函数

把重复的逻辑封装起来

HGNU-ACM 实验室算法组

基础精讲 30 余页；精讲 15 道典型题；附本章 35 道练习索引





本章目录 | 前半

为什么要写函数

函数的定义、声明与调用

形参、实参与值传递

返回值与 return

局部变量与作用域

本章目录 | 后半

函数的复用与拆分

递归函数入门

常见错误与调试

本章练习索引

先看一段没有函数的代码

统计 $2 \sim n$ 中的素数，主程序里直接写判断：

```
int cnt = 0;
for (int i = 2; i <= n; ++i) {
    bool ok = true;
    for (int d = 2; d * d <= i; ++d) {
        if (i % d == 0) {
            ok = false; // 找到因子，不是素数
            break;
        }
    }
    if (ok) ++cnt;
}
```

这段代码的三个问题

- 这段代码回答了一个小问题：i 是不是素数。
- 下一题若还要判断绝对素数、哥德巴赫，同样的循环得再抄一遍。
- 三份代码很难同步：改了一处，容易漏掉另外两处。

本页要点

函数的起点：把“判断素数”这段逻辑从主程序里拿出来，给它起一个名字。

函数：给一段逻辑起名字



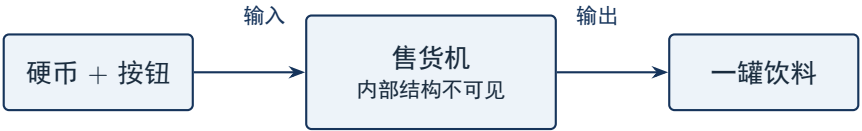
内部细节对调用者隐藏

- 调用者只需要知道三件事：传什么进去、得到什么、函数叫什么。
- 函数体内部可以独立修改，只要“输入—输出”的约定不变。
- 这就是封装：把细节藏起来，只暴露稳定的接口。

本页要点

`isPrime(17)` 读作：把 17 交给 `isPrime`，取回一个 `bool`。

函数像一台加工机



你不需要知道弹簧怎么转

对应关系

投币与按钮 → 实参；机器内部 → 函数体；掉出的饮料 → 返回值。

- 机器可以反复使用，每次投不同的币、选不同的按钮。
- 只要接口不变，机器内部换零件，使用者不用改习惯。

本页要点

函数就是程序里的“加工机”：给原料，出成品。

函数带来的三件事

1. 复用：一次编写，多处调用。判断素数、判断回文都能反复使用。
2. 拆分：主程序变成“读入—处理—输出”的短流程，每步一个函数。
3. 定位：结果不对时，先判断是哪个函数出错，再单独测试它。

对比

没有函数：主程序几十行，逻辑缠在一起。
有函数：主程序像目录，细节在各函数里；改一个功能只动一个地方。

- 函数不是“高级写法”，而是把复杂问题变小、让代码可读的基本工具。
- 本章先学语法，再学怎样把一道题拆成几个函数。

函数的四要素

```
返回类型 函数名(参数表) {
    函数体;
    return 结果; // 非 void 函数
}
```

- 返回类型：int、double、bool、void 等。
- 函数名：见名知意，如 isPrime、sumProper。
- 参数表：每个参数写“类型名字”，逗号分隔；可以为空。
- 函数体：完成工作的语句；return 把结果交回调用者。

本页要点

定义函数就是写一份契约：给我什么，我还什么。

最小的函数：求平方

```
int square(int x) {
    return x * x; // 结果交回调用者
}
int main() {
    int y = square(5); // 5 是实参
    cout << y << '\n'; // 输出 25
    return 0;
}
```

- `int` 是返回类型：调用一次，得到一个整数。
- `square` 是函数名；`int x` 是形参，调用时新建。
- `return x * x;` 既算出结果，又立即结束函数。

本页要点

先写最小的函数并跑通，再逐步加参数、加分支。

定义、声明、调用各是什么

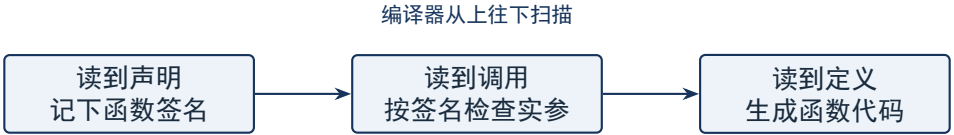
角色	写法	作用
定义	`int f(int x) { 函数体 }`	写完整实现，生成代码
声明	`int f(int x);`	只写签名，末尾加分号
调用	`f(3)`	让函数执行一次，得到一个值

- 定义只能有一份；声明可以写很多份。
- 调用处必须已经见过声明或定义，否则编译器不认识这个名字。
- 声明中的参数名可以省略：`int f(int);` 与 `int f(int x);` 等价。

本页要点

定义负责“怎么做”，声明负责“怎么用”，调用负责“现在做一次”。

为什么需要声明：编译器从上往下读



- 调用写在定义前面时，编译器还没见过这个名字，会报“未声明”。
- 先在前面写一句声明，编译器就能检查参数个数和类型。
- 大型程序把声明集中放进头文件，各个源文件都能共用。

本页要点

声明是给编译器的“预告”：这个名字待会儿会出现，先按这个签名检查。

声明与定义分开写

```cpp
bool isPrime(int x); // 声明：只写签名，末尾有分号

int main() {
    cout << isPrime(17) << '\n'; // 调用：编译器查声明
    return 0;
}

bool isPrime(int x) { // 定义：这里才真正实现
    if (x < 2) return false;
    for (int d = 2; d * d <= x; ++d)
        if (x % d == 0) return false;
    return true;
}
```

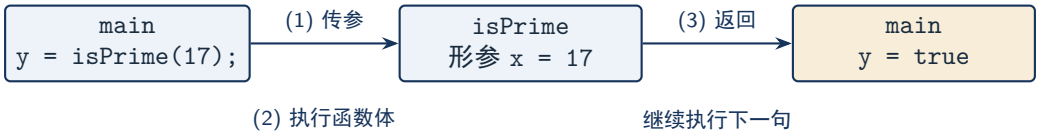
- 定义写在 main 前面时，声明可以省掉。
- 声明和定义的签名必须完全一致，否则链接阶段报错。

调用可以出现在哪些地方

```cpp
int square(int x) {
    return x * x;
}
int main() {
    int a = square(3); // 1. 初始化变量
    cout << square(4) << '\n'; // 2. 作为输出项
    if (square(5) > 20) { // 3. 作为条件
        a = square(6) + square(7); // 4. 参与算术运算
    }
    return 0;
}
```

- 调用表达式本身就是一个值，可以放在任何需要值的位置。
- 实参也可以是一个表达式，如 `square(2 + 3)`，先算实参再调用。

一次调用发生了什么



- (1) 先计算实参，再把值按位置复制给形参。
- (2) 控制权跳到函数体，从第一行执行到 `return`。
- (3) 返回值回到调用处，主程序接着往下走。

