第5章

递　归

CHAPTER 5

本章学习目标

- 了解递归的基本概念
- 理解递归调用的实现原理
- 掌握递归算法的设计

在线自测题

在计算机科学中，递归算法是一种将问题不断分解为同一类子问题来解决问题的方法。递归方法可解决许多计算机科学问题，因此它是计算机科学中非常重要的概念。

本章先介绍递归的基本概念，再介绍递归调用的实现原理，最后介绍递归算法设计方法。


视频讲解

5.1 递归的定义

5.1.1 递归的基本概念

如果一个对象部分包含它自己,或者利用自己定义自己,则称这个对象是递归的;如果一个过程直接或间接调用自己,则称这个过程是一个递归过程。

递归不仅是数学中的一个重要概念,也是计算技术中重要的概念之一。20 世纪 30 年代,递归函数理论、图灵机演算理论和 POST 规范系统等理论一起为计算理论奠定了基础。

在人们的思考过程中,普遍存在着递归现象和递归机制。它是一种从简单到复杂、从低级到高级的可连续操作的问题解决方法。

5.1.2 何时使用递归

以下三种情况适用于递归方法解决问题。

1. 问题的定义是递归的

阶乘函数、幂函数和斐波那契数列等函数的定义是递归的。求解这些问题可以将其递归定义直接转换为相应的递归算法。

例如,求函数 n! 的递归算法如下:

```
1. long Factorial(long n)
2. {
3.     if(n == 1)
4.         return 1;
5.     else
6.         return n * Factorial(n - 1);
7. }
```

在函数 Factorial(long n)的求解过程中调用 Factorial(n－1),即函数 Factorial()自己调用自己,所以它是一个直接递归函数,又由于递归调用 Factorial(n－1)是最后一条语句,所以它又属于尾递归。

2. 问题所涉及的数据结构是递归的

单链表的数据结构是递归的,其结点类型定义如下:

```
1. typedef struct LNode
2. {
3.      ElemType data;
4.      struct LNode * next;
5. }LinkNode;
```

该定义中,结点 LNode 的定义中用到了它自身,即指针域 next 是指向其自身类型的指针,所以结点 LNode 是一种递归数据结构。

3. 问题的解法满足递归的性质

Hanoi 问题的解决方法是递归的。一块板上有三根针 X,Y,Z。X 针上套有 n 个大小不等的圆盘,大的在下,小的在上。要把这 n 个圆盘从 X 针移到 Z 针上,移动时需要遵守以下规则,每次只能移动一个圆盘,移动时可以借助 Y 针。任何时候圆盘都必须保持大盘在下,小盘在上的规则。

Hanoi 问题的递归分解过程是:

```
                    分解    Hanoi(n-1,x,z,y);
Hanoi(n,x,y,z)      ⇒       move(n,x,z);(将第 n 个圆盘从 x 移向 z)
                            Hanoi(n-1,y,x,z);
```

首先调用函数 Hanoi(n−1,x,z,y),将 x 塔座上的 n−1 个盘片借助 z 塔座移动到 y 塔座上;此时 x 塔座上只有一个盘片,调用函数 move(n,x,z)将其直接移动到 z 塔座上;再调用函数 Hanoi(n−1,y,x,z)将 y 塔座上的 n−1 个盘片借助 x 塔座移动道 z 塔座上。

5.1.3　递归模型

为了更好地利用递归求解问题,通过求解 n!问题的递归算法认识递归结构,并抽象出递归模型。

```
1. long Factorial(long n)
2. {
3.     if(n == 1)
4.         return 1;
5.     else
6.         return n * Factorial(n - 1);
7. }
```

一般地,一个递归模型是由递归出口和递归体两部分组成。递归出口确定递归到何时结束;递归体确定递归求解时的递推关系。如此可类推出递归出口和递归体的一般格式。

递归出口的一般格式:

$$f(s_1) = m_1$$

其中,s_1 与 m_1 均为常量,有些递归问题可能有几个递归出口。求解 n! 的过程中,fun(1)=1 就是 $f(s_1)=m_1$,是唯一递归出口。

递归体,一般格式:

$$f(s_n) = g(f(s_i), f(s_{i+1}), \ldots, f(s_{n-1}), c_j, c_{j+1}, \ldots, c_m)。$$

