Introduction to C Programming

Lecture 11: review II

Wenjin Wang

wangwj3@sustech.edu.cn

12-2-2022

Course syllabus

Nr.	Lecture	Date
1	Introduction	2022.9.9
2	Basics	2022.9.16
3	Decision and looping	2022.9.23
4	Array & string	2022.9.30
5	Functions	2022.10.9 (补)
6	Pointer	2022.10.14
7	Self-defined types	2022.10.21
8	I/O	2022.10.28

Nr.	Lecture	Date
9	Head files	2022.11.4
10	Review of lectures I	2022.11.25
11	Review of lectures II	2022.12.2
12	Review of lectures III	2022.12.9
13	AI in C programming	2022.12.16
14	AI in C programming	2022.12.23
15	AI in C programming	2022.12.30
16	Summary	2023.1.6

Course syllabus

Review of lectures I	1	Introduction	2022.9.9
	2	Basics	2022.9.16
	3	Decision and looping	2022.9.23
Review of lectures II	4	Array & string	2022.9.30
	5	Functions	2022.10.9 (补)
	6	Pointer	2022.10.14
Review of lectures III	7	Self-defined types	2022.10.21
	8	I/O	2022.10.28
	9	Head files	2022.11.4

Objective of this lecture

Review the learned lectures 5 – 6:

Array & string, Functions, Pointer

Content

1. Array & string
2. Functions
3. Pointer

Content

1. Array & string
2. Functions
3. Pointer

Why do we need array?

```
main()
{
   float student_1;
   float student_2;
   float student_3;
   …
   float student_30;

   scanf("%f", &student_1);
   scanf("%f", &student_2);
   scanf("%f", &student_3);
   …
   scanf("%f", &student_30);
}
```

Array可以批量存储和处理数据！

```
main()
{
    for (int i = 0; i < 30; i++)
    {
        float student_i;
        scanf("%f", &student_i);
    }
}
```

1-D array

C provides a data structure called **array**. It stores a fixed-size collection of elements of the same type.

int array[10]	3	2	1	5	…	8
float array[10]	1.2	4.5	-1.9	3.4	…	8.8
char array[10]	H	R	O	Y	…	P

1-D array

Declare, initialize and access an int array:

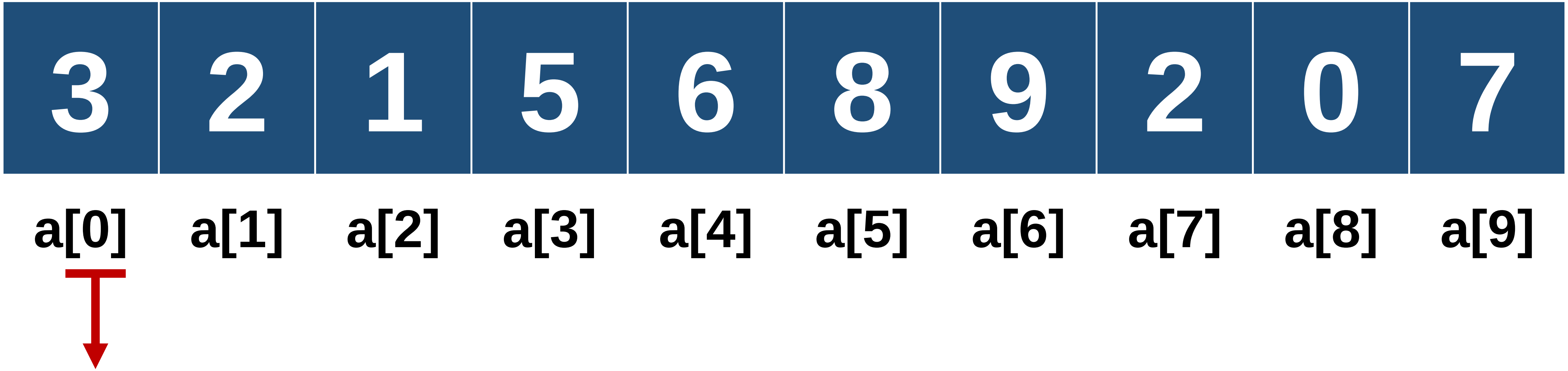
- int a[10]; **// declare**
- a[0] = 3, a[1] = 2, …., a[9] = 7; **// initialize**

- int a[10] = {3, 2, 1, 5, 6, 8, 9, 2, 0, 7}; **// declare and initialize**
- int a[] = {3, 2, 1, 5, 6, 8, 9, 2, 0, 7}; **// declare and initialize**

- printf(“a[5] = %d”, a[5]); **// access the array**

1-D array

```
int a[10] = {3, 2, 1, 5, 6, 8, 9, 2, 0, 7}; // length is 10
```



3	2	1	5	6	8	9	2	0	7
a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]	a[9]

Index starts at 0

Can we access array by a[10]?

1-D array

int a[10] = {3, 2, 1}; // length is 10, fit rests with 0

3	2	1	0	0	0	0	0	0	0
a[0]	**a[1]**	**a[2]**	**a[3]**	**a[4]**	**a[5]**	**a[6]**	**a[7]**	**a[8]**	**a[9]**

int a[] = {3, 2, 1}; // length is 3

3	2	1
a[0]	**a[1]**	**a[2]**

1-D array

You can also define float array and char array

float array: float a[] = {1.2, -0.6, 1000, -32, 5.34};

1.2	-0.6	1000	-32	5.34

char array: char c[] = {‘h’, ‘e’, ‘l’, ‘l’, ‘o’, ‘!’};

‘h’	‘e’	‘l’	‘l’	‘o’	‘!’

2-D array

Declare and initialize a 2D int array

3	2	5
1	7	6

- int a[2][3]; **// 2 rows x 3 columns**
- a[0][0] = 3; a[0][1] = 2; a[0][2] = 5;
- a[1][0] = 1; a[1][1] = 7; a[1][2] = 6;

Access array: printf(“a[1][1] = %d”, a[1][1]);

1	0	0	2
0	1	0	0
0	2	1	4

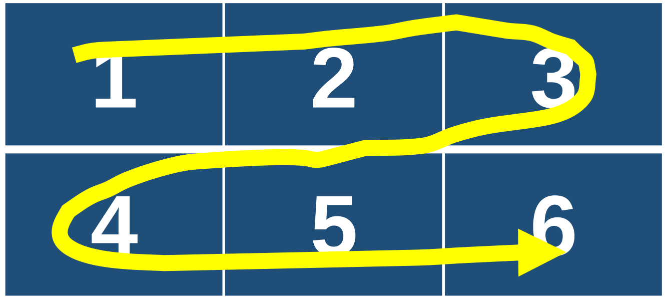
- int a[3][4]; **// 3 rows x 4 columns**
- a[0][0] = 1; a[0][1] = 0; a[0][2] = 0; a[0][3] = 2;
- a[1][0] = 0; a[1][1] = 1; a[1][2] = 0; a[1][3] = 0;
- a[2][0] = 0; a[2][1] = 2; a[2][2] = 1; a[2][3] = 4;

Access array: printf(“a[2][3] = %d”, a[2][3]);

2-D array

Declare and initialize a 2D int array

- int a[2][3] = {{1, 2, 3}, {4, 5, 6}};
- int a[2][3] = {1, 2, 3, 4, 5, 6}; // **preferred!**
- int a[][3] = {1, 2, 3, 4, 5, 6}; // 2 x 3 mat
- int a[3][4] ={{1}, {5, 6}}; // 3 x 4 mat



1	2	3
4	5	6

1	0	0	0
5	6	0	0
0	0	0	0

3-D/N-D array

Declare and initialize a 3-D/N-D int array

- int a[2][3][4];
- a[0][0][0] = 1; a[0][1][2] = 3; a[1][0][3] = 2; // **preferred!**
- int a[2][3][4]= {{{1, 2, 3}, {4, 5, 6}},{{2, 4, 5}, {2, 4, 2}}…};

- int a[2][3][4][2];
- a[0][0][0][0] = 1; a[0][1][2][0] = 3; a[1][0][3][1] = 2;

Use for loop to define 2D/3D array

2D array

```
int n[4][5];
for (int x = 0; x < 4; x++)
{
    for (int y = 0; y < 5; y++)
    {
        n[x][y] = x+y;
    }
}
```

3D array

```
int n[2][2][3];
for (int x = 0; x < 2; x++)
{
    for (int y = 0; y < 2; y++)
    {
          for (int z = 0; z < 3; z++)
          {
                n[x][y][z] = x+y+z;
           }
        }
    }
```

String

String is an array of characters.

```
char c[10] = {‘I’, ‘ ’, ‘a’, ‘m’, ‘ ’, ‘h’, ‘a’, ‘p’ , ‘p’, ‘y’}; // length is 10
char c[10] = {“I am happy”};
char c[] = {“I am happy”};
char c[] = “I am happy”; // preferred
```

I		a	m		h	a	p	p	y
c[0]	c[1]	c[2]	c[3]	c[4]	c[5]	c[6]	c[7]	c[8]	c[9]

1D and 2D String

1D char array holds the characters!

```
char c[10] = “I am happy”;
```

Machine thinks it as a single “word”!

I		a	m		h	a	p	p	y

2D char array holds the words!

```
char c[3][10] = {“I”, “am”, “happy”};
```

Machine thinks it as a group of word!

I									
a	m								
h	a	p	p	y					

String

```
char c[10] = {‘S’, ‘ U’, ‘S’, ‘T’, ‘e ’, ‘c’, ‘h’}; // length is 10
char c[10] = {“SUSTech”};
char c[] = {“SUSTech”};
char c[] = “SUSTech”; // preferred
```

S	U	S	T	e	c	h	\0	\0	\0
c[0]	c[1]	c[2]	c[3]	c[4]	c[5]	c[6]	c[7]	c[8]	c[9]

String operations

C supports a wide range of functions that manipulate strings.

Operators	**Description**	**Example s1=A, S2 = B;**
strcpy(s1, s2)	Copy s2 into s1	s1 = B
strcat(s1, s2)	Concatenate s1 and s2	S1 = AB
strlen(s1)	Return length of s2	Length = 1
strcmp(s1, s2)	Compare s1 and s2	A<B, return -1
strlwr(s1)	Convert s1 to lower case	A to a
strupr(s1)	Convert s1 to upper case	A to A

strcpy(s1, s2)

```
char str1[12] = "Hello";
char str2[12] = "World";
char str3[12];

strcpy(str3, str1);
printf("str3 = %s\n", str3); //Hello

strcpy(str3, str2);
printf("str3 = %s\n", str3); //World
```

strcat(s1, s2)

```
char str1[12] = "Hello";
char str2[12] = "World";
char str3[12] = "123";

strcat(str1, str2);
printf("str1 = %s\n", str1); //HelloWorld

strcat(str3, str2);
printf("str3 = %s\n", str3); //123World
```

strlen(s1)

```
char str1[12] = "Hello";
char str2[] = "World";
char str3[12];

printf("str1 = %s\n", strlen(str1)); //5

printf(“str2 = %s\n”, strlen(str2)); //5

printf("str3 = %s\n", strlen(str3)); //0
```

sizeof(s1)

```
char str1[12] = "Hello";
char str2[] = "World";
char str3[12];

printf("str1 = %s\n", sizeof(str1)); //12

printf(“str2 = %s\n”, sizeof(str2)); //6, end with ‘\0’

printf("str3 = %s\n", sizeof(str3)); //12
```

strcmp(s1, s2)

```
char str1[] = "ABCD";
char str2[] = "BCD";
char str3[] = "ABCE";
char str4[] = "1234";
```

```
str1 > str2 →  1
str1 < str2 → -1
str1 = str2 →  0
```

```
printf("cmp = %d\n", strcmp(str1, str2)); //-1

printf("cmp = %d\n", strcmp(str1, str3)); //-1

printf("cmp = %d\n", strcmp(str1, str1)); //0
```

strlwr(s1)

```
char str1[] = "ABCD";
char str2[] = "abcd";
char str3[] = “012abcDE";

printf("strlwr = %d\n", strlwr(str1)); //abcd

printf("strlwr = %d\n", strlwr(str2)); //abcd

printf("strlwr = %d\n", strlwr(str3)); //012abcde
```

strupr(s1)

```
char str1[] = "ABCD";
char str2[] = "abcd";
char str3[] = "012abcDE";

printf("strupr = %d\n", strupr(str1)); //ABCD

printf("strupr = %d\n", strupr(str2)); //ABCD

printf("strupr = %d\n", strupr(str3)); //012ABCDE
```

Summary

- We can use **array** to hold many data for group processing
- Array has the **fixed size** and can only be used to hold data with **same type**
- **Different types of array** can be created, e.g. int array, float array, char array (string)
- **Different dimensional array** can be created, from 1D array to ND array
- Array enables the processing of **vectors, matrices, strings**, etc.

5 Questions

1. If we use the array name as the argument for the function, what does the argument stand for? ( )

A. The value of the first element in the array
B. The value of all elements in the array
C. The address of the first element in the array
D. The address of all elements in the array

2. How to declare a 1D array? ( )

A. int a(10); B. int a{10}; C. int [10]a; D. int a[10];

3. How to declare a 2D array? ( )

A. int a[3][]; B. float a(3,4); C. double a[3][4]; D. float a(3)(4);

5 Questions

4. Which statement can initialize a 2D array correctly? ( )

A. int a[2][3]={{1,2},{3,4},{5,6}};
B. int a[2][3]={{1,2},{},{4,5}};
C. int a[][3]={1,2,3,4,5,6};
D. int a[2][]={{1,2},{3,4},{4,5}};

5. Which statement is correct in checking if string s1 equals to string s2? ( )

A. if(s1 == s2) B. if(s1 = s2) C. if(strcpy(s1,s2)) D. if(strcmp(s1,s2) == 0)

Content

1. Array & string
2. Functions
3. Pointer

Function

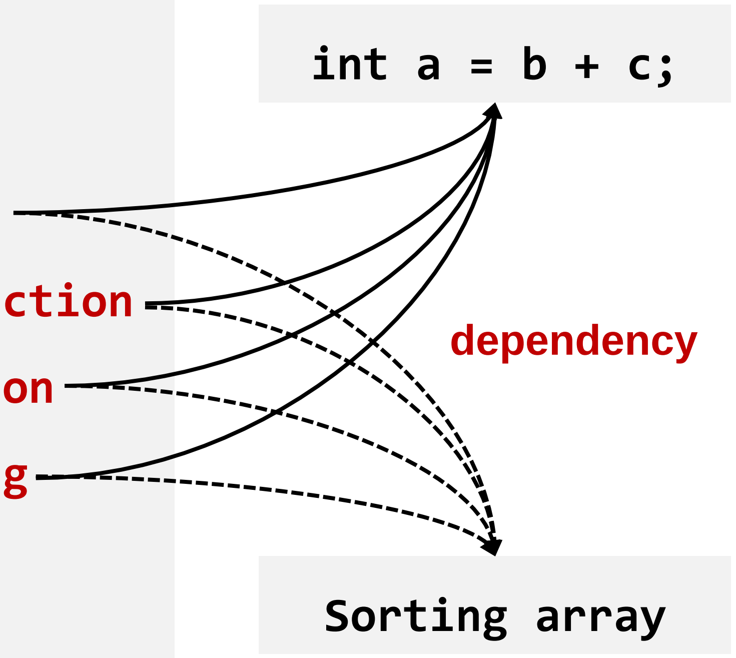
Main is a function, performing a task!

```
int main()
{
    // do nothing or do something!!!
    return 0;
}
```

Function

Main usually includes multiple tasks, preferred not to write all tasks in a big main!!!

```
int main()
{
    // face detection
    // face mask detection
    // face recognition
    // decision making
    return 0;
}
```



`int a = b + c;`

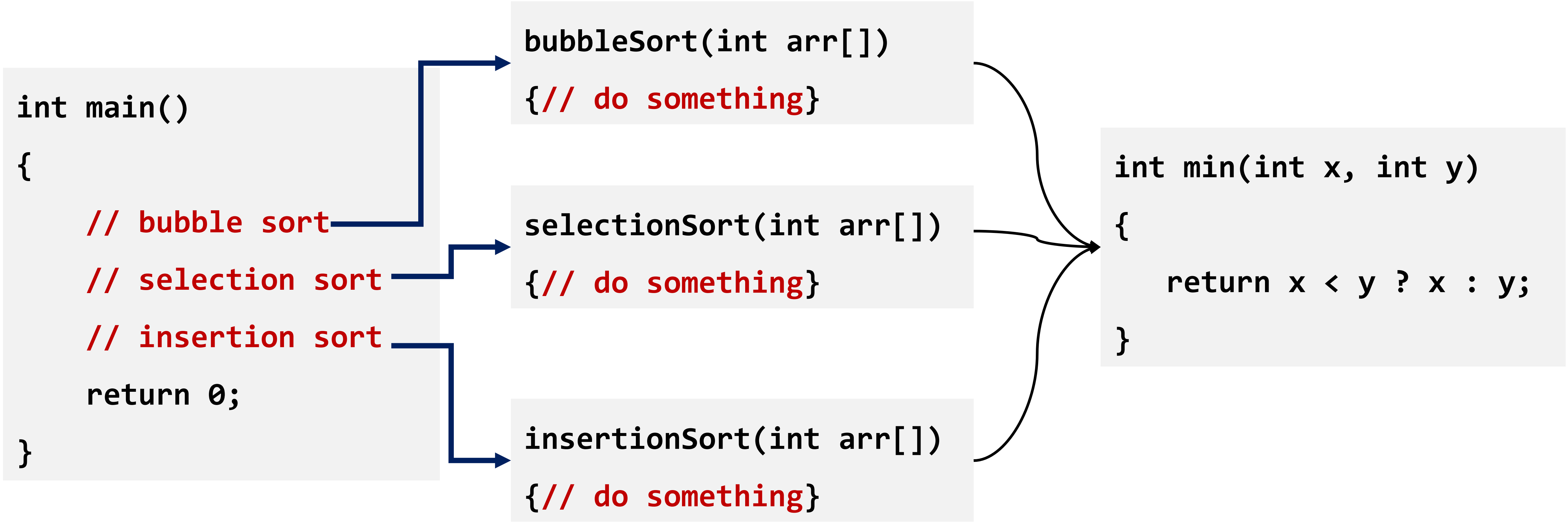
dependency

`Sorting array`

- Not modular
- Difficult to maintain
- Difficult to hand-over
- Cannot be re-used

Function

Make functions independent of the main!



```
int main()
{
    // bubble sort
    // selection sort
    // insertion sort
    return 0;
}
```

```
bubbleSort(int arr[])
{// do something}
```

```
selectionSort(int arr[])
{// do something}
```

```
insertionSort(int arr[])
{// do something}
```

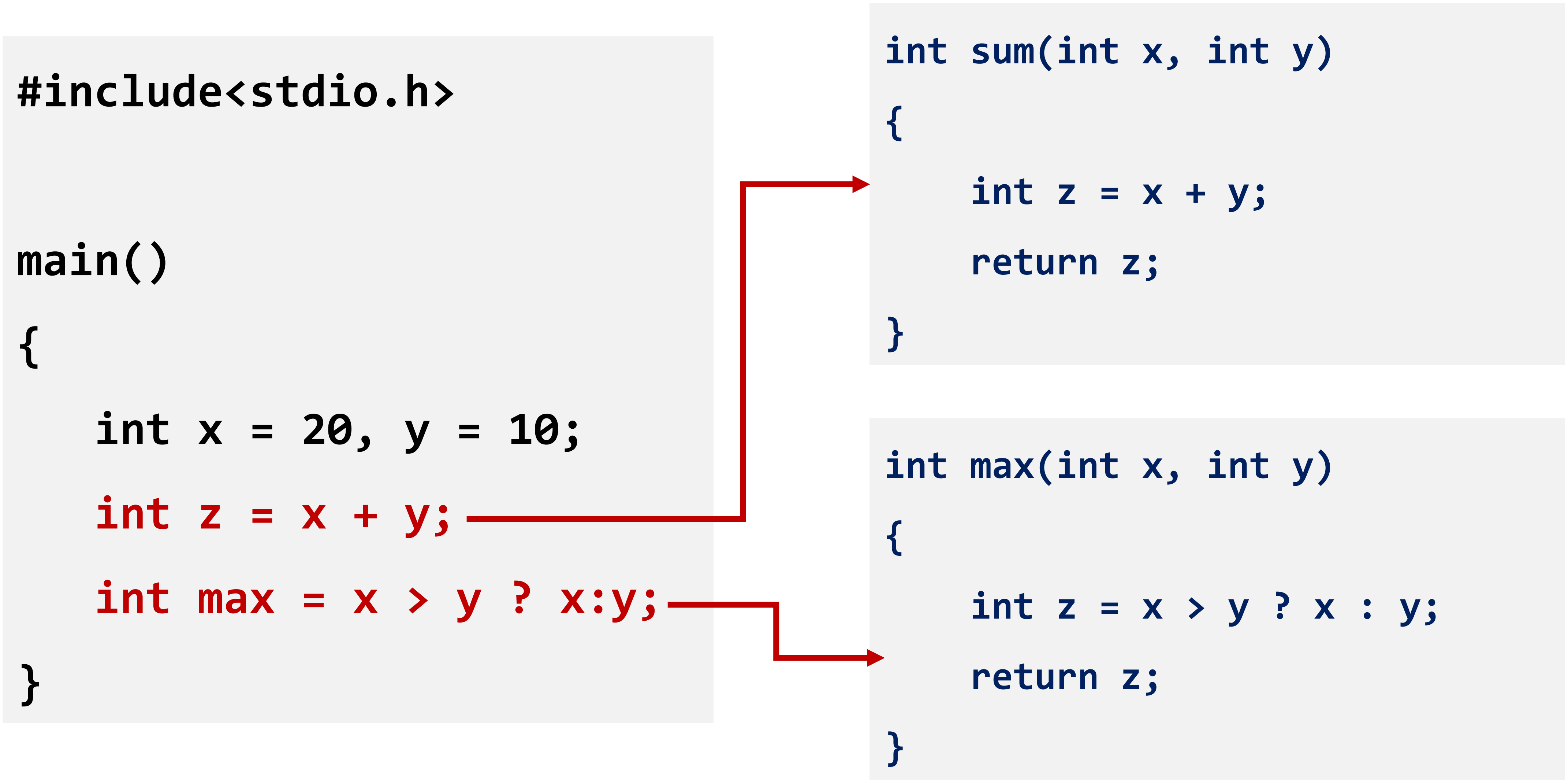
```
int min(int x, int y)
{
   return x < y ? x : y;
}
```

Function

Function is a group of statements that together perform a task. C provides numerous built-in functions, we can also define our own functions.

```
return_type function_name(parameters)
{
    body of the function

    return;
}
```

Function makes operations more independent



```
#include<stdio.h>


main()
{
    int x = 20, y = 10;
    int z = x + y;
    int max = x > y ? x:y;
}
```

```
int sum(int x, int y)
{
    int z = x + y;
    return z;
}
```

```
int max(int x, int y)
{
    int z = x > y ? x : y;
    return z;
}
```

