# Introduction to C Programming

# Lecture 6: pointer

Wenjin Wang

wangwj3@sustech.edu.cn

10-14-2022

# Course syllabus

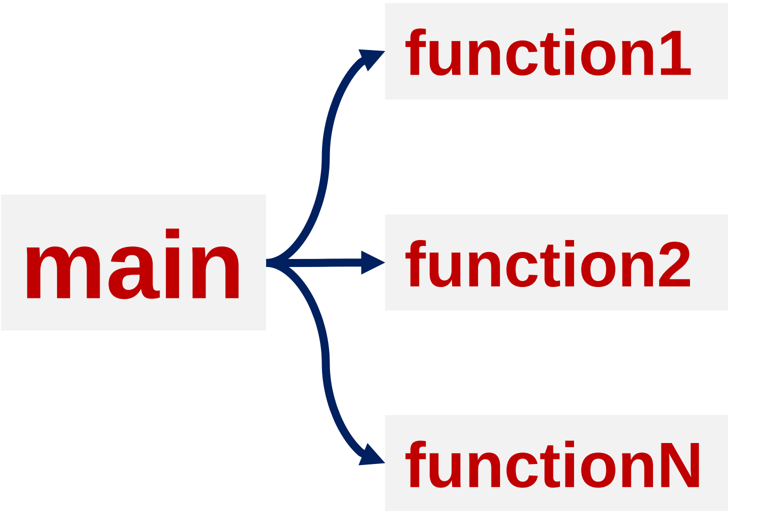
| Nr. | Lecture | Date |
|---|---|---|
| 1 | Introduction | 2022.9.9 |
| 2 | Basics | 2022.9.16 |
| 3 | Decision and looping | 2022.9.23 |
| 4 | Array & string | 2022.9.30 |
| 5 | Functions | 2022.10.9 (补) |
| 6 | Pointer | 2022.10.14 |
| 7 | Self-defined types | 2022.10.21 |
| 8 | File I/O | 2022.10.28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

| Nr. | Lecture | Date |
|---|---|---|
| 9 | Head files & pre-processors | 2022.11.4 |
| 10 | Review of lectures | 2022.11.11 |
| 11 | Soul of programming: Algorithms I | 2022.11.25 |
| 12 | Soul of programming: Algorithms II | 2022.12.2 |
| 13 | R&D project | 2022.12.9 |
| 14 | R&D project | 2022.12.16 |
| 15 | R&D project | 2022.12.23 |
| 16 | Summary | 2023.12.30 |

