DOI: 10.19392/j.cnki.1671-7341.202118028

基于阶乘的递归算法内部研究

辜双佳[1] 栗 智[2]

1.重庆理工大学计算机科学与工程学院 重庆 400000;2.重庆大学 重庆 400000

摘 要:直接或间接地调用自身的函数称为递归函数。不管数据结构本身是否具有递归属性,当用到递归技术时,我们可以更加直观地解释函数与算法,让阅读者更好地理解算法的内部运行。但其实很多人并不知道调用递归函数时程序的具体执行过程,这要归结于“栈”这种数据结构。在该论文中,我们将在“阶乘”的基础上,分析递归函数的具体执行过程。本文代码采用 C 语言实现。

关键词:递归;阶乘;C 语言

1 绪论

1.1 递归的概念

直接或间接地调用自身的算法称为递归算法[1]。用函数自身给出定义的函数称为递归函数[1]。直接调用自己称为直接递归,间接调用自己称为间接递归。在计算机算法设计与分析中,递归技术是十分有用的。使用递归技术往往使函数的定义和算法的描述简捷且易于理解。有些数据结构,如二叉树等,由于其本身固有的递归特性的递归特性特别适合用递归的形式来描述。有些问题,虽然本身没有递归结构,但用递归技术可以使设计的算法简洁易懂且易于分析[1]。

1.2 分治和递归的关系

将一个难以直接解决的大问题分割成一些规模较小的相同问题,以便各个击破,即分而治之[1]。分治和递归就像是一对孪生兄弟,经常一起应用于算法设计中,并由此产生了许多的高效算法。我们可以把它理解为:分治策略用递归算法实现。

1.3 递归适用的场合

递归算法的基本思想是:把规模大的、较难解决的问题变成规模较小的、易解决的同一问题。规模较小的问题又变成规模更小的问题,并且小到一定程度可以直接得出它的解(递归出口)[2]。故在解决现实问题中,对于求解一个复杂的或者问题规模较大的问题,我们在将其划分为一些简单的或者规模较小的问题时,如果满足:(1)所划分成的子问题性质与原来的大问题相同。(2)当问题规模小到一定程度的时候直接有解。对于满足以上条件的问题我们就可以考虑使用递归的方法求解。

1.4 栈(stack)

栈是只允许在一端进行插入或删除操作的线性表。首先栈是一种线性表,但是限定这种线性表只能在某一端进行插入或删除操作[3]。形象地理解就是:栈就像一个圆柱形的桶,先放进去的东西被压在了最底下(栈底),后放进去的东西依次放在了上面(栈顶),不放任何东西(元素)的栈叫作空栈。放入东西的操作由栈顶指针(top)来控制,即每放进去一样东西则栈顶指针(top)自动加一。初学者请记住一句最关键的话就是:先进后出。

1.5 递归过程与递归工作栈

递归过程在实现时,需要自己调用自己。层层向下递归,退出时的次序正好相反,主程序第一次调用递归过程为外部调用,递归过程每次递归调用自己为内部调用,它们返回调用它的过程的地址不同。函数调用针对主调函数和被调用函数分别进行三步操作。

√主调函数:

①开辟空间(工作栈);

②实参→形参;

③控制权给被调用函数(即 top(栈顶指针)移动)。

工作栈在内存中,工作栈里面会依次存放返回地址(下一条指令)、局部变量、参数

√被调函数:

①返回值;

②释放空间;

③控制权给被调用函数(即 top(栈顶指针)移动)。

每一次递归调用时,需要为过程中使用的参数、局部变量等另外分配存储空间。每层递归调用需分配的空间形成递归工作记录,按后进先出的栈组织。如图 1 所示:

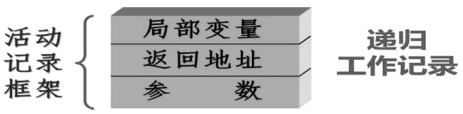

图 1 阶乘的递归表达式

1.6 总结

本文将结合“阶乘”这个经典的递归算法,用“栈”这种数据结构分析数据的出栈入栈过程,让学习递归的人真正知道调用递归函数时程序的具体执行过程。

2 阶乘

2.1 引言

所谓阶乘,即递减数字的乘积。一个正整数的阶乘(factorial)是所有小于及等于该数的正整数的积,并且 0 的阶乘为 1。自然数 n 的阶乘写作 n!。亦即 $n! = 1\times2\times3\times\cdots\times n$。

从阶乘的定义,我们不难找到规律,所以阶乘亦可用递归方式定义:$0! = 1$,$n! = (n-1)! \times n$。初学者请特别注意一点,那就是递归函数一定由两部分组成,一个叫作递归方程,