本页要点

函数不是把代码“展开复制”，而是一次有去有回的跳转。

例题 1151 | 素数个数：题意与函数契约

题意

输入 n（$2 \le n \le 50000$），统计 $2 \sim n$ 中有多少个素数。

- 判断 x 是否素数：$x < 2$ 不是素数；否则用 $2, 3, \ldots, \lfloor \sqrt{x} \rfloor$ 试除。
- 代码里用 `d * d <= x` 代替 `d <= sqrt(x)`，避免浮点误差。
- 函数契约：`bool isPrime(int x);` 是素数返回 `true`，否则 `false`。

复杂度

单次判断 $O(\sqrt{x})$，总计 $O(n\sqrt{n})$；$n = 50000$ 时约 10^7 次运算，可以接受。

本页要点

先把“判断一个数”写对，再用循环统计个数。

例题 1151 | 完整程序

```cpp
#include <bits/stdc++.h>
using namespace std;
bool isPrime(int x) {
    if (x < 2) return false; // 0 和 1 不是素数
    for (int d = 2; d * d <= x; ++d) // 只试除到 sqrt(x)
        if (x % d == 0) return false; // 找到一个因子
    return true; // 没有因子，是素数
}
int main() {
    int n, cnt = 0;
    cin >> n;
    for (int i = 2; i <= n; ++i)
        if (isPrime(i)) ++cnt; // 复用判断函数
    cout << cnt << '\n';
    return 0;
}
```

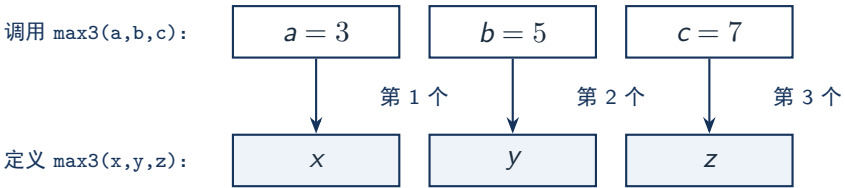
例题 1151 | 样例手推与易错点

- 样例 $n = 10$：素数有 $2, 3, 5, 7$，共 4 个，输出 `4`。
- 易错 1：忘记 `x < 2`，把 1 当成素数。
- 易错 2：把 `return true` 写进 `for` 循环，只检查了除数 2。
- 易错 3：循环条件写成 `d <= sqrt(x)`，浮点误差可能漏判完全平方数。
- 易错 4：函数返回 `bool`，主程序却忘记用 `if`，把返回值丢掉。

本页要点

先手推 $n = 10$，再提交；函数题的错误往往在边界，而不是主体。

形参与实参



- 定义中的参数叫形参（形式参数），如 `bool isPrime(int x)` 中的 `x`。
- 调用中的参数叫实参（实际参数），如 `isPrime(17)` 中的 `17`。
- 调用时按位置一一对应，类型要能转换。
- 形参名字只在函数内有效，和实参名字无关。

本页要点

形参是“占位符”，实参是“这次真正传进去的值”。

多个参数：顺序和类型都要对上

- 实参按位置对应形参，名字不需要相同。
- 一般函数交换两个实参，结果可能不同；参数顺序是契约的一部分。
- 整数传给 `double` 形参会自动转换；`double` 传给 `int` 会截断小数。
- 标准 `max` 只接受两个参数，三个数要嵌套：`max(x, max(y, z))`。

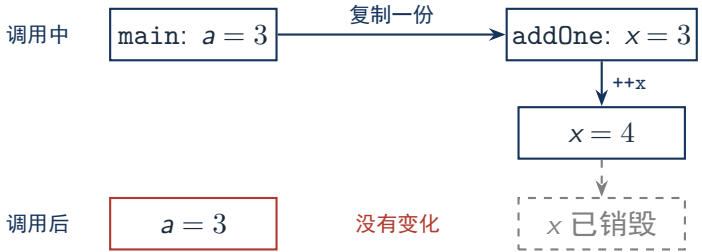
调用前先标注

在草稿上写：第 1 个位置是 _____，第 2 个位置是 _____，第 3 个位置是 _____。

本页要点

参数顺序写错，编译器通常不报错，但答案会悄悄变错。

值传递：函数拿到的是副本



- 形参是实参的副本；函数内改形参，实参不变。
- 要把结果带出来，用返回值；确实要改外部变量时才用引用 `int&`。

本页要点

值传递的方向是“复制进去”，不是“把变量本身搬进去”。

值传递代码：输出仍然是 3

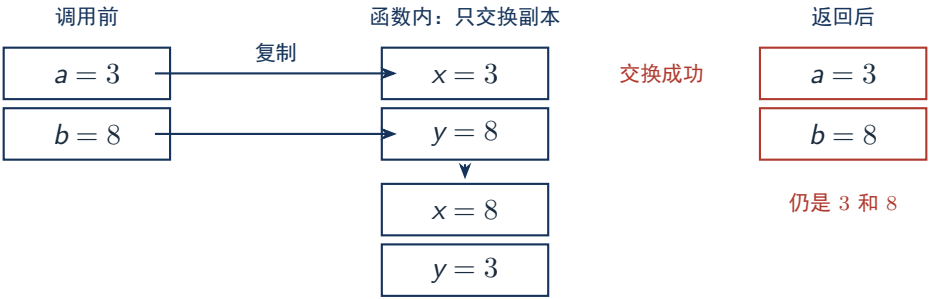
```
void addOne(int x) {
    ++x; // 改的是副本
}
int main() {
    int a = 3;
    addOne(a);
    cout << a << '\n'; // 仍然输出 3
    return 0;
}
```

- 调用 addOne(a) 时，系统新建形参 x，把 3 复制进去。
- 函数里的 ++x 只让副本变成 4；函数返回，副本销毁。

本页要点

在函数里改形参，调用处的实参“毫发无损”。

为什么 swap 改不了实参



- `swapTwo(a,b)` 交换的只是两个副本，实参从未被改动。
- 要真正交换，必须使用引用参数，让形参成为实参的别名。

本页要点

值传递下，“函数内交换”与“调用处交换”是两回事。

swap 的代码与手推

```
void swapTwo(int x, int y) {
    int old = x;
    x = y;
    y = old; // 只交换了副本
}
int main() {
    int a = 3, b = 8;
    swapTwo(a, b);
    cout << a << ' ' << b << '\n'; // 输出 3 8
    return 0;
}
```

- 进入函数：x=3，y=8；交换后：x=8，y=3。
- 返回主程序：a=3，b=8，输出 3 8。

引用参数：让函数改到原变量



同一块内存：改 x 就是改 a

- 形参写成 `int& x`，表示 x 是实参的别名，不复制。
- 引用参数会改动调用方变量，函数契约里必须写清楚。
- 只读的大对象可用 `const int&`，避免复制又不让修改。

本页要点

默认用值传递；只有“需要把修改传回去”时才用引用。

引用参数代码：输出 4

```cpp
void addOne(int& x) {
    ++x; // 改的是 a 本身
}
int main() {
    int a = 3;
    addOne(a);
    cout << a << '\n'; // 输出 4
    return 0;
}
```