# 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

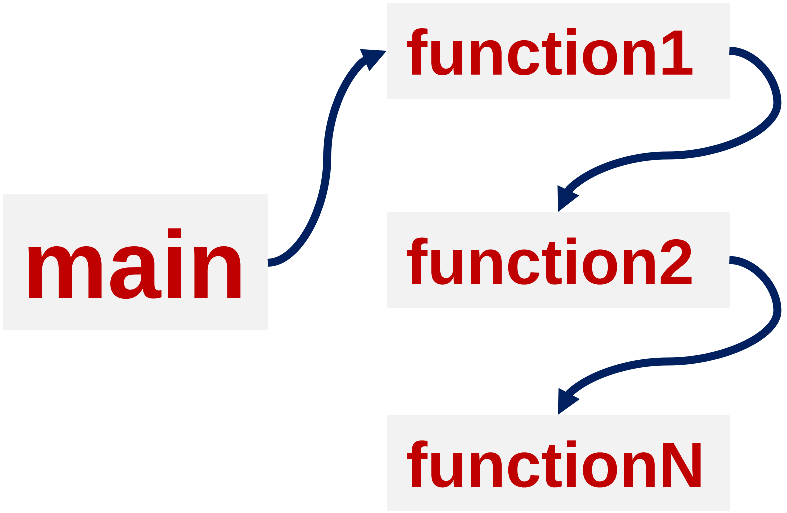


main

function1

function2

functionN

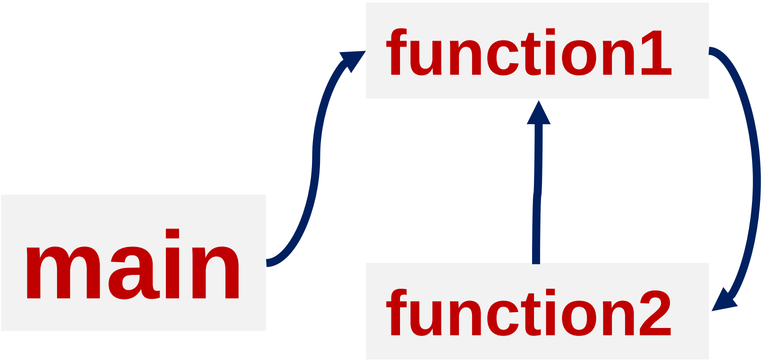


main

function1

function2

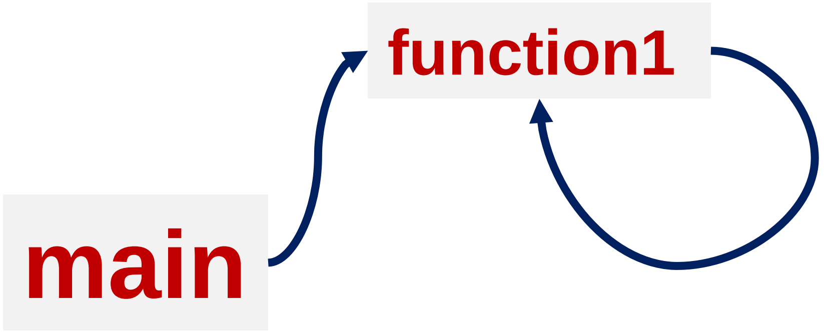
functionN



main

function1

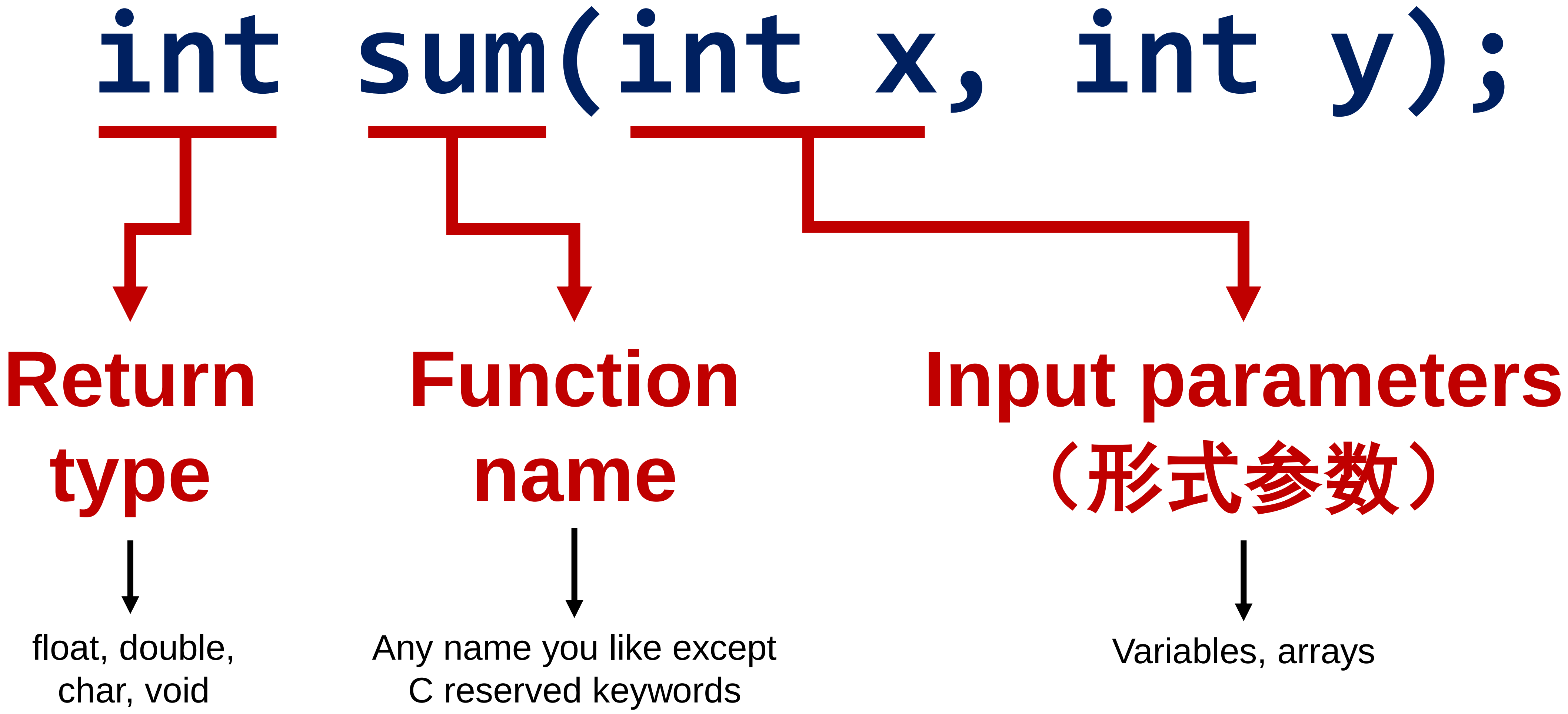
function2



main

function1

# Step 1: declare a function



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

**Return type**

float, double, char, void

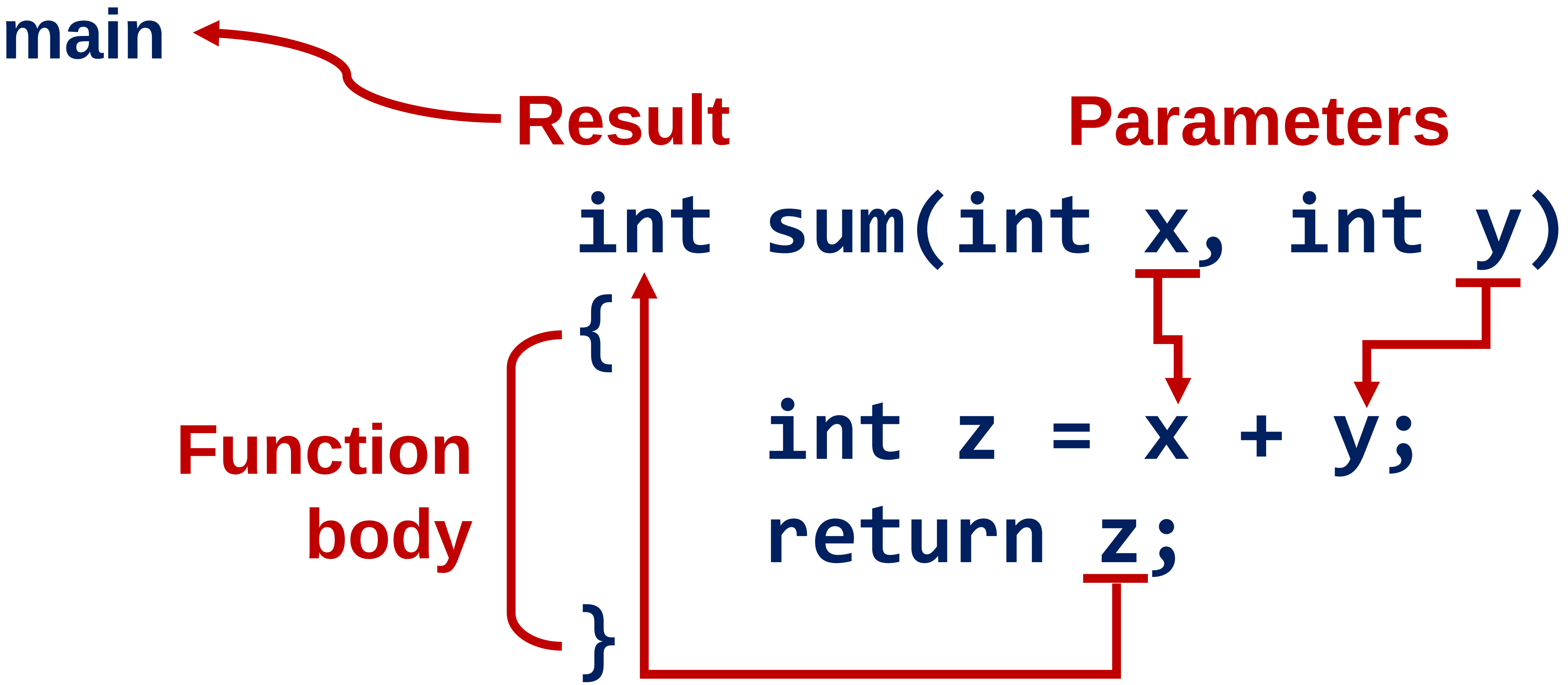
**Function name**

Any name you like except C reserved keywords

**Input parameters （形式参数）**

Variables, arrays

# Step 2: define a function



main

Result

Parameters

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

Function body

# Step 3: call a function

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

