函数与程序结构

胡船长

初航我带你，远航靠自己

一、本期内容

1. 第一节：C 语言中的『函数』
2. 第二节：重要的『递归函数』
3. 第三节：函数的拓展知识
4. 第四节：函数-实战练习

一. C 语言中的『函数』

聊聊：作用域

聊聊：作用域

定义的变量，**作用**的区**域**

函数的基本结构

返回值 函数名(参数列表) 函数体

```
int sum(int a, int b)
{
    return a + b;
}
```

函数的基本结构


```c
int sum(int a, int b)
{
    return a + b;
}
```

函数的基本结构


```c
int sum(int a, int b)
{
    return a + b;
}
```

函数的基本结构


```c
int sum(int a, int b)
{
    return a + b;
}
```

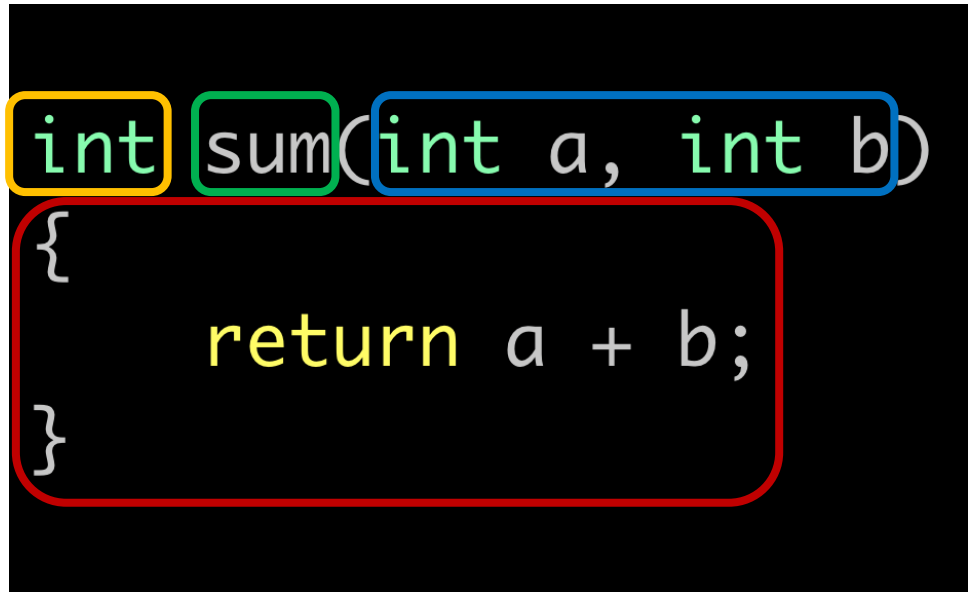
函数的基本结构

返回值 函数名(参数列表) 函数体

```
int sum(int a, int b)
{
    return a + b;
}
```

函数的基本结构

返回值 函数名(参数列表) 函数体



思考：为什么一定要有函数结构?

