C程序设计基础

Introduction to C programming
Lecture 10: Pointer

张振国 zhangzg@sustech.edu.cn

南方科技大学/理学院/地球与空间科学系

Review on L9 Function

Declare/define/call a function

Variable scope

Recursion

Recap last lecture

- We can create our own functions in 3 steps: **function declaration, definition, calling**. Function needs to 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. Variables can have **identifiers** (auto, static, extern, register).
- **Recursion** can be implemented by calling a function itself repeatedly.

Recap last lecture

$$\begin{tikzcd}
 & \text{function1} \\
\text{main} \arrow[ur, bend left] \arrow[r] \arrow[dr, bend right] & \text{function2} \\
 & \text{functionN}
\end{tikzcd}$$

$$\begin{tikzcd}
 & \text{function1} \arrow[d, bend left] \\
\text{main} \arrow[ur, bend left] & \text{function2} \arrow[d, bend left] \\
 & \text{functionN}
\end{tikzcd}$$

$$\begin{tikzcd}
 & \text{function1} \arrow[d, bend left] \\
\text{main} \arrow[ur, bend left] & \text{function2} \arrow[u]
\end{tikzcd}$$

$$\begin{tikzcd}
 & \text{function1} \arrow[loop right] \\
\text{main} \arrow[ur, bend left] &
\end{tikzcd}$$

Step 1: declare a function

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

Return type

float, double,
char, void

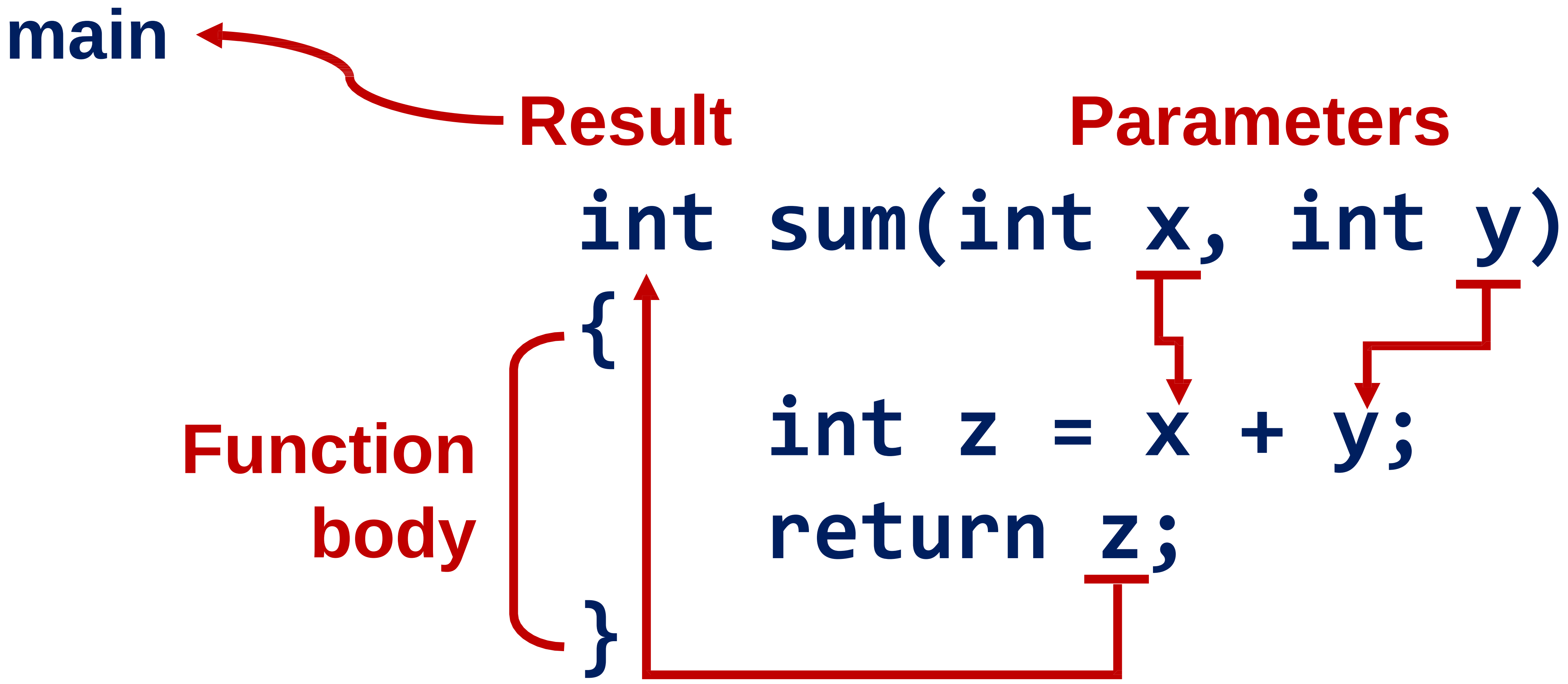
Function name

Any name you like except
C reserved keywords

Input parameters （形式参数）

Variables, arrays

Step 2: define a function



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

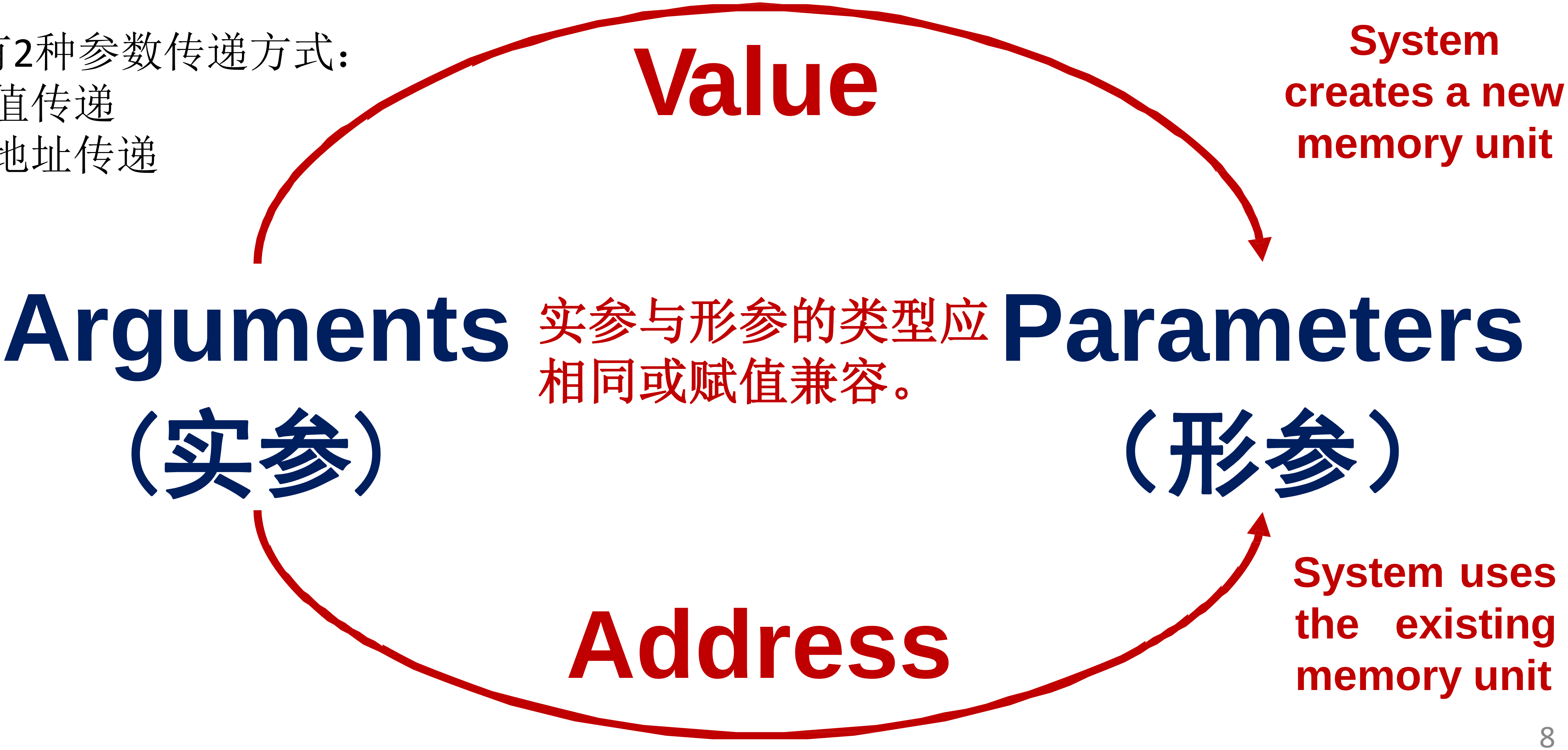
Step 3: call a function

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

Arguments
（实际参数）

Arguments and parameters

C有2种参数传递方式：
- ✓ 值传递
- ✓ 地址传递


Value
System creates a new memory unit
Arguments （实参）
实参与形参的类型应相同或赋值兼容。
Parameters （形参）
Address
System uses the existing memory unit

Content

1. Declare/define/call a function
2. Variable scope
3. Recursion

Variable scope

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

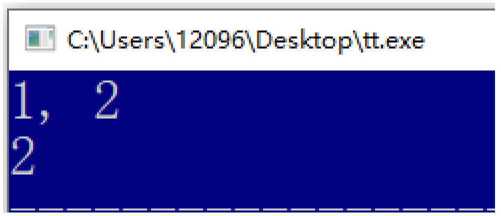
Global variable（全局变量）

Local variable（局部变量）

```
int b; //outside main

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

Variable scope

Global variable can be accessed everywhere



C:\Users\12096\Desktop\tt.exe

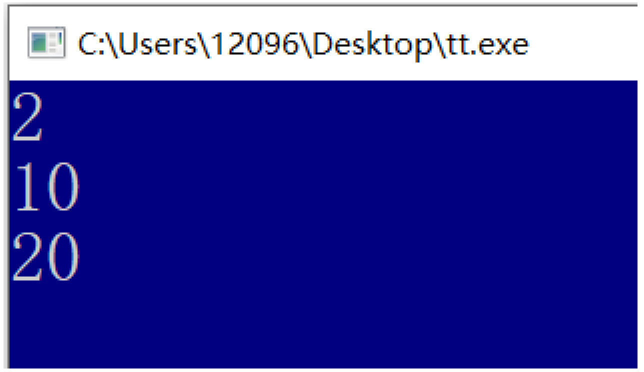
```
1, 2
2
```

Local variable can only be accessed inside the function where it was defined

```
int b = 2; // global
void func()
{
    printf(" %d", b);
}
int main()
{
   int a = 1; // local
   printf("%d, %d", a, b);
   func();return 0;
}
```

Variable scope

Global variable can be changed everywhere and keep the changes


C:\Users\12096\Desktop\tt.exe

```
2
10
20
```

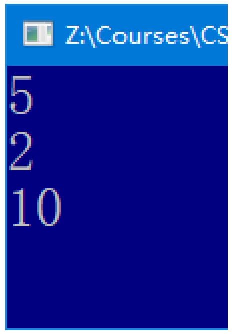
```
int b = 2; // global
void func()
{                                    ②
    printf("%d\n", b); //print 10
    b = 20;
}
int main()
{                                 ①
    printf("%d\n", b); //print 2
    b = 10;
    func();                       ③
    printf("%d\n", b);  //print 20
    return 0;
}
```

Variable scope

- **Global and local variables can share the same name**

- **Local variable has the priority!!!**

```
#include <stdio.h>
int b = 2; // global
void func() {
    printf("%d", b); //print 2  ②
    b = 20;
}
int main() {
        int b = 5; // local       ①
        printf("%d", b); //print 5
        b = 10;
        func();
        printf("%d", b); //print 10 ③
        return 0;
}
```



```
5
2
10
```

Variable scope

```
float PI = 3.14; // global
 float func(float a)
{
      return a * PI;
}
main()
{
   float a = 5; // local
   float b = func(a);
   printf("%f", b);
}
```

✓ **Do not use global variables unless**
- It is a **constant** that can be used everywhere (consensus一致性)
- 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;
```

$\downarrow$

自动 `auto int a = 5;`

静态 `static int a =  5;`

常用 $\left\{ \begin{array}{l} \\ \\ \end{array} \right.$

外部 `extern int a =  5;`

寄存器 `register int a  = 5;`

Auto variable

Memory for variable is automatically created when the function is invoked and destroyed when a block exits. **By default, local variables have automatic storage duration.**

```
#include <stdio.h>
main(){
   auto int i = 10;
   float j = 2.8;
}

void myFunction(){
    int a;
    auto int b;
}
```

Both i and j are auto variables

Both a and b are auto variables

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
}
```

Static variable

（1） 静态局部变量属于静态存储类别，在静态存储区内分配存储单元。**在程序整个运行期间都不释放**。而自动变量（即动态局部变量）属于动态存储类别，占动态存储区空间而不占静态存储区空间，**函数调用结束后即释放**。

（2）对静态局部变量是在编译时赋初值的，即**只赋初值一次**，在程序运行时它已有初值。以后每次调用函数时不再重新赋初值而只是保留上次函数调用结束时的值。而对自动变量赋初值不是在编译时进行，而是在函数调用时进行，每调用一次重新给一次初值，相当于执行一次赋值语句。

Static variable

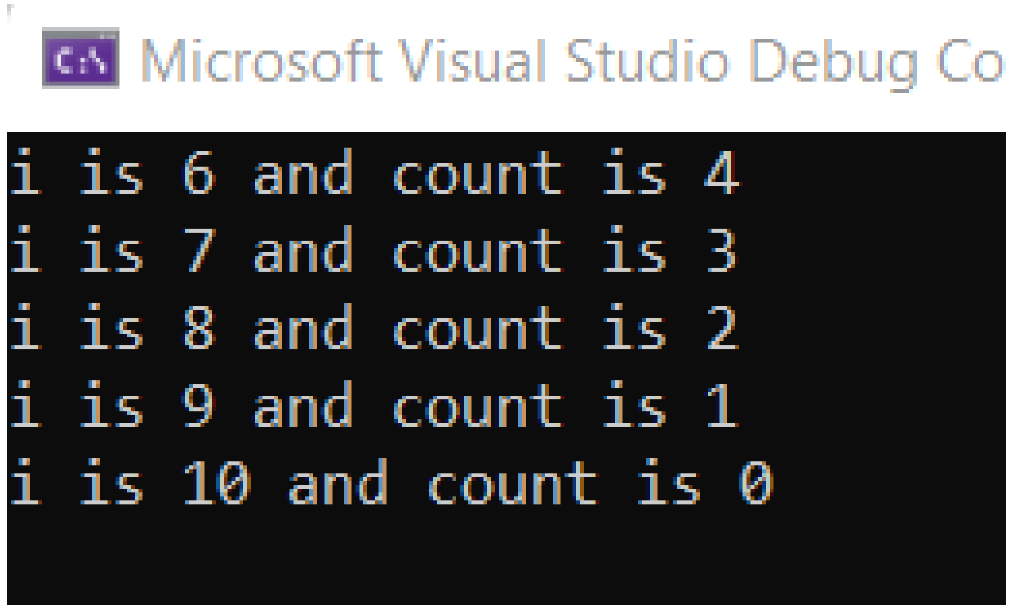
（3）如在定义局部变量时不赋初值的话，则对静态局部变量来说，编译时自动赋初值 0 （对数值型变量）或’\0’ （对字符变量）。而对自动变量来说，如果不赋初值则它的值是一个不确定的值。

（4）虽然静态局部变量在函数调用结束后仍然存在，但其他函数不能引用它。

Case study: static variable

```
#include <stdio.h>

void func(void);

static int count = 5;
int main(void) {

    while (count--) {
        func();
    }
        return 0;
}

void func(void){
    static int i = 5;
    i++;
    printf("i is %d and count is %d\n", i, count);
}
```

Case: create two static counters (increment and decrement).

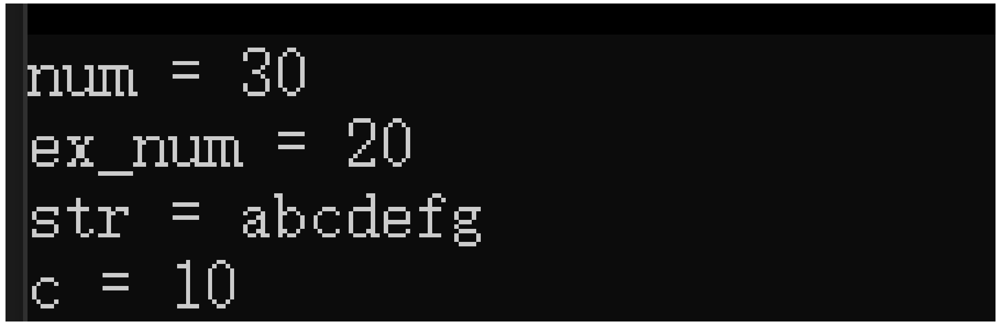

Microsoft Visual Studio Debug Co

```
i is 6 and count is 4
i is 7 and count is 3
i is 8 and count is 2
i is 9 and count is 1
i is 10 and count is 0
```

Extern variable

Extern can only be used to define global variables. An extern variable can be assessed across different C files.

