第4讲-程序与递归--二看计算机的本质

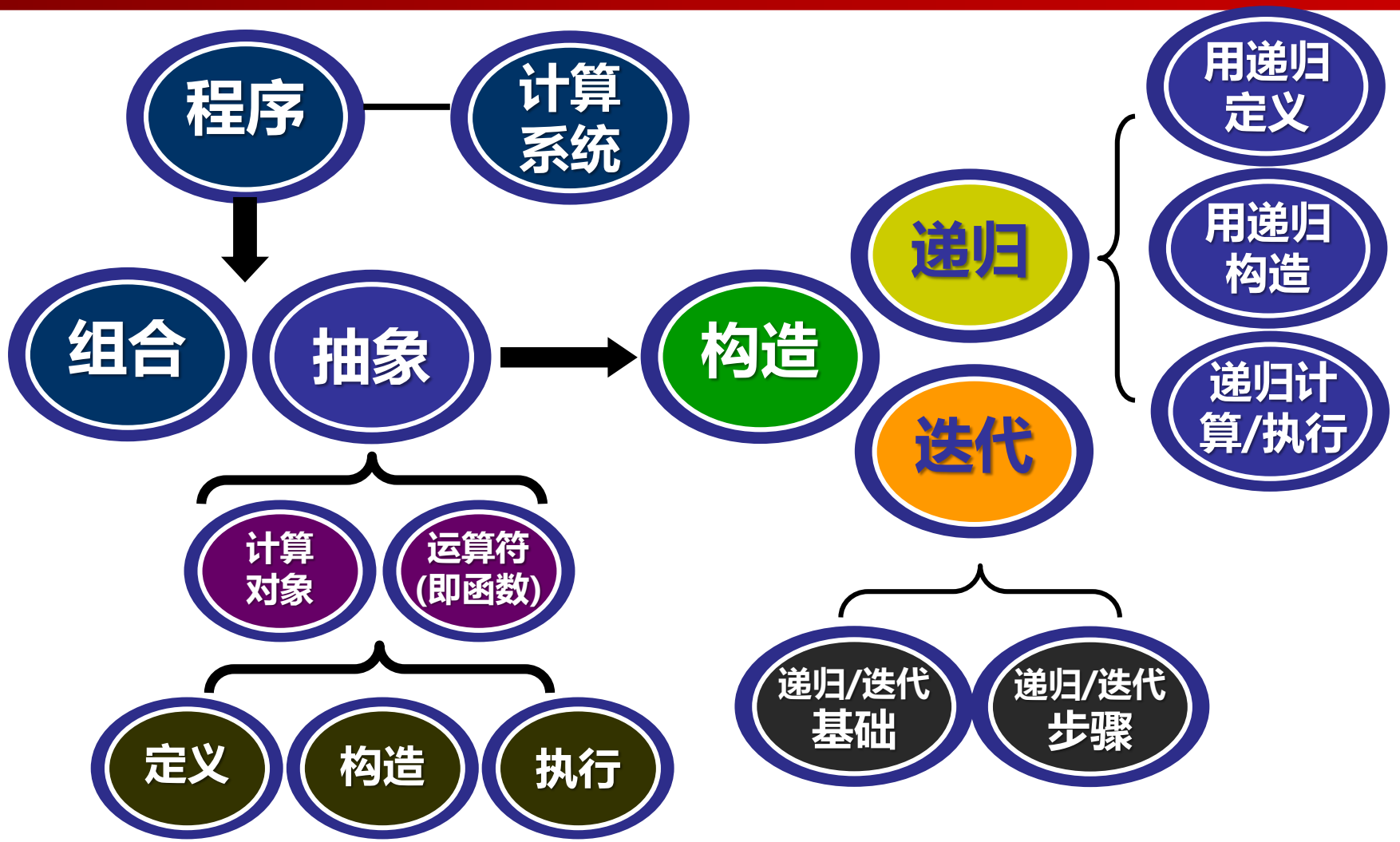
黄宏

华中科技大学计算机学院

honghuang@hust.edu.cn

第4讲-程序与递归--二看计算机的本质

本讲要学习的概念和思维



第4讲-程序与递归--二看计算机的本质

一、计算系统与程序

二、程序构造：组合与抽象示例（运算组合式）

三、递归的概念

四、两种不同的递归函数--递归与迭代

五、运用递归和迭代：构造与自动执行

计算系统与程序

（1）怎样构造一台计算机器?

已知：（1）“加减乘除运算都可转换为加减法运算来实现”

（2）“加减法运算又可以转换为逻辑运算来实现”

首先，设计并实现系统可以执行的基本动作(可实现的)，例如：

“与”动作、 **“或”**动作、 **“非”**动作、 **“异或”**动作

那么，复杂的动作呢？

计算系统与程序

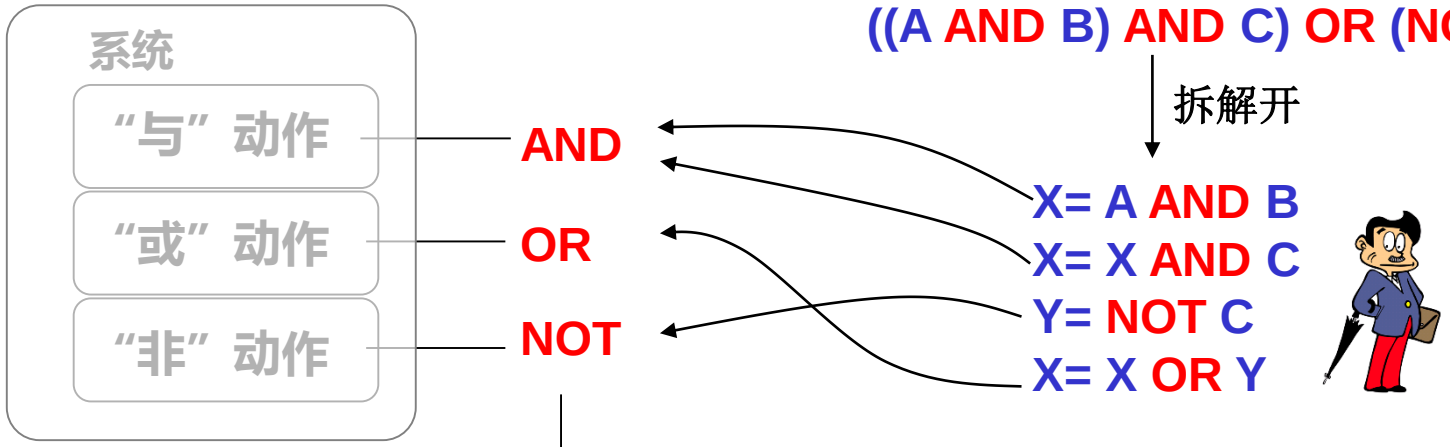
（2）构造计算机器的基本思维：程序

程序：由基本动作指令构造的，若干指令的一个组合或一个执行序列，用以实现复杂动作

复杂动作

((A AND B) AND C) OR (NOT C))

拆解开



指令：控制基本动作执行的命令

计算系统与程序

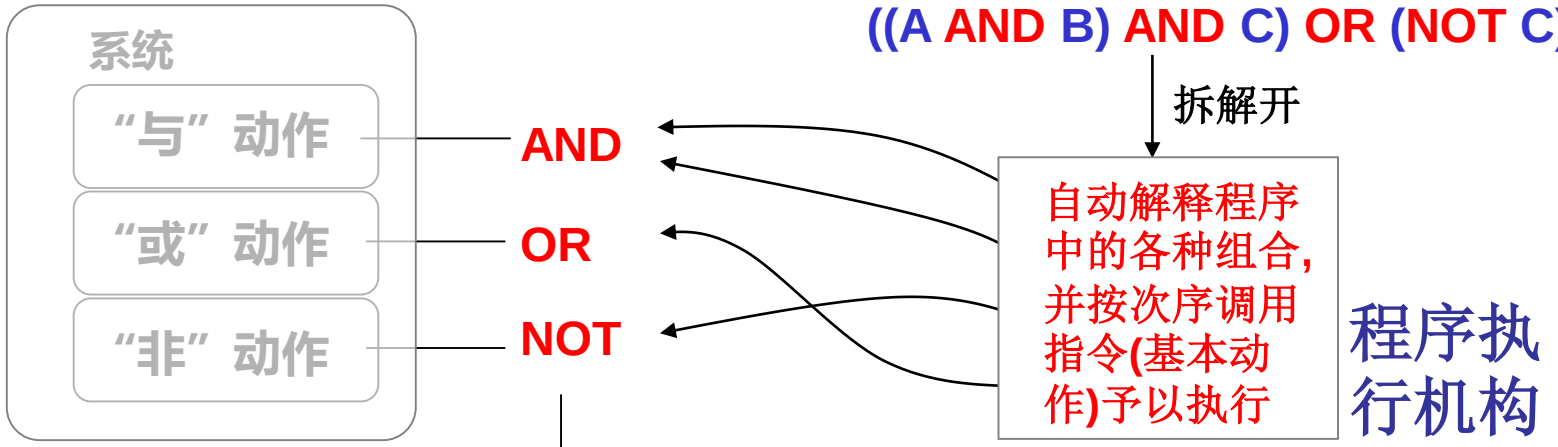
（3）自动计算机器：程序 与 程序执行机构

程序：由基本动作指令构造的，若干指令的一个组合或一个执行序列，用以实现复杂动作

复杂动作

((A AND B) AND C) OR (NOT C))