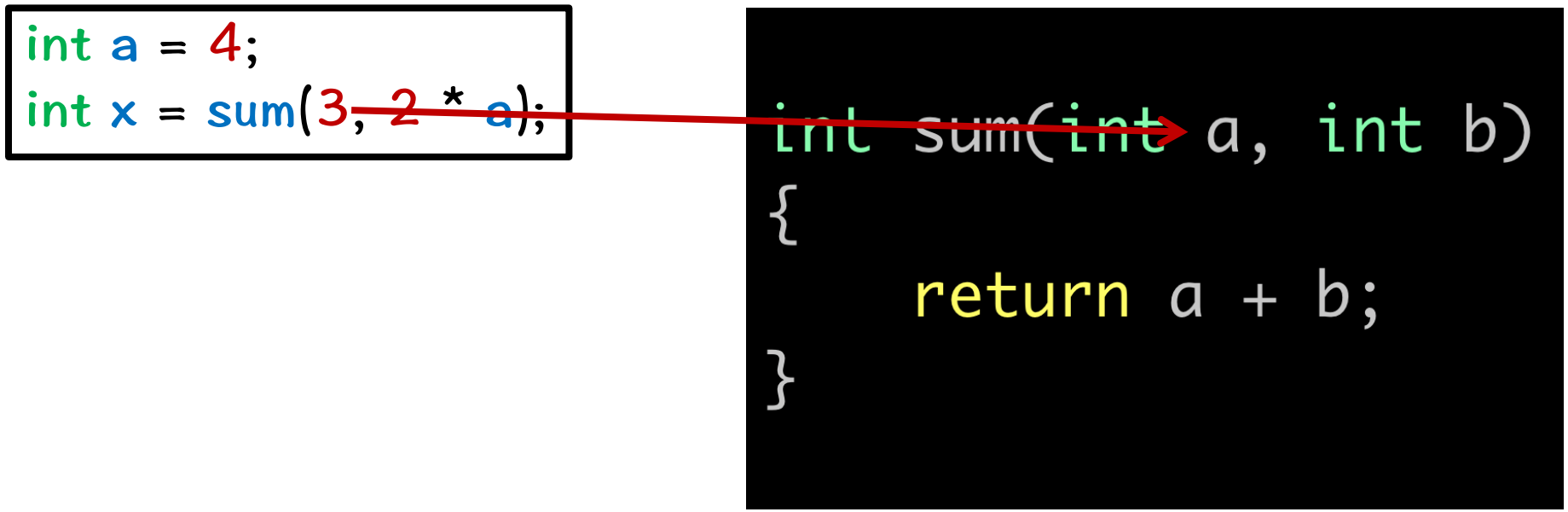
实参与形参：李逵与李鬼

实参与形参

```
int a = 4;
int x = sum(3, 2 * a);
```

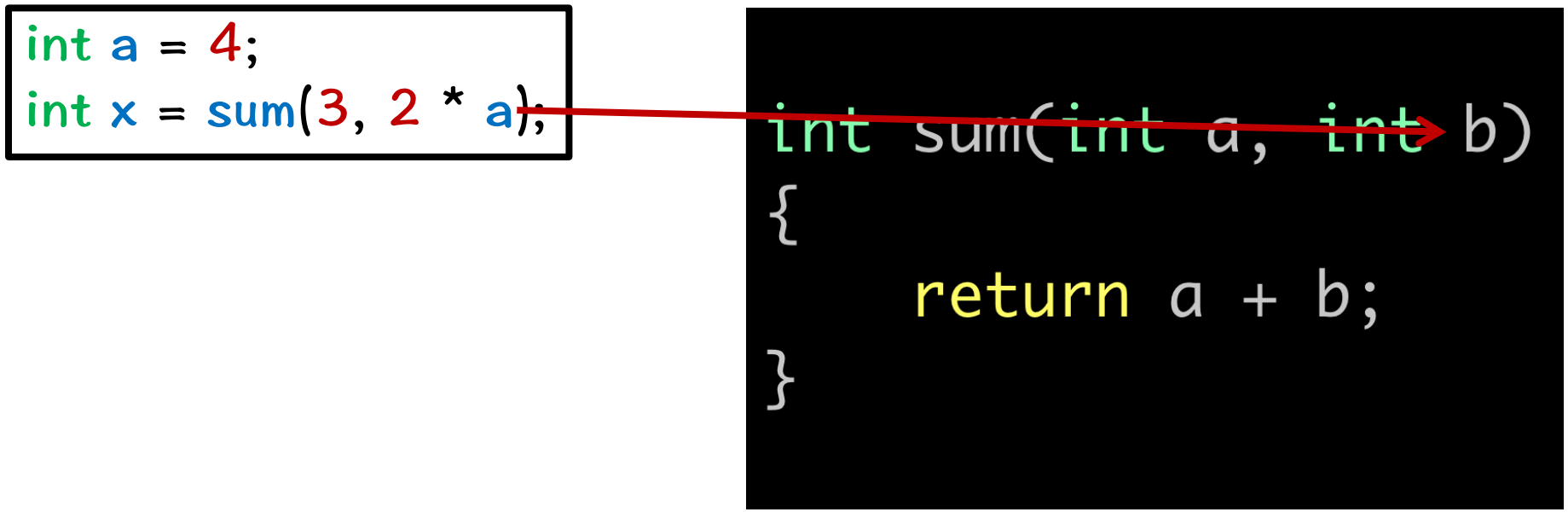
```
int sum(int a, int b)
{
    return a + b;
}
```

