

12 | 数学归纳法：搞定循环与递归的钥匙

2020-02-11 胡光

人人都能学会的编程入门课

[进入课程 >](#)



讲述：胡光

时长 18:24 大小 14.75M



你好，我是胡光，今天我们正式开始“编码能力训练篇”的学习。

这里给你一个建议，在刚刚完成了语言基础篇的学习后，我希望你用心地体验“螺旋式上升”的学习过程。就是前面的基础篇虽然学完了，可并不是意味着，不需要再学习更多的语言相关的东西了，你可以做如下两件事情：


1. 对于语言基础，你可以选择学习第二遍，当你站在第一遍的基础上，再回头看的时候，肯定会对之前的知识有更深入的理解；
2. 选择在其他参考资料中，继续学习语言中更多的知识点。你会发现，某些之前自己认为晦涩难懂的东西，可以自学搞明白了，这就是我提到的“螺旋式上升”的学习方法。



在接下来的“编码能力训练篇”里，我将着重给你讲解一些编程中的重要技巧。今天呢，我们就从理解循环与递归的编码技巧开始吧！


今日任务

循环结构，你已经不陌生了，如下代码所示，是一个单层循环的程序，依次地输出从 1 到 n 的每一个数字，每个数字占一行：

 复制代码

```
1 #include <stdio.h>
2
3 int main() {
4     int n;
5     scanf("%d", &n);
6     for (int i = 1; i <= n; i++) {
7         printf("%d\n", i);
8     }
9     return 0;
10 }
```

当我们输入 4 的时候，程序的输出结果如下所示：

 复制代码

```
1 1
2 2
3 3
4 4
```

上面这个是单层循环的情况。下面这个例子，是一个双层循环的例子，每层循环都从 1 循环到 n，循环内部每次输出两个循环遍历的值：


 复制代码

```
1 #include <stdio.h>
2
3 int main() {
4     int n;
5     scanf("%d", &n);
6     for (int i = 1; i <= n; i++) {
7         for (int j = 1; j <= n; j++) {
8             printf("%d %d\n", i, j);
9         }
10 }
```

```
10     }  
11     return 0;  
12 }
```

当我们输入 3 的时候，程序的输出结果如下所示：

```
1  1  1  
2  1  2  
3  1  3  
4  2  1  
5  2  2  
6  2  3  
7  3  1  
8  3  2  
9  3  3
```

 复制代码

看了上面单层循环和双层循环的例子以后，如果让你改写成类似的三层循环的程序，想必这个你一定会做，无非就是在两层循环的内部，多加一层循环，然后 printf 输出的时候，输出的是三个变量的值即可。如果你可以自己理解到这个程序，那么你就可以理解今天这个任务。

今天这个任务，和上面的例子类似，但它不是实现一层循环的程序，也不是实现三层循环的程序，而是实现一个 k 层循环的程序。什么意思呢？就是 k 是一个读入参数，之后再读入一个参数 n，含义和上述程序中的 n 一致，而这个程序的输出结果，与上述例子中的输出结果类似，只不过每行输出 k 个数字。

简单来说，你要实现的是一个可变循环层数的程序。这下你清楚今天的任务了吧？那么我们正式开始学习吧。

必知必会，查缺补漏

理解了上面这个任务要做什么了，你可能还会发懵：为什么循环层数是可变的，代码结构不是确定性的么？别着急，今天我们将学习一个重要的编程技巧，那就是递归。

这里我要提醒一下，**递归是一种编程技巧**。你可能会在某些资料中，看到递归算法这种说法，其实这种说法是不合适的，因为明显的事实是，能够用循环实现的算法，都可以用递归

这种编程技巧实现。如果递归算作算法，那你听过循环算法一说么？所以，用一个编程技巧，给一类算法命名，实际是不合适的。

1. 温故知新：数学归纳法

你知道么，计算机的本质，是一个用来计算的工具，它最开始就是帮助我们完成一些现实世界里面的计算任务，并且完成的又快又好。那么现实世界的问题，是如何转换成可以在计算机中计算的任务呢？这个转换的过程中，都有哪些必不可少的东西呢？请看下图：