拆解开



指令：控制基本动作执行的命令

计算系统与程序

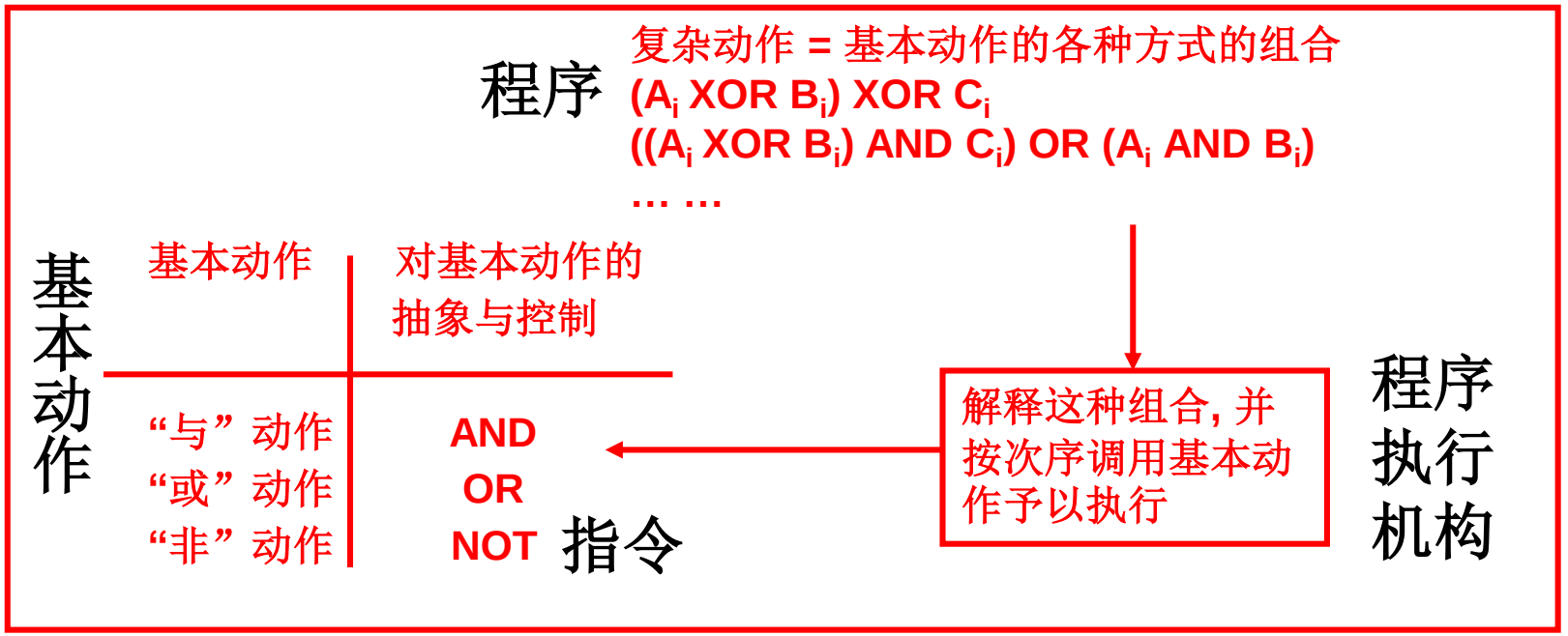
（3）自动计算机器：程序 与 程序执行机构

计算系统 = 基本动作 + 指令 + 程序执行机构

指令 = 对可执行基本动作的抽象，即控制基本动作执行的命令

程序 = 基本动作指令的一个组合或执行序列，用以实现复杂的动作

程序执行机构 = 负责解释程序即解释指令之间组合并按次序调用指令即调用基本动作执行的 机构

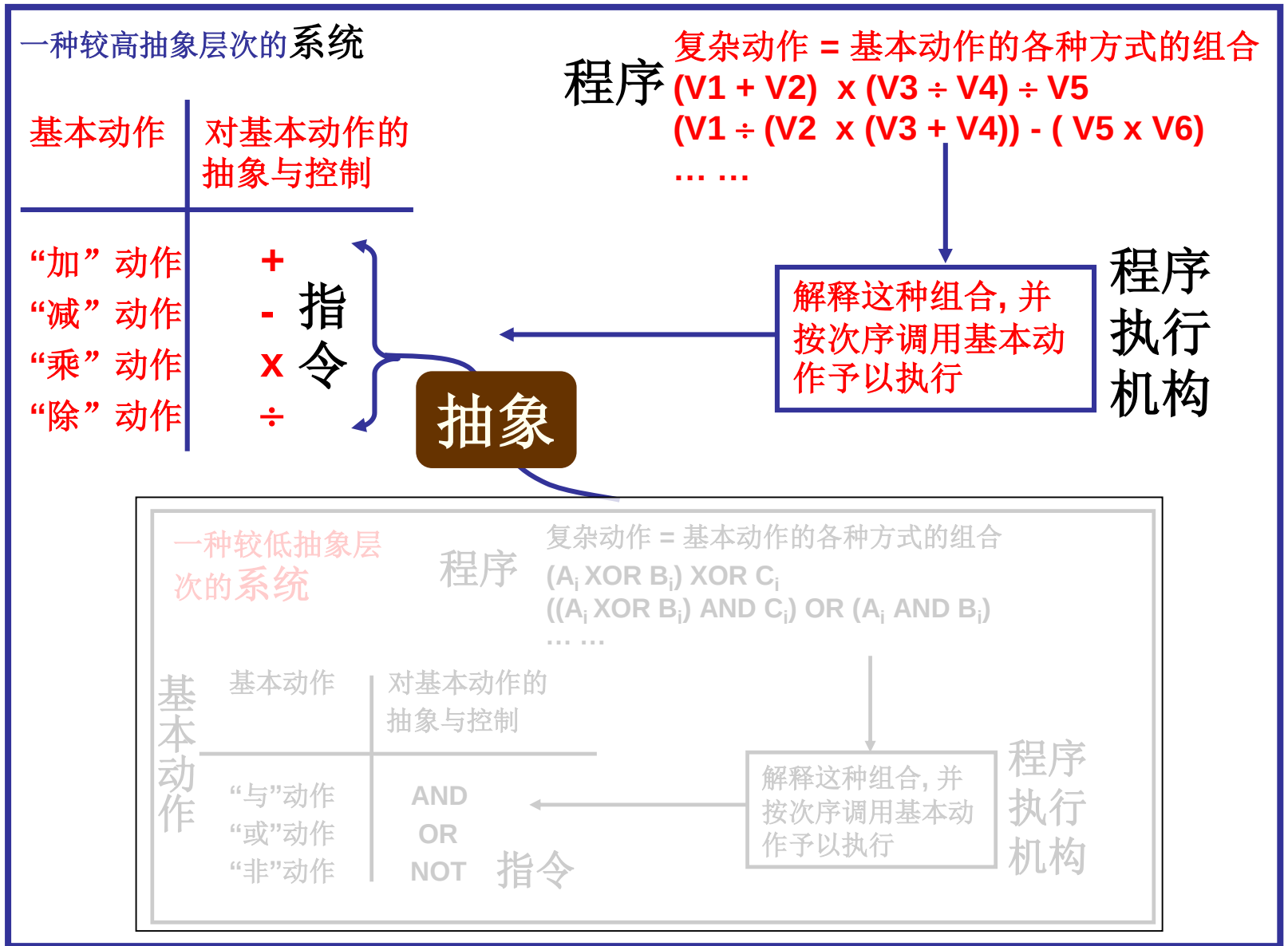


计算系统与程序

（4）分层式构造

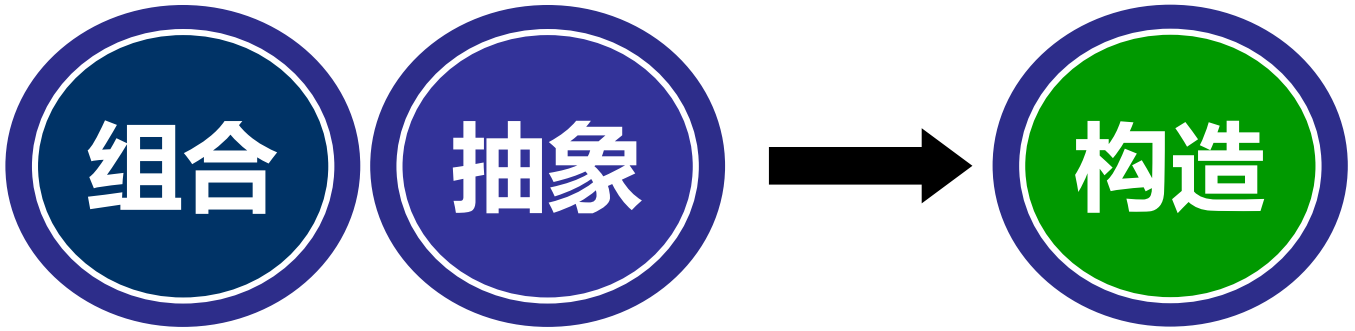
高抽象层次vs. 低抽象层次

抽象：将经常使用的、可由低层次系统实现的一些复杂动作，进行命名，以作为高层次系统的指令被使用



(5) 小结

什么是程序?
程序的本质是什么?



第4讲-程序与递归--二看计算机的本质

一、计算系统与程序

二、程序构造：组合与抽象示例（运算组合式）

三、递归的概念

四、两种不同的递归函数--递归与迭代

五、运用递归和迭代：构造与自动执行

程序构造：组合与抽象示例（运算组合式）

（1）运算组合式——一种程序的表达方法

由数值，到基本运算组合式

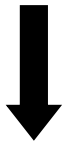
100

205

实际的数值

(100 + 205)

中缀表示法，用运算符(即前述的指令)
将两个数值组合起来，运算符在中间



(+ 100 205)

前缀表示法，用运算符(即前述的指令)
将两个数值组合起来，运算符在前面

将运算符表示的操作应用于后面的一组数值上，求出结果

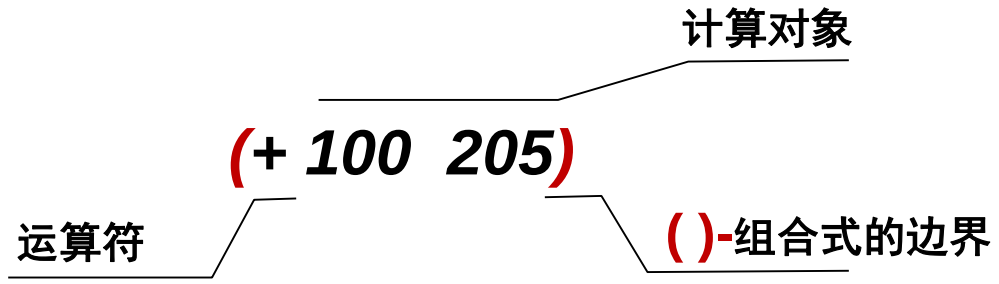
(+ 100 205 300 400 51 304) 一个运算符可以表示连加，连减等情况，

程序构造：组合与抽象示例（运算组合式）

（1）运算组合式——一种程序的表达方法

由数值，到基本运算组合式

(+ 100 205)	*(100 + 205)*
(- 200 50)	*(200 - 50)*
(200 5)*	*(200 * 5)*
(20 5 4 2)*	*(20 * 5 * 4 * 2)*
(- 20 5 4 2)	*(20 - 5 - 4 - 2)*
(+ 20 5 4 2)	*(20 + 5 + 4 + 2)*



- 一个组合式内只能有一个运算符，可有多个计算对象
- 基本运算符只有+、−、*、/，可以被识别和计算。

一起练习,书写程序, … …

程序构造：组合与抽象示例（运算组合式）

（2）运算组合式的“组合-嵌套”（构造）及其计算过程（执行）

(+ 100 205)　　*(100 + 205)*　　(运算符1 计算对象1 计算对象2)

(+ (+ 60 40) (- 305 100))　　*((60 + 40) + (305- 100))*　　(运算符2 计算对象3 计算对象4)

((* 3 (+ (* 2 4) (+ 3 5))) (+ (- 10 7) 6))*

计算过程

(运算符1 (运算符2 计算对象3 计算对象4) 计算对象2)

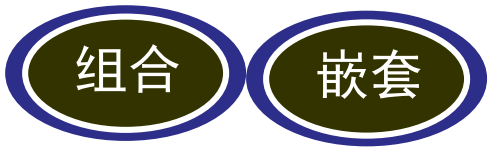
((* 3 (+ (* 2 4) (+ 3 5))) (+ (- 10 7) 6))*

