中山大學
SUN YAT-SEN UNIVERSITY

函数和作用域 I

中山大学计算机学院

主讲人：黎卫兵　　**中山大学MOOC课程组**

目录

CONTENTS

大话三国

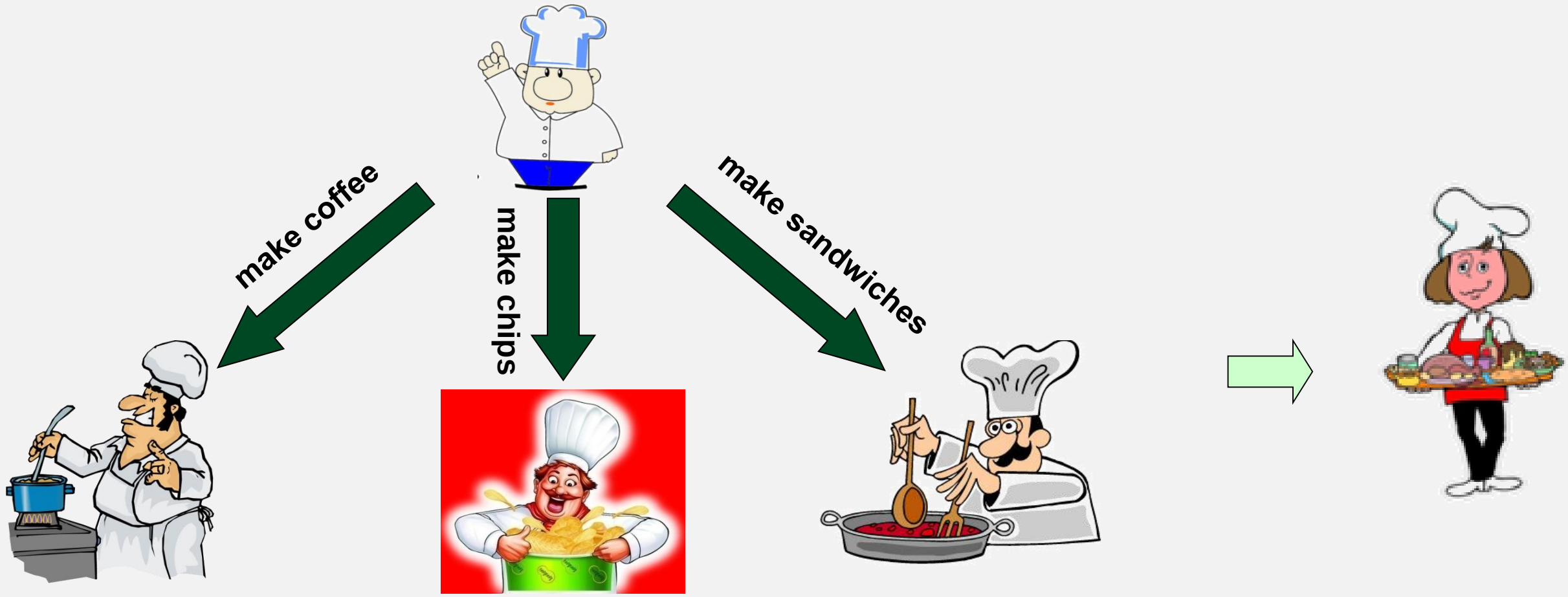
- 诸葛亮六出祁山之时，司马懿在上方谷固守不出，以逸待劳。诸葛亮派使臣送“女裙”激其出兵，但司马懿不为所动
- “诸葛亮”式的工作方式：事无巨细，事必躬亲，病故于五丈原

早餐

- 咖啡
- 薯条
- 三明治

管理学、经济学观点认为工作必须分工，各司其职

早餐

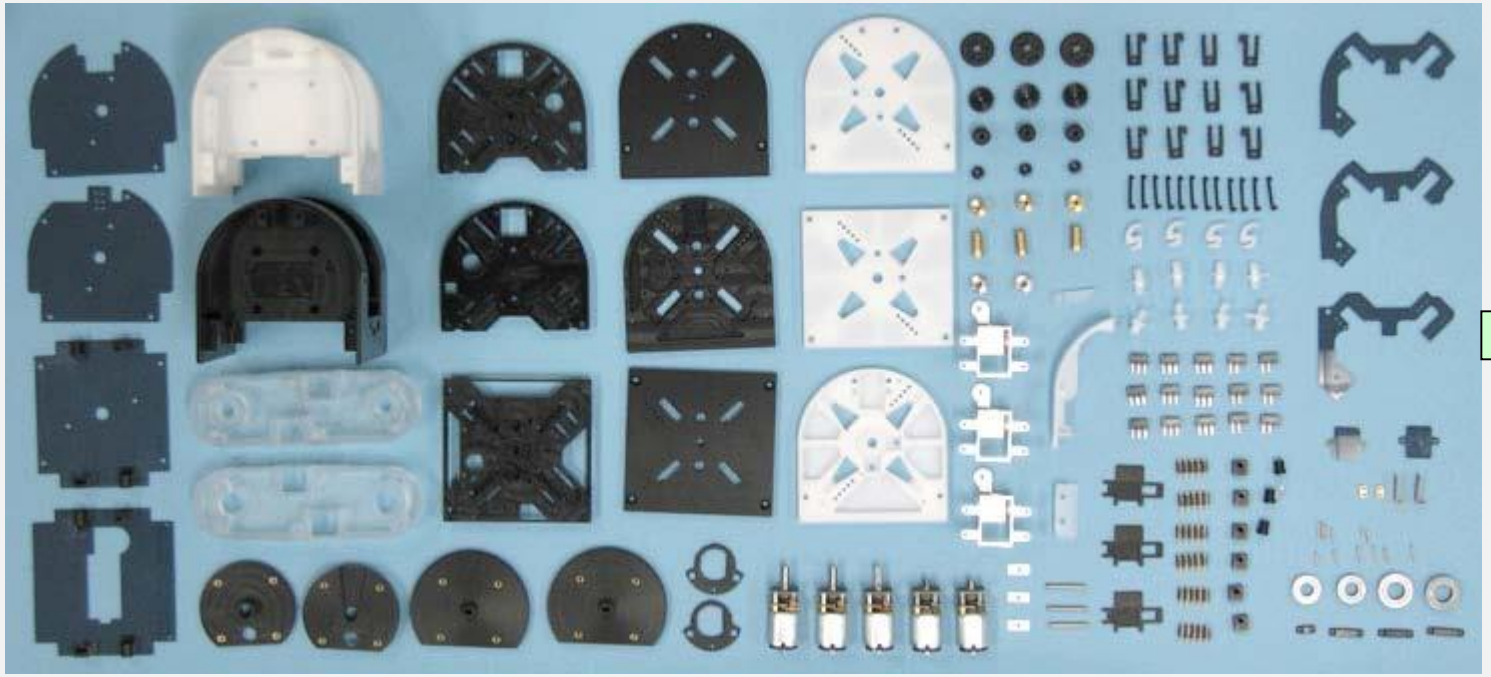


分工明确，各司其职，并行工作

模块化编程

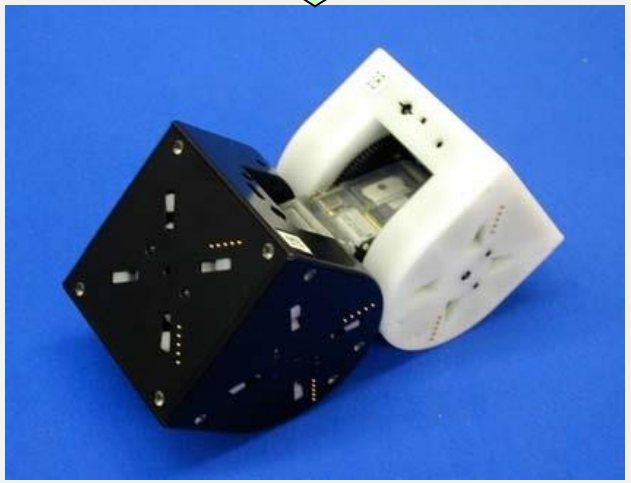
- 分而治之(Divide and Conquer, Wirth, 1971)
  - 把一个复杂的问题分解为若干个简单的问题，提炼出公共任务，把不同的功能分解到不同的模块中，逐个解决
  - 复杂问题求解的基本方法
  - 模块化编程的基本思想