实参与形参



实参与形参

```c
int a = 4;
int x = sum(3, 2 * a);
```



实参与形参

形参的所有修改，都不会改变实参

实参：李逵

形参：李鬼

函数的定义与声明

函数的定义与声明

定义

```c
int sum(int a, int b)
{
    return a + b;
}
```

声明

```c
int sum(int, int);
```

二. 重要的『递归函数』

递归函数：自己调用自己的函数

递归函数：求阶乘

```c
int f(int n) {
    if (n == 1) return 1;
    return n * f(n - 1);
}
```

计算5的阶乘

n=5

```
int f(int n) {
    if (n == 1) return 1;
    return n * f(n - 1);
}
```

计算5的阶乘

n=5

```
int f(int n) {
    if (n == 1) return 1;
    return n * f(n - 1);
}
```

计算5的阶乘

n=5

```
int f(int n) {
    if (n == 1) return 1;
    return n * f(n - 1);
}
```

计算5的阶乘

n=5

```
int f(int n) {
    if (n == 1) return 1;
    return n * f(n - 1);
}
```

n=4

```
int f(int n) {
    if (n == 1) return 1;
    return n * f(n - 1);
}
```

计算5的阶乘

n=5

```
int f(int n) {
    if (n == 1) return 1;
    return n * f(n - 1);
}
```

n=4

```
int f(int n) {
    if (n == 1) return 1;
    return n * f(n - 1);
}
```

计算5的阶乘

n=5

```
int f(int n) {
    if (n == 1) return 1;
    return n * f(n - 1);
}
```

n=4

```
int f(int n) {
    if (n == 1) return 1;
    return n * f(n - 1);
}
```

n=3

```
int f(int n) {
    if (n == 1) return 1;
    return n * f(n - 1);
}
```

计算5的阶乘

n=5

```
int f(int n) {
    if (n == 1) return 1;
    return n * f(n - 1);
}
```

n=4

```
int f(int n) {
    if (n == 1) return 1;
    return n * f(n - 1);
}
```

n=3

```
int f(int n) {
    if (n == 1) return 1;
    return n * f(n - 1);
}
```

计算5的阶乘

n=5

```
int f(int n) {
    if (n == 1) return 1;
    return n * f(n - 1);
}
```

n=4

```
int f(int n) {
    if (n == 1) return 1;
    return n * f(n - 1);
}
```

n=3

```
int f(int n) {
    if (n == 1) return 1;
    return n * f(n - 1);
}
```

n=2

```
int f(int n) {
    if (n == 1) return 1;
    return n * f(n - 1);
}
```

计算5的阶乘

n=5

```
int f(int n) {
    if (n == 1) return 1;
    return n * f(n - 1);
}
```

n=4

```
int f(int n) {
    if (n == 1) return 1;
    return n * f(n - 1);
}
```

n=3

```
int f(int n) {
    if (n == 1) return 1;
    return n * f(n - 1);
}
```

n=2

```
int f(int n) {
    if (n == 1) return 1;
    return n * f(n - 1);
}
```

计算5的阶乘

n=5

```
int f(int n) {
    if (n == 1) return 1;
    return n * f(n - 1);
}
```

n=4

```
int f(int n) {
    if (n == 1) return 1;
    return n * f(n - 1);
}
```

n=3

```
int f(int n) {
    if (n == 1) return 1;
    return n * f(n - 1);
}
```

n=2

```
int f(int n) {
    if (n == 1) return 1;
    return n * f(n - 1);
}
```

n=1

```
int f(int n) {
    if (n == 1) return 1;
    return n * f(n - 1);
}
```

计算5的阶乘

n=5

```
int f(int n) {
    if (n == 1) return 1;
    return n * f(n - 1);
}
```

n=4

```
int f(int n) {
    if (n == 1) return 1;
    return n * f(n - 1);
}
```

n=3

```
int f(int n) {
    if (n == 1) return 1;
    return n * f(n - 1);
}
```

n=2

```
int f(int n) {
    if (n == 1) return 1;
    return n * f(n - 1);
}
```

n=1 **Return 1**

```
int f(int n) {
    if (n == 1) return 1;
    return n * f(n - 1);
}
```

计算5的阶乘

n=5

```
int f(int n) {
    if (n == 1) return 1;
    return n * f(n - 1);
}
```

n=4