- `int& x` 让形参指向 a，没有发生复制。
- 引用参数在函数调用时绑定，不能绑定常量，如 `addOne(3)` 会报错。

本页要点

看到 &，就要意识到“这个函数可能会改我的变量”。

例题 1152 | 最大数 max(x,y,z)：题意与公式

题意

已知

$$m = \frac{\max(a, b, c)}{\max(a + b, b, c) \times \max(a, b, b + c)},$$

输入 a, b, c，求 m，保留到小数点后三位。

- 先写 `max3(x,y,z)`，内部用两次标准 `max`。
- 再按公式调用三次：分子一次，分母两次。
- 输出用 `fixed << setprecision(3)`。

本页要点

公式中的括号决定每个 `max3` 的三个参数，抄公式时不要漏括号。

例题 1152 | 完整程序

```cpp
#include <bits/stdc++.h>
using namespace std;
double max3(double x, double y, double z) {
    return max(x, max(y, z)); // 两次 max 求三个数最大值
}
int main() {
    double a, b, c;
    cin >> a >> b >> c;
    double m = max3(a, b, c) / (max3(a + b, b, c) * max3(a, b, b + c));
    cout << fixed << setprecision(3) << m << '\n';
    return 0;
}
```

例题 1152 | 样例与易错点

- 样例 $a = 1, b = 2, c = 3$：

$$m = \frac{3}{\max(3,2,3) \times \max(1,2,5)} = \frac{3}{3 \times 5} = 0.2,$$

  输出 `0.200`。
- 易错 1：把 `max3(a+b,b,c)` 写成 `max3(a,b,c)+b`，参数位置错了。
- 易错 2：`max3` 用 `int`，$3/15$ 变成 0。
- 易错 3：忘记 `fixed`，`setprecision(3)` 表示有效数字而不是小数位。
- 易错 4：写 `max(x,y,z)`；标准库没有三参数版本。

本页要点

样例只给一组，自己再代 $a = b = c = 1$ 检查公式。

例题 1164 | digit 函数：题意与思路

题意

定义函数 `digit(n,k)`，分离出整数 n 从右边数第 k 个数字。样例：$n = 31859$，$k = 3$，输出 8。

- 从右数第 1 位是个位：`n % 10`。
- 第 k 位：先去掉右边 $k-1$ 位，即 `n / 10^(k-1)`，再对 10 取余。
- 不用 `pow`，用循环做 $k-1$ 次 `n /= 10`，全程整数运算。

手推

$31859 \to 318$（去掉右边两位），$318 \bmod 10 = 8$。

本页要点

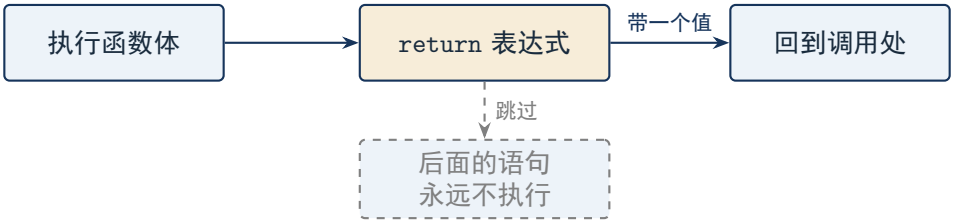
先想清楚“要删掉几位”，再决定循环次数。

例题 1164 | 完整程序与易错点

```
int digit(int n, int k) {
    while (--k > 0) n /= 10; // 去掉右边 k-1 位
    return n % 10; // 取个位
}
int main() {
    int n, k;
    cin >> n >> k;
    cout << digit(n, k) << '\n';
    return 0;
}
```

- $k = 1$ 时 `--k` 为 0，循环不执行，直接取个位。
- n 为正整数；若 k 超过位数，会得到 0（本题数据保证不出现）。
- 函数只返回数字，输出交给 `main`。

return 的两个作用



- 作用一：把表达式的值交给调用者，成为调用表达式的值。
- 作用二：立即结束当前函数，后面的语句不再执行。
- `void` 函数可以写 `return;` 提前结束；也可以不写。
- 返回类型是 `double` 时，`return 1;` 会转换成 `1.0`。

本页要点

`return` 之后的语句是“死代码”，编译器可能给出警告。

提前返回：一票否决

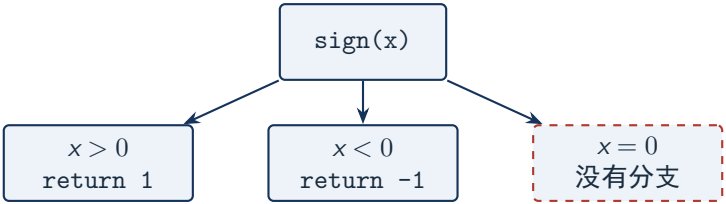
```cpp
bool isPrime(int x) {
    if (x < 2) return false; // 提前返回
    for (int d = 2; d * d <= x; ++d)
        if (x % d == 0) return false; // 一票否决
    return true; // 全部试完才为真
}
```

- 只要找到一个因子，就可以立刻断定“不是素数”，不必再试。
- 提前返回让代码更短，也避免再设一个 `flag` 变量。
- 但所有路径仍必须覆盖：最后一行 `return true` 不能漏。

本页要点

“先排除、再确认”是判断函数最常用的写法。

每条路径都要有返回值




```
int sign(int x) {
    if (x > 0) return 1;
    if (x < 0) return -1;
    // x == 0 时没有 return，会掉出函数
}
```

- 非 `void` 函数从末尾“掉出去”是未定义行为，不是稳定返回 0。
- 写法：最后放一个兜底 `return`，或让所有分支都 `return`。

返回值类型会自动转换

```
double half(int x) {
    return x / 2; // 整数除法，再转成 double
}
double half2(int x) {
    return x / 2.0; // 2.0 是 double，得到小数
}
int main() {
    cout << half(3) << ' ' << half2(3) << '\n'; // 1 1.5
    return 0;
}
```

- return 表达式的类型会转换成函数的返回类型。
- double 转 int 直接截断小数，不是四舍五入。
- 想让除法得到小数，分子或分母至少有一个是浮点数。

用 bool 函数表达“判断”

- 提前返回：一旦发现反例就 `return false`，最后 `return true`。
- 判断函数只回答“是/否”，不要在里面对外输出。

```
bool isEven(int x) {
    return x % 2 == 0;
}
bool isPrime(int x) {
    if (x < 2) return false;
    for (int d = 2; d * d <= x; ++d)
        if (x % d == 0) return false;
    return true;
}
```

本页要点

把“判断”和“输出”分开，函数才能被下一道题直接复用。

void 函数与提前返回

```cpp
void printFactors(int n) {
    if (n < 1) return; // 提前结束
    for (int d = 1; d <= n; ++d) {
        if (n % d == 0) cout << d << ' ';
    }
    cout << '\n';
}
int main() {
    printFactors(12); // 1 2 3 4 6 12
    return 0;
}
```