```
/***> File Name: extern_test.c ***/
#include <stdio.h>
int ex_num = 20;    // 定义外部变量
int num = 30;
char str[81] = "abcdefg";
```



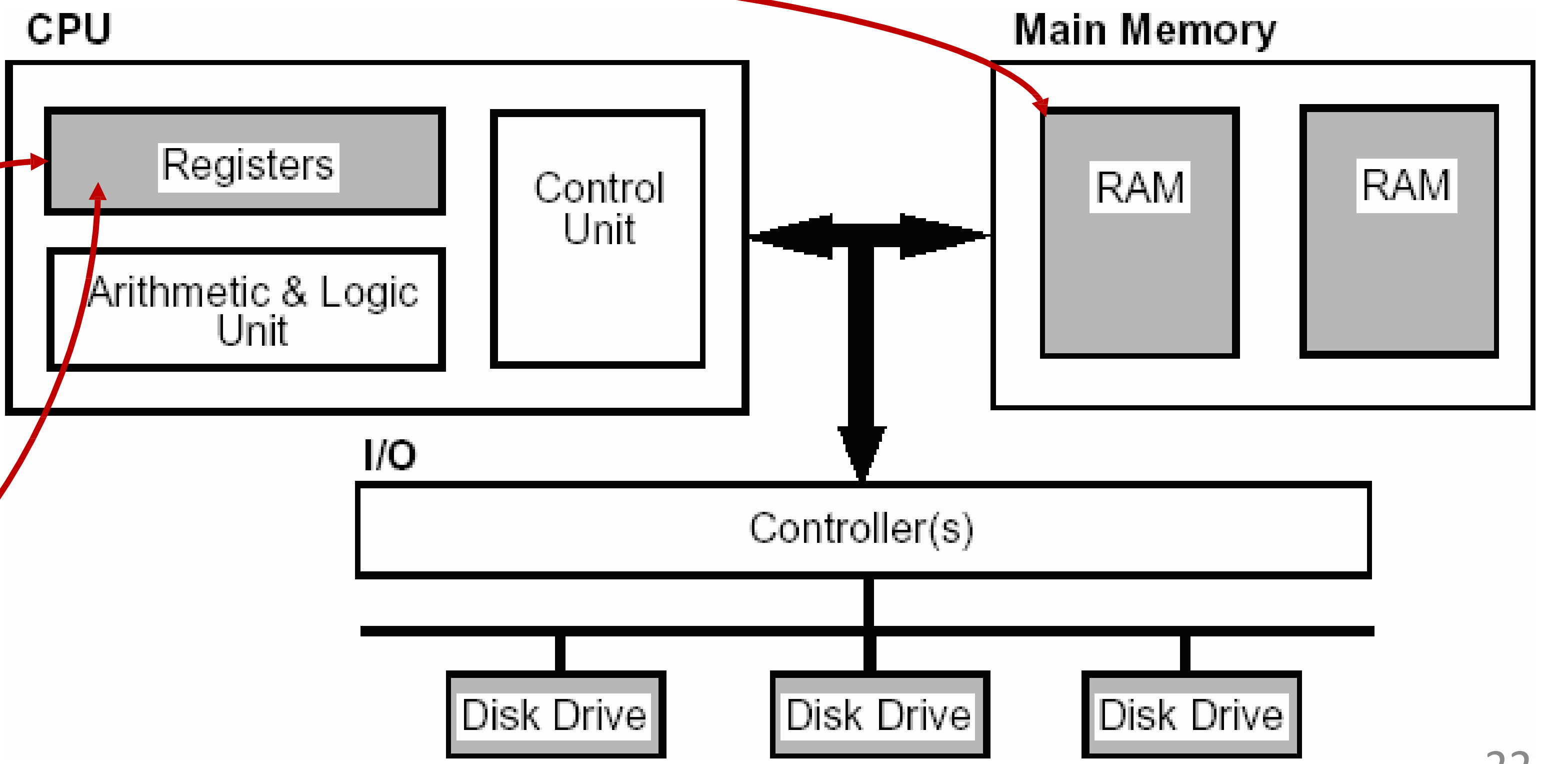
```
num = 30
ex_num = 20
str = abcdefg
c = 10
```

```
/***> File Name: main_test.c***/
#include <stdio.h>
//将已定义的外部变量的作用域扩展到本文件
extern int num;
extern int ex_num;
extern char str[81];
int c=10;
main(){
    printf("num = %d\n", num);
    printf("ex_num = %d\n", ex_num);
    printf("str = %s\n", str);
    printf("c = %d\n", c);
}
```

Register variable

Register is used to define local variables that are stored in a register (faster) instead of RAM. **By default, local variables have register storage.**

```
#include <stdio.h>

int b = 1;

myFunction(){
    int c = 1;
    register int d = 2;
}

main(){
    int a = 10;
}
```

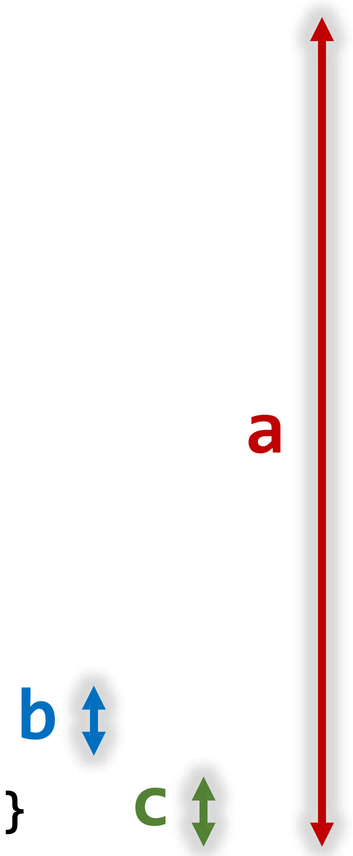


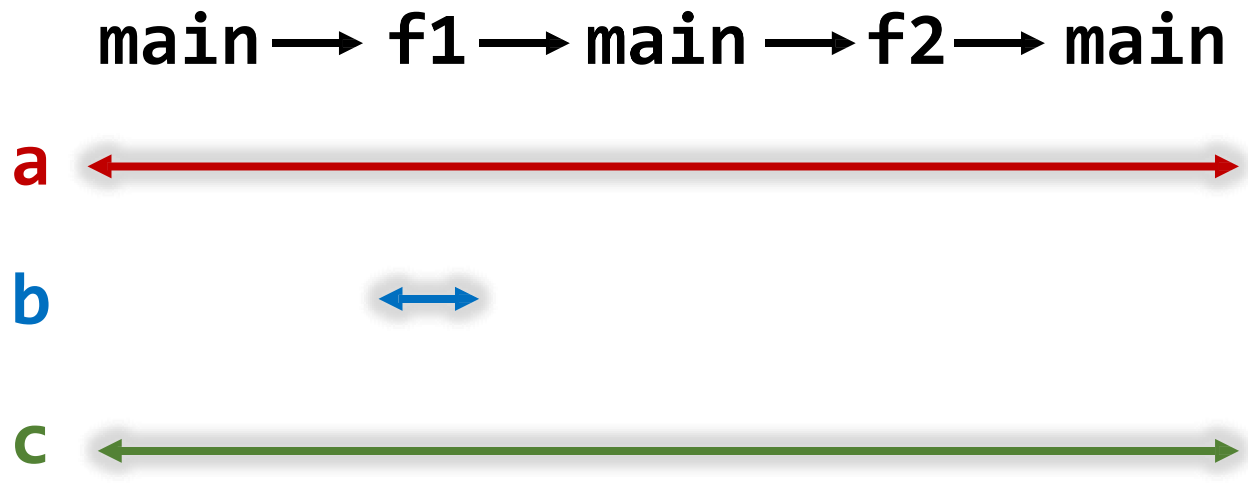
Variable scope

Scope in space

Scope in time

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





Summary

（1）从作用域角度分，有局部变量和全局变量。它们采用的存储类别如下：

- 局部变量包括：

  自动变量、 静态局部变量、寄存器变量。

  形式参数可以定义为自动变量或寄存器变量。

- 全局变量包括：

  静态外部变量、外部变量。

Summary

（2）从变量存在的时间来区分，有动态存储和静态存储两种类型。静态存储是程序整个运行时间都存在，而动态存储则是在调用函数时临时分配单元。

- 动态存储 ： 自动变量、寄存器变量、形式参数。
- 静态存储：静态局部变量、静态外部变量 、外部变量。

Summary

（3）从变量值存放的位置来区分,可分为：

内存中静态存储区：静态局部变量、静态外部变量、外部变量。

内存中动态存储区：自动变量和形式参数。

CPU中的寄存器：寄存器变量。

（4） static对局部变量和全局变量的作用不同。对局部变量来说,它使变量由动态存储方式改变为静态存储方式。而对全局变量来说,它使变量局部化，但仍为静态存储方式。从作用域角度看,凡有static声明的，其作用域都是局限的，或者是局限于本函数内，或者局限于本文件内。

Content

1. Declare/define/call a function
2. Variable scope
3. **Recursion**

Recursion in life

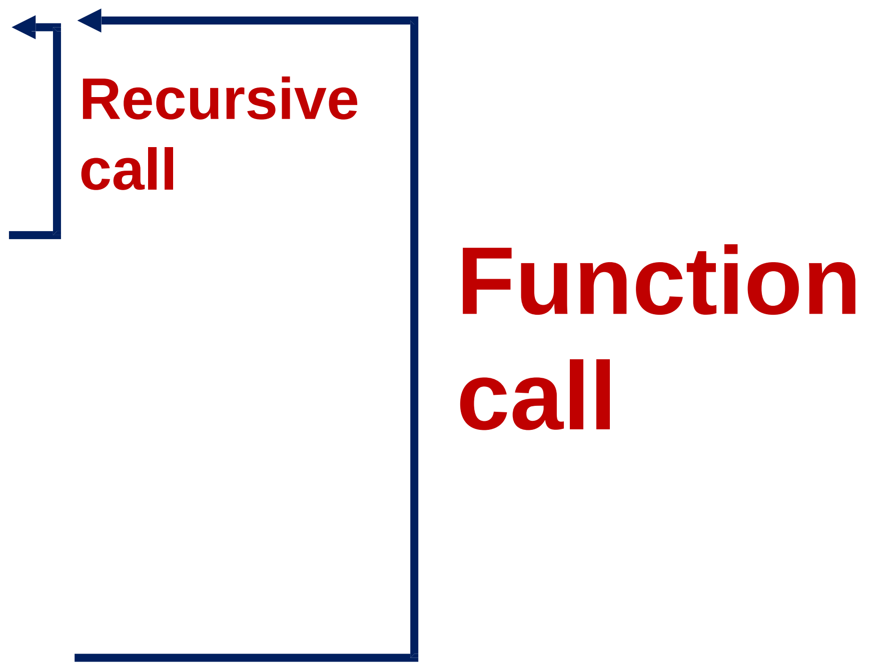
Recursion in life

Recursion in life

Recursion

Recursion is to repeat the same procedure again and again

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



Recursion

Recursively subtract

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

100

99, 98, 97,…,0,-1,…

Recursively add

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

0

1, 2, 3, …,100, 101,…

Recursion

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

Which one can leave?

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

Recursion

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

Which one is better?

```
int main(void)
{
    int n = 100
    for(; n > 0; n--)
    {
    }
}
```

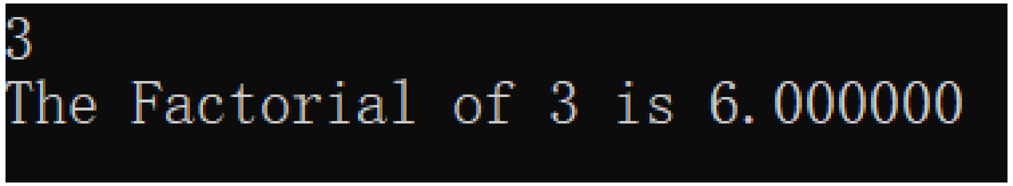
Case study: factorial calculator

```c
#include <stdio.h>

double factorial(int i)
{
    if (i <= 1)
    {
        return 1;
    }
    return i * factorial(i - 1);
}

int main(void)
{
    int input;
    scanf("%d", &input);
    printf("The Factorial of %d is %f\n", input,
factorial(input);
}
```

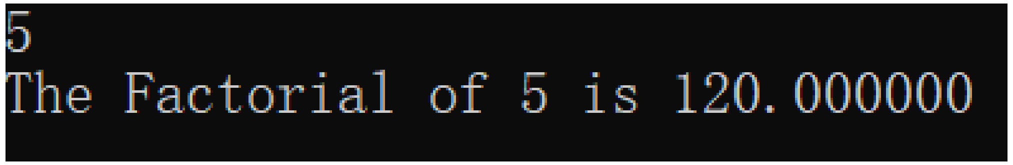
Case: use recursion to design a factorial calculator.

$$n! = 1 \times 2 \times 3 \times \cdots \times (n-1) \times n$$

$$n! = n \times (n-1)!$$



```
3
The Factorial of 3 is 6.000000
```



```
5
The Factorial of 5 is 120.000000
```

Case study: Fibonacci series

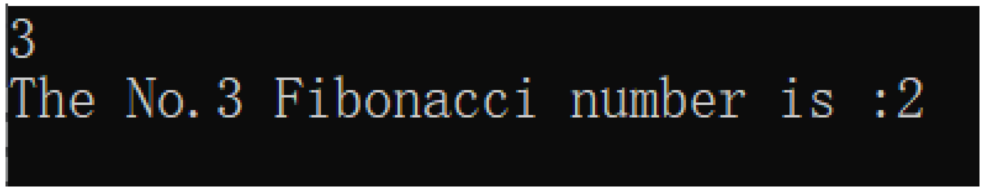
```c
#include <stdio.h>

int fib(int input) {
	if (input <= 2) {
		return 1;//first two numbers are 1
	}
	return fib(input - 1) + fib(input - 2);
}

int main(void) {
	int input = 0;
	scanf("%d", &input);
	fib(input);
	printf("The No.%d Fibonacci number is :%d",
input, fib(input));
}
```

Case: use recursion to implement a Fibonacci series.

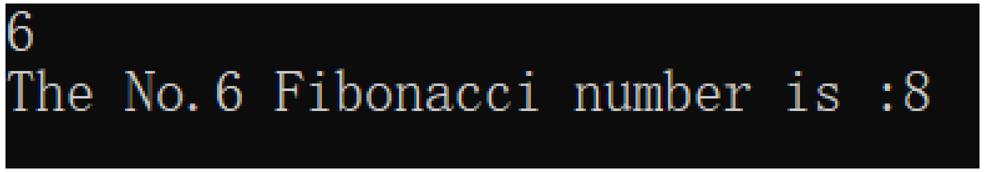
F(1)=1, F(2)=1,

F(n)=F(n - 1)+F(n - 2) （n ≥3， n ∈ N*）

1、1、2、3、5、8、13、21、34、…