C-defined functions

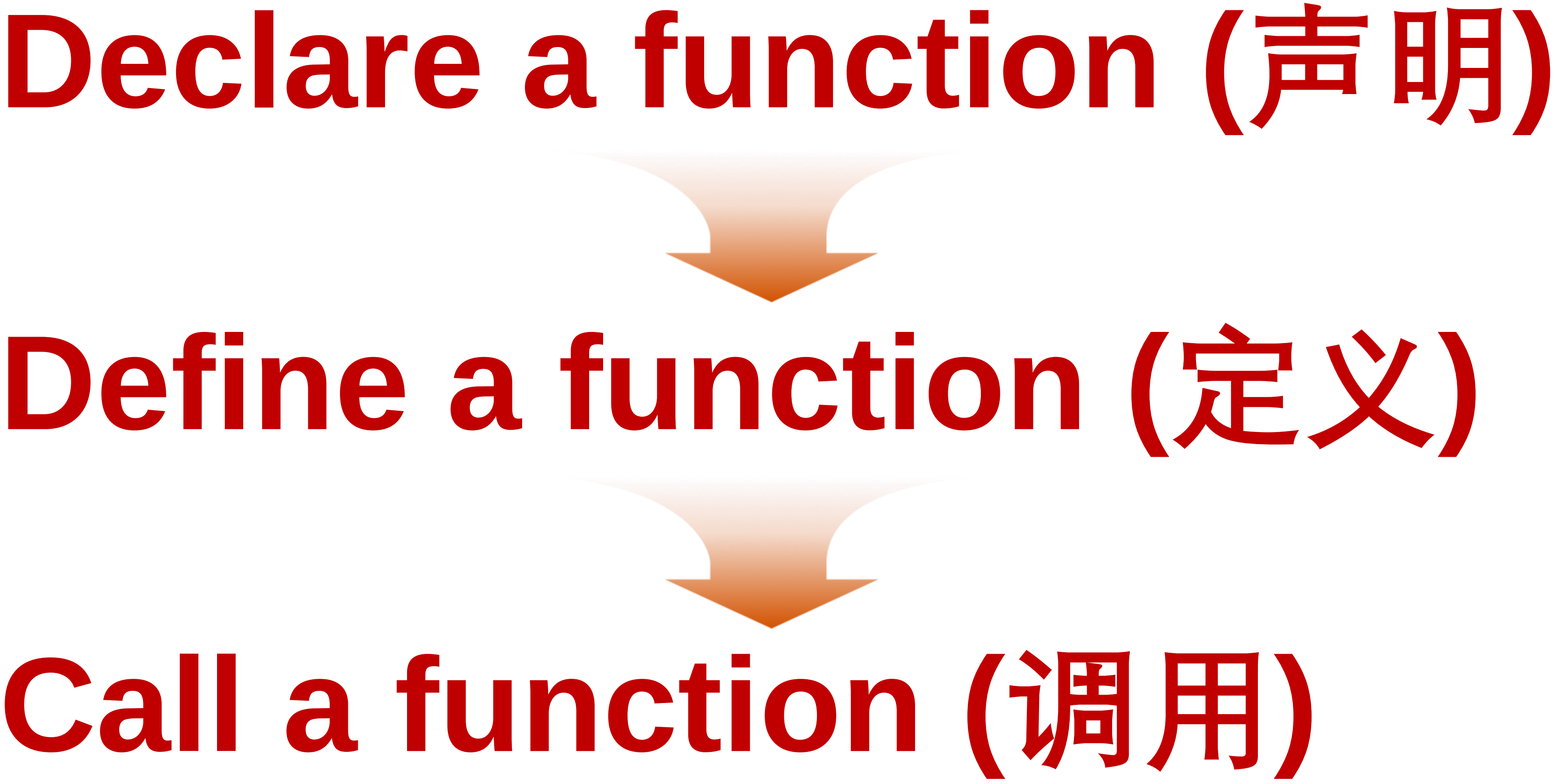
sqrt function in math.h

```
float sqrt(float number)
{
    long i;
    float x2, y;
    const float threehalfs = 1.5F;
    x2 = number * 0.5F;
    y = number;
    i = *(long*)&y;
    i = 0x5f3759df - (i >> 1); y = *(float*)&i;
    y = y * (threehalfs - (x2 * y * y));
    #ifndef Q3_VM
    #ifdef __linux__
    assert(!isnan(y)); #endif
    #endif
    return y;
}
```

printf function in stdio.h

```
int printf(const char* fmt, ...)
{
    int i;
    char buf[256];

    va_list arg = (va_list)((char*)(&fmt) + 4);
    i = vsprintf(buf, fmt, arg);
    write(buf, i);

    return i;
}
```

Self-defined function

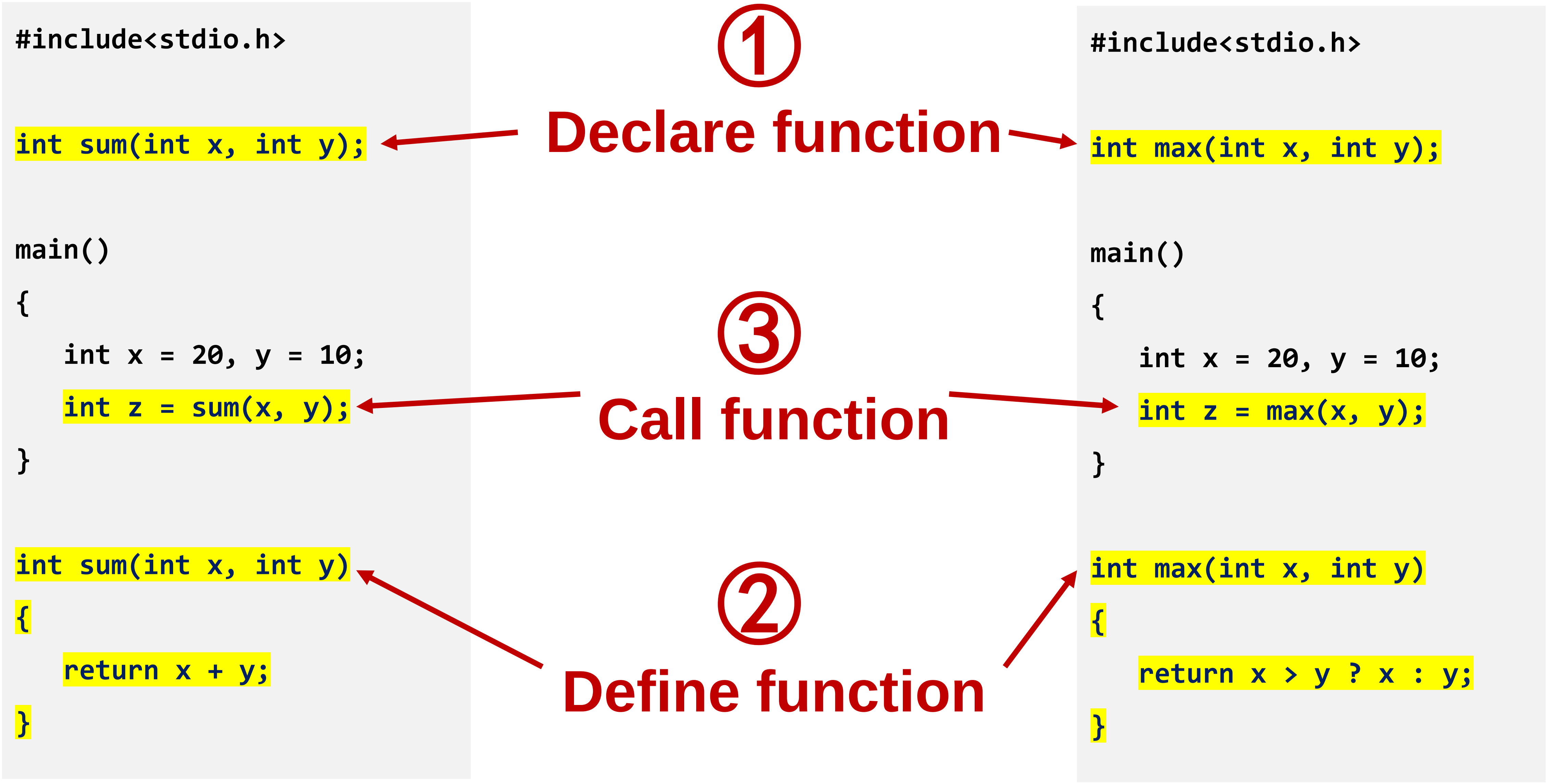


Declare a function (声明)

Define a function (定义)

Call a function (调用)

Function



```
#include<stdio.h>

int sum(int x, int y);

main()
{
   int x = 20, y = 10;
   int z = sum(x, y);
}

int sum(int x, int y)
{
   return x + y;
}
```

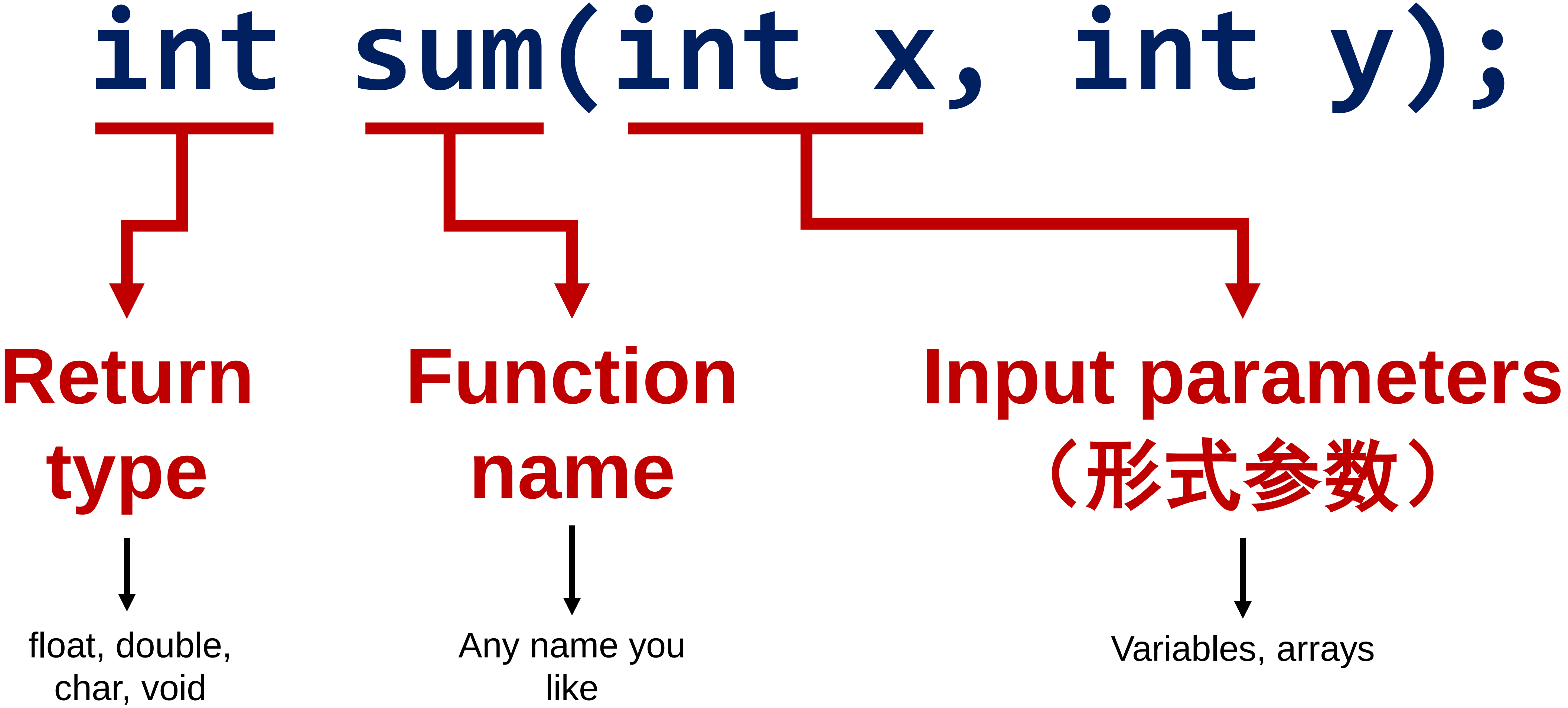
① Declare function

③ Call function

② Define function

```
#include<stdio.h>

int max(int x, int y);

main()
{
   int x = 20, y = 10;
   int z = max(x, y);
}

int max(int x, int y)
{
   return x > y ? x : y;
}
```

Declare a function



```
int sum(int x, int y);
```

Return type

float, double, char, void

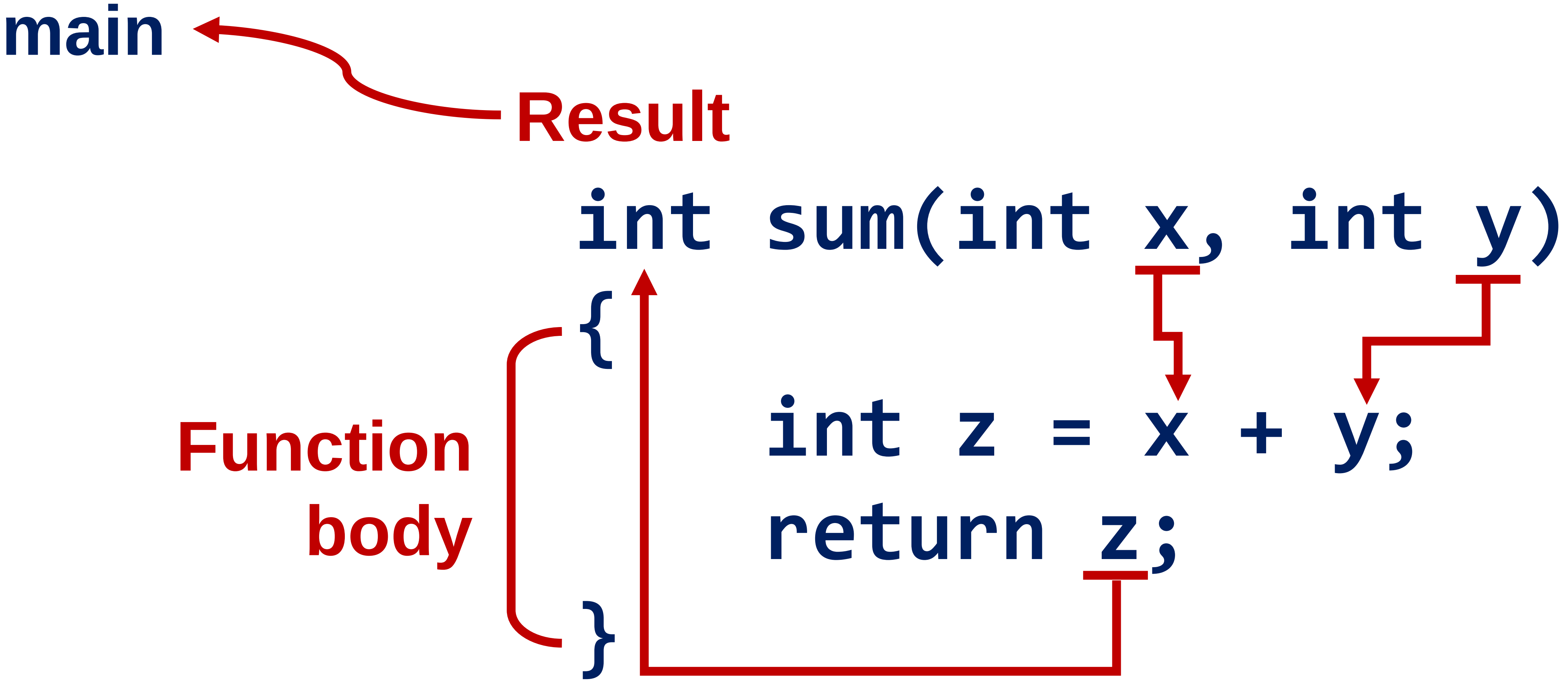
Function name

Any name you like

Input parameters （形式参数）

Variables, arrays

Define a function



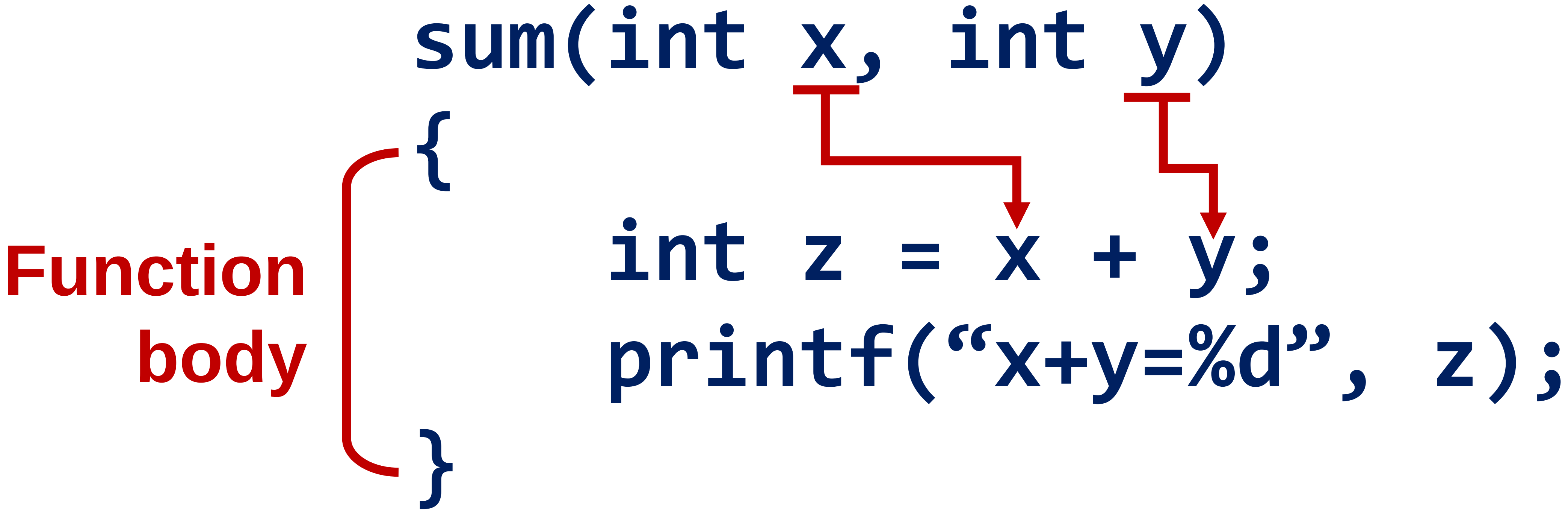
main

Result

```
int sum(int x, int y)
{
    int z = x + y;
    return z;
}
```

Function body

Define a function



```
sum(int x, int y)
{
  int z = x + y;
  printf(“x+y=%d”, z);
}
```

Function body

Call a function

```
main()
{
    int x = 20, y = 10;
    int z = sum(x, y);
}
```

Arguments
（实际参数）

Function positioning matters

```
#include<stdio.h>


int sum(int x, int y)
{
    return x + y;
}


main()
{
    int x = 20, y = 10;
    int z = sum(x, y);
}
```

①

Declare and define function before main!!!

②

Call function

```
#include<stdio.h>


int max(int x, int y)
{
    return x > y ? x : y;
}


main()
{
    int x = 20, y = 10;
    int z = max(x, y);
}
```

Function positioning matters

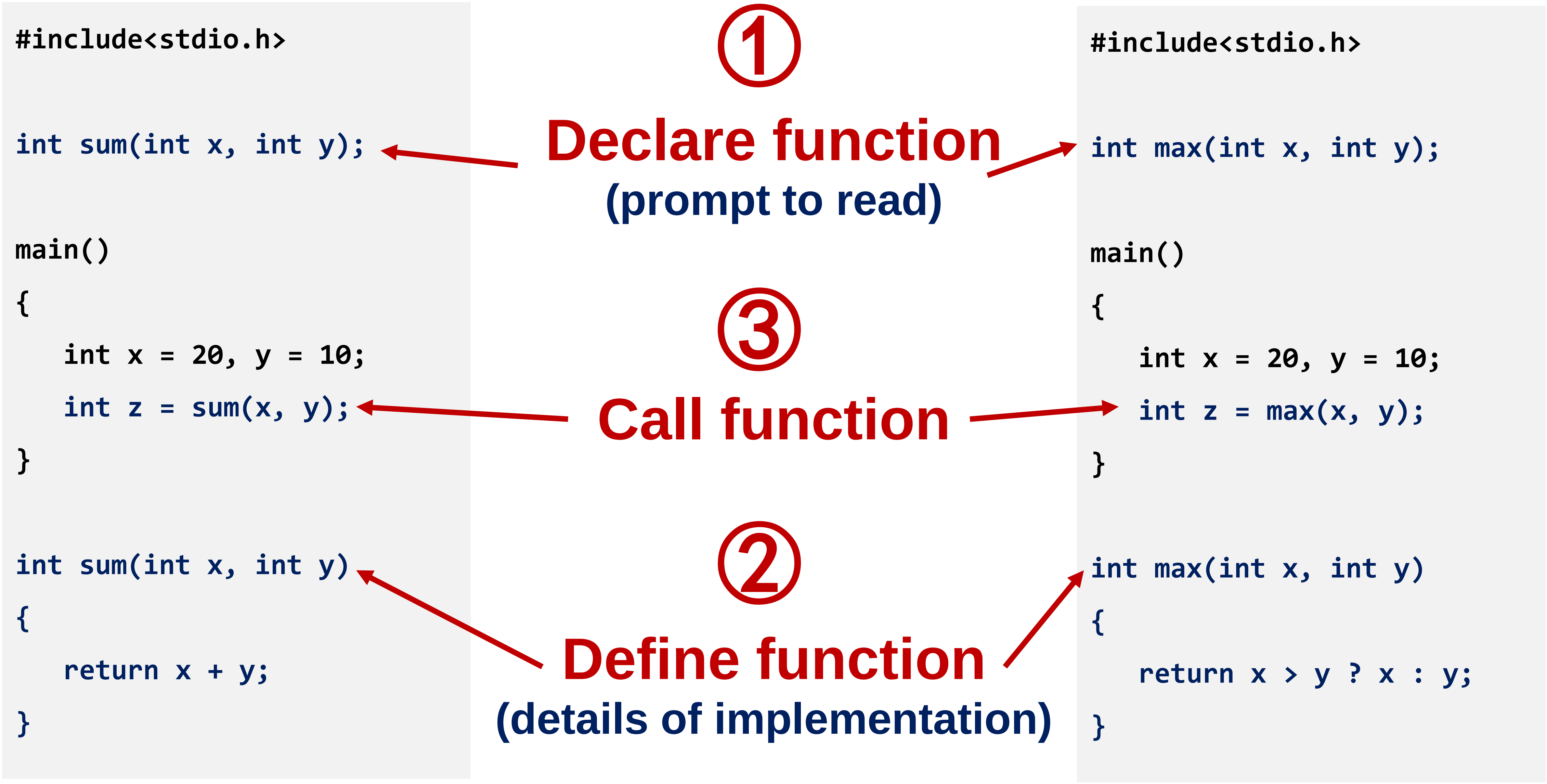
```c
#include<stdio.h>


main()
{
   int x = 20, y = 10;
   int z = sum(x, y);
}


int sum(int x, int y)
{
   return x + y;
}
```

Wrong!