- void 表示“不交回值”，函数只完成动作，如输出、修改引用参数。
- void 函数里的 return; 只能提前结束，后面不能跟表达式。
- 能用返回值表达的结果，优先用返回值，方便测试和复用。

例题 1153 | 绝对素数：复用判断函数

题意

如果一个两位自然数是素数，且交换十位与个位后仍是素数，就称为绝对素数。求所有两位绝对素数，由小到大每行一个。

- 枚举 $x = 10 \sim 99$。
- 反序数：`r = x % 10 * 10 + x / 10`。
- 两个条件都调用 `isPrime`，直接复用 1151 的函数。
- 按 x 递增枚举，输出自然有序。

本页要点

11 是绝对素数：交换后还是 11。

例题 1153 | 完整程序

```
bool isPrime(int x) {
    if (x < 2) return false;
    for (int d = 2; d * d <= x; ++d)
        if (x % d == 0) return false;
    return true;
}
int main() {
    for (int x = 10; x <= 99; ++x) {
        int r = x % 10 * 10 + x / 10; // 交换十位与个位
        if (isPrime(x) && isPrime(r)) // 两个都是素数
            cout << x << '\n';
    }
    return 0;
}
```

例题 1153 | 易错点

- 10 的反序是 1，`isPrime(1)` 必须返回 `false`。
- 不要在 `isPrime` 里输出，否则会打印一堆 `true/false`。
- `&&` 具有短路求值：前一个为假时，后一个不会执行。
- 输出顺序由 x 递增保证，不要再额外排序。

本页要点

先写“判断一个数”，再把“判断两个数”写成一行 `if`。

局部变量与作用域

- 在函数内声明的变量叫局部变量，只在函数内可见。
- 形参也是局部变量；每次调用都会新建一份。
- 不同函数可以有同名局部变量，互不干扰。

```
int f() {
    int t = 1; // 只在 f 内可见
    return t;
}
int g() {
    int t = 2; // 与 f 的 t 无关
    return t;
}
```

本页要点

作用域回答的是“这个名字在哪里能被看见”。

块作用域：一对花括号就是一层

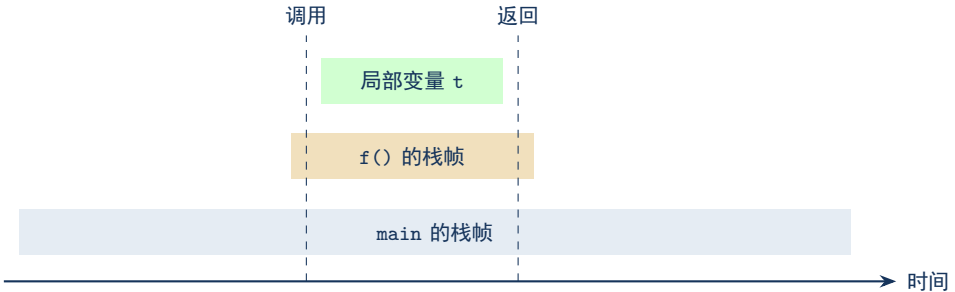
```cpp
int main() {
    int x = 1;
    {
        int y = 2; // y 只在这个块内
        cout << x + y; // 3
    }
    // cout << y; // 错误：y 已经销毁
    return 0;
}
```

- 内层块可以访问外层变量，外层不能访问内层变量。
- 内层声明同名变量会遮蔽外层变量，容易看错。

本页要点

花括号不仅分组语句，也划分了变量的可见范围。

生命周期：变量什么时候生、什么时候死

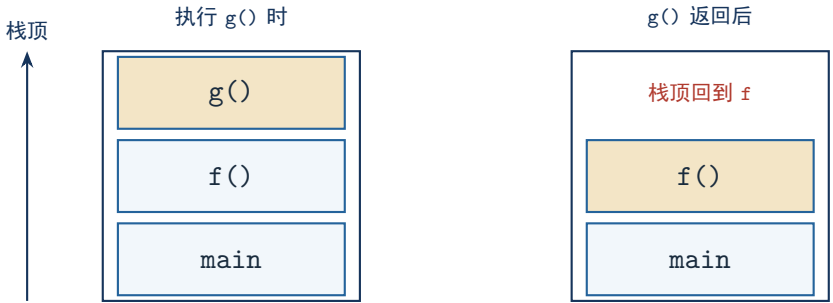


- 作用域是空间范围：名字在哪些代码里可见。
- 生命周期是时间范围：变量从创建到销毁的这段时间。

本页要点

作用域管“看得见”，生命周期管“活得久”。

调用栈：main 调用 f，f 调用 g

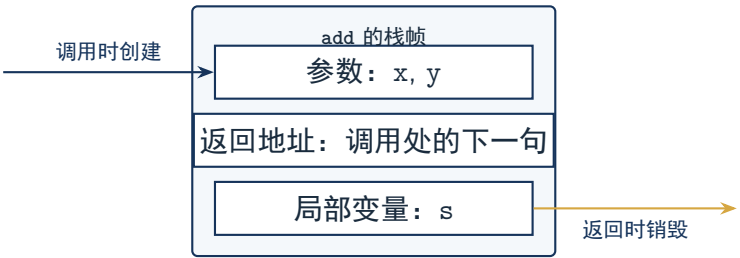


- 每调用一次函数，就在栈顶压入一帧；返回时弹出。
- 栈是“后进先出”：最后调用的函数最先返回。

本页要点

调用栈记录了“现在执行到哪、返回后回到哪”。

栈帧里有什么



- 参数、局部变量、返回地址都放在同一块栈帧里。
- 返回地址告诉 CPU：函数结束后回到调用处的哪一行。
- 所以递归每深入一层，就多一份参数和局部变量。

本页要点

理解栈帧，就能理解“局部变量为什么返回后消失”。

命名冲突与全局变量

- 内层块可以遮蔽（shadow）外层同名变量，读代码时容易看错。
- 全局变量所有函数都能改，出错时很难定位是谁改的。
- 推荐：数据用参数传入，结果用返回值传出；全局只放常量。

```
int cnt = 0; // 全局变量
void add() {
    ++cnt; // 改的是全局 cnt
}
int main() {
    int cnt = 10; // 局部变量遮蔽全局
    add();
    cout << cnt << '\n'; // 输出 10
    return 0;
}
```

static 局部变量：只初始化一次

```cpp
void counter() {
    static int cnt = 0; // 只初始化一次
    ++cnt;
    cout << cnt << ' ';
}
int main() {
    counter(); // 1
    counter(); // 2
    counter(); // 3
    return 0;
}
```

- 普通局部变量每次调用都重新创建；static 变量只创建一次。
- static 局部变量的作用域仍在函数内，但生命周期到程序结束。
- 初学阶段少用；需要跨调用保存状态时，优先考虑把状态作为参数传递。