```
3
The No.3 Fibonacci number is :2
```



```
6
The No.6 Fibonacci number is :8
```

Case study: Hanoi tower汉诺塔

一个源于印度的古老传说：大梵天创造世界的时候做了三根石柱子，在一根柱子按照大小顺序摞着**64片黄金圆盘**。大梵天命令婆罗门把圆盘从下面开始按大小顺序重新摆放在另一根柱子上。并且规定，在小圆盘上不能放大圆盘，在三根柱子之间一次只能移动一个圆盘。

大梵天说：**当移动完所有黄金圆盘，将海枯石烂，天荒地老。**

Case study: Hanoi tower

3 disks:

4 disks:

Hanoi tower rules:

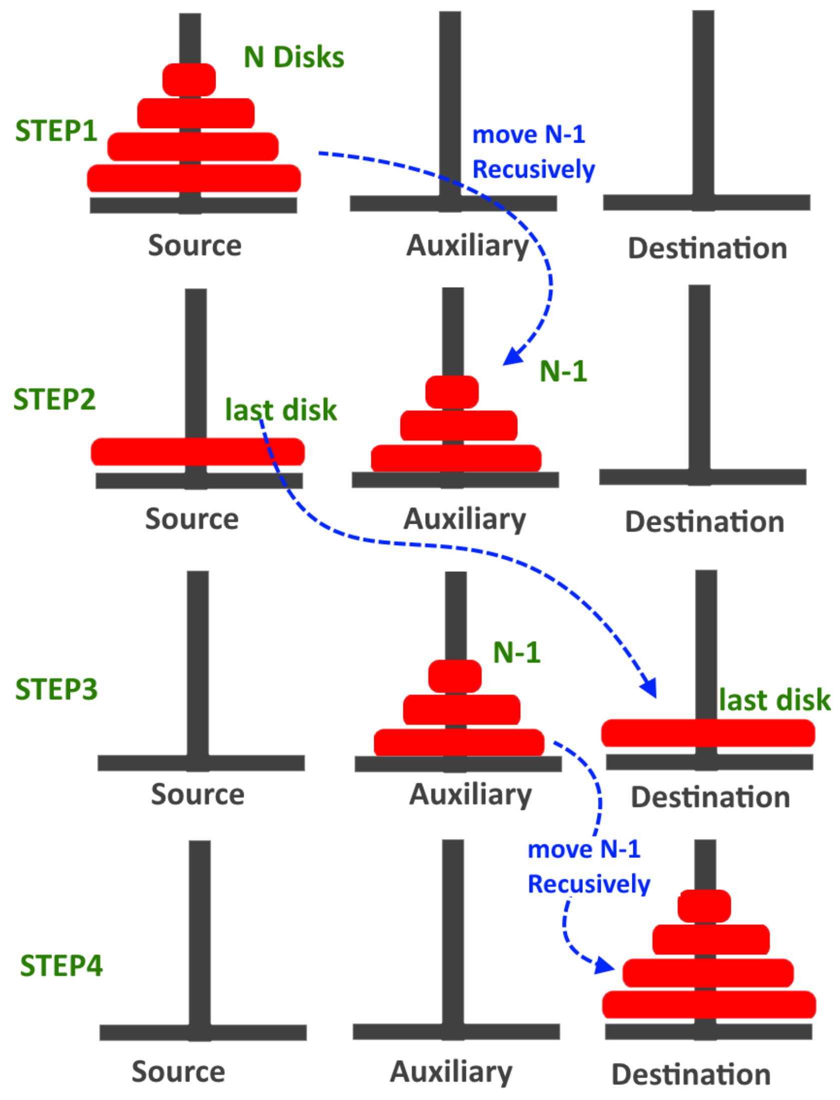
1. Only one disk can be moved at a time.
2. Each move consists of taking the upper disk from one of the stacks and placing it on top of another stack or on an empty rod.
3. No disk can be placed on top of a disk that is smaller than it.

Assume n disks, move time is f(n), f(1)=1,f(2)=3,f(3)=7 f(k+1)=2*f(k)+1, total moves is f(n)=2^n-1, n=64 means 2^{64} - 1, 1 sec 1 disk, **5845.42亿年** 以上，而地球存在至今不过45亿年!!!

😵 😵 😵

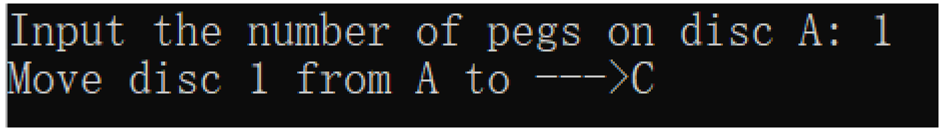
Case study: Hanoi tower

```c
#include<stdio.h>
void move(char A, char C, int n){
    printf("Move disc %d from %c to --->%c\n", n, A, C);
}
void HanoiTower(char A, char B, char C, int n){
    if (n == 1){
        move(A, C, n);
    }
    else{
        HanoiTower(A, C, B, n - 1); //Move n-1 discs
    from peg A to peg B using extra peg C
        move(A, C, n); //Move the No. n peg from A to C
        HanoiTower(B, A, C, n - 1); //Move n-1 discs
        from peg B to peg C using extra peg A
    }
}
main(){
    int n = 0;
    printf("Input the number of pegs on disc A: ");
    scanf("%d", &n);
    HanoiTower('A', 'B', 'C', n);
}
```

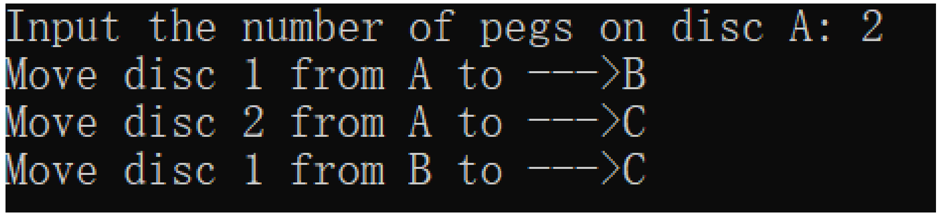


Case study: Hanoi tower

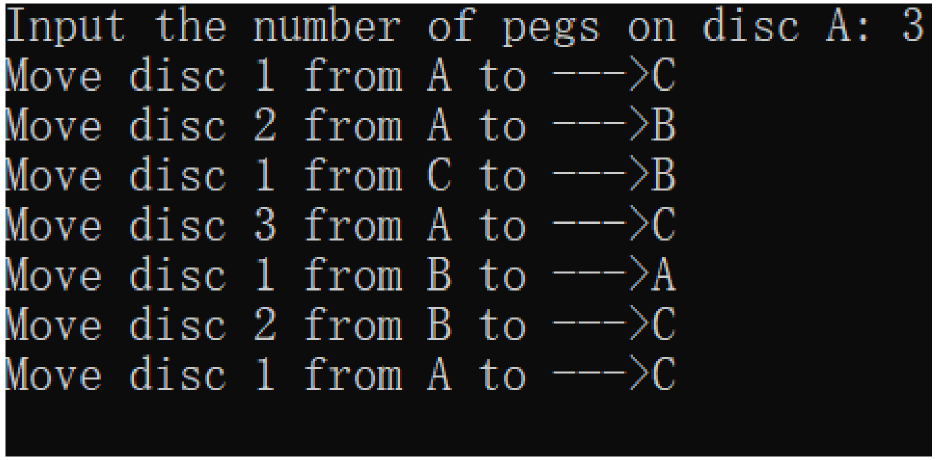
```c
#include<stdio.h>
void move(char A, char C, int n){
    printf("Move disc %d from %c to --->%c\n", n, A, C);
}
void HanoiTower(char A, char B, char C, int n){
    if (n == 1){
        move(A, C, n);
    }
    else{
        HanoiTower(A, C, B, n - 1); //Move n-1 discs
    from peg A to peg B using extra peg C
        move(A, C, n); //Move the No. n peg from A to C
        HanoiTower(B, A, C, n - 1); //Move n-1 discs
        from peg B to peg C using extra peg A
    }
}
main(){
    int n = 0;
    printf("Input the number of pegs on disc A: ");
    scanf("%d", &n);
    HanoiTower('A', 'B', 'C', n);
}
```



```
Input the number of pegs on disc A: 1
Move disc 1 from A to --->C
```



```
Input the number of pegs on disc A: 2
Move disc 1 from A to --->B
Move disc 2 from A to --->C
Move disc 1 from B to --->C
```



```
Input the number of pegs on disc A: 3
Move disc 1 from A to --->C
Move disc 2 from A to --->B
Move disc 1 from C to --->B
Move disc 3 from A to --->C
Move disc 1 from B to --->A
Move disc 2 from B to --->C
Move disc 1 from A to --->C
```

Objective of this lecture

You know how to use pointer!

Content

1. Memory address
2. Pointer
3. Pointer and Array
4. Memory management(advanced uses)
5. Pointer and function

Content

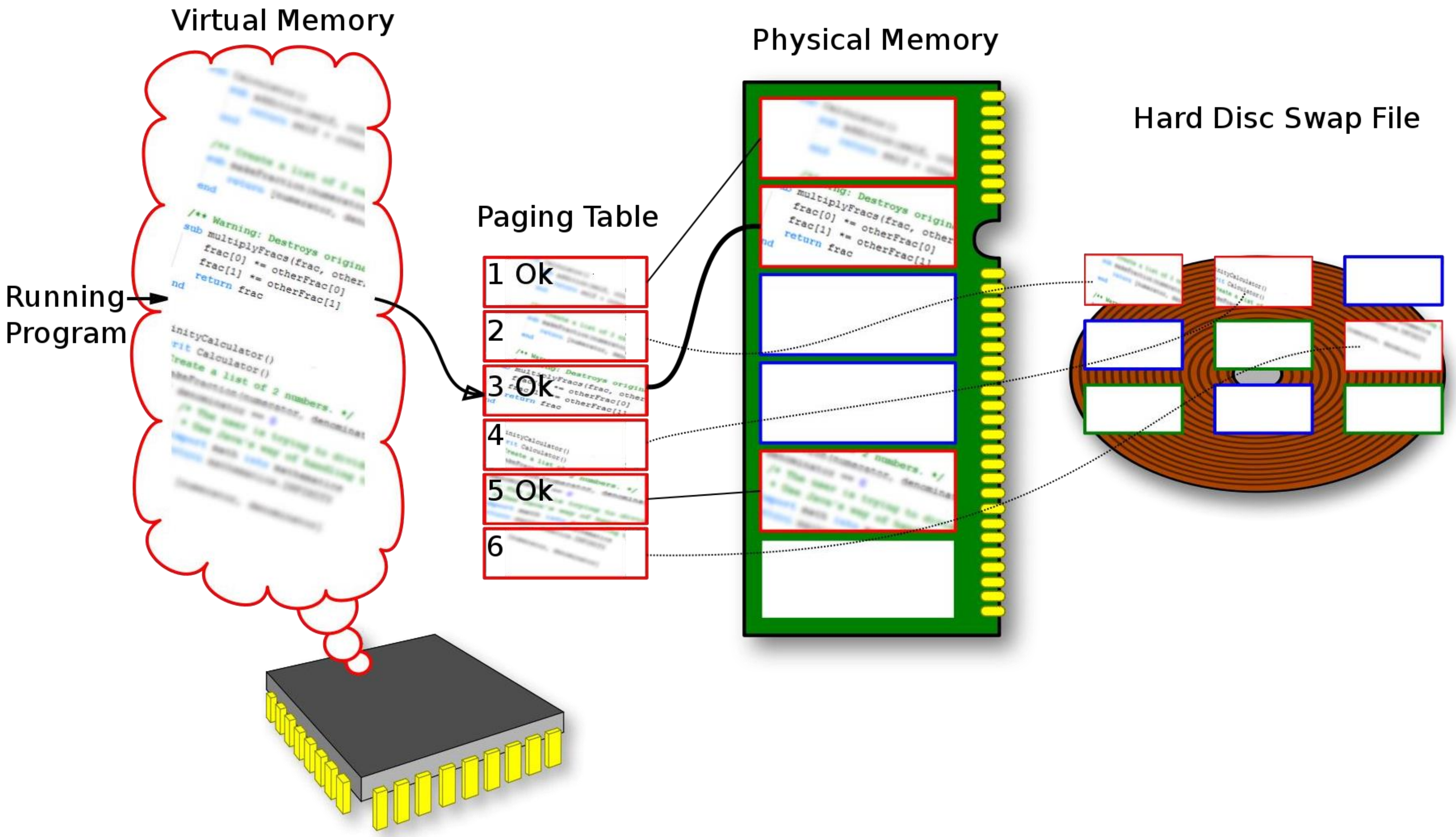
1. Memory address
2. Pointer
3. Pointer and Array
4. Memory management
5. Pointer and function

Memory address

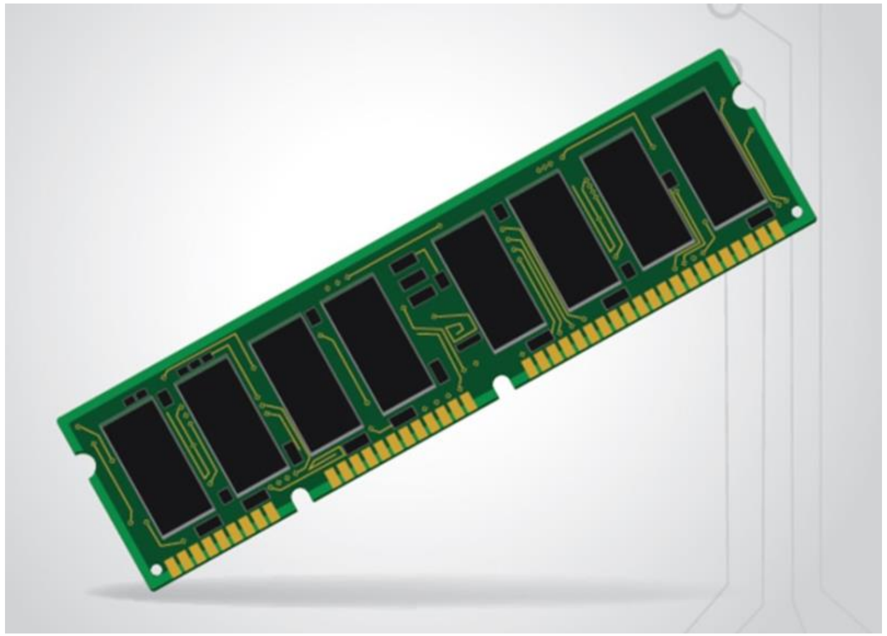
How can we find a person?

Address

Memory address



Memory address

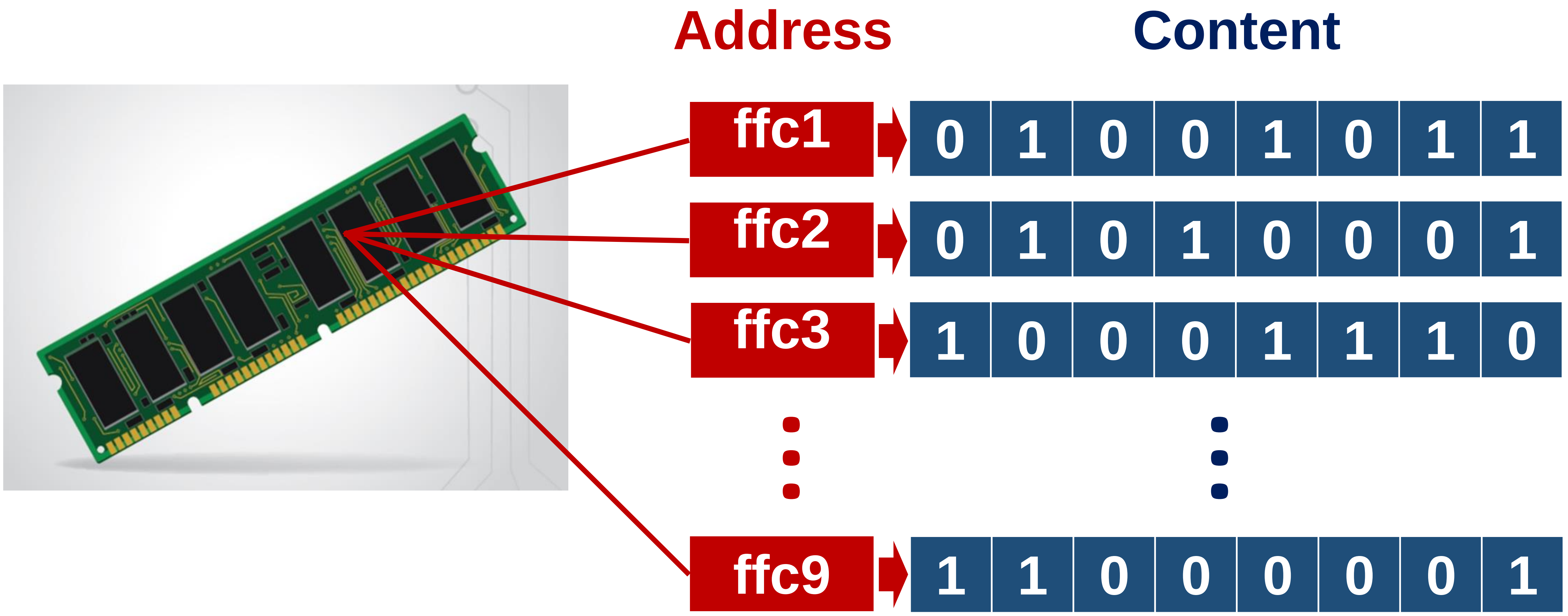


The memory address is the location of where the variable is stored on a PC.

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.

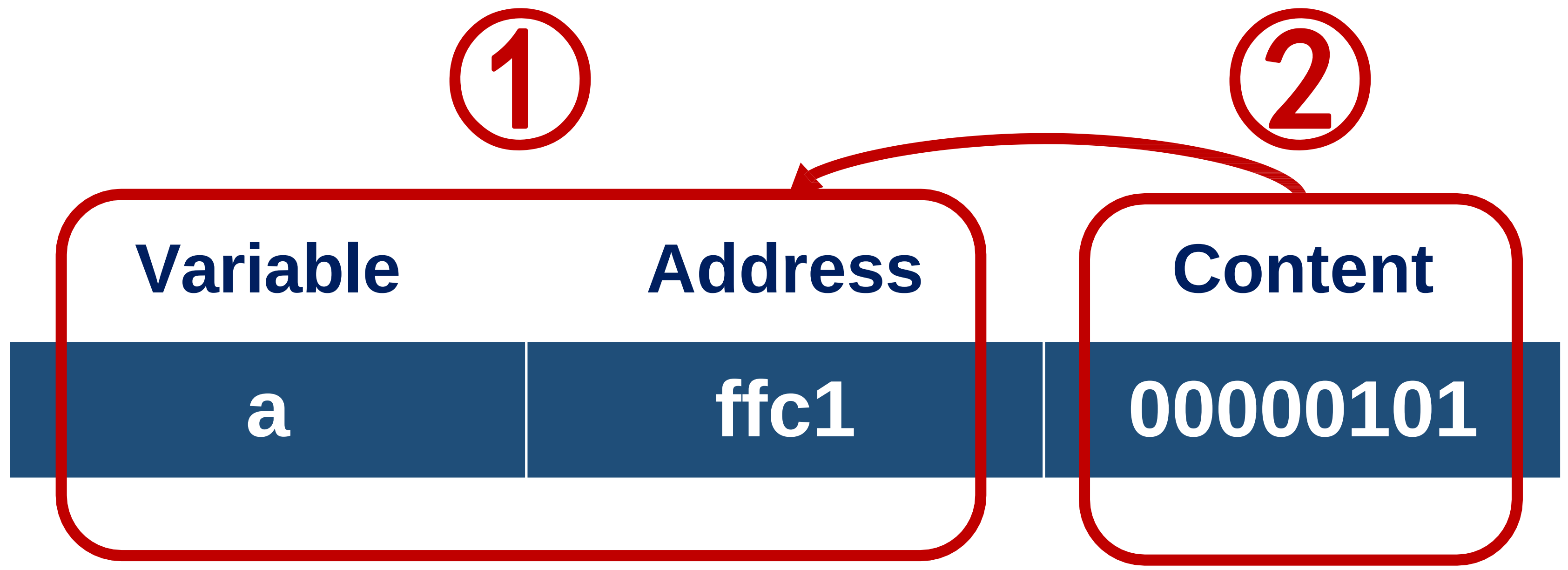
Memory address



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
```



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

You can find the content by indexing the variable name or its address!

