用高级程序设计语言进行程序设计的能力，特别是在复杂数据组织以及基本算法设计方面的能力。
- 四个专题内容为：模块化程序设计与递归、结构/链表、图形程序设计基础、查找/排序与算法分析。

程序设计专题

● 教学安排（1学分）

- 每个专题理论课连续上2次
- 每个专题汇报课1次

参考资料



- ⌘ C语言程序设计(第3版)，何钦铭、颜晖主编，高等教育出版社
- ⌘ C程序设计基础课程设计，张引、何钦铭等，浙江大学出版社，2007.9
- ⌘ C语言的科学和艺术，罗伯茨(Eric S.Roberts)著，翁惠玉等译，机械工业出版社，2011.6
- ⌘ The C Programming Language, Second Edition, Brian W.Kernighan & Dennis M.Ritchie, 机械工业出版社（影印），Prentice-Hall International Inc
- ⌘ C语言程序设计经典实验案例集，何钦铭主编，高等教育出版社，2012.5

程序设计专题

■ 交流

□ QQ: 599168489

□ 教学平台网站:

<http://10.71.45.100>

■ 机房地点

□ 机房地点:西一教学楼的西面,建工学院的北面



2018春夏-程序设计专题
扫一扫二维码，加入该群。

程序设计专题

● 成绩评定

- 总评= 期末考试50%
+ 课后练习作业 20%
+ Project和课堂报告 25 （2个Project/专题）
+ 课堂表现5%
- 大程序加分【最多5分，且限制在平时成绩50分以内】

● Project课堂汇报

- 2人一组自由组队
- 每组随机（现场）选代表做报告，其他组现场打分



程序设计专题一：

结构化程序设计与递归函数

专题要点



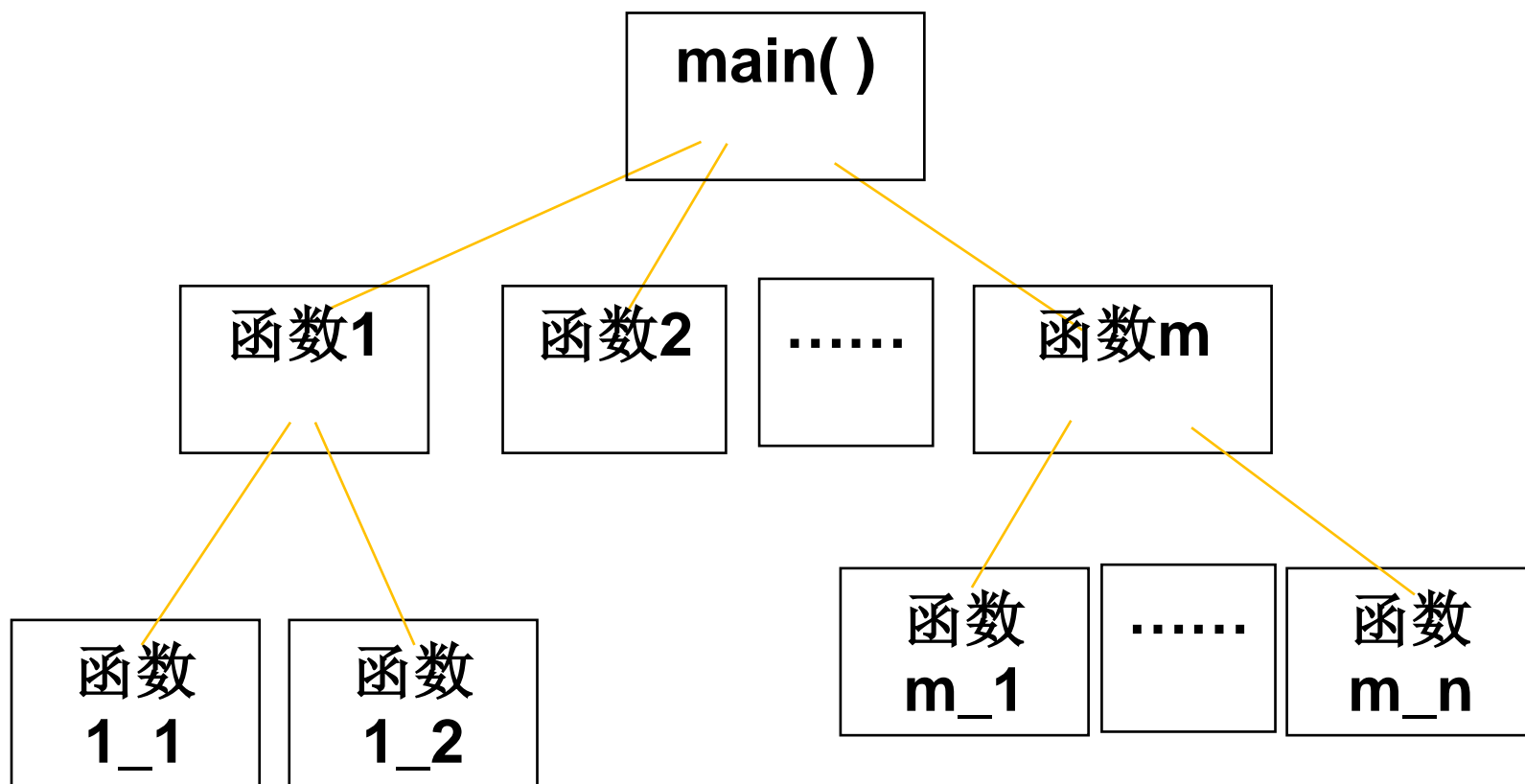
- ⌘ 用结构化程序设计的思想解决问题
- ⌘ 将多个函数组织起来，将多个源程序文件组织起来
- ⌘ 理解程序设计规范及其重要性
- ⌘ 函数嵌套求解复杂的问题
- ⌘ 理解和使用函数递归
- ⌘ 宏定义
- ⌘ 编译预处理

结构化程序设计



- ⌘ 使用结构化程序设计方法解决复杂的问题
 - ☑ 把大问题分解成若干小问题，小问题再进一步分解成若干更小的问题
 - ☑ 写程序时，用main()解决整个问题，它调用解决小问题的函数
 - ☑ 这些函数又进一步调用解决更小问题的函数，从而形成函数的嵌套调用

