

2025-3-22 日 递归基础

在函数内部，可以调用其他函数。如果一个函数在内部调用自己本身，这个函数就是递归函数。

如果一个函数在调用自己，则被调用的“自己”再调用“自己”，想象一下，这将会无限循环下去，所以递归函数中肯定有办法在某种情况下停止调用自己，即停止递归，一般来说使用 语句实现在满足特定的条件下终止递归。

函数不在递归地情况称作基本情形（base case, 也称基本情况）。

函数调用自身来执行子任务的情况就称作递归情形（recursive case）。

递归通常把一个大型复杂的问题层层转化为一个与原问题相似的规模较小的问题来求解，递归策略只需少量的程序就可描述出解题过程所需要的多次重复计算，大大的减少程序的代码量。

练习一：上台阶 easy

现在小瓜想通过台阶走上平台，最底层（小瓜所在的层）编号为 1，最顶层编号为 n 。由于小瓜的腿比较短，他一次只能向上走 1 级或者 2 级台阶。小瓜想知道他有多少种方法走上平台，你能帮帮他吗？

输入格式

一个整数 n ，其中 $2 \leq n \leq 25$ 。

输出格式

一行一个整数，表示小瓜上台阶的方案数

参考程序：

```
#include<bits/stdc++.h>
using namespace std;
int solve(int n){
    if(n==1||n==2) return 1;
    else return solve(n-1)+solve(n-2);
}
int main(){
    int n;
    cin>>n;
    cout<<solve(n);
    return 0;
}
```

练习二：上台阶 v4

小明的面前有一个 n 级的台阶，他每次可以上一级或是两级，现在他想知道每一种上台阶的方法，请你告诉他。

具体的，请你求出有多少种上台阶的方法；对于每一种上台阶的方法，请你用数字

表示每一步的台阶数，并将不同方法按字典序输出。

例如， $n=3$ ，可行方法为 111，12，21（3 种），其中 12 表示第一步 1 个台阶，第二步 2 个台阶，因此按字典序输出：

```
3
111
12
21
```

输入格式

输入一个正整数 n ，表示台阶数量。

输出格式

第一行输出一个数 m ，表示上台阶的方法数。

之后 m 行每行输出一个由“1”、“2”组成的字符串，表示一种上台阶的方法。

参考程序：

```
#include<bits/stdc++.h>
using namespace std;
string paths[11000];//字符串数组，每一个元素都是字符串
int cnt=0;
void solve(int n,string path){//有 n 阶要爬
    if(n==0) paths[cnt++]=path;//爬完了
    else if(n>0){
        solve(n-1,path+"1");//我上了一阶，还有 n-1 阶
        solve(n-2,path+"2");//我上了 2 阶，还有 n-1 阶
    }
}

int main(){
    int n;
    cin>>n;
    solve(n,"");
    cout<<cnt<<endl;
    for(int i=0;i<cnt;i++) cout<<paths[i]<<endl;
    return 0;
}
```

练习三：小华与布丁

小华非常喜欢吃布丁。这一天她打算购买原味和抹茶味两种布丁，共 n 个，之后按照一定顺序将 n 个布丁全部吃完。

然而如果小华连续吃到三个同口味的布丁，她会感到厌烦而无法继续吃。请你帮助小华规划吃布丁的每一种可行的口味顺序，使她能够全部吃完而不产生厌烦感。

输入格式

输入一个数 n，表示布丁的数量。

输出格式

每行输出一个由'0'和'1'组成的字符串,表示一种可行的吃布丁的顺序。其中0表示原味,1表示抹茶味。请将可行的每种方案按字典序升序输入。

输入样例

3

输出样例

001
010
011
100
101
110

参考程序：

```
#include<bits/stdc++.h>
using namespace std;
int n;
void solve(int idx,string path){
    if(idx==n) {
        cout<<path<<endl;
        return;
    }
    //吃的布丁个数小于 2、前 2 个吃的布丁中至少有一个不是原味
    if(idx<2||path[idx-1]!='0'||path[idx-2]!='0')
        //满足以上情况 这次就吃原味
        solve(idx+1,path+"0");
    //吃的布丁个数小于 2、前 2 个吃的布丁中至少有一个不是抹茶
    if(idx<2||path[idx-1]!='1'||path[idx-2]!='1')
        //满足以上情况 这次就吃抹茶
        solve(idx+1,path+"1");
}

int main(){
    cin>>n;
    //一个布丁也没吃
    solve(0,"");
    return 0;
}
```