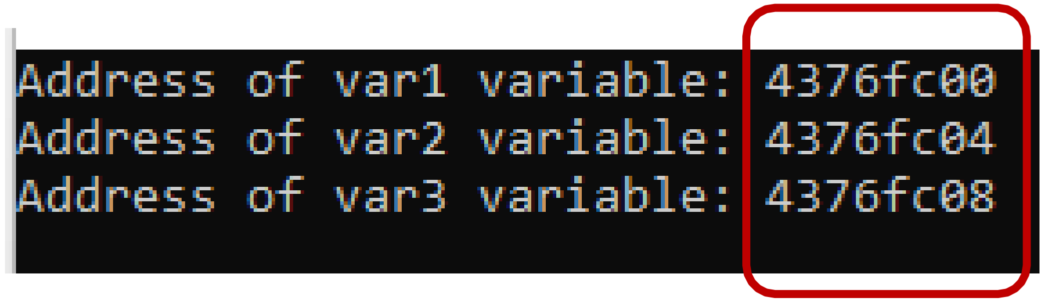
How to check variable address

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

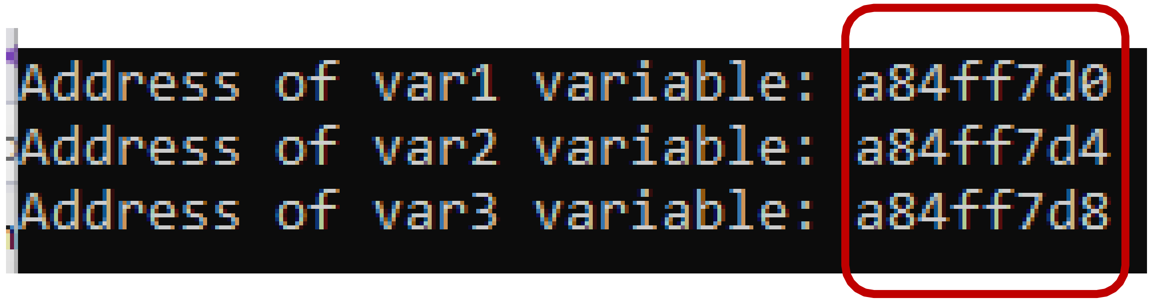
```
#include <stdio.h>

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

How to check variable address

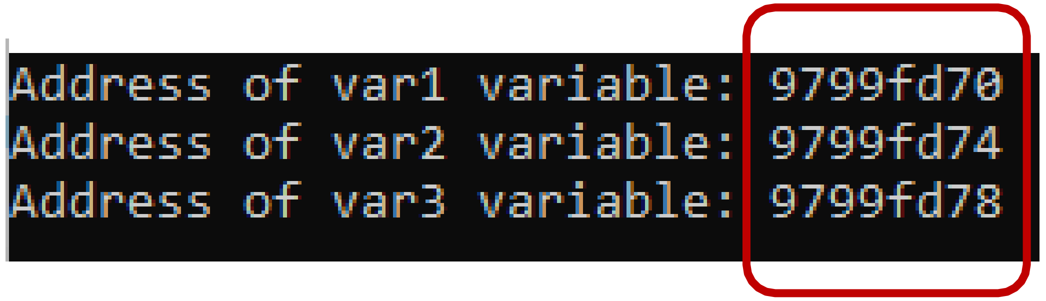
Run multiple times, every time the address is different, but it has orders!



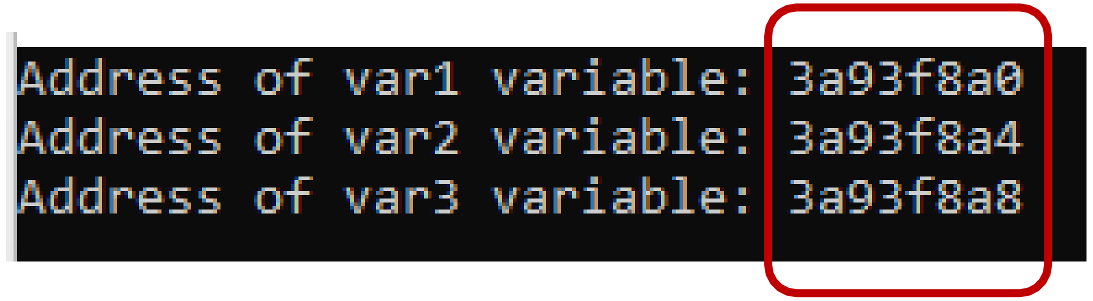
```
Address of var1 variable: 4376fc00
Address of var2 variable: 4376fc04
Address of var3 variable: 4376fc08
```



```
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

$$\downarrow$$

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

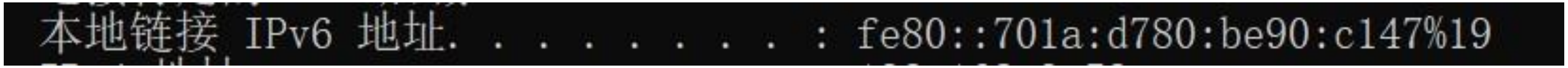
0 1 2 3 4 5 6 7 8 9 A B C D E F 10

$$\uparrow \quad \uparrow$$

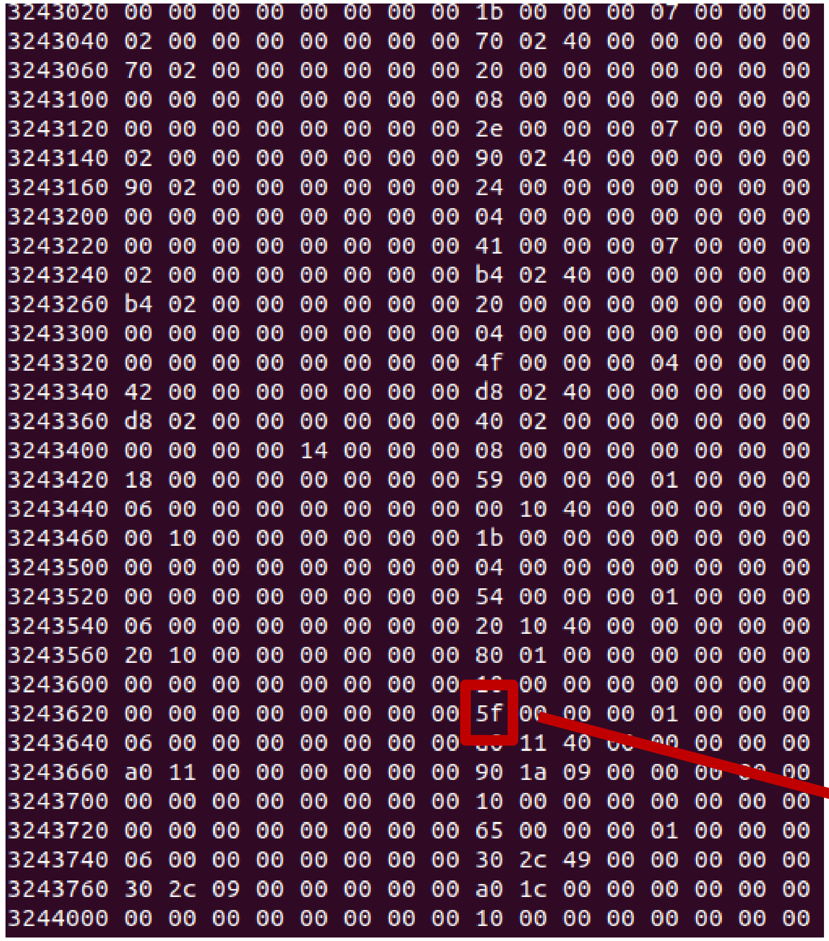
Hexadecimal number system

二位数

Hexadecimal is everywhere

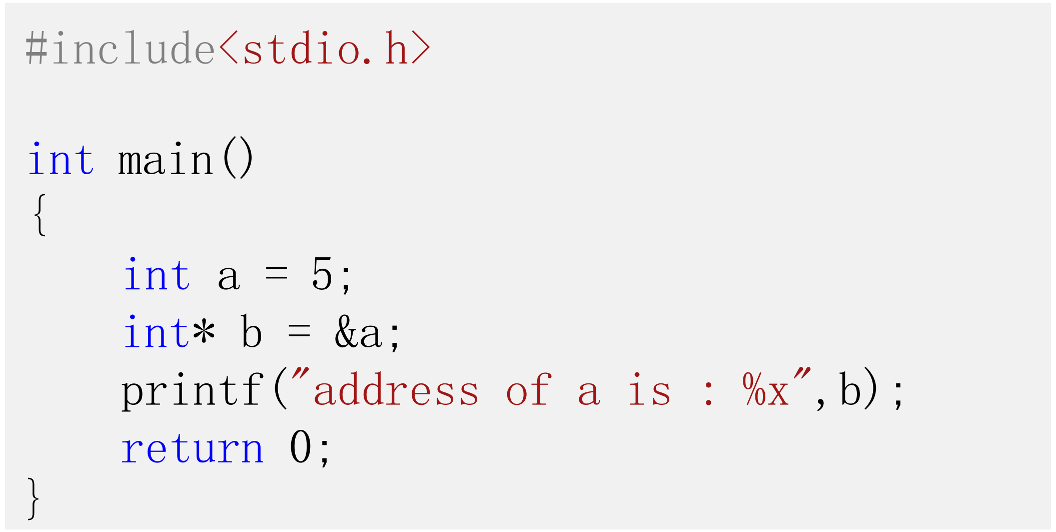


```
本地链接 IPv6 地址. . . . . . . . : fe80::701a:d780:be90:c147%19
```

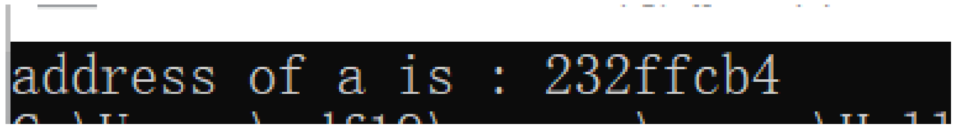


```
3243020 00 00 00 00 00 00 00 00 1b 00 00 00 07 00 00 00
3243040 02 00 00 00 00 00 00 00 70 02 40 00 00 00 00 00
3243060 70 02 00 00 00 00 00 00 20 00 00 00 00 00 00 00
3243100 00 00 00 00 00 00 00 00 08 00 00 00 00 00 00 00
3243120 00 00 00 00 00 00 00 00 2e 00 00 00 07 00 00 00
3243140 02 00 00 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 [illegible] 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 [illegible] 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



```c
#include<stdio.h>

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



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

Content

1. Memory address
2. Pointer
3. Pointer and Array
4. Memory management
5. Pointer and function

What is pointer?

$$\text{Variable} \longrightarrow \text{Address}$$

What is pointer?

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

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

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

} **Stores value**

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

} **Stores address**

声明指针变量时必须指定
指针所指向变量的类型

What is pointer?

int a;

- a has type of int
- a stores value

int *b;

- b has type of int*
- b stores address

Pointer declaration and definition

Variable stores an integer value

```
int a = 10;
```

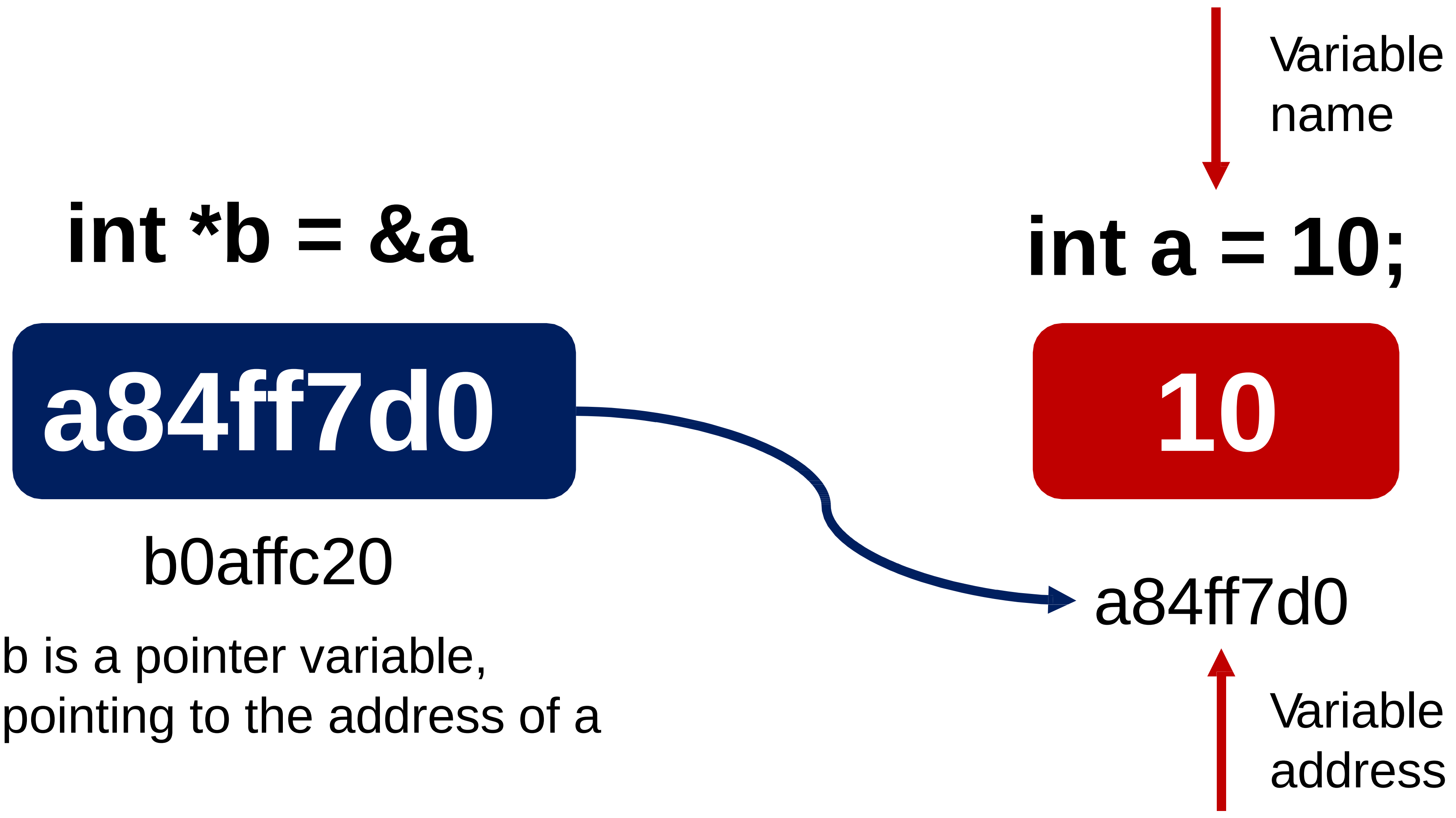
注意:指针变量中只能存放地址（指针），不要将一个整数（或任何其他非地址类型的数据）赋给一个指针变量。

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

Pointer stores the address of an integer variable

Get the address

Pointer declaration and definition



```
int *b = &a
```

```
int a = 10;
```

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

Pointer declaration and definition

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

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

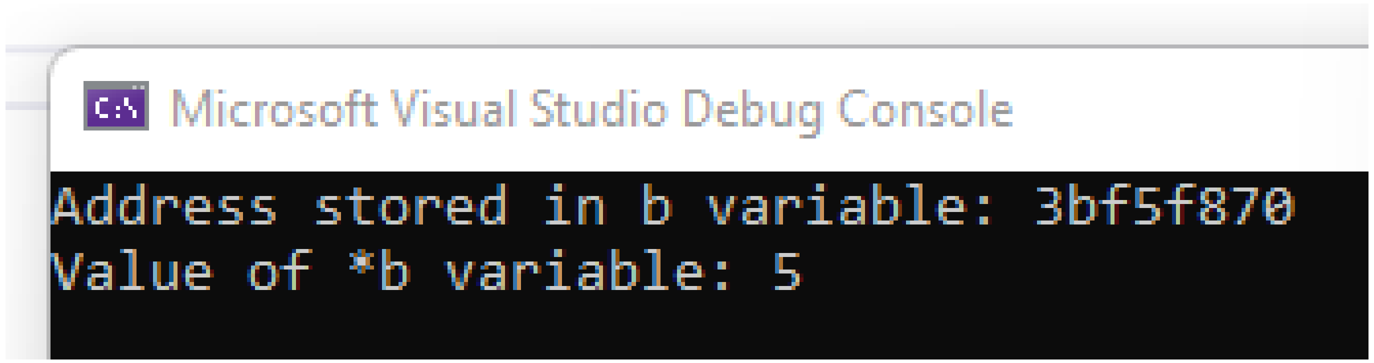
How to interpret pointer?