```
int f(int n) {
    if (n == 1) return 1;
    return n * f(n - 1);
}
```

n=3

```
int f(int n) {
    if (n == 1) return 1;
    return n * f(n - 1);
}
```

n=2

```
int f(int n) {
    if (n == 1) return 1;
    return n * f(n - 1);
}
```

Return 1

计算5的阶乘

n=5

```
int f(int n) {
    if (n == 1) return 1;
    return n * f(n - 1);
}
```

n=4

```
int f(int n) {
    if (n == 1) return 1;
    return n * f(n - 1);
}
```

n=3

```
int f(int n) {
    if (n == 1) return 1;
    return n * f(n - 1);
}
```

n=2 **Return 2**

```
int f(int n) {
    if (n == 1) return 1;
    return n * f(n - 1);
}
```

计算5的阶乘

n=5

```
int f(int n) {
    if (n == 1) return 1;
    return n * f(n - 1);
}
```

n=4

```
int f(int n) {
    if (n == 1) return 1;
    return n * f(n - 1);
}
```

n=3

```
int f(int n) {
    if (n == 1) return 1;
    return n * f(n - 1);
}
```

Return 2

计算5的阶乘

n=5

```
int f(int n) {
    if (n == 1) return 1;
    return n * f(n - 1);
}
```

n=4

```
int f(int n) {
    if (n == 1) return 1;
    return n * f(n - 1);
}
```

n=3 **Return 6**

```
int f(int n) {
    if (n == 1) return 1;
    return n * f(n - 1);
}
```

计算5的阶乘

n=5

```
int f(int n) {
    if (n == 1) return 1;
    return n * f(n - 1);
}
```

n=4

```
int f(int n) {
    if (n == 1) return 1;
    return n * f(n - 1);
}
```

Return 6

计算5的阶乘

n=5

```
int f(int n) {
    if (n == 1) return 1;
    return n * f(n - 1);
}
```

n=4 **Return 24**

```
int f(int n) {
    if (n == 1) return 1;
    return n * f(n - 1);
}
```

计算5的阶乘

n=5

```
int f(int n) {
    if (n == 1) return 1;
    return n * f(n - 1);
}
```

Return 24

计算5的阶乘

n=5 **Return 120**

```
int f(int n) {
    if (n == 1) return 1;
    return n * f(n - 1);
}
```

计算5的阶乘

Return 120

递归函数：斐波那契

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

求f(3)的值

n=3

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

求f(3)的值

n=3

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

求f(3)的值

n=3

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

求f(3)的值

n=3

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

n=2

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

求f(3)的值

n=3

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

n=2

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

求f(3)的值

n=3

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

n=2 **1**

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

求f(3)的值

n=3

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

1

求f(3)的值

n=3

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

1

n=1

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

求f(3)的值

n=3

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

1

n=1

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

求f(3)的值

n=3

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

1

n=1 1

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

求f(3)的值

n=3

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

1 1

求f(3)的值

n=3 2

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

1 1

求f(3)的值

2

递归函数：欧几里得算法

整数 a，b 的最大公约数一般表示为 gcd(a，b)