模块机器人

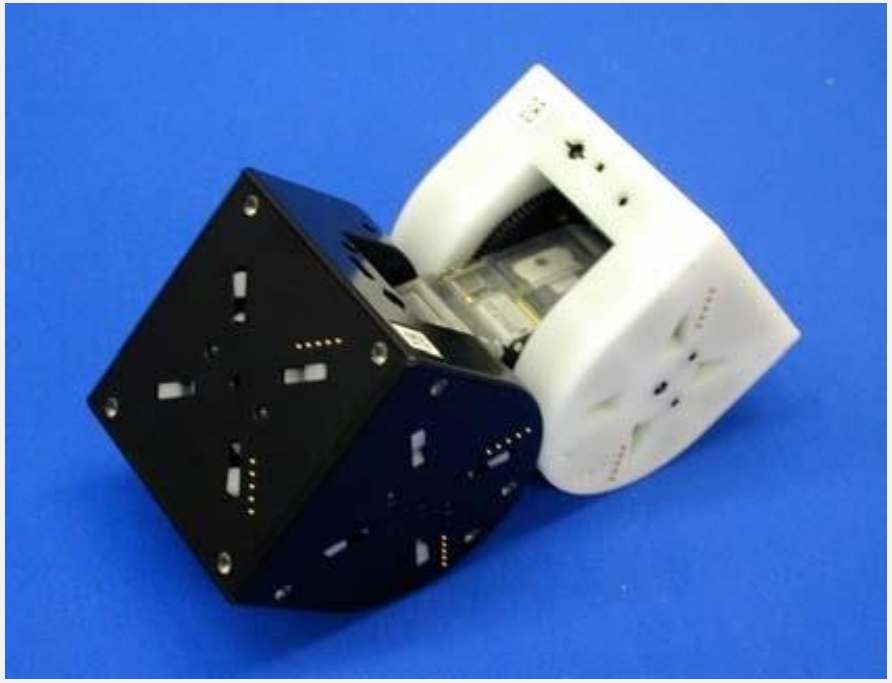


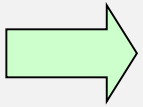
65 x 65 x 130 mm

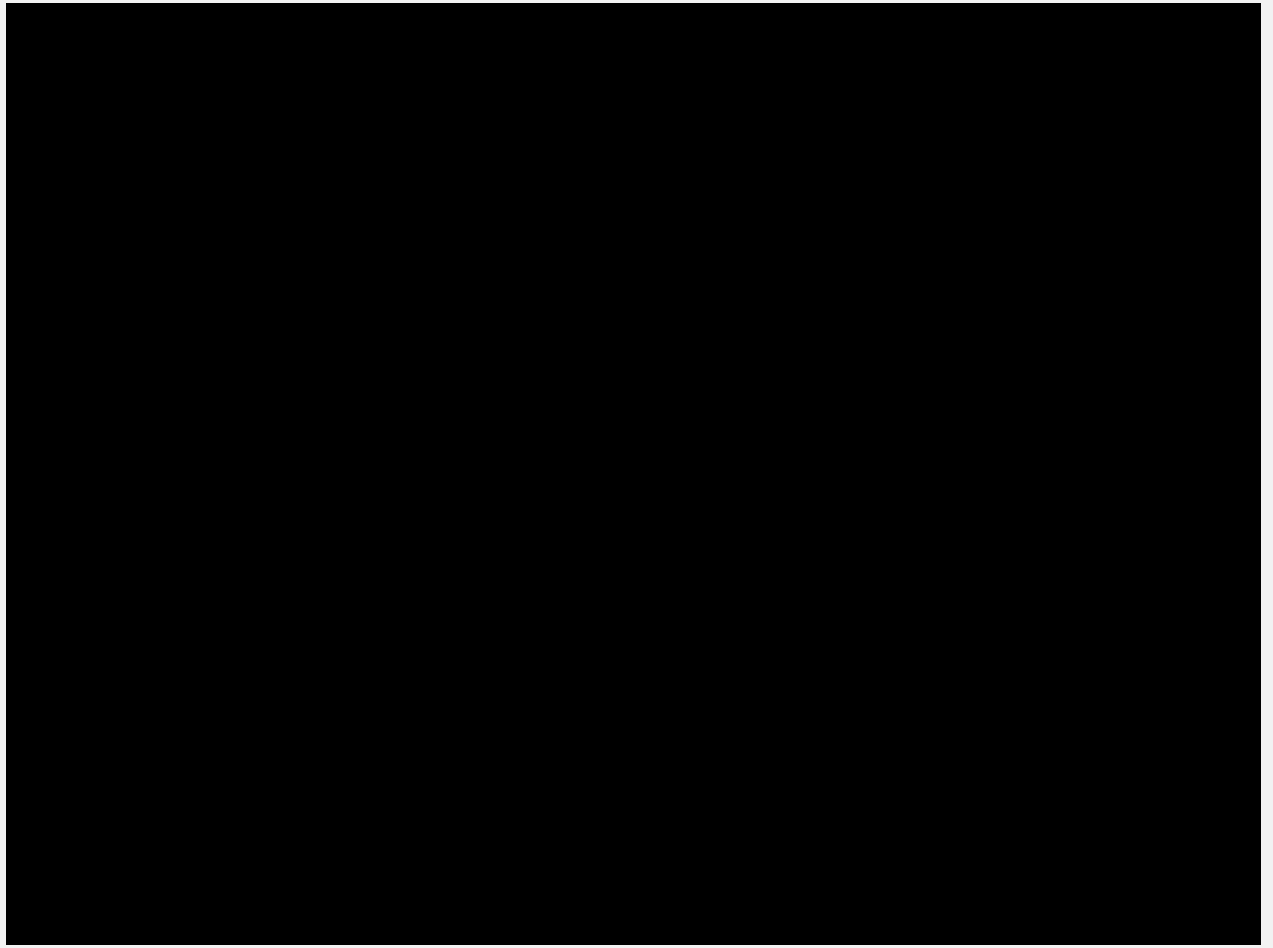
420 g



模块机器人







函数是什么

01

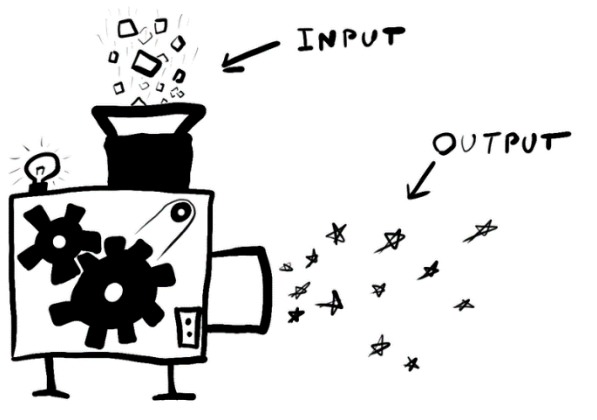
实现某个功能的**代码块**。

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

一个实现a+b功能的代码块

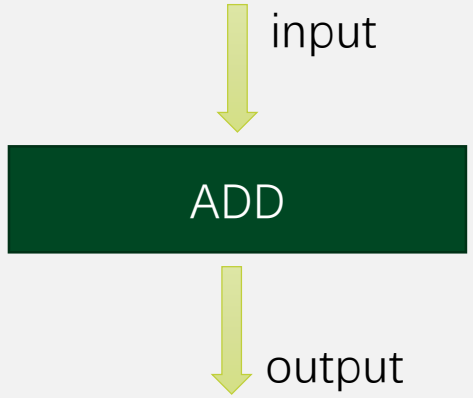
02

相当于一个机器，接受**输入**，对输入进行**处理**，给出**输出**。



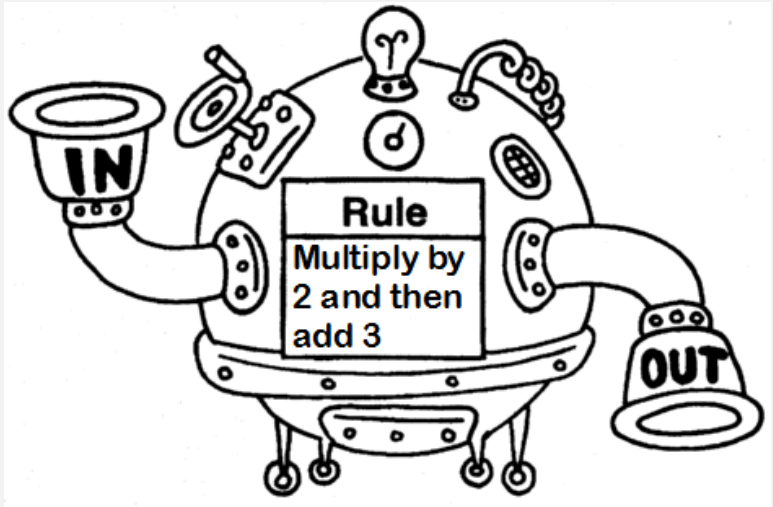
03

函数是一种抽象，它隐藏了算法实现的细节



函数有什么用 – 构建程序的积木块

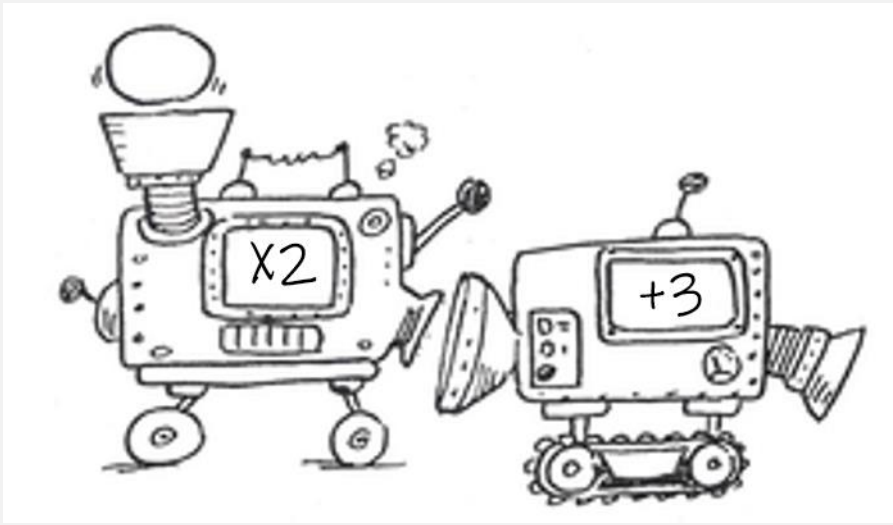
函数能够将一大块代码分解成若干个模块



一个大程序

分解

组装



分解成两个小函数

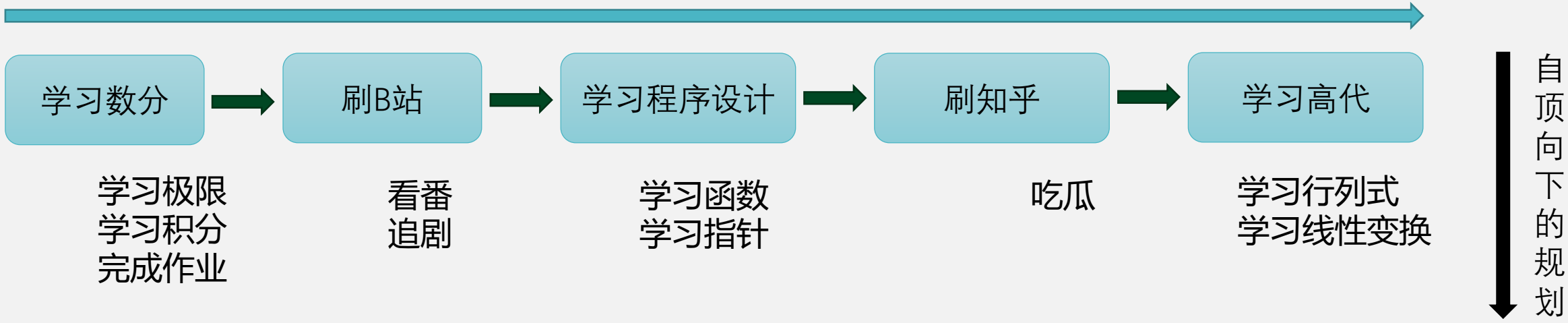
易于理解!
易于调试!

函数的优点

- 使代码的可重用性高!
- 能够以**模块化**的形式编程
- 代码优化，不必书写更多代码
- 容易Debug!

以模块化的形式对一天的生活进行规划!

时间

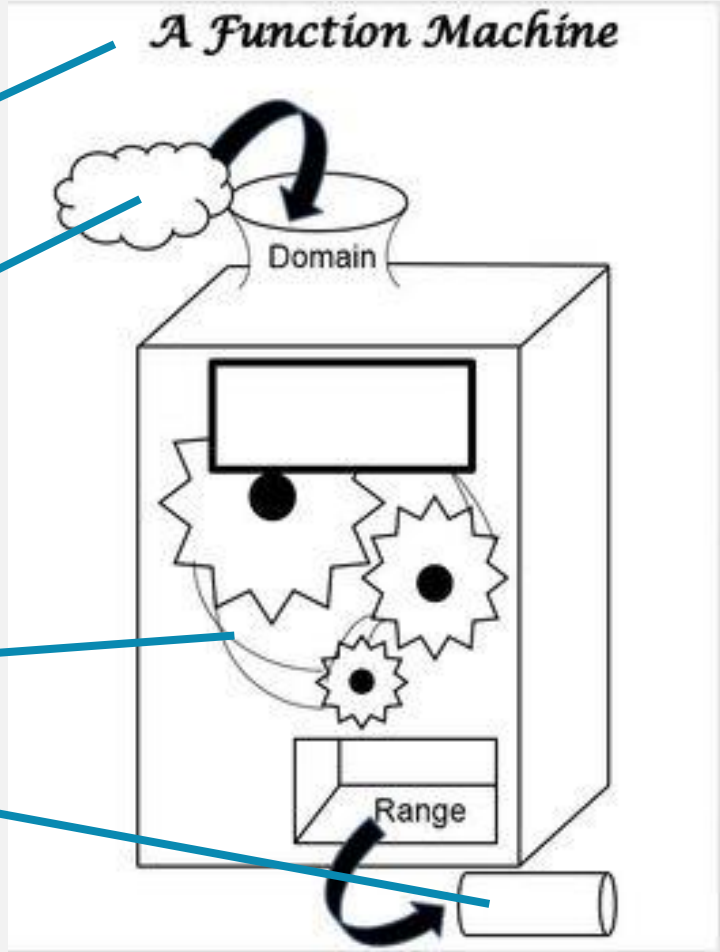


函数的声明与定义

函数的定义（Definitions）

- 给出函数名（function_name），
  输入参数（parameter），
  输出类型（return_type），
  函数主体（function body）。
- 函数主体即函数的内部逻辑代码

```
return_type function_name(parameter)
{
function body;
}
```



函数的声明与定义

函数的定义（Definitions）

- 给出函数名（function_name），
  输入参数（parameter），
  输出类型（return_type），
  函数主体（function body）。
- 函数主体即函数的内部逻辑代码

```
return_type  function_name(parameter)
{
function body;
}
```

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

函数名：add
输入参数：整型a，整型b
输出类型：整型
函数主体：c=a+b, 返回c

C语言中，函数名必须是唯一的！

函数的声明与定义

	数学中的函数
数学表达式	y = sin(x)
函数名	sin
自变量	x
因变量	y
功能	主要是计算

	程序设计中的函数
C语言表达式	y = sin(x)
函数名	sin
函数的参数	x
函数的返回值	y
功能	不局限于计算，还有判断推理

常用数学函数 – 要求掌握

- 在标头 <stdlib.h> 定义
  - `int abs( int n );`
- 定义于头文件 <math.h>
  - `double fabs( double arg );`
  - `double exp( double arg );`
  - `double log( double arg );`
  - `double log10( double arg );`
  - `double pow( double base, double exponent );`
  - `double sqrt( double arg );`
  - `double sin( double arg );`
    - `cos, tan, asin, acos, atan, atan2`
  - `double ceil( double arg );`
  - `double floor( double arg );`

常用数学函数 - cppreference.com

函数的声明与定义

函数的声明（Declaration）

- 给出函数名（function_name），
  输入参数（parameter），
  输出类型（return_type），
- 没有函数主体（function body）。
- 也叫函数的接口信息。

```
return_type  function_name(parameter);
```

```
int add(int a, int b);
```

函数名：add
输入参数：整型a，整型b
输出类型：整型
函数主体：未定义

Q：为何会有声明？直接定义不就好了吗？
A：声明的意义是告诉编译器函数信息，这样定义能在另一个地方书写。
虽然我还不会写，但先放一个函数在这里.jpg

声明函数只有一句话，可以把所有的函数声明放在单独一个文件里。
作为用户使用者，我们不关心函数是如何实现，我们只关心如何调用函数。而函数声明就只保留了这些信息。

函数的声明与定义

函数的声明（Declaration）

```
double __cdecl sin(double _X);
double __cdecl cos(double _X);
double __cdecl tan(double _X);
double __cdecl sinh(double _X);
double __cdecl cosh(double _X);
double __cdecl tanh(double _X);
double __cdecl asin(double _X);
double __cdecl acos(double _X);
double __cdecl atan(double _X);
double __cdecl atan2(double _Y,double _X);
double __cdecl exp(double _X);
double __cdecl log(double _X);
double __cdecl log10(double _X);
double __cdecl pow(double _X,double _Y);
double __cdecl sqrt(double _X);
double __cdecl ceil(double _X);
double __cdecl floor(double _X);
```