((* 3 (+ 8 8)) (+ 3 6))*

((* 3 16) 9)*

(48 9)*

432



程序构造：组合与抽象示例（运算组合式）

（2）运算组合式的“组合-嵌套”（构造）及其计算过程（执行）

((* 3 (+ (* 2 4) (+ 3 5))) (+ (- 10 7) 6))*

转化为中缀表示法

((3 ((2 * 4) + (3 + 5))) * ((10 - 7) + 6))*

*((3 * ((2 * 4)+ (3 + 5))) * ((10 - 7) + 6))*

*((3 * ((2 * 4)+ (3 + 5))) * ((10 - 7) + 6))*

*((3 * ((2 * 4)+ (3 + 5))) * ((10 - 7) + 6))*

((3(8+8))*(3+6))*

*(48 * 9) = 432*

程序构造：组合与抽象示例（运算组合式）

（3）用名字替代计算对象（构造）

命名计算对象和构造中使用名字及计算中以计算对象替换名字

(define height 2) ——— 名字的定义：定义名字height与2关联，以后可以用height来表示2
一种类型的名字：数值型的名字

(+ (+ height 40) (- 305 height))

((height + 40) + (305 - height)) ——— 名字的使用

(+ (50 height) (- 100 height))*

*((50 * height) + (100 - height))*

注意：不同类型的对象可以有不同的定义方法。这里统一用define来表示，在具体的程序设计语言中是用不同的方法来定义的

程序构造：组合与抽象示例（运算组合式）

（3）用名字替代计算对象（构造）

```
(define pi 3.14159)
(define radius 10)
(*  pi  (* radius radius))
(pi * (radius * radius))
```

```
(*  pi  (*  radius radius)) (pi *  (radius * radius))
(*  pi  (* 10  10)) (pi *  (10 * 10))
(*  pi  100) (pi  * 100)
(* 3.14159 100) (3.14159 * 100)
314.159
```

计算

```
(define  circumference  (* 2 pi radius))
                        (2* pi* radius))

(*  circumference  20)
```

```
(*  circumference  20) (circumference * 20)
(* (* 2  pi  radius) 20) ( (2* pi* radius) * 20)
(* (* 2  3.14159  10) 20) ( (2* 3.14159* 10) * 20)
(* 62.8318  20) (62.8318 *  20)
1256.636
```

计算

程序构造：组合与抽象示例（运算组合式）

（4）定义新运算符与使用新运算符（构造）

命名新运算符和构造中使用新运算符及执行中以过程替换新运算符

(define (square x) (x x))*

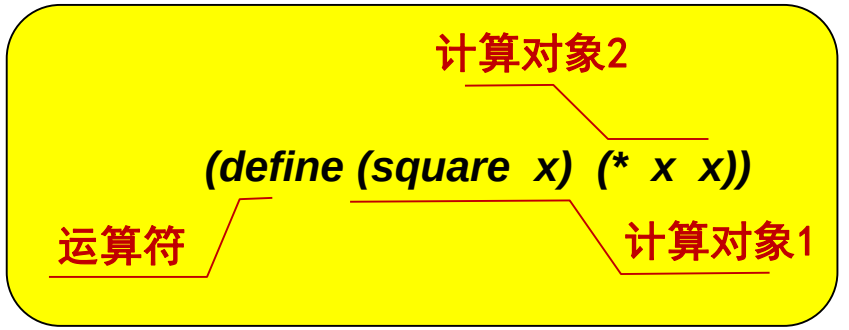
x^2

名字的定义：定义名字**square**为一个新的运算，即**过程**或称**函数**。另一种类型的名字：运算符型的名字

新运算符，*即过程名或函数名*

形式参数，使用时将被实际参数所替代

过程体，用于表示新运算符的具体计算规则，其为关于形式参数x的一种计算组合。



(square 3)
(square 6)

名字的使用

注意：不同类型的对象可以有不同的定义方法。这里统一用define来表示，在具体的程序设计语言中是用不同的方法来定义的

程序构造：组合与抽象示例（运算组合式）

（5）新运算符定义与构造示例

(square 10)

(square (+ 2 8))

(square (square 3))

(square (square (+ 2 5))) *(square (square (2 + 5)))*

(define (SumOfSquare x y) (+ (square x) (square y)))

x^2+y^2

(SumOfSquare 3 4)

$=3^2+4^2$

(+ (SumOfSquare 3 4) height)

$=(3^2+4^2)+height$

程序构造：组合与抽象示例（运算组合式）

（5）新运算符定义与构造示例

(define (NewProc a) (SumOfSquare (+ a 1) (a 2)))*

$(a+1)^2+(a*2)^2$

(NewProc 3) $=(3+1)^2+(3*2)^2$

(NewProc (+ 3 1)) $=((3+1)+1)^2+((3+1)*2)^2$

程序构造：组合与抽象示例（运算组合式）

（6）运算组合式的执行（计算方法1）

求值、代入、计算

```
(NewProc (+  3  1))
(NewProc 4)
(SumOfSquare (+  4  1) (*  4  2))
(SumOfSquare 5  8)
(+  (Square  5)  (Square  8))
(+  (*  5  5)  (*  8  8))
(+  25  64)
89
```

先求值，再代入、计算

```
(NewProc (3 + 1))
(NewProc 4)
(SumOfSquare (4 + 1) (4 * 2))
(SumOfSquare 5  8)
( (Square  5) +  (Square  8))
((5 * 5) + (8 * 8))
(25 + 64)
89
```

程序构造：组合与抽象示例（运算组合式）

（7）运算组合式的执行（计算方法2）

代入、求值、计算

(NewProc (+ 3 1))
(SumOfSquare(+ (+ 3 1) 1) (* (+ 3 1) 2))

(+ (Square (+ (+ 3 1) 1)) (Square (* (+ 3 1) 2)))

(+ (* (+ (+ 3 1) 1) (+ (+ 3 1) 1)) (* (* (+ 3 1) 2) (* (+ 3 1) 2)))

(+ (* (+ 4 1) (+ 4 1)) (* (* 4 2) (* 4 2))) ——代入阶段

(+ (* 5 5) (* 8 8)) ——求值阶段

(+ 25 64)

89

代入到只有基本运算符时再开始求值

先代入，后求值、计算

(NewProc (3 + 1))
(SumOfSquare((3 + 1) + 1) ((3 + 1) * 2))

((Square ((3 + 1) + 1)) + (Square ((3 + 1) * 2)))

((((3 + 1) + 1) * ((3 + 1) + 1)) + (((3 + 1) * 2) * ((3 + 1) * 2)))

(((4 + 1) * (4 + 1)) + ((4 * 2) * (4 * 2)))

((5 * 5) + (8 * 8))

(25 + 64)

89

程序构造：组合与抽象示例（运算组合式）

（8）构造与执行练习

对于计算式，其正确的运算组合式(前缀表示法)为_____。

(A) (/ (+ 10 / 20 + 8 4) (+ * 3 6 * 8 2));

(B) ((10 + (20 / (8 + 4))) / ((3 * 6) + (8 * 2)));

(C) (/ (+ 10 (/ 20 (+ 8 4))) (+ (* 3 6) (* 8 2)));

(D) (/ (/ 20 (+ 10 (+ 8 4))) (* (+ 3 6) (+ 8 2)))。

$$\frac{10+\frac{20}{8+4}}{3*6+8*2}$$

(/ 操作数1 操作数2)

(/ (+ 10 操作数a2) (+ 操作数a3 操作数a4))

(/ (+ 10 (/ 20 (+ 8 4))) (+ 操作数a3 操作数a4))

(/ (+ 10 (/ 20 (+ 8 4))) (+ (* 3 6) (* 8 2)))

程序构造：组合与抽象示例（运算组合式）

（9）构造与执行练习

◆问题1：用前缀表示法书写下述表达式

$$\frac{10 + 4 + (8- (12 - (6 + 4 \div 5)))}{3*(6-2)(12-7)}$$

◆问题2：请定义一个过程，求某一数值的立方 a^3

◆问题3：进一步以问题2定义的过程，再定义一个过程，求某两个数值的立方和 a^3+b^3 。 进一步求 5^3+8^3 ，并模拟给出计算过程。

程序构造：组合与抽象示例（运算组合式）

(10)带有条件的运算组合式如何表达?

带有条件的运算组合式

$$|x| = \begin{cases} x & if\ x > 0 \\ 0 & if\ x = 0 \\ -x & if\ x < 0 \end{cases}$$

```
(cond  ( <p1> <e1>)
       ( <p2> <e2>)
       ...
       ( <pn> <en>) )
```

```
(define  (abs  x) (cond  ((>  x  0)  x)
                         ((=  x  0)  0)
                         ((<  x  0)  (-  x))))
```

```
(define  (abs  x) (cond  ((x > 0)  x)
                         ((x = 0)  0)
                         ((x < 0)  (-  x))))
```

程序构造：组合与抽象示例（运算组合式）

(11)带有条件的运算组合式如何表达?

◆问题4：请定义一个过程，计算下列函数

$$f(x)=\begin{cases} x^2-x & if\ \ x>0 \\ 0 & if\ \ x=0 \\ -x^2+x & if\ \ x<0 \end{cases}$$

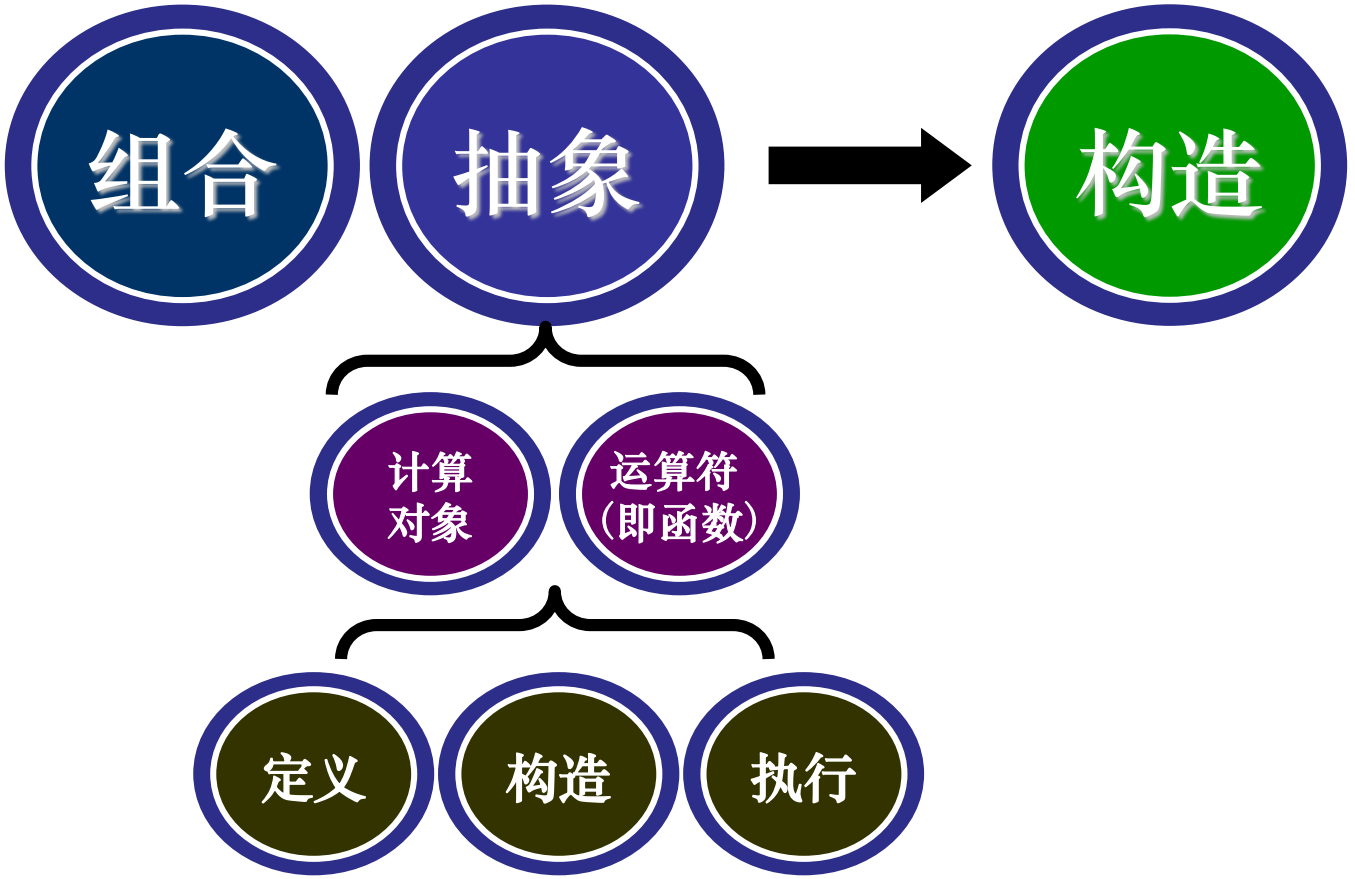
```
(cond  ( <p1> <e1>)
       ( <p2> <e2>)
       ...
       ( <pn> <en>) )
```

```
(define  (f  x) (cond  ((>  x  0)  (-  (Square x)  x))
                       ((=  x  0)  0)
                       ((<  x  0)  (-  x  (Square x)) ))  ))
```

```
(define  (f  x) (cond  ((x > 0)  ( (Square x) -  x))
                       ((x =  0)  0)
                       ((x < 0)  (x - (Square x)) )) ))
```