他是递归函数的入口，是函数调用自身的地方，这里也就是 n! =(n-1)! ×n，其中 n 为正整数。另一个叫边界条件，他是递归函数的出口，递归函数的出口一般为一个返回值，这里也就是 0! =1，每个递归函数都必须有非递归定义的初始值(边界条件)，否则递归函数将永远无限地调用下去导致递归函数无法计算[1]。

2.2 阶乘

阶乘的数学表达式为：n! =1×2×3×…×n，阶乘的递归定义如图 2 所示：

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

图 2 阶乘的递归表达式

边界条件与递归方程是递归函数的二个要素，递归函数只有具备了这两个要素，才能在有限次计算后得出结果。

2.3 运行结果

通过在 vc6.0 上面编辑、编译、连接、运行 C 语言代码，得到了以下的运行结果，为了便于后面栈的分析，我们这里选择输入一些比较小的数，比如 3 和边界值 0，如图 3、如图 4 所示：

图 3 3! 的运行结果

图 4 0! 的运行结果

2.4 结果分析

特别说明，为了后面描述方便，有以下代替：

"1"：m=factorial(s)；

"2"：printf("%d 的阶乘为：\n",s)；

"3"：int factorial(int n)；

"4"：return n* factorial(n-1)；

"5"：else 后面的}

"6"：if(n==0)

当输入 3 的时候，s=3，然后去执行"1"(函数调用)，此时在内存开辟了空间，也就是开辟了一个工作栈，里面放了三样东西，分别是一会儿的返回地址(即"2")、局部变量的值(m)、参数(即 s=3)。接着把实参传给形参，也就是把 s 的值传给 n，则 n=3。最后将控制权给被调函数，也就是"3"，此时栈顶指针(top)移动。然后来到我们的被调用函数"3"，此时 n=3，执行 else 进入递归(即"4")，此时的返回值为"5"，局部变量 n=3，然后又执行 else 进入递归(即"4")，此时的返回值为"5"。局部变量 n=2，然后又执行 else 进入递归(即"4")，此时的返回值为"5"。局部变量 n=1，然后执行 if 进入边界条件(出口)(即"6")，此时的返回值为"5"。

到此为止，我们的递归函数终于找到出口了，大家可能也再次深刻地理解了为什么一个递归函数必须有边界条件，即"每个递归函数都必须有非递归定义的初始值(边界条件)，否则递归函数将永远无限地调用下去导致递归函数无法计算"。

接着才是返回，这也是一个收获的过程，人们最容易在这个地方感到困惑，感觉脑子里面理不清的退回。因为栈有"先进后出"的特点，所有返回时的次序正好相反，而这个时候，最重要的就是知道每一次递归调用时，使用的参数、局部变量和返回值具体是什么。

基于前面仔细地记录，我们知道当执行到局部变量 n=1，得到返回值，进入出口(即"6")，此时的返回值为"5"，即结束被调用函数，此时整个结果为 1，然后返回给调用它的地方，即"4"里面的 factorial(n-1)的结果为 1(此时 n=2)，然后执行"4"，返回 n* factorial(n-1)，即 2* 1，此时的返回值为"5"，即结束被调用函数，此时整个结果为 2* 1，然后返回给调用它的地方，即"4"里面的 factorial(n-1)的结果为 2* 1(此时 n=3)，然后执行"4"，返回 n* factorial(n-1)，即 3* 2* 1，此时的返回值为"5"，即到此为止得到最终的递归值为 3* 2* 1，即 3! =6，即最终返回的递归值为 6 传给 m 进而输出到控制台。

我们可以看到这是一个非常绕的过程，但是只要将返回地址、局部变量、参数的值分别压进栈里面再依次弹出，那么就不会乱。到此递归最精华和核心的地方就说完了。当然，后面还有释放空间和将控制权交回给被调用函数，到此为止才是一个完整的递归函数调用过程。

3 总结

本节以一个简单又经典的例子："阶乘"，详细地讲解了递归调用的内部原理，我们在初学时深入了解递归的内部原理和执行过程才可以真正学懂递归，也为我们会用递归创造了可能。但从文字表述也可以发现这个过程很绕，有很多重复的工作，但那是计算机关心的过程，我们在进行程序设计的时候只需要掌握这种思想就可以了。在理解了"阶乘"的基础上，大家可以进一步用画工作栈中存放的返回地址(下一条指令)、局部变量、参数来验证"Fibonacci 数列""全排列""Hanoi 塔""快速排序""二分查找"等经典的递归算法。

参考文献：

[1] 王晓东. 计算机算法设计与分析(第五版). 电子工业出版社，2013：11.

[2] Thomas H. Cormen Charles E. Leiserson，Ronald L. Rivest Clifford Stein. Introduction to Algorithms 3nd. 2013：52.

[3] 王道论坛组编. 数据结构联考复习指南(第一版). 电子工业出版社，2016：4.