



C语言与程序设计

The C and Programming

第12章 递 归

王多强

1097412466

dqwang@hust.edu.cn

华中科技大学计算机学院

12.1 递归概述

- **递归调用**：函数直接或间接调用自己，这种行为称为**递归调用**。

```
int f(int n)
{
    return n*f(n-1);
}
```

直接递归

```
int g(int x)      int h(int y)
{
    .....
    p = ...;
    h(p);
    .....
}

    .....
    q = .....
    g(q);
    .....
}
```

间接递归

- **递归函数**：含有递归调用的函数称为递归函数。
- **递归程序设计**：设计递归程序的编程技术。

递归分为**直接递归**和**间接递归**

- **直接递归**：函数在自己的函数体中直接调用自己；
- **间接递归**：A函数调用B函数，B函数又调用A函数。

```
int f(int n)
{
    return n*f(n-1);
}
```

直接递归

```
int g(int x)      int h(int y)
{
    .....
    p = ...;
    h(p);
    .....
}

    .....
    q = .....
    g(q);
    .....
}
```

间接递归

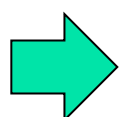
例12.1 用递归法计算阶乘n!

■ n!的递归定义为:

$$n! = \begin{cases} 1 & n = 0, 1 \\ n \times (n-1)! & n > 1 \end{cases}$$

□数学定义的递归式

□n=0或1是边界条件

 $f(n) = \begin{cases} 1 & n = 0, 1 \\ n * f(n-1) & \text{其它} \end{cases}$

递归函数

□递归函数表达式

□n=0或1是递归结束条件

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

```
long factorial(int n); /* 函数原型 */
```

递归程序

```
int main()
```

```
{ int n;
```

```
    printf ("Input an integer n (n>=0) : ");
```

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

```
    if (n < 0)
```

```
        printf ("Input error: n can't be negative\n");
```

```
    else
```

```
        printf ("%d! is %ld\n", n, factorial(n));
```

```
    return 0;
```

```
}
```

$$f(n) = \begin{cases} 1 & n = 0, 1 \\ n * f(n - 1) & \text{其它} \end{cases}$$

递归函数的初次调用

```
long factorial(int n)
```

```
{
```

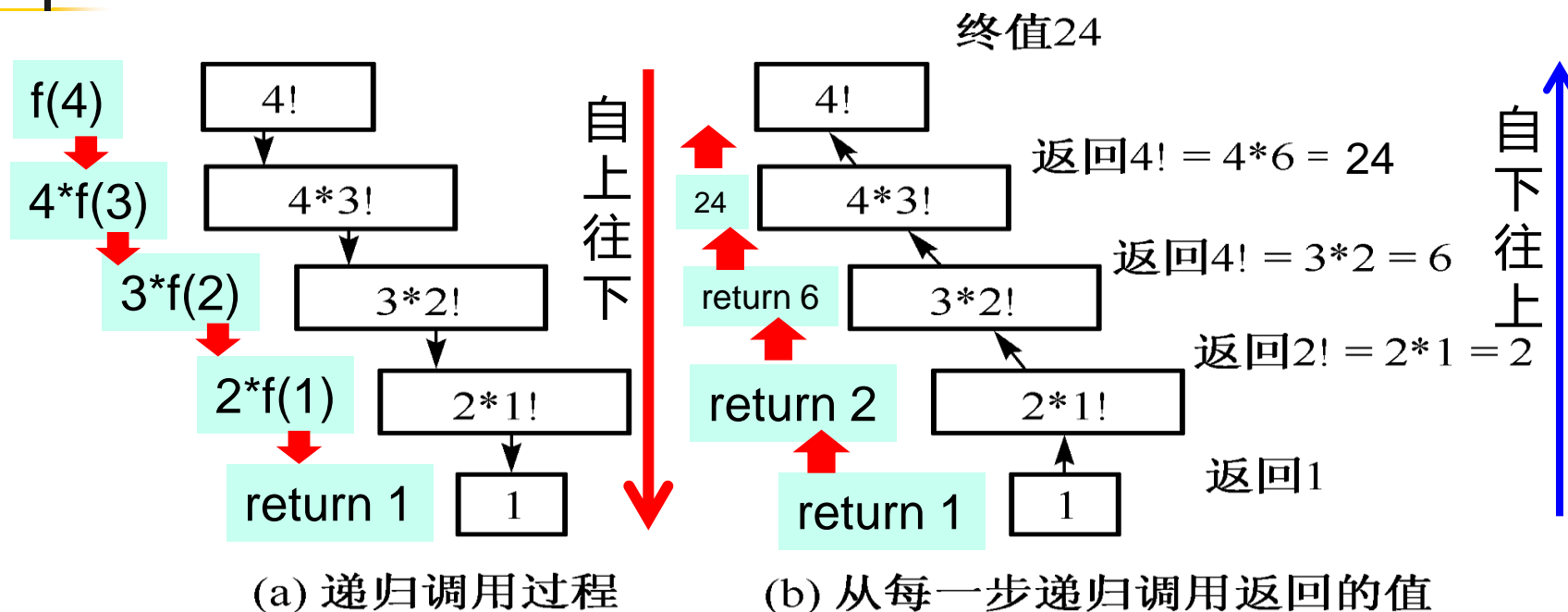
```
    if (n==0 || n==1) /* 递归结束条件 */
```

```
        return 1;
```

```
    else return (n * factorial(n-1)); /* 递归调用 */
```

```
}
```