程序构造：组合与抽象示例（运算组合式）

(12)小结



第4讲-程序与递归--二看计算机的本质

一、计算系统与程序

二、程序构造：组合与抽象示例（运算组合式）

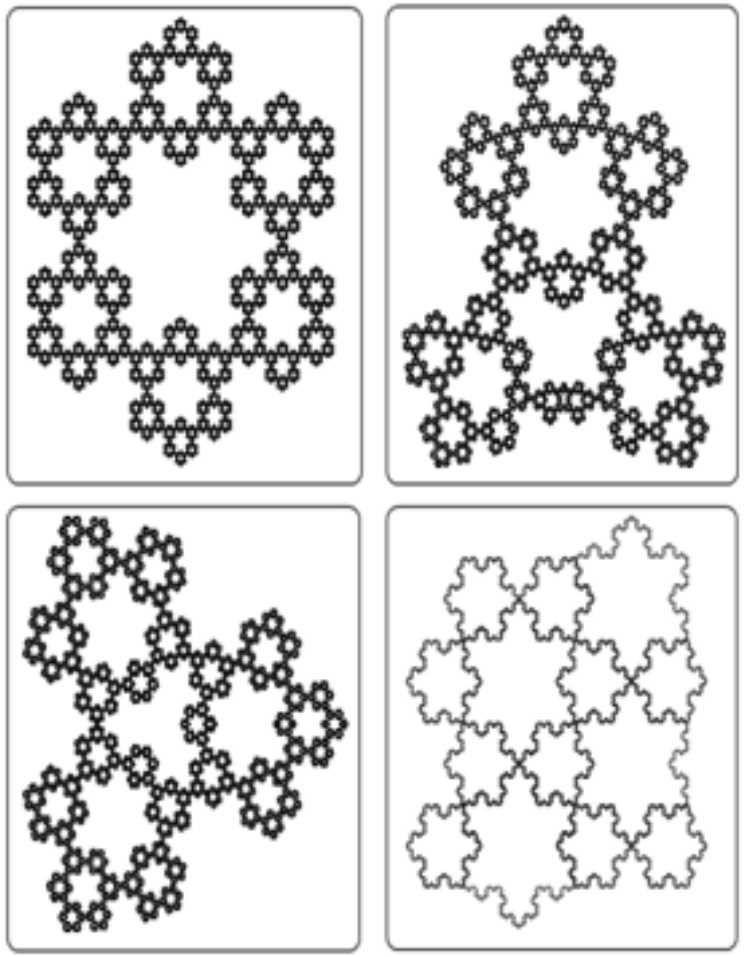
三、递归的概念

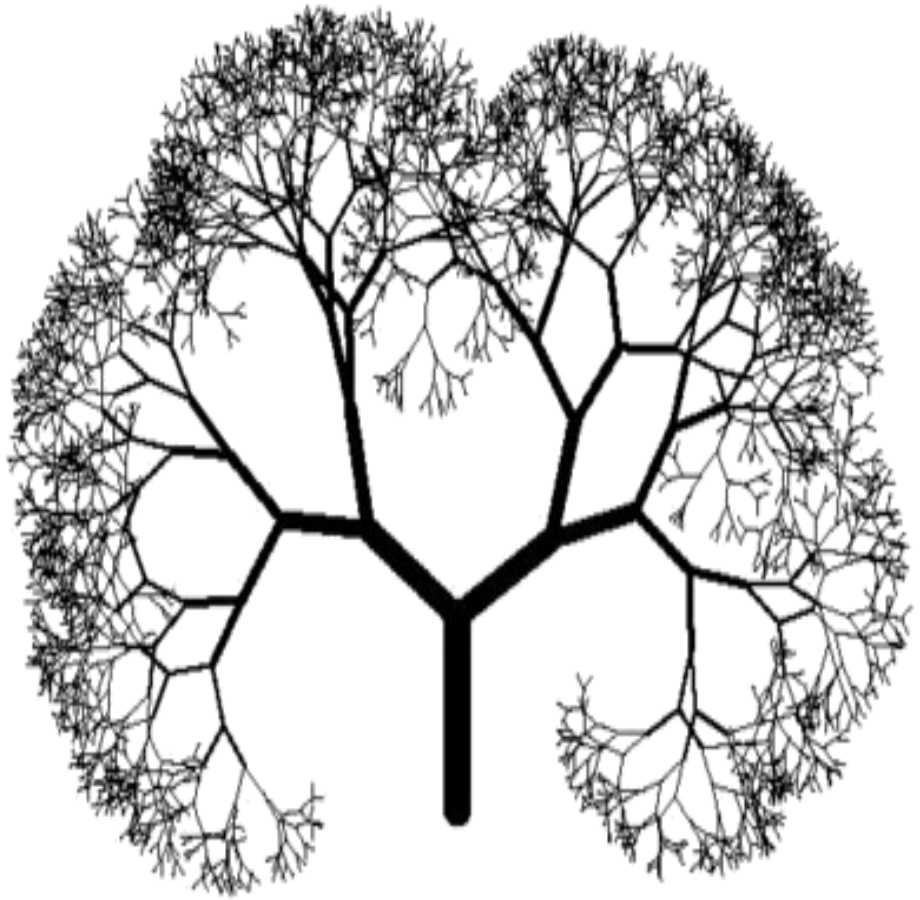
四、两种不同的递归函数--递归与迭代

五、运用递归和迭代：构造与自动执行

递归的概念

递归的典型视觉形式---自相似性事物的无限重复性构造





递归的概念

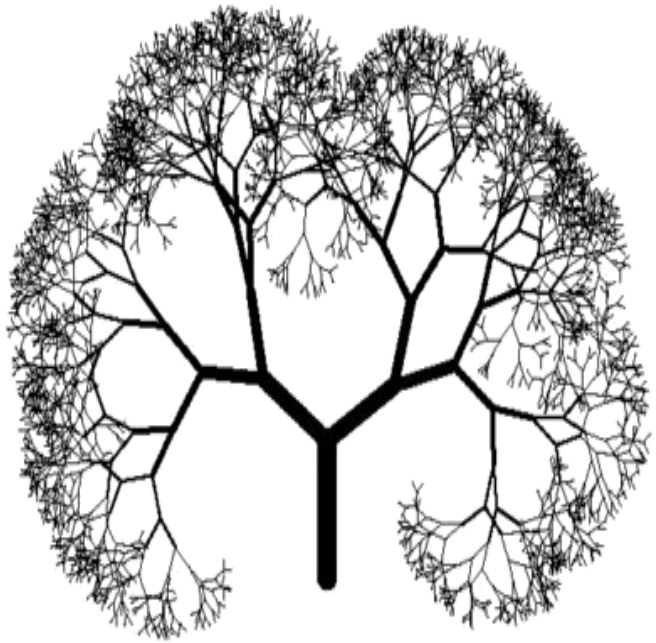
（3）递归的典型视觉形式---自相似性事物的无限重复性构造

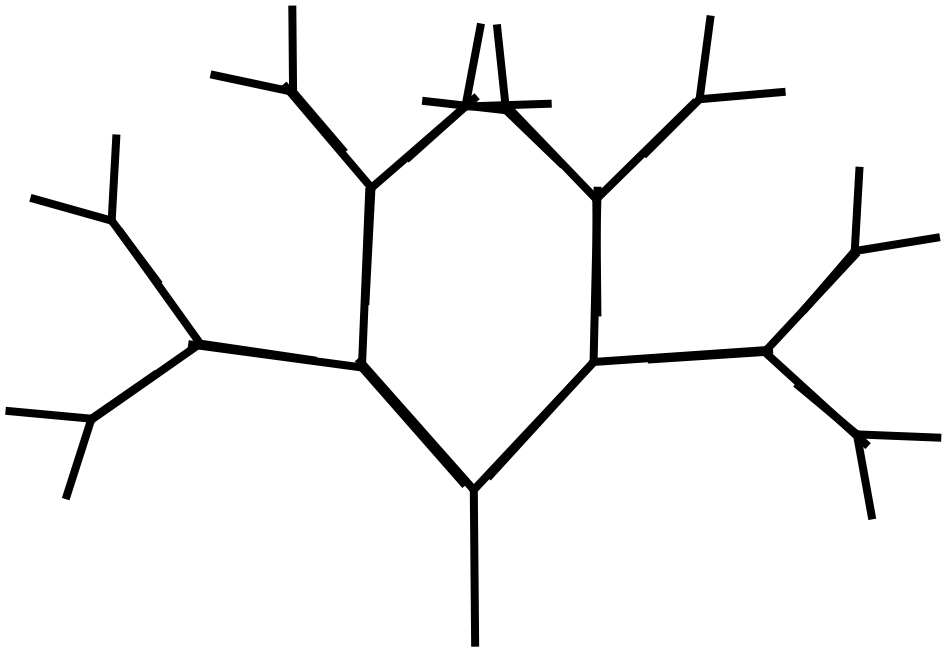
递归的概念

什么是递归?