**Arguments**
**（实际参数）**

# Recap last lecture

```
#include<stdio.h>

int max(int x, int y);

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

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

```
#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

# Recap last lecture

```c
#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);
}
```

```c
#include<stdio.h>

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

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

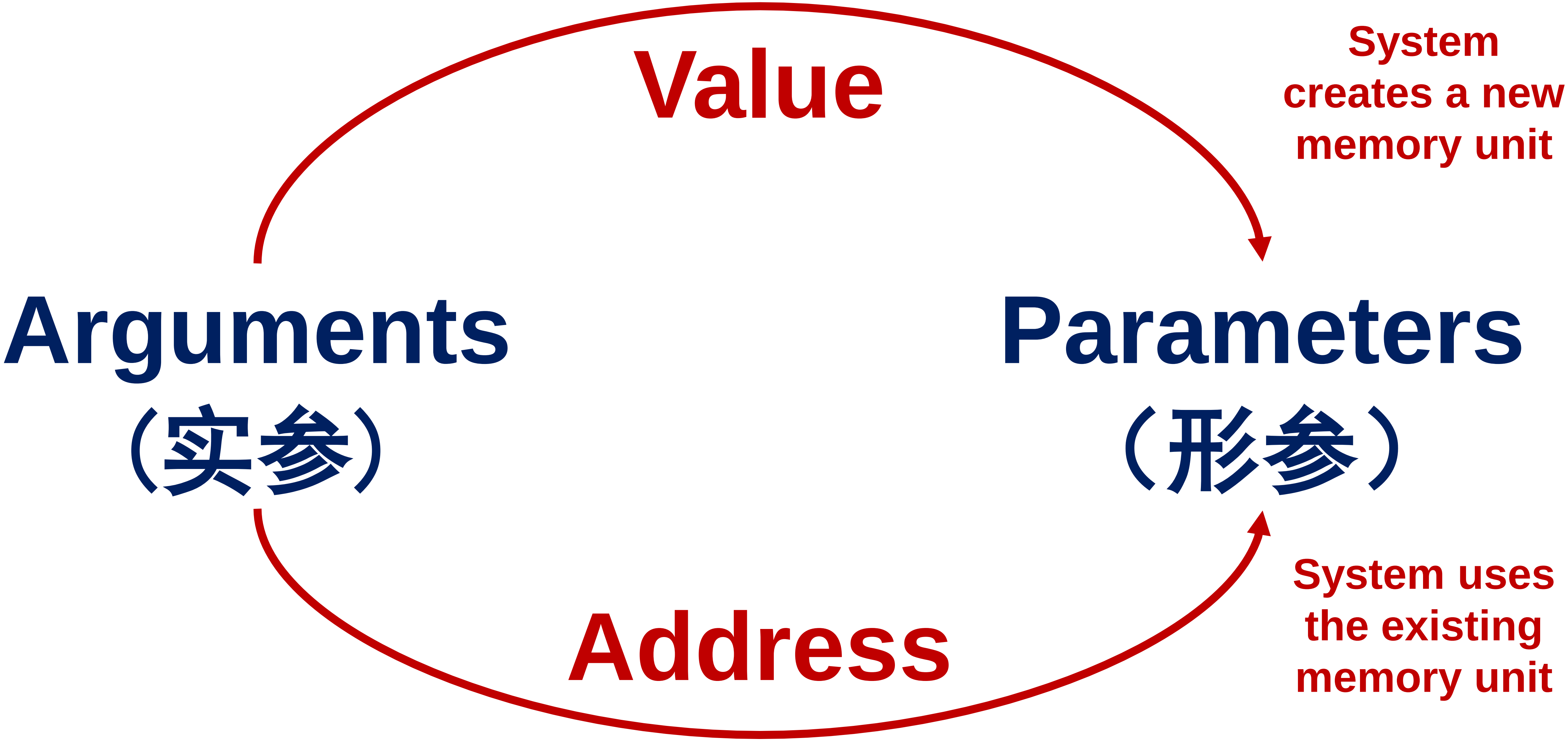
①

**Declare and define function before main!!!**

②

**Call function**

# Recap last lecture



Value

System creates a new memory unit

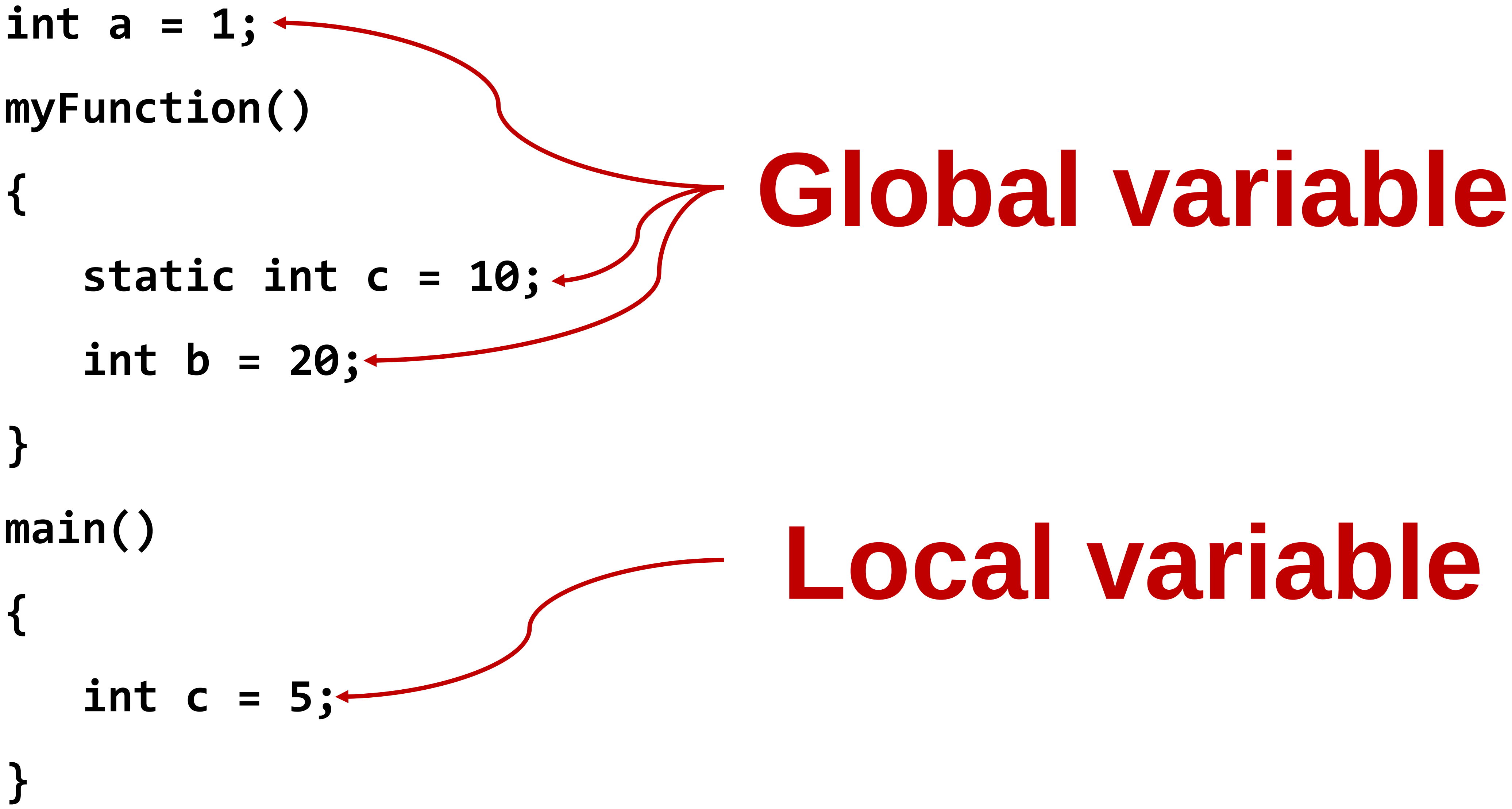
Arguments （实参）

Parameters （形参）

Address

System uses the existing memory unit

# Recap last lecture