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

- ✓ int *b: b is a pointer with type int*
- ✓ int *b: *b is a variable with type int

How to interpret pointer?

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


Microsoft Visual Studio Debug Console

```
Address stored in b variable: 3bf5f870
Value of *b variable: 5
```

```
printf("Address stored in b variable: %x\n", b);

printf("Value of *b variable: %d\n", *b);
```

Pointer stores address

Pointer is used to store address, not value!

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

✓

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

✓

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

✗

Pointer stores address

Pointer needs to be assigned with an address; the value it points to can be changed!

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

✓

```
int *b = 5;
```

✗

Pointer must be initialized with address

```
int *a;
*a = 10;
```

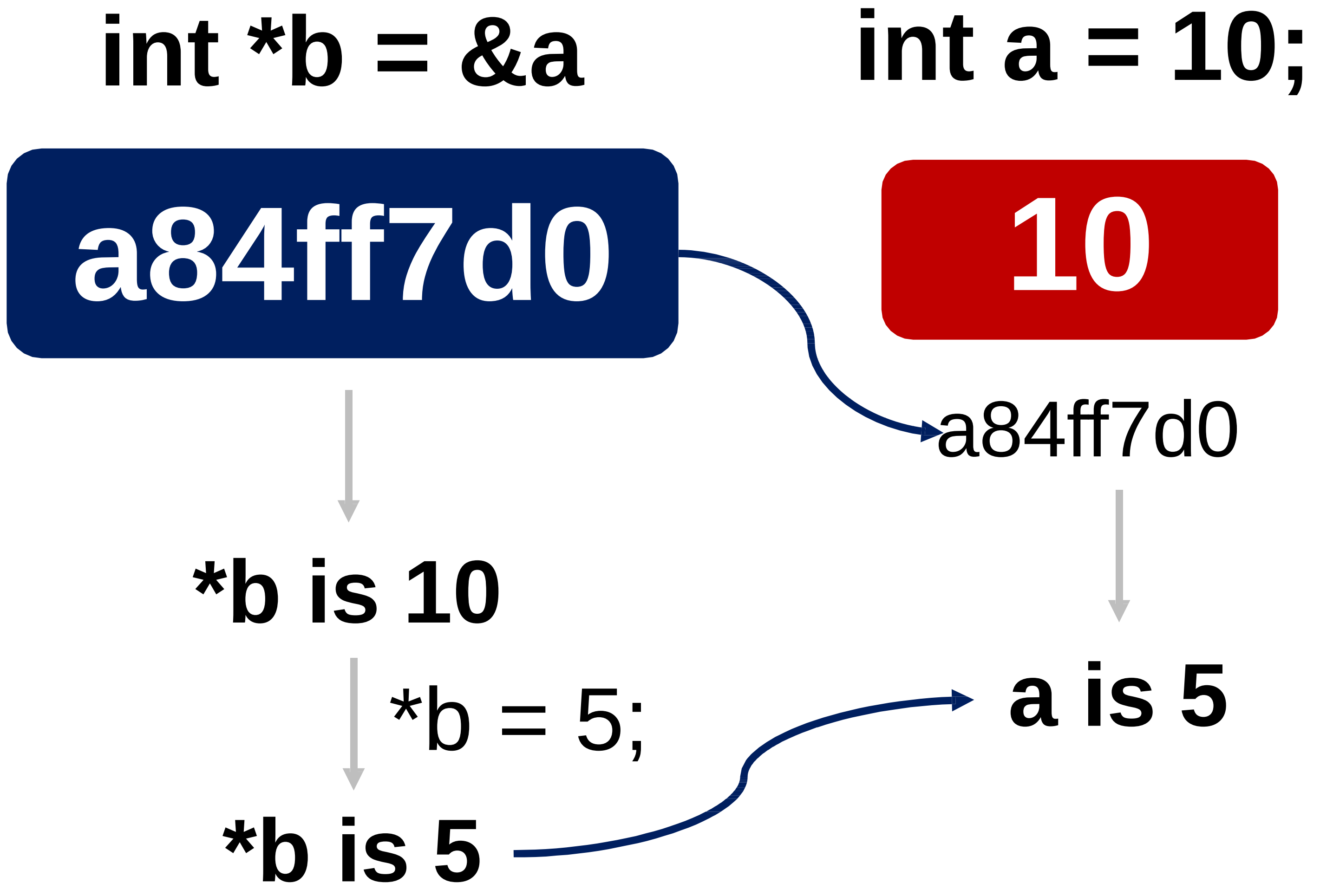
✗

Pointer must be initialized

Pointer allows changing source

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

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

What is a?

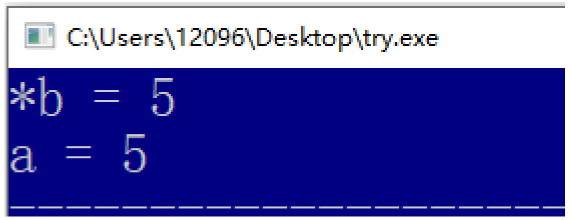


Pointer allows changing source

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

printf(“*b = %d”, *b);
printf(“a = %d”, a);
```

Variable	Address	Content
a	ffc1	00000101
b	ffc2	ffc1

Change the value of *b will influence a!!!


C:\Users\12096\Desktop\try.exe

```
*b = 5
a = 5
```

& and *

地址运算符：&

一般注解：后跟一个变量名，&给出该变量的地址

示例：&nurse表示nurse的地址

间接（或解引用）运算符：*

一般注解：后跟一个指针名或地址时，*给出存储在指针指向地址上的值

示例：val = *ptr; //把ptr指向的地址上的值赋给val

& and *

如果已执行了语句 pointer_1＝＆ a ；

(1) ＆* pointer_1的含义是什么？

“＆”和“*”两个运算符的优先级别相同，但按自右而左方向结合。因此，＆* pointer_1与＆ a 相同，即变量a的地址。

如果有pointer_2 ＝＆* pointer_1 ；

它的作用是将＆ a （ a 的地址）赋给pointer_2 ，如果pointer_2原来指向 b ，经过重新赋值后它已不再指向 b 了，而指向了 a 。

& and *

如果已执行了语句 pointer_1＝& a ;

(2) *& a 的含义是什么?

先进行& a 运算，得 a 的地址，再进行*运算。*& a 和*pointer_1的作用是一样的，它们都等价于变量 a 。即*& a 与 a 等价。

(3) （*pointer_1）＋＋相当于 a ＋＋。

Case of value swap

How to swap values between two variables?

```
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);
}


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

v1 = 10, v2 = 5

v1 = 5, v2 = 10

Changes inside function cannot influence outside

a = 10, b = 5

a = 10, b = 5

Case of value swap

Procedure

```
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);
}


int main(void)
{
    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

Case of value swap

How to use pointer to swap values?

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


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

*v1 = 10, *v2 = 5

*v1 = 5, *v2 = 10

Changes made to memory address influence outside

a = 10, b = 5

a = 5, b = 10

Case of value swap

Procedure

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


int main(void) {
   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 (5→10)
v2	ffc4	ffc2 (10→5)
temp	ffc5	10

Case of value swap

How to use pointer to swap values?

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


int 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);
}
```

*v1 = 10, *v2 = 5

*v1 = 5, *v2 = 10

Changes made to memory address do not influence outside

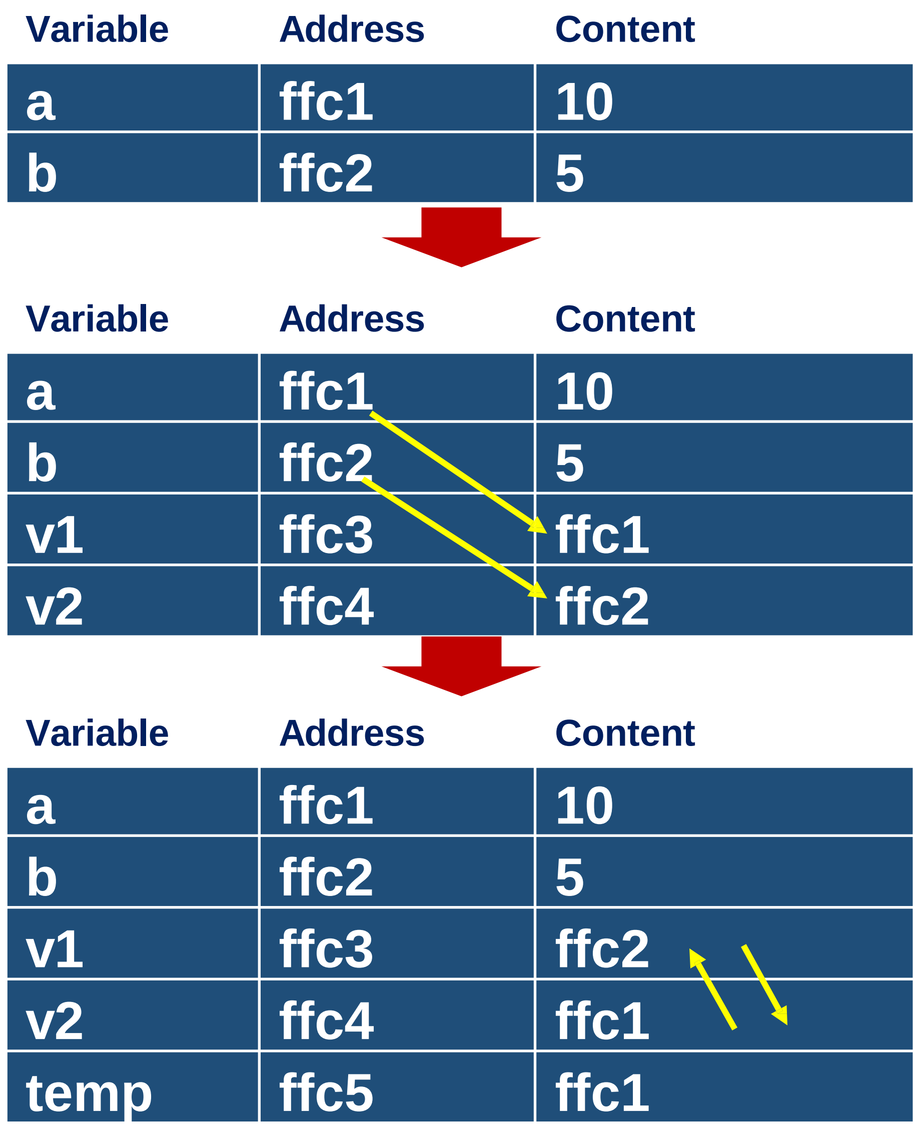
a = 10, b = 5

a = 10, b = 5

Case of value swap

Procedure

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


int 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	10
b	ffc2	5
v1	ffc3	ffc2
v2	ffc4	ffc1
temp	ffc5	ffc1

Case of multiple function outputs

How to output multiple results from a function?

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

int main(void)
{
    int a = 10, b = 5;
    int c = func(a, b);
    return 0;
}
```

We did multiple operations but only return one result!

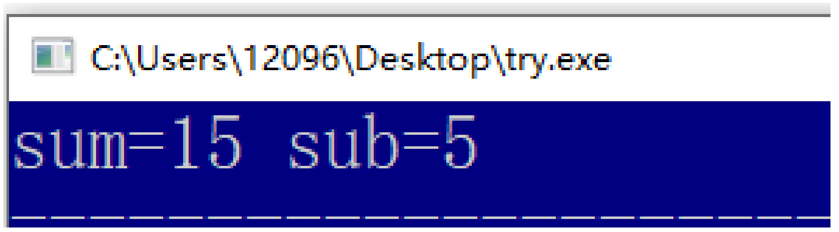
Case of multiple function outputs

How to output multiple results from a function?

```
int func(int v1, int v2, int* sub)
{
    int sum = v1 + v2;
    *sub = v1 - v2;
    return sum;
}

int main(void)
{
    int a = 10, b = 5, sub;
    int sum = func(a, b, &sub);
    printf("sum=%d sub=%d", sum, sub);
    return 0;
}
```

Pass out sub result

Return sum result



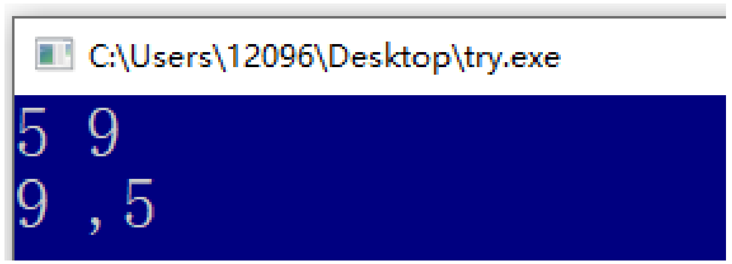
C:\Users\12096\Desktop\try.exe