例题 1150 | 完全数：题意与思路

题意

求正整数 2 和 n 之间的所有完全数（$n \le 5000$），一行一个。完全数：因子之和等于它本身，如 $6 = 1 + 2 + 3$。

- 写 `sumProper(x)`：枚举 $d = 1 \sim x - 1$，能整除就累加。
- 主程序枚举 $x = 2 \sim n$，若 `sumProper(x) == x` 就输出。
- $n \le 5000$，$O(n^2)$ 约 1.25×10^7 次取模，可以接受。

本页要点

“真因子”不包含 x 本身，所以循环到 $x - 1$。

例题 1150 | 完整程序

```
int sumProper(int x) {
    int s = 0;
    for (int d = 1; d < x; ++d) {
        if (x % d == 0) s += d; // d 是 x 的真因子
    }
    return s;
}
int main() {
    int n;
    cin >> n;
    for (int x = 2; x <= n; ++x) {
        if (sumProper(x) == x) cout << x << '\n';
    }
    return 0;
}
```

例题 1150 | 易错点

- 1 不是完全数：它的真因子和为 0，不是 1。
- 累加器 `s` 必须从 0 开始；不要漏掉 $d = 1$。
- x 递增枚举，输出自然由小到大；5000 以内 `int` 不会溢出。
- 循环写 `d < x` 而不是 `d <= x`，否则每个数都会被自己整除。

本页要点

“真因子”与“因子”只差一个 x，边界就错在这里。

例题 1154 | 亲和数：两个函数结果互相验证

题意

若 a 的真因子之和为 b，且 b 的真因子之和又等于 a，则 a, b 是一对亲和数。求最小的一对（$a \neq b$，输出 $a < b$）。

- 复用 `sumProper`，不用再写一遍因子求和。
- 枚举 a，令 $b = \texttt{sumProper}(a)$。
- 若 $b > a$ 且 `sumProper(b) == a`，就找到一对，输出并结束。
- 只输出 $a < b$ 的一对，避免同一对输出两次。

本页要点

最小的一对是 220 和 284，可以用程序验证。

例题 1154 | 完整程序

```
int sumProper(int x) {
    int s = 0;
    for (int d = 1; d < x; ++d)
        if (x % d == 0) s += d;
    return s;
}
int main() {
    for (int a = 2; ; ++a) {
        int b = sumProper(a);
        if (b > a && sumProper(b) == a) { // 找到亲和数对
            cout << a << ' ' << b << '\n';
            break;
        }
    }
    return 0;
}
```

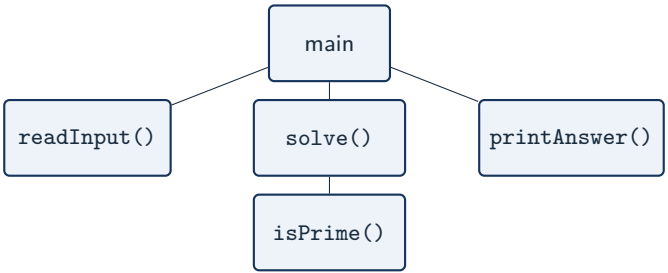
例题 1154 | 易错点

- 若 $b \le a$ 就跳过：$b < a$ 时这一对已输出过，$b = a$ 时 a 是完全数。
- 循环不设上界，靠 `break` 结束；题目保证有解。
- 两次调用 `sumProper`，第二次的参数是 b，不要写成 a。
- 输出顺序要求 $a < b$，所以先判断 `b > a`。

本页要点

用同一个函数互相验证，是“复用”最直接的收益。

自顶向下：把大问题拆成小函数



- 先写主程序的步骤（中文伪代码），每一步取一个动词。
- 每个动词对应一个函数；函数只做一件事。
- 函数之间通过参数和返回值连接，不靠全局变量。

本页要点

好的拆分让主程序像目录一样可读：读入、判断、输出各自独立。

函数设计三问

1. 输入是什么？类型、范围、特殊值（0、负数、空串）。
2. 输出是什么？返回类型、含义、无解时返回什么。
3. 会不会改动外部？只读参数用值传递；需要修改才用引用。

命名建议

判断用 `isPrime`、`isEven`；计算用 `sumProper`、`digit`、`max3`。名字要能读出“做什么”，而不是“怎么做”。

本页要点

先写契约，再写函数体；契约不清，函数一定写乱。

反例：一个函数做了三件事

```
void process() {
    int n;
    cin >> n; // 1. 读入
    int s = 0;
    for (int i = 1; i <= n; ++i) {
        s += i; // 2. 计算
    }
    cout << s << '\n'; // 3. 输出
}
```

- 读入、计算、输出混在一起，换个输入方式就得改整个函数。
- 想单独测试“求和”是否正确，必须先准备输入、再抓输出。

本页要点

一个函数里出现“读入 + 计算 + 输出”，就是该拆分的信号。

改写：拆成三个小函数

```cpp
int readN() {
    int n;
    cin >> n;
    return n;
}
int sumTo(int n) {
    int s = 0;
    for (int i = 1; i <= n; ++i) {
        s += i; // 只负责求和
    }
    return s;
}
void printAnswer(int s) {
    cout << s << '\n'; // 只负责输出
}
```

- 三个函数各自只做一件事，可以分别测试、分别复用。

main 只剩三步




```
int main() {
    int n = readN(); // 1. 读入
    int ans = sumTo(n); // 2. 求解
    printAnswer(ans); // 3. 输出
    return 0;
}
```

本页要点

主程序越短，越能一眼看出整体流程；细节都在函数里。

代码规范：名字、缩进、注释

```
int sumProper(int x) { // 名字说明“做什么”
    int s = 0;
    for (int d = 1; d < x; ++d) {
        if (x % d == 0) {
            s += d; // 缩进 4 空格
        }
    }
    return s; // 关键行有注释
}
```

- 一条语句一行，不写 `if (x) y=1; else y=2;` 这种压行。
- 花括号成对对齐；缩进统一 4 个空格。
- 注释解释“为什么”，不要重复“做了什么”。
- 参数不宜过多；超过四五个时，考虑拆函数或合并成结构。

例题 1157 ｜哥德巴赫猜想：题意与拆分

题意

任一大于 2 的偶数都可以写成两个素数之和。本题要求把 $6 \sim 100$ 的所有偶数表示成两个素数之和。每个数只拆一次，保证第一个加数最小，格式如 6=3+3。

- 对每个偶数 n，从小到大枚举 p。
- 若 p 和 $n-p$ 都是素数，输出并 `break`。
- p 只枚举到 $n/2$，保证第一个加数不大于第二个。
- 主程序只负责枚举和输出，判断交给 `isPrime`。

本页要点

先固定“第一个加数最小”，再让 p 递增，就自然满足要求。

例题 1157 | 完整程序

```cpp
bool isPrime(int x) {
    if (x < 2) return false;
    for (int d = 2; d * d <= x; ++d)
        if (x % d == 0) return false;
    return true;
}
int main() {
    for (int n = 6; n <= 100; n += 2) {
        for (int p = 3; p <= n / 2; ++p) {
            if (isPrime(p) && isPrime(n - p)) {
                cout << n << '=' << p << '+' << n - p << '\n';
                break; // 每个偶数只输出一次
            }
        }
    }
    return 0;
}
```