其中,g 是一个非递归函数,$c_j, c_{j+1}, \ldots, c_m$ 为常量。如求 n!,fun(n)=n * fun(n−1)(n>1)就是 $f(s_n)=g(f(s_i), f(s_{j+1}), \ldots, f(s_{n-1}), c_j, c_{j+1}, \ldots, c_m)$,只是"$c_j, c_{j+1}, \ldots, c_m$"均为 0,"$f(s_i), f(s_{i+1}), \cdots, f(s_{n-1})$"中仅存在一个 fun(n−1),此外 $f(s_n)$为 f(n),g 为 n * fun(n−1)。

在递归模型中,我们可以看到递归等式左边的 $f(s_n)$与等式右边的"$f(s_i), f(s_{i+1}), \ldots, f(s_{n-1})$"格式相同,其代表原求解的大问题转化成若干个相似子问题。

递归的思路就是把一个不能或不好直接求解的"大问题"转换成一个或几个与"大问题"

相似的“小问题”来解决；若仍无法解决，则再把这些“小问题”进一步分解成更小的相似的“小问题”来解决。


视频讲解

5.2　递归调用的实现原理

递归调用的内部实现原理可以理解为调用与自己有相同的代码和同名的局部变量的子程序。

在执行调用时，计算机内部执行如下操作：

1. 开辟栈顶存储空间，用于保存返回地址、被调层函数中的形参和局部变量的值。
2. 为被调层函数准备计算实参的值，并在栈顶元素中赋给对应的形参。
3. 转入子程序执行。

在执行返回操作时，内部执行如下操作：

1. 若函数需要求值，将其值保存到回传变量中。
2. 从栈顶取出返回地址，并退栈，同时撤销被调层子程序的局部变量及形参。
3. 按返回地址返回。

在返回后自动执行如下操作：

若函数需要求值，从回传变量中取出所保存的值并传送到相应的实变参或位置上。

算法求解 Factorial(5)的递归调用过程中，程序执行及栈的变化情况如图 5.1 所示。

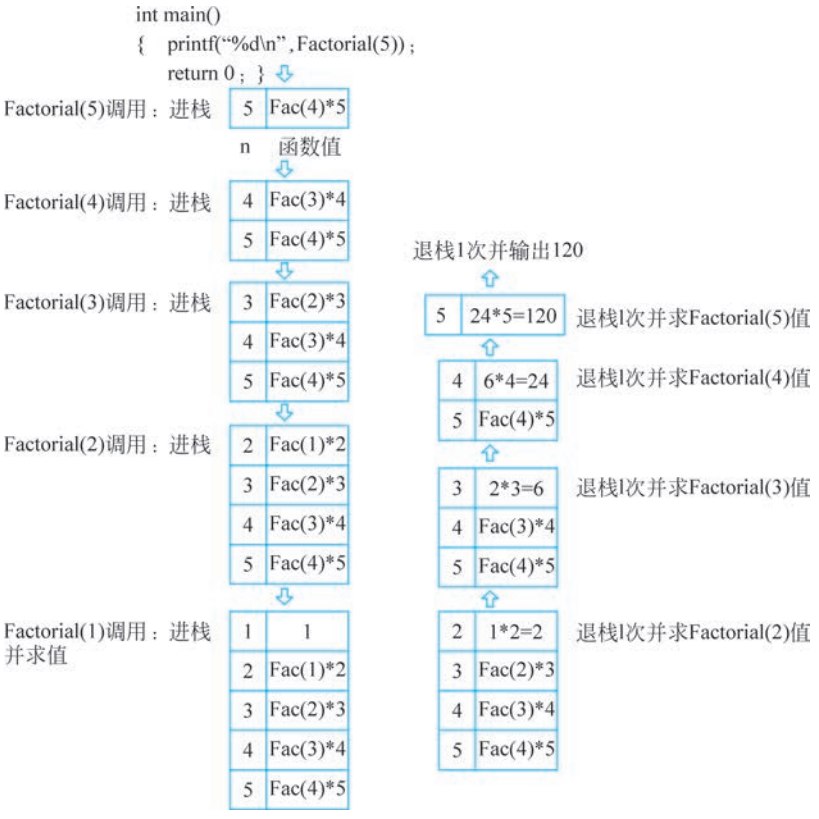

图 5.1　fun(5)的执行过程示意图

5.3　递归算法的设计


视频讲解

5.3.1　递归算法设计的步骤

递归算法设计先要确定递归模型，再转换成对应的 C 语言函数。

（1）对原问题 $f(s_n)$进行分析，抽象出合理的“小问题”$f(s_{n-1})$；

（2）若 $f(s_{n-1})$是可解的，则在此基础上确定 $f(s_n)$的解，即给出 $f(s_n)$与 $f(s_{n-1})$之间的关系；

（3）确定一个待定情况（如 f(1)或 f(0)）的解，作为递归出口。

【例 5-1】　求顺序表$(a_1,a_2,\cdots,a_n)$中的最大值。

将顺序表分解成左子表$(a_1,a_2,\cdots,a_m)$和右子表$(a_{m+1},a_{m+2},\cdots,a_n)$两个子表，分别求出子表中的最大元素 a_i 和 a_j，求出二者中最大元素，即为整个顺序表的最大元素。