```
int a = 1;
myFunction()
{
    static int c = 10;
    int b = 20;
}
main()
{
    int c = 5;
}
```

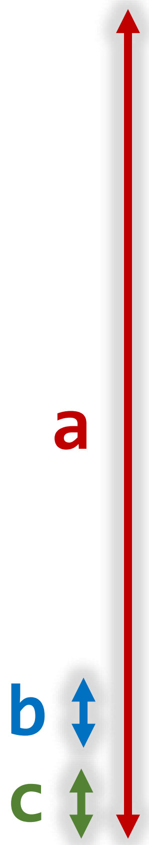
Global variable

Local variable

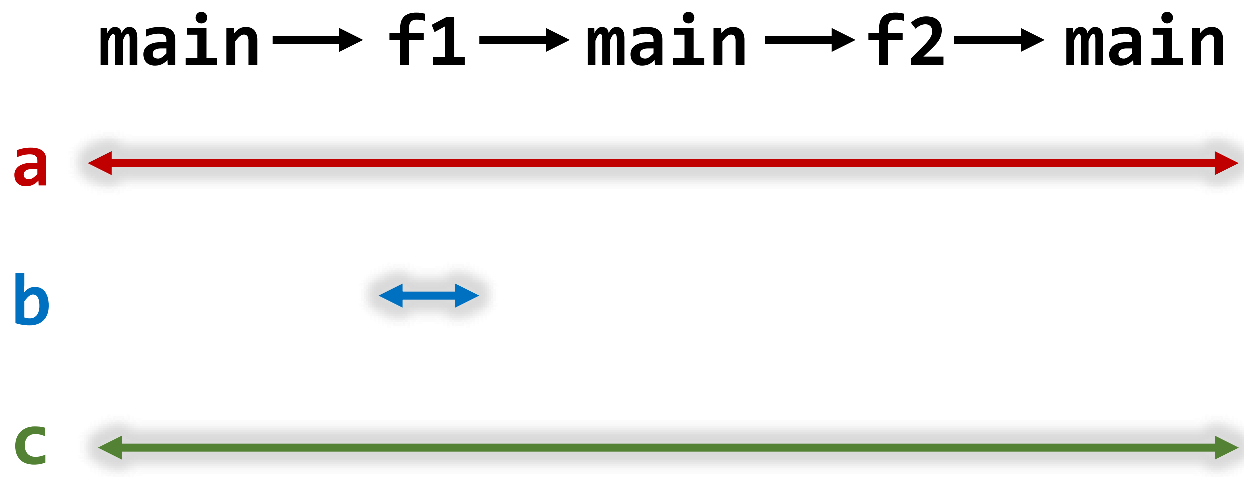
# Recap last lecture

## Scope in space

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



## Scope in time


main → f1 → main → f2 → main

# Objective of this lecture

**You know how to use pointer!**

# Content

1. Memory address
2. Pointer
3. Memory management

# Content

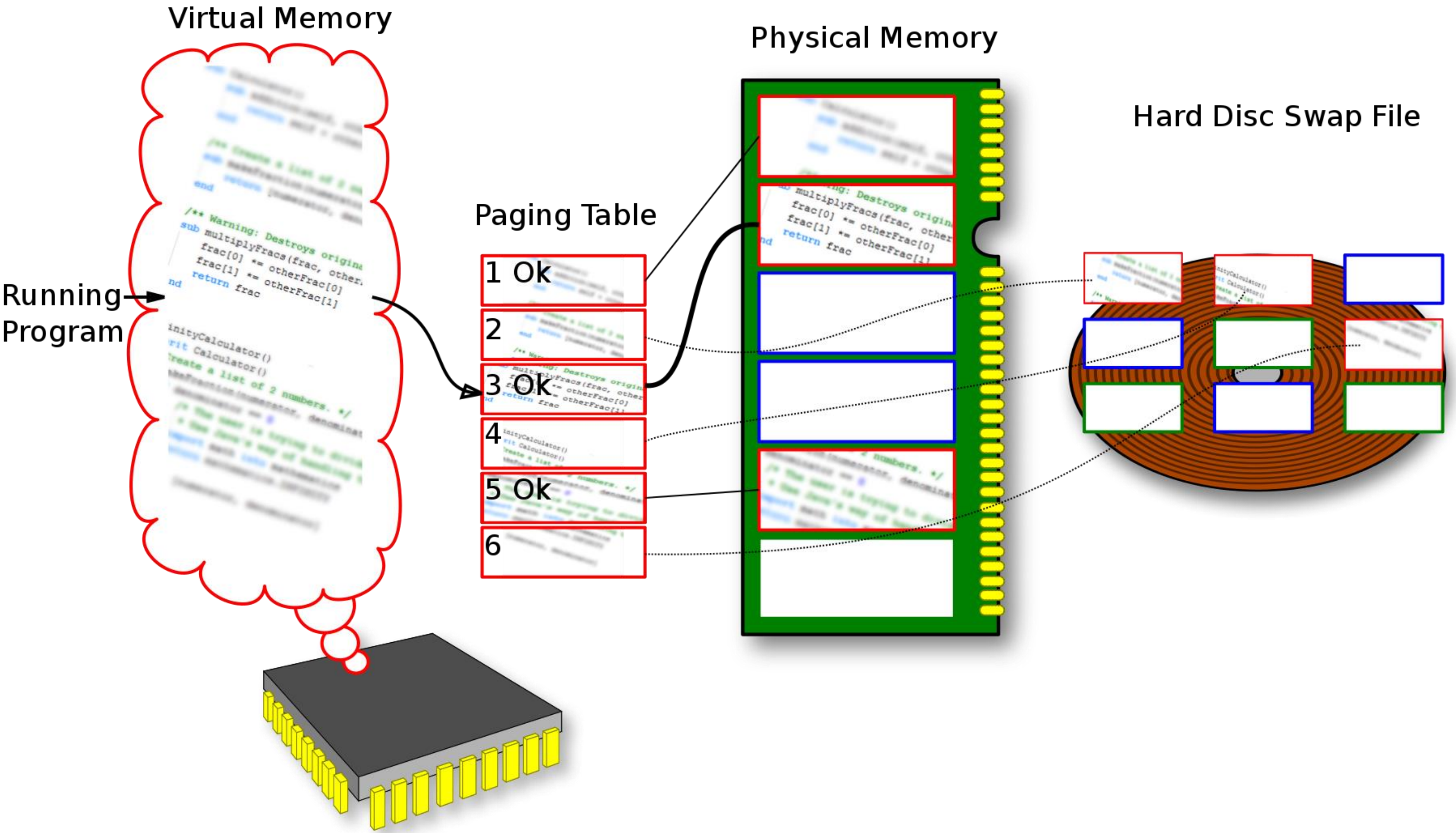
1. **Memory address**
2. Pointer
3. Memory management

# Memory address

How can we find a person?

**Address**

# Memory address



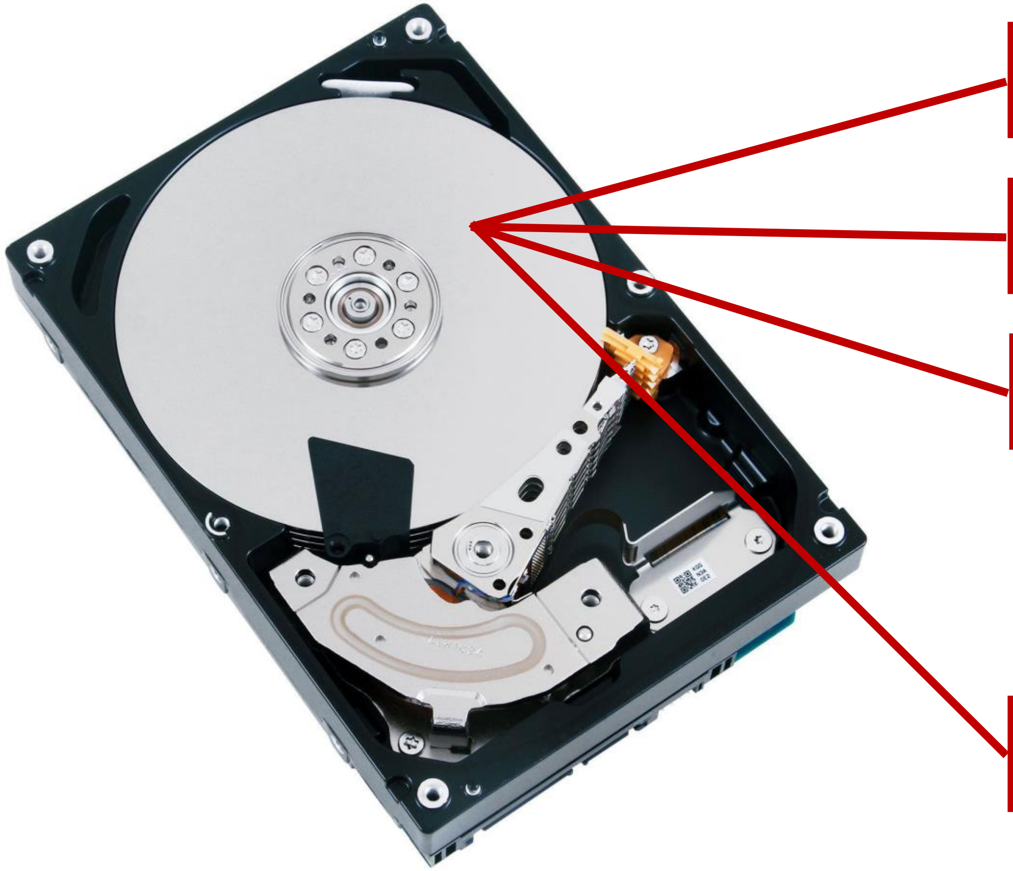
# Memory address

**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.

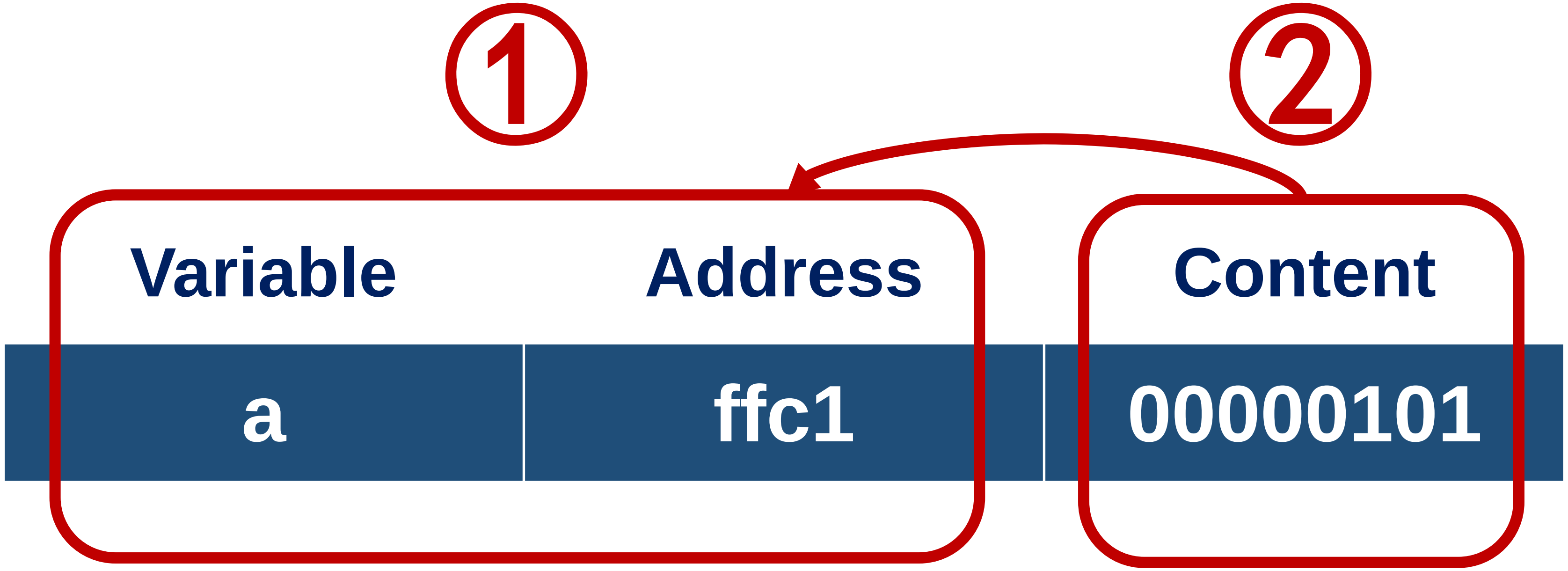
# Memory address



| 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;//declare
int a = 5; ⎨
           ⎩ a = 5;//initialize
```