例题 1157 | 输出与易错点

- 前几行应为 6=3+3、8=3+5、10=3+7；最后一行是 100=3+97。
- 易错 1：忘记 `break`，同一个偶数会输出多组拆分。
- 易错 2：p 枚举到 $n-1$，会重复输出 $3+97$ 和 $97+3$。
- 易错 3：输出格式多了空格，如 `6 = 3 + 3`，与样例不符。
- 易错 4：循环写成 `n < 100`，漏掉 100。

本页要点

先手算 $6, 8, 10$，再让程序跑全表；格式错误在 OJ 上同样算错。

例题 1156 | 求 π 的值：公式与思路

题意

根据

$$\arctan x = x - \frac{x^3}{3} + \frac{x^5}{5} - \frac{x^7}{7} + \cdots, \qquad \pi = 6\arctan\left(\frac{1}{\sqrt{3}}\right),$$

定义函数 `arctanx(x)`，当最后一项小于 10^{-6} 时求 π，保留 10 位小数。

- 维护当前项 `term` 和分母 `k`。
- 递推：`term = -term * x * x * k / (k + 2)`；然后 `k += 2`。
- 累加到 $|\texttt{term}| < 10^{-6}$ 为止。

本页要点

符号在递推中自动交替；判断停止要用绝对值。

例题 1156 | 完整程序

```
#include <bits/stdc++.h>
using namespace std;
double arctanx(double x) {
    double sum = 0, term = x;
    int k = 1;
    while (fabs(term) >= 1e-6) { // 最后一项小于 1e-6 停止
        sum += term;
        term = -term * x * x * k / (k + 2);
        k += 2;
    }
    return sum;
}
int main() {
    double pi = 6 * arctanx(1 / sqrt(3.0));
    cout << fixed << setprecision(10) << pi << '\n';
    return 0;
}
```

例题 1156 | 精度与易错点

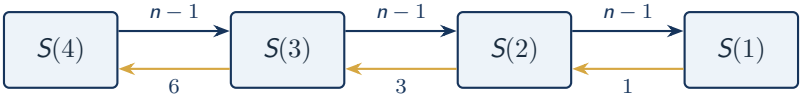
- `1 / sqrt(3.0)` 是 `double`；写成 `1 / 3` 会得到 0。
- `fabs` 用于浮点数；`abs` 主要处理整数。
- 循环条件用 $\geq$：最后一项小于 10^{-6} 时不再加入。
- 输出 10 位小数；不同编译器末位可能差 1，属于正常浮点误差。
- 可用 `M_PI` 对照，误差应在 10^{-6} 以内。

本页要点

浮点题先检查“整数除法”和“停止条件”，再检查公式。

递归：函数调用自己

去：不断把问题变小

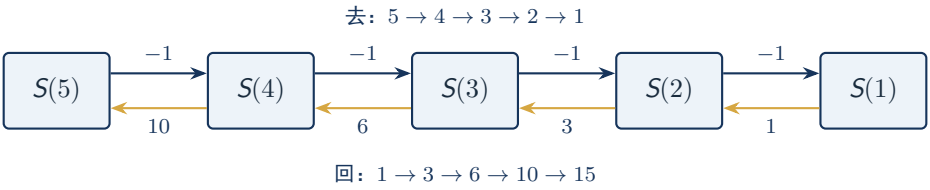


回：逐层把结果带回来

```
long long fact(int n) {
    if (n == 0) return 1; // 终止条件
    return n * fact(n - 1); // 递归关系
}
```

- 递归函数在函数体里调用自己，形式更贴近数学定义。
- 每次调用都必须让问题变小，否则永远到不了终止条件。

递归展开：从 $S(5)$ 到 $S(1)$ 再回来

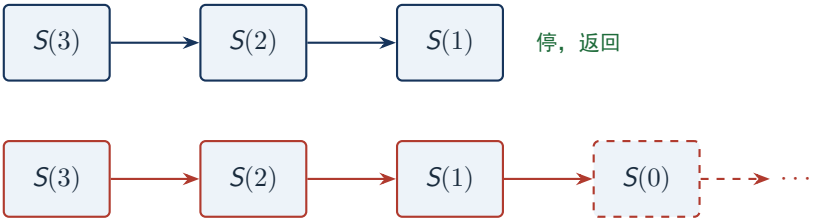


- $S(5) = 5 + S(4) = 5 + 4 + S(3) = \cdots = 5 + 4 + 3 + 2 + 1 = 15$。
- 去的过程只做“压栈”，真正的加法在回来的路上才做。
- 手推时先写终止值，再从最内层往外代。

本页要点

递归的执行顺序是“先一路向下，再一路向上”。

递归的两项必要条件

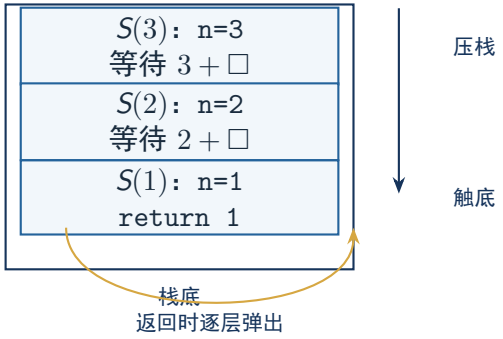


没有终止条件：永远停不下来

1. 终止条件（base case）：最小问题直接给出答案。
2. 递归关系：每次调用规模更小，最终能到达终止条件。

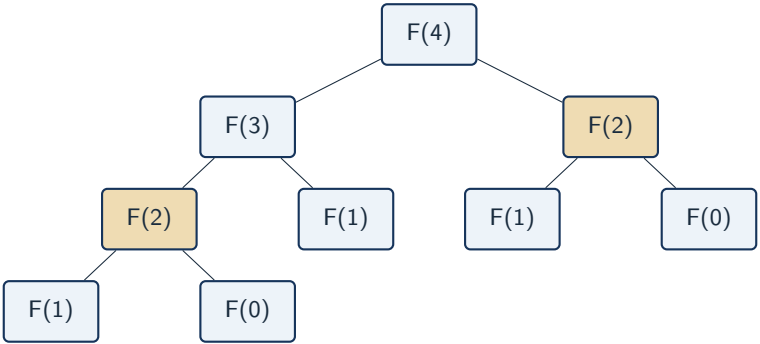
- 缺终止条件：无限递归，栈溢出，程序崩溃。
- 规模没有变小：同样永远到不了终止条件。

栈帧：递归如何返回



- 压栈：每层保存自己的参数和返回地址。
- 触底：最小问题直接返回。
- 出栈：把子问题的结果带回上一层计算。

斐波那契：同一个子问题被算多次



- 求 $F(4)$ 时，两个金色 `F(2)` 被各自完整计算一遍。
- 每增加一层，重复的子问题迅速增多，调用次数约 $O(2^n)$。
- 解决：记忆化（把算过的答案存起来）或改成递推循环。