一目了然！
一眼知道如何调用！
舒心！

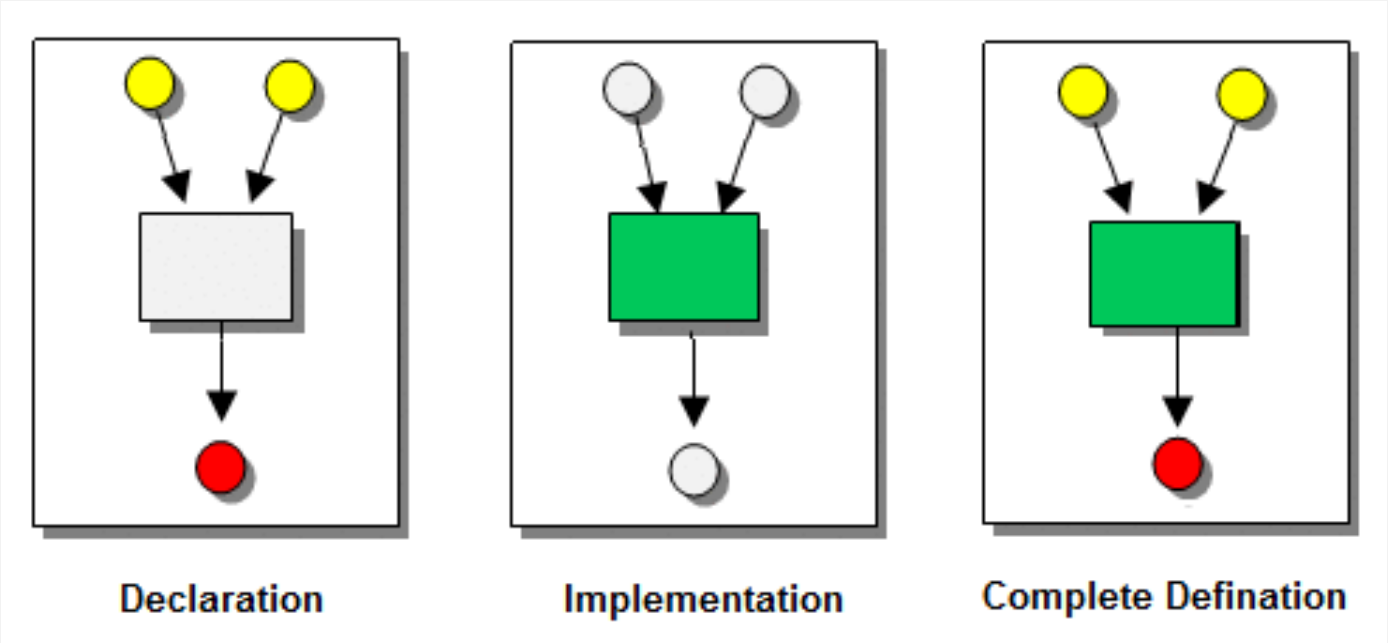
C语言自带的标准库函数

```
__mingw_ovr
__attribute__((__format__ (gnu_scanf, 1, 2))) __MINGW_ATTRIB_NONNULL(1)
int scanf(const char *__format, ...)
{
  int __retval;
  __builtin_va_list __local_argv; __builtin_va_start( __local_argv, __format );
  __retval = __mingw_vfscanf( stdin, __format, __local_argv );
  __builtin_va_end( __local_argv );
  return __retval;
}

__mingw_ovr
__attribute__((__format__ (gnu_scanf, 2, 3))) __MINGW_ATTRIB_NONNULL(2)
int fscanf(FILE *__stream, const char *__format, ...)
{
  int __retval;
  __builtin_va_list __local_argv; __builtin_va_start( __local_argv, __format );
  __retval = __mingw_vfscanf( __stream, __format, __local_argv );
  __builtin_va_end( __local_argv );
  return __retval;
}
```

不关心函数怎么实现
需要在万千代码中辛苦找到函数接口
烦心！

函数的声明与定义

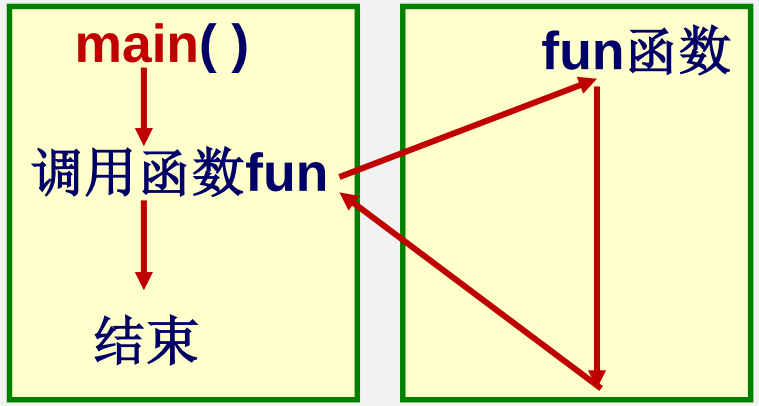
函数的定义与声明



C语言至少有一个函数：main函数！

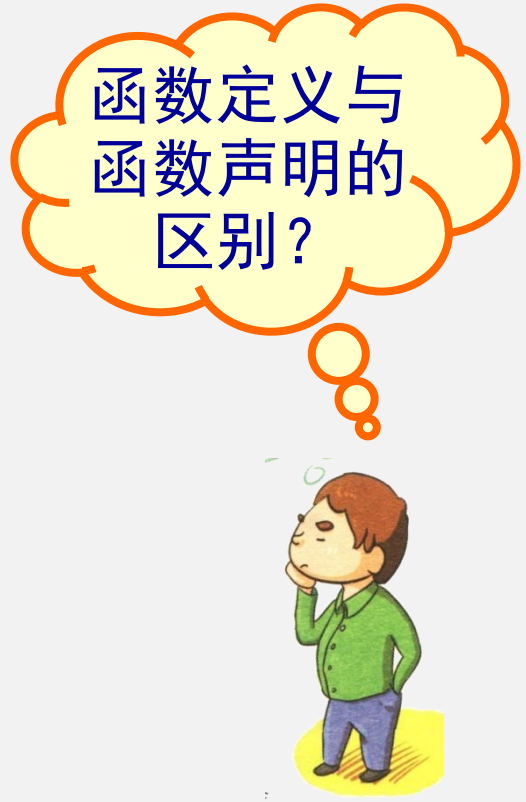
函数的声明与定义

- C程序的执行从main函数开始
- 调用其他函数后流程回到main函数
- 在main函数中结束整个程序的运行



函数的声明与定义

函数定义	函数声明
指函数功能的确立	对函数名、返回值类型、形参类型进行声明
有函数体	不包括函数体
是完整独立的单位	是一条语句，以分号结束，只起声明作用
编译器做实事	编译器对声明的态度是“我知道了”
分配内存，把函数装入内存	不申请内存，只保留一个引用，执行程序链接时，将正确的内存地址链接到那个引用上



C 程序与函数（Functions）

- C程序的基本结构
  - 程序由若干个函数构成
  - 必须有一个 `main` 函数
  - 函数不能重名
  - 函数**应该（should be）**先声明，后实现
- 如右程序，有三个函数
  - `main`
  - `Square`
  - `Cube`

```c
#include <stdio.h>

int Square( int );     // function declarations
int Cube( int );

int main() {
    printf( "The square of 3 is %d\n", Square(3));
    printf( "The cube of 3 is %d \n", Cube(3));
    return 0;
}

// function implementations
int Square(int n) {
    return n*n;
}

int Cube(int n) {
    return n*n*n;
}
```

请编译，修改，并运行程序 function-syntax.c

函数的形参与实参

- 形式参数/形参（Formal parameter）
  - 函数在声明和定义时用的参数
  - 参数定义时由于没有具体值，只是一个符号或可存储特定数值的空间
  - 声明时可仅声明参数类型
  - 定义时必须给出形式参数名称，便于函数体使用
- 实际参数/实参（Actual argument）
  - 函数在使用/调用时实际替代形式参数的值
  - 实际参数与声明的形式参数必须数量相同，类型兼容
  - 如果实参形参类型兼容但不一致，则按赋值规则发生强制转换

请修改 function-syntax.c 的实参为double，如3.14，并运行程序

函数的调用

函数根据参数和返回值可以分为四类：

无参数，无返回值

```
void add(void){
    ...
    ...
}
```

void
空，无

有参数，无返回值

```
void add(int a){
    ...
    ...
}
```

无参数，有返回值

```
int add(void){
    ...
    ...
    return 6;
}
```

有参数，有返回值

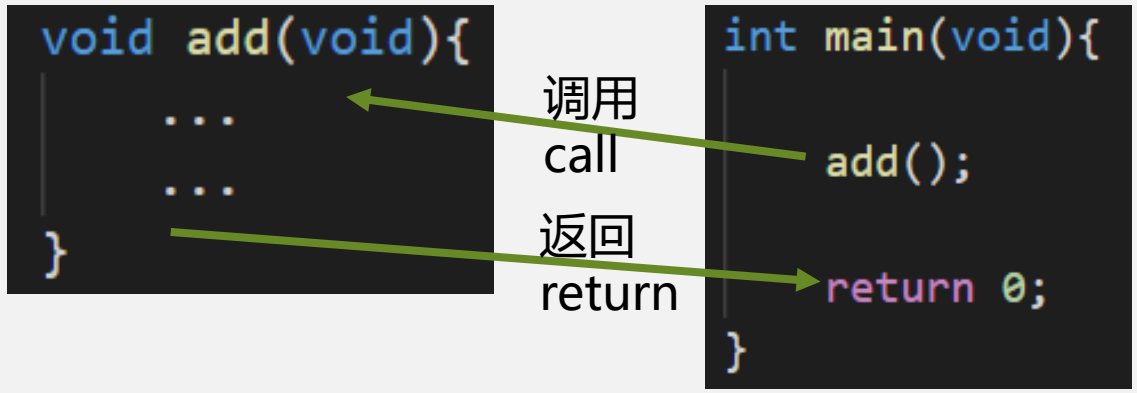
```
int add(int a){
    ...
    ...
    return 6;
}
```