| Variable | Address | Content |
|---|---|---|
| a | ffc1 | 00000101 |

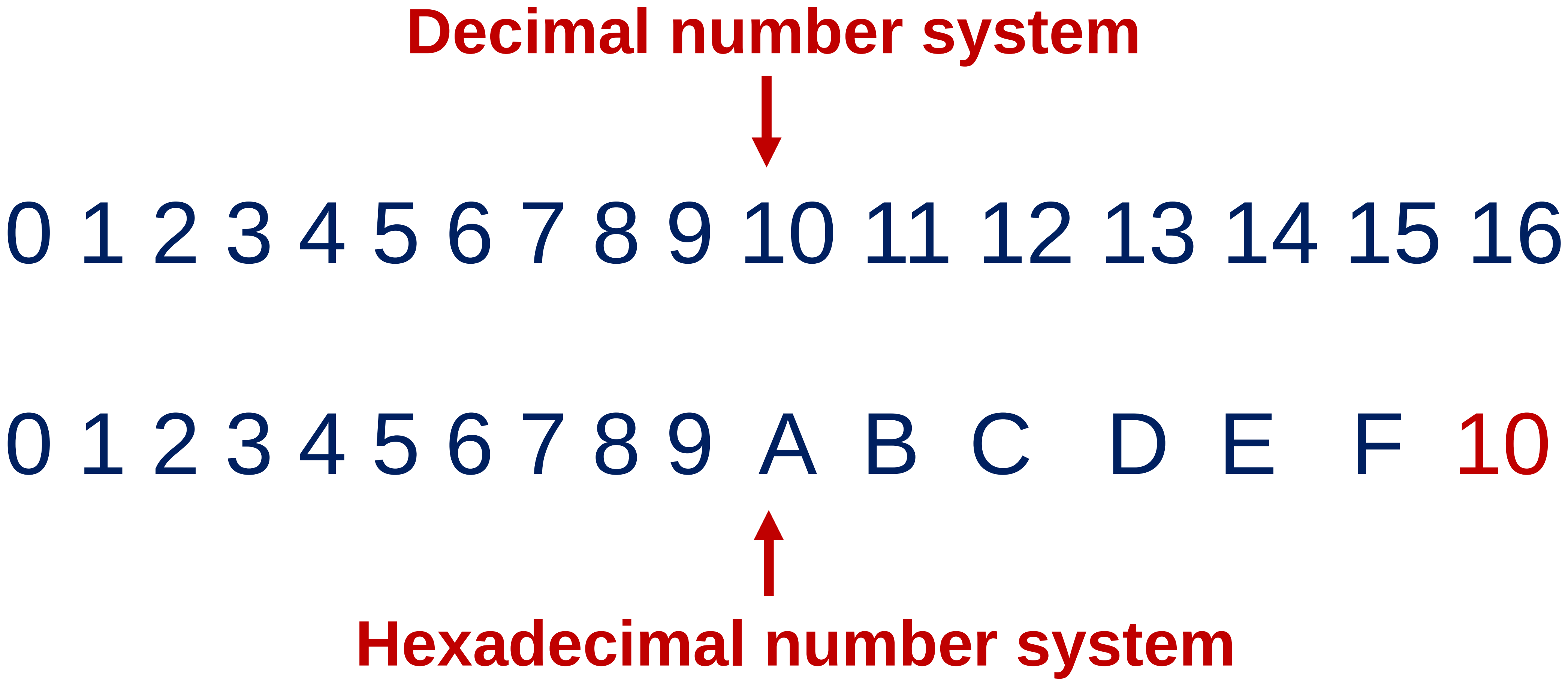
# Memory address

```
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!**

# What is Hexadecimal?



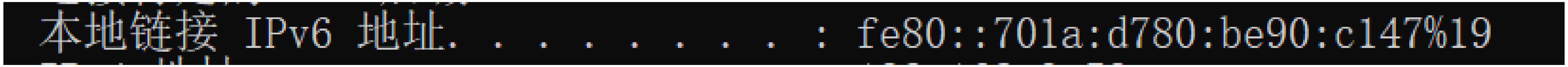
Decimal number system

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

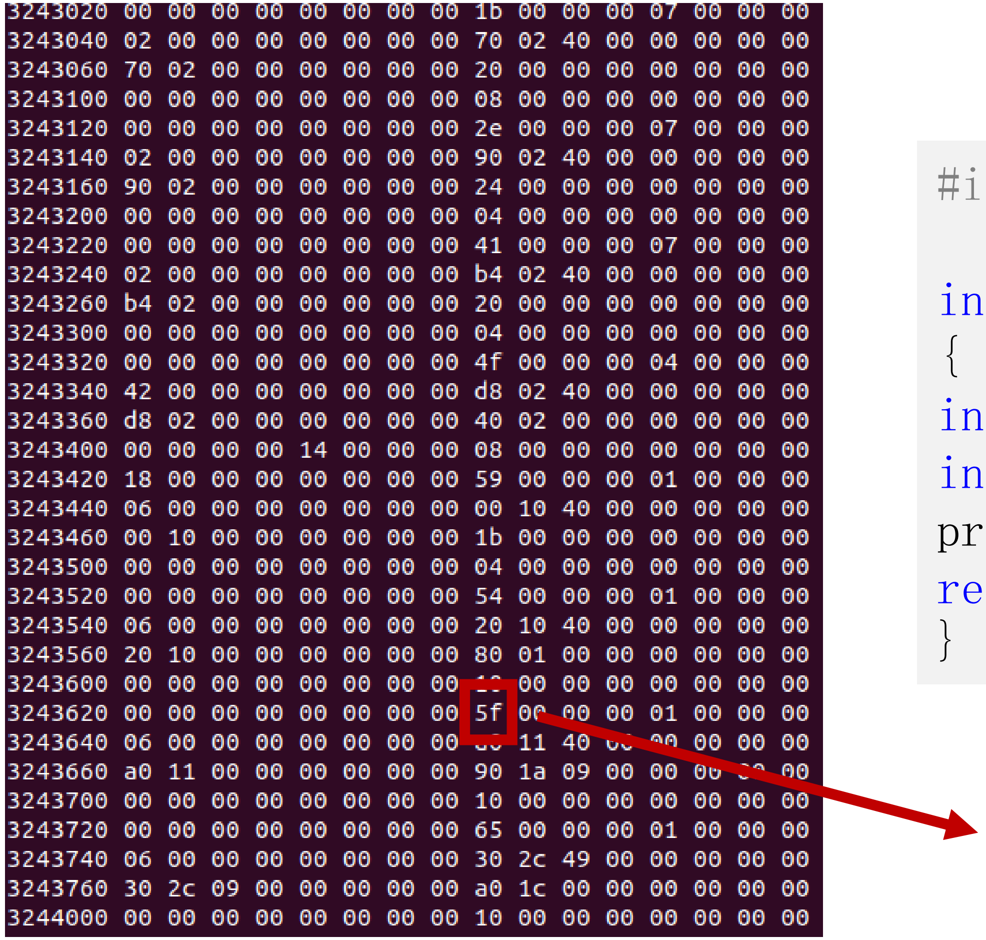
0 1 2 3 4 5 6 7 8 9 A B C D E F 10

Hexadecimal number system

# Hexadecimal is everywhere

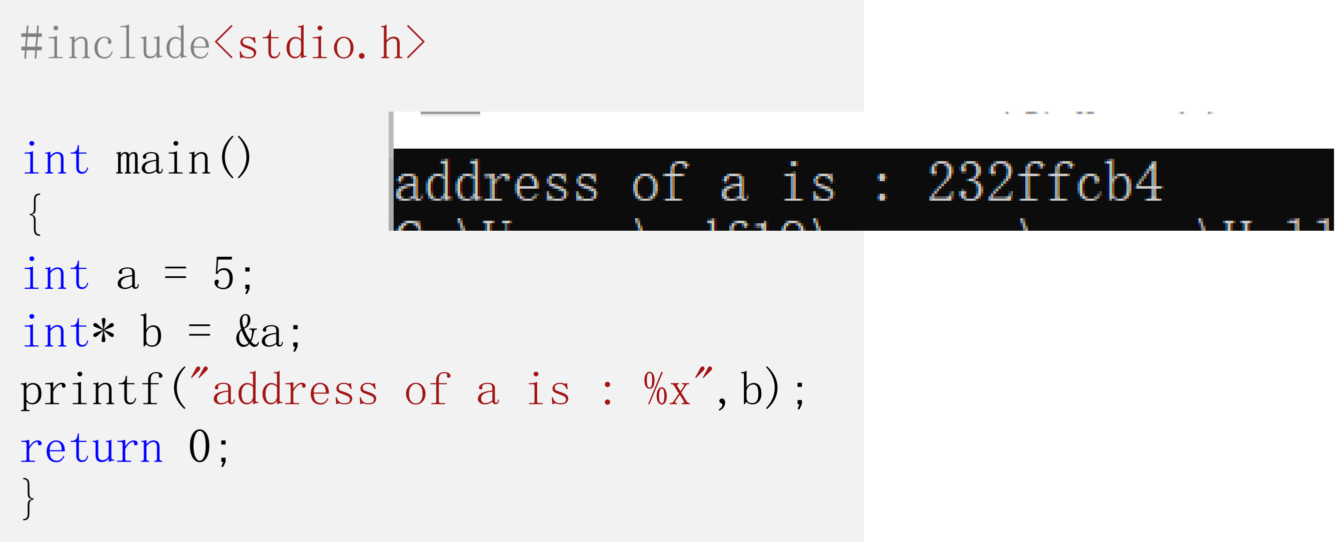


```
本地链接 IPv6 地址. . . . . . . . : fe80::701a:d780:be90:c147%19
```



```
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**



```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
```

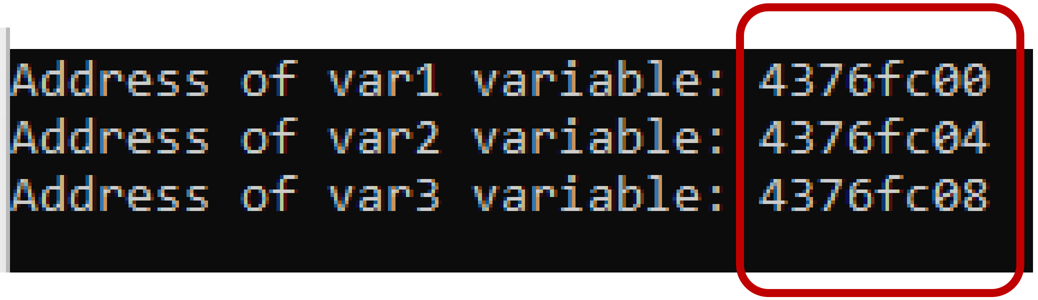
# How to check variable address

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

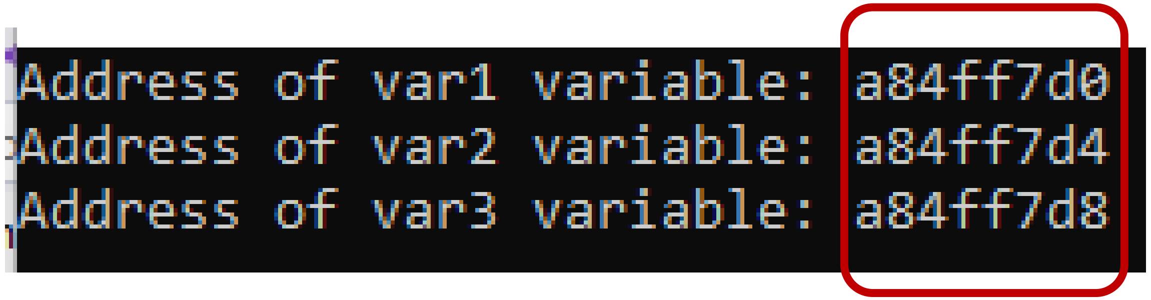
```
#include <stdio.h>

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

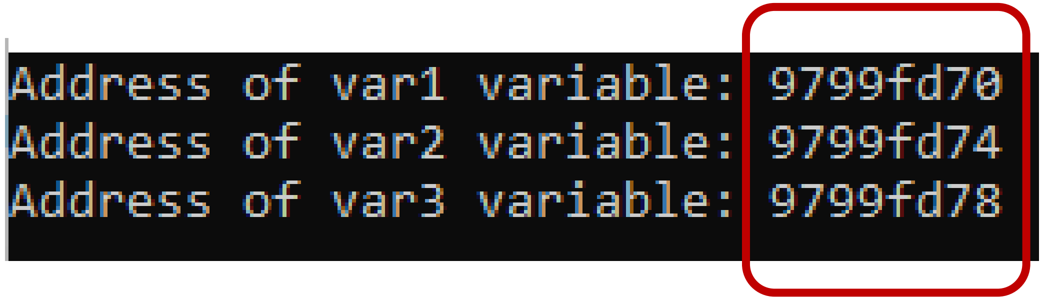
Run multiple times, every time the address is different, but it has orders!



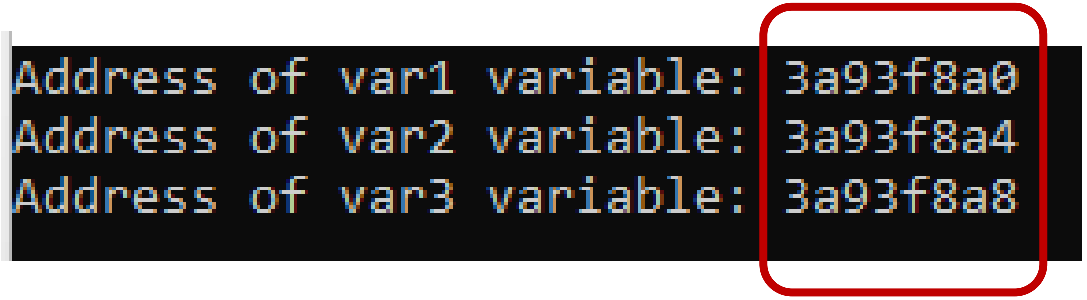
```
Address of var1 variable: 4376fc00
Address of var2 variable: 4376fc04
Address of var3 variable: 4376fc08
```



