递归算法

有 5 个人坐在一起，问第 5 个人多少岁，他说比第 4 个人大 2 岁；问第 4 个人多少岁，他说比第 3 个人大 2 岁；问第 3 个人多少岁，他说比第 2 个人大 2 岁； 问第 2 个人多少岁，他说比第 1 个人大 2 岁，最后问第 1 个人多少岁，他说是 10 岁。请问第 5 个人多少岁？

6.2-0

```
#include<iostream>// 函数的递归（函数调用自身）选自《编程竞赛宝典》
using namespace std;
int age(int n)
{
	if(n==1)
		return 10;
	else
		return age(n-1)+2;// 调用自身
}
int main()
{
	cout<<age(5);
	return 0;
}
```

1846. 阿尔法乘积

计算一个整数的阿尔法乘积。对于一个整数 x 来说，它的阿尔法乘积是这样来计算的：如果 x 是一个个位数，那么它的阿尔法乘积就是它本身；否则的话，x 的阿尔法乘积就等于它的各位非 0 的数字相乘所得到的那个整数的阿尔法乘积。

例如：4018224312 的阿尔法乘积等于 8 ，它是按照以下的步骤来计算的：4018224312 → 4×1×8×2×2×4×3×1×2 → 3072 → 3×7×2 → 42 → 4×2 → 8；编写一个程序，输入一个正整数（该整数的值在 int 范围内），输出它的阿尔法乘积。

输入：输入只有一行，即一个正整数。

输出：输出相应的阿尔法乘积。

输入	输出
3072	8

```
#include<bits/stdc++.h>//1846
using namespace std;
int alpha(int a){
	int t=1;
	while(a!=0)		// 计算整数非 0 位的乘积
	{
		if(a%10!=0)			// 判断尾数不等于 0 就累乘
			t = t*(a%10);
		a = a/10;
	}
	a = t;
	if (a>=10) return alpha(a);
	// 当这个数各个位的乘积不是一位数时，对它的乘积继续求解
	else return a;
}
int main() {
	int x;
	cin>>x;
	cout<<alpha(x);
	return 0;
}
```

土地分割

把一块 M*N 米的土地分割成同样大的正方形，要求 1：没有土地剩余，2：分割出的正方形土地最大。 没有剩余就是说 M、N 都能被分割出来的正方形边长整除； 分割出来的土地最大，也就是说在可以整除 M、N 的边长中找最大的那个数 ； 结论：求 M、N 的最大公约数。 利用欧几里得算法轻松实现：（注意数据范围）

```
#include <iostream>// 东方博宜 1335    anselxu
using namespace std;
long long gcd(long long x,long long y)// 辗转相除法求最大公约数
{
	if(y==0)	return x;	// 除数为 0，输出 x
	return	gcd(y,x%y); // 否则，除数为被除数，余数为除数继续调用 gcd
}
int main()
{
	long long n,m;
	cin>>n>>m;
	cout<<gcd(n,m);
	return 0;
}
```

```
#include <bits/stdc++.h>
using namespace std;
long long fun(long x,long long y)// 求 x 和 y 的最大公约数
{
	if(x % y != 0)	return fun(y , x % y);
	else		return y;
}
long long a,b;
int main()
{
	cin>>a>>b;
	cout<<fun(a,b);
	return 0;
}
```

1695. 阿克曼（Ackmann）问题描述

阿克曼（ Ackmann ）函数 A（ m，n ）中， m，n 定义域是非负整数，函数值定义为：

$$Ack(m,n)=\begin{cases} n+1 & m=0 \\ Ack(m-1,1) & m\neq 0, n=0 \\ Ack(m-1, Ack(m, n-1)) & m\neq 0, n\neq 0 \end{cases}$$

写出计算 Ack(m，n) 的递归算法程序。

输入：两个非负整数 m 和 n 。

输出：阿克曼函数 A(m,n) 的值。测试数据保证结果不超过 int 范围，直接用递归不超时。（提示：阿克曼函数的值增长速度非常高，仅是对于 A(4,2) 的输出就有 19729 位，而 A(4,3) 则即使是位数也不易估计。）