这两者类型要一致

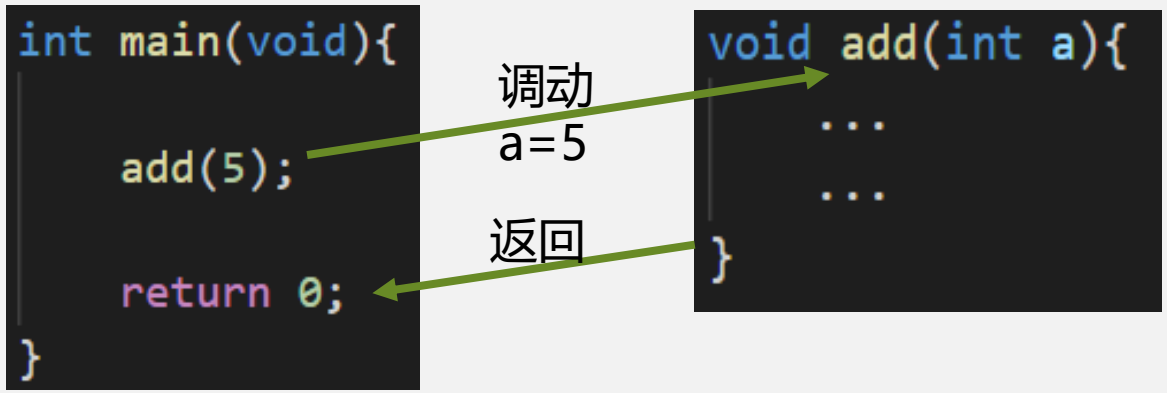
函数的调用

函数根据参数和返回值可以分为四类：

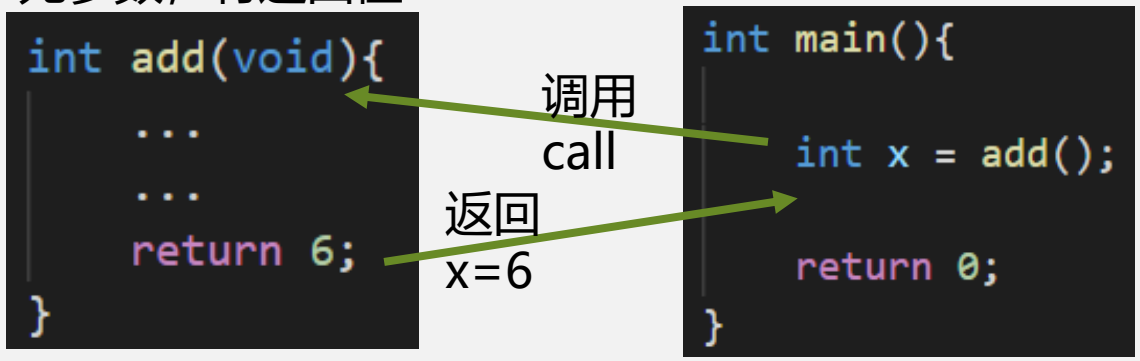
无参数，无返回值



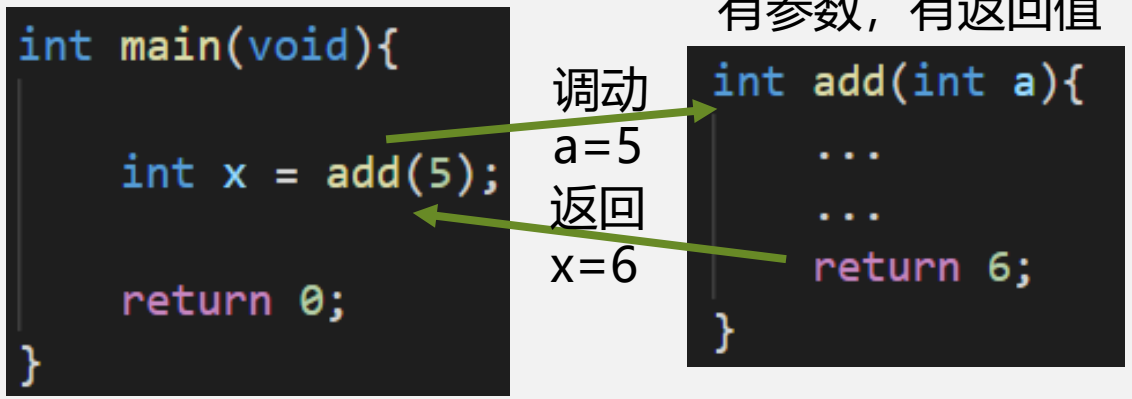
有参数，无返回值



无参数，有返回值



有参数，有返回值



函数的调用

函数根据参数和返回值可以分为四类：

无参数，无返回值

```
void add(void){
    ...
    ...
}
```

```
int main(void){

    add();

    return 0;
}
```

```
int main(void){

    add(5);

    return 0;
}
```

有参数，无返回值

```
void add(int a){
    ...
    ...
}
```

无参数，有返回值

```
int add(void){
    ...
    ...
    return 6;
}
```

```
int main(){

    int x = add();

    return 0;
}
```

```
int main(void){

    int x = add(5);

    return 0;
}
```

两者类型一致

有参数，有返回值

```
int add(int a){
    ...
    ...
    return 6;
}
```

函数的调用

```c
int main(void){

    int y = 5;

    int x = add(y);

    return 0;
}
```

a=5

x=6

```c
int add(int a){
    ...
    a = 9;
    return 6;
}
```

调用add函数后
y=5 or 9？

函数的调用

```c
int main(void){

    int y = 5;

    int x = add(y);

    return 0;
}
```

a=5

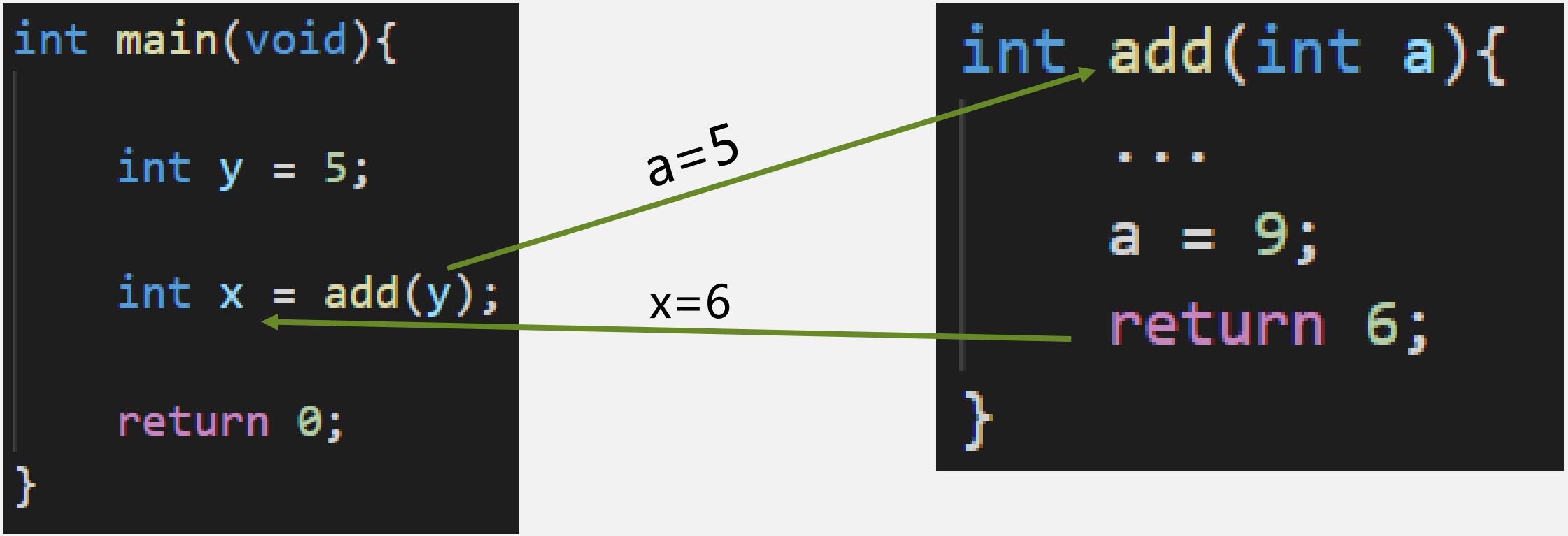
x=6

```c
int add(int a){
    ...
    a = 9;
    return 6;
}
```

调用add函数后
y=5!

(不是5的阶乘，是5 感叹号.....

函数的调用

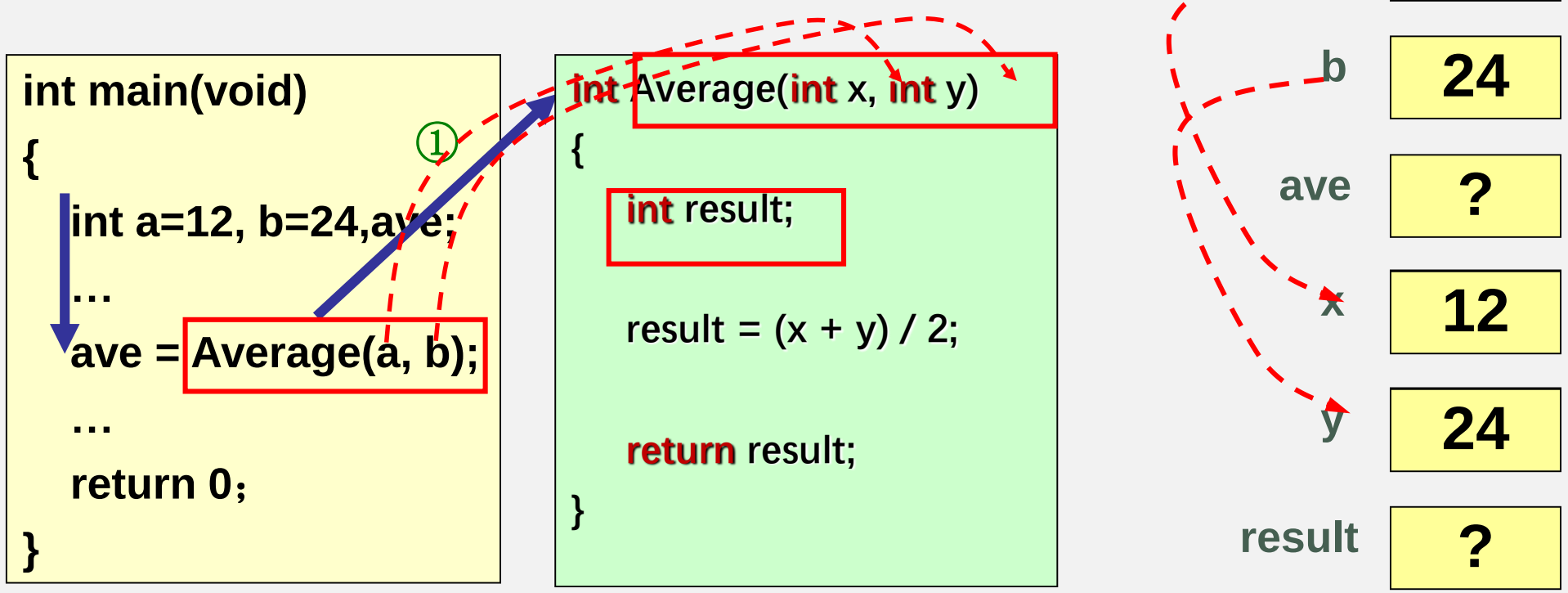


函数参数的按值传递（Call By Value）

main函数里的变量y和add函数里的变量a不是同一个变量，尽管它们的值一样
add函数里无论对变量a作什么操作，不会影响到main函数里的变量y。

函数的调用

- 每次执行函数调用时
  - 现场保护并为函数的内部变量（包括形参）分配内存
  - 把实参值复制给形参，单向传值（实参→形参）
  - 实参与形参数目一致，类型匹配（否则类型自动转换）

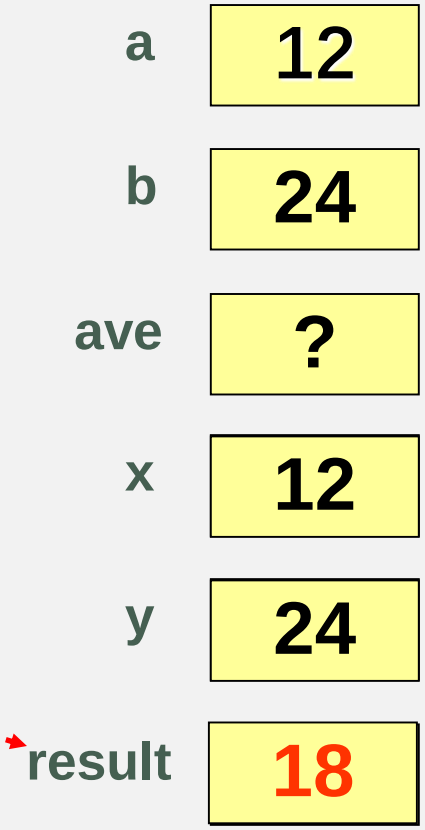


函数的调用

- 执行函数内的语句
- 当执行到return语句或}时，从函数退出

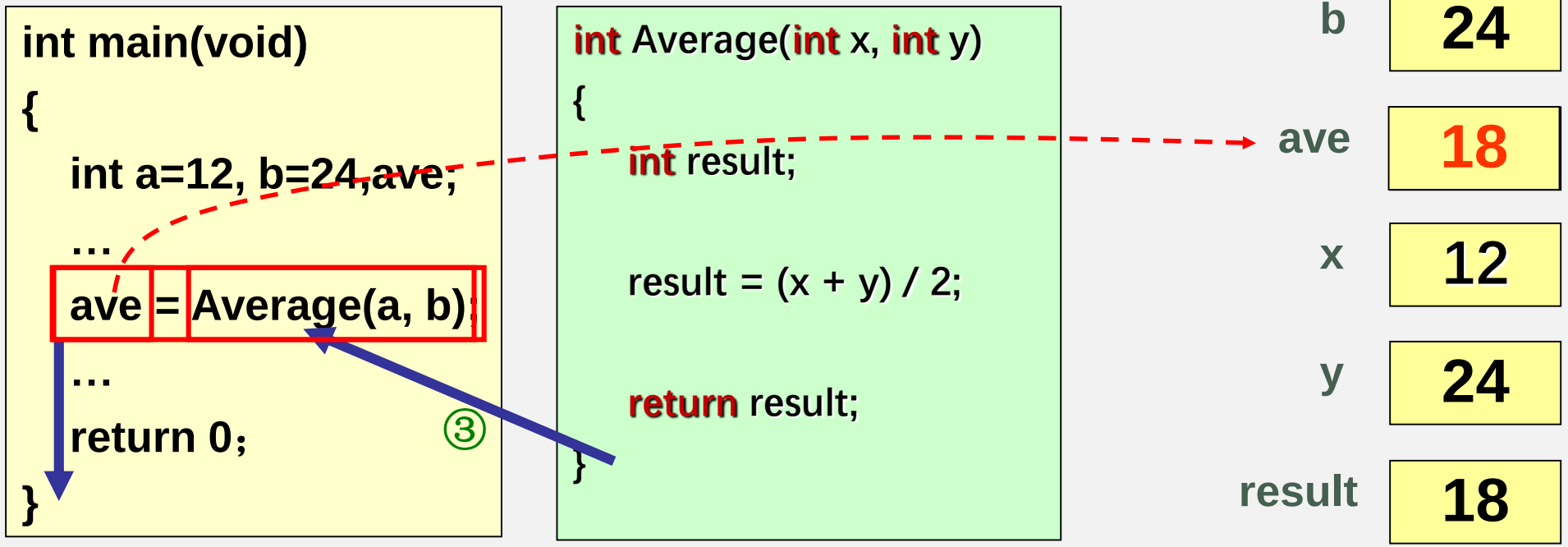
```
int main(void)
{
    int a=12, b=24,ave;
    …
    ave = Average(a, b);
    …
    return 0;
}
```

```
int Average(int x, int y)
{
    int result;
②
    result = (x + y) / 2;

    return result;
}
```