□ 计算4!的递归执行过程



- 图(a)是递归调用的过程，递归过程中每次 n 减1，直到 n 等于1时终止递归调用；
- 图(b)是当递归调用终止时，把值返回给调用处的过程，这个过程直到计算出并返回最终值为止。

■ 递归的两个要素

(1) 递归结束条件

■ n最小的时候

- 这个时候不再做深层次的递归调用，**直接计算当前的值**；
 - 如果没有结束条件，递归过程将不会终止，陷入无穷递归（类似死循环）。
 - 无穷递归的最后结果是耗尽内存，使系统不能正常工作甚至死机。

(2) 递归定义

■ 递归调用部分，给出递归计算的方法

- 递归定义必须能使问题变得越来越简单，**使问题向结束条件转化。**

■ 递归算法的特点

- **递归的本质是一个循环执行过程**，可向循环转换；
- **优点：**符合数学模型，只需少量代码就可描述出需要多次重复计算的解题过程，简单、易于理解。
结构紧凑、清晰、可读性强、代码简洁。
- **缺点：****存在大量的函数调用，运行效率较低。**
 - 函数调用的时、空开销比较大
 - 重复子问题的计算带来的问题

12.2 递归函数设计

- 递归是一种强大的解决问题的技术，其**基本思想**是将一个复杂问题逐步转化为简单问题进行求解。
在高级程序设计中，递归是一个很重要的概念。
- 设计递归程序的关键是构建 “**自循环**” 计算过程。

12.2.1 字符串的递归处理

- 将字符串看成一种递归的定义：**字符串或者是一个空串，或者是 “一个字符后面再跟一个字符串”。**
- 可以用递归的方法对一些基本的字符串处理函数进行改写。

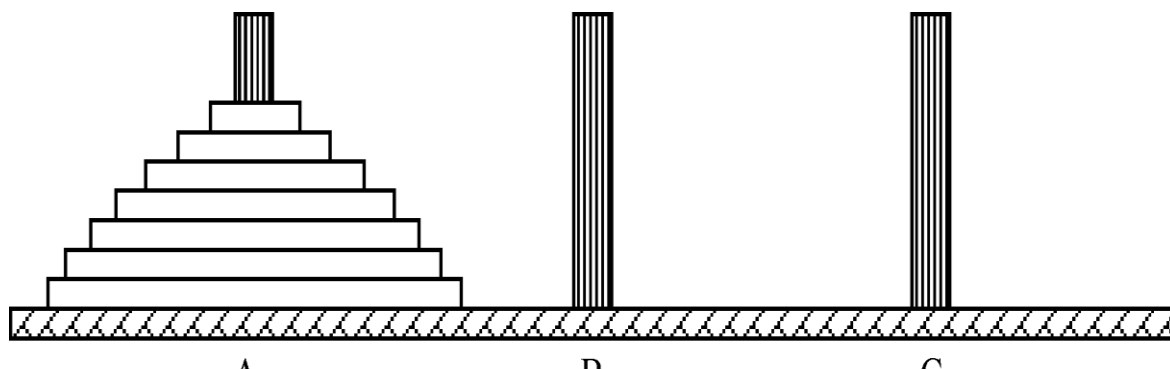
例 12.2 用递归实现字符串的比较函数

```
int Strcmp(char *s, char *t)
{
    if(*s!=*t||*s=='\0')           /* 递归结束条件 */
        return(*s-*t);
    else
        return(Strcmp(++s, ++t)); /* 递归调用 */
}
```