Compiler cannot recognize the function declared after main

```c
#include<stdio.h>


main()
{
   int x = 20, y = 10;
   int z = max(x, y);
}


int max(int x, int y)
{
   return x > y ? x : y;
}
```

Strongly suggested structure!



```
#include<stdio.h>

int sum(int x, int y);

main()
{
   int x = 20, y = 10;
   int z = sum(x, y);
}

int sum(int x, int y)
{
   return x + y;
}
```

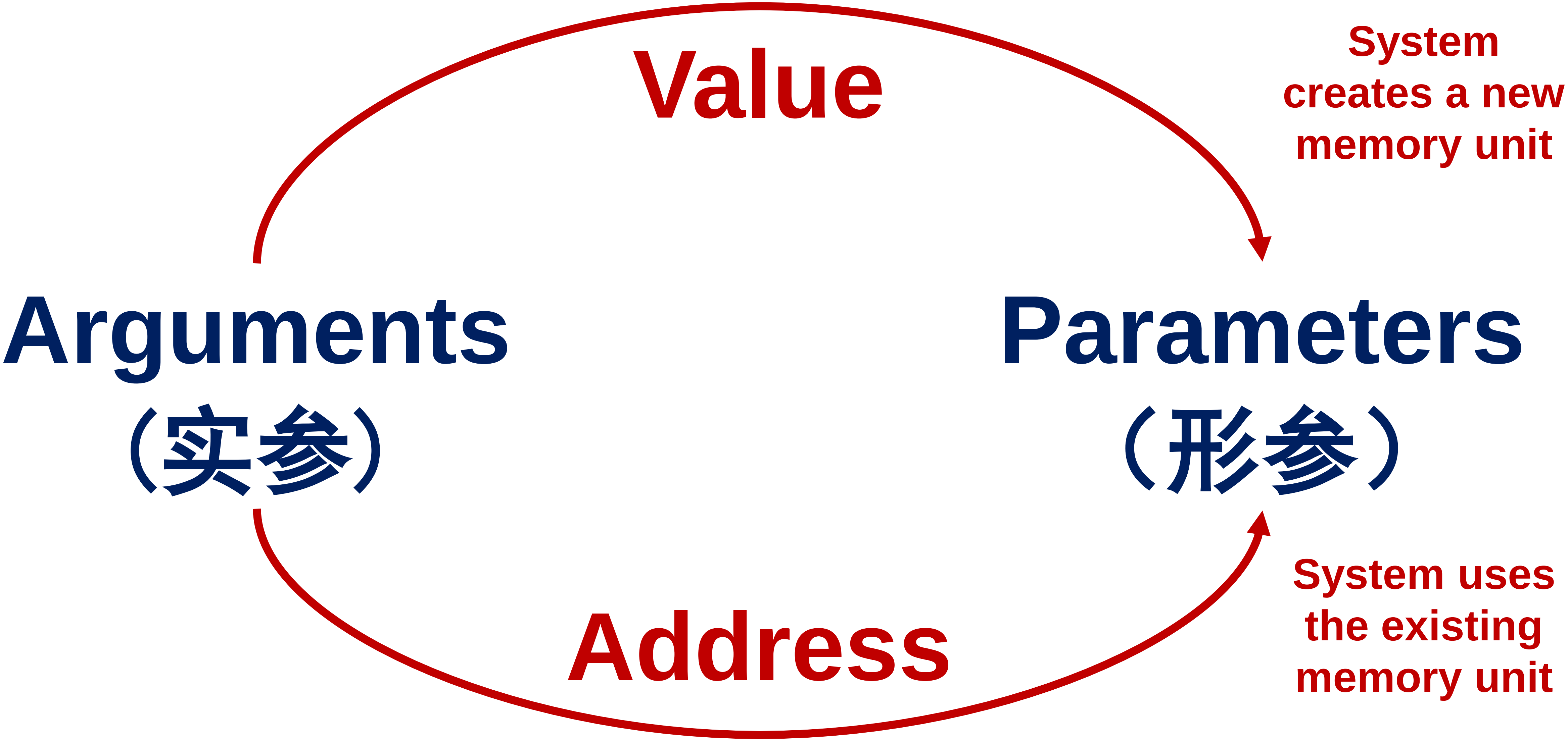
① Declare function (prompt to read)

③ Call function

② Define function (details of implementation)

```
#include<stdio.h>

int max(int x, int y);

main()
{
   int x = 20, y = 10;
   int z = max(x, y);
}

int max(int x, int y)
{
   return x > y ? x : y;
}
```

Arguments and parameters



Value

System creates a new memory unit

Arguments （实参）

Parameters （形参）

Address

System uses the existing memory unit

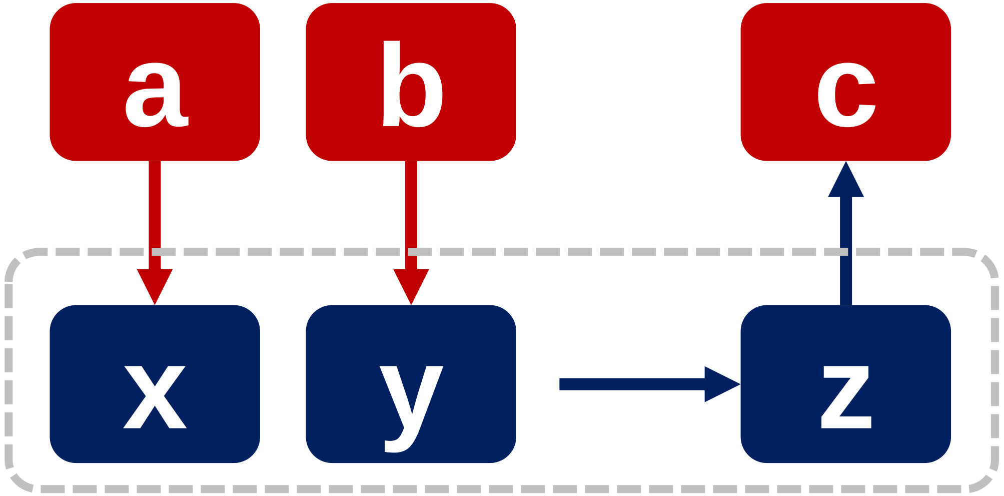
Arguments and parameters

```
#include <stdio.h>

int sum(int x, int y);

main ()
{
    int a = 100;
    int b = 200;

    int c = sum(a, b);
}
```

a and b are arguments（实参）

x and y are parameters（形参）

```
int sum(int x, int y)
{
    int z = x + y;
    return z;
}
```



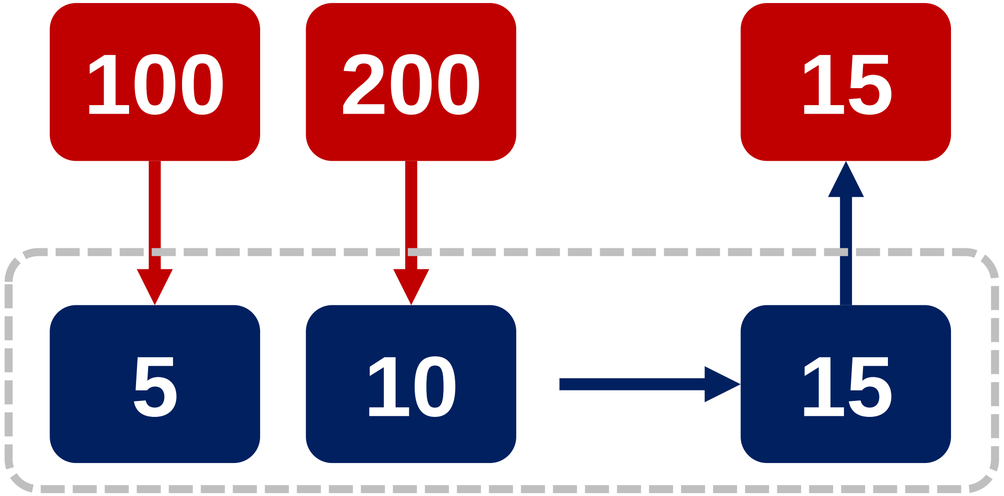
Arguments and parameters

```
#include <stdio.h>

int sum(int x, int y);

main ()
{
    int a = 100;
    int b = 200;

    int c = sum(a, b);
}
```

a and b are arguments（实参）

x and y are parameters（形参）

```
int sum(int x, int y)
{
    x = 5, y = 10;
    int z = x + y;
    return z;
}
```



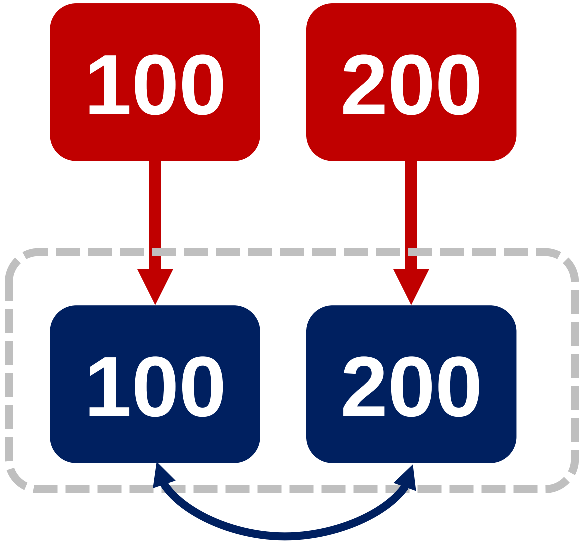
Arguments and parameters

```
#include <stdio.h>

void swap(int x, int y);

main ()
{
     int a = 100;
     int b = 200;

     swap(a, b);
}
```

a and b are arguments（实参）

x and y are parameters（形参）

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



Can a and b be swapped?

Arguments and parameters

```
#include <stdio.h>

void swap(int *x, int *y);

main ()
{
    int a = 100;
    int b = 200;

    swap(&a, &b);
}
```

&a and &b are address of arguments

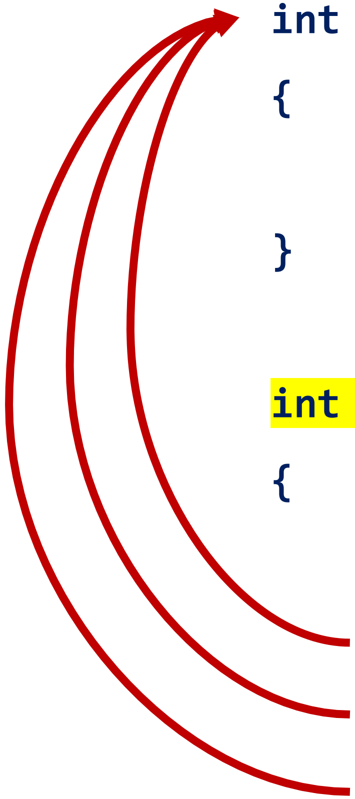
Pointers

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

Can a and b be swapped now?

Functions can be nested

```
#include<stdio.h>

int max(int x, int y);
int max_4(int a, int b, int c, int d);

main()
{
   int a = 20, b = 10, c = 4, d = 1;
   int z = max_4(a, b, c, d);
}
```



```
int max(int x, int y)
{
    return x > y ? x : y;
}

int max_4(int a, int b, int c, int d)
{
     int z;
     z = max(a, b);
     z = max(z, c);
     z = max(z, d);
     return z;
}
```

max_4() calls max() 3 times!!!

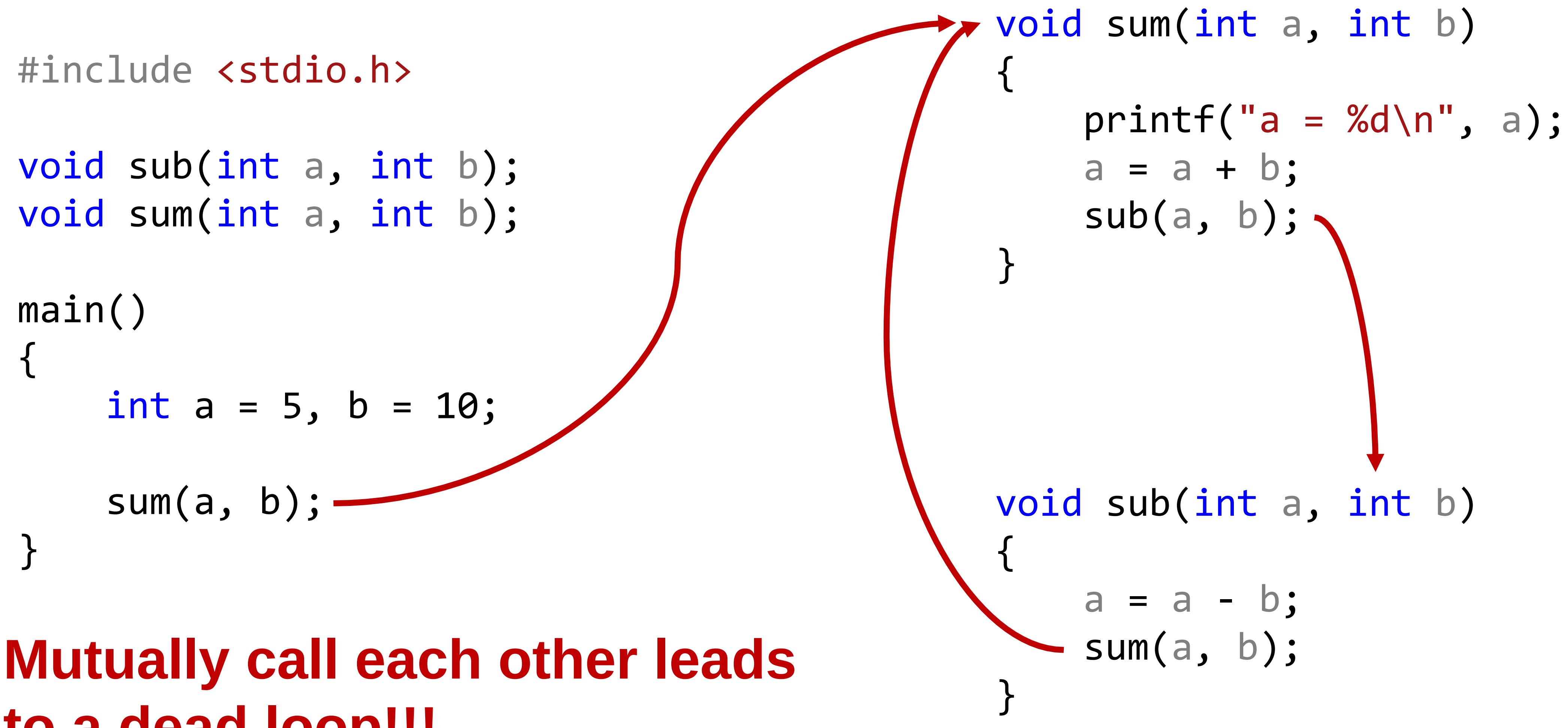
Functions can be nested

```
#include<stdio.h>

int add(int x, int y);
int sum(int a[]);

main()
{
   int a[] = {1,3,5,7,2,9,-3,2};
   int z = sum(a, sizeof(a)/sizeof(a[0]));
}
```

sum() repeatedly calls add() in a loop!!!

```
int add(int x, int y)
{
    return x + y;
}

int sum(int a[], int len)
{
     int z = 0;
     for(int i = 0; i < len; i++)
     {
         z = add(z, a[i]);
     }
     return z;
}
```

Functions can be mutually nested



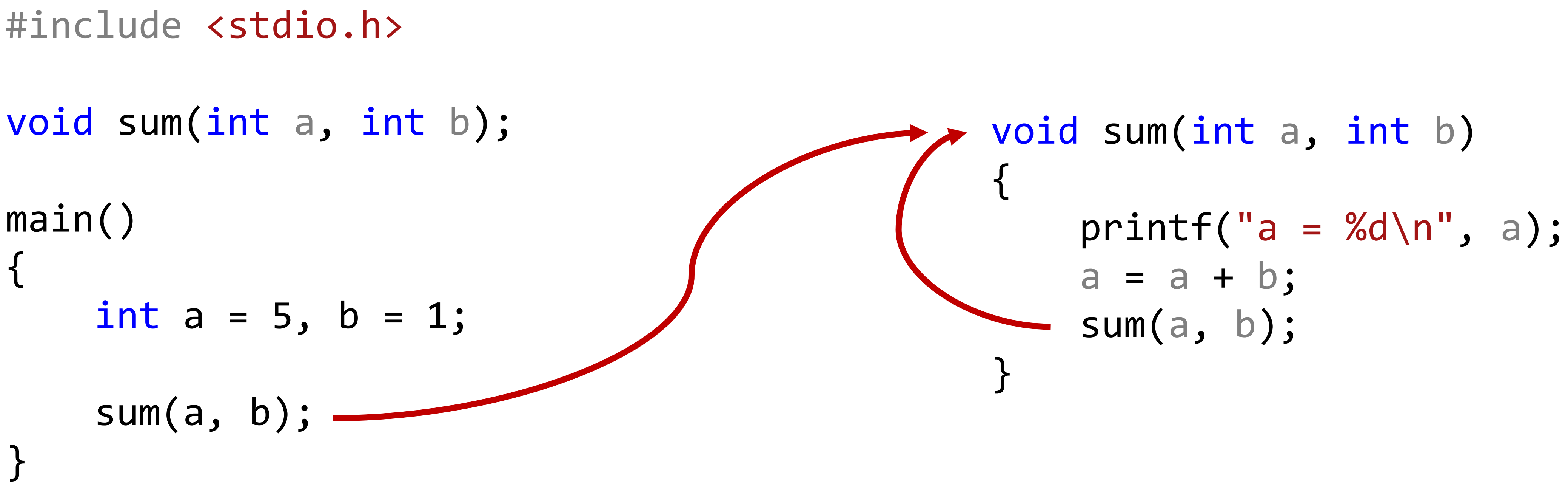
```c
#include <stdio.h>

void sub(int a, int b);
void sum(int a, int b);

main()
{
    int a = 5, b = 10;

    sum(a, b);
}
```

```c
void sum(int a, int b)
{
    printf("a = %d\n", a);
    a = a + b;
    sub(a, b);
}
```

```c
void sub(int a, int b)
{
    a = a - b;
    sum(a, b);
}
```

Mutually call each other leads to a dead loop!!!

Function can be self-called

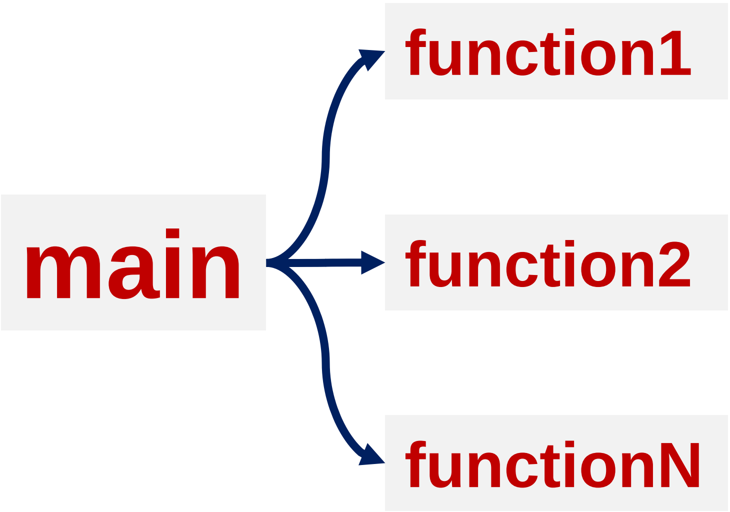


```c
#include <stdio.h>

void sum(int a, int b);

main()
{
    int a = 5, b = 1;

    sum(a, b);
}
```

```c
void sum(int a, int b)
{
    printf("a = %d\n", a);
    a = a + b;
    sum(a, b);
}
```

This is recursion!!!

Functions

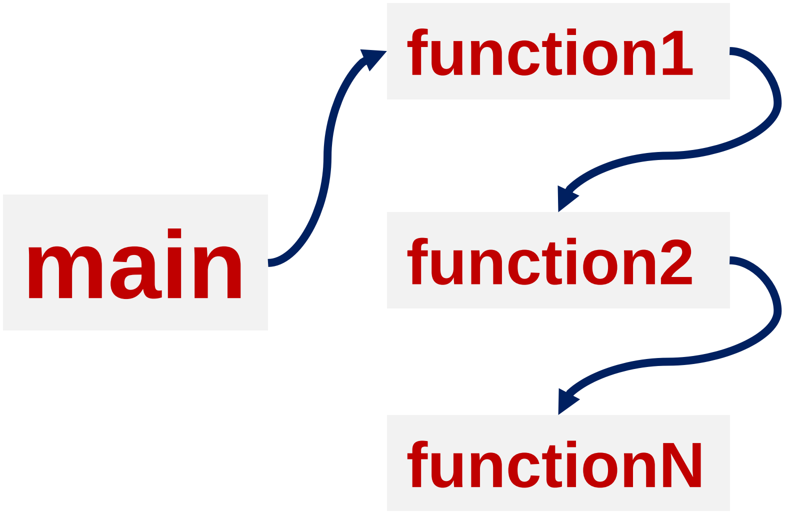


main

function1

function2

functionN

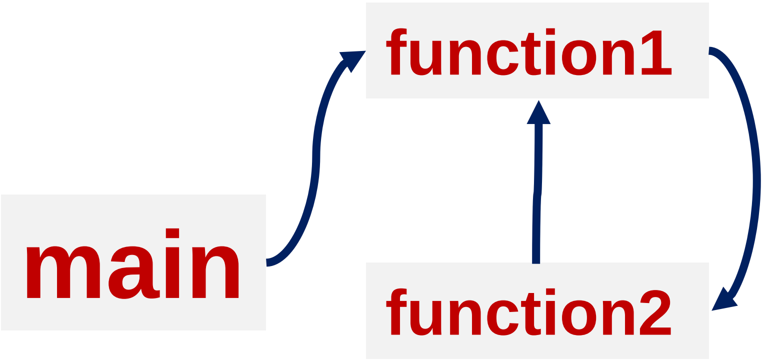


main

function1

function2

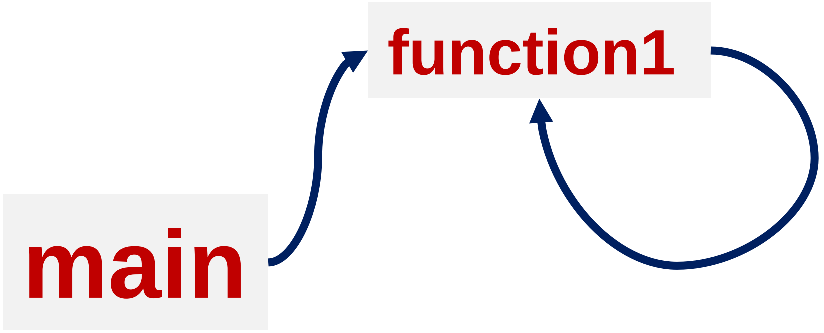
functionN



main

function1

function2

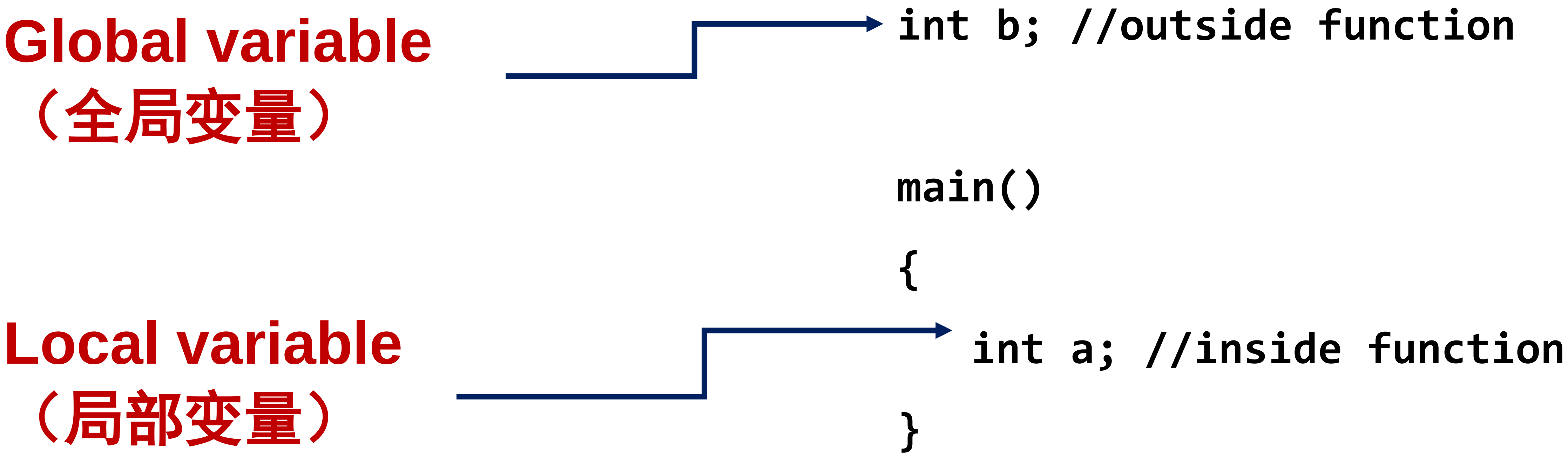


main

function1

Variable scope

Scope is a region of the program where defined variables are valid and beyond that variables cannot be accessed.