样例

输入：2 3

输出：9

```
#include <bits/stdc++.h>//3-1695  hyb
using namespace std;
int ack(int m,int n)
{
	if(m == 0) return n + 1;
	else if(m != 0 && n == 0) return ack(m-1,1);
	else if(m != 0 && n != 0) return ack(m-1,ack(m,n-1));
}

int main()
{
	int m,n;
	cin>>m>>n;
	cout<<ack(m,n);
	return 0;
}
```

经典递归问题——汉诺塔

A 为存放盘子的塔，C 为目标塔，B 为辅助塔

```
#include <bits/stdc++.h>// 东方博宜 1222  javacn
using namespace std;
/*
    移动的过程：
    1、先将 n-1 个金片（n 个金片除了最下面一个以外的金片）
       从 A 位置，借助 C 位置，移动到 B 位置，需要 fun(n-1) 步
    2、将 A 位置的最下方的一个金片，直接移动到 C 位置，需要 1 步
    3、将 B 位置的 n-1 个金片，从 B 位置借助 A 位置移动到 C 位置
       需要 fun(n-1) 步
    因此得到结论如下：fun(n) = fun(n-1) * 2 + 1
  函数的作用：将 n 个金片从 p1 位置，借助 p2 位置，移动到 p3 位置
  p1：源位置（金片所在的源位置）
  p2：辅助位置（移动的过程中要借助的位置）
  p3：目标位置（移动到的位置）
*/
void move(int n,char p1,char p2,char p3){
      // 递归的出口：只要有金片就要递归，没有金片递归停止
      if(n > 0){
            // 第 1 步：将 n-1 个金片，从 p1 位置，借助 p3 位置，移动到 p2 位置
            move(n-1,p1,p3,p2);
            // 第 2 步：将 p1 位置的第 n 个金片直接移动到 p3 位置
            cout<<p1<<" To "<<p3<<endl;
            // 第 3 步：将 p2 位置的 n-1 个金片，借助于 p1 位置，移动到 p3 位置
            move(n-1,p2,p1,p3);
      }
}
int main(){
      int n;
      cin>>n;
      // 递归调用，打印移动的步骤
      move(n,'A','B','C');
      return 0;
}
```

```cpp
#include<bits/stdc++.h>// 东方博宜 1208-1  螺旋方阵 递归，深搜 jiangyf70
using namespace std;
int q[20][20];
int dx[] ={0, 1, 0, -1};
int dy[] ={1, 0, -1, 0};
int n, i = 0;
void fun(int x, int y, int k)
{
	if(x > 0 && x <= n && y > 0 && y <= n && q[x][y] == 0)
	{                                //不越界，并且数值为0
		q[x][y] = k;
		int tx, ty;
		tx = x + dx[i], ty = y + dy[i];
		if(tx < 1 || tx > n || ty < 1 || ty > n || q[tx][ty])
		{                    //越界或者 q[tx][ty] = true（数值不为0）
			i = (i + 1) % 4;//拐弯
			tx = x + dx[i], ty = y + dy[i];
		}
		fun(tx, ty, k+1);
	}
}
int main()
{
	cin >> n;
	fun(1, 1, 1);
	for(int i = 1; i <= n; i++)
	{
		for(int j = 1; j <= n; j++)
			cout << setw(3) << q[i][j];
		cout << endl;
	}
	return 0;
}
```