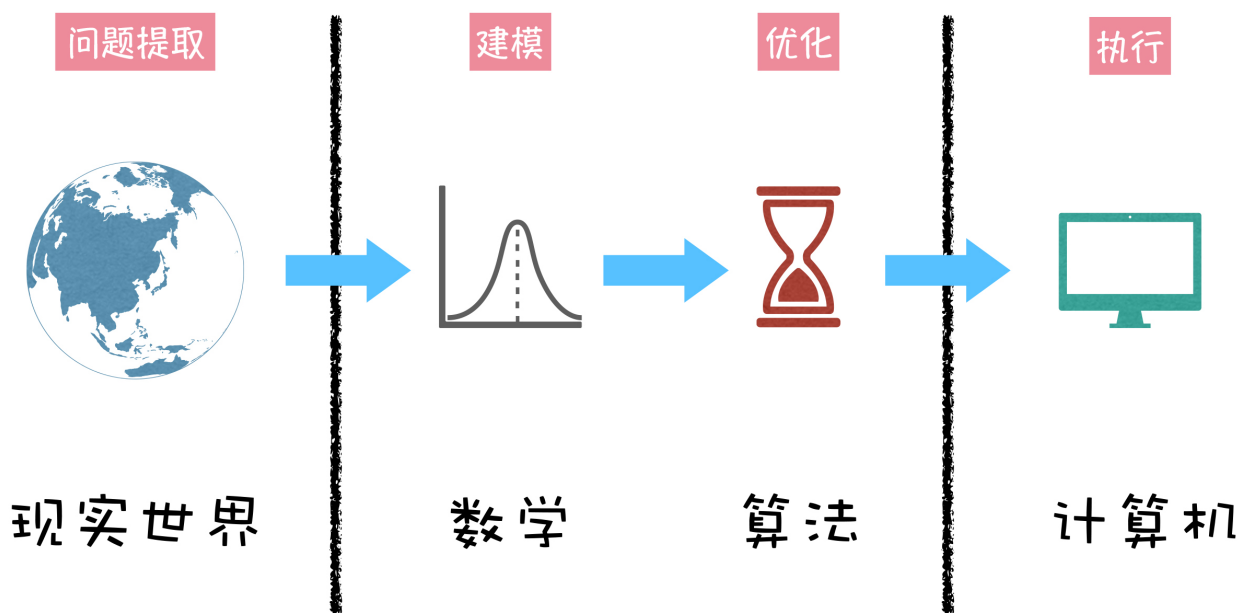


图1:从现实问题到可计算任务

在这幅图中，我们把转换过程分成四个部分：“现实世界” “数学” “算法” 和 “计算机”。这四个部分形成了一个路线，也就是从现实世界中的实际问题，到计算机中的可计算任务的过程。

我稍微来详细解释一下这幅图所表达的含义。首先我们来想想，如果没有数学，现实生活中我们会遇到什么困难？我会毫不夸张地告诉你，可能会面临生存危机。试想一下，因为没有数学，我们不会计算每日食物的消耗，无法合理分配资源，导致食物匮乏，引发生存危机。这也是为什么人类最早的文字记录，或者说是信息传递，用的是结绳记事，以“算术”的形式来解决现实世界问题。可以说，现实世界中的问题，本质是可以计算的，也就是说实际问题都可以做数学建模。

然后，我们说说算法。算法是将数学问题，转换到计算机中的计算任务的桥梁。因为计算机是依靠指令序列来执行的，而不同的指令序列代表了不同的效率，不同的效率在很多时候就意味着可行或者不可行。试想一个数学抽象出来的公式，需要计算机运算 1000 年才能得出结果，你认为这种任务可以放到计算机上面做么？答案显然是否定的。算法就是使得计算任务变得更高效，更可行。