递归是表达相似性对象及动作的无限性重复性构造的方法。

- **递归基础**：定义、构造和计算的起点，直接给出。即h(0)。
- **递归步骤**：由**第n项**或前n项**定义第n+1项**。即: **h(n+1) = g(h(n),n)**，g是需要明确给出的，以说明h(n+1)怎样由h(n)和n构造出来。



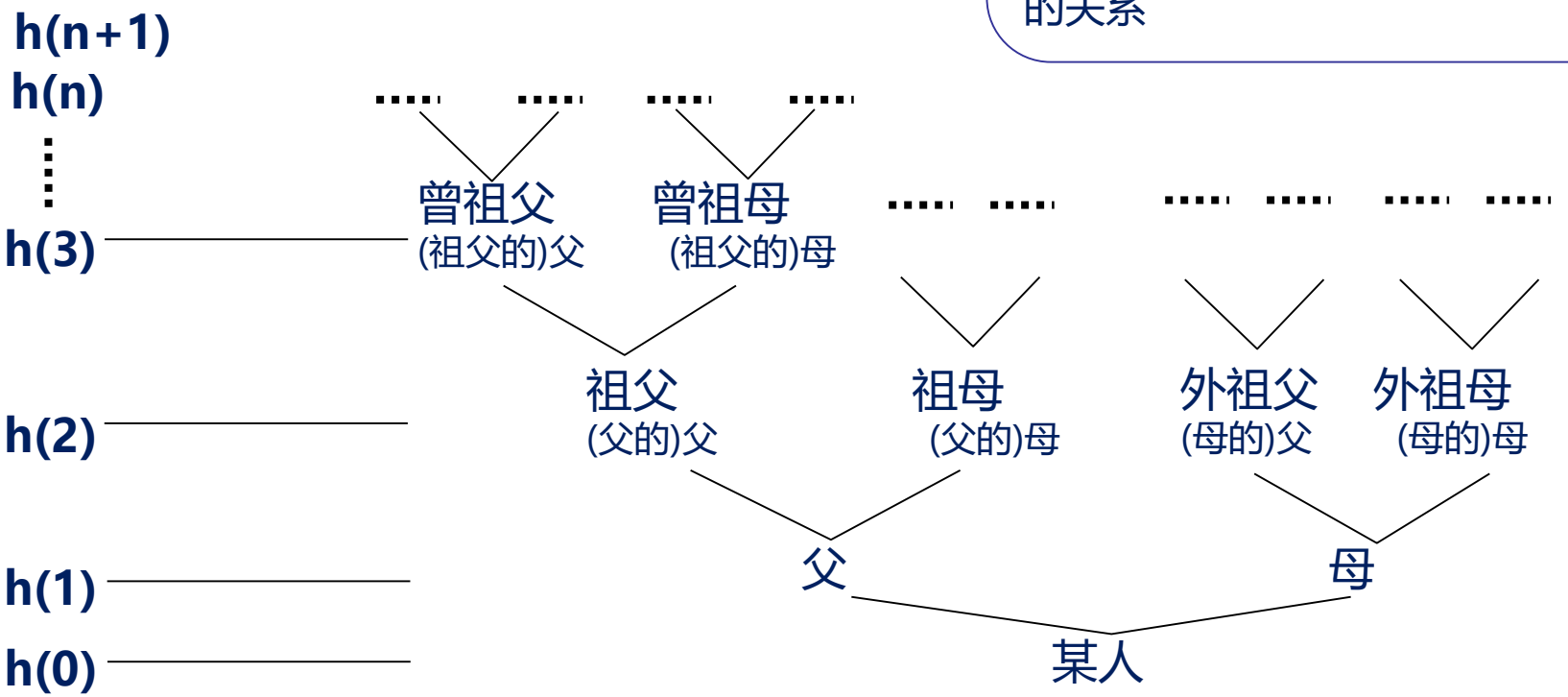


用递归进行定义

怎样递归?

【示例】怎样定义你的祖先?

(1)某人的双亲(父母)是他的祖先（递归基础）
--- h(1)直接给出
(2)某人祖先的双亲(父母)同样是某人的祖先（递归步骤/公式）
--- h(n+1) = g(h(n) , n)
--- n: 第几代；h(n)是第几代祖先；g是h(n+1)与h(n)和n的关系



递归的理解比较难

递归思维是计算机最重要的特征

想学计算机就要理解递归思维

体验两种不同的递归函数

一种类型的递归函数：定义是递归的，但执行可以是递归的也可以是迭代的

- Fibonacci数列，无穷数列1，1，2，3，5，8，13，21，34，55，……，

$$F(n)=\begin{cases}1 & n=0\\ 1 & n=1\\ F(n-1)+F(n-2) & n>1\end{cases}$$

递归定义

计算规则的表达—告诉执行者要递归计算

F(0)=1;

F(1)=1;

F(2)=F(1)+F(0)=2;

F(3)=F(2)+F(1)= 3;

F(4)=F(3)+F(2)= 3+2=5;… …

递推计算/迭代计算/迭代执行

执行者要一步一步地完成计算获得结果

体验两种不同的递归函数

另一种类型的递归函数：定义是递归的，但执行也是递归的（用迭代不容易完成）

- **阿克曼递归函数**

$$A(m,n)=\begin{cases} n+1 & 若m=0 \\ A((m-1),1) & 若n=0 \\ A(m-1,A(m,n-1)) & 若m,n>0 \end{cases}$$

递归定义

函数本身是递归的，
函数的变量也是递归的

计算规则的表达—告诉执行者要递归计算

A(1,n) =A(0, A(1,n-1))=A(0. <...代入前式计算过程>)=A(0,n+1)=n+2。

A(2,1) = A(1， A(2， 0)) --A(2,1)按m,n>0公式代入
= A(1， A(1， 1)) --A(2,0)按n=0公式代入
=A(1， A(0， A(1， 0))) --A(1,1)按m,n>0公式代入
= A(1， A(0， A(0， 1))) --A(1,0)按n=0公式代入
= A(1， A(0， 2)) --A(0,1)按m=0公式代入
= A(1， 3) --A(0,2)按m=0公式代入
=A(0， A(1， 2)) --A(1,3)按m,n>0公式代入
= A(0， A(0， A(1， 1))) --A(1,2)按m,n>0公式代入
= A(0， A(0， A(0， A(1， 0)))) --A(1,1)按m,n>0公式代入
= A(0， A(0， A(0， A(0,1))) --A(1,0)按n=0公式代入
= A(0， A(0， A(0， 2))) --A(0,1)按m=0公式代入
= A(0， A(0， 3)) = A(0， 4) = 5。 --依次按m=0公式代入

执行者要一步一步地完成计算获得结果

递归计算/递归执行

由后向前代入，
再由前向后计算

递归与迭代--两种不同的递归函数

（2）递归与迭代的差异

◆**迭代(递推)**：可以自递归基础开始，由前向后依次计算或直接计算；

◆**递归**：可以自递归基础开始，由前向后依次计算或直接计算；但有些，只能由后向前代入，直到递归基础，寻找一条路径，然后再由前向后计算。

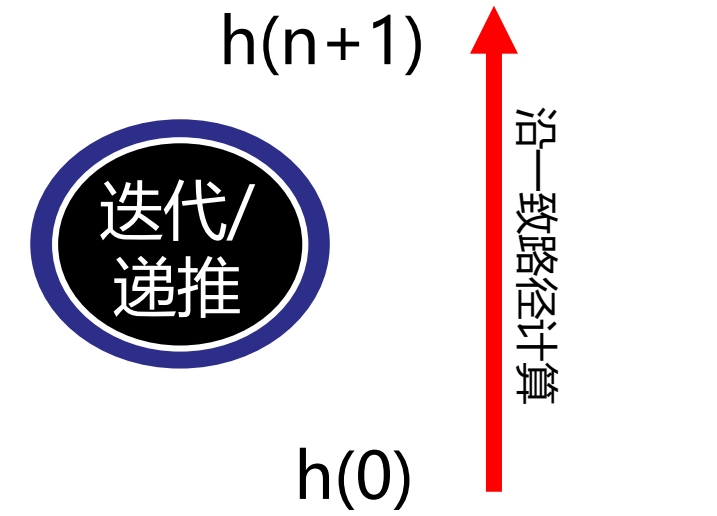
◆**递归包含了递推(迭代)，但递推(迭代)不能覆盖递归。**

两种不同的递归函数：递归与迭代

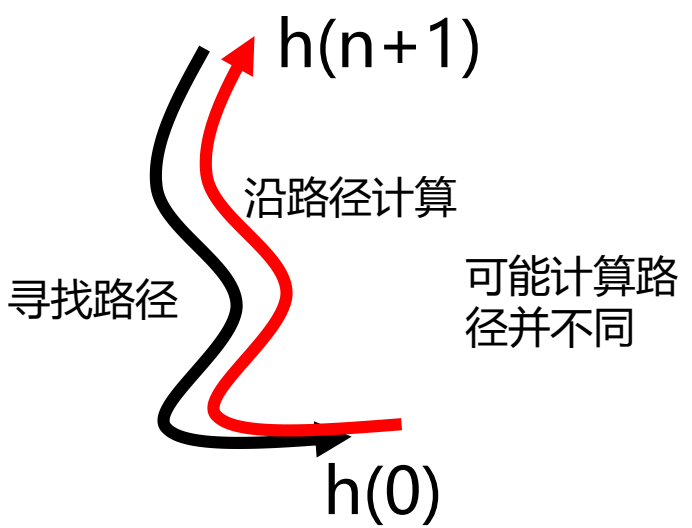
递归与迭代的差异

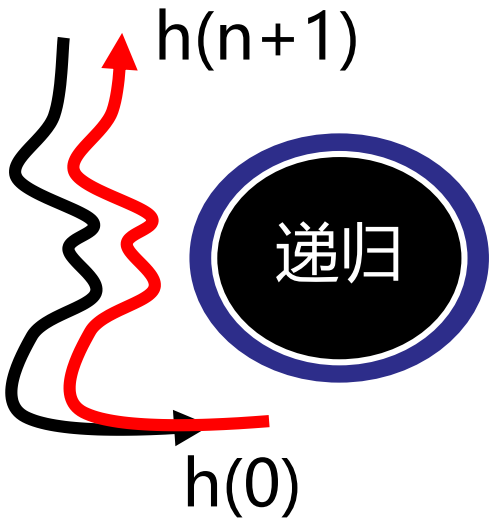
$$h(n)=\begin{cases}1 & n=0\\ 1 & n=1\\ h(n-1)+h(n-2) & n>1\end{cases}$$

$$h(m,n)=\begin{cases}n+1 & 若m=0\\ h((m-1),1) & 若n=0\\ h(m-1,h(m,n-1)) & 若m,n>0\end{cases}$$



在前次结果基础上进行计算
可采用**循环**结构实现





通常，只能由**函数**结构实现

递归程序的构造与执行

运用递归进行程序构造：函数结构，自身调用自身，高阶调用低阶

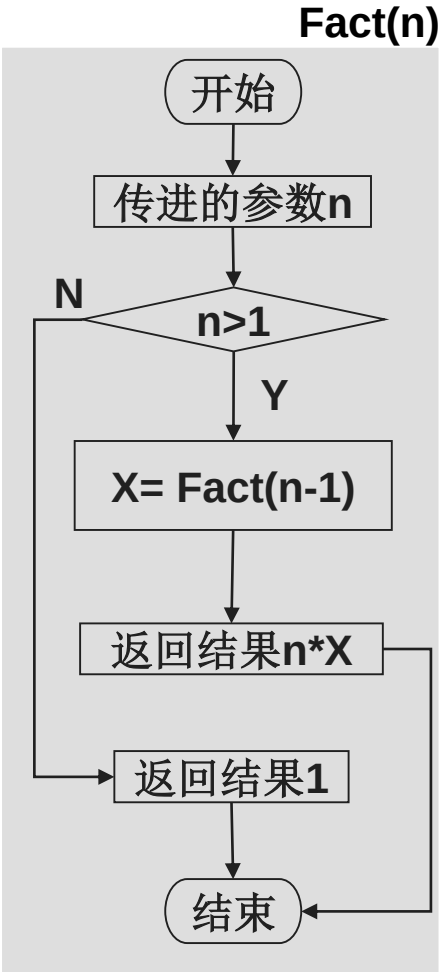
【示例】求n!的算法或程序--用递归方法构造：函数调用

$$n\,! = \begin{cases} 1 & \text{当} n \le 1 \text{ 时} \\ n \times (n-1)\,! & \text{当} n > 1 \text{ 时} \end{cases}$$

```
long int Fact(int n)
{    long int x;
     If (n > 1)
     { x = Fact(n-1);
  /*递归调用*/
        return n*x;   }
     else return 1;
  /*递归基础*/
}
```