求子表中最大元素的方法与总表相同，即将子表分成两个更小的子表，如此不断分解，直至表中只有一个元素，该元素是该表最大元素。

```
1. ElemType Max(SqList L,int i,int j)
2. {
3.       int mid;
4.       ElemType max,max_r,max_l;
5.       if(i == j)
6.            max = L.data[i];
7.       else
8.       {
9.       mid = (i + j)/2;
10.      max_r = Max(L,i,mid);
11.      max_l =  Max(L,mid + 1,j);
12.      if(max_r > max_l)
13.            max = max_r;
14.      else
15.            max = max_l;
16.      }
17.      return max;
18.}
```

5.3.2　递归数据结构的递归算法设计

具有递归特性的数据结构称为递归数据结构。递归数据结构通常采用递归方式定义。递归的明显特征是一对象可以表示成包含它本身的结构，如不带头结点的单链表，其结点类型定义如下：

```
1. typedef struct Lnode
2. {
3.       ElemType data;
4.       struct LNode * next;
5. }LinkList;
```

该定义中,结构体 Lnode 的定义中用到了它自身,即指针域 next 是一种指向自身类型的指针,它是一种递归数据结构。

对于递归数据结构,采用递归的方法编写算法既方便又有效。

【例 5-2】 求一个不带头结点的单链表 head 的所有 data 域之和的递归算法。

```
1. ElemType Sum(LinkList *L)
2. {
3.      if(L==NULL)
4.           return 0;
5.      else
6.           return L->data+Sum(L->next);
7. }
```

5.3.3 递归求解方法的递归算法设计

有些问题的解法是递归的,典型的有 Hanoi(汉诺)塔问题求解。

【例 5-3】 汉诺塔问题。

设 Hanoi(n,x,y,z)表示将 n 个盘片从 X 通过 Y 移动到 Z 上,递归分解的过程如图 5.2 所示:

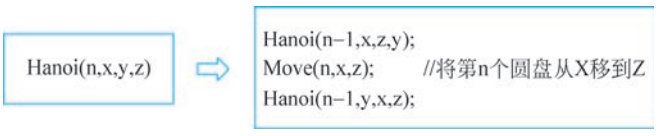

图 5.2 Hanoi 递归分解过程示意图

```
1. void Hanoi(int n,char x,char y,char z)
2. {
3.      if(n==1)
4.           printf("将第%d个盘子从%c上移动到%c上.\n",n,x,z);
5.      else
6.      {
7.           Hanoi(n-1,x,z,y);
8.           printf("将第%d个盘子从%c上移动到%c上.\n",n,x,z);
9.           Hanoi(n-1,y,x,z);
10.      }
11. }
```

汉诺塔的解题思路很简单,就是按照移动规则向一个方向移动盘片:如果只有一个盘片,则把该盘片从 X 移动到 Z,结束;如果有 n 个盘片,则把前 n-1 个盘片移动到辅助的 Y,然后把 X 上的盘片移动到 Z,最后再把前 n-1 个移动到 Z。

汉诺塔问题是递归求解方法中的经典递归问题。

5.4 本章小结

本章首先介绍了递归的基本概念;然后介绍了递归调用的实现原理,并说明了递归执

行的过程；最后介绍了递归算法的设计方法。通过本章的学习，可以对于递归有了基础认识，能够运用递归算法解决一些较复杂的应用问题。

习题 5

一、简答题

1. 什么叫递归？
2. 阶乘问题的循环结构算法和递归算法哪个的时间效率好？为什么？

二、算法设计题

1. 编写一个函数求 n 的阶乘算法。
2. 编写一个函数求斐波那契数列前 n 项的和算法。
3. (1) 写出求 1,2,3,…,n 的 n 个数累加的递推公式；
   (2) 编写求 1,2,3,…,n 的 n 个数累加的递推算法，假设 n 个数存放在数组 a 中。
4. 设有一个不带表头结点的单链表 L，设计一个递归算法，求以 L 为首结点指针的单链表的结点个数。
5. 设有一个不带表头结点的单链表 L，设计一个递归算法，求单链表 L 中的最小结点值。