输入

5

输出

```
  1  2  3  4  5
 16 17 18 19  6
 15 24 25 20  7
 14 23 22 21  8
 13 12 11 10  9
```

```cpp
#include<bits/stdc++.h>//1208-2    递归 jiangyf70 仅供参考
using namespace std;
int q[20][20];
int dx[] ={0, 1, 0, -1};
int dy[] ={1, 0, -1, 0};
int n;
void lxfz(int a, int b, int c)
{
	if(b > 0)
	{
		int x = a , y = a, k = 0;
		q[x][y] = c++;
		for(int i = 0; i < 4; i++)// 四个方向
		{
			for(int j = 0; j < b - 1; j++ )
			{
				int x1 = x + dx[i], y1 = y + dy[i];
				if(q[x1][y1] == 0)
				{    x = x1, y = y1;    q[x][y] = c++;      }
			}
		}
		lxfz(a + 1, b - 2, c);
	}
}
int main(){
	cin >> n;
	lxfz(1, n, 1);
	for(int i = 1; i <= n; i++)
	{
		for(int j = 1; j <= n; j++)
			cout << setw(3) << q[i][j];
		cout << endl;
	}
	return 0;
}
```

```cpp
#include<bits/stdc++.h>// 东方博宜 1209-1 回形方阵 递归 jiangyf70
using namespace std;
int q[30][30];
int dx[] ={0, 1, 0, -1};
int dy[] ={1, 0, -1, 0};
void hxfz(int a, int n) // 在 a 位置填充 n
{
    if(n >= 0)
    {
        int x = a , y = a;
        q[x][y] = n;
        for(int i = 0; i < 4; i++)
        {
            for(int j = 0; j < 2 * n; j++ )
            {
                x = x + dx[i], y = y + dy[i];
                q[x][y] = n;
            }
        }
        hxfz(a + 1, n - 1);
    }
}
int main()
{
    int n;
    cin >> n;
    hxfz(1, n);
    for(int i = 1; i <= 2 * n + 1; i++)
    {
        for(int j = 1; j <= 2 * n + 1; j++)
            cout << setw(2) << q[i][j];
        cout << endl;
    }
    return 0;
}
```

输入

8

输出

```
8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8
8 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 8
8 7 6 6 6 6 6 6 6 6 6 6 6 6 6 7 8
8 7 6 5 5 5 5 5 5 5 5 5 5 5 6 7 8
8 7 6 5 4 4 4 4 4 4 4 4 4 5 6 7 8
8 7 6 5 4 3 3 3 3 3 3 3 4 5 6 7 8
8 7 6 5 4 3 2 2 2 2 2 3 4 5 6 7 8
8 7 6 5 4 3 2 1 1 1 2 3 4 5 6 7 8
8 7 6 5 4 3 2 1 0 1 2 3 4 5 6 7 8
8 7 6 5 4 3 2 1 1 1 2 3 4 5 6 7 8
8 7 6 5 4 3 2 2 2 2 2 3 4 5 6 7 8
8 7 6 5 4 3 3 3 3 3 3 3 4 5 6 7 8
8 7 6 5 4 4 4 4 4 4 4 4 4 5 6 7 8
8 7 6 5 5 5 5 5 5 5 5 5 5 5 6 7 8
8 7 6 6 6 6 6 6 6 6 6 6 6 6 6 7 8
8 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 8
8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8
```

```cpp
#include <bits/stdc++.h>// 东方博宜 1209-2  回形方阵  递归  javacn
using namespace std;
int a[20][20], n;
void fun(int start, int len, int x) // 为二维数组的第 x 圈赋值
{//start：起始点的坐标 //len：赋值的宽度 //x：起始值
	if(len > 0)		// 递归出口
	{
		int i, j;
		for(j = start; j <= start + len - 1; j++) // 循环第 1 行的列（向右）
			a[start][j] = x;

		for(i = start + 1; i <= start + len - 1; i++) // 循环向下，循环行
			a[i][start + len - 1] = x;

		for(j = start + len - 2; j >= start; j--) // 循环向左，循环列
			a[start + len - 1][j] = x;

		for(i = start + len - 2; i >= start + 1; i--)	// 循环向上，循环行
			a[i][start] = x;

		fun(start + 1, len - 2, x - 1);		// 递归为 start+1 这一圈赋值
	}
}
int main() {
	cin >> n;
	fun(1, n * 2 + 1, n);	// 为从 1,1 点开始的一圈赋值，边长为 n * 2 + 1
	n = n * 2 + 1;	// 输出
	int i, j;
	for(i = 1; i <= n; i++) {
		for(j = 1; j <= n; j++)
			cout << setw(2) << a[i][j];
		cout << endl;
	}
	return 0;
}
```