终极奥义：gcd(a，b) = gcd(b，a%b)

证明1：b和a%b的最大公约数，是a和b的公约数

证明2：b和a%b的最大公约数也是a和b的最大公约数

欧几里得算法-证明1

欧几里得算法-证明2

递归函数设计的终极奥义

见见世面

```c
void quick_sort(int *arr, int l, int r) {
    if (r - l <= 2) {
        if (r - l <= 1) return ;
        if (arr[l] > arr[l + 1]) swap(arr[l], arr[l + 1]);
        return ;
    }
    // partition
    int x = l, y = r - 1, z = arr[l];
    while (x < y) {
        while (x < y && z <= arr[y]) --y;
        if (x < y) arr[x++] = arr[y];
        while (x < y && arr[x] <= z) ++x;
        if (x < y) arr[y--] = arr[x];
    }
    arr[x] = z;
    quick_sort(arr, l, x);
    quick_sort(arr, x + 1, r);
    return ;
}
```

见见世面

```c
int *buff;
void merge_sort(int *arr, int l, int r) {
    if (r - l <= 1) return ;
    int mid = (l + r) / 2;
    merge_sort(arr, l, mid);
    merge_sort(arr, mid, r);
    // merge
    int p1 = l, p2 = mid, k = 0;
    while (p1 < mid || p2 < r) {
        if (p2 == r || (p1 < mid && arr[p1] <= arr[p2])) {
            buff[k++] = arr[p1++];
        } else {
            buff[k++] = arr[p2++];
        }
    }
    for (int i = l; i < r; i++) arr[i] = buff[i - l];
    return ;
}
```

见见世面

```c
int dfs(int i, int t1, int t2, int t3, int n) {
    if (i > n) {
        if (total_ans) print_one_result(n);
        return 1;
    }
    int ans = 0;
    for (int t = t1; t; t -= (-t & t)) {
        int j = ind[-t & t];
        if ((t2 & (1 << (i + j - 1))) && (t3 & (1 << (i - j + n)))) {
            arr[i] = j;
            ans += dfs(i + 1, t1 ^ (1 << j),
                t2 ^ (1 << (i + j - 1)),
                t3 ^ (1 << (i - j + n)),
                n
            );
        }
    }
    return ans;
}
```