程序结构



函数定义

- ⌘ 好的函数名字：描述函数所做的所有事情。如：
 - ☒ checkOrderInfo(...)
 - ☒ calcMonthlyRevenues(...)
- ⌘ 一个函数只实现一个功能
- ⌘ 函数参数：
 - ☒ 按照输入-修改-输出的顺序排列参数
 - ☒ 考虑对参数采用某种表示输入、修改、输出的命名规则
 - ☒ 使用所有的参数
 - ☒ 把状态或出错变量放在最后
 - ☒ 不要把函数的参数用作工作变量
 - ☒ 在接口中对参数的假定加以说明
 - ☒ 尽可能少的参数（限制在大约7个以内）

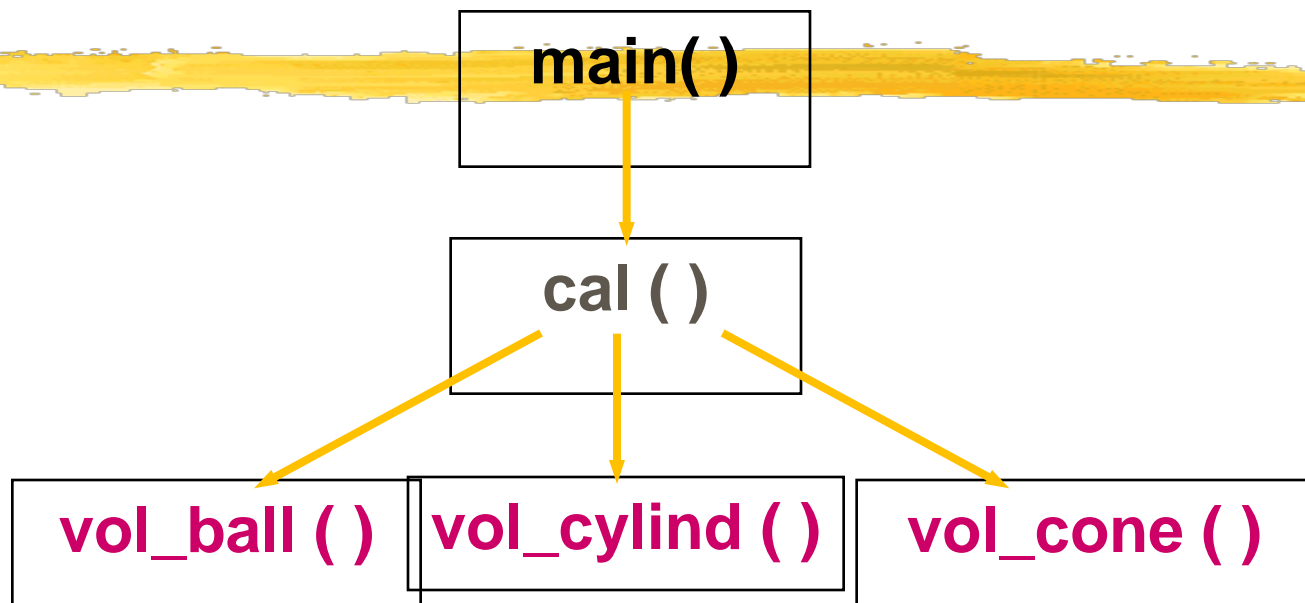
实例：计算常用圆形体体积

设计一个常用圆形体体积计算器，采用命令方式输入1、2、3，分别选择计算球体、圆柱体、圆锥体的体积，并输入计算所需相应参数。

⌘分析：

- ☒输入1、2、3选择计算3种体积，其他输入结束计算
- ☒设计一个控制函数cal()，经它辨别圆形体的类型再调用计算球体、圆柱体、圆锥体体积的函数
- ☒设计单独的函数计算不同圆形体的体积

实例：计算常用圆形体体积——程序结构



⌘ 3层结构，5个函数

⌘ 降低程序的构思、编写、调试的复杂度

⌘ 可读性好

实例：计算常用圆形体体积——程序

```
#define PI 3.14159265
void cal ( int sel );
int main(void)
{
    int sel;
    while( 1 ){
        printf(" 1-计算球体体积\n");
        printf(" 2-计算圆柱体体积\n");
        printf(" 3-计算圆锥体体积\n");
        printf(" 其他 -退出运行\n");
        printf("请输入计算命令： ");
        scanf("%d",&sel);
```

```
        if (sel<1 || sel>3)
            break
        else
            cal(sel);
    }
    return 0;
}
```

实例：计算常用圆形体体积——程序

```
/* 主控函数 */
```

```
void cal ( int sel )
```

```
{
```

```
    double  vol_ball(void );
```

```
    double  vol_cylind(void );
```

```
    double  vol_cone(void );
```

```
    switch (sel) {
```

```
        case 1: printf("球体积为: %.2f\n", vol_ball( ));  
                break;
```

```
        case 2: printf("圆柱体积为: %.2f\n", vol_cylind( ));  
                break;
```

```
        case 3: printf("圆锥体积为: %.2f\n", vol_cone( ));  
                break;
```

```
    }
```

```
}
```

实例：计算常用圆形体体积——程序

```
/*球体体积函数*/  
double vol_ball(  
{  
    double r ;  
    printf("请输入球的半径: ");  
    scanf("%lf",&r);  
    return(4.0/3.0*PI*r*r*r);  
}
```


实例：计算常用圆形体体积——程序

```
/*圆柱体积函数*/  
double vol_cylind( )  
{  
    double r , h ;  
    printf("请输入圆柱的底圆半径和高: ");  
    scanf("%lf%lf",&r,&h);  
    return(PI*r*r*h);  
}
```