函数的调用

- 从函数退出时
  - 根据函数调用栈中保存的返回地址，返回到当次函数调用的地方
  - 程序控制权交给调用者，返回值作为函数调用表达式的值
  - 收回分配给函数内所有变量（包括形参）的内存



函数的调用

Q：我想对变量a的修改能影响到变量y怎么办？比如我想写一个函数swap(a,b)用来交换a，b这两个变量的值……
A：那就用到另一种传递方式，叫引用传递（Call By Reference）

```
int main(){

    int a = 1, b = 2;

    swap(&a, &b);

    return 0;
}
```

地址传递

指针类型

```
void swap(int *x, int *y){
    int t = *x;
    *x = *y;
    *y = t;
}
```

由于变量是储存在计算机内存的某个地方，我们用**地址**来描述。
如果两个变量名指向同一个内存地方，
那么对该内存地址的操作就能影响到其他变量的值

函数的调用 – 小结与练习

- 函数是一个初等表达式
  - 函数返回值的类型就是表达式的类型
  - 函数遇到 `return` 语句或函数块执行完毕函数表达式完成求值
  - 除了返回 `void` 类型，必须使用 `return` 语句返回一个值
- 函数实参与形参
  - 形参是传引用则实参必须是“&左值表达式”或“地址”
  - 形参是传值是实参必须是类型兼容表达式
  - 实参与形参必须数量一致，类型兼容
  - 函数调用时实参表达式按从右至左顺序赋予形参

请运行 call-byref-swap.c 体验传值和引用的区别

函数的调用 – 小结与练习

- 函数参数表达式求值顺序

```
/*function parameter expression*/
#include <stdio.h>

void testfun(int a, int b, int c){
    printf("%d,%d,%d\n", a, b, c);
}

int main() {
    int a = 0;
    testfun(a++,a++,a++);
}
```

请问程序输出是什么?

变量的作用域

变量的作用域（scope），指能够访问（access）该变量的**范围**，可以是全局，可以是函数，可以是for循环，也可以是任意用大括号{}括起来的**代码块**。

```
int main(void){

    int y = 5;

    int x = add(y);

    return 0;
}
```

在main函数里无法访问到add函数里的变量a，但能访问到变量x。

在add函数里，无法访问到main函数里的变量y，但能访问到变量a。

```
int add(int a){

    ...

    a = 9;

    return 6;
}
```

变量x的作用域是main函数
变量a的作用域是add函数

变量的作用域

根据作用域的范围，变量可以分为三类

位置	种类
在函数或代码块里	局部变量 (local variables)
在所有函数外面	全局变量 (Global variables)
在函数参数列表里	形式参数 (Formal parameters)

变量的作用域：局部变量

局部变量在函数（块）内声明，仅在该函数（块）内能够访问到。

一个代码块

```
void add(void){
    int a;
    a = 1;
    {
        int b = 2;
        b = a;
    }
    a = b; //error
    {
        int c = b;  //error
    }
}
```

另一个代码块

其实函数和代码块都是由大括号{}包裹起来的

在该大括号内声明的变量仅在该大括号内可以访问

变量b在内部大括号内声明，外部访问不了
但内部大括号可以访问外部大括号的变量a，
但访问不了同级大括号内的变量

变量的作用域：局部变量

局部变量在函数（块）内声明，仅在该函数（块）内能够访问到。

一个代码块

```
void add(void){
    int a;
    a = 1;
    {
        int b = 2;
        b = a;
    }
    a = b; //error
    {
        int c = b;  //error
    }
}
```

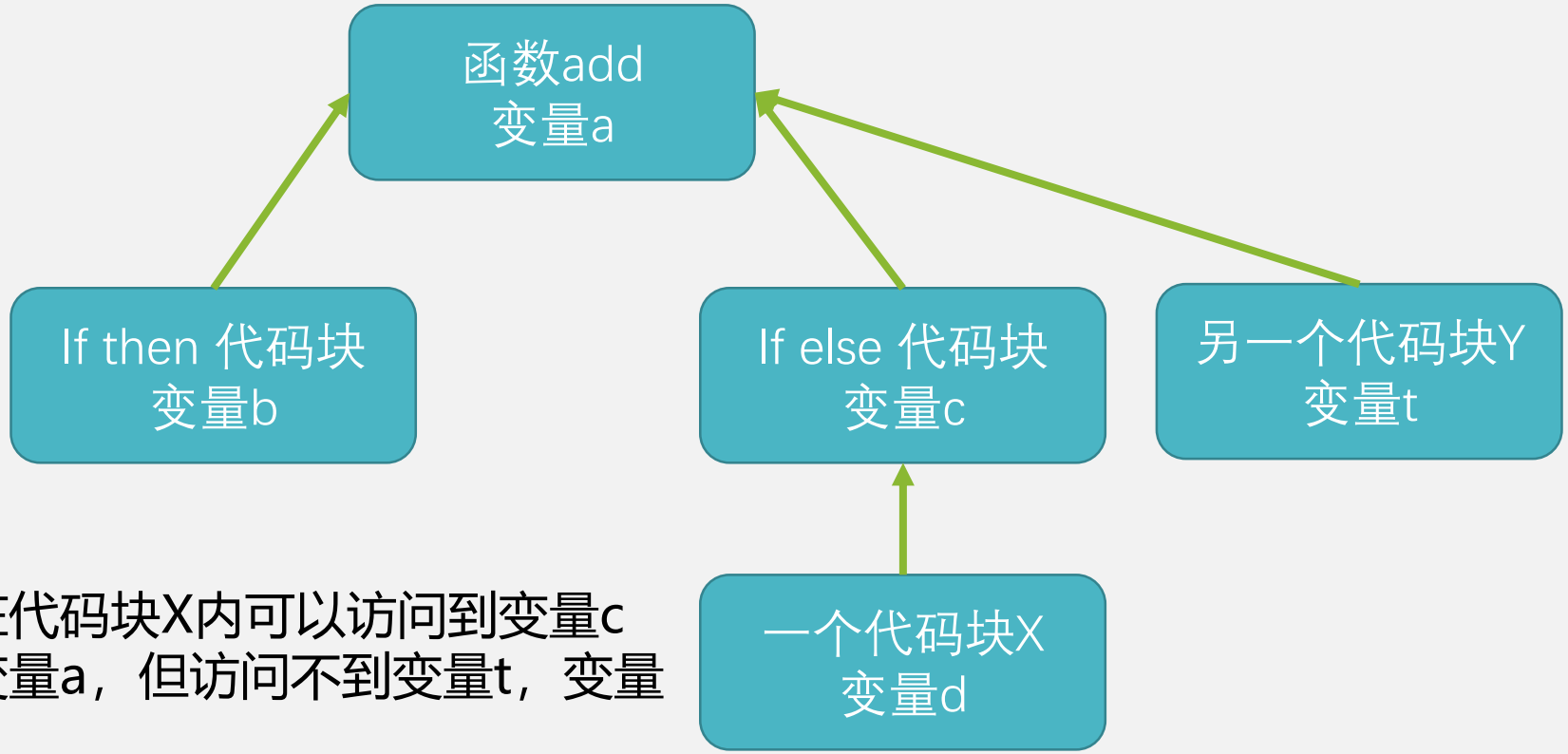
另一个代码块

在一个域内，可以访问包含其域的块（父亲）内变量
局部变量的生命周期也仅在这个块内
一旦当前执行的代码离开了这个块，在该块的变量就被销毁

变量的作用域：局部变量

稍复杂的例子

```
void add(void){
    int a = 0;
    if (a == 1){
        int b = 1;
    }else{
        int c = 2;
        {
            int d = a;
        }
        c = d; //error
    }
    a = 9;
    {
        int t = c; //error
    }
}
```

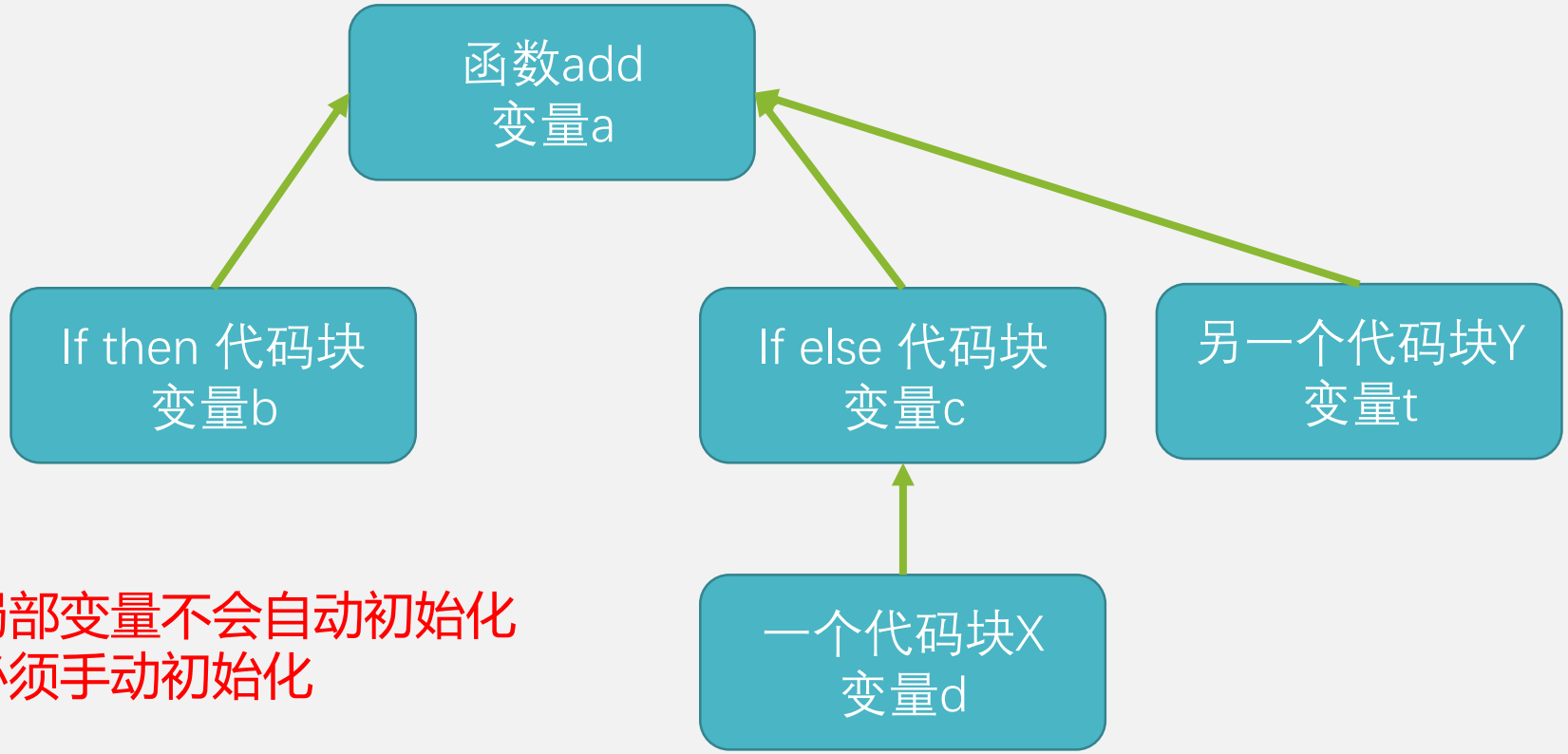


在代码块X内可以访问到变量c变量a，但访问不到变量t，变量b

变量的作用域：局部变量

稍复杂的例子

```
void add(void){
    int a = 0;
    if (a == 1){
        int b = 1;
    }else{
        int c = 2;
        {
            int d = a;
        }
        c = d; //error
    }
    a = 9;
    {
        int t = c; //error
    }
}
```