至此，你就对我所说的内容，有个大致的体会了：计算机的核心是算法，算法的核心是数学。接下来呢，我们就需要介绍一种，可以指导我们进行程序设计的数学方法：数学归纳法。

高中的时候，我们就接触过数学归纳法，你可能已经对这个概念了然于胸，不过我们还是来回顾一下数学归纳法证明过程中重要的三步骤。

Step 1 : 验证边界条件 k_0 成立

Step 2 : 假设 k_i 成立，那么证明 k_{i+1} 也成立

Step 3 : 得出结论，所有 k_n 都成立

图2:数学归纳法的三个步骤

其实数学归纳法的三个步骤，总结起来就是，有一个已知正确的初始状态，然后证明如果前一个状态成立，那么后一个状态也成立（这一步主要在做过程正确性的证明），最后就是得出结论，在这个初识状态和转移过程的正确保证下，所有问题中的状态都成立。

举个例子，便于你更好地理解。假设我们要利用数学归纳法来证明：如果我推倒了第一块多米诺骨牌，那么所有的多米诺骨牌都会倒下。那么放到这三个步骤里，就是：

第一步，验证边界条件，第一块多米诺骨牌倒下了。

第二步，就是假设，第 n 块倒下了，根据多米诺骨牌的结构性质，那么如果存在 $n + 1$ 块，第 $n + 1$ 块也一定会倒下。

第三步，得出结论，只要第一块倒了，所有的多米诺骨牌都会倒下。

注意，上面说的这个是广义层面数学归纳法，这个过程对于循环过程的正确性证明，是非常有效的。

想一想，进入循环之前的程序中关键变量的值，就是上面所说的第一步中的 k_0 ；而每一次的循环，其实就是第二步中所要证明的那个上一个状态到下一个状态的过程。如果这两者都正确，我们就能很确信地知道，我们的整个循环过程就是正确的。

关于上面说的数学归纳法和循环程序之间的这一点联系，在日后的学习中，我还会详细地去举例说明，尤其是到了后续，我们学到了递推算法和动态规划算法的时候，会尤为明显。所以你要有足够的耐心和信心，咱们一起把这些问题搞懂。

2. 深入浅出：理解递归函数

放在编程的语境中，什么是递归呢？我这里先强调一句：递归是一种编程技巧。

你学完了函数以后，已经可以熟练地掌握在一个函数中，调用另外一个函数的方法了。可你有没有想过，如果在某个函数内部，调用自己同名函数过程，会发生什么？其实，和普通的函数调用过程一样，在具体执行过程中，只有等内部调用的函数执行完后，本层函数才会继续执行。

递归是一个过程，这个过程的每一步都类似，只是面对的问题规模不同。

下面我来举个例子：假如今年我上小学 5 年级，我现在想知道 1~5 年级的年级主任名字，但我现在只知道 5 年级的年级主任的名字，我可能会问一个 4 年级的学弟，希望他能告诉我 1~4 年级主任的姓名。

我这个学弟呢，也只知道他们年级主任的名字，那么我这个学弟就会问 3 年级学弟，问他 3 年级及以下的年级主任都有谁，依次类推，最后到了 1 年级的小学弟。

1 年级的小学弟，就会告诉 2 年级的学长自己年级主任的名字，2 年级的学长拿到 1 年级的年级主任的名字以后，会把 2 年级年级主任的名字填上去，然后再交给 3 年级的他学长□.....这样最终到我手里的就会是 1~4 年级的年级主任的所有名字，再加上我自己知道的 5 年级的年级主任姓名，这样，我就知道了全部信息。整个过程，如下图所示：