实例：计算常用圆形体体积——程序

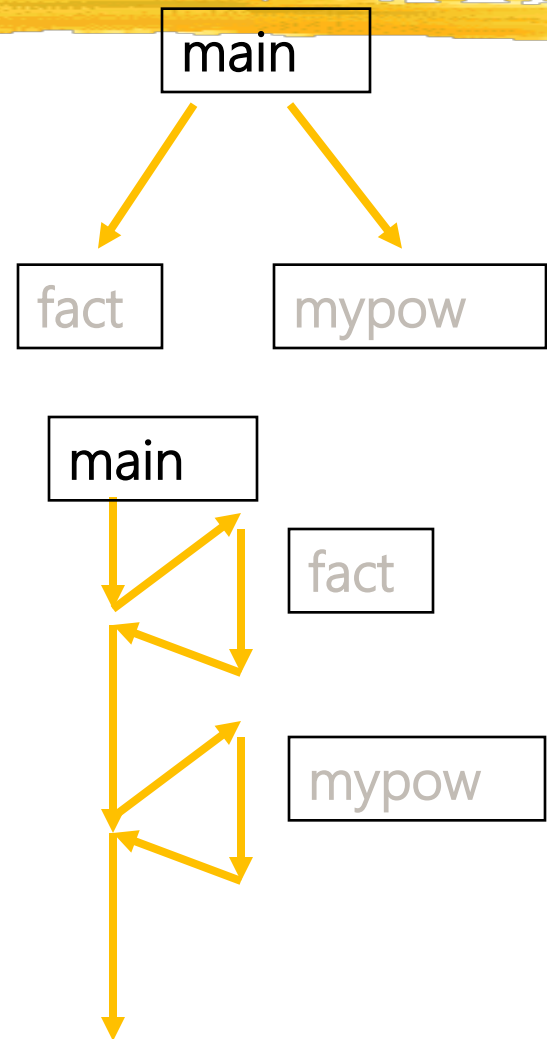
```
/*圆锥体积函数*/  
double vol_cone( )  
{  
    double r , h ;  
    printf("请输入圆锥的底圆半径和高: ");  
    scanf("%lf%lf",&r,&h);  
    return(PI*r*r*h/3.0);  
}
```

函数的嵌套调用

```
int main(void)
{
    .....
    y = fact(3);
    .....
    z = mypow(3.5, 2);
    .....
}

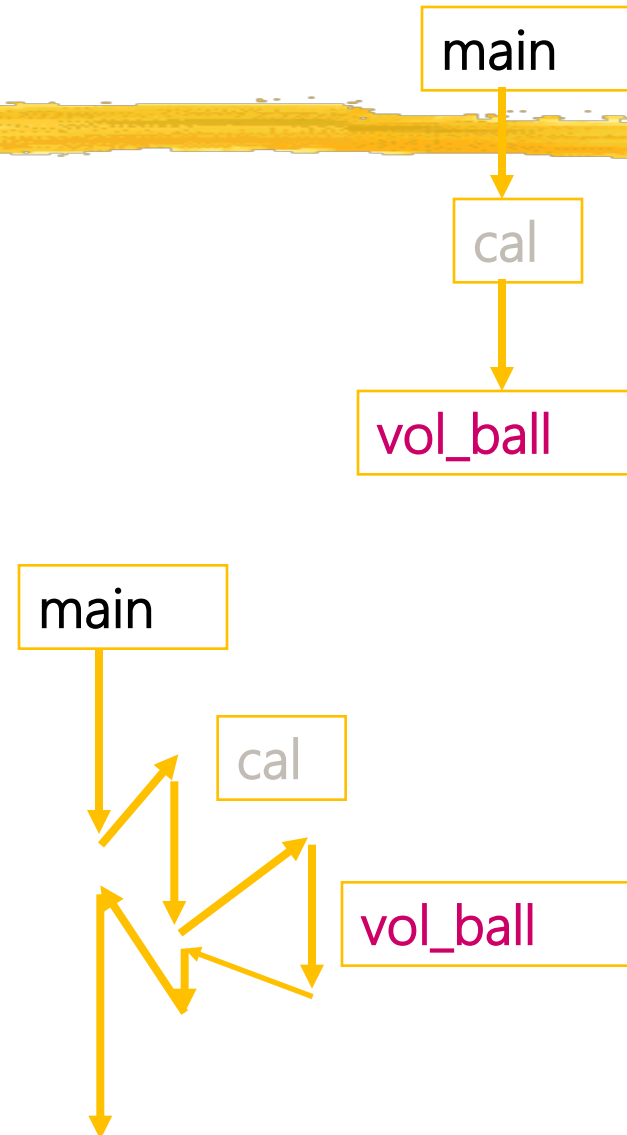
double fact(int n)
{
    .....
}

double mypow(double x, in n)
{
    .....
}
```



函数的嵌套调用

```
int main(void)
{
    .....
    cal (sel);
    .....
}
void cal (int sel)
{
    .....
    vol_ball()
    .....
}
double vol_ball( )
{
    .....
}
```



函数的嵌套调用



- ⌘ 在一个函数中再调用其它函数的情况称为函数的嵌套调用。
- ⌘ 如果函数A调用函数B，函数B再调用函数C，一个调用一个地嵌套下去，构成了函数的嵌套调用。
- ⌘ 具有嵌套调用函数的程序，需要分别定义多个不同的函数体，完成不同的功能，它们合起来解决复杂的问题。

递归

- ◆ 递归是指把一个**大问题**转化成**同样形式**但小一些的问题加以解决。
- ◆ **问题**：有 5 个人坐在一起，问第 5 个人多少岁？他说比第 4 个人大 2 岁。问第 4 个人岁数，他说比第 3 个人大 2 岁。问第 3 个人，又说比第 2 个人大 2 岁。问第 2 个人，说比第 1 个人大 2 岁。最后问第 1 个人， he 说是 10 岁。请问第 5 个人多大。

$$\text{age}(5) = \text{age}(4) + 2$$

$$\text{age}(4) = \text{age}(3) + 2$$

$$\text{age}(3) = \text{age}(2) + 2$$

$$\text{age}(2) = \text{age}(1) + 2$$

$$\text{age}(1) = 10$$

- ◆ 用数学公式表述如下：

$$\text{age}(n) = \begin{cases} 10 & (n = 1) \\ \text{age}(n-1) + 2 & (n > 1) \end{cases}$$

递推法与递归法求阶乘

⌘ 递推法

$$n! = 1 * 2 * 3 * \dots * n$$

函数fact(n)

```
for (result = 1, i = 1; i <= n; i++)  
    result = result * i;
```

