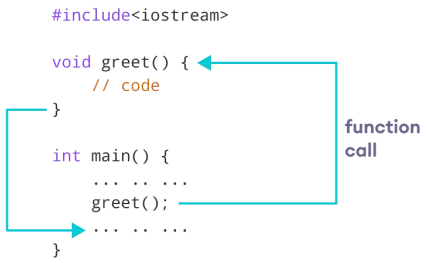
一、函数与递归（Function & Recursion）

（一）函数定义与调用、形参与实参

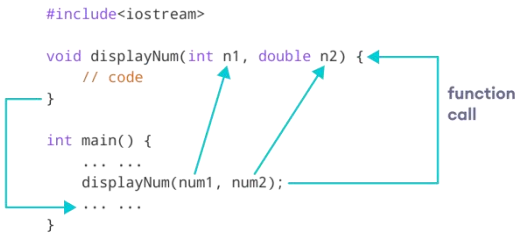
函数（Function）是一个可以重复使用的小程序，用来完成一个特定的任务。

- **函数定义**：就像写一段指令说明书，告诉它该怎么做。
- **函数调用**：就是我们在程序里“喊它出来干活”。
- **形参**（形式的参数）：定义函数时写的“占位符”。
- **实参**（实际的参数）：调用函数时真正传进去的数据。



函数就像厨房里的“榨汁机”：

- **定义**：告诉榨汁机怎么榨汁（榨几秒，加不加水）
- **形参**：比如“水果”这个空位
- **实参**：你真的放进去一个“苹果”或“香蕉”



C++ 示例代码：

```
#include <iostream>
using namespace std;
// 定义函数：sayHello
void sayHello(string name) {
    cout << "你好，" << name << "！" << endl;
}
int main() {
    sayHello("小明"); // 调用函数，实参是“小明”
    return 0;
}
```

- string name 是 形参
- "小明" 是 实参

考试易错点：

易错点	正确写法说明
忘记写返回类型	函数前必须写 void、int 等
忘记写形参	定义函数时必须写 ()
函数没调用就不会执行	定义 ≠ 执行，调用才会生效

（二）传值参数 vs 传引用参数

- 传值参数（Pass by Value）：函数拿到的是**值的复制品**，外面的变量不受影响。
- 传引用参数（Pass by Reference）：函数直接操作**原变量本体**，外面的变量会改变。

示例 1：传值参数（值不会变）

```
void addTen(int x) {
    x = x + 10;
    cout << "函数里 x 是：" << x << endl;
}
int main() {
    int a = 5;
    addTen(a);
    cout << "外面 a 是：" << a << endl; // a 仍然是 5
}
```

示例 2：传引用参数（值会变）

```
void addTen(int &x) { // 加了 &
    x = x + 10;
}
int main() {
```

```
    int a = 5;
    addTen(a);
    cout << "现在 a 是：" << a << endl; // a 变成 15
}
```

考试易错点：

易错点	正确方式说明
忘记加 & 就以为是引用	只有 int &x 才是引用
改错变量	外部变量名与形参不必相同，但容易混
以为值会变，其实没变	要根据是否引用判断是否会改变

（三）常量与变量的作用范围（Scope）

☑什么是“作用范围”？

作用范围（Scope）就是“这个变量在哪些地方能看到、能使用”。

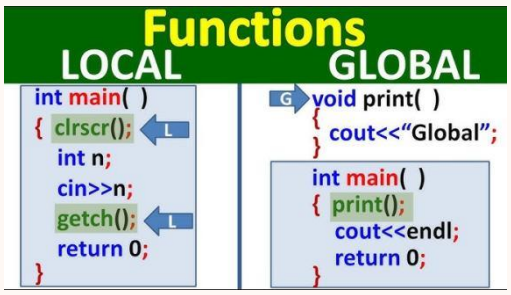
就像一个玩具放在你房间，只能你用；放在客厅，全家人都能用。

分类：三种作用范围

类型	英文名	解释
局部变量	Local Variable	在函数或代码块里定义，只能在这段代码里使用。
全局变量	Global Variable	在所有函数外面定义的变量，**整个程序**都能用。
静态变量	Static Variable	在函数内部，但**值会被记住，不会每次都清零。**

C++ 示例 1：局部变量（Local）

```
void showAge() {
    int age = 10; // 局部变量
    cout << age << endl;
}
```



```
int main() {
    showAge();
    // cout << age; // ❌ 报错：age 在 main 里没定义
}
```

C++ 示例 2：全局变量（Global）

```
int score = 100; // 全局变量
void showScore() {
    cout << score << endl;
}
int main() {
    showScore();   // 输出 100
    cout << score << endl; // 也能访问
}
```

C++ 示例 3：静态变量（Static）

```
void counter() {
    static int count = 0;
    count++;
    cout << "这是第 " << count << " 次调用" << endl;
}
int main() {
    counter(); // 第 1 次
    counter(); // 第 2 次
    counter(); // 第 3 次
}
```

为什么用 static?