Global variable
（全局变量）

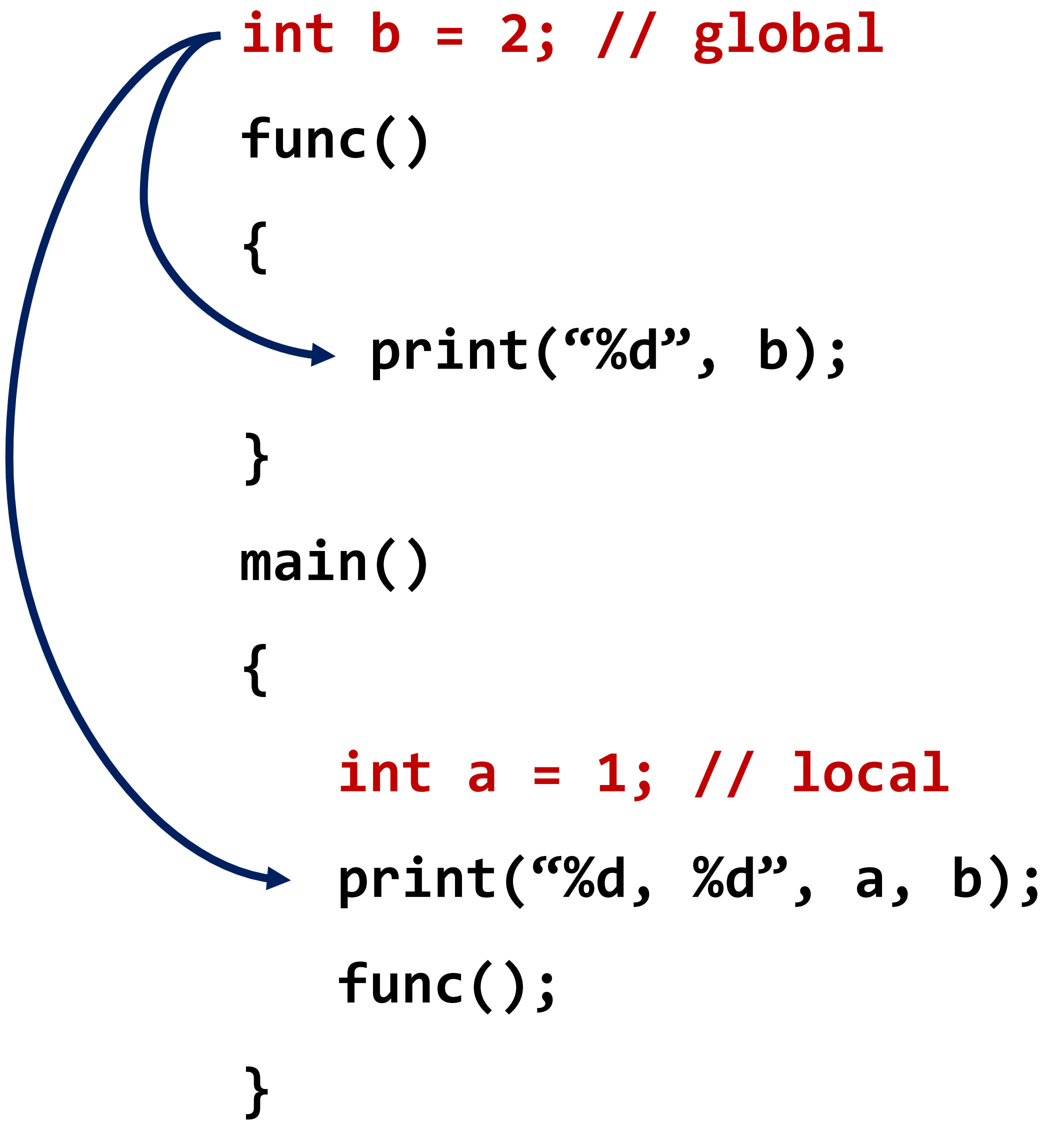
Local variable
（局部变量）

```
int b; //outside function

main()
{
   int a; //inside function
}
```

Variable scope

Global variable is visible everywhere

Local variable is only visible inside the function where it is defined



```
int b = 2; // global
func()
{
    print(“%d”, b);
}
main()
{
    int a = 1; // local
    print(“%d, %d”, a, b);
    func();
}
```

Variable scope

Global and local variables can share the same name

Local variable has the priority!!!

```
int b = 2; // global
func()
{                  ②
    print(“%d”, b); //print 2
    b = 20;
}
main()
{
   int b = 5; // local  ①
   print(“%d”, b); //print 5
   b = 10;
   func();         ③
   print(“%d”, b); //print 10
}
```

Variable scope

```
float PI = 3.14; // global
float func(float a)
{
    return a * PI;
}
main()
{
   float a = 5; // local
   float b = func(a);
   print(“%d”, b);
}
```

✓ **Do not use global variables unless**

- It is a **constant** that can be used everywhere (consensus)
- Its value needs to be **shared and changed** in multiple blocks or threads (e.g. bank account)
- Limited **memory** resources (embedded system)

✓ **Use local variables as much as possible!**

Storage classes for variable

identifier int a = 5;

自动（局部变量默认）auto int a = 5;

静态 static int a = 5;

外部 extern int a = 5;

寄存器（局部变量默认）register int a = 5;

Static variable

For static variables, memory is allocated only once and storage duration remains until the program terminates. **By default, global variables have static storage duration.**

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

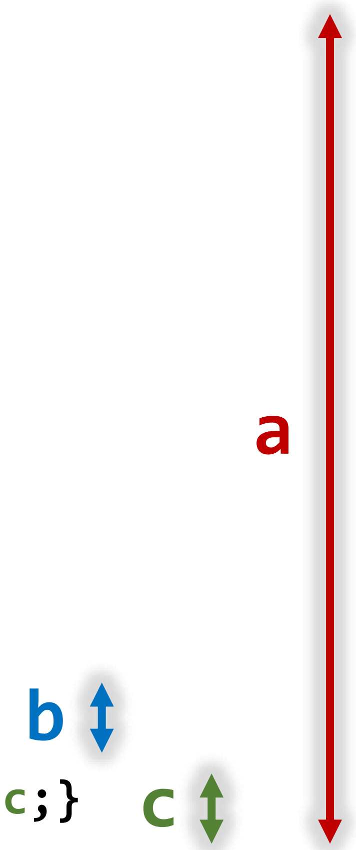
```
#include <stdio.h>
void increment()
{
   int x = 1;
   printf("%d\n", x);
   x = x + 1;
}
main()
{
  increment();  1
  increment();  1
}
```

```
#include <stdio.h>
void increment()
{
   static int x = 1;
   printf("%d\n", x);
   x = x + 1;
}
main()
{
  increment();  1
  increment();  2
}
```

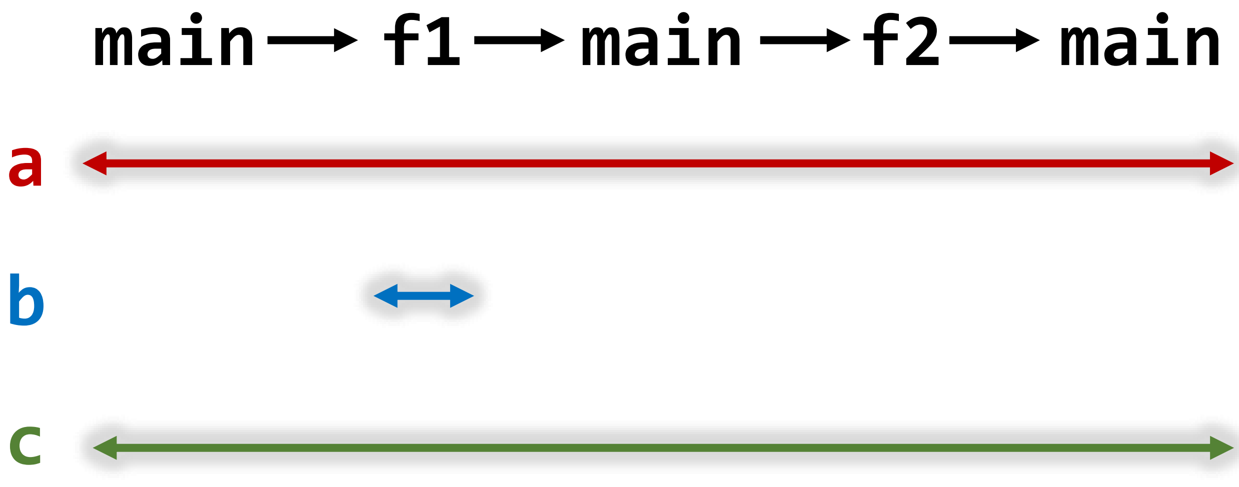
Variable scope

Scope in space

```
int a;
f1();
f2();
main()
{
    f1();
    f2();
}
f1(){int b;}
f2(){static int c;}
```



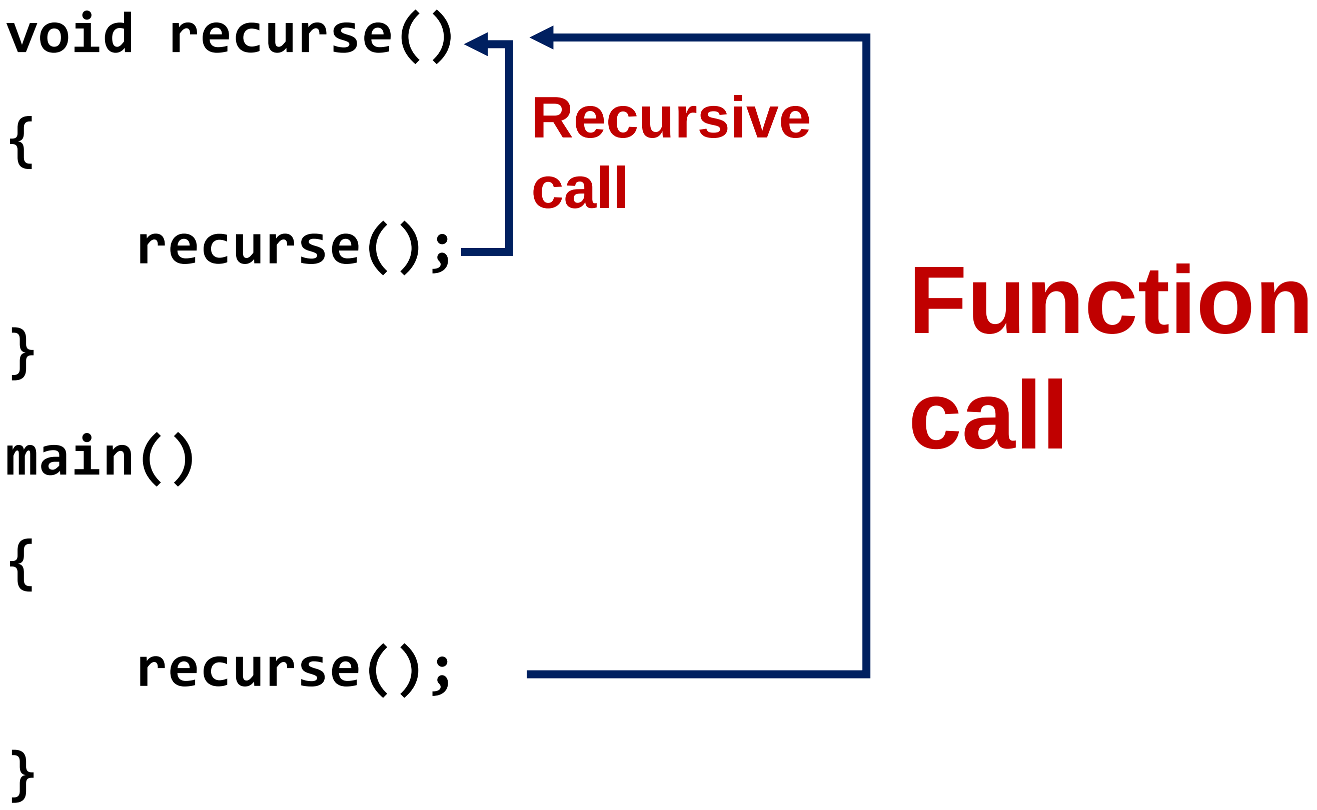
Scope in time


main → f1 → main → f2 → main

局部静态变量一直存在，但只在函数内可见

Recursion

Recursion is to repeat the same procedure again and again



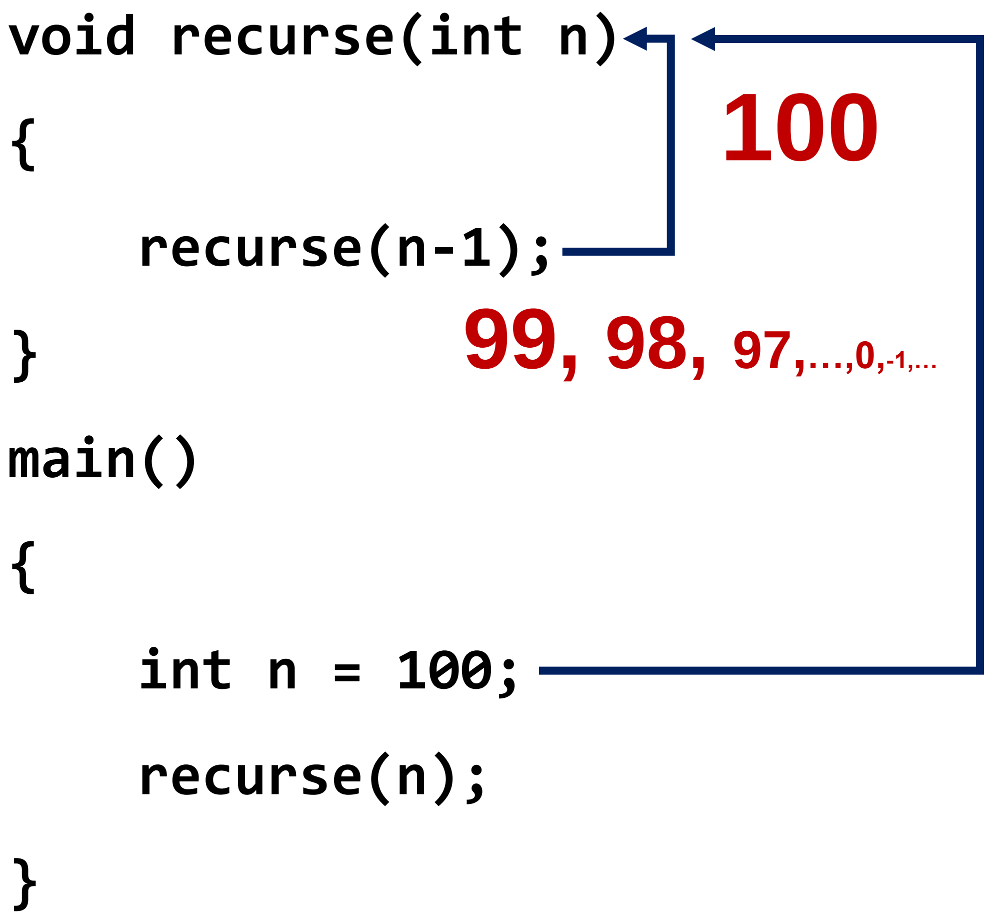
```
void recurse()
{
    recurse();
}
main()
{
    recurse();
}
```

Recursive call

Function call

Recursion

Recursively subtract

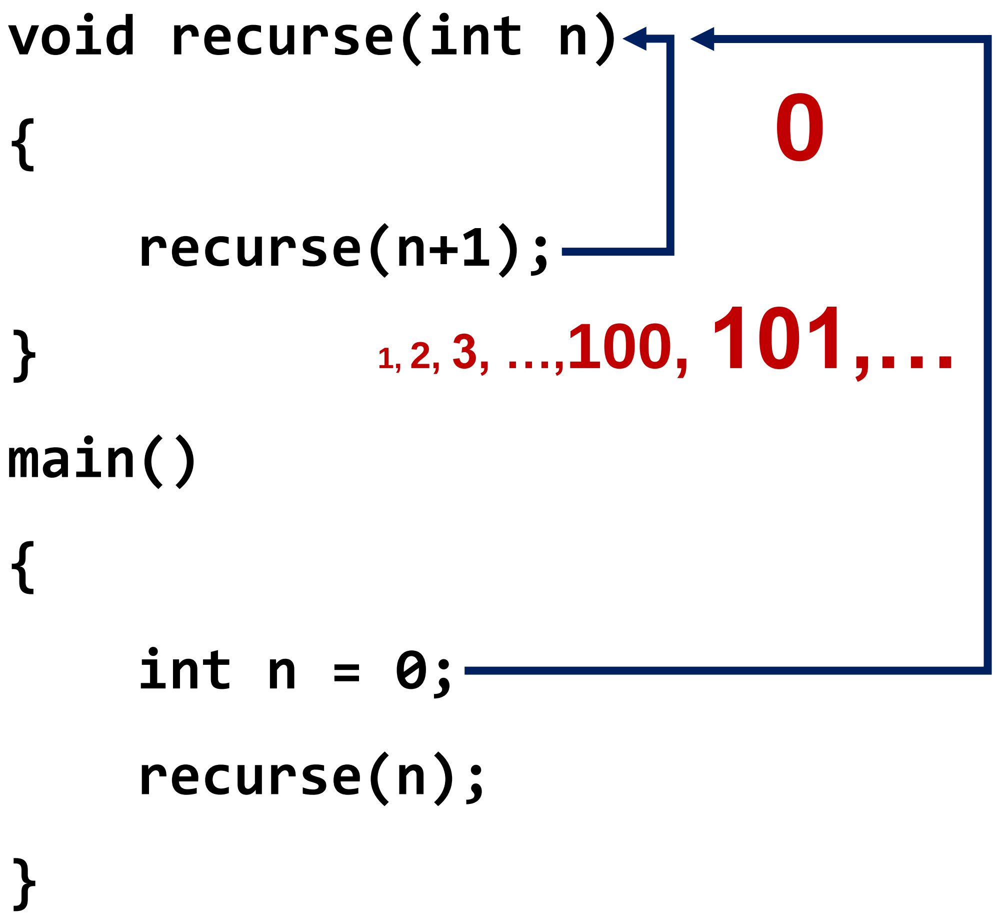


```
void recurse(int n)
{
    recurse(n-1);
}
main()
{
    int n = 100;
    recurse(n);
}
```

100

99, 98, 97,...,0,-1,...

Recursively add



```
void recurse(int n)
{
    recurse(n+1);
}
main()
{
    int n = 0;
    recurse(n);
}
```

0

1, 2, 3, ...,100, 101,...

Recursion

```
void recurse(int n)
{
   if (n == 0) return;
   recurse(n-1);
}
main()
{
    int n = 100;
    recurse(n);
}
```

Which one can leave?

```
void recurse(int n)
{
    recurse(n-1);
    if (n == 0) return;
}
main()
{
    int n = 100;
    recurse(n);
}
```

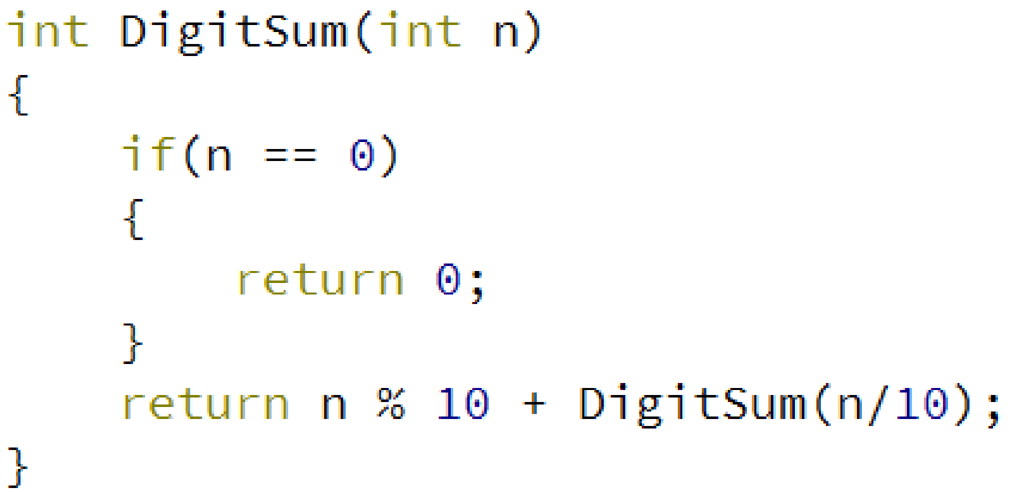
Summary

- Three steps to create a function: **function declaration, definition, calling**. Function must be declared in front of the place where it is called (e.g. before the main)
- Variable has its scope both in space and time. **Global variable (outside function)** is visible everywhere, **local variable (inside function)** is only visible in the function block. Variable can have **identifiers** (auto, static, extern, register).
- **Recursion** can be implemented by calling a function itself repeatedly.

5 questions

1. You can only output the results from the function by returning a value. Yes | No

2. When input is “1024”, which is output of following function? ( )
A. 4201   B. 7   C. 1024   D. 10



```
int DigitSum(int n)
{
    if(n == 0)
    {
        return 0;
    }
    return n % 10 + DigitSum(n/10);
}
```

3. What are the differences between (i) local variable and global variable, (ii) parameters and arguments?

5 questions

4. What is the result of following code? ()

A. a = 1, b = 2
B. a = 5, b = 10
C. a = 15, b = 10
D. a = 1, b = 10

```
#include<stdio.h>

int a = 1;

int fun(int a, int *b)
{
    a = 5;
    *b = 10;

    return a + *b;
}

int main()
{
    int a = 1, b = 2;
    a = fun(a, &b);
    printf("a = %d, b = %d", a, b);
    return 0;
}
```

5. Write a function to recursively sum from 1 to 100, in step of 2 (e.g. 1 + 3 + 5 + … 99)

```
int recurse(int N)
{
    if (N > 100)
    {
        return N;
    }
    else
    {
        return N + recurse(N + 2);
    }
}
```

Content

1. Array & string
2. Functions
3. Pointer

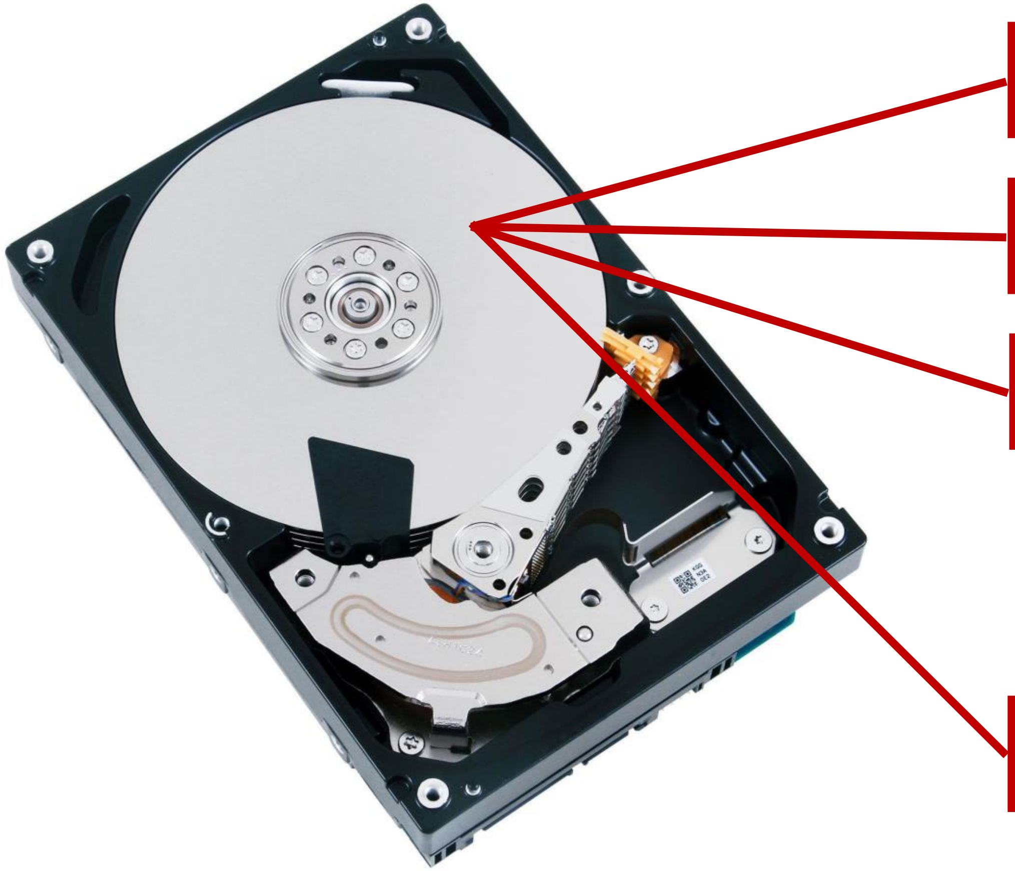
Memory address

The memory address is the location of where the variable is stored on a PC.