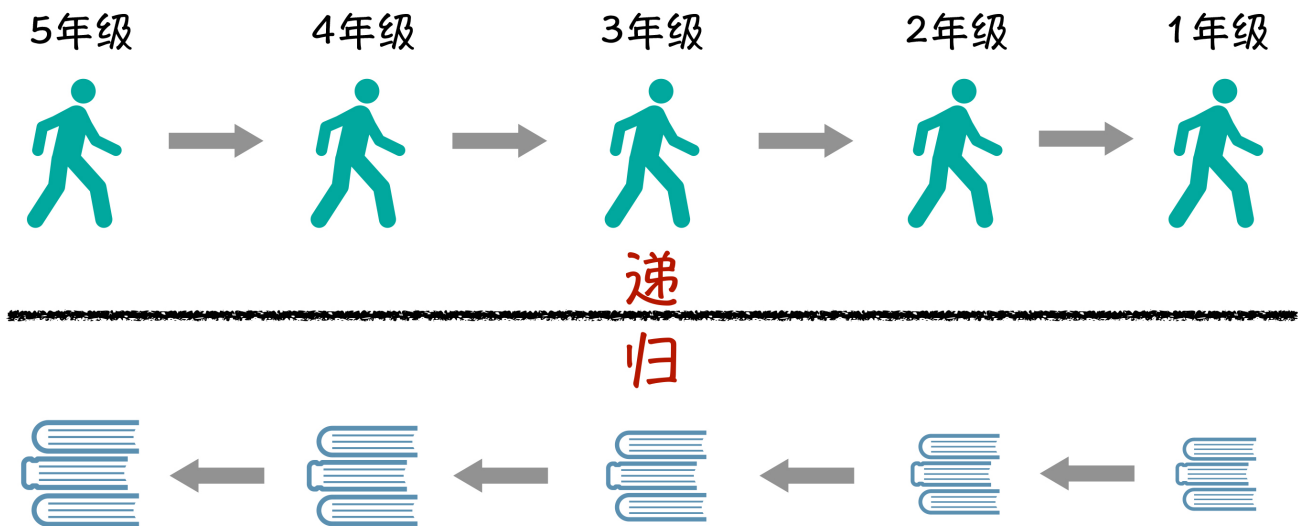



图3:年级主任问题示意图

在这个过程中，每个人问学弟的过程，就是我们所谓的“递”，而拿到学弟给的结果名单以后，再加上自己知道的结果反馈给自己学长的这个过程，就是“归”，整个过程就是我们所谓的“递归”。“递归”的过程，每一步的过程类似，可是问题规模不同。

接下来，我来举一个编程中的具体递归例子，看如下代码：

 复制代码

```
1 #include <stdio.h>
2
3 int f(int n) {
4     if (n == 1) return 1;
5     return f(n - 1) * n;
6 }
7
8 int main() {
9     int n;
10    scanf("%d", &n);
11    printf("%d\n", f(n));
12    return 0;
13 }
```

这段代码中，f 函数的作用，是计算 n 的阶乘的值，也就是从 1 乘到 n 的结果。在 f 函数内部，首先是一个边界条件，就是当 $n == 1$ 的时候，直接返回 1 的阶乘的结果。否则， n 的阶乘的结果，应该等于 $n - 1$ 阶乘的结果再乘上 n ，就得到了 n 的阶乘。在得到 $n - 1$ 阶乘结果的过程中，我们调用的不是别的函数，还是 f 函数本身，只不过传入的参数范围，是一个比 n 更小的范围 $n - 1$ 。

关于这个 f 函数，类比于上面年级主任的那个例子， $f(n)$ 就是我整理的信息， $f(n - 1)$ 就是比我要小 1 个年级的学弟所整理得到的信息，而 $n == 1$ 的边界条件判断，就是我那个最小的 1 年级的学弟。最后 $f(n - 1) * n$ 当中的 $* n$ 这个过程，就相当于每个人拿到了学弟整理的信息以后，再加上自己知道的信息，最后递交给自己的学长。