如果没有 static，每次进函数 count 都会重新变成 0。

考试易错点：

易错点	正确说明
在函数外用了局部变量	局部变量只能在自己那一对 {} 里用
忘记加 static	变量会重置，从头开始数
多个函数名重复变量	虽然变量名一样，但各自是“局部的”，互不干扰

（四）递归函数（Recursive Function）

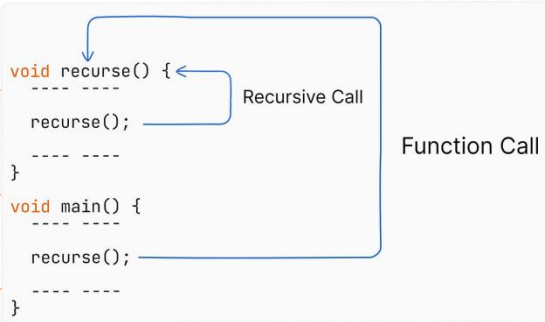
☑什么是递归函数?

递归（Recursion） 就是 “**一个函数自己调用自己**”，像照镜子，镜子里还有镜子，一层套一层。

它解决的问题通常是**一件事可以分成重复的小事**，例如数学题、楼梯、汉诺塔……

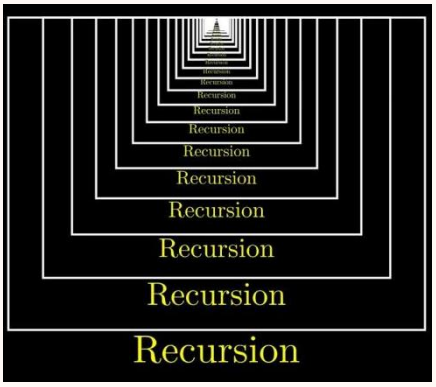
常见格式：

```
return 条件 ? 终止值 : 自己调用自己;
```



或者更完整地写：

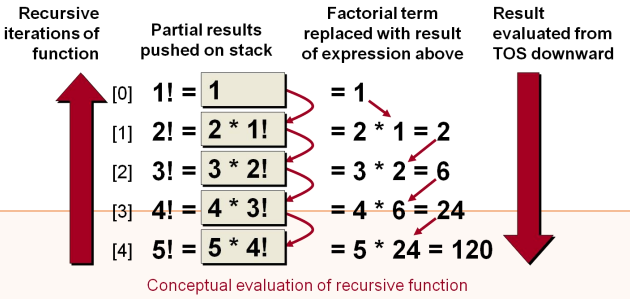
```
函数名(参数) {
    if (终止条件)
        return 某值;
    else
        return 函数名(变更后的参数);
}
```



示例 1：阶乘函数 factorial(n)

阶乘是：

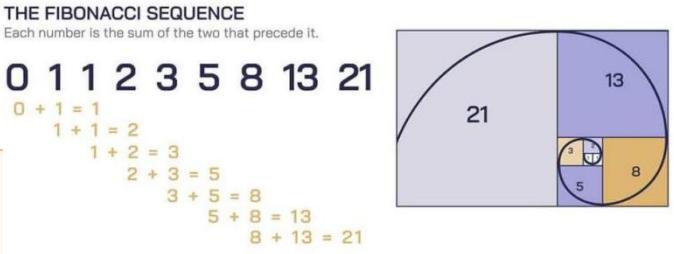
- 3! = 3 × 2 × 1 = 6
- 5! = 5 × 4 × 3 × 2 × 1 = 120



Conceptual evaluation of recursive function

```
int factorial(int n) {
    if (n == 1) return 1; // 终止条件
    return n * factorial(n - 1); // 自己调用自己
}
int main() {
    cout << factorial(5); // 输出 120
}
```

示例 2：斐波那契数列（Fibonacci）



数列如下：1, 1, 2, 3, 5, 8, 13……

```
int fib(int n) {
    if (n == 1 || n == 2) return 1;
    return fib(n - 1) + fib(n - 2);
}
```

递归就像你玩套娃娃时，一个娃娃里有另一个娃娃，打开后还有一个娃娃……直到最小的娃娃不能再打开为止，这就像递归！

考试时注意：

易错点	正确做法说明
没写终止条件	会进入无限调用，程序会崩溃
自己调用自己写错了	函数名要和定义的一致
返回值类型忘写 return	如果函数是 int 类型，必须要 return
变量范围冲突	内部变量名重复注意不要搞混