When a variable is created in C, a memory address is assigned to the variable.

When we assign a value to the variable, it is stored in this memory address.

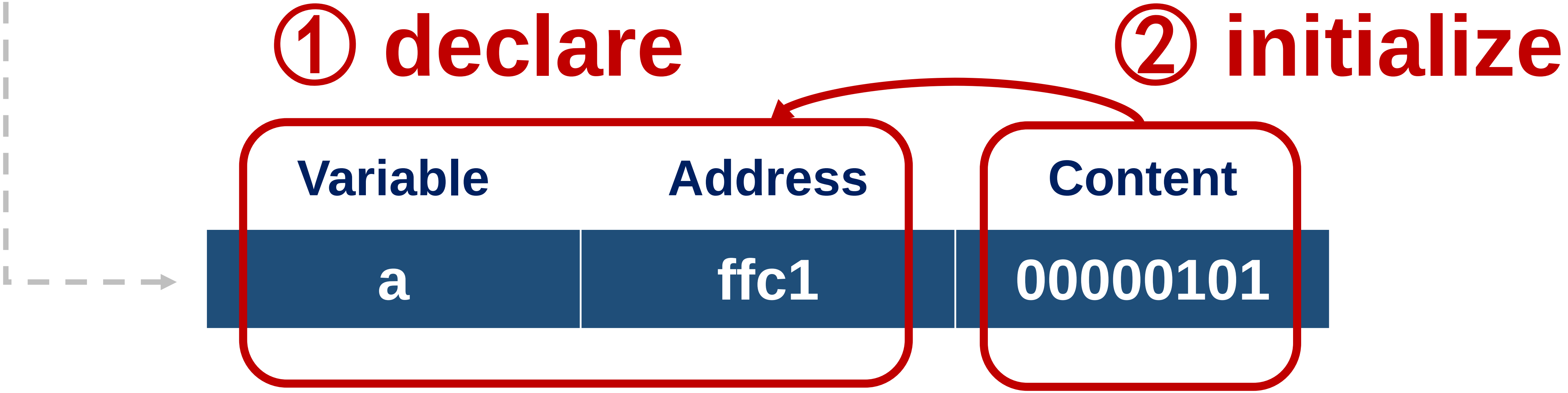
Memory address



Address	Content							
ffc1	0	1	0	0	1	0	1	1
ffc2	0	1	0	1	0	0	0	1
ffc3	1	0	0	0	1	1	1	0
⋮	⋮							
ffc9	1	1	0	0	0	0	0	1

Memory address

```
int a = 5; ⎰ int a;//declare
           ⎱ a = 5;//initialize
```



① declare

② initialize

Variable	Address	Content
a	ffc1	00000101

Memory address

```
int a = 5;
int b = 2;
int c = 1;
```

Variable	Address	Content
a	ffc1	00000101
b	ffc2	00000010
c	ffc3	00000001

What happens in the memory allocation?

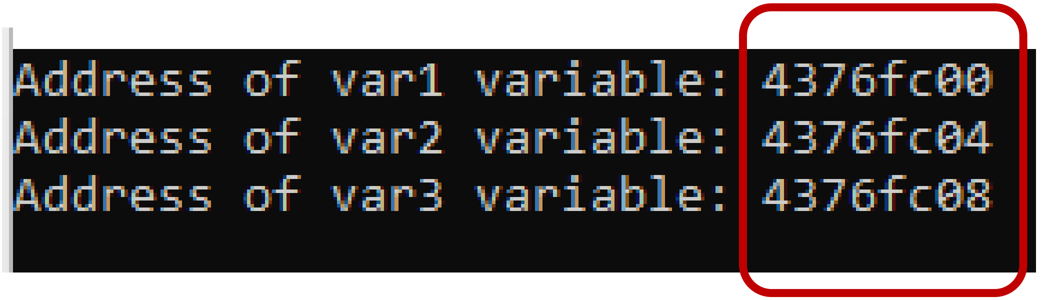
How to check variable address

Use **& (reference operator)** to check the variable address

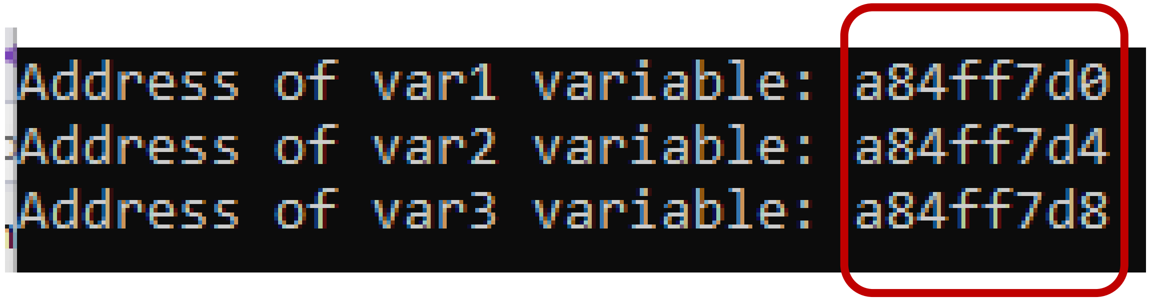
```
#include <stdio.h>

main ()
{
   int var1;
   float var2;
   char var3;
   printf("Address of var1 variable: %x\n", &var1);
   printf("Address of var2 variable: %x\n", &var2);
   printf("Address of var3 variable: %x\n", &var3);
}
```

How to check variable address

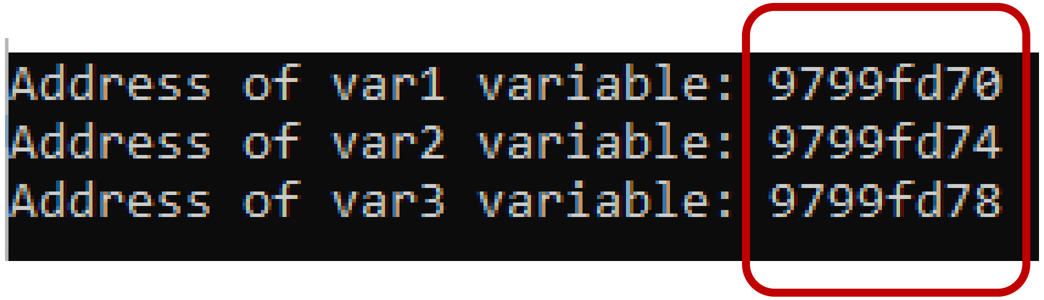
Run multiple times, every time the address is different, but it has orders!



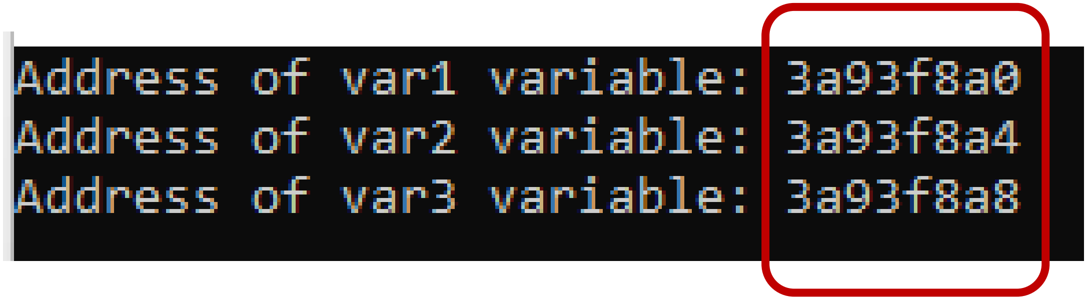
```
Address of var1 variable: 4376fc00
Address of var2 variable: 4376fc04
Address of var3 variable: 4376fc08
```



```
Address of var1 variable: a84ff7d0
Address of var2 variable: a84ff7d4
Address of var3 variable: a84ff7d8
```



```
Address of var1 variable: 9799fd70
Address of var2 variable: 9799fd74
Address of var3 variable: 9799fd78
```



```
Address of var1 variable: 3a93f8a0
Address of var2 variable: 3a93f8a4
Address of var3 variable: 3a93f8a8
```

What is Hexadecimal?

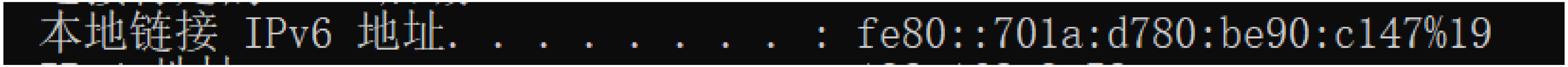
Decimal number system

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16

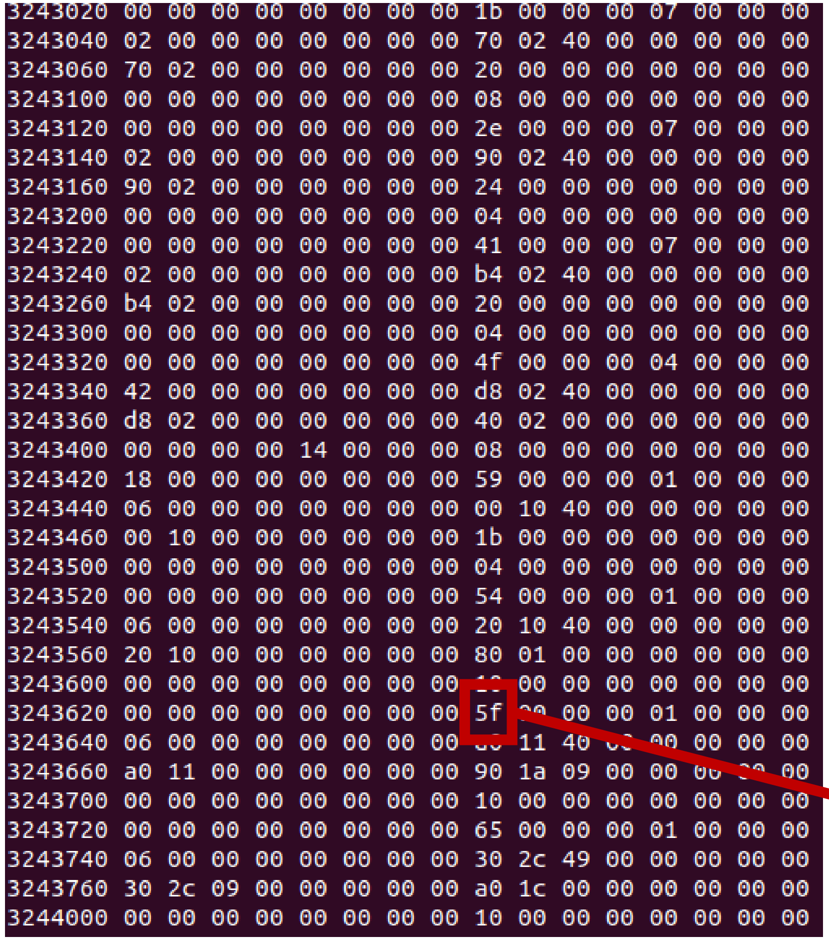
0 1 2 3 4 5 6 7 8 9 A B C D E F 10

Hexadecimal number system

Hexadecimal is everywhere

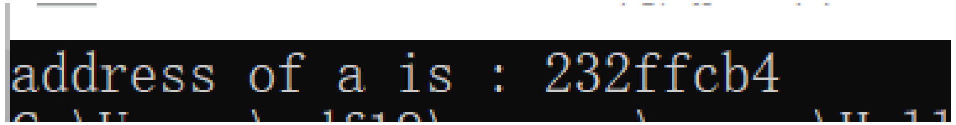


```
本地链接 IPv6 地址. . . . . . . . : fe80::701a:d780:be90:c147%19
```

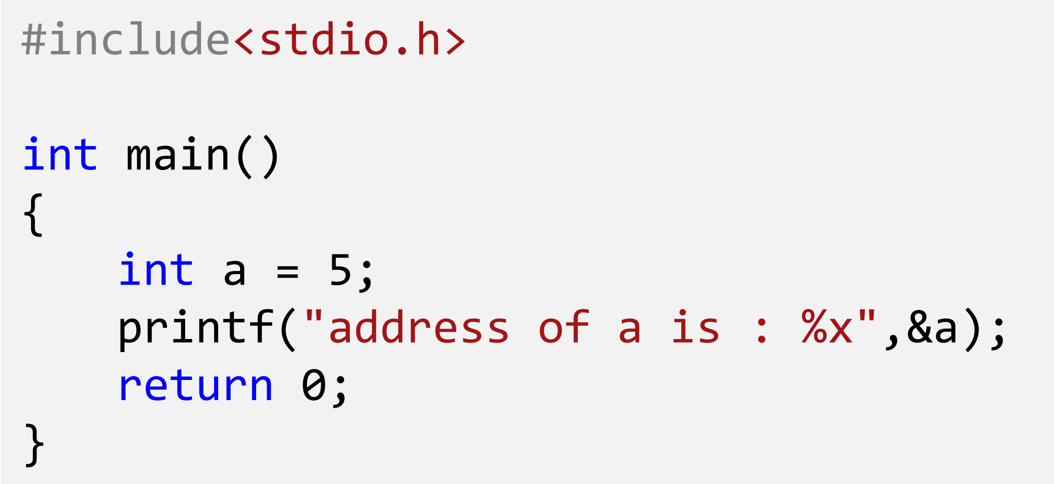


```
3243020 00 00 00 00 00 00 00 00 1b 00 00 00 07 00 00 00
3243040 02 00 00 00 00 00 00 00 70 02 40 00 00 00 00 00
3243060 70 02 00 00 00 00 00 00 20 00 00 00 00 00 00 00
3243100 00 00 00 00 00 00 00 00 08 00 00 00 00 00 00 00
3243120 00 00 00 00 00 00 00 00 2e 00 00 00 07 00 00 00
3243140 02 00 00 00 00 00 00 00 90 02 40 00 00 00 00 00
3243160 90 02 00 00 00 00 00 00 24 00 00 00 00 00 00 00
3243200 00 00 00 00 00 00 00 00 04 00 00 00 00 00 00 00
3243220 00 00 00 00 00 00 00 00 41 00 00 00 07 00 00 00
3243240 02 00 00 00 00 00 00 00 b4 02 40 00 00 00 00 00
3243260 b4 02 00 00 00 00 00 00 20 00 00 00 00 00 00 00
3243300 00 00 00 00 00 00 00 00 04 00 00 00 00 00 00 00
3243320 00 00 00 00 00 00 00 00 4f 00 00 00 04 00 00 00
3243340 42 00 00 00 00 00 00 00 d8 02 40 00 00 00 00 00
3243360 d8 02 00 00 00 00 00 00 40 02 00 00 00 00 00 00
3243400 00 00 00 00 14 00 00 00 08 00 00 00 00 00 00 00
3243420 18 00 00 00 00 00 00 00 59 00 00 00 01 00 00 00
3243440 06 00 00 00 00 00 00 00 00 10 40 00 00 00 00 00
3243460 00 10 00 00 00 00 00 00 1b 00 00 00 00 00 00 00
3243500 00 00 00 00 00 00 00 00 04 00 00 00 00 00 00 00
3243520 00 00 00 00 00 00 00 00 54 00 00 00 01 00 00 00
3243540 06 00 00 00 00 00 00 00 20 10 40 00 00 00 00 00
3243560 20 10 00 00 00 00 00 00 80 01 00 00 00 00 00 00
3243600 00 00 00 00 00 00 00 00 10 00 00 00 00 00 00 00
3243620 00 00 00 00 00 00 00 00 5f 00 00 00 01 00 00 00
3243640 06 00 00 00 00 00 00 00 a0 11 40 00 00 00 00 00
3243660 a0 11 00 00 00 00 00 00 90 1a 09 00 00 00 00 00
3243700 00 00 00 00 00 00 00 00 10 00 00 00 00 00 00 00
3243720 00 00 00 00 00 00 00 00 65 00 00 00 01 00 00 00
3243740 06 00 00 00 00 00 00 00 30 2c 49 00 00 00 00 00
3243760 30 2c 09 00 00 00 00 00 a0 1c 00 00 00 00 00 00
3244000 00 00 00 00 00 00 00 00 10 00 00 00 00 00 00 00
```

5f



```
address of a is : 232ffcb4
```



```c
#include<stdio.h>

int main()
{
    int a = 5;
    printf("address of a is : %x",&a);
    return 0;
}
```

What is pointer?

Variable → Address

指针：存储地址的变量

What is pointer?

Pointer is a variable that stores the address of another variable.

```
type var1;
type *var2 = &var1;
```

```
int a;
float f;
char c;
```

Stores value

```
int *a;
float *f;
char *c;
```

Stores address

What is pointer?

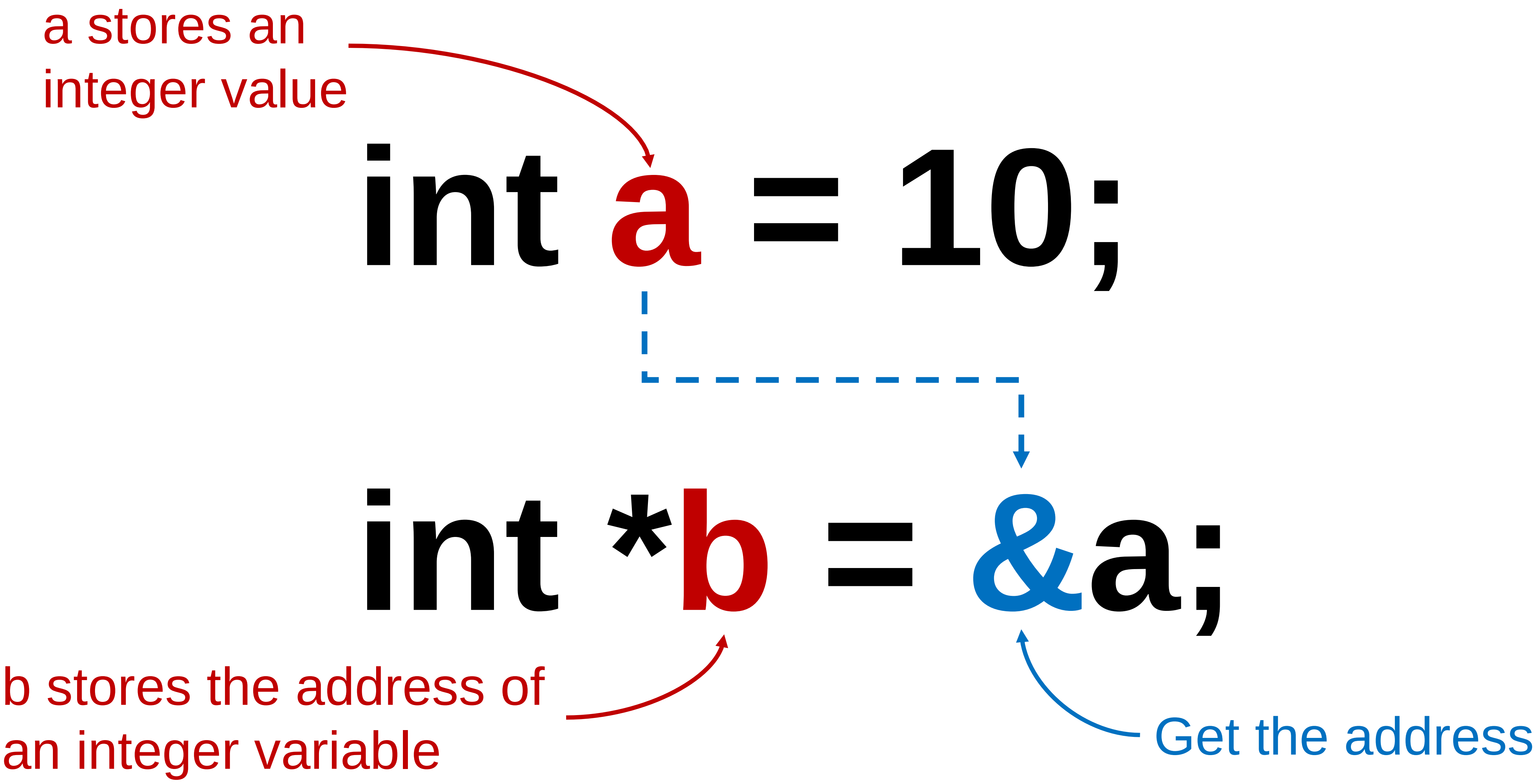
```
int a;
```

- a has type of int
- a stores value

```
int *b;
```

- b has type of int*
- b stores address

What is pointer?