⌘ 递归法

递归定义

$$n! = n * (n-1)! \quad (n > 1) \quad \leftarrow \text{递归式}$$

$$n! = 1 \quad (n = 0, 1) \quad \leftarrow \text{递归出口}$$

递归式
递归出口

递归函数 fact(n)

```
#include <stdio.h>
```

```
double fact(int n);
```

```
int main(void)
```

```
{ int n;
```

```
    scanf ("%d", &n);
```

```
    printf ("%f", fact (n) );
```

```
    return 0;
```

```
}
```

```
double fact(int n)
```

```
{
```

```
    double result;
```

```
    if (n==1 || n == 0)
```

```
        result = 1;
```

```
    else
```

```
        result = n * fact(n-1);
```

```
    return result;
```

```
}
```

编写递归函数，求n! 递归定义：

$n! = n * (n-1)! \quad (n > 1)$ ← 递归式

$n! = 1 \quad (n = 0, 1)$ ← 递归出口

函数fact自己调用自己
称作：递归调用

递归出口

递归式

~~fact(n)=n*fact(n-1);~~

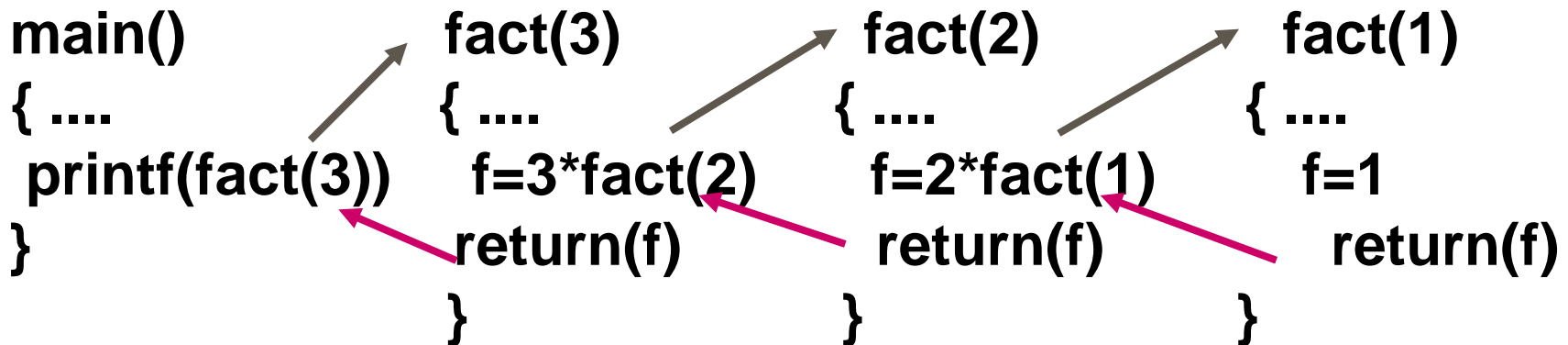
递归函数 fact(n)的实现过程

$$\text{fact}(3) = 3 * \text{fact}(2) = \mathbf{3 * 2 = 6}$$

同时有4个函数在运行，且都未完成

$$2 * \text{fact}(1) = \mathbf{2 * 1 = 2}$$

$$\text{fact}(1) = 1$$



```

#include <stdio.h>
double fact(int n);
int main(void)
{
    int n;
    scanf ("%d", &n);
    printf ("%f", fact (3) );
    return 0;
}

double fact(int n)
{
    double result;
    if (n==1 || n == 0)
        result = 1;
    else
        result = n * fact(n-1);
    return result;
}

```

result: 6	main()
result: 6	fact(3)
result: 2	fact(2)
result: 1	fact(1)

函数递归调用

- ⌘ 调用有着相同名字和相同代码的函数。
- ⌘ 调用时,分配参数和局部变量的存储空间
 - ☒ 退出时释放。
- ⌘ 随着递归调用的层层深入,存储空间的一端逐渐增加,然后随着函数调用的层层返回,存储空间的这一端又逐渐缩短。
 - ☒ 函数堆栈
- ⌘ 递归存在着可用堆栈空间过度使用的危险

函数递归调用

函数直接递归调用↵	函数间接递归调用↵	
<pre>int f(int x)↵ {↵ int y ;↵ ↵ y = f (x)↵ ↵ return y ;↵ }↵</pre>	<pre>int f(int x)↵ {↵ int y ;↵ ↵ y = g (x)↵ ↵ return y ;↵ }↵</pre>	<pre>int g(int x)↵ {↵ int z ;↵ ↵ z = f (x)↵ ↵ return z ;↵ }↵</pre>

递归程序设计

用递归实现的问题，满足两个条件：

⌘ 问题可以逐步简化成自身较简单的形式（递归式）

▢ 参数逐渐减小

⌘ 递归最终能结束(递归出口)

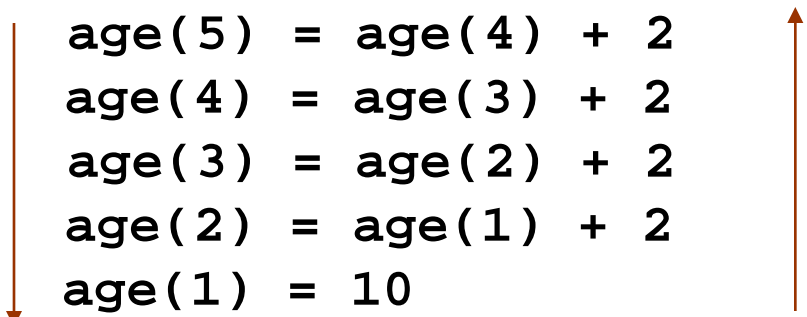
```
int factorial(int n)
{
    int nResult = n * factorial(n-1);
    result nResult;
}
```

这个函数会永远调用下去，直到操作系统为程序预留的栈空间耗尽程序崩溃为止，这称为**无穷递归**。

- ◆ 递归存在着可用堆栈空间过度使用的危险，这能导致严重的错误。在安全相关系统中强制规定：不能使用递归函数调用。

课堂练习:有5个人坐在一起，问第5个人多少岁？他说比第4个人大2岁。第4个人说比第3个人大2岁。第3个人说比第2个人大2岁。第2个人说比第1个人大2岁。第1个人说他10岁。请问第5个人多大？

【分析】


$$\begin{aligned} \text{age}(5) &= \text{age}(4) + 2 \\ \text{age}(4) &= \text{age}(3) + 2 \\ \text{age}(3) &= \text{age}(2) + 2 \\ \text{age}(2) &= \text{age}(1) + 2 \\ \text{age}(1) &= 10 \end{aligned}$$