局部变量不会自动初始化
必须手动初始化

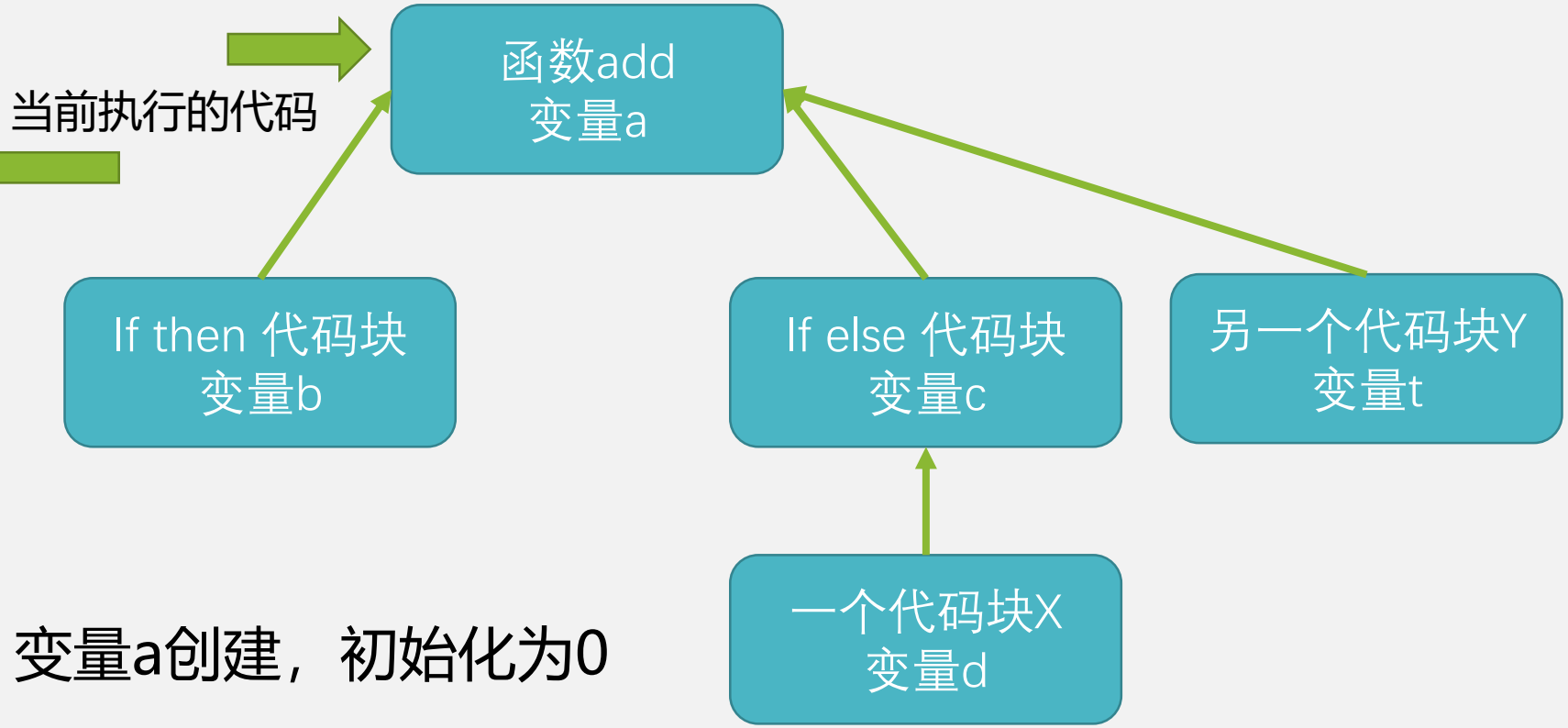
局部变量的生命周期仅在所在块内

代码执行离开了该块，其变量均被销毁

变量的作用域：局部变量

稍复杂的例子

```
void add(void){
    int a = 0;
    if (a == 1){
        int b = 1;
    }else{
        int c;
        {
            int d = a;
        }
        c = a;
    }
    a = 9;
    {
        int t = a;
    }
}
```

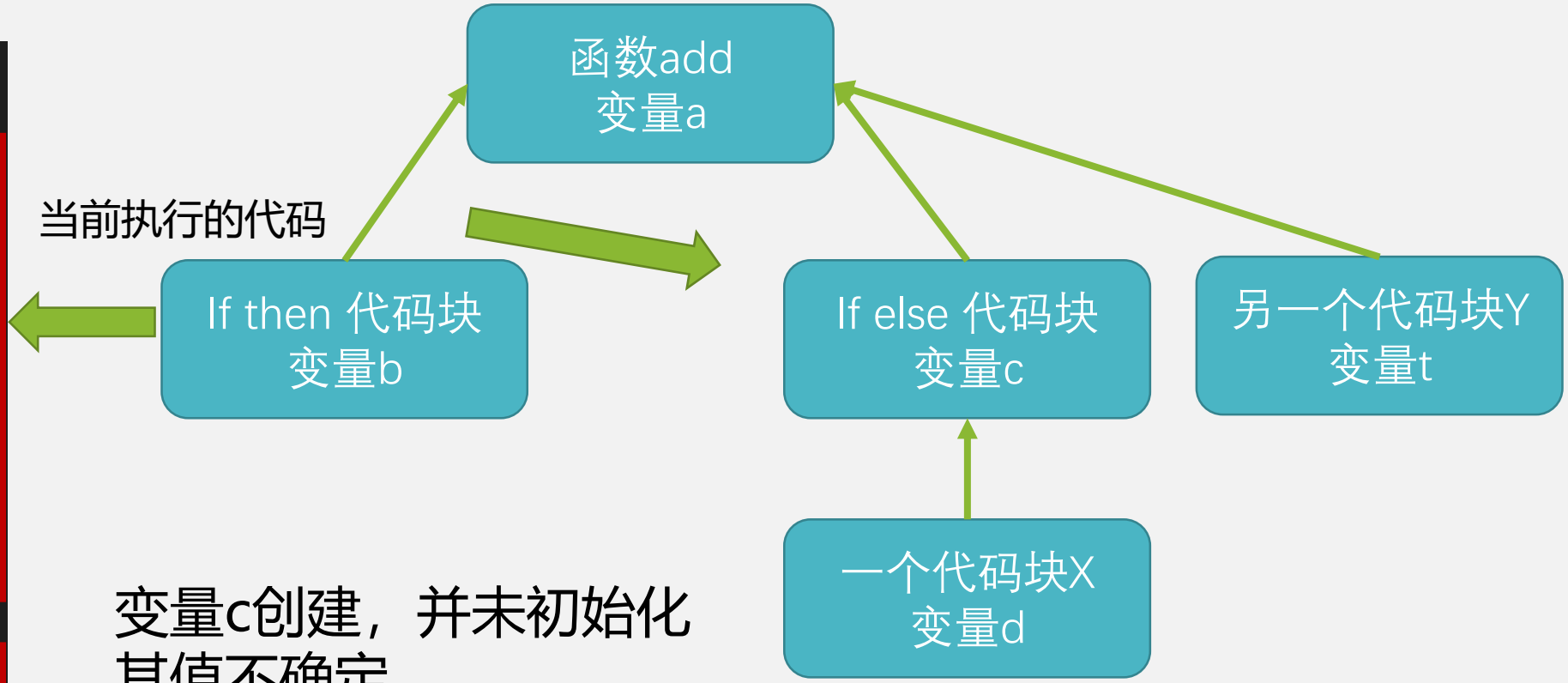


变量a创建，初始化为0

变量的作用域：局部变量

稍复杂的例子

```
void add(void){
    int a = 0;
    if (a == 1){
        int b = 1;
    }else{
        int c;
        {
            int d = a;
        }
        c = a;
    }
    a = 9;
    {
        int t = a;
    }
}
```

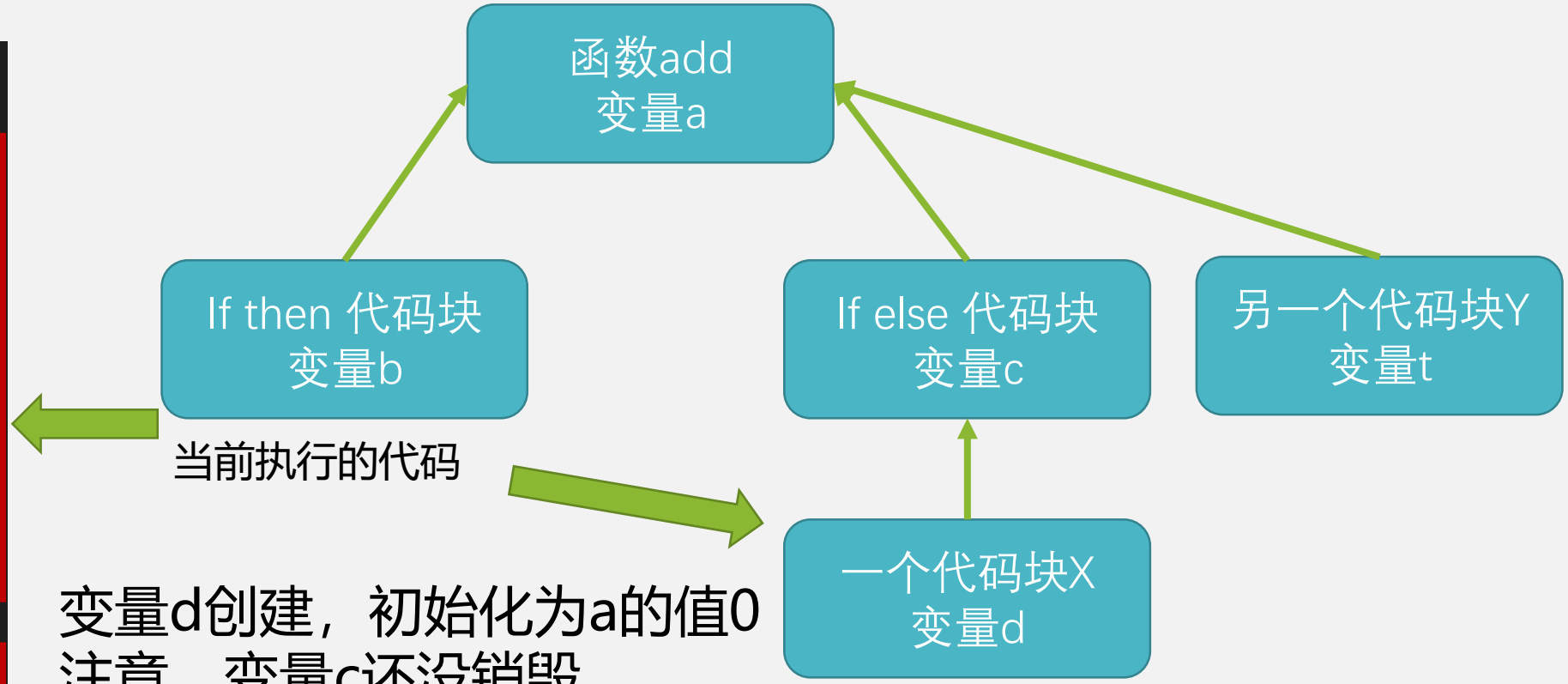


变量c创建，并未初始化
其值不确定。
注意，变量a不会销毁
因为此时还在函数块内

变量的作用域：局部变量

稍复杂的例子

```
void add(void){
    int a = 0;
    if (a == 1){
        int b = 1;
    }else{
        int c;
        {
            int d = a;
        }
        c = a;
    }
    a = 9;
    {
        int t = a;
    }
}
```

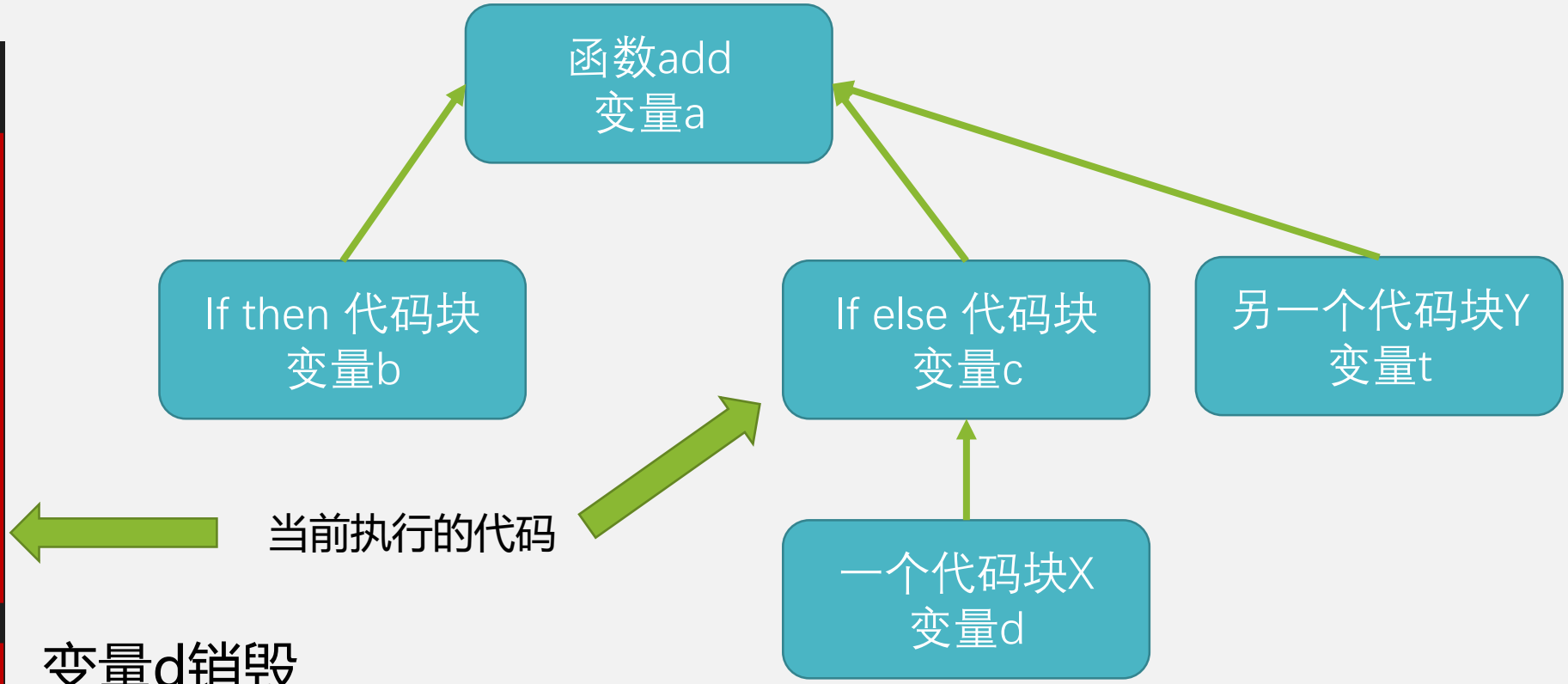


变量d创建，初始化为a的值0
注意，变量c还没销毁
因为此时还在if else块内

变量的作用域：局部变量

稍复杂的例子

```
void add(void){
    int a = 0;
    if (a == 1){
        int b = 1;
    }else{
        int c;
        {
            int d = a;
        }
        c = a;
    }
    a = 9;
    {
        int t = a;
    }
}
```