12.2.2 汉诺塔(HANOI)问题

- 著名问题，来自于印度布拉玛神庙中教士玩的游戏。

问题：一块木板上有**A**、**B**、**C**三个木桩。木桩A上有64个盘子，盘子大小不等，大的在下，小的在上。把木桩A上的64个盘子都移到木桩C上，条件是一次只允许移动一个盘子，且不允许大盘放在小盘的上面，在移动过程中可以借助木桩B。



汉诺塔问题号称“世界末日问题”，因为“当所有的64个圆盘全部移完的那天就是世界的末日”了——64个盘子的总移动次数是 $2^{64}-1=18446744073709551615$ 。就算每一微妙能够计算一次移动，也需要几乎一百万年。

例12.3 设计一个求解汉诺塔问题的算法。

□ 分析：

- 原问题可以看成：A柱上的 n 个盘子借助B柱移到C柱上。
- 从而可以形成下面的步骤：
 - (1) 把A柱上面的前 $n-1$ 个盘子借助C柱移到B柱上；
 - (2) 把A柱上剩下的那个最大的盘子移到C柱；
 - (3) 把B柱上的 $n-1$ 个盘子借助A柱移到C柱——递归子问题。

第(1)步和第(3)可以看成较小子问题 ($n-1$ 个盘子) 的求解，
从而可以构造一个递归过程，

递归结束条件是 n 为1。

- 设计一个函数: **move(n,a,b,c)**, 表示将n个盘子按照上述规则从a柱上借助b柱移动到c柱上。则上述第(1)步和第(3)步的实现可以用递归表示为:

move(n-1,a,c,b)和**move(n-1,b,a,c)**。

- 步骤(2)只简单地将第n个盘子移到C柱上。

```
void move(int n, int a, int b, int c)
{
    static int i = 1;          /*统计移动总数*/
    if (n == 1)                /* 递归结束条件 */
        printf("step %d: %c-->%c\n", i++, a, c); /* 把a上盘子移到c */
    else {
        move (n-1, a, c, b);    /* 步骤(1)的递归调用 */
        printf("step %d: %c-->%c\n", i++, a, c); /* 把a上剩下的盘子移到c */
        move(n-1, b, a, c);    /* 步骤(2)的递归调用*/
    }
}
```

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

\源程序\ex12_3.c

```
#include<stdlib.h>
```

```
void move(int, int, int, int); /* 函数原型 */
```

```
int main()
```

```
{
```

```
int n, a = 'A', b = 'B', c = 'C';
```

```
printf ("\n-----TOWERS OF HANOI-----\n");
```

```
printf ("The problem starts with n disks on Tower A.\nInput n : " );
```

```
if (scanf("%d", &n)!=1 || n<1)
```

```
{ printf("\nERROR:Positive integer not found\n"); return -1; }
```

```
move(n, a, b, c); /*递归调用*/
```

```
return 0;
```

```
}
```