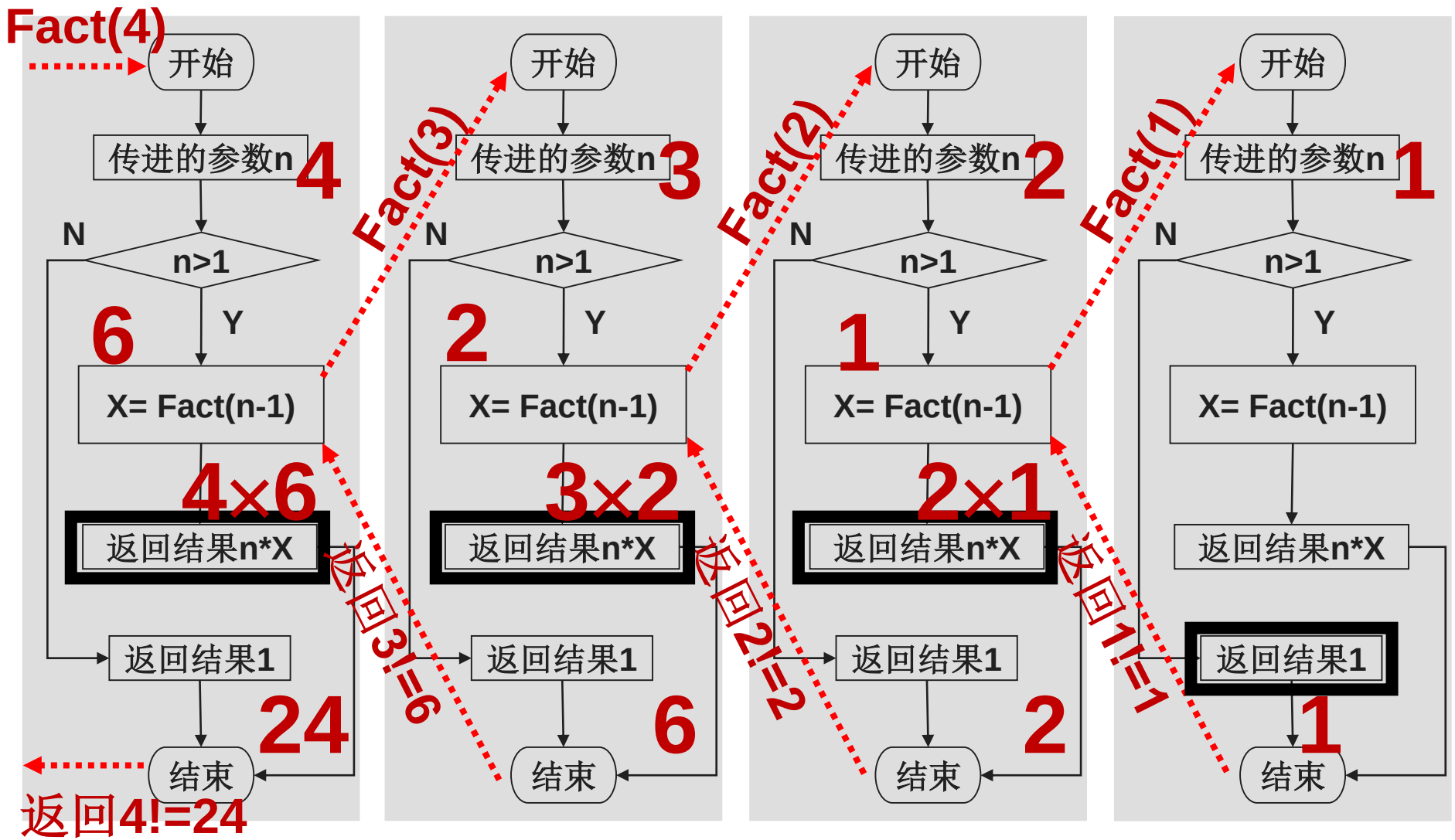
Python

```
def Fact(n):
    if (n > 1):
        x = Fact(n-1)
        return n*x
    else:
        return 1
```



递归程序的构造与执行

递归程序的执行过程：利用【堆栈】保存待计算的数据



1	Fact(4)的程序断点
2	Fact(3)的程序断点
3	Fact(4)的程序断点
4	
5	
6	

堆栈：后进先出

迭代程序的构造与执行

迭代程序的执行过程?

【示例】求n!的算法或程序--用迭代方法构造：循环-替代-递推

$$n!=\begin{cases}1 & \text{当} n\le 1\text{时}\\ 1\times 2\times\ldots(n-1)\times n & \text{当} n>1\text{时}\end{cases}$$

```
long int Fact(int n)
{   int  counter；
    long product=1;
    for  counter=1 to n step 1
    { product = product * counter；  }
  /*迭代*/
return product;
}
```

Fact(5)的执行过程

	Counter	Product
初始值		1
循环第1次	1	1
循环第2次	2	2
循环第3次	3	6
循环第4次	4	24
循环第5次	5	120
退出循环	6	120

递归与迭代

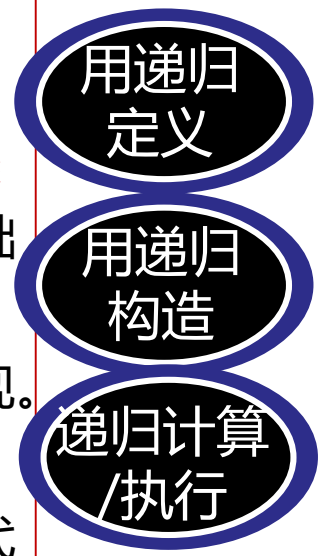
小结

递归是表达相似性对象及动作的无限性重复性构造的方法。

■递归基础：定义、构造和计算的起点，直接给出。即h(0)。

■递归步骤：由第n项或前n项定义第n+1项的一个或一组公式。

- 递归是一种关于抽象的表达方法---用递归定义无限的相似事物
- 递归是一种算法或程序的构造技术---自身调用自身，高阶调用低阶，构造无限的计算步骤
- 递归是一种典型的计算/执行过程---由后向前代入，直至代入到递归基础，再由递归基础向后计算直至计算出最终结果，即由前向后计算
- 数学上用递归定义，但算法上有些可由递归算法/程序实现，有些可由迭代算法/程序实现。
- 迭代算法/程序的执行效率要远高于递归算法/程序，因此能用迭代实现的尽量不用递归。
- 能用迭代算法实现的，都能用递归算法实现；而能用递归算法实现的，却未必都能用迭代算法实现。
- 递归算法是用【函数】结构来表达的，而迭代算法是用【循环】结构来表达的。
- 递归算法执行需要【堆栈】（一种后进先出的数据结构）予以支持。



第4讲-程序与递归--二看计算机的本质

一、计算系统与程序

二、程序构造：组合与抽象示例（运算组合式）

三、递归的概念

四、递归与迭代--两种不同的递归函数

五、运用递归和迭代：构造与自动执行

运用递归和迭代：构造与自动执行

（1）递归计算过程的体验很重要

$$A(m,n)=\begin{cases} n+1 & \text{若} m=0 \\ A((m-1),1) & \text{若} n=0 \\ A(m-1,A(m,n-1)) & \text{若} m,n>0 \end{cases}$$

A(1,n) =A(0, A(1,n-1))=A(0. <…代入前式计算过程>)=A(0,n+1)=n+2。

A(2,1)= A(1， A(2， 0)) *--A(2,1)按m,n>0公式代入*

= A(1， A(1， 1)) *--A(2,0)按n=0公式代入*

=A(1， A(0， A(1， 0))) *--A(1,1)按m,n>0公式代入*

= A(1， A(0， A(0， 1))) *--A(1,0)按n=0公式代入*

= A(1， A(0， 2)) *--A(0,1)按m=0公式代入*

= A(1， 3) *--A(0,2)按m=0公式代入*

=A(0， A(1， 2)) *--A(1,3)按m,n>0公式代入*

= A(0， A(0， A(1， 1))) *--A(1,2)按m,n>0公式代入*

= A(0， A(0， A(0， A(1， 0)))) *--A(1,1)按m,n>0公式代入*

= A(0， A(0， A(0， A(0,1))) *--A(1,0)按n=0公式代入*

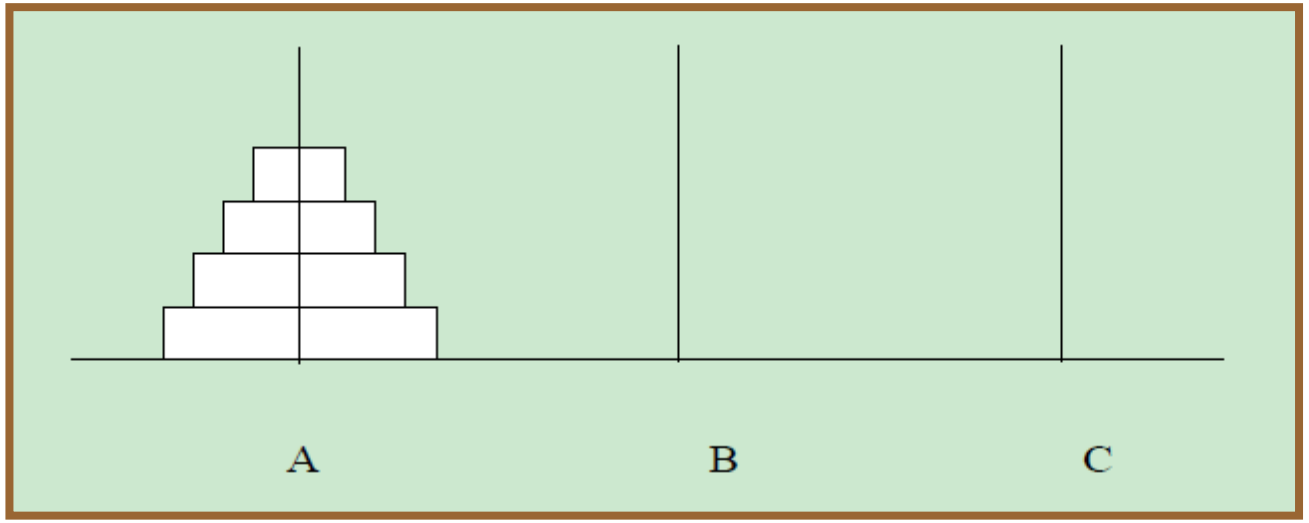
= A(0， A(0， A(0， 2))) *--A(0,1)按m=0公式代入*