```
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
```

# Content

1. Memory address
2. Pointer
3. Memory management

# 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

# What is pointer?

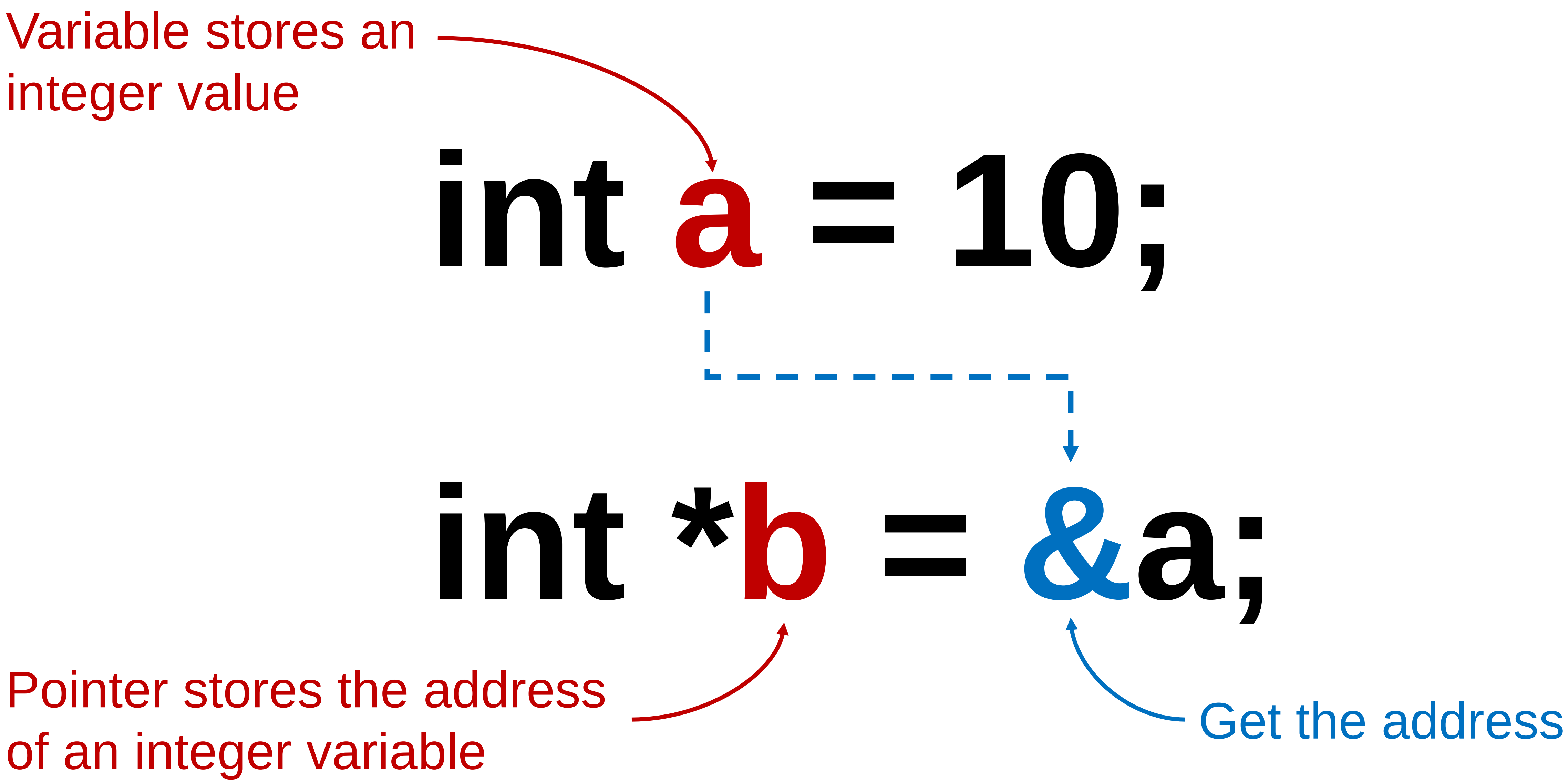
```
int a;
```

- a has type of int
- a stores value

```
int *b;
```

- b has type of int*
- b stores address

# Pointer declaration and definition



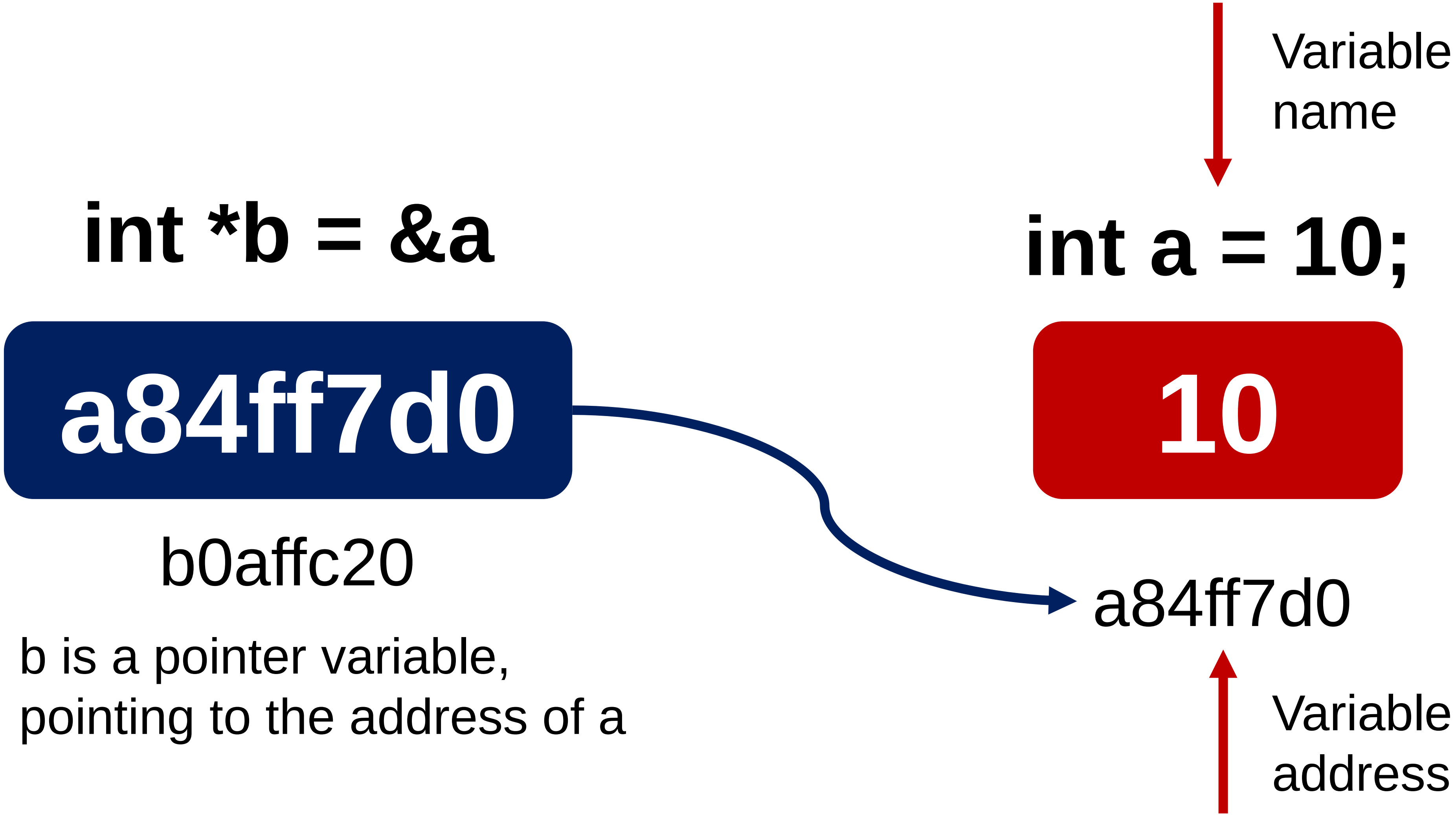
Variable stores an integer value