```
sum=15 sub=5
```

Case of compare size

How to output two input integers in size order?

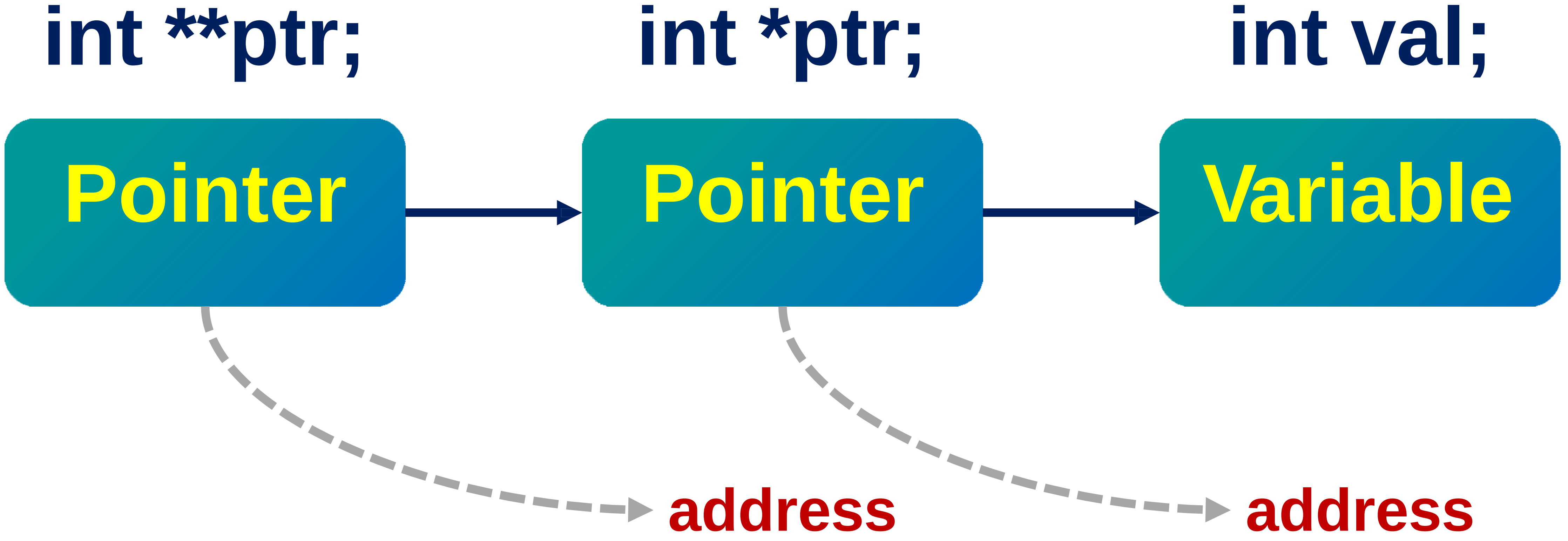
```
#include <stdio.h>
void swap(int *p1, int *p2);
int main(void) {
        int a, b;
        int *pointer_1, *pointer_2;
        scanf("%d %d", &a, &b);
        pointer_1 = &a;
        pointer_2 = &b;
        swap(pointer_1, pointer_2);
        printf("%d ,%d", a, b);
}
void swap(int *p1, int *p2) {
        int temp;
        if (*p1 < *p2) {
                temp = *p1;
                *p1 = *p2;
                *p2 = temp;
        }
}
```



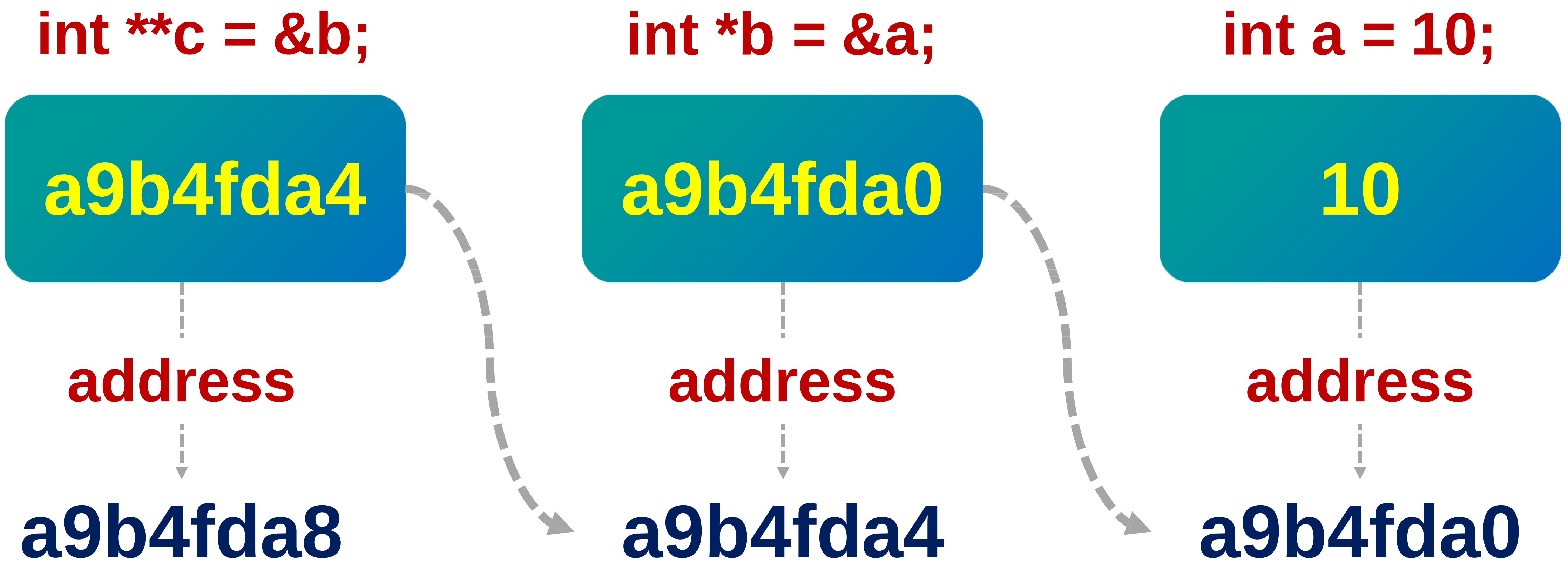
C:\Users\12096\Desktop\try.exe

```
5 9
9 ,5
```

Pointer to pointer



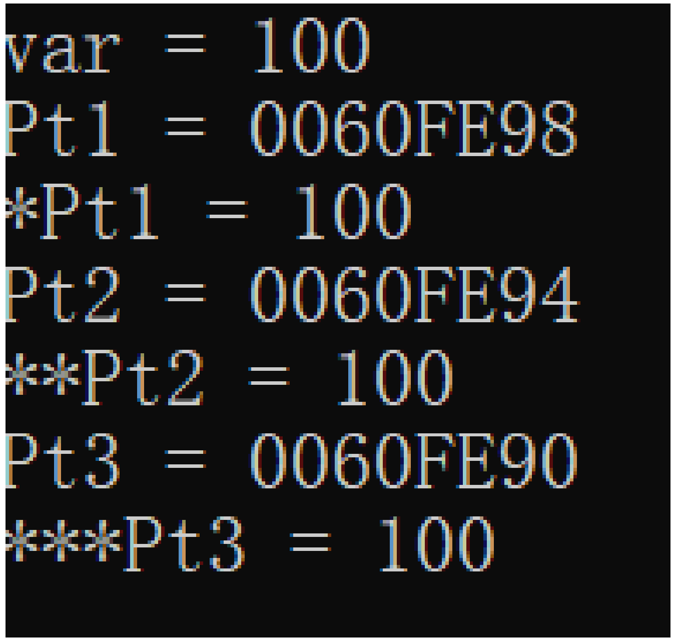
Pointer to pointer



Pointer to pointer

An example of pointer to value, pointer to pointer, pointer to pointer to pointer…

```c
#include <stdio.h>

main()
{
    int  V = 100;
    int* Pt1 = &V;
    int** Pt2 = &Pt1;
    int*** Pt3 = &Pt2;

    printf("var = %d\n", V);
    printf("Pt1 = %p\n", Pt1);
    printf("*Pt1 = %d\n", *Pt1);//100
    printf("Pt2 = %p\n", Pt2);
    printf("**Pt2 = %d\n", **Pt2);//100
    printf("Pt3 = %p\n", Pt3);
    printf("***Pt3 = %d\n", ***Pt3);//100
}
```



```
var = 100
Pt1 = 0060FE98
*Pt1 = 100
Pt2 = 0060FE94
**Pt2 = 100
Pt3 = 0060FE90
***Pt3 = 100
```

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
}
```

Content

1. Memory address
2. Pointer
3. **Pointer and Array**
4. Memory management
5. Pointer and function

Pointer points to array

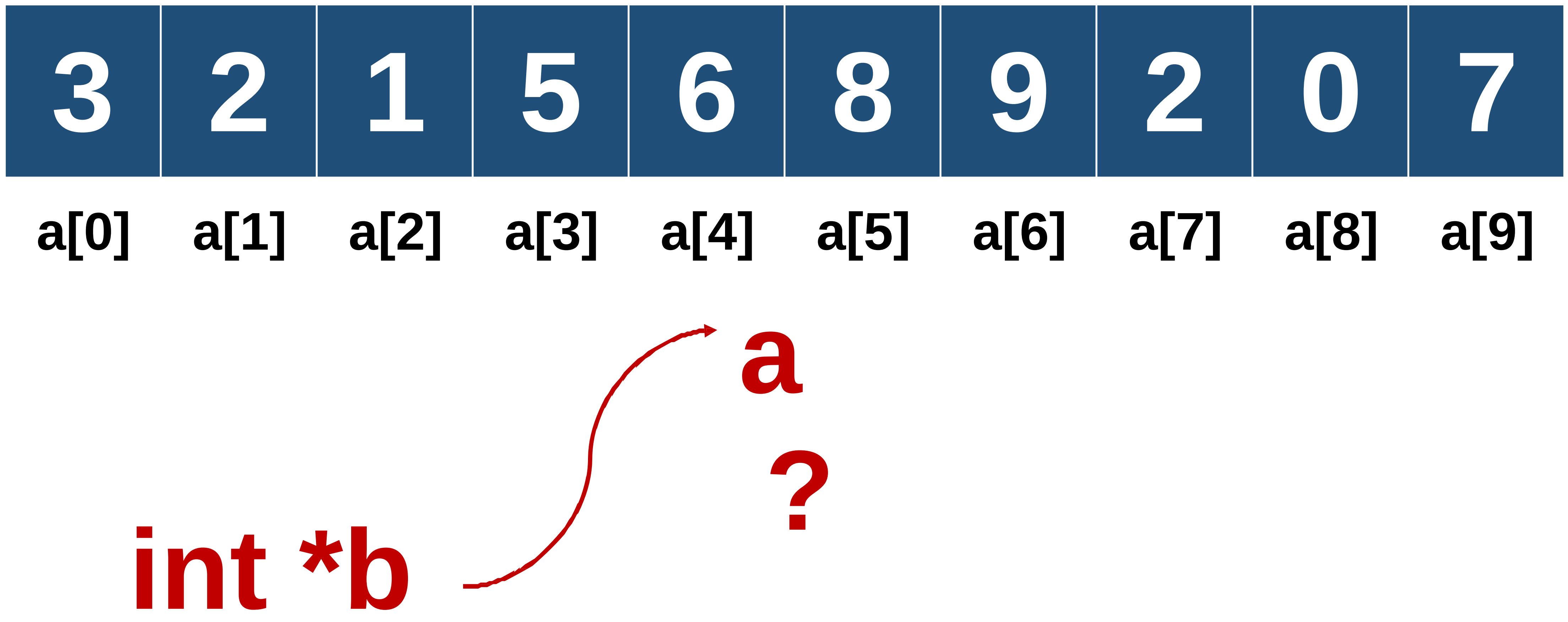
```
int a = 5;
int *b = &a;
```

$\downarrow$

Give the address of a to b!

Pointer points to array

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



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

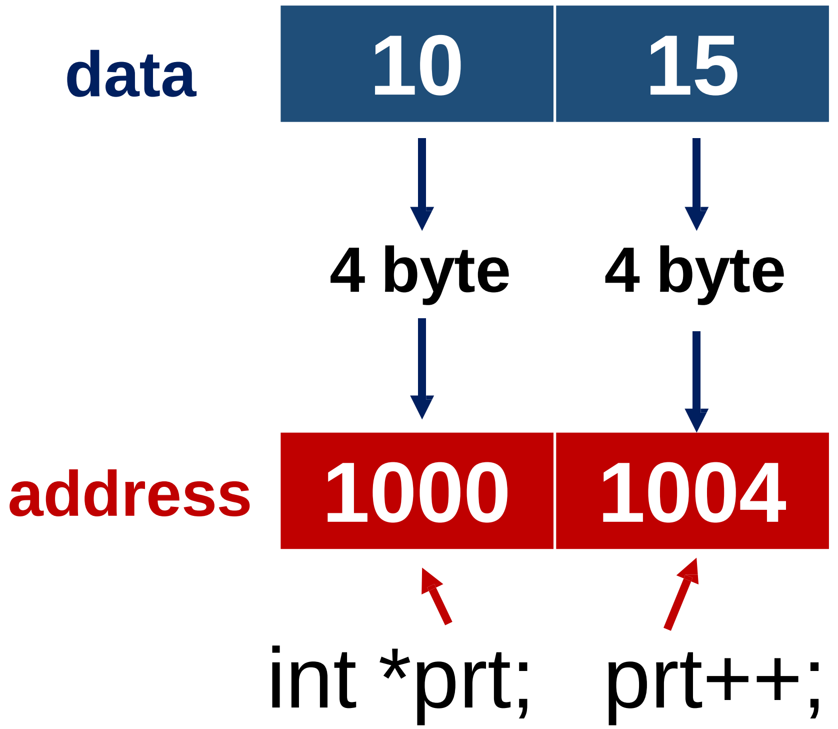
```
int a[3]={1,2,3};

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

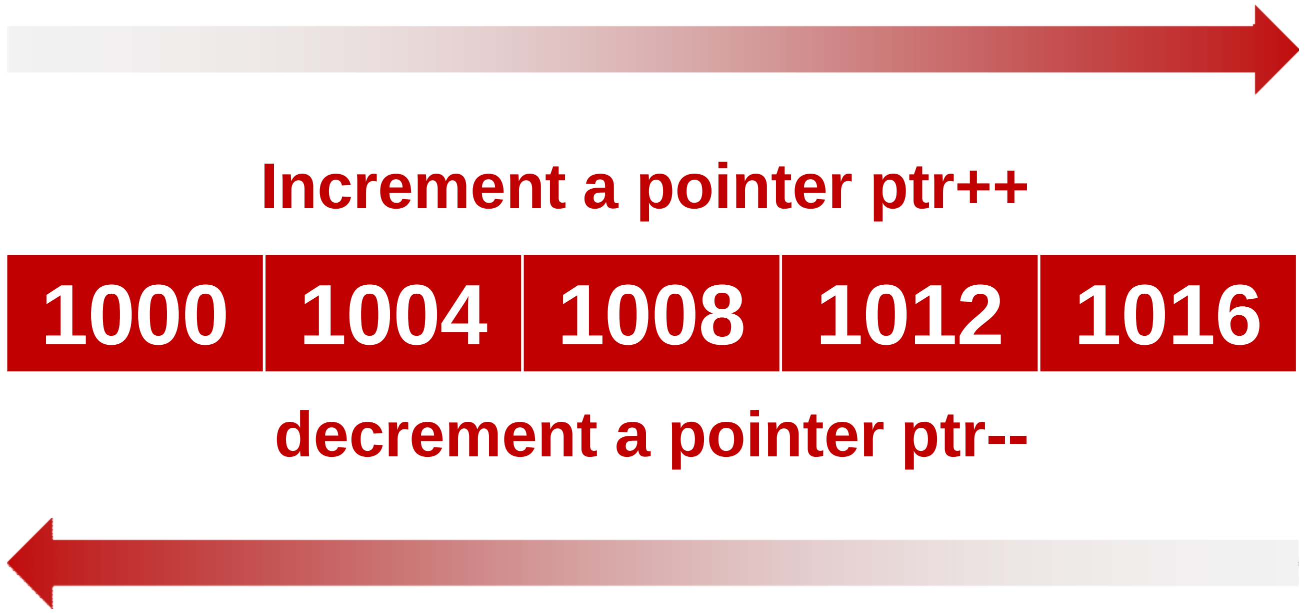
Array	Address	Content
a[0]	17d8f780	1
a[1]	17d8f784	2
a[2]	17d8f788	3
…	…	

Address of the first element is assigned to pointer

Pointer points to array

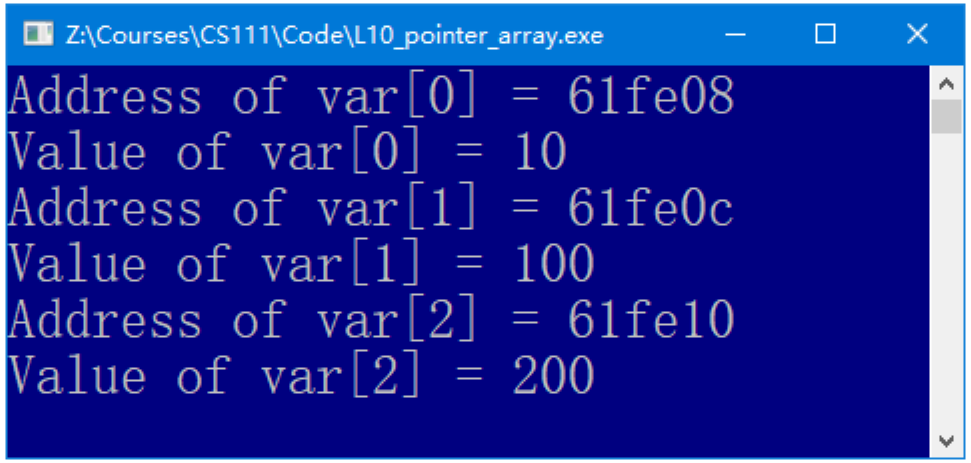
Four arithmetic operators that can be used on pointers: ++, --, +, -





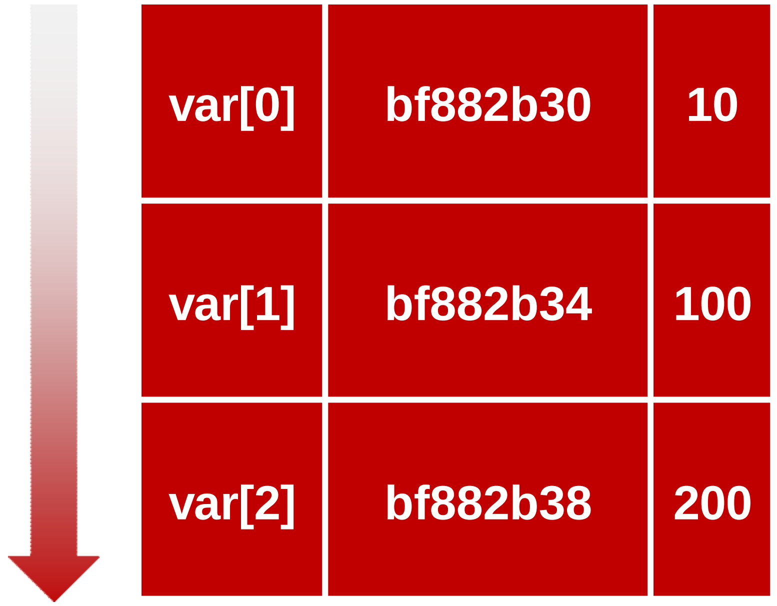
Pointer points to array