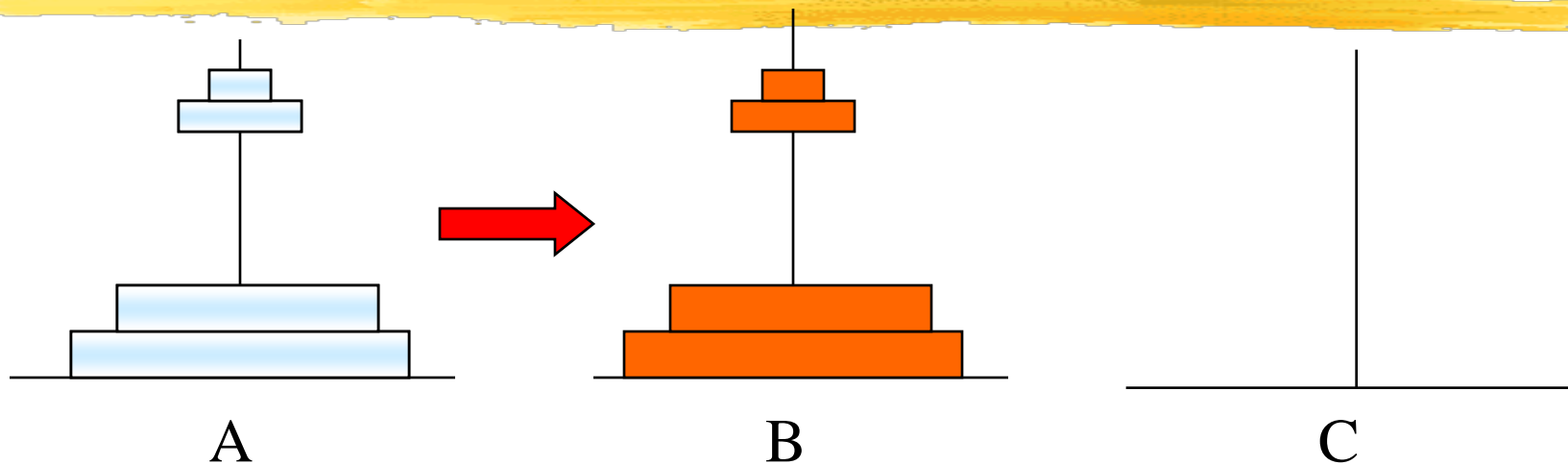
【求解】

$$\text{age}(n) = \begin{cases} 10 & n = 1 \\ \text{age}(n-1) + 2 & n > 1 \end{cases}$$

```
#include <stdio.h>
int age(int n)
{
    int c;
    if (n == 1)
        c = 10;
    else
        c = age(n-1) + 2;
}

void main()
{
    printf("%d", age(5));
}
```

例: 汉诺塔问题



将**64** 个盘从座**A**搬到座**B**

- (1) 一次只能搬一个盘子
- (2) 盘子只能插在**A**、**B**、**C**三个杆中
- (3) 大盘不能压在小盘上

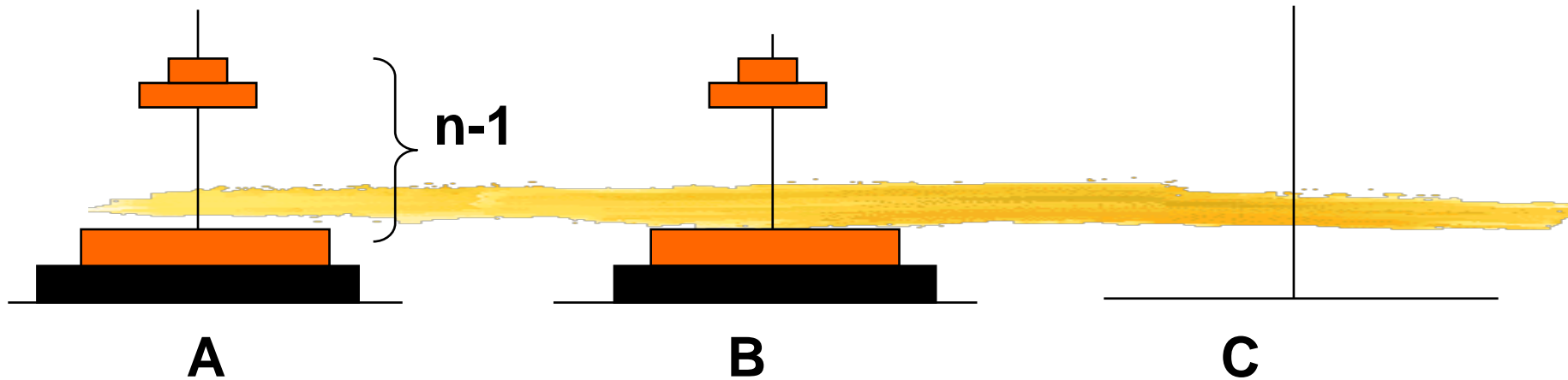
汉诺(Hanoi)塔问题解析

⌘ 递归方法的两个要点

- ☒ (1) 递归出口：一个盘子的解决方法；
- ☒ (2) 递归式子：如何把搬动64个盘子的的问题简化成搬动63个盘子的问题。

⌘ 把汉诺塔的递归解法归纳成三个步骤：

- ☒ $n-1$ 个盘子从座A搬到座C
- ☒ 第 n 号盘子从座A搬到座B
- ☒ $n-1$ 个盘子从座C搬到座B



算法:

hanio(n 个盘, $A \rightarrow B$, C 为过渡)

{ if ($n == 1$)