```
int a = 10;

int *b = &a;
```

Pointer stores the address of an integer variable

Get the address

# Pointer declaration and definition



int *b = &a

a84ff7d0

b0affc20

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

Variable name

int a = 10;

10

a84ff7d0

Variable address

# 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?

# How to interpret pointer?

**int *b**

b has data type int*

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

**int *b**

*b has data type int

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

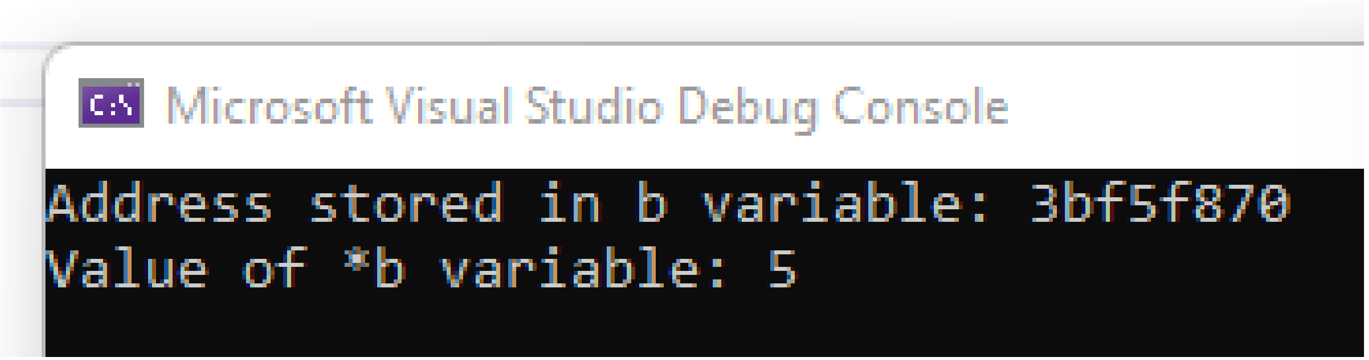
# How to interpret pointer?

```
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;**



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

✗

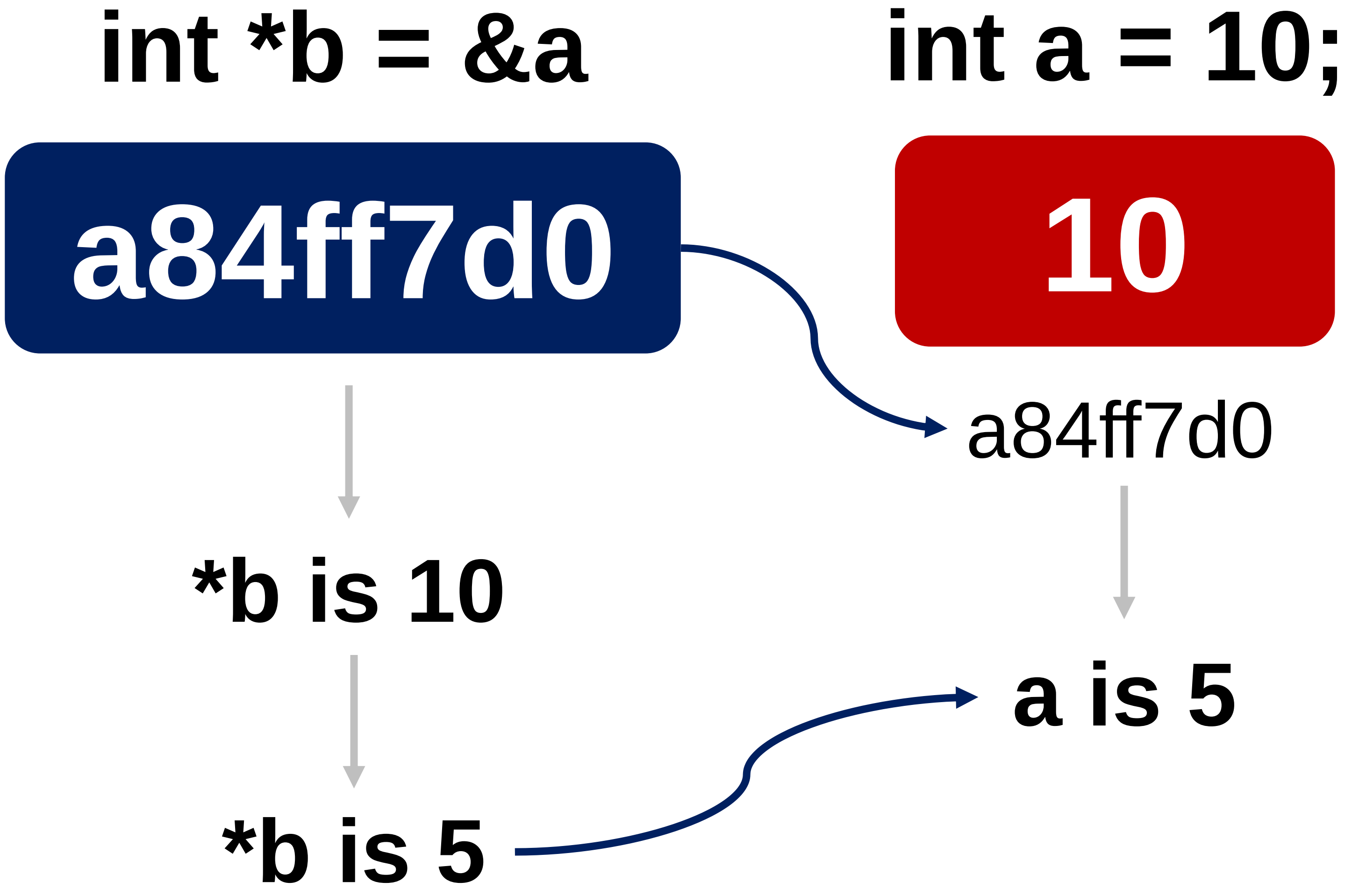
**Pointer must be initialized**

# Pointer allows changing source

```
int a = 10;
a = 5;
```

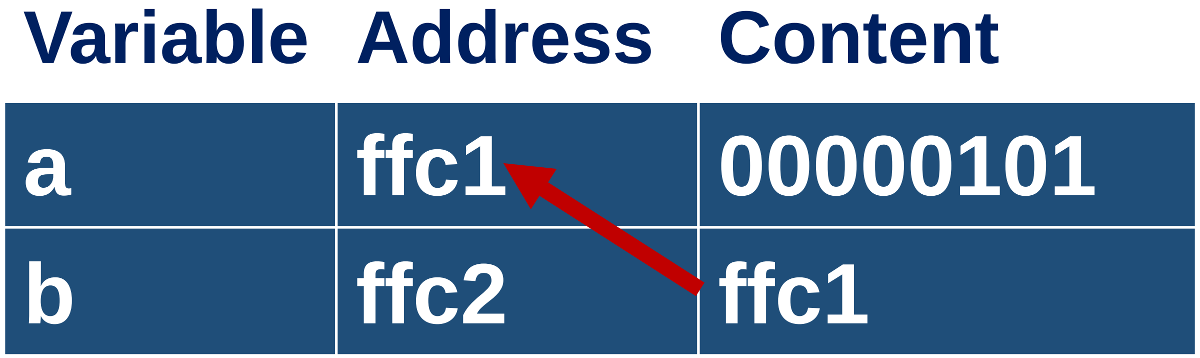
```
int a = 10;
int *b;
b = &a; // int* type
*b = 5; // int type
```

**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!!!**

# 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);
}


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 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);
}