```c
#include <stdio.h>

int main (void)
{
    int var[] = {10, 100, 200};
    int *ptr = var; //&var[0]

    for ( int i = 0; i < 3; i++)
    {
        printf("Address of var[%d] = %x\n", i, ptr );
        printf("Value of var[%d] = %d\n", i, *ptr );

        ptr++; /* move to the next location */
    }
}
```



Z:\Courses\CS111\Code\L10_pointer_array.exe

```
Address of var[0] = 61fe08
Value of var[0] = 10
Address of var[1] = 61fe0c
Value of var[1] = 100
Address of var[2] = 61fe10
Value of var[2] = 200
```

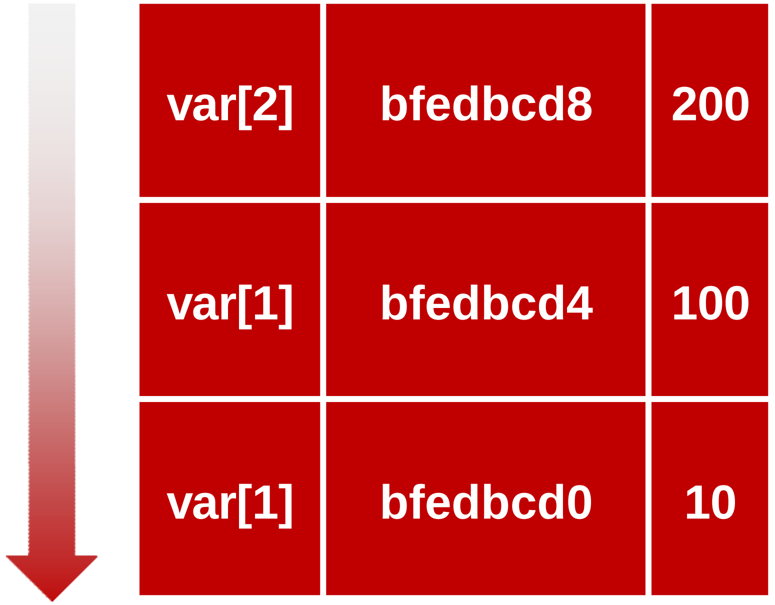


var[0]	bf882b30	10
var[1]	bf882b34	100
var[2]	bf882b38	200

Increments a pointer

Pointer points to array

```
#include <stdio.h>

int main (void)
{
    int var[] = {10, 100, 200};
    int *ptr = &var[3];

    for ( int i = 2; i >= 0; i--)
    {
        printf("Address of var[%d] = %x\n", i, ptr );
        printf("Value of var[%d] = %d\n", i, *ptr );

        ptr--; /* move to the previous location */
    }
}
```

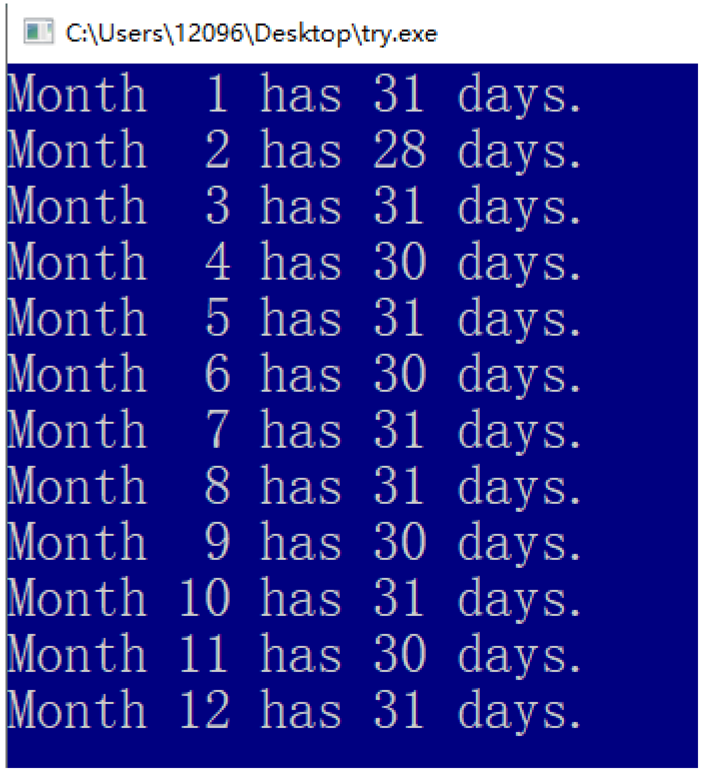


var[2]	bfedbcd8	200
var[1]	bfedbcd4	100
var[1]	bfedbcd0	10

Decrements a pointer

Pointer points to array

```
#include <stdio.h>
#define MONTHS 12

int main(void) {
	int days[MONTHS] = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
	int index;

	for (index = 0; index < MONTHS; index++) {
		printf("Month %2d has %d days.\n", index + 1, *(days + index));
	}
	return 0;
}
```


C:\Users\12096\Desktop\try.exe

```
Month  1 has 31 days.
Month  2 has 28 days.
Month  3 has 31 days.
Month  4 has 30 days.
Month  5 has 31 days.
Month  6 has 30 days.
Month  7 has 31 days.
Month  8 has 31 days.
Month  9 has 30 days.
Month 10 has 31 days.
Month 11 has 30 days.
Month 12 has 31 days.
```

days + index 与(days + index)不一样！！！

间接运算符（*）优先级高于+

Pointer points to array

Use pointer to compare memory address: >, <, ==

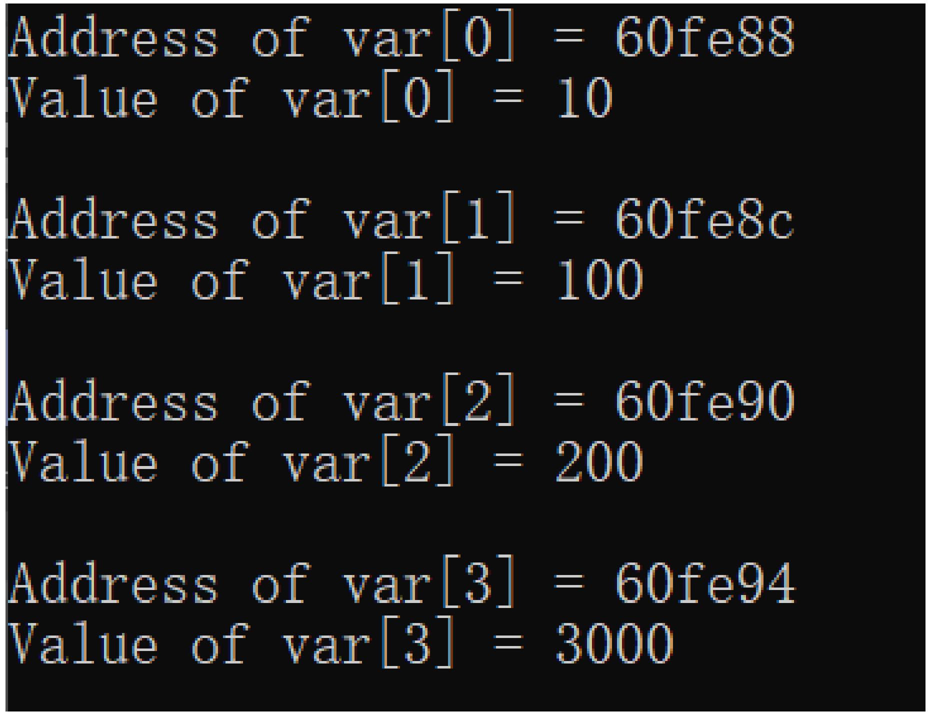
```
int var[] = {10, 100, 200};

int *ptr1 = &var[0];
int *ptr2 = &var[0];
int *ptr3 = &var[1];
```

```
ptr1 == ptr2                    ptr1 < ptr3
```

Pointer points to array

Use pointer to compare memory address: >, <, ==

```c
#include <stdio.h>

int main (void)
{
   int  var[] = {10, 100, 200, 3000};
   int  i, *ptr;

   ptr = var;
   i = 0;
   while (ptr <= &var[3])
   {
      printf("Address of var[%d] = %x\n", i, ptr );
      printf("Value of var[%d] = %d\n\n", i, *ptr );
      ptr++;
      i++;
   }
   return 0;
}
```



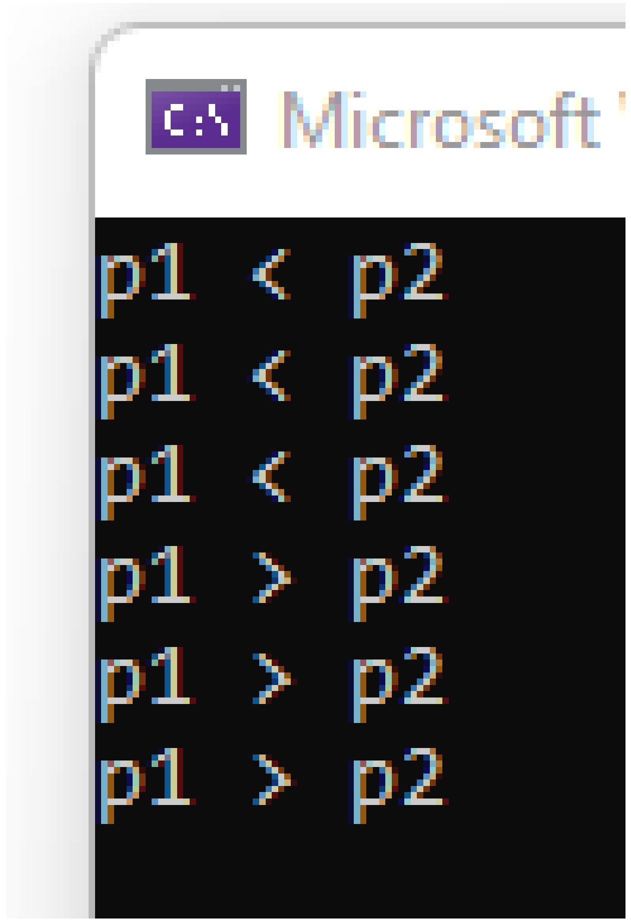
```
Address of var[0] = 60fe88
Value of var[0] = 10

Address of var[1] = 60fe8c
Value of var[1] = 100

Address of var[2] = 60fe90
Value of var[2] = 200

Address of var[3] = 60fe94
Value of var[3] = 3000
```

Pointer points to array

Use pointer to compare memory address: >, <, ==

```c
#include <stdio.h>
main()
{
    char str[7] = "dfghjk", * p1, * p2;
    p1 = str;
    p2 = p1 + 5;

    for (int i = 0; i < 6; i++)
    {
        if (p1 < p2)
        {
            printf("p1 < p2\n");
        }
        else {
            printf("p1 > p2\n");
        }
        *p1++;
        *p2--;
    }
}
```

L10_compare.c



```
p1 < p2
p1 < p2
p1 < p2
p1 > p2
p1 > p2
p1 > p2
```

Pointer points to array

输出数组元素的三种方法：

下标法

```
void  main() {
        int a[10];
        int i;
        for (i = 0; i < 10; i++)
                scanf("%d", &a[i]);
        printf("\n");
        for (i = 0; i < 10; i++)
                printf("%d", a[i]);
}
```

通过数组名计算数组元素地址，找出元素的值

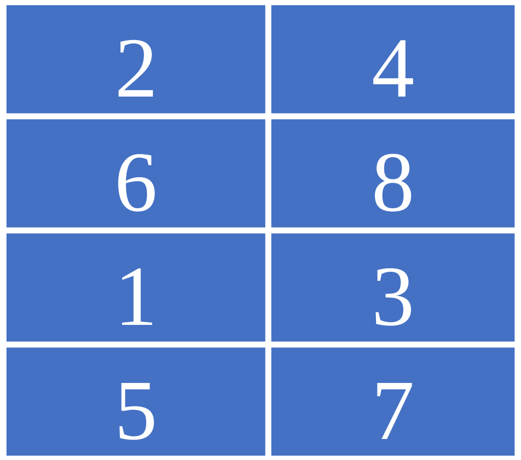
```
void  main() {
        int a[10];
        int i;
        for (i = 0; i < 10; i++)
                scanf("%d", &a[i]);
        printf("\n");
        for (i = 0; i < 10; i++)
                printf("%d",*(a+i));
}
```

用指针变量指向数组元素

```
void  main() {
        int a[10];
        int *p, i;
        for (i = 0; i < 10; i++)
                scanf("%d", &a[i]);
        printf("\n");
        for (p = a; p < (a + 10); p++)
                printf("%d", *p);
}
```

Pointer points to 2D/ND array

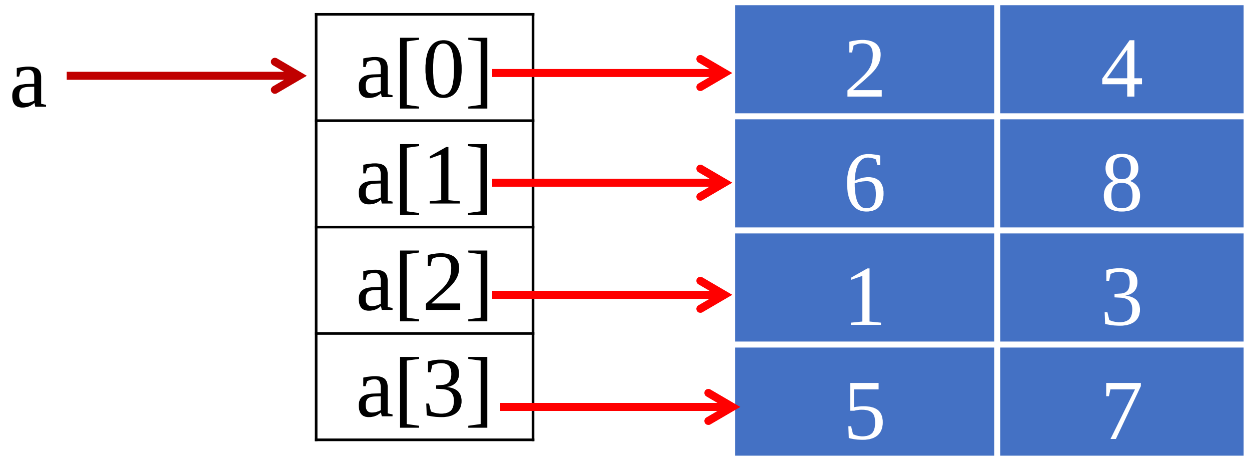
```
#include <stdio.h>

int main(void) {
     int *p;
     int a[4][2]={{2,4},{6,8},{1,3},{5,7}};