12.2.3 排列问题

▣ **排列问题**：从 n 个不同元素中任取 m 个，按任意一种次序排成一行， $1 < m \leq n$ ，称为**排列**。

例12.4 找出从 $1 \sim n$ 中任取 m 个数的所有排列

例如， $n=3$ 、 $m=3$ 时，所有排列为：

$(1,2,3)$ 、 $(1,3,2)$ 、 $(2,1,3)$ 、 $(2,3,1)$ 、 $(3,1,2)$ 、 $(3,2,1)$ 。

分析：

设计递归函数PrintPerm(a,n,m,cur): 从1 ~ n依次选1个之前在a[0] ~ a[cur-1]中未出现的元素, 放到a[cur]中。

- ◆ 初始调用: PrintPerm(a,n,m,0);
- ◆ 将某个数放入cur位置后, 有两种可能的选择:
 - (1) 当cur=m时, 已取了m个数, 输出这m个数的排列。
 - (2) 当cur<m时, 递归调用**PrintPerm(a,n,m,cur+1)**, 找下一个数。


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

\源程序\ex12_4.c

```
void PrintPerm(int *a, int n, int m, int cur);  /* 函数原型 */
```

```
int main()
```

```
{
```

```
    int n, m, a[100];
```

```
    printf("Input n:");    scanf("%d", &n);
```

```
    printf("Input m:");    scanf("%d", &m);
```

```
    PrintPerm(a, n, m, 0);
```

```
    return 0;
```

```
}
```

初始调用



```
void PrintPerm(int *a, int n, int m, int cur)
{ int i, j, count;
  if (cur == m) /* 已取了m个数，输出这m个数构成的排列*/
    { for (i=0; i<m; i++) printf("%d ", a[i]); printf("\n"); }
  else /* 否则 */
    for (i=1; i<=n; i++)
      { int ok = 1; /* 标志变量*/
        for (j=0; j<cur&&ok; j++) /* 检查i是否已用过 */
          if (a[j] == i) { ok = 0; break; } /* i出现过，不能再选 */
        if (ok) /* 未出现过 */
          { a[cur] = i;
            PrintPerm(a, n, m, cur+1); /* 递归调用*/
          }
      }
}
```

12.3 分治法与快速排序（自学）

- **分治法的基本思想**：当问题的规模比较大的时候，将大问题划分成若干**互相独立**且与原问题具有**相同的性质**的子问题，然后求解子问题，最后将子问题的解合并成原始问题的解。
 - 上述过程中，如果子问题的规模还比较大，可以继续分解子问题，直到得到的子问题足够小、可以直接求解为止。
- **由分治法产生的子问题是原问题的较小模式，从而可以使用递归方法求解。**

例12.5 QuickSort排序算法

- 快速排序算法(QuickSort)是C.A.R.Hoare于1962年发明的。
- **其基本思想是：**
 - 对于一个给定的数组，从中选择一个元素（pivot）；
 - 把其余元素与pivot进行比较，从而划分为两个子集合：一个是由所有小于等于pivot的元素组成的子集合，另一个是由所有大于pivot的元素组成的子集合。调整后pivot放在两个子集合之间。
 - 然后对两个子集合递归应用同一过程。当子集合中的元素个数小于2时，不需要再划分，递归返回。

▣ 划分数组的函数：partition定义如下。

```
int partition(int a[ ],int left,int right )
{
    /* 将数组a中left~right的元素分成两部分，返回切分点的下标 */
    int i=left, j=right+1;
    for ( ;; )
    {
        while(a[++i]<=a[left] && i <= right);          /*从左至右扫描 */
        while(a[--j]> a[left]);                          /*从右至左扫描 */
        if(i>=j) break;                                  /* 扫描相遇（或交叉）结束循环 */
        swap(a,i,j);                                     /* 交换左右两边的元素 */
    }
    swap(a,left,j);          /*j 是切分元素的位置，将切分元素重新移到j位置*/
    return j;              /* 返回切分元素的下标 */
}
```

QuickSort程序实现如下:

```
void QuickSort(int a[ ],int left,int right)
```

```
{
```

```
    int split;          /* 分区位置 */
```

```
    if(left<right)
```

```
    {
```

```
        split =partition(a,left,right);      /* 将数组元素分成两部分,左边部分的所有元素  
                                              小于等于右边部分的所有  
                                              元素*/
```

```
        QuickSort(a,left,split-1);          /* 对左边部分递归排序 */
```

```
        QuickSort(a,split+1,right);        /* 对右边部分递归排序 */
```

```
    }
```

```
}
```



■ 本章作业

12.1 12.2

提高题： 12.3