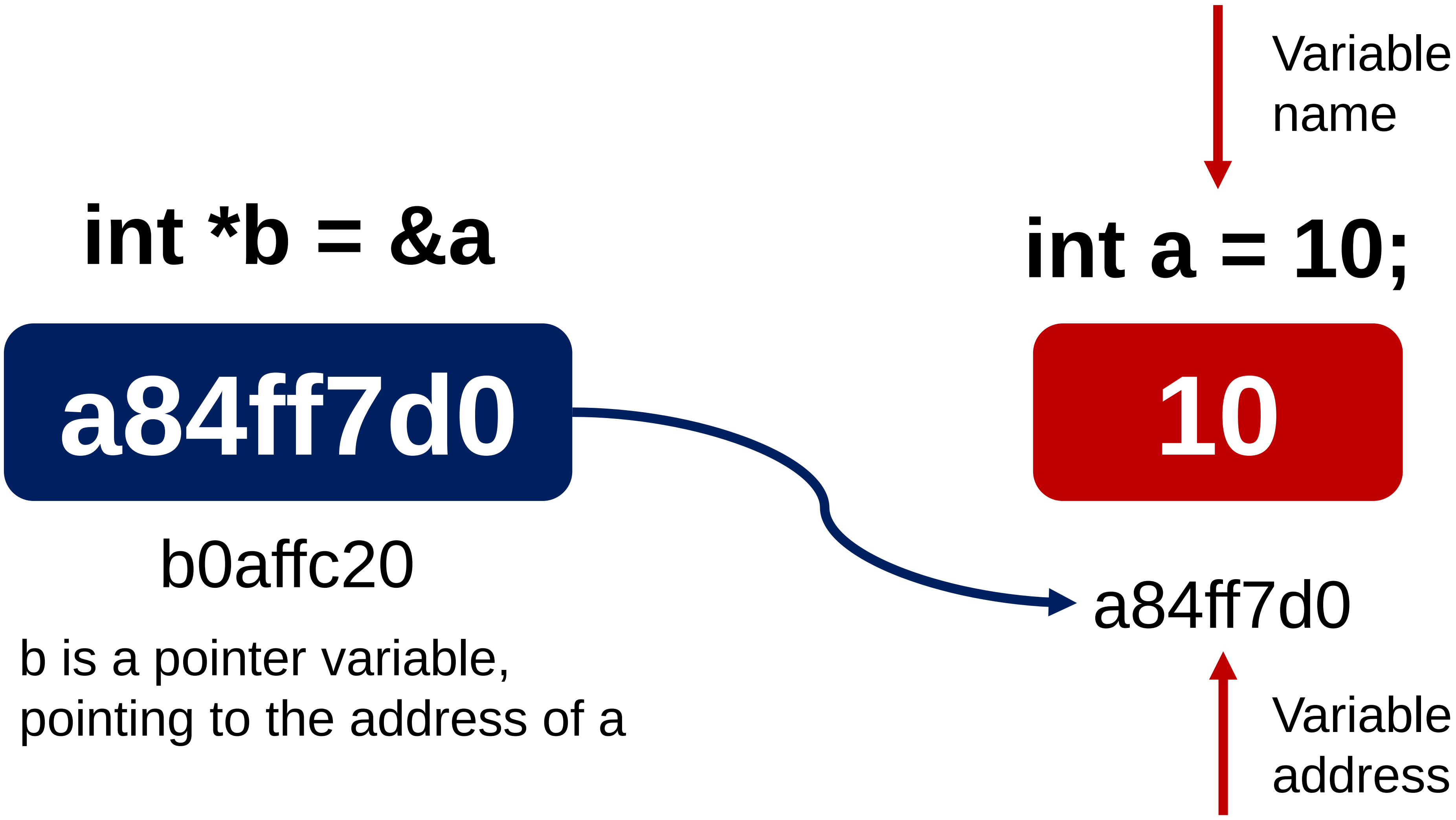
a stores an
integer value

```
int a = 10;

int *b = &a;
```

b stores the address of
an integer variable

Get the address

What is pointer?



Variable name

int *b = &a

int a = 10;

a84ff7d0

10

b0affc20

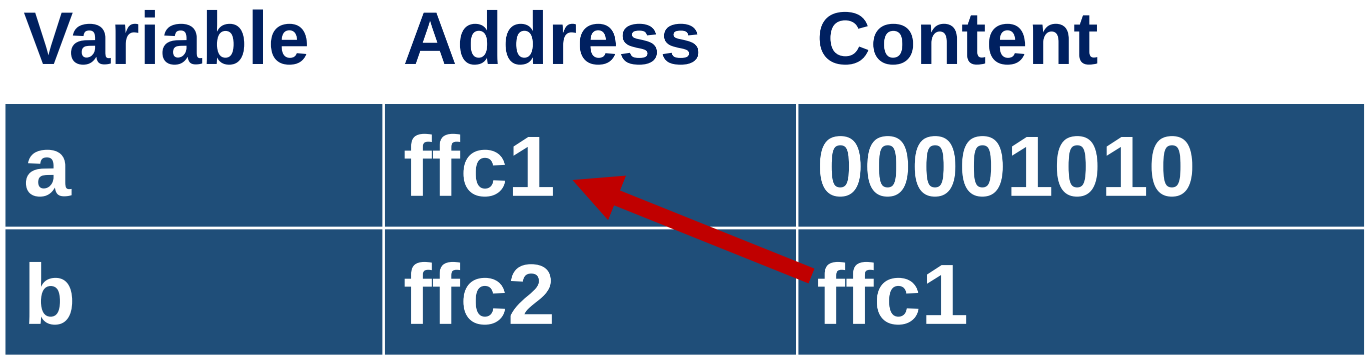
a84ff7d0

b is a pointer variable, pointing to the address of a

Variable address

What is pointer?

```
int a = 5;
int *b = &a;
```



Variable	Address	Content
a	ffc1	00001010
b	ffc2	ffc1

- a stores the value of 10
- b stores the address of a

How to interpret pointer?

int *b

b has data type int*

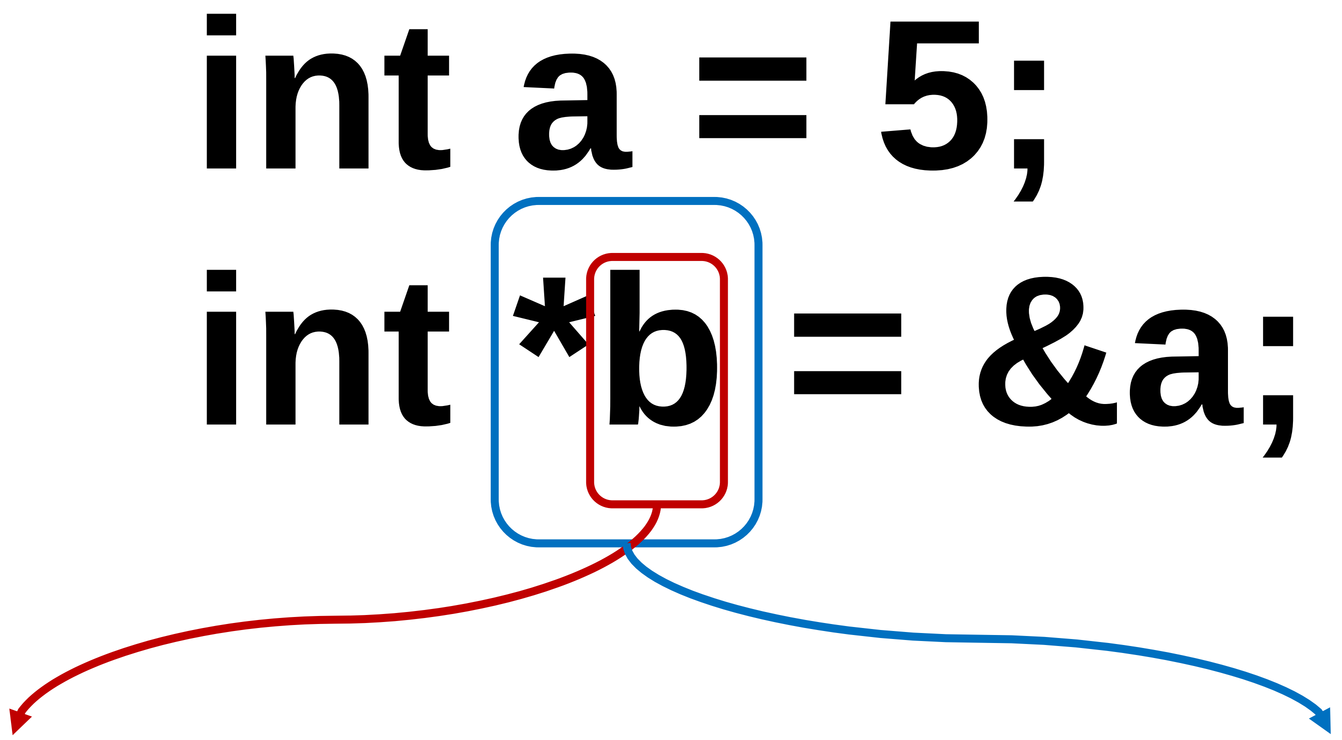
```
printf("%x", b);//address
```

int *b

*b has data type int

```
printf("%d", *b);//value
```

How to interpret pointer?



```
int a = 5;
int *b = &a;
```

Use **b** to check
the address of a

Use __*b__ to check
the value of a

How to define pointer?

Pointer stores address, not value!

```
int a = 5;
int *b = &a;
```

✓

```
int a = 5;
int *b = 10;
```

✗

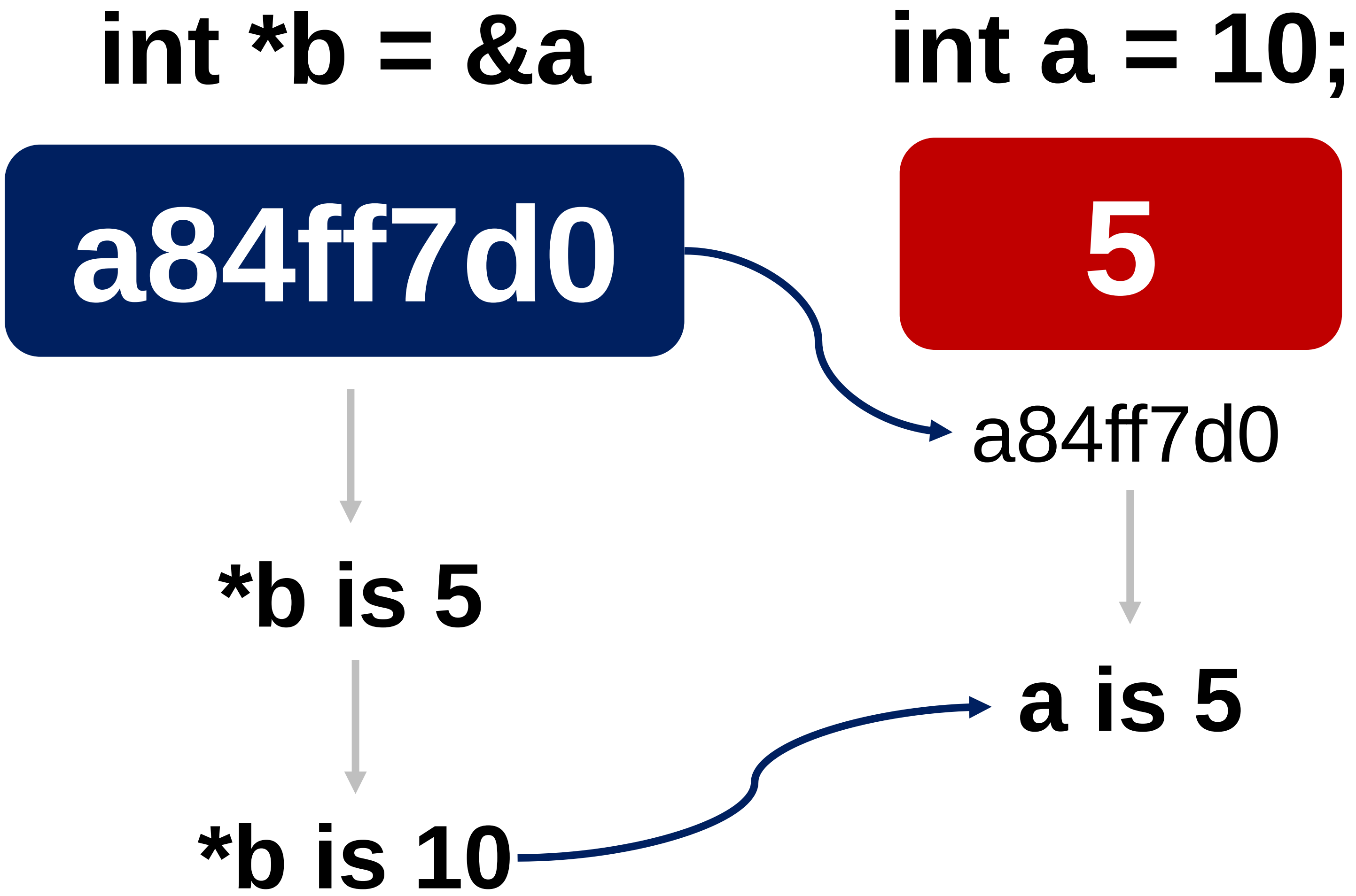
```
int a = 5;
int *b = &a;
*b = 10;
```

✓

How to define pointer?

```
int a = 5;
int *b = &a;

*b = 10;
```

What is a?



int *b = &a

a84ff7d0

int a = 10;

5

a84ff7d0

***b is 5**

a is 5

***b is 10**

How to use pointer?

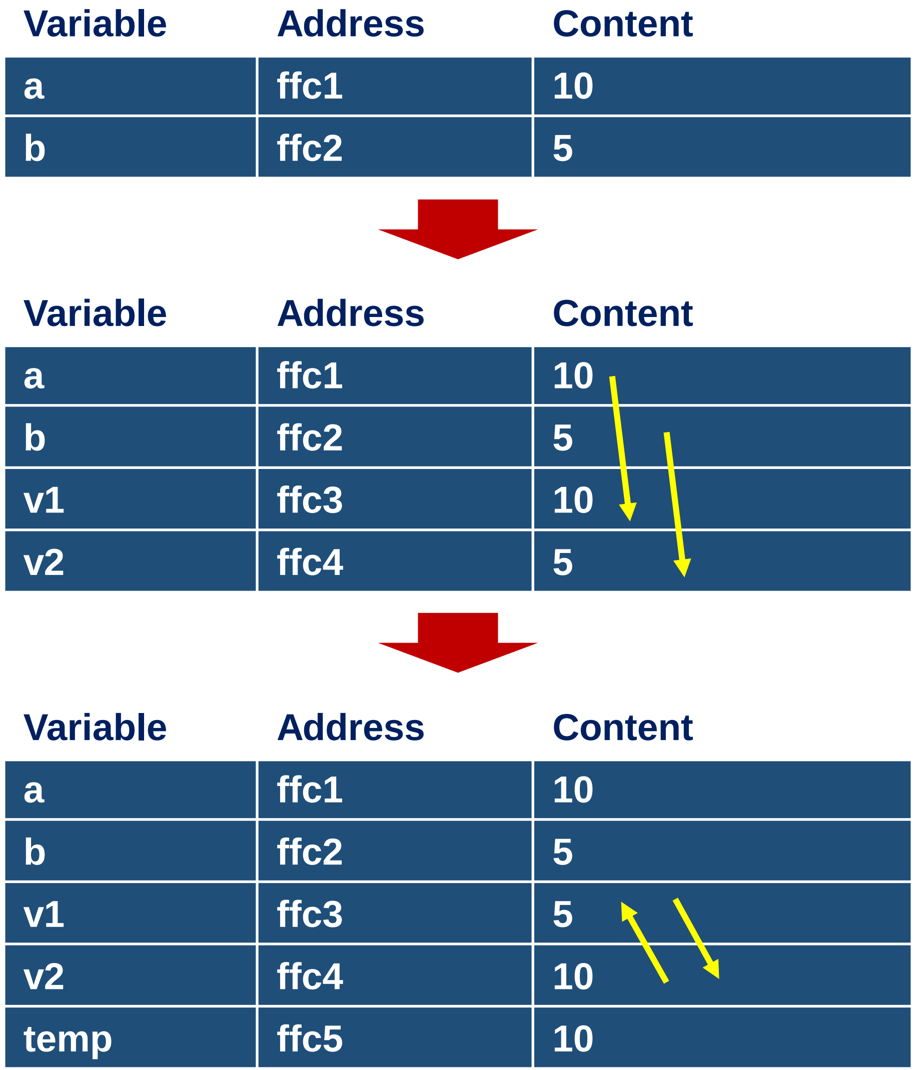
1. Use pointer for functions to pass values

2. Use pointer for array operations (elements in an array has continuous address)

Use pointer for functions

Values cannot be swapped

```
void swap(int v1, int v2)
{
   printf(“Before: v1=%d, v2=%d\n", v1, v2);
   int temp;
   temp = v1;
   v1 = v2;
   v2 = temp;
   printf(“After: v1=%d, v2=%d\n", v1, v2);
}

main()
{
    int a = 10, b = 5;
    printf(“Before: a=%d, b=%d\n", a, b);
    swap(a, b);
    printf(“After: a=%d, b=%d\n", a, b);
}
```



Variable	Address	Content
a	ffc1	10
b	ffc2	5

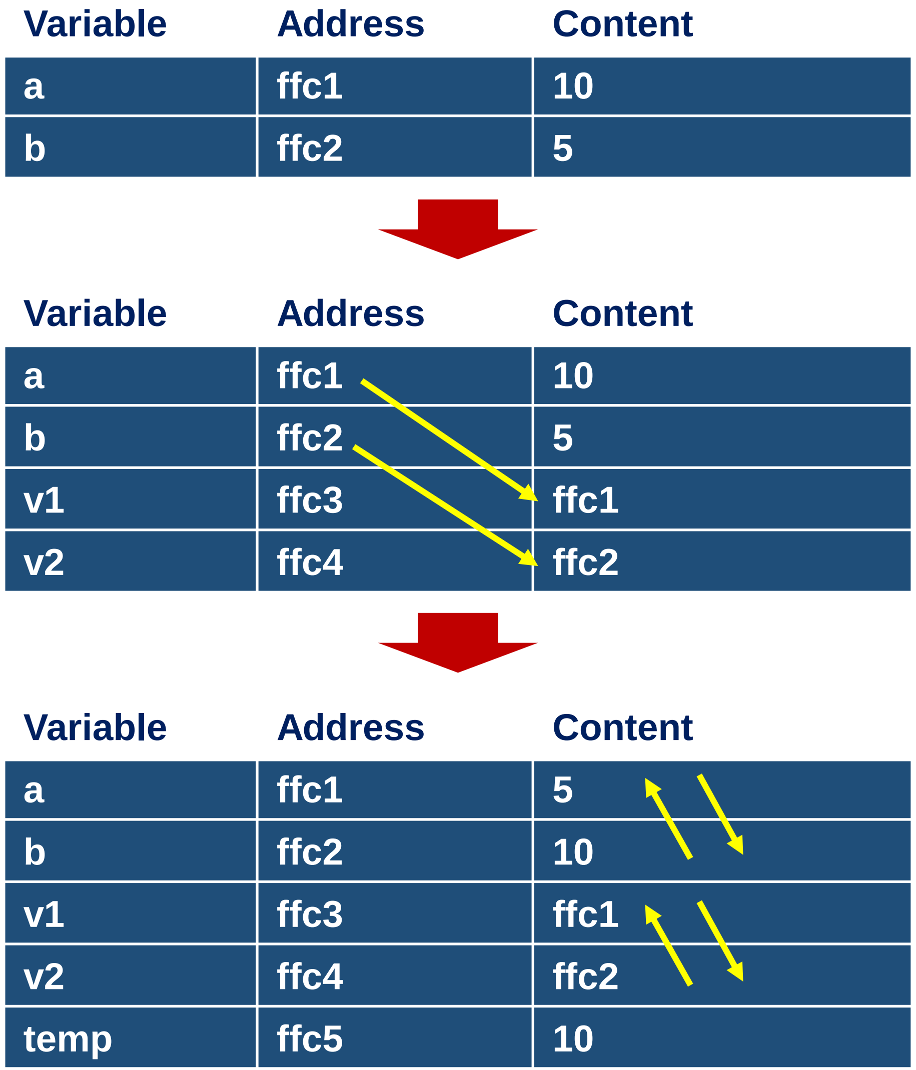
Variable	Address	Content
a	ffc1	10
b	ffc2	5
v1	ffc3	10
v2	ffc4	5

Variable	Address	Content
a	ffc1	10
b	ffc2	5
v1	ffc3	5
v2	ffc4	10
temp	ffc5	10

Use pointer for functions

Values can be swapped

```
void swap(int *v1, int *v2)
{
   int temp;
   temp = *v1;
   *v1 = *v2;
   *v2 = temp;
}

main()
{
    int a = 10, b = 5;
    printf(“Before: a=%d, b=%d\n", a, b);
    swap(&a, &b);
    printf(“After: a=%d, b=%d\n", a, b);
}
```



Variable	Address	Content
a	ffc1	10
b	ffc2	5

Variable	Address	Content
a	ffc1	10
b	ffc2	5
v1	ffc3	ffc1
v2	ffc4	ffc2

Variable	Address	Content
a	ffc1	5
b	ffc2	10
v1	ffc3	ffc1
v2	ffc4	ffc2
temp	ffc5	10

Use pointer for functions

How to output multiple results from a function?

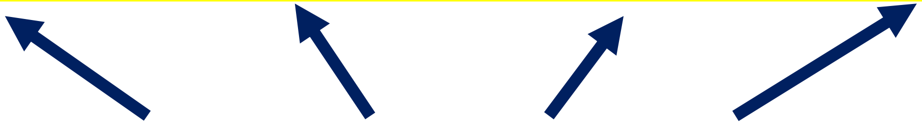
```
int func(int v1, int v2)
{
    int v3 = v1 + v2;
    int v4 = v1 - v2;
    return v3;
}

main()
{
    int a = 10, b = 5;
    int c = func(a, b);
}
```

We did multiple operations but only return one result!

Use pointer for functions

How to output multiple results from a function?

```
void func(int v1, int v2, int* sum, int* sub, int* mul, int* div)
{
    *sum = v1 + v2;
    *sub = v1 - v2;
    *mul = v1 * v2;
    *div = v1 / v2;
}

main()
{
    int a = 10, b = 5, sum, sub, mul, div;
    int sum = func(a, b, &sum, &sub, &mul, &div);
}
```



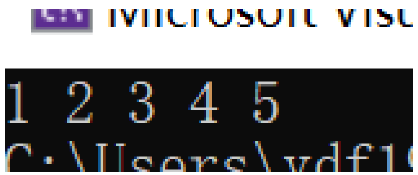
Pass out four results

Use pointer for functions

```
int * myFunction()
{

 . . .

}
```

```
int* merge(int a, int b, int c, int d, int e)
{
    int* array = (int*)malloc(sizeof(int) * 5);
    array[0] = a;
    array[1] = b;
    array[2] = c;
    array[3] = d;
    array[4] = e;
    return array;
}

main()
{
    int* array = merge(1, 2, 3, 4, 5);
    for (int i = 0; i < 5; i++)
    printf("%d ", array[i]);
}
```

动态数组



Microsoft Visu

```
1 2 3 4 5
C:\Users\ydf1
```

Pointer points to array

```
int a = 5;
int *b = &a;
```

↓

Give the address of a to b!