为什么这么做，能保证每个人所得到的信息都是正确的呢？在证明这个过程的时候，我们就需要用到前面提到的数学归纳法了。首先，我们知道 1 年级的学弟肯定能给出正确的信息，这就是数学归纳法中的边界条件。然后我们假设，如果上一个学弟，给出的信息是正确的，那么我所整理出来的信息，就一定是正确的，这就是数学归纳法中的证明过程的正确性。最终，我们就可以得到结论，在这个过程中，所有人获得的信息都是正确的，包括我自己。

其实，到了这里，我们也就得到了递归程序设计中的重要两部分：**边界条件**和**处理过程**。

所谓边界条件，就是当递归函数中的参数等于多少的时候，可以直接返回的条件。

处理过程呢，就是设计程序过程，处理递归调用的返回结果，根据递归调用的返回结果，得到本函数的结果。

这两部分，分别对应了数学归纳法中的两步，step1 和 step2。当这两步都可以保证正确，所涉及的递归函数程序，也绝对是正确的。

一起动手，搞事情

今天的思考题呢，是关于一段递归程序的：

 复制代码

```
1 #include <stdio.h>
2
3 int fib(int n) {
4     if (n == 1 || n == 2) return 1;
5     return fib(n - 1) + fib(n - 2);
6 }
7
8 int main() {
9     int n;
10    scanf("%d", &n);
11    printf("%d\n", fib(n));
12    return 0;
13 }
```


上面这段程序中，fib 函数是求菲波那契数列第 n 项值的函数。菲波那契数列的定义如下：

$$f(n) = \begin{cases} 1 & (n = 1, 2) \\ f(n - 1) + f(n - 2) & (n = other) \end{cases}$$

图4:斐波那契数列

根据如上内容，你需要完成两个小的思考题：


1. 请将上述菲波那契数列求解的程序从递归程序，改成循环程序。
2. 请将上述递归程序的代码和数学归纳法中的步骤做一一对应，留在留言区中。

完成不定层数的循环程序

准备完了基础知识以后，让我们回到今天的任务，完成一个可变循环层数的程序。我们可以一开始假设，有一个函数，是实现 5 层循环打印的程序，那么它会循环 n 次，每次调用一个实现 4 层循环打印的程序。

依照这个大体的思路，我们就可以写出如下代码框架：

```
1 int print_loop(int k, int n) {
2     if (k == 0) {
3         // 打印一行
4     }
5     for (int i = 1; i <= n; i++) {
6         print_loop(k - 1, n);
7     }
8     return ;
9 }
```

 复制代码

在这个代码框架中，我们先来看递归的过程，print_loop(k, n) 代表 k 层循环的程序，然后循环 n 次，每次调用一个 k - 1 层循环的程序。而递归的边界条件就是当 k == 0 的时候，

就是所谓的 0 层循环，也就是程序打印一行具体内容的地方，可打印的这行内容究竟是什么呢？

你会发现，要打印的这行内容，与每层循环遍历到的数字有关系，那么我们就需要记录每层循环遍历到的数字。这个信息，我们可以记录在一个数组中，数组中存储的，就是当前要打印这行的每一个数字。基于上述代码框架，我们就可以得到下面这个更完善的代码：

 复制代码

```
1  int arr[100];
2  int print_loop(int k, int n, int total_k) {
3      if (k == 0) {
4          for (int i = total_k; i >= 1; i--) {
5              if (i != total_k) printf(" ");
6              printf("%d", arr[i]);
7          }
8          printf("\n");
9      }
10     for (int i = 1; i <= n; i++) {
11         arr[k] = i;
12         print_loop(k - 1, n, total_k);
13     }
14     return ;
15 }
```

正如你看到的，我们把每一层循环的值，放到了一个 arr 数组中，第 k 层循环变量的值，存储到 arr[k] 的位置。而在上述代码中，多了一个递归参数，就是 total_k，代表了一共有多少层循环，这个参数是为了方便我们最后确定循环输出的上界。至此，我们就完成了今天的任务。