拿起武器：数学归纳法

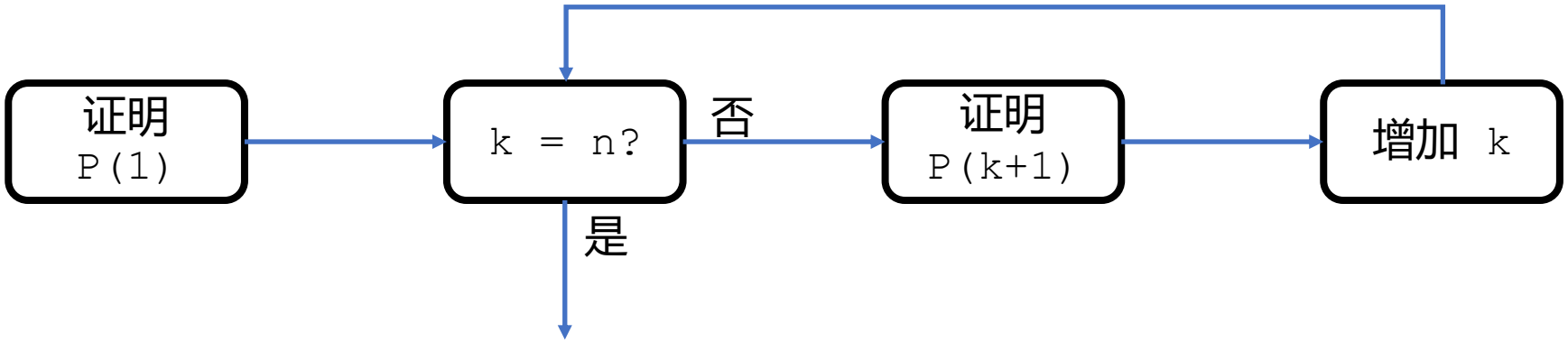
拿起武器：数学归纳法

Step1： 验证 P(1) 成立

Step2： 证明如果 P(k) 成立，那么 P(k+1) 也成立

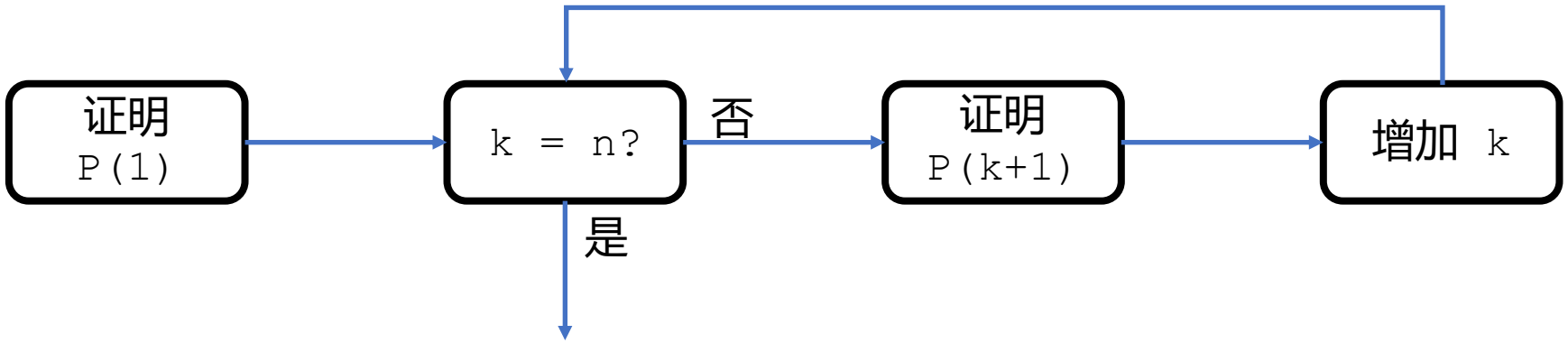
Step3： 联合 Step1 与 Step2，证明由 P(1)->P(n)成立

拿起武器：数学归纳法



拿起武器：数学归纳法

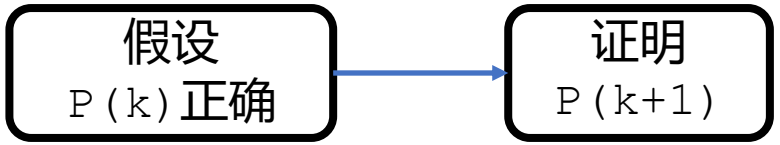
证明： $1 + 3 + \ldots + (2n - 1) = n^2$



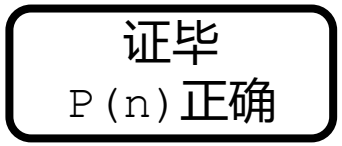
拿起武器：数学归纳法

第一步

第二步



第三步



拿起武器：数学归纳法

证明： $1 + 3 + \ldots + (2n - 1) = n^2$

拿起武器：数学归纳法

证明：$1 + 3 + \ldots + (2n - 1) = n^2$

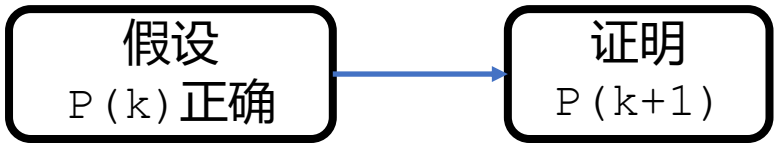
第一步

证明
P(1)