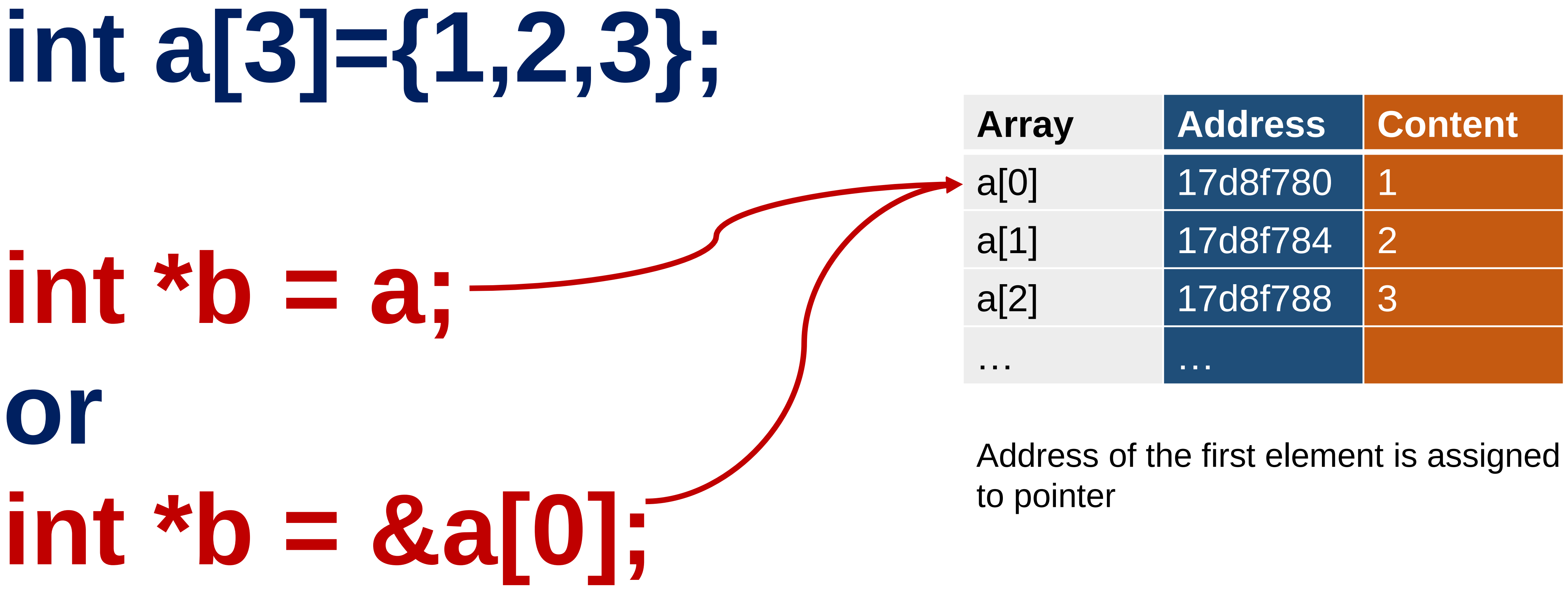
```
int a[10];
int *b = a;
```

↓

Give the address of **first element of a** to b!

```
int *b = &a[0];
```

Pointer points to array



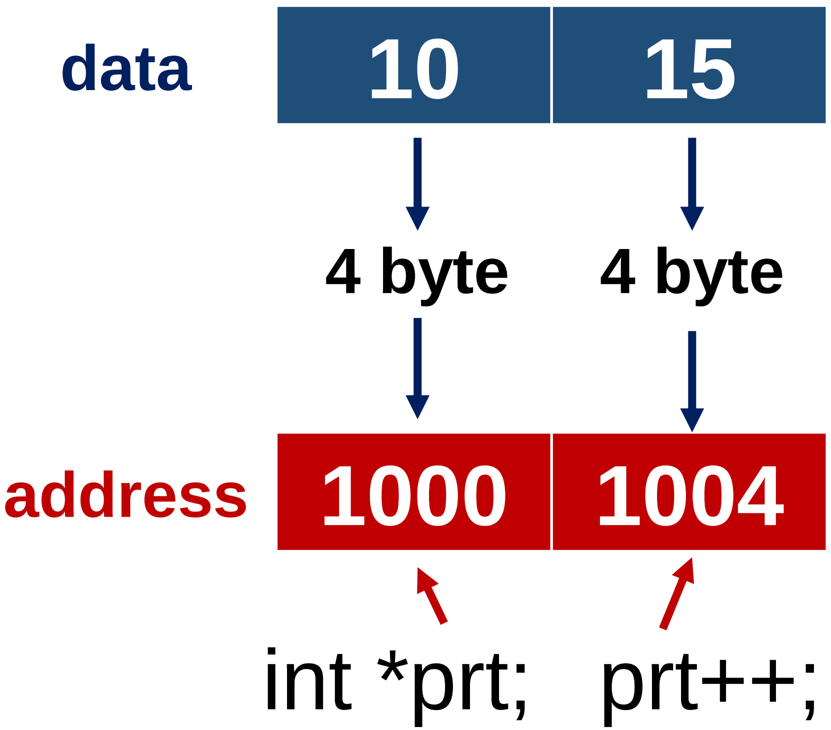
```
int a[3]={1,2,3};

int *b = a;
or
int *b = &a[0];
```

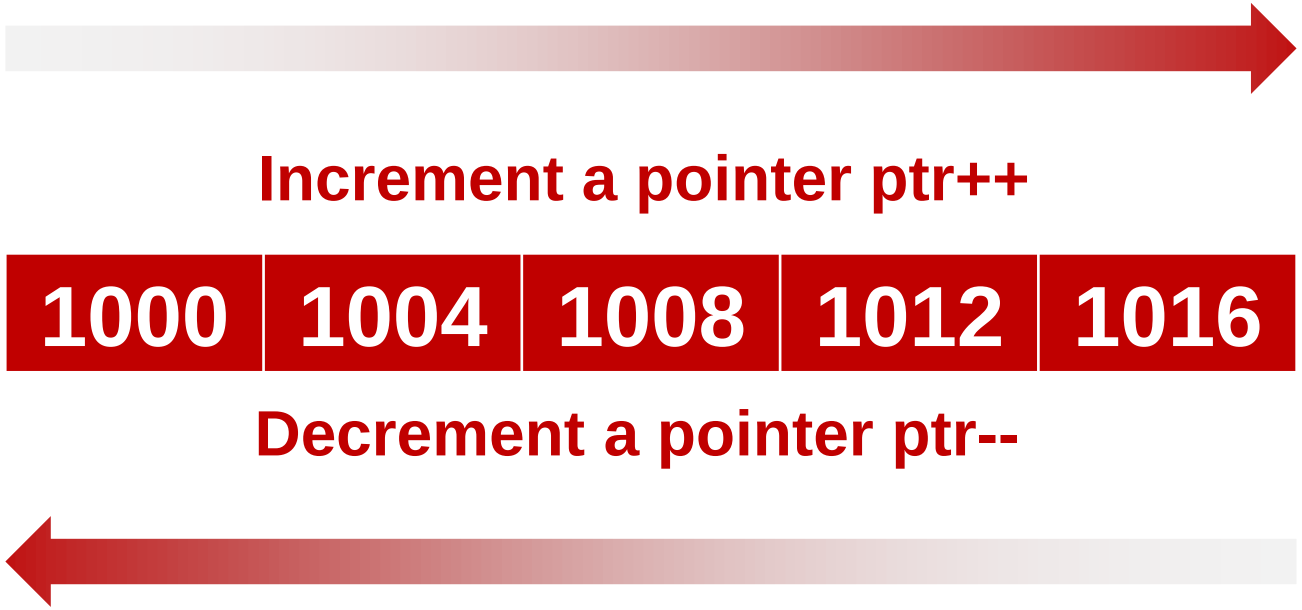
Array	Address	Content
a[0]	17d8f780	1
a[1]	17d8f784	2
a[2]	17d8f788	3
…	…	

Address of the first element is assigned to pointer

Pointer points to array

Four arithmetic operators that can be used on pointers: ++, --, +, -





Pointer points to array

How to access the elements in array?

```
int a[10];
int *b = a;
```

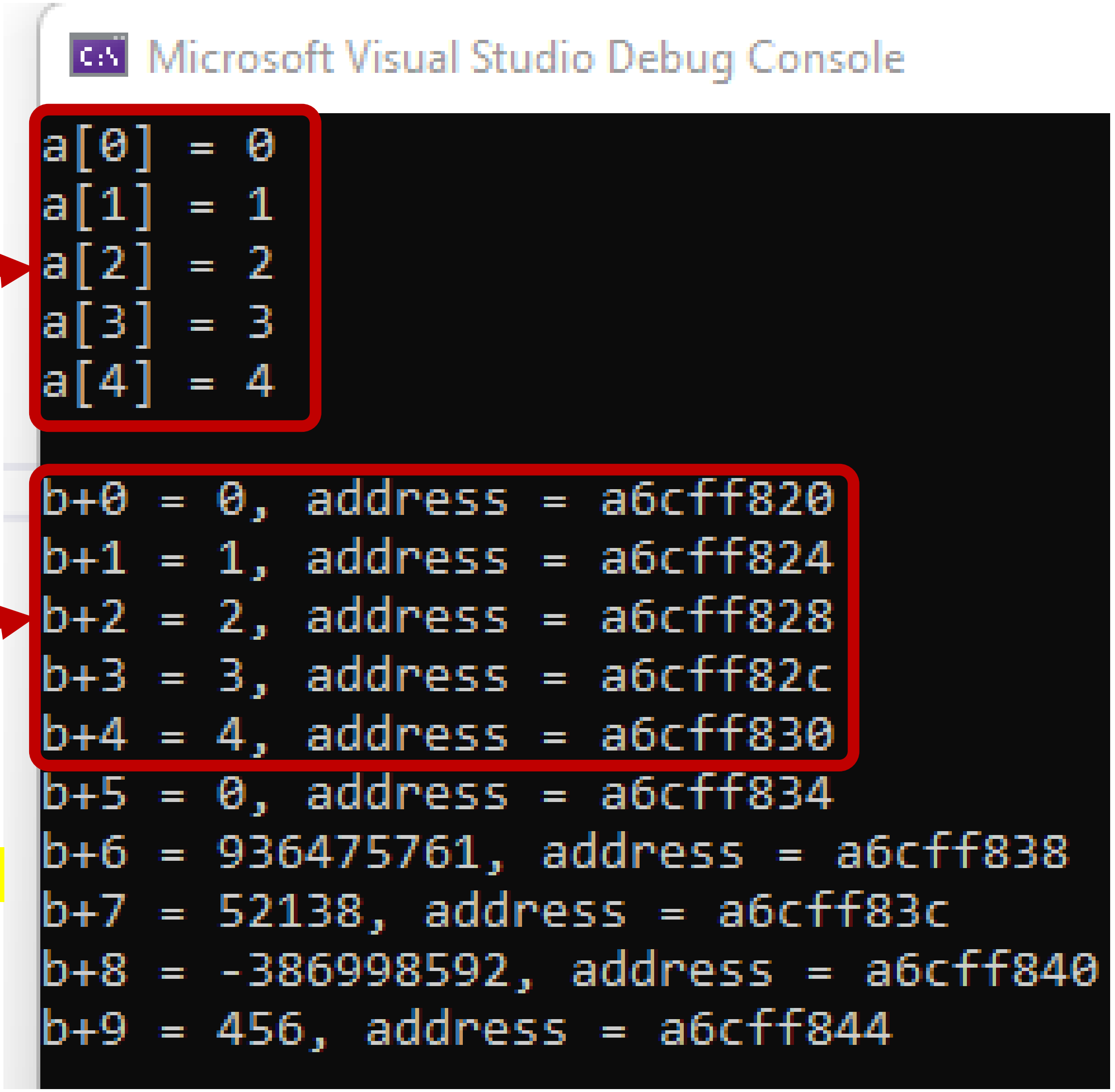
下标法：a[5]

指针法：*(b+5)

Use pointer for array

Use pointer to access array

```
#include<stdio.h>
main()
{
    int a[5] = {0, 1, 2, 3, 4};
    int* b = a;

    for (int i = 0; i < 5; i++)
    {
        printf("a[%d] = %d\n", i, a[i]);
    }

    for (int i = 0; i < 10; i++)
    {
        printf("b+%d = %d, address = %x\n", i,
*(b+i), b+i);
    }
}
```



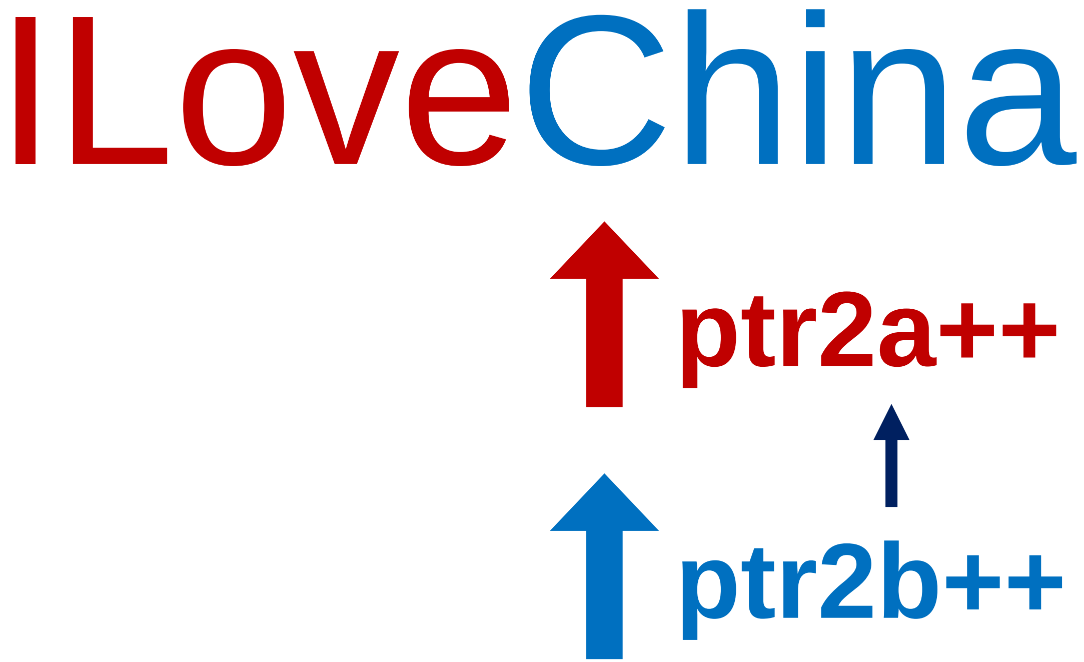
Microsoft Visual Studio Debug Console

```
a[0] = 0
a[1] = 1
a[2] = 2
a[3] = 3
a[4] = 4

b+0 = 0, address = a6cff820
b+1 = 1, address = a6cff824
b+2 = 2, address = a6cff828
b+3 = 3, address = a6cff82c
b+4 = 4, address = a6cff830
b+5 = 0, address = a6cff834
b+6 = 936475761, address = a6cff838
b+7 = 52138, address = a6cff83c
b+8 = -386998592, address = a6cff840
b+9 = 456, address = a6cff844
```

Use pointer for array

```
#include<stdio.h>

main()
{
    char a[100] = "ILove";
    char b[] = "China";

    char* ptr2a = &a[5]; // last address + 1
    char* ptr2b = &b[0]; // first address

    for (int i = 0; i < sizeof(b); i++)
    {
        *ptr2a = *ptr2b;
        ptr2a++;
        ptr2b++;
    }

    printf("%s\n", a);
}
```

How to concatenate 2 strings?



ILoveChina

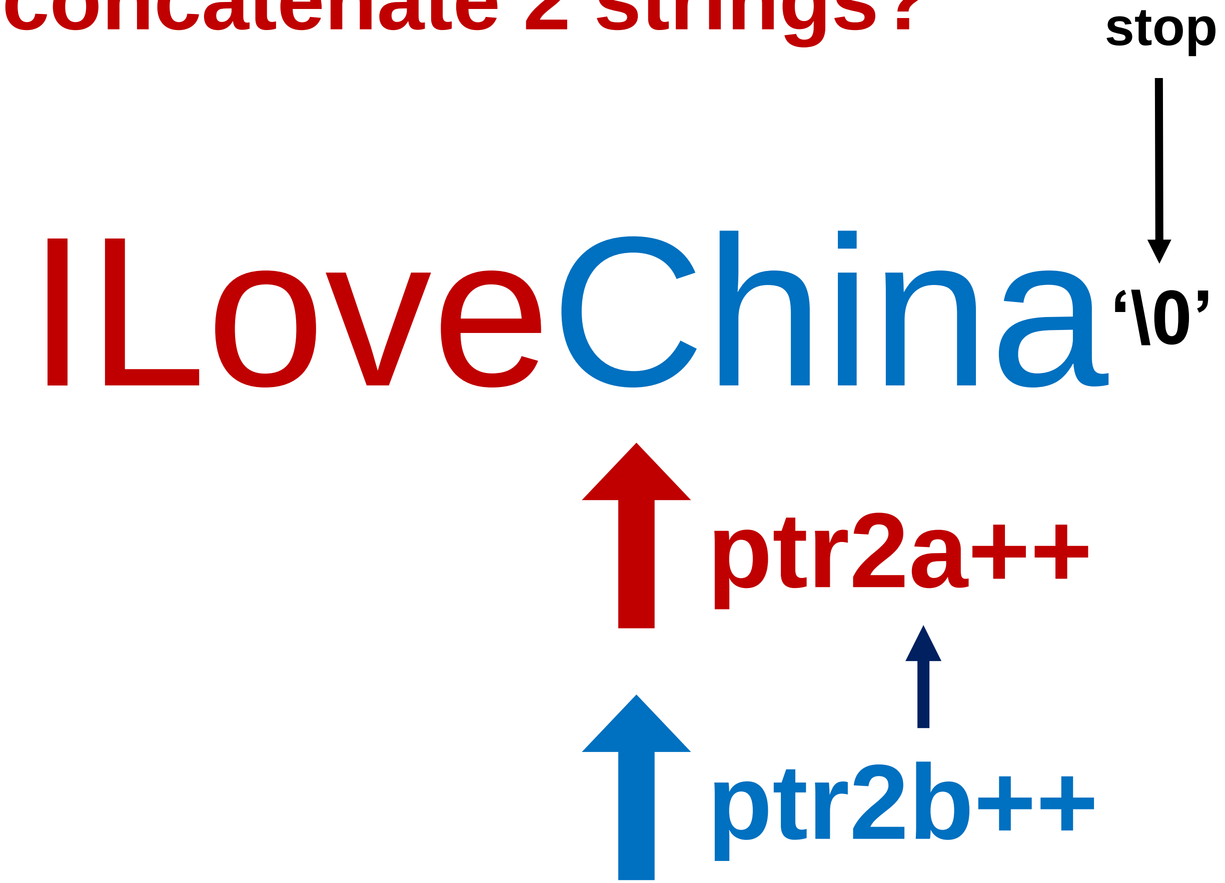
ptr2a++

ptr2b++

Use pointer for array

How to concatenate 2 strings?

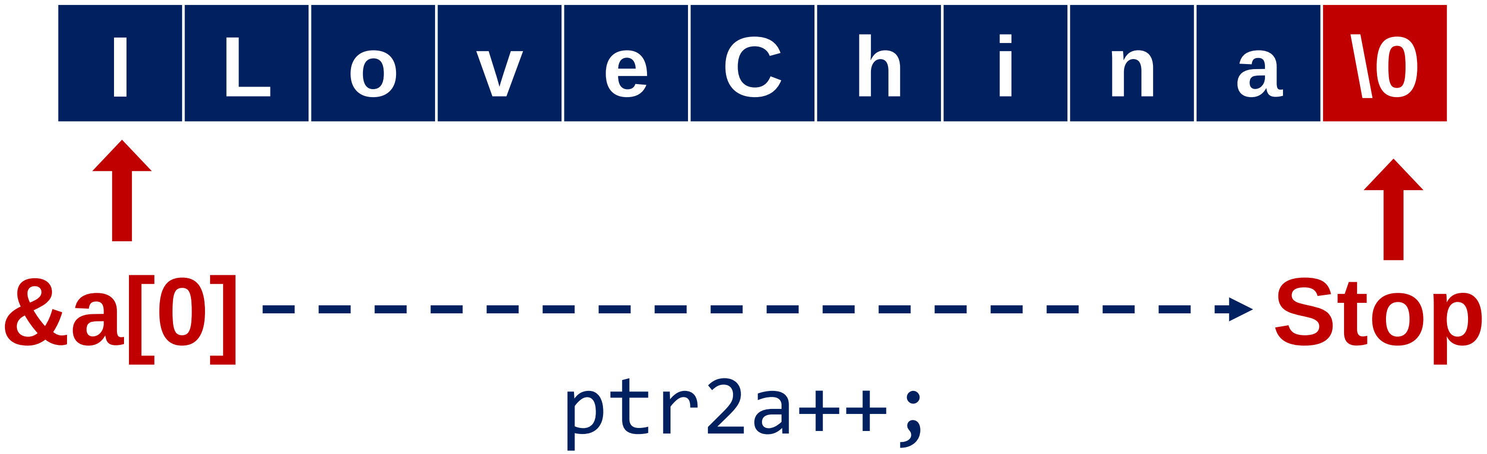
```c
#include<stdio.h>

main()
{
    char a[100] = "ILove";
    char b[] = "China";

    char* ptr2a = &a[5]; // last address + 1
    char* ptr2b = &b[0]; // first address

    while (*ptr2b != '\0')
    {
        *ptr2a = *ptr2b;
        ptr2a++;
        ptr2b++;
    }

    printf("%s\n", a);
}
```



Use pointer for array

What is the length of a string?

```
#include<stdio.h>

main()
{
    char a[100] = "ILoveChina";
    char* ptr2a = &a[0];

    int length = 0;
    while(*ptr2a != '\0’)
    {
        ptr2a++;
        length++;
    }

    printf("Length of a is %d\n", length);
}
```



I	L	o	v	e	C	h	i	n	a	\0

&a[0] ---------------→ Stop

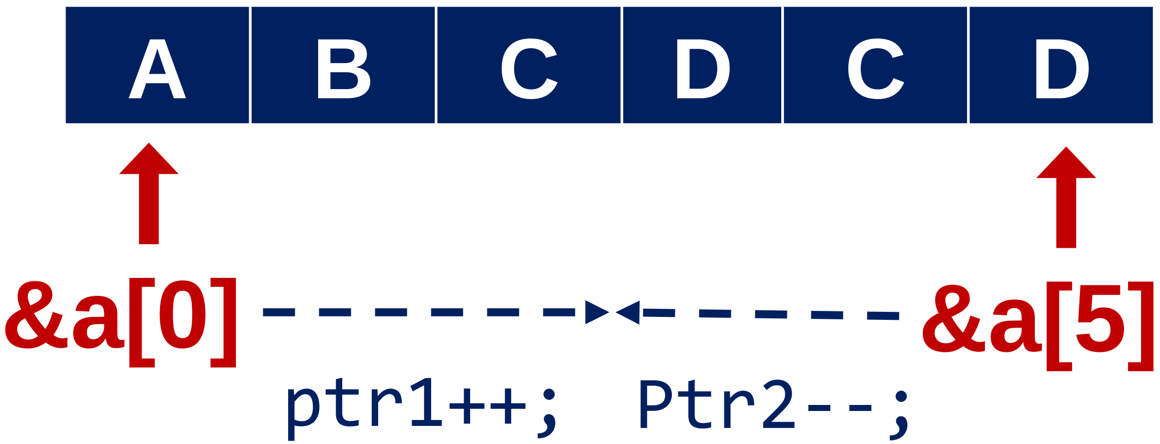
ptr2a++;

Length = 10

Use pointer for array

```c
#include<stdio.h>

main()
{
    char a[6] = "ABCDEF";
    char* ptr1 = &a[0]; //first address
    char* ptr2 = &a[5]; //last address

    int length = 0;
    while(ptr1 < ptr2)
    {
        char temp = *ptr1;
        *ptr1 = *ptr2;
        *ptr2 = temp;

        ptr1++;
        ptr2--;
    }
    printf("Inversion is %s\n", a);
}
```

How to invert a string?



A	B	C	D	C	D

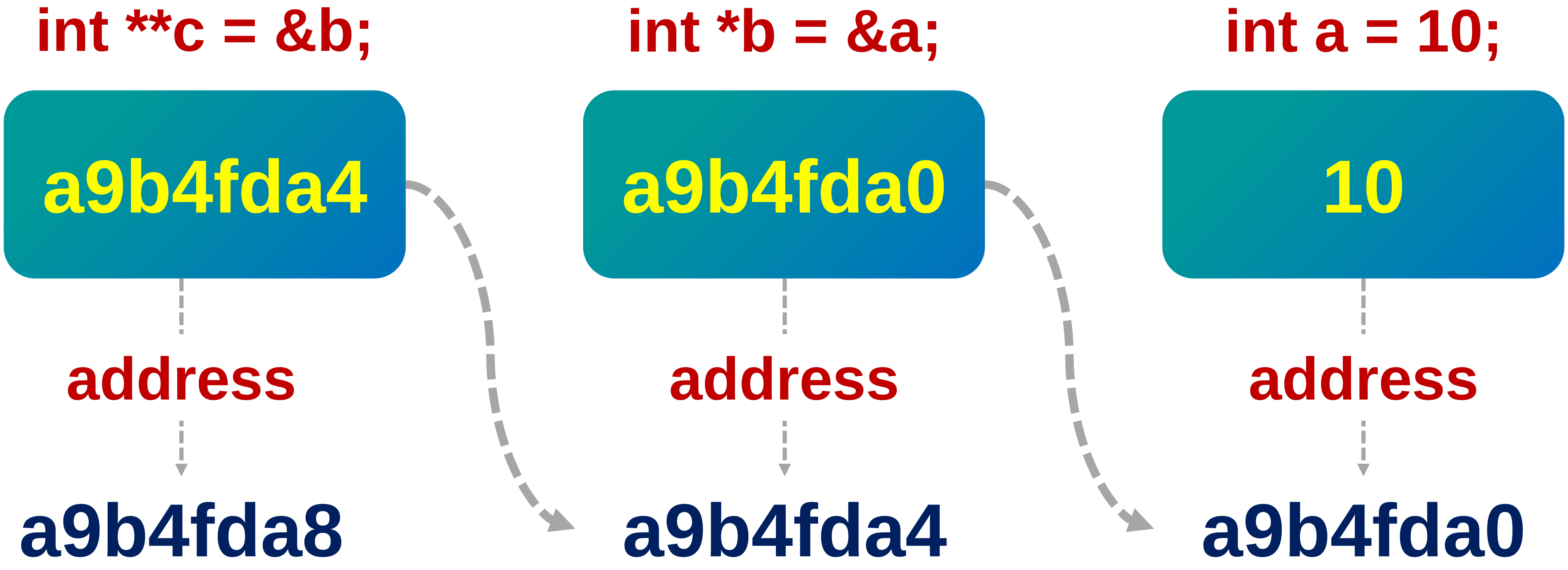
&a[0] &a[5]

ptr1++; Ptr2--;

Inversion is FEDCBA

Double pointer

Pointer to pointer



int **c = &b;

a9b4fda4

address

a9b4fda8

int *b = &a;

a9b4fda0

address

a9b4fda4

int a = 10;

10

address

a9b4fda0

Double pointer

Double pointer can represent matrix!