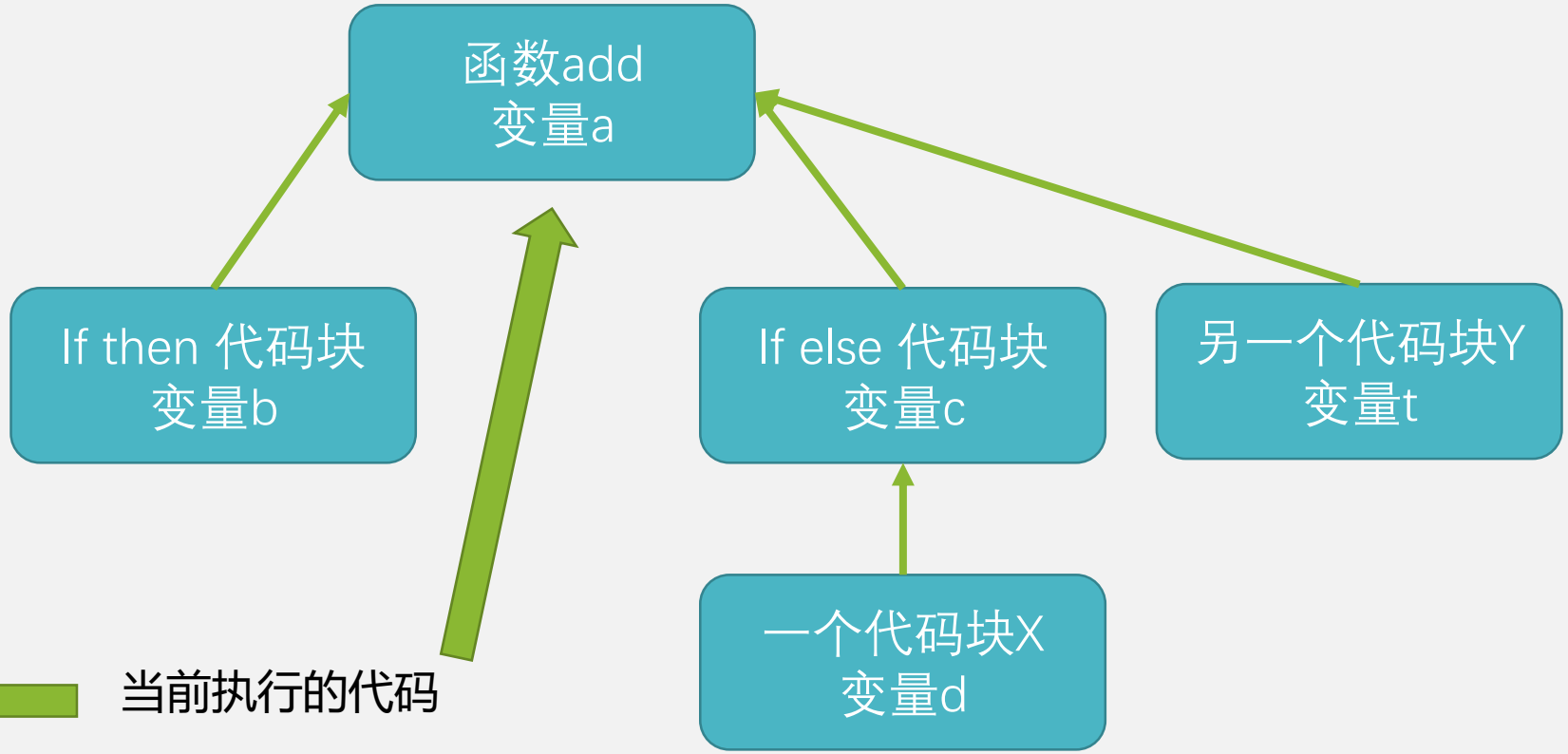


变量d销毁
因为此时已经离开了代码块X

变量的作用域：局部变量

稍复杂的例子

```
void add(void){
    int a = 0;
    if (a == 1){
        int b = 1;
    }else{
        int c;
        {
            int d = a;
        }
        c = a;
    }
    a = 9;
    {
        int t = a;
    }
}
```

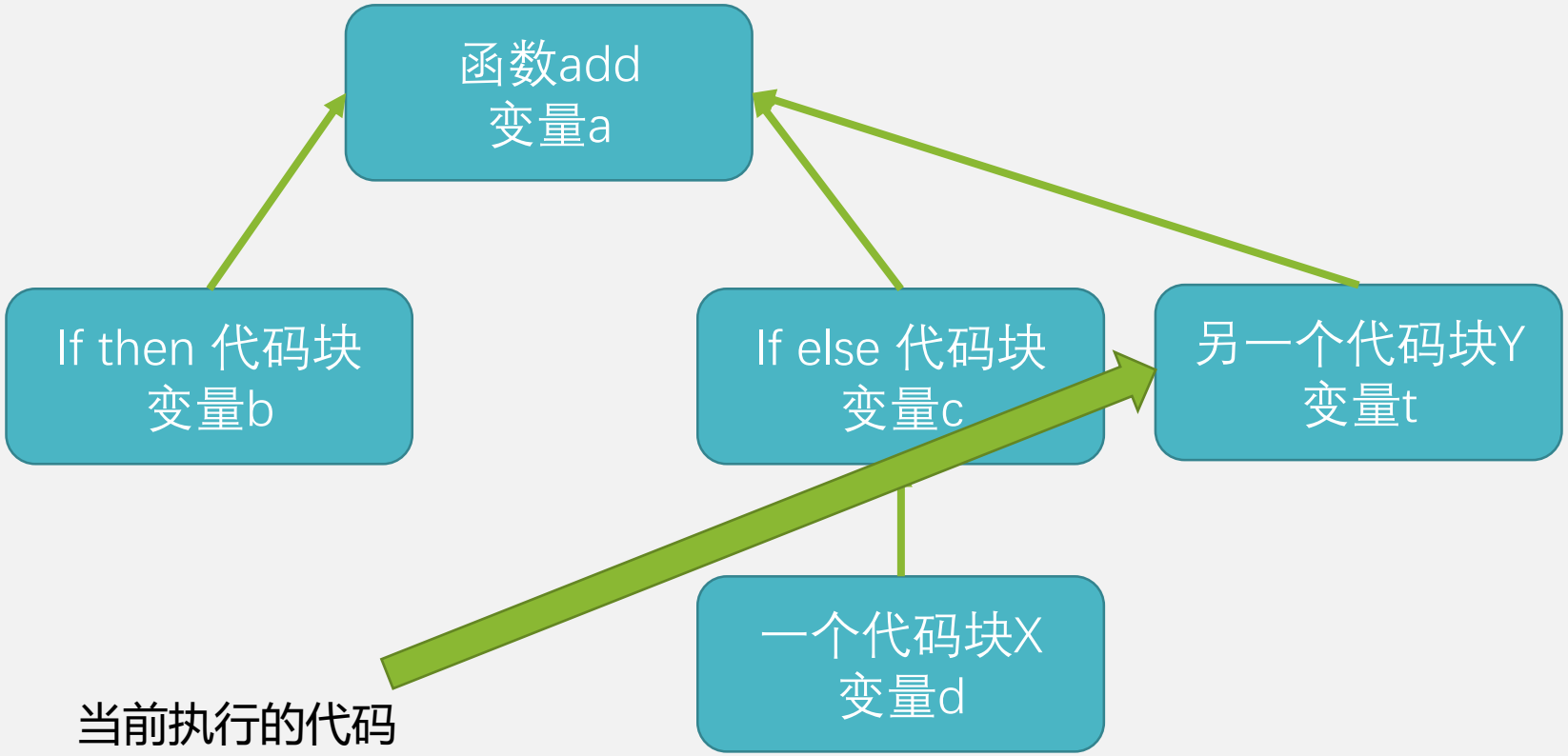


变量c销毁
因为此时已经离开了if else代码块

变量的作用域：局部变量

稍复杂的例子

```
void add(void){
    int a = 0;
    if (a == 1){
        int b = 1;
    }else{
        int c;
        {
            int d = a;
        }
        c = a;
    }
    a = 9;
    {
        int t = a;
    }
}
```