     p = a;
     printf("Value of p: %d\n", *p);
}
```



[错误] 无法转换 'int [3][2]' 到 'int*' 在赋值时

Pointer points to 2D/ND array

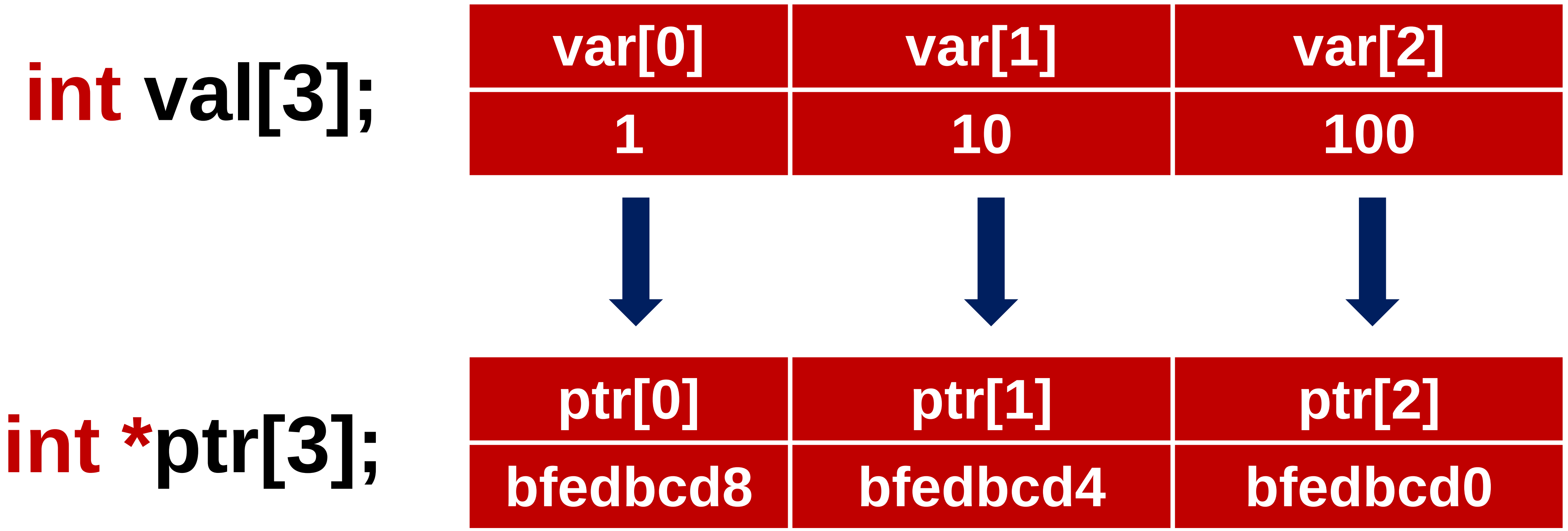
```
int a[4][2]={{2,4},{6,8},{1,3},{5,7}};
```



a[0]、a[1]、a[2]、a[3]分别表示为一维数组

Array of pointers

An array to store pointers



`int val[3];`

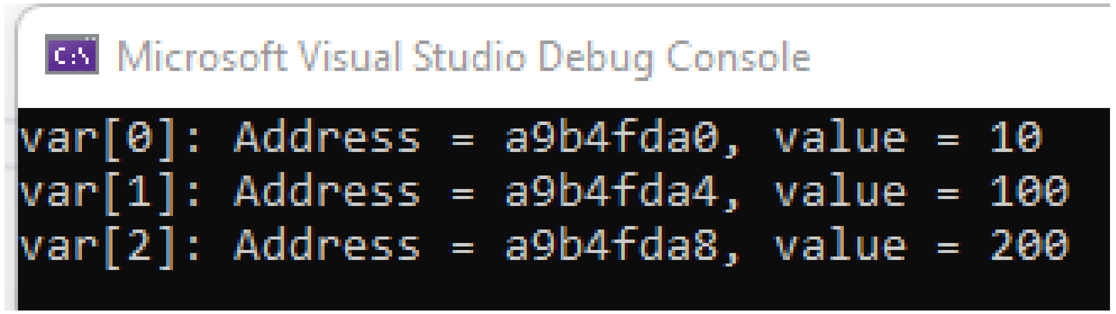
var[0]	var[1]	var[2]
1	10	100

`int *ptr[3];`

ptr[0]	ptr[1]	ptr[2]
bfedbcd8	bfedbcd4	bfedbcd0

Array of pointers

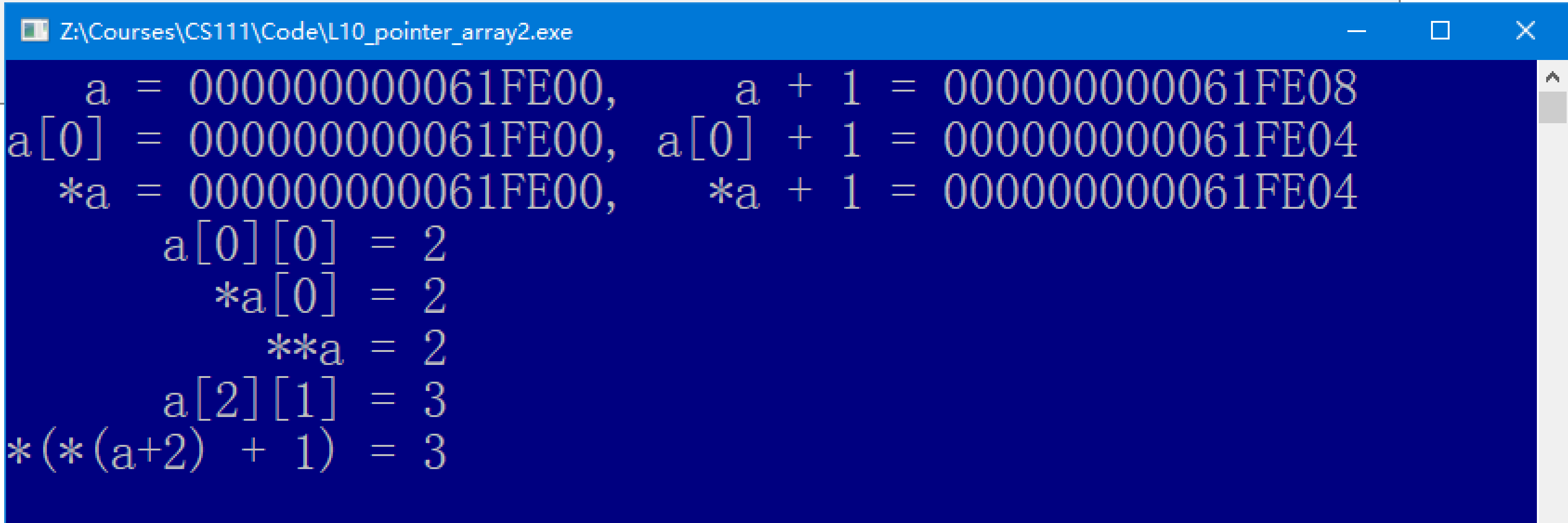
```c
#include <stdio.h>

int main (void)
{
    int var[] = {10, 100, 200};
    int *ptr[3];
    for (int i = 0; i < 3; i++)
    {
        ptr[i] = &var[i];
        printf("Address of var[%d] = %d\n", i, ptr[i]);
        printf("Value of var[%d] = %d\n", i,  *ptr[i]);
    }
}
```



```
var[0]: Address = a9b4fda0, value = 10
var[1]: Address = a9b4fda4, value = 100
var[2]: Address = a9b4fda8, value = 200
```

Pointer points to 2D/ND array

```c
int main(void) {
	int a[4][2] = { { 2, 4 }, { 6, 8 }, { 1, 3 },{ 5, 7 }};
	printf("   a = %p,     a + 1 = %p\n", a, a + 1);
	printf("a[0] = %p, a[0] + 1 = %p\n", a[0], a[0] + 1);
	printf("  *a = %p,    *a + 1 = %p\n", *a, *a + 1);
	printf("      a[0][0] = %d\n", a[0][0]);
	printf("        *a[0] = %d\n", *a[0]);
	printf("          **a = %d\n", **a);
	printf("      a[2][1] = %d\n", a[2][1]);
	printf("*(*(a+2) + 1) = %d\n", *(*(a + 2) + 1));
	return 0;
}
```



Z:\Courses\CS111\Code\L10_pointer_array2.exe

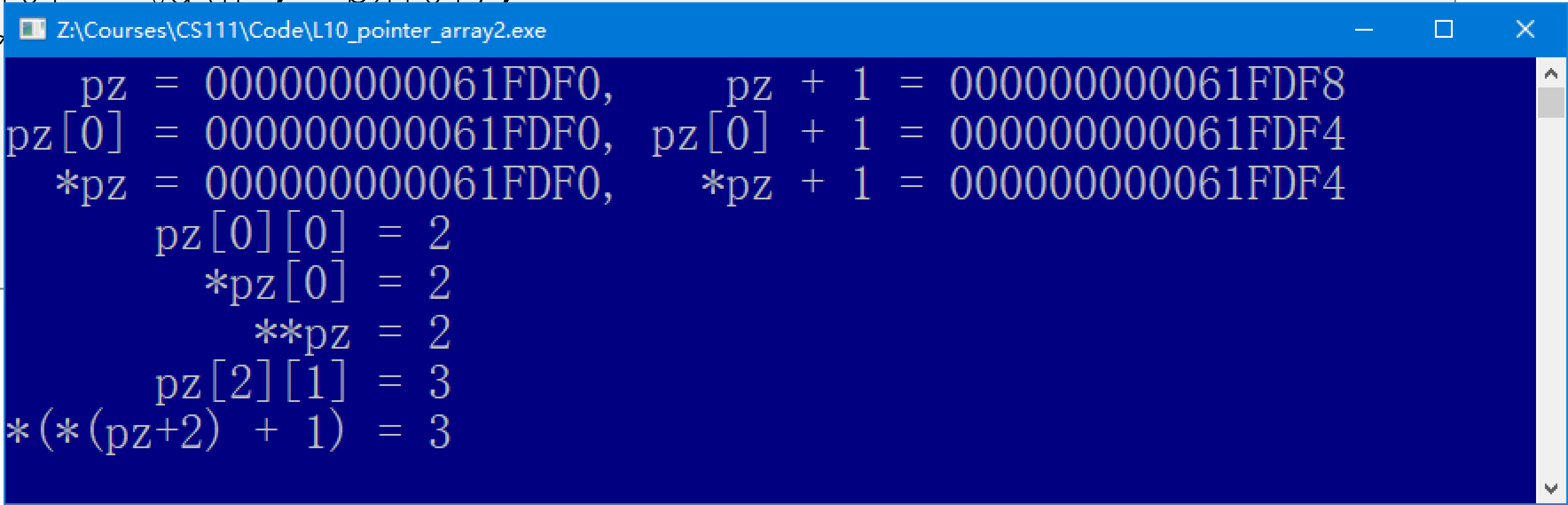
```
   a = 000000000061FE00,     a + 1 = 000000000061FE08
a[0] = 000000000061FE00, a[0] + 1 = 000000000061FE04
  *a = 000000000061FE00,    *a + 1 = 000000000061FE04
      a[0][0] = 2
        *a[0] = 2
          **a = 2
      a[2][1] = 3
*(*(a+2) + 1) = 3
```

Pointer points to 2D/ND array

声明指向数组的指针：（数组指针）

type (*name)[2]

```
int main(void) {
        int a[4][2] = { { 2, 4 }, { 6, 8 }, { 1, 3 }, { 5, 7 } };
        int(*pz)[2];
        pz = a;
        printf("    pz = %p,     pz + 1 = %p\n", pz, pz + 1);
        printf("pz[0] = %p, pz[0] + 1 = %p\n", pz[0], pz[0] + 1);
        printf("   *pz = %p,    *pz + 1 = %p\n", *pz, *pz + 1);
        printf("       pz[0][0] = %d\n", pz[0][0]);
        printf("         *pz[0] = %d\n", *pz[0]);
        printf("           **
        printf("       pz[2]
        printf("*(*(pz+2) +
        return 0;
}
```


Z:\Courses\CS111\Code\L10_pointer_array2.exe

```
    pz = 000000000061FDF0,     pz + 1 = 000000000061FDF8
pz[0] = 000000000061FDF0, pz[0] + 1 = 000000000061FDF4
   *pz = 000000000061FDF0,    *pz + 1 = 000000000061FDF4
       pz[0][0] = 2
         *pz[0] = 2
           **pz = 2
       pz[2][1] = 3
*(*(pz+2) + 1) = 3
```

Pointer points to 2D/ND array

假设有如下声明：

```
int * pt;
int (*pa)[3];
int ar1[2][3];
int ar2[3][2];
int **p2; // 一个指向指针的指针
```

有如下的语句：

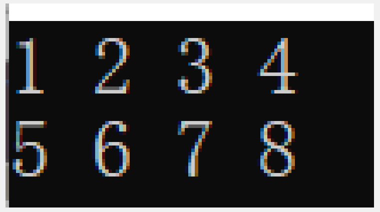
```
pt = &ar1[0][0]; // 都是指向int的指针
pt = ar1[0];      // 都是指向int的指针
pt = ar1;         // 无效
pa = ar1;         // 都是指向内含3个int类型元素数组的指针
pa = ar2;         // 无效
p2 = &pt;         // 都是指向int *的指针
*p2 = ar2[0];     // 都是指向int的指针
p2 = ar2;         // 无效
```

?

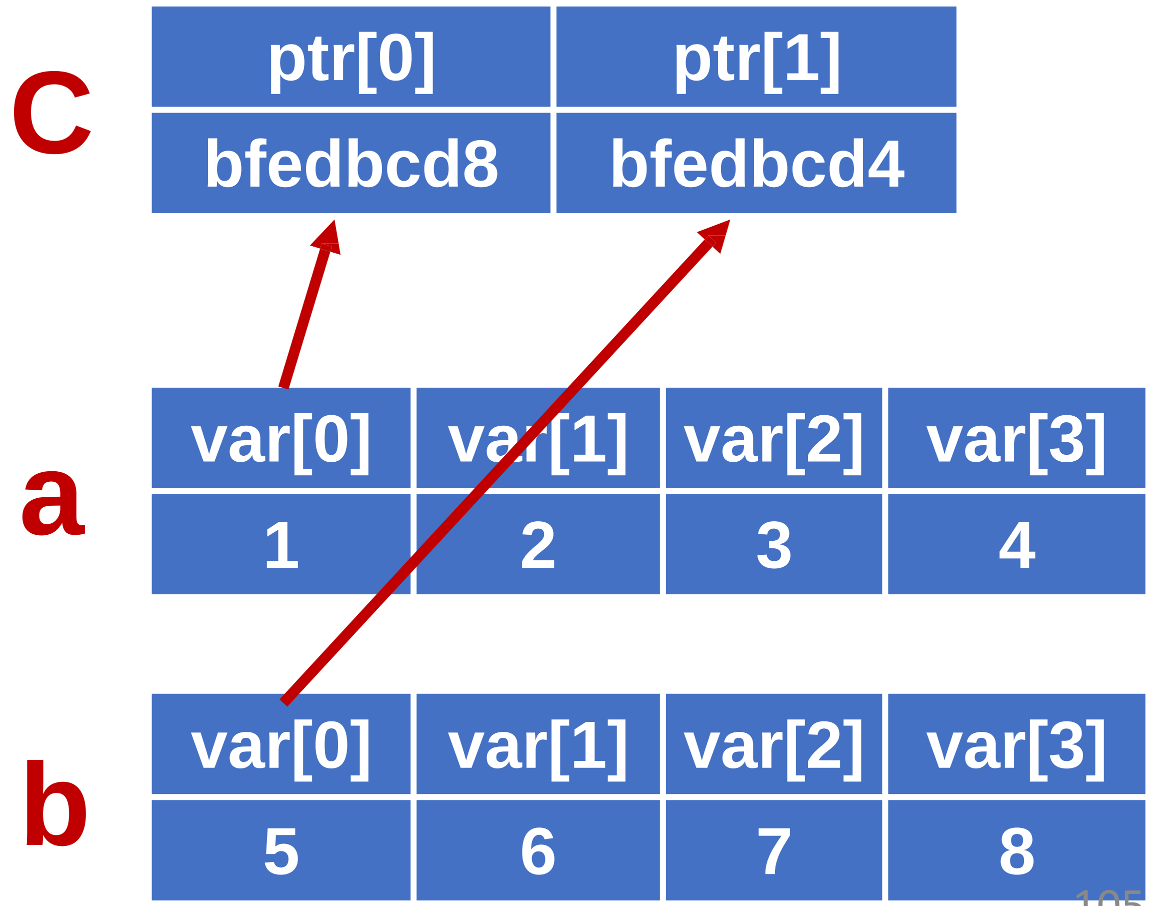
Array of pointers

We can use a 1-D array of pointers to store a 2-D int array

```
#include<stdio.h>

int main(void)
{
  int a[4]={1,2,3,4};
  int b[4]={5,6,7,8};
  int* c[]={a,b};//指针数组
  for(int i = 0; i < 2; i++)
{
    for(int j = 0;j < 4;j++)
    printf("%d ",c[i][j]);
    printf("\n");
}
return 0;
}
```

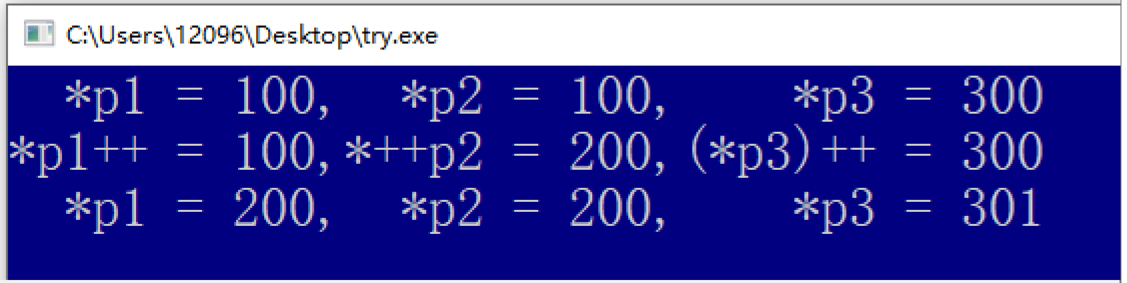


```
1 2 3 4
5 6 7 8
```



operators

```
#include <stdio.h>

int data[2] = {100, 200};
int moredata[2] = {300, 400};

int main(void) {
        int *p1, *p2, *p3;
        p1 = p2 = data;
        p3 = moredata;
        printf("  *p1 = %d,   *p2 = %d,     *p3 = %d\n", *p1, *p2, *p3);
        printf("*p1++ = %d,*++p2 = %d,(*p3)++ = %d\n", *p1++, *++p2, (*p3)++);
        printf("  *p1 = %d,   *p2 = %d,     *p3 = %d\n", *p1, *p2, *p3);
        return 0;
}
```


C:\Users\12096\Desktop\try.exe

```
  *p1 = 100,   *p2 = 100,     *p3 = 300
*p1++ = 100,*++p2 = 200,(*p3)++ = 300
  *p1 = 200,   *p2 = 200,     *p3 = 301
```