Single pointer

```
main()
{
    int r = 3, c = 4;
    int* ptr = malloc((r * c) * sizeof(int));

    for (int i = 0; i < r * c; i++)
        ptr[i] = i + 1;

    for (int i = 0; i < r; i++) {
        for (int j = 0; j < c; j++)
            printf("%d ", ptr[i * c + j]);
        printf("\n");
    }
}
```

1	2	3	4	5	6	7	8	9	10	11	12

Double pointer

```
main()
{
    int r = 3, c = 4;
    int** arr = (int**)malloc(r * sizeof(int*));

    for (int i = 0; i < r; i++)
        arr[i] = (int*)malloc(c * sizeof(int));

    int count = 0;
    for (int i = 0; i < r; i++)
        for (int j = 0; j < c; j++)
            arr[i][j] = ++count;
}
```

1	2	3	4
5	6	7	8
9	10	11	12

NULL pointer

Always good to assign NULL to a pointer variable if no address is assigned.

```
#include <stdio.h>
main()
{
   int *ptr = NULL;
   printf("The address of ptr is : %x\n", &ptr);
   printf("The value of ptr is : %x\n", ptr); //0
}
```

Memory management

You can use this library to manage the **memory** of C program

```
#include <stdlib.h>
```

Memory management

C provides several functions for memory allocation and management.

function	Description
calloc(int num, int size)	Allocate an array of **num** elements each with **size (in byte)**
malloc(int num)	Allocate an array of num bytes and leave them initialized
realloc(void *addr, int newsize)	Re-allocate memory at **addr**ess with **newsize**
free(void *addr)	Release a block of memory at **addr**ess

calloc() & malloc()

calloc()

contiguous/连续的
allocation

↓

allocates memory and
initializes all bits to zero

malloc()

memory
allocation

↓

allocates memory and leaves
the memory uninitialized

calloc() function

Fixed array size, fixed memory

```
char name[100];
```

```
char *name;
name = (char*)calloc(200, sizeof(char));
```

Dynamic memory at address of name (200 bytes)

malloc() function

Fixed array size, fixed memory

```
char name[100];
```

```
char *name;
name = (char*)malloc(200*sizeof(char));
```

Dynamic memory at address of name (200 bytes)

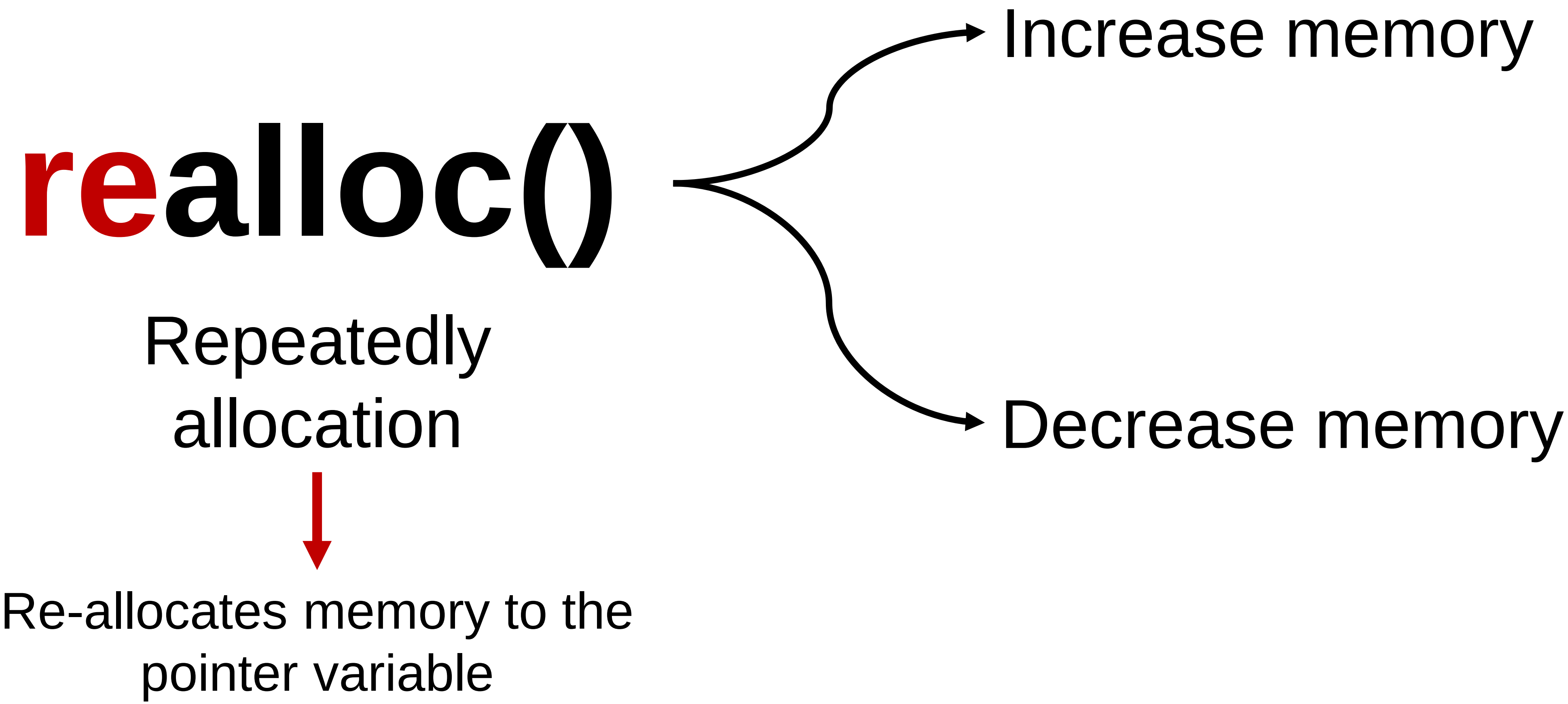
calloc() & malloc()

allocates memory and initializes all bits to zero

```
char *name;

name = (char*)calloc(200, sizeof(char));

name = (char*)malloc(200*sizeof(char));
```

allocates memory and leaves the memory uninitialized

realloc() function



realloc()

Repeatedly allocation

Re-allocates memory to the pointer variable

Increase memory

Decrease memory

realloc() function

Allocate memory at address of name (200 bytes)

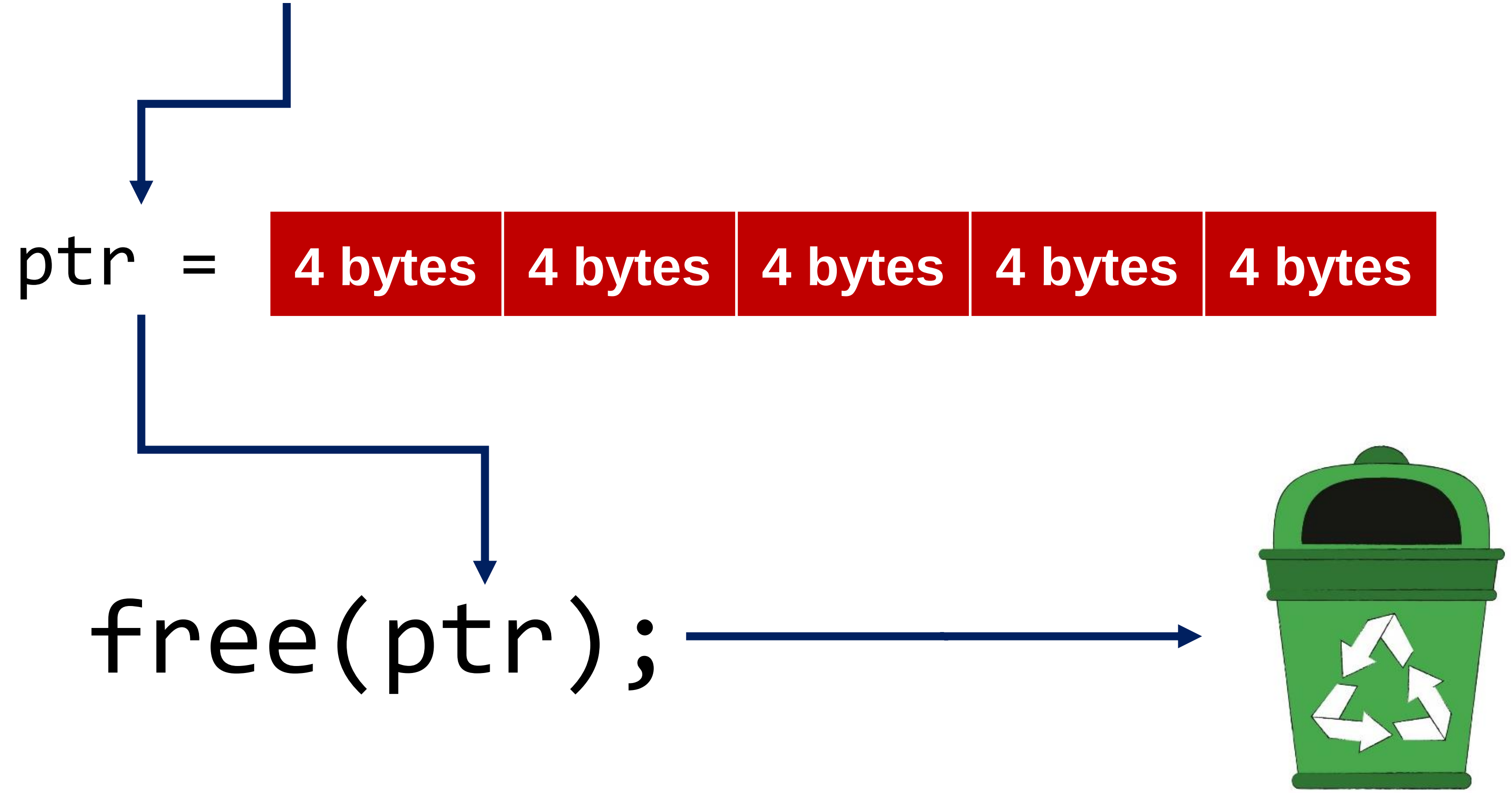
```
char *name;
name = (char*)malloc(200*sizeof(char));


name = (char*)realloc(name, 100*sizeof(char));
```

Resize the merry at address of name (100 bytes)

free() function

```
int* ptr = (int*)calloc(5, sizeof(int));
```

4 bytes



```
ptr = 
```

4 bytes	4 bytes	4 bytes	4 bytes	4 bytes

```
free(ptr);
```

Dynamic memory allocation

Use pointers as **output** of function to return results!

```
int* func(int v1, int v2)
{
    int* ptr = (int*) calloc(4, sizeof(int));
    ptr + 0 = v1 + v2;
    ptr + 1 = v1 - v2;
    ptr + 2 = v1 * v2;
    ptr + 3 = v1 / v2;
    return ptr;
}
```

```
ptr[0] = v1 + v2;
ptr[1] = v1 - v2;
ptr[2] = v1 * v2;
ptr[3] = v1 / v2;
```

Output a pointer (array)

```
main()
{
    int a = 10, b = 5;
    int *ptr = func(a, b);
    printf("sum=%d, sub=%d, mul=%d, div=%d", *ptr, *(ptr+1), *(ptr+2), 
*(ptr+3)));
}
```

Dynamic memory allocation

Static array & Dynamic array

静态 动态

```
int main(void)
{
    char str1[] = "Ilove";

    char* str1_ = (char*)malloc(sizeof(char) * 6);

    for (int i = 0; i < strlen(str1) + 1; i++)
        str1_[i] = str1[i];

return 0;
}
```

ILove

We can convert static array to dynamic array

Summary

- **Pointer** is a variable that stores the address of another variable.
- We can access the **memory address** directly using the pointer.
- By changing the pointer value, the value stored at the address will be modified, typically useful for **functions** to pass values.
- Pointer can point to arrays, using **arithmetic and logical operations** (++, --, ==, >, <) to scan the memory address.
- We can **manage the memory** using C provided functions in **stdlib.h,** e.g. calloc(), malloc(), relloc(), free().

5 questions

1. What is the difference between the variable and pointer variable?

2. Which of following is the correct statement for a pointer ( )
A. int a = 5; int *p = &a;
B. char a = ‘a’; char *p = a;
C. int *p = 5;
D. Above are correct

3. Given int a[] = {1,2,3,4,5,6}, assume int *ptr = &a[2]; which of following is true( )
A. *(ptr+2) is 3
B. *(ptr+2) is 4
C. *(ptr+2) is 5
D. *(ptr+2) is 6

5 questions

4. Assuming the function is **void f(int, int*)**, in the main function, we have

```
int a  = 2;
int *p = &a;
```

Which following function calling is correct?

A. f(a, &p)
B. f(*p, p)
C. f(p, a)
D. f(*p, a)

5. Assume A = 2, B = 10, write a function to swap the value between A and B (B = 2, A = 10) ?

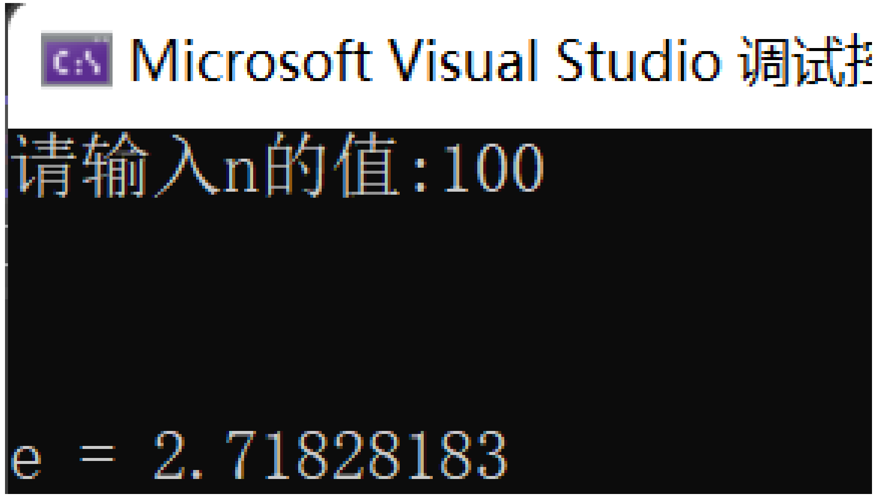
Assignment

1. The natural constant e (≈2.71) can be approximated using the series 1+1/1!+1/2!+⋯+1/n!+⋯write a function to calculate the approximation of e.

a) Use n as the arguement of the function and return the approximation of e

b) Call this function in main() and print the approximation of e

c) Test input n = 100



```
请输入n的值:100


e = 2.71828183
```

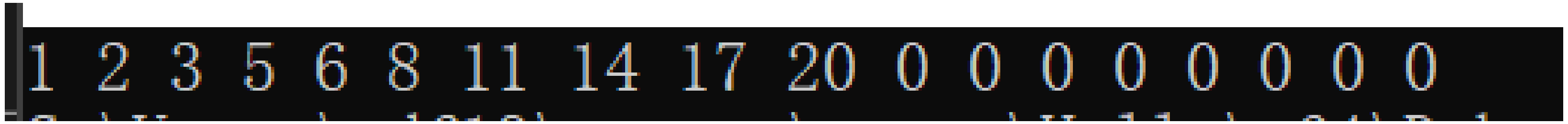
Assignment

2. There is an array where elements (integers) are in the ascending order, please delete the recurring elements in the array so that each element appears only once, and print the new length of the array after deletion. The relative order of elements should be consistent.

a) Since the length of a static array cannot be changed, you can place the remaining elements in a new array.

b) You can place the remaining elements in the front of the new array and set the vacated position to 0

c) Test input 1, 1, 2, 3, 5, 6, 6, 6, 8, 8, 11, 14, 14, 14, 14, 17, 17, 20

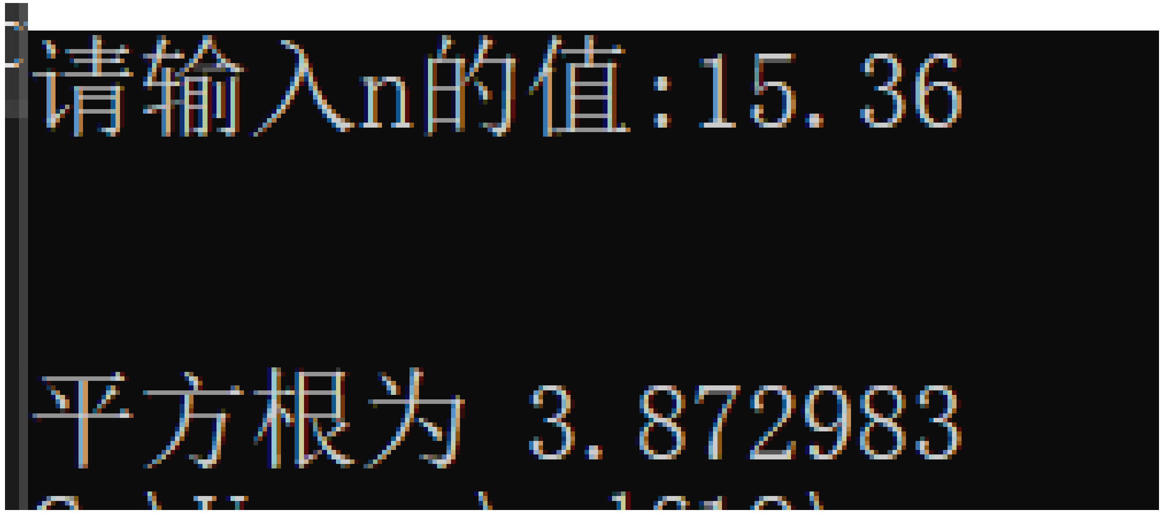


```
1  2  3  5  6  8  11  14  17  20  0  0  0  0  0  0  0  0
```

Assignment

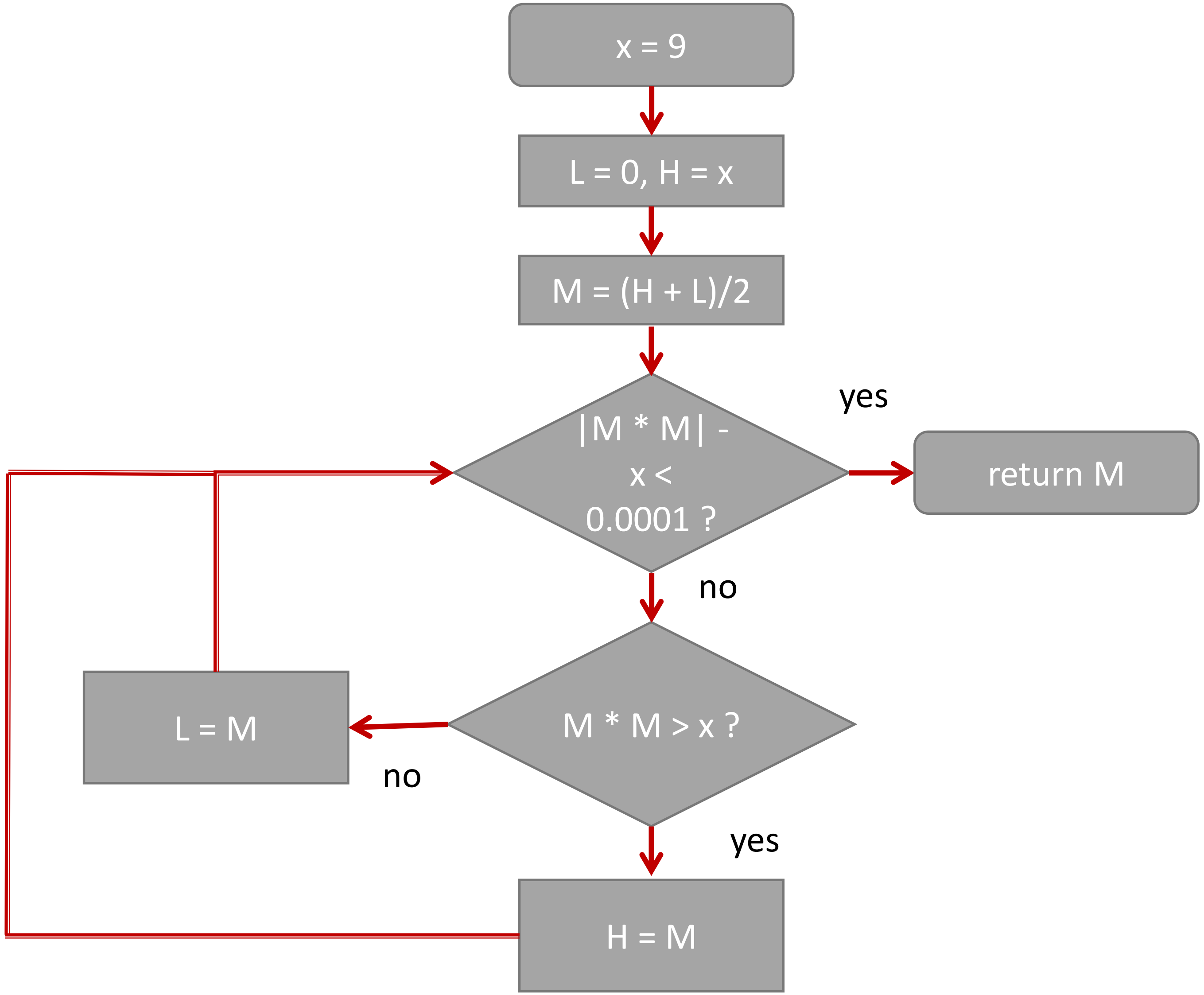
3. Write a “sqrt()” function to calculate square root by bisection and call the function in “main”

a) Use “scanf” to enter the number
b) Test input : 45.76
c) You are not allowed to use sqrt() which is defined in math.h
d) What is bisection? I will show you on the next page



```
请输入n的值:15.36


平方根为 3.872983
```

Assignment



If we want to calculate the square root of 9 , we need a lower limit L(L = 0),and an upper limit H(H = 9). We try if middle(M = (0+9)/2 = 4.5) of H and L is the answer. Because of 4.5*4.5 > 9, we know the squre root of 9 is smaller than 4.5. then we let the upper limit H = 4.5, try the middle of H and L. M = (4.5 + 0) /2 = 2.25. Because of 2.25 * 2.25 < 9, we know the squre root of 9 is bigger than 2.25. then we let L = 2.25 and try the middle of H and L again ……

Try again and again until |M * M – 9| < 0.0001. we see this M is the square root of 9