递归与循环：怎样选

- 线性求和、求阶乘：循环更省内存，也不会栈溢出。
- 嵌套根式、连分式、进制转换：递归直接对应数学定义，代码短。
- 朴素斐波那契递归会重复计算，$F(30)$ 的调用次数已经很大。

```
long long fib(int n) {
    if (n <= 1) return n; // 终止条件
    return fib(n - 1) + fib(n - 2); // 两次递归
}
```

- 优化：记忆化（保存算过的答案）或改成递推循环。

本页要点

先写递归，再判断是否需要优化；不要为了递归而递归。

例题 1158 | 递归求和：定义与代码

题意与递推式

用递归求 $1+2+\cdots+N$：$S(1)=1$，$S(n)=n+S(n-1)$。

```cpp
#include <bits/stdc++.h>
using namespace std;
long long sumTo(int n) {
    if (n == 1) return 1; // 终止条件
    return n + sumTo(n - 1); // 递归关系
}
int main() {
    int n;
    cin >> n;
    cout << sumTo(n) << '\n';
    return 0;
}
```

例题 1158 | 展开与易错点

- $S(5) = 5 + S(4) = 5 + 4 + S(3) = \cdots = 5 + 4 + 3 + 2 + 1 = 15$，与样例一致。
- 易错 1：终止条件写成 `n == 0`，但 $S(0)$ 没有定义；本题 $N \geq 1$。
- 易错 2：返回类型用 `int`，N 很大时溢出；用 `long long` 更稳。
- 易错 3：写成 `sumTo(n) + sumTo(n - 1)`，参数没有变小，无限递归。
- 复杂度：时间 $O(n)$，递归深度 $O(n)$。

本页要点

递归求和与循环结果相同；这题的重点是理解“层层返回”。

例题 1161 | 转进制：题意与思路

题意

用递归把十进制数 X 转换成 M 进制（$M \le 16$）。样例：31 转 16 进制，输出 1F。

- $X \bmod M$ 是当前最低位。
- X/M 是去掉最低位后的数，继续递归。
- 先递归处理高位，再输出当前位，才能得到从高到低的顺序。
- $10 \sim 15$ 映射为 A∼F。

本页要点

递归负责“先处理高位”，输出语句放在递归调用之后。

例题 1161 | 完整程序与易错点

```cpp
void convert(int x, int m) {
    if (x >= m) convert(x / m, m); // 先递归处理高位
    int d = x % m;
    if (d < 10) cout << d; // 0~9 直接输出
    else cout << char('A' + d - 10); // 10~15 输出 A~F
}
int main() {
    int x, m;
    cin >> x >> m;
    convert(x, m);
    cout << '\n';
    return 0;
}
```

- 输出顺序：先递归高位，再输出低位；反过来会得到倒序。
- 31：$31/16=1$，先输出 1；再输出 $31\%16=15$ 对应的 F。
- $X=0$ 时直接输出 0；本题 X 为正整数。

例题 1162 | 字符串逆序：题意与思路

题意

输入一串以 ! 结束的字符，按逆序输出。样例：abc! 输出 cba。

- 读一个字符 c。
- 若 c 是 !，结束。
- 否则先递归读后面的字符，等后面的都输出完，再输出 c。
- 这样最先读入的字符最后输出，自然逆序。

本页要点

“递归之后再输出”是逆序的关键。

例题 1162 | 完整程序与易错点

```cpp
#include <bits/stdc++.h>
using namespace std;
void reverseStr() {
    char c = cin.get(); // 读一个字符，包括空格
    if (c == '!') return; // 终止条件
    reverseStr(); // 先处理后面的字符
    cout << c; // 回来后再输出自己
}
int main() {
    reverseStr();
    cout << '\n';
    return 0;
}
```

- 用 `cin.get()` 读单个字符；`cin >> c` 会跳过空格。
- `!` 本身不输出；输出没有空格。
- 递归深度等于字符串长度；本题长度很小。

例题 1163 | 阿克曼函数：定义

定义

$$A(m,n)=\begin{cases} n+1, & m=0, \\ A(m-1,1), & m>0,\ n=0, \\ A(m-1,A(m,n-1)), & m>0,\ n>0. \end{cases}$$

- 第一行处理 $m=0$；第二行处理 $n=0$；第三行同时递归两层。
- 先判断 `m == 0`，再判断 `n == 0`，顺序不能反。

例题 1163 | 完整程序

```cpp
long long akm(long long m, long long n) {
    if (m == 0) return n + 1; // 第一行
    if (n == 0) return akm(m - 1, 1); // 第二行
    return akm(m - 1, akm(m, n - 1)); // 第三行
}
int main() {
    long long m, n;
    cin >> m >> n;
    cout << akm(m, n) << '\n';
    return 0;
}
```

例题 1163 | 展开与易错点

- 样例 $A(2,3)=9$。展开时先算内层 $A(2,2)$，再代回外层。
- 易错 1：先判断 $n=0$。$m=0, n=0$ 时应返回 1，不是 $A(-1,1)$。
- 易错 2：把 $A(m-1, A(m,n-1))$ 写成 $A(A(m-1,m), n-1)$，参数位置全错。
- 易错 3：递归层数增长极快；题目限制 $m \le 3, n \le 10$ 是有原因的。
- 返回值用 `long long`，避免中间结果溢出。

本页要点

照定义逐行翻译，不要凭感觉改写递归式。

例题 1165 | Hermite 多项式：定义

定义

$$h_n(x) = \begin{cases} 1, & n = 0, \\ 2x, & n = 1, \\ 2xh_{n-1}(x) - 2(n-1)h_{n-2}(x), & n > 1. \end{cases}$$

- 两个终止条件 $n = 0$ 与 $n = 1$ 都要写。
- $n > 1$ 时同时用到前两项，属于“二阶递归”。

例题 1165 | 完整程序

```
double h(int n, double x) {
    if (n == 0) return 1; // 终止条件
    if (n == 1) return 2 * x; // 终止条件
    return 2 * x * h(n - 1, x) - 2 * (n - 1) * h(n - 2, x);
}
int main() {
    int n;
    double x;
    cin >> n >> x;
    cout << fixed << setprecision(2) << h(n, x) << '\n';
    return 0;
}
```

例题 1165 | 样例与易错点

- 样例 $n=1, x=2$：$h_1(2)=2\times 2=4$，输出 `4.00`。
- 易错 1：只写 $n=0$ 的终止条件，$n=1$ 时继续递归到 $n=-1$。
- 易错 2：`2 * x` 中 `x` 是 `int`，先算整数再转 `double`；形参用 `double`。
- 易错 3：系数写成 $2n$ 而不是 $2(n-1)$，公式抄错。
- 每层调用两次，时间约 $O(2^n)$；本题 n 很小，可以接受。