直接把盘子 $A \rightarrow B$

else{

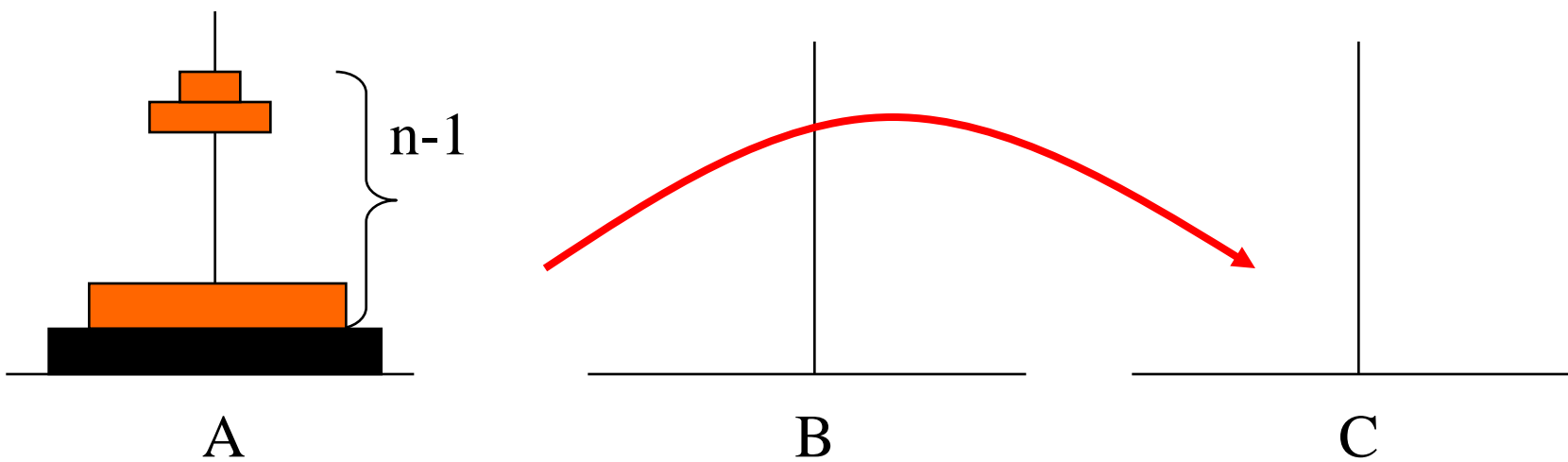
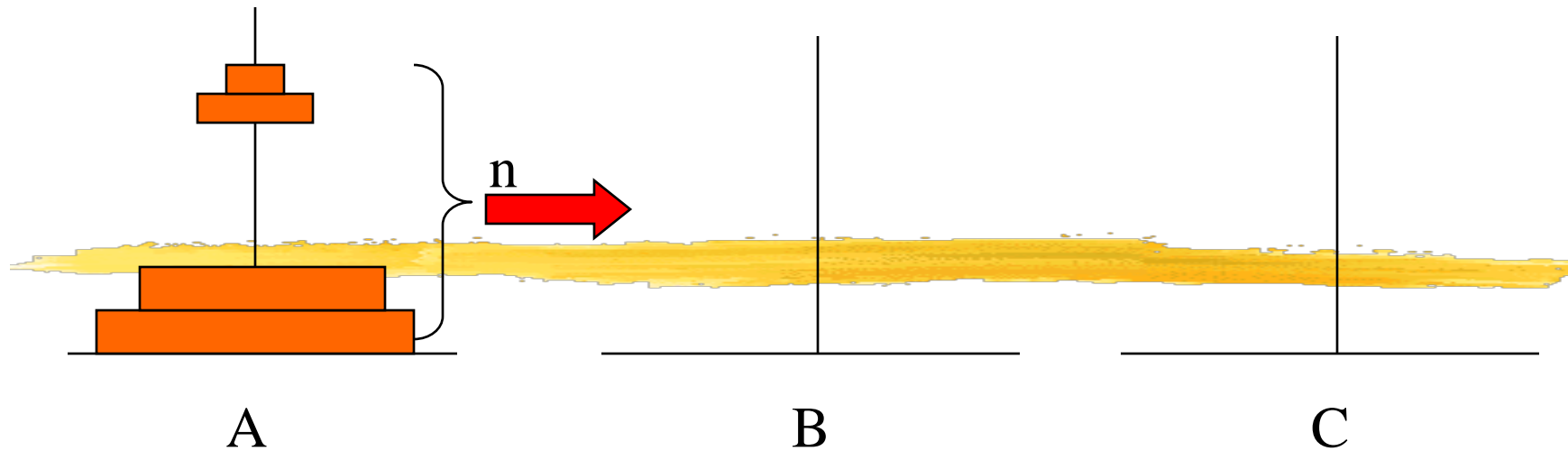
hanio($n-1$ 个盘, $A \rightarrow C$, B 为过渡)