拿起武器：数学归纳法

证明：$1 + 3 + \ldots + (2n - 1) = n^2$

第二步



拿起武器：数学归纳法

证明：$1 + 3 + \ldots + (2n - 1) = n^2$

第三步

证毕
P(n)正确

拿起武器：数学归纳法

证明：如下程序的正确性

```
20 int main() {
21     int sum = 0;
22     for (int i = 1; i <= 100; i++) {
23         sum += i;
24     }
25     cout << sum << endl;
26     return 0;
27 }
```

拿起武器：数学归纳法

数学归纳法　　**结构**归纳法

递归函数设计的三个重要部分

递归函数设计的三个重要部分

1. **重要：**给『递归函数』一个明确的语义

2. 实现边界条件时的程序逻辑

3. 假设递归函数调用返回结果是正确的，实现本层函数逻辑

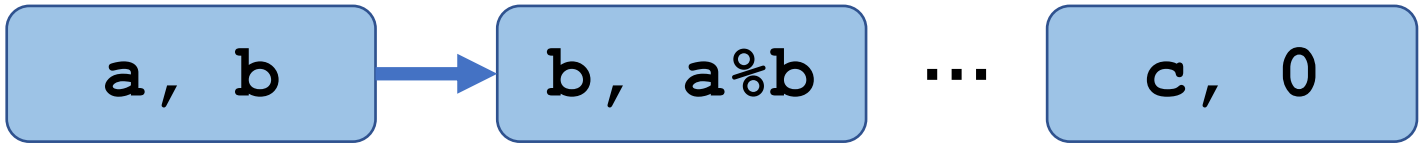
扩展欧几里得算法

贝 祖 等 式

$$ax + by = \gcd(a, b) = c$$

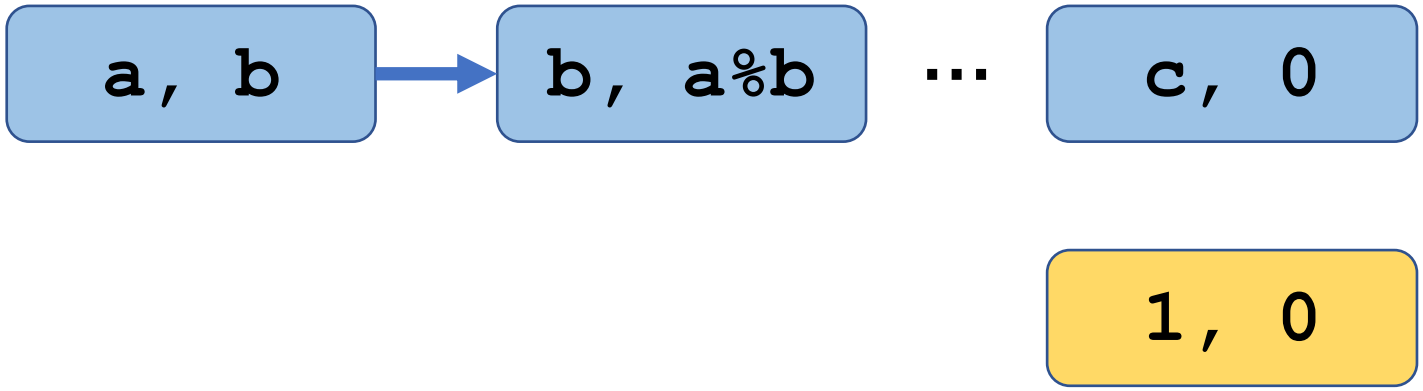
$$ax + by = \gcd(a, b) = c$$

GCD 过程



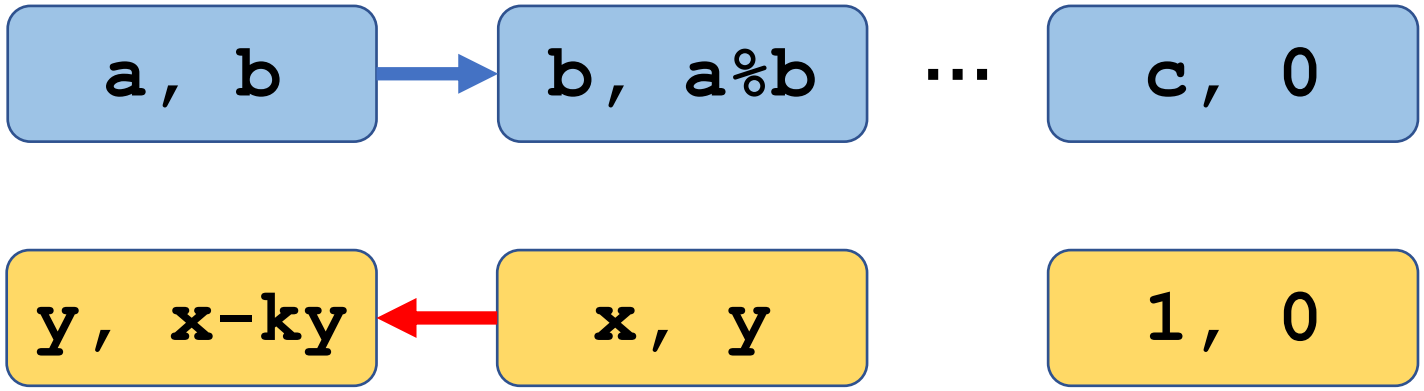
$$ax + by = \gcd(a, b) = c$$

GCD 过程



$$ax + by = \gcd(a, b) = c$$

GCD 过程



随堂练习题-1

请在主函数中实现从1打印到100。

要求：不使用循环，不使用额外的函数定义。

三. 函数的拓展知识

变参函数

变参函数

实现可变参数 max_int，从若干个传入的参数中返回最大值。

```
int max_int(int a, ...);
```

如何获得 a 往后的参数列表？ va_list 类型的变量
如何定位 a 后面第一个参数的位置？ va_start 函数
如何获取下一个可变参数列表中的参数？va_arg 函数