本页要点

输出保留两位小数：`fixed << setprecision(2)`。

例题 1166 | 求 $f(x,n)$：定义

定义与递推

$$f(x,1)=\sqrt{1+x},\qquad f(x,n)=\sqrt{n+f(x,n-1)}\quad (n>1).$$

例如 $f(x,4)=\sqrt{4+\sqrt{3+\sqrt{2+\sqrt{1+x}}}}$，从最内层往外算。

- 最内层是 $n=1$，不是 $n=0$。
- 每一层只增加一个根号，递归方向是 $n\to n-1$。

例题 1166 | 完整程序

```
double f(double x, int n) {
    if (n == 1) return sqrt(1 + x); // 最内层
    return sqrt(n + f(x, n - 1)); // 向外套一层根号
}
int main() {
    double x;
    int n;
    cin >> x >> n;
    cout << fixed << setprecision(2) << f(x, n) << '\n';
    return 0;
}
```

例题 1166 | 样例与易错点

- 样例 $x = 4.2, n = 10$，输出 `3.68`。
- 易错 1：终止条件写成 $n = 0$，但定义从 $n = 1$ 开始。
- 易错 2：写成 $\sqrt{n + f(x, n-1)}$，根号只包住 n，与定义不符。
- 易错 3：n 用 `double` 做循环变量，递归边界不稳定；n 用 `int`。
- 易错 4：输出忘记 `fixed`，小数位数不对。

本页要点

把最内层写对，外层照定义套；递归方向是 $n \to n-1$。

例题 1167 | 再求 $f(x, n)$：定义

定义

$$f(x,1)=\frac{x}{1+x},\qquad f(x,n)=\frac{x}{n+f(x,n-1)}\quad (n>1).$$

- 最内层 $n=1$ 直接算 $x/(1+x)$。
- 外层每层做一次“n+ 子结果”的除法。

例题 1167 | 完整程序

```
double f(double x, int n) {
    if (n == 1) return x / (1 + x); // 最内层
    return x / (n + f(x, n - 1)); // 连分式
}
int main() {
    double x;
    int n;
    cin >> x >> n;
    cout << fixed << setprecision(2) << f(x, n) << '\n';
    return 0;
}
```

例题 1167 | 样例与易错点

- 样例 $x=1, n=2$：$f(1,2)=1/(2+1/2)=0.4$，输出 `0.40`。
- 易错 1：`x` 用 `int`，`x / (1 + x)` 会做整数除法。
- 易错 2：终止条件应写 $n=1$，写成 $n=0$ 会继续递归到负数。
- 易错 3：分母写成 `n + f(x,n-1)` 时漏掉外层括号，运算顺序会变。

本页要点

连分式从最内层写起，外层只做一次除法。

五类高频错误

错误	表现	改法
忘记 `return`	走到末尾，结果随机	每条路径都返回
返回类型不符	整数除法，小数丢失	写 `1.0` 或显式转换
参数顺序错	编译通过，答案变错	调用前标注每个位置
命名冲突	遮蔽外层，改错对象	换名或缩小作用域
函数里直接输出	无法复用，输出混杂	返回值交给 `main`

共同点

这些错误大多不会让编译失败，只会让答案错；所以必须靠样例和边界测试发现。

用样例定位错误：一次调试记录

现象

`isPrime(9)` 返回 `true`，但 $9 = 3 \times 3$，应为 `false`。

输入 x	试除 d	返回值	说明
1	无	`false`	边界正确
2	无	`true`	最小素数正确
9	$2, 3$	`false`	$d = 3$ 时整除，正确
17	$2, 3, 4$	`true`	$d^2 > 17$ 时停止

- 用 $1, 2, 9, 17$ 四个输入覆盖“边界、最小、合数、素数”。
- 若 9 出错，先看循环条件是否写成 `d < x` 或忘记 `return false`。

本页要点

把函数单独拿出来测，比在完整程序里猜快得多。

提交前的检查清单

1. 函数的返回类型和每个 `return` 表达式是否匹配？
2. 是否所有路径都有返回值？终止条件是否覆盖最小输入？
3. 实参的数量、顺序、类型是否与形参一致？
4. 局部变量是否在使用前初始化？有没有意外遮蔽？
5. 函数是否只做一件事？输出是否集中在 `main`？
6. 样例、边界（0、1、最大值）是否都手推过？

本页要点

调试顺序：先测函数，再测主程序；先小样例，再大规模。

练习索引：使用说明

- 本章共 35 道练习，全部来自一本通 1150–1167、1397–1413。
- 有学校 OJ 题号的，点击蓝色题名进入；没有的标“题包”，在题包中练习。
- 建议顺序：先做 1151、1152、1164，再做 1150、1154、1157、1156，最后做递归 1158–1167。
- 每道题先写函数契约，再写代码；写完用样例和小边界各测一次。

函数题笔记格式

函数名与契约（输入、返回）　终止条件/边界　易错点　复杂度

本章练习题索引 | 1–7 / 35

1. P5150 | 求正整数 2 和 n 之间的完全数（一本通 1150）
2. P5151 | 素数个数（一本通 1151）
3. P5152 | 最大数 max(x,y,z)（一本通 1152）
4. P5153 | 绝对素数（一本通 1153）
5. P5154 | 亲和数（一本通 1154）
6. P5155 | 回文三位数（一本通 1155）
7. P5156 | 求 π 的值（一本通 1156）

本章练习题索引 | 8–14 / 35

8. P5157 | 哥德巴赫猜想（一本通 1157）
9. 简单算术表达式求值（一本通 1397） P4955
10. 短信计费（一本通 1398） P4956
11. 甲流病人初筛（一本通 1399） P4957
12. P20117 | 统计单词数（一本通 1400）
13. 机器翻译（一本通 1401） P4958
14. Vigenère 密码（一本通 1402） P4959

本章练习题索引 | 15–21 / 35

15. 素数对（一本通 1403） P4960
16. 我家的门牌号（一本通 1404） P4961
17. 质数的和与积（一本通 1405） P4962
18. 单词替换（一本通 1406） P4963
19. P20097 | 笨小猴（一本通 1407）
20. 素数回文数的个数（一本通 1408） P4964
21. 判决素数个数（一本通 1409） P4965

本章练习题索引 | 22–28 / 35

22. 最大质因子序列（一本通 1410） P4966
23. 区间内的真素数（一本通 1411） P4967
24. 二进制分类（一本通 1412） P4968
25. 确定进制（一本通 1413） P4969
26. P5158 | 求 1+2+3+...（一本通 1158）
27. P5159 | 斐波那契数列（一本通 1159）
28. P5160 | 倒序数（一本通 1160）

本章练习题索引 | 29–35 / 35

29. P5161 | 转进制（一本通 1161）
30. P5162 | 字符串逆序（一本通 1162）
31. P5163 | 阿克曼 (Ackmann) 函数（一本通 1163）
32. P5164 | digit 函数（一本通 1164）
33. P5165 | Hermite 多项式（一本通 1165）
34. P5166 | 求 f(x,n)（一本通 1166）
35. P5167 | 再求 f(x,n)（一本通 1167）