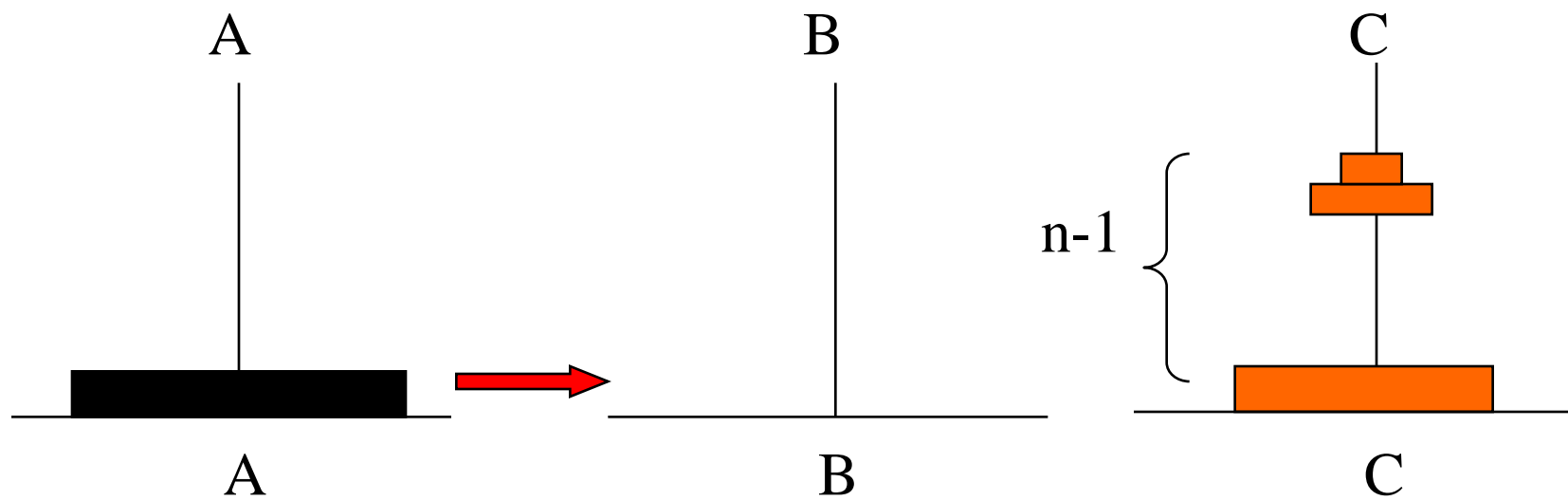
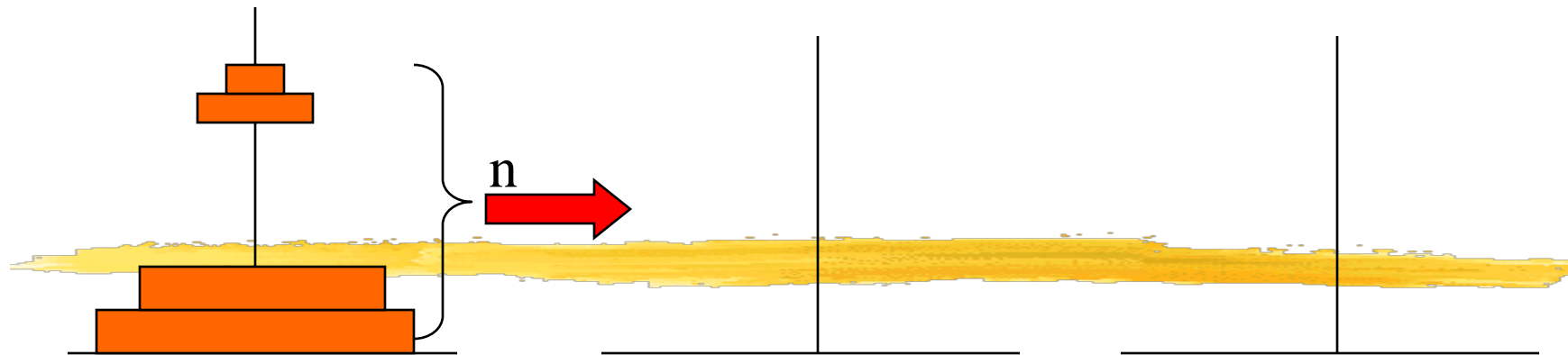
把第 n 号盘 $A \rightarrow B$

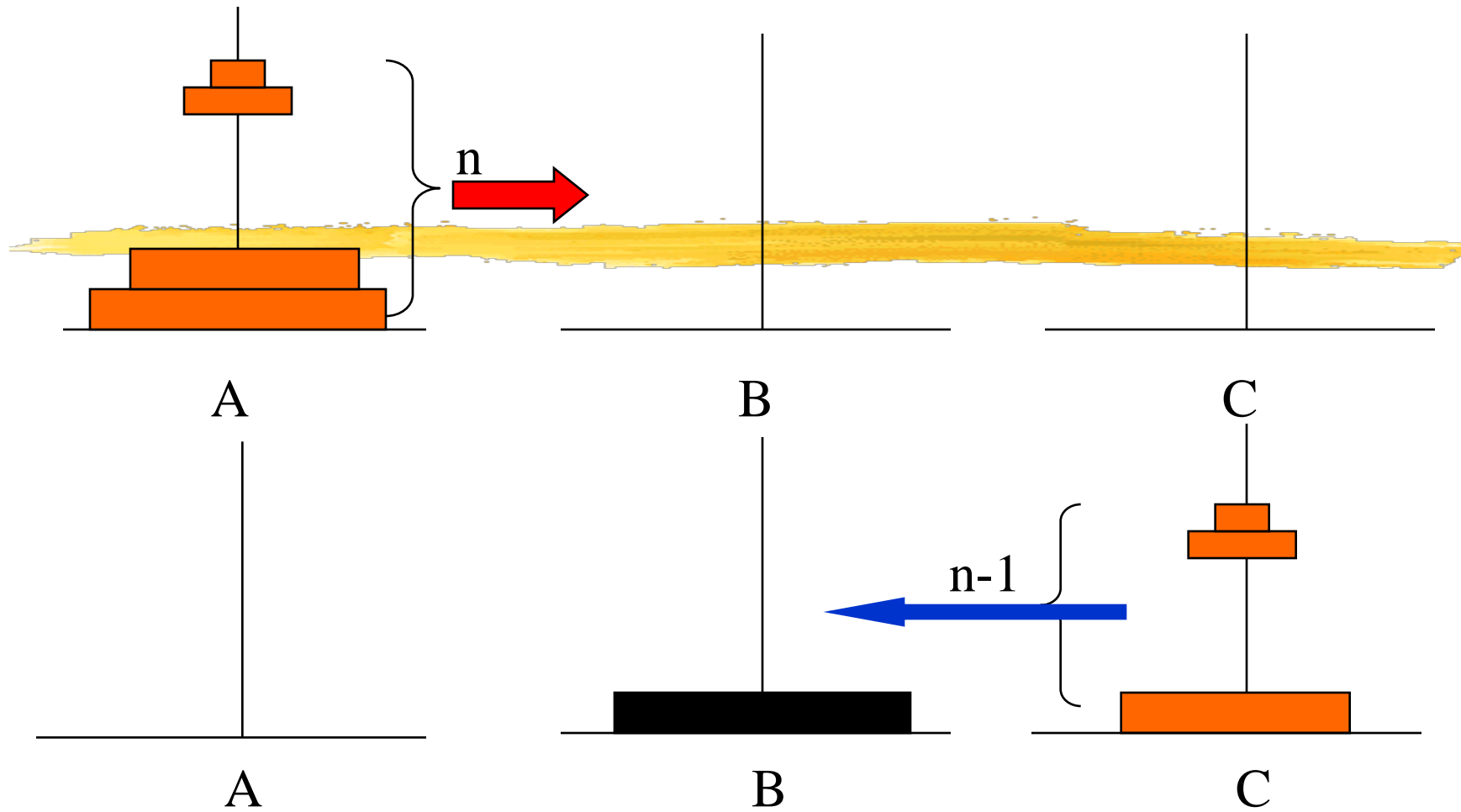
hanio($n-1$ 个盘, $C \rightarrow B$, A 为过渡)