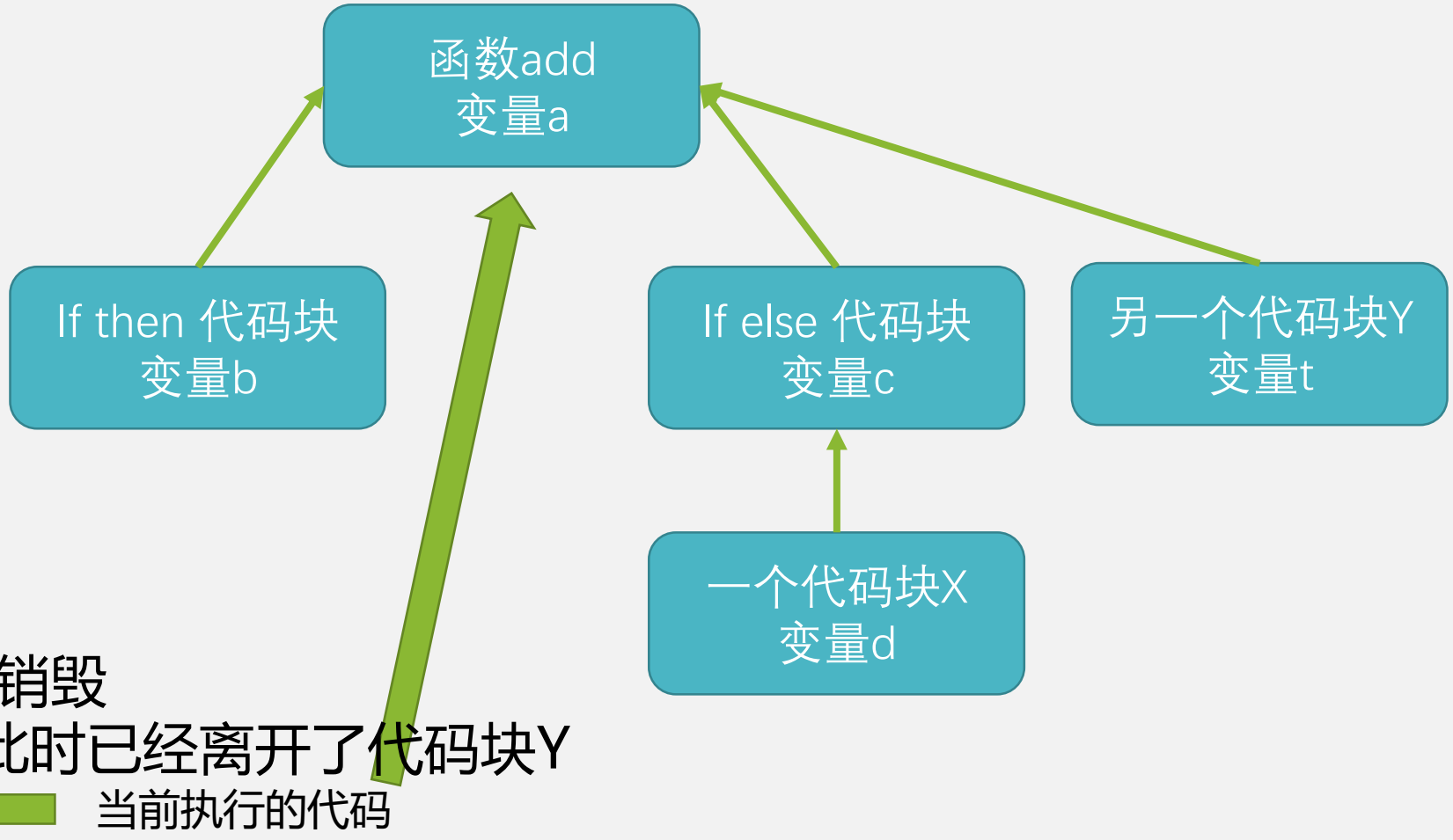


变量t创建，初始化为a的值9

变量的作用域：局部变量

稍复杂的例子

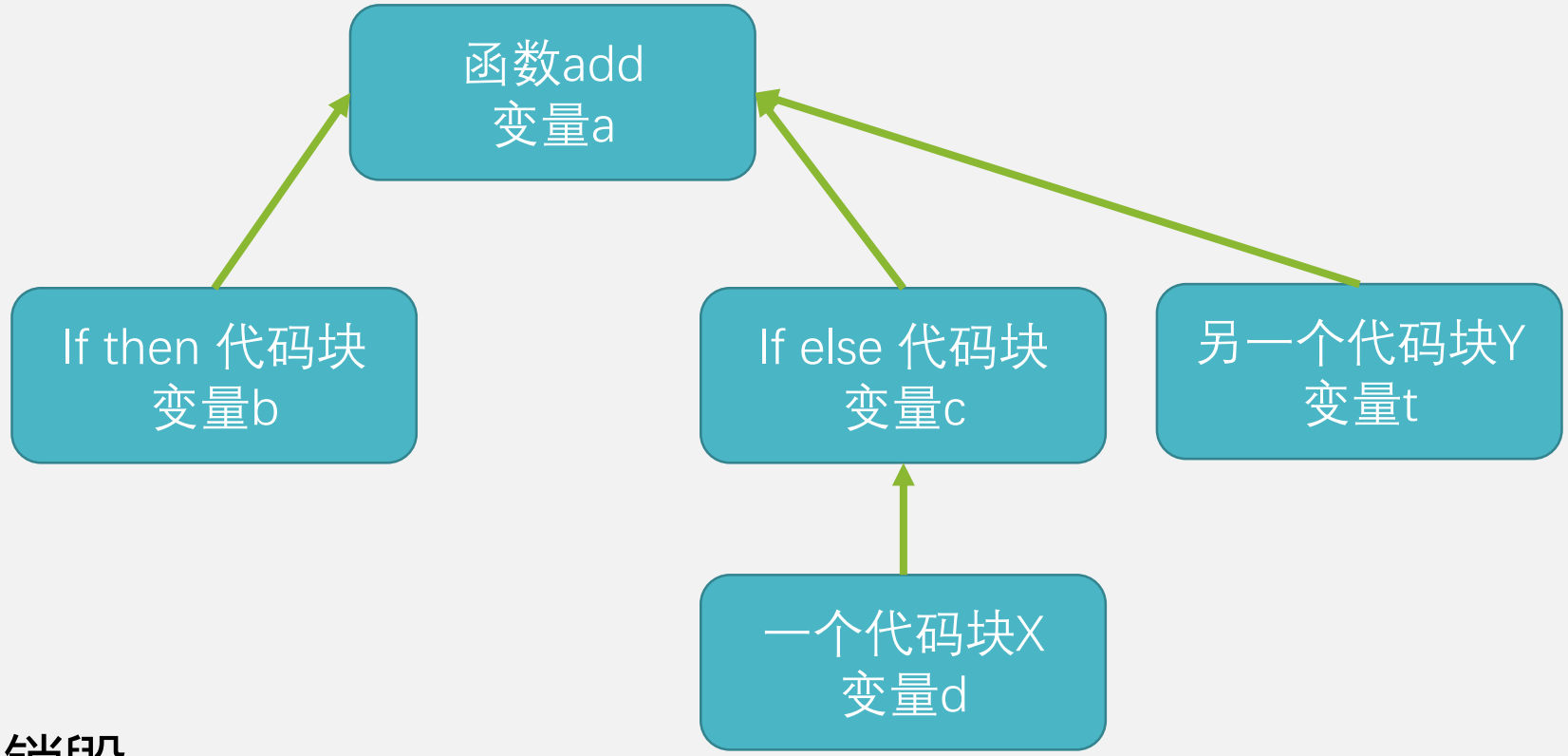
```
void add(void){
    int a = 0;
    if (a == 1){
        int b = 1;
    }else{
        int c;
        {
            int d = a;
        }
        c = a;
    }
    a = 9;
    {
        int t = a;
    }
}
```



变量的作用域：局部变量

稍复杂的例子

```
void add(void){
    int a = 0;
    if (a == 1){
        int b = 1;
    }else{
        int c;
        {
            int d = a;
        }
        c = a;
    }
    a = 9;
    {
        int t = a;
    }
}
```



变量a销毁
因为此时已经离开了函数块add

当前执行的代码

变量的作用域：全局变量

全局变量在函数外声明，一般在代码的顶部
在任何函数内都可以访问

```c
#include <stdio.h>

int tot;

void add(void){
    int a = tot;
}

int main(){
    add();
    return 0;
}
```

全局变量tot

数据类型	初始值
int	0
char	'\0'
float	0
double	0
pointer	NULL

函数add里可以访问全局变量tot

全局变量会在程序初始时给予默认初始化（零）值
生命周期是整个程序的运行
直到程序结束才会被销毁

变量的作用域：形式参数

形式参数在函数参数列表声明
其性质和局部变量r一样，但会被调用函数的参数初始化

```
void add(int a, int b){
    int r = 1;
}

int main(){

    add(1, 2);

    return 0;
}
```

形式参数a, b，分别被初始化为1和2

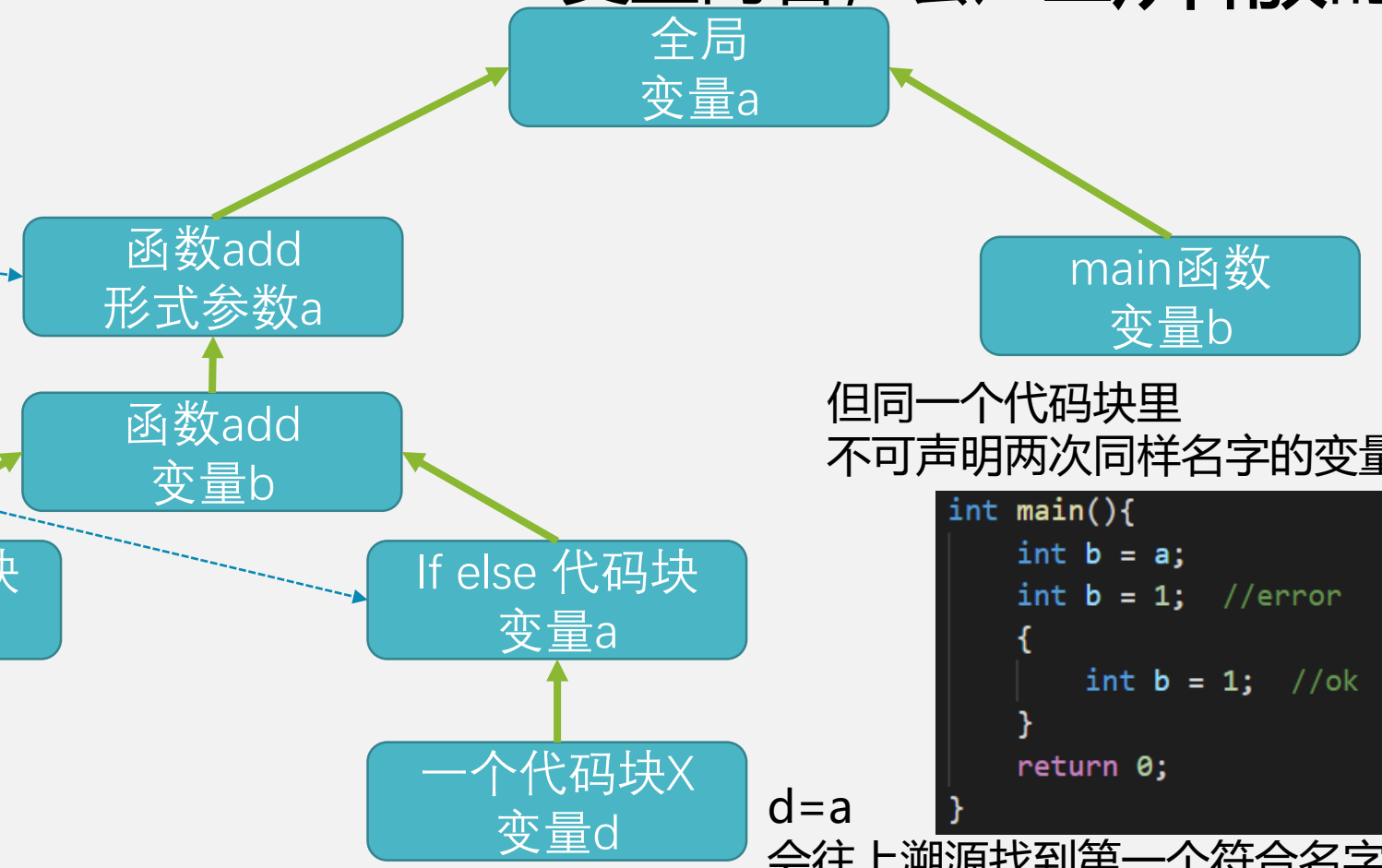
在函数add内均可被访问，
生命周期持续整个函数执行过程，当函数执行完后才销毁

变量的作用域：变量同名

```c
#include <stdio.h>

int a;

void add(int a){
    int b = 0;
    if (a == 1){
        int b = 1;
    }else{
        int a = 2;
        {
            int d = a;
        }
    }
    a = 9;
}

int main(){
    add(0);
    int b = a;
    return 0;
}
```

全局变量a

变量同名，会产生屏蔽的效果



但同一个代码块里
不可声明两次同样名字的变量

```c
int main(){
    int b = a;
    int b = 1;  //error
    {
        int b = 1;  //ok
    }
    return 0;
}
```

d=a
会往上溯源找到第一个符合名字的变量

变量的作用域：变量同名

- 并列语句块内各自定义（不同作用域）的变量同名
- 互不干扰，形参和实参的作用域不同
- 编译器可以区分不同作用域的同名变量，因为它们的内存地址不同

```
#include <stdio.h>
void Function(int x,  int y);
int main(void)
{
    int  x = 1;
    int  y = 2;
    Function(x, y);
    printf("x=%d,y=%d\n",x,y);
    return 0;
}
```

```
void Function(int x,  int y)
{
    x = 2;
    y = 1;
    printf("x=%d,y=%d\n",x,y);
}
```

变量的作用域

位置	种类	初始化	生命周期
在函数或代码块里	局部变量 (local variables)	必须手动初始化	整个函数（块）
在所有函数外面	全局变量 (Global variables)	自动初始化为0	整个程序
在函数参数列表里	形式参数 (Formal parameters)	根据传输参数值 自动初始化	整个函数（块）

案例分析：dotcpp，题目 1042: [编程入门]电报加密

```c
#include<stdio.h>

/*****************************
 * encipher 编码字符。将字母变成其下一字母
 *  （如'a'变成'b'……'z'变成'a'其它字符不变）
 *
 *  output: 编码后的字符
 *  parameter:
 *      int 需要编码的字符
 *      int 偏移量 = 1
 * ***************************/
int encipher(int, int);
```

```c
int main()
{
    char ch;
    // 读入一行电报输入
    scanf("%c",&ch);
    while (ch != '\n') {
        printf("%c",encipher(ch, 1));
        scanf("%c",&ch);
    }
    printf("\n");
    return 0;
}
```

请打开 ex-encipher.c 给出函数 encipher 的定义?

案例分析：dotcpp，题目 1042: [编程入门]电报加密

解答:

```
int encipher(int c, int b) {
    if (c >= 'a' && c <= 'z')
        c = (c - 'a' + b) % 26 + 'a';
    if (c >= 'A' && c <= 'Z')
        c = (c - 'A' + b) % 26 + 'A';
    return c;
}
```

- 使用函数的好处
  - …
- 一些基本概念
  - 函数是**初级表达式**(primary expression)
  - 函数声明的参数叫**形参**(form parameter)
  - 函数调用输入的参数叫**实参**(argument)
  - 当实参与形参不匹配时，会自动**强制转换**
  - 右值表达式等级低于 int 的整数，会自动**整数提升**

函数（1）- 课后练习

Leetcode-cn.com，题目 7. 整数反转

（请课后研究数学类题目，例如：题目 9. 回文数）

请问程序输出是什么？

中山大學
SUN YAT-SEN UNIVERSITY

谢谢

中山大学计算机学院

主讲人：黎卫兵　　中山大学MOOC课程组