如何结束整个获取可变参数列表的动作？va_end 函数

变参函数

```c
#include <stdio.h>
#include <stdarg.h>

int max_int(int num, ...) {
    int ans = 0, temp;
    va_list arg;
    va_start(arg, num);
    while (num--) {
        temp = va_arg(arg, int);
        if (temp > ans) ans = temp;
    }
    va_end(arg);
    return ans;
}

int main() {
    printf("%d\n", max_int(3, 1, 5, 3));
    printf("%d\n", max_int(2, 1, 3));
    printf("%d\n", max_int(6, 6, 5, 3, 7, 9, 10));
    printf("%d\n", max_int(3, 1, 9, 10));
    return 0;
}
```

代码讲解：

第 13 行，定义一个代表参数列表的变量

第 14 行，初始化参数列表

第 15--18 行，循环读入 num 个参数，取出其中的最大值，放到 ans 变量中

第 19 行，销毁参数列表

主函数的参数

main 函数参数

```
int main();

int main(int argc, char *argv[]);

int main(int argc, char *argv[], char **env);
```

四. 函数-课后实战题

四、函数-课后实战题

1.HZOJ-464：统计闰年

2.HZOJ-465：数的分离

3.HZOJ-466：回文数的个数

4.HZOJ-467：求阶乘

5.HZOJ-468：最大公约数

6.HZOJ-185：斐波那契数列

7.HZOJ-183：递归函数

1.HZOJ-235：指数枚举

2.HZOJ-236：组合枚举

3.HZOJ-237：排列枚举

4.HZOJ-239：不规则的街道

不要考虑太多，坚持看完，

你就已经超过了95%的人。