课程小结

今天的重点，一个关于数学归纳法，一个关于递归，需要你记住如下两点：

1. 数学归纳法中重要的两部分，一是要边界条件成立，二是证明转移过程成立。
2. 程序设计最重要的是正确性，递归函数的正确性可以利用数学归纳法来保证。

关于数学归纳法和递归函数的设计，还需要你在日后不断的加以练习。注意总结两者的联系，能够使得你在接下来的学习中事半功倍。

好了，今天就讲到这里，我是胡光，我们下期见。

课程学习计划

关注极客时间服务号 每日学习签到

月领 25+ 极客币

【点击】保存图片，打开【微信】扫码>>>



© 版权归极客邦科技所有，未经许可不得传播售卖。页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 期中测试 | 给语言基础篇交一份满分答卷，去迎接新的挑战！

下一篇 13 | 程序设计原则：把计算过程交给计算机

精选留言 (10)

写留言



Geek_Andy_Lee00

2020-02-13

斐波那契循环部分：

①

```
int fib(int n) {  
    int f1 = 0, f2 = 1;  
    int i, res;...
```

展开 ∨

作者回复: 你应该是学过 python 吧，C 是不支持这种 `f1, f2 = f2, f1 + f2` 这种语法的。这段代码，在 C 语言中，会被认为是一个逗号表达式，其中包括三个独立的语句，第一个是 `f1`、第二个是 `f2 = f2`，第三个是 `f1 + f2`。

**胖胖胖**

2020-02-18

老师，能不能把 $k=0$ 的时候每次输出的一行数认为是 $k=1$ 的循环的一个元素，把 $k=1$ 认为是边界条件呀，这样理解可以吗

展开

作者回复: 可以的。

**胖胖胖**

2020-02-17

感觉理解示例代码得关键在于理解“递归的边界条件就是当 $k=0$ 的时候，就是所谓的0层循环，也就是程序打印一行具体内容的地方”这句话，看了半个小时才反应过来。。。

作者回复: 我想换一种表述。

**奔跑的八戒**

2020-02-13

```
int f1, f2 ;  
f1 = f2 = 1 ;  
for(int i = 3; i < n; i++){  
    f2 = f1 + f2;  
    f1 = f2 - f1;...
```

展开

作者回复: 完美!

**一步**

2020-02-12

斐波那契循环方法

```
int fib(n) {  
    int first = 1;
```

```
int second = 1;
for(int i = 3; i < n; i++) {...
```

展开 ▾

作者回复: 我猜, 你这段代码, 绝对没有经过测试。有一点儿小瑕疵, 但是逻辑是正确的。



徐洲更

2020-02-12

不知道这样子算不算只用了f1和f2

```
int fib(int n){
    if ( n < 2) return 1;
    int f1=1;
    int f2=1;...
```

展开 ▾

作者回复: 不算哦~~~~其中有一个 tmp 变量是额外使用的, 所以, 这段程序还是可以优化的。



我思故我在

2020-02-11

斐波那契改为循环结构:

主要代码:

```
int f1,f2;
f1 = f2 = 1;
```

...

展开 ▾

作者回复: 不错! 试试能不能只使用f1和 f2



我思故我在

2020-02-11

日常做事, 都说结果不重要, 过程才是最重要的, 当然只结果也很重要。而递归往往是结果重要, 过程不重要。当你去试图理清过程的时候, 往往会陷入死循环之中。理解递归, 只要搞清楚头和尾就行了, 中间细节只要逻辑正确, 就别去care了。

展开 ▾

作者回复: d(^_^o)



柒~龟虽寿!

2020-02-11

```
#include <stdio.h>
int arr[100];
int print_loop(int k, int n, int total_k) {
    if (k == 0) {
        for (int i = total_k; i >= 1; i--) {...
```

展开 ▾

作者回复: 递归的边界条件里面，没有return语句



Bradley

2020-02-11

递归是非常重要的，越基础的概念越理解不到位。

展开 ▾

作者回复: 希望能使你对递归，有一个全新的了解。 ^_^