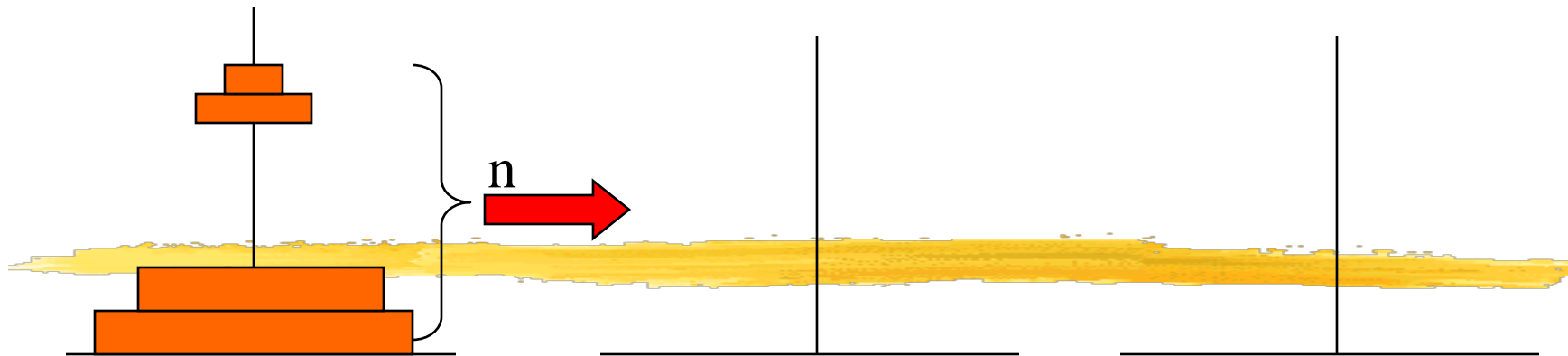
}

}

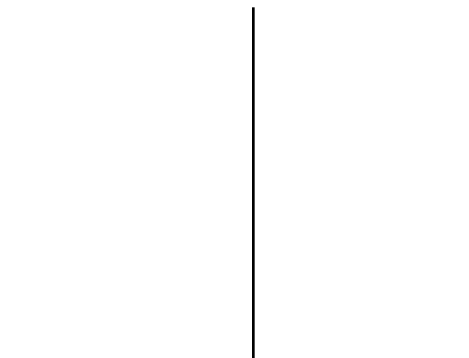






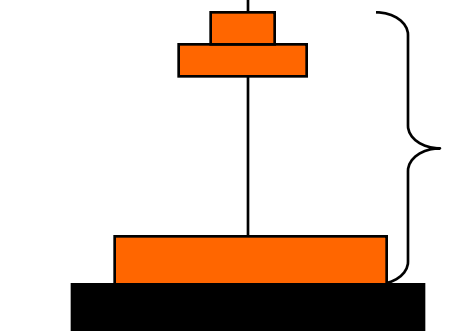


A



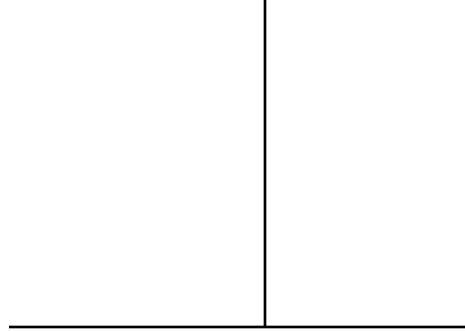
A

B



B

C



C

汉诺(Hanoi)塔 问题求解

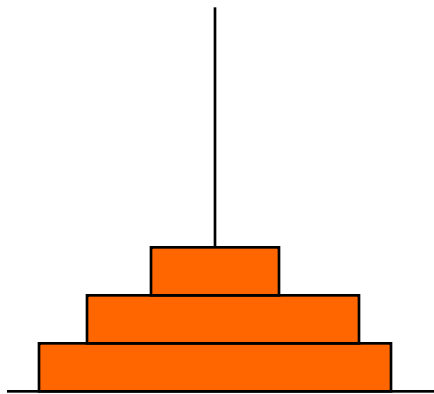
伪代码描述

```
hanio(n个盘, A→B, C为过渡)
{
    if (n == 1)
        直接搬运盘子: A→B
    else
    {
        hanio(n-1个盘, A→C,
            B为过渡)
        搬运第n个盘子: A→B
        hanio(n-1个盘, C→B,
            A为过渡)
    }
}
```

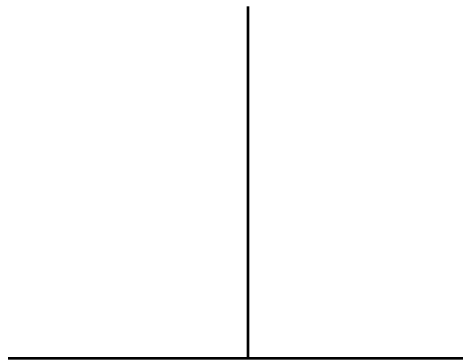
C语言代码

```
void
hanio(int n, char A, char B, char C)
{
    if (n == 1)
        printf("%c-->%c\n", A, B);
    else
    {
        hanio(n-1, A, C, B);
        printf("%c-->%c\n", A, B);
        hanio(n-1, C, B, A);
    }
}
```

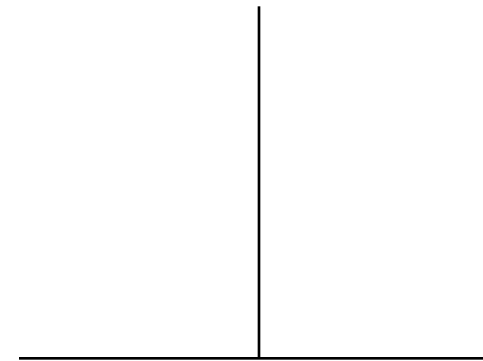
input the number of
disk: 3
the steps for 3 disk are:
a-->b
a-->c
b-->c
a-->b
c-->a
c-->b
a-->b



A



B



C

课堂练习：用递归函数实现逆序输出以回车结束输入的一串字符

```
#include <stdio.h>

void pri()
{
    char ch;
    if ((ch=getchar())!='\n') pri();
    putchar(ch);
}

void main()
{ pri(); }
```