= A(0， A(0， 3)) = A(0， 4) = 5。 *--依次按m=0公式代入*

运用递归和迭代：构造与自动执行

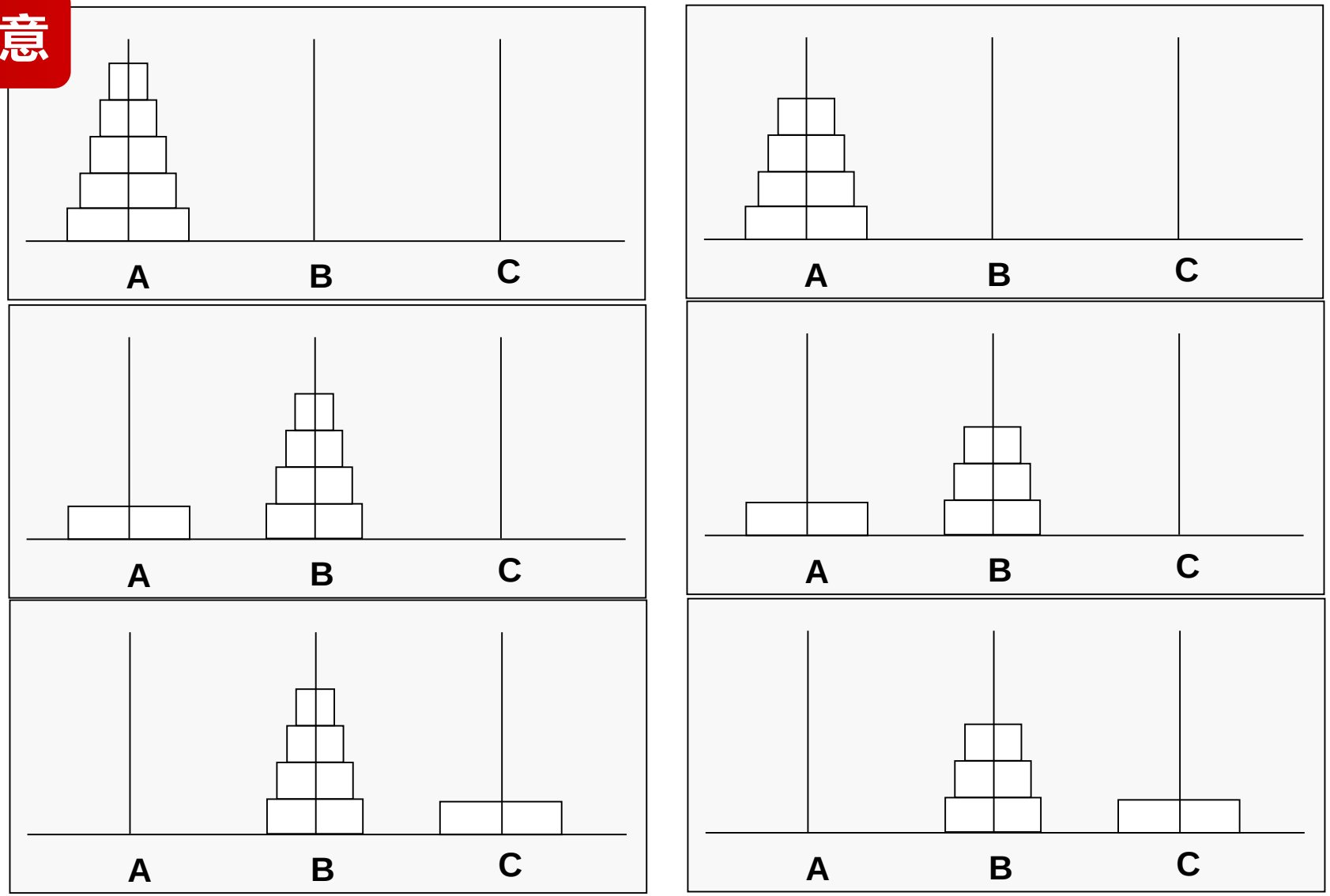
（2）汉诺塔问题

梵天塔(汉诺塔)问题：有三根柱子，梵天将64个直径大小不一的金盘子按照从大到小的顺序依次套放在第一根柱子上形成一座金塔，要求每次只能移动一个盘子，盘子只能在三根柱子上来回移动不能放在他处，在移动过程中三根柱子上的盘子必须始终保持大盘在下小盘在上。



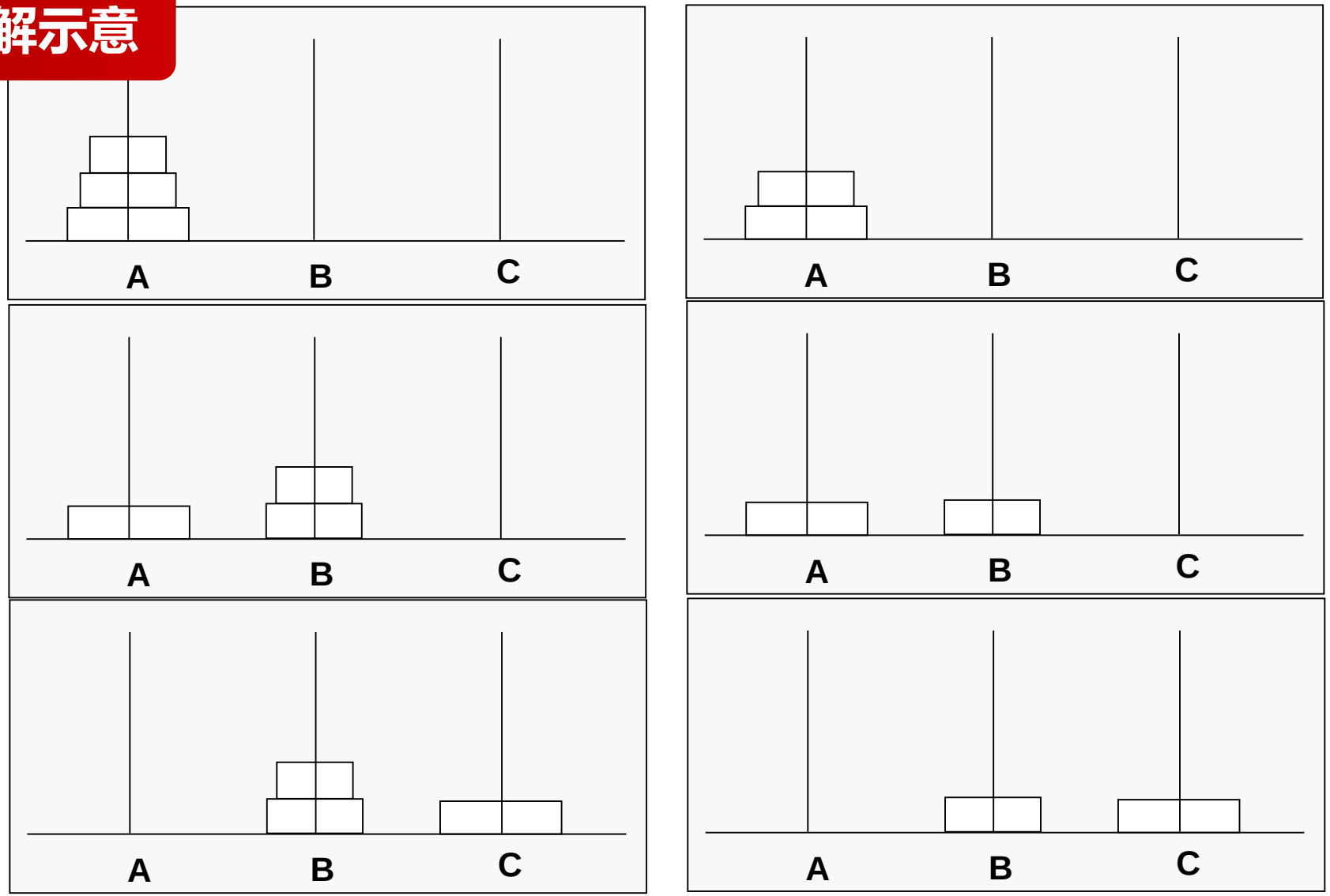
运用递归和迭代：构造与自动执行

（2）汉诺塔问题求解示意



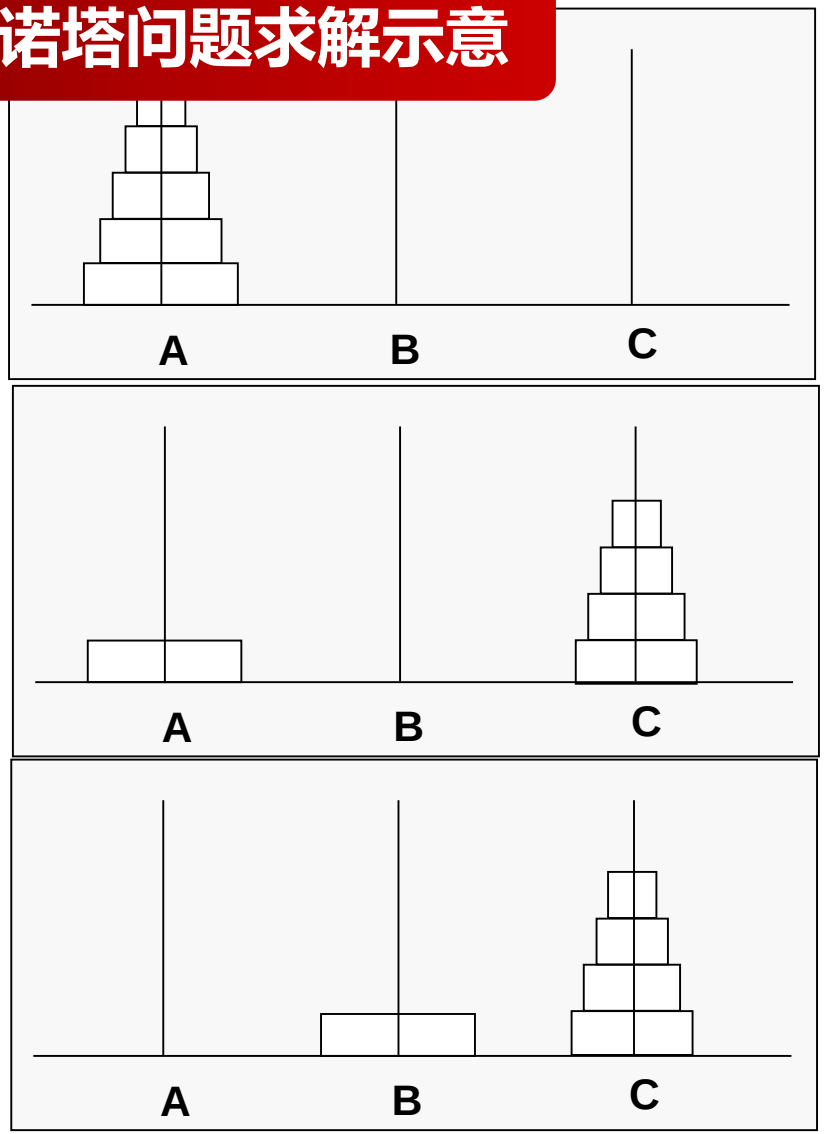
运用递归和迭代：构造与自动执行

（2）汉诺塔问题求解示意



运用递归和迭代：构造与自动执行

（3）汉诺塔问题求解示意

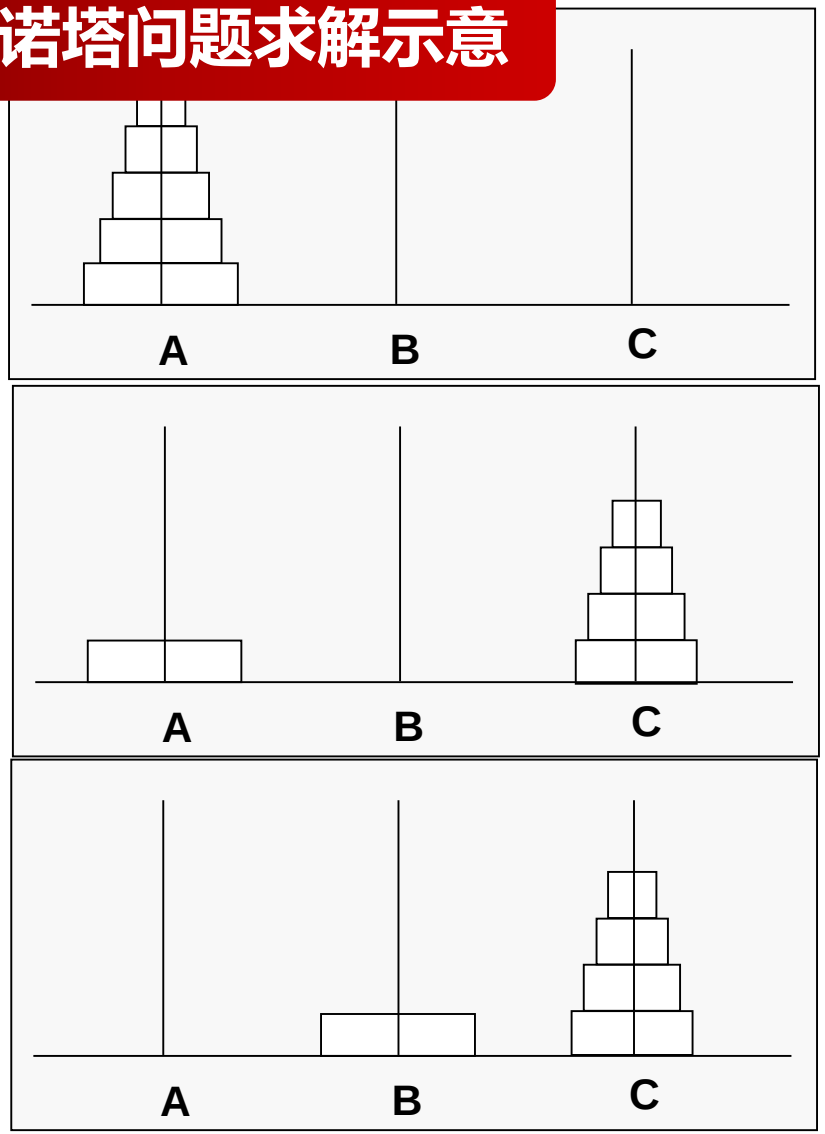



```
C语言
Main()
{  //假设有5个盘子的汉诺塔
     Hanoi(5, “A”, “B”, “C”);
}

int Hanoi (int N, int X, int Y, int Z)
{  //该函数是将N个盘子从X柱，以Z柱做中转，移动到Y柱上
   If  N > 1 Then
   {
     //先把n-1个盘子从X放到Z上(以Y做中转)
          Hanoi (N - 1, X, Z, Y);
     //然后把X上最下面的盘子放到Y上
          Printf(“%d→%d”, X, Y);
     //接着把n-1个盘子从Z上放到Y上(以X做中转)
          Hanoi (N - 1, Z, Y, X);
   }
   Else
   { //只有一个盘子时，直接把它从X放到Y上
          Printf(“%d→%d”, X, Y);
   }
}
```

运用递归和迭代：构造与自动执行

（3）汉诺塔问题求解示意



Python

```
def Hanoi (N, X, Y, Z):
#该函数是将N个盘子从X柱，以Z柱做中转，移动到Y柱上
    if  N > 1:
     #先把n-1个盘子从X放到Z上(以Y做中转)
          Hanoi (N - 1, X, Z, Y)
     #然后把X上最下面的盘子放到Y上
          print( X, Y)
      #接着把n-1个盘子从Z上放到Y上(以X做中转)
          Hanoi (N - 1, Z, Y, X)
    else:
     #只有一个盘子时，直接把它从X放到Y上
          print(X, Y)

def Main():
#假设有5个盘子的汉诺塔
      Hanoi(5, “A”, “B”, “C”)
```

运用递归和迭代：构造与自动执行

(6)递归与迭代构造程序及其执行过程的另一示例

示例：求Fibonacci数列的算法或程序---递归

递归程序

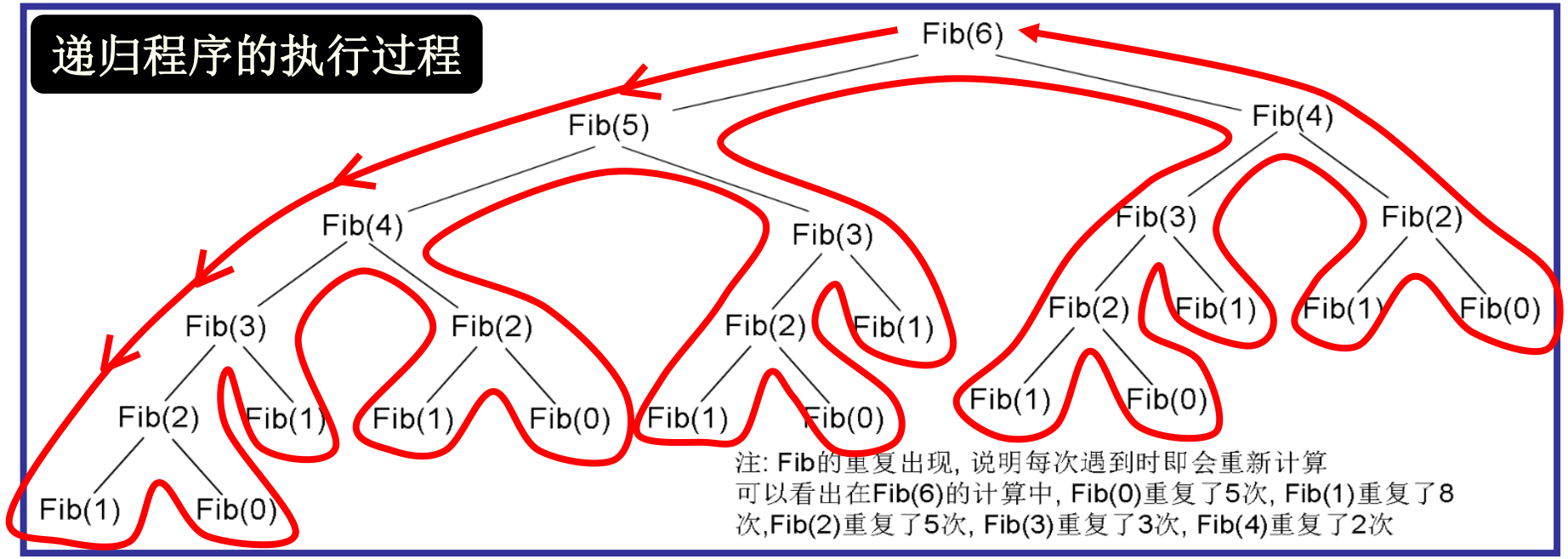
```
(define (fib n) ( cond ((= n 0) 0)
                       ((= n 1) 1)
                       ((> n 1) (+ (fib (- n 1)) (fib (- n 2)))) ))
```

```
(define (fib n) ( cond ((n = 0) 0)
                       ((n = 1) 1)
                       ((n > 1) ( (fib (n - 1)) + (fib (n - 2)))) ))
```

递归定义

$$F(n)=\begin{cases}1 & n=0\\ 1 & n=1\\ F(n-1)+F(n-2) & n>1\end{cases}$$

递归程序的执行过程



运用递归和迭代：构造与自动执行

(6)递归与迭代构造程序及其执行过程的另一示例

示例：求Fibonacci数列的算法或程序---递归

递归程序

Python:

```
def  fib (n):
        if n  ==  0:
                return 0
        elif n==1:
                return 1
        elif n>1:
                return fib(n-1)+fib(n-2)
```

递归定义

$$F(n)=\begin{cases}1 & n=0\\ 1 & n=1\\ F(n-1)+F(n-2) & n>1\end{cases}$$

运用递归和迭代：构造与自动执行

(6)递归与迭代构造程序及其执行过程的另一示例

示例：求Fibonacci数列的算法或程序---迭代

迭代程序

```
(define (fib n) ( fib-iter 1 0 n) )
(define (fib-iter a b count)
        ( cond ((= count 0) b)
               ((> count 0) ( fib-iter (+ a b) a (- count 1) ))))
```

```
(define (fib-iter a b count)
        ( cond ((count = 0) b)
               ((count > 0) ( fib-iter (a + b) a (count - 1) ))))
```

$$F(n)=\begin{cases}1 & n=0\\ 1 & n=1\\ F(n-1)+F(n-2) & n>1\end{cases}$$

迭代程序的执行过程

a	b	Count		(+ a b)	计算内容
1	0	7		1	初始调用
1	1	6		2	f(0)+f(1)
2	1	5		3	f(1)+f(2)
3	2	4		5	f(2)+f(3)
5	3	3		8	f(3)+f(4)
8	5	2		13	f(4)+f(5)
13	8	1		21	f(5)+f(6)
21	13	0			f(6)+f(7)

运用递归和迭代：构造与自动执行

(6)递归与迭代构造程序及其执行过程的另一示例

示例：求Fibonacci数列的算法或程序---迭代

$$F(n)=\begin{cases}1 & n=0\\ 1 & n=1\\ F(n-1)+F(n-2) & n>1\end{cases}$$

迭代程序

Python:

```
def  fib (n):
        return fib_iter(1, 0, n)
def fib_iter(a, b, count):
        if count  ==  0:
                return b
        elif count >0:
                return fib_iter((a + b), a, (count  - 1) )
```

运用递归和迭代：构造与自动执行

(7)递归还有什么?

关于递归的进一步学习

◆递归是计算技术的典型特征，是以有限的表达方式来表达无限对象实例或无限计算步骤的一种经典的计算思维

◆递归覆盖了重复、迭代和递归，递归是最典型的构造手段

◆递归函数是可计算函数的精确的数学描述---计算理论的重要研究内容；

◆(后面将介绍的)图灵机本质上也是递归：图灵可计算函数与递归函数等价，凡可计算函数都是一般递归函数---丘奇-图灵命题---计算理论的重要研究内容

第4讲-程序与递归--二看计算机的本质

本讲小结

什么是程序?
程序的本质是什么?

概念/原理与案例相结合
知识伴随思维，思维贯通知识
相互关联且递进的方式展开与贯通

递归定义、递归算法、递归计算

程序构造的基本方法：递归与迭代

组合/抽象 → 递归

实例层面：运算组合式
概念层面：计算系统与程序

程序本质---组合、抽象、构造与执行

程序---对基本动作的组合
计算系统---执行程序的系统

计算系统的构造