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 |

# 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;
}

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 influence outside

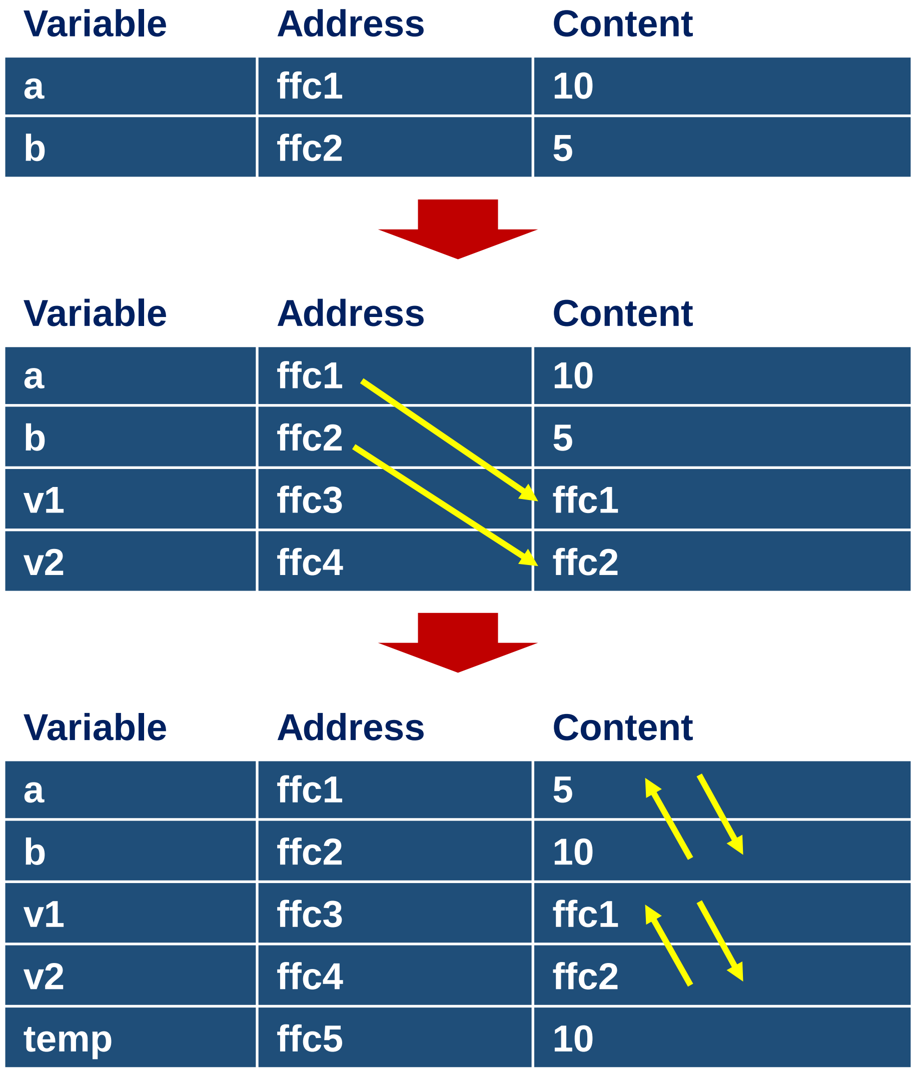
a = 10, b = 5

a = 5, b = 10

# Case of value swap

## Procedure

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

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 |

# 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;
}

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

**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;
}

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

Pass out sub result

Return sum result

# Case of multiple function outputs

## 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;
}
```

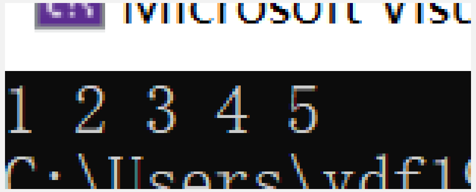
Pass out four results

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

# Function can return pointer

```
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;
}

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

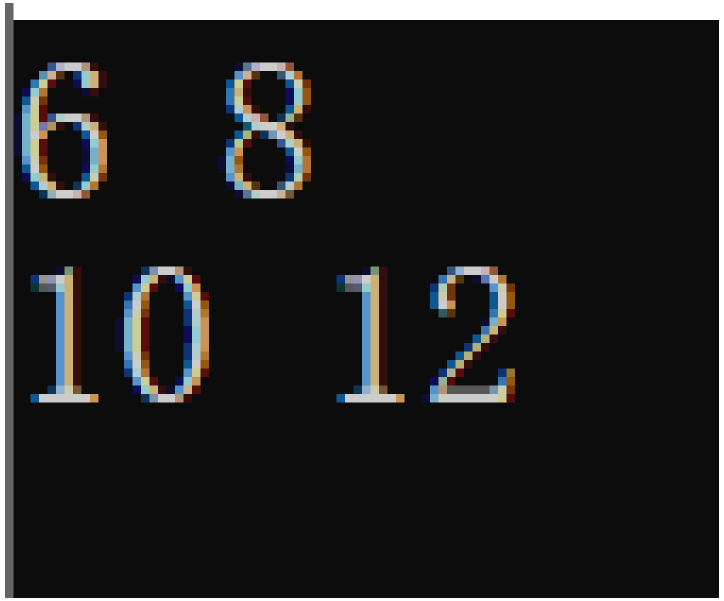


```
Microsoft Visu
1 2 3 4 5
C:\Users\xdf1
```

# Case study: Matrix addition

```
int* add_mats(int* A, int* B, int rows, int cols){
int* C = (int*)malloc(sizeof(int) * rows * cols);
        for (int i = 0; i < rows; i++)
        for (int j = 0; j < cols; j++)
                C[i * cols + j] = B[i * cols + j] +
A[i * cols + j];
        return C;
}
int main(){
        int A[4] = { 1,2,3,4 };
        int B[4] = {5,6,7,8};
        int * C = add_mats(A, B, 2, 2);
        for (int i = 0; i < 2; i++){
                for (int j = 0; j < 2; j++){
                printf("%d ",C[i * 2 + j]);
                }
                printf("\n");
        }
}
```

Case: add 2 matrices

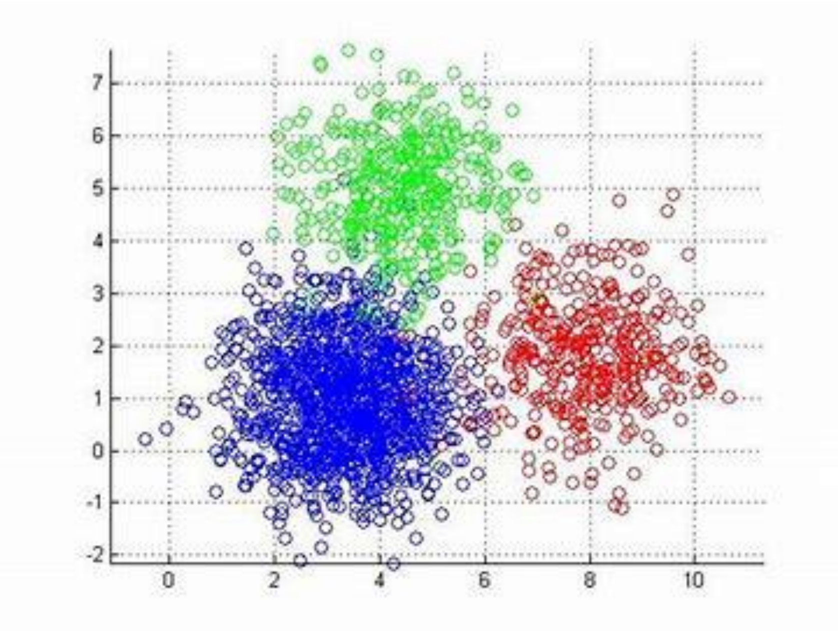


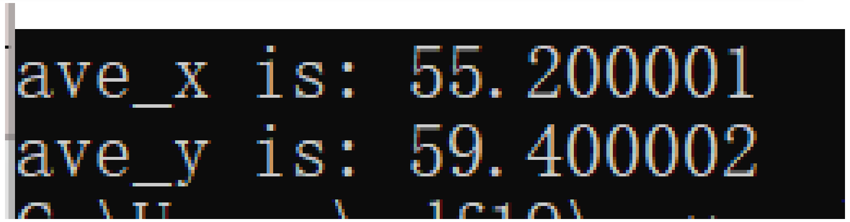
```
6 8
10 12
```

# Case study: calculate coordinate mean

```
void cal_ave_coor(int points[][2],int num, float* 
ave_x, float* ave_y)
{
    int sum_x = 0;
    int sum_y = 0;
    for (int i = 0; i < num; i++)
    {
        sum_x += points[i][0];
        sum_y += points[i][1];
    }
    *ave_x = (float)sum_x / num;
    *ave_y = (float)sum_y / num;
}
main(){
    float ave_x, ave_y;
    int points[5][2] =
    { 23,43,65,67,78,53,74,85,36,49 };
    cal_ave_coor(points, 5, &ave_x, &ave_y);
    printf("ave_x is: %f\nave_y is: %f",ave_x,ave_y);
}
```

Case: calculate mean of 2D points





```
ave_x is: 55.200001
ave_y is: 59.400002
```

# 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

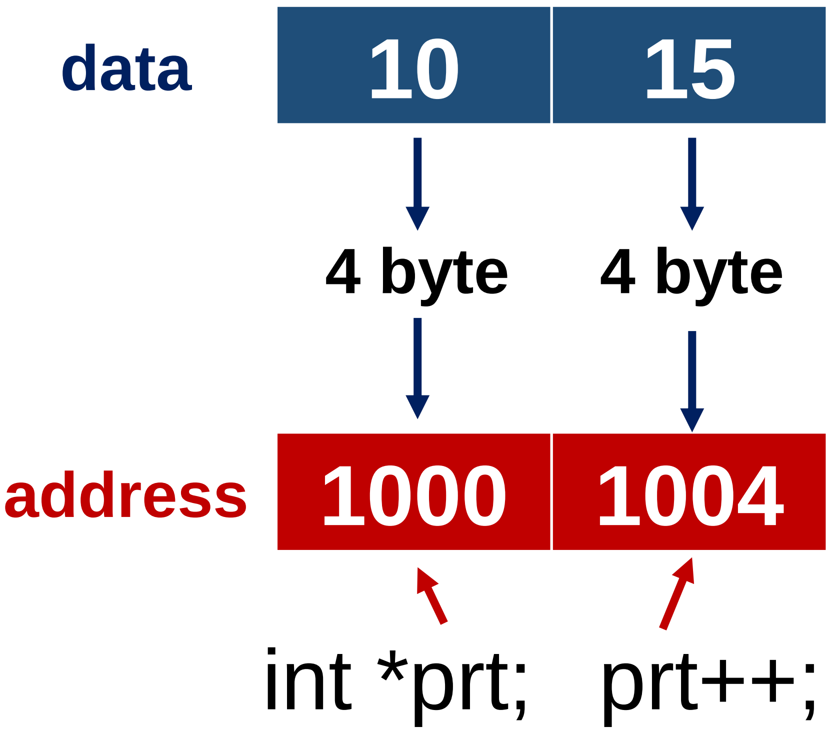
```
int a[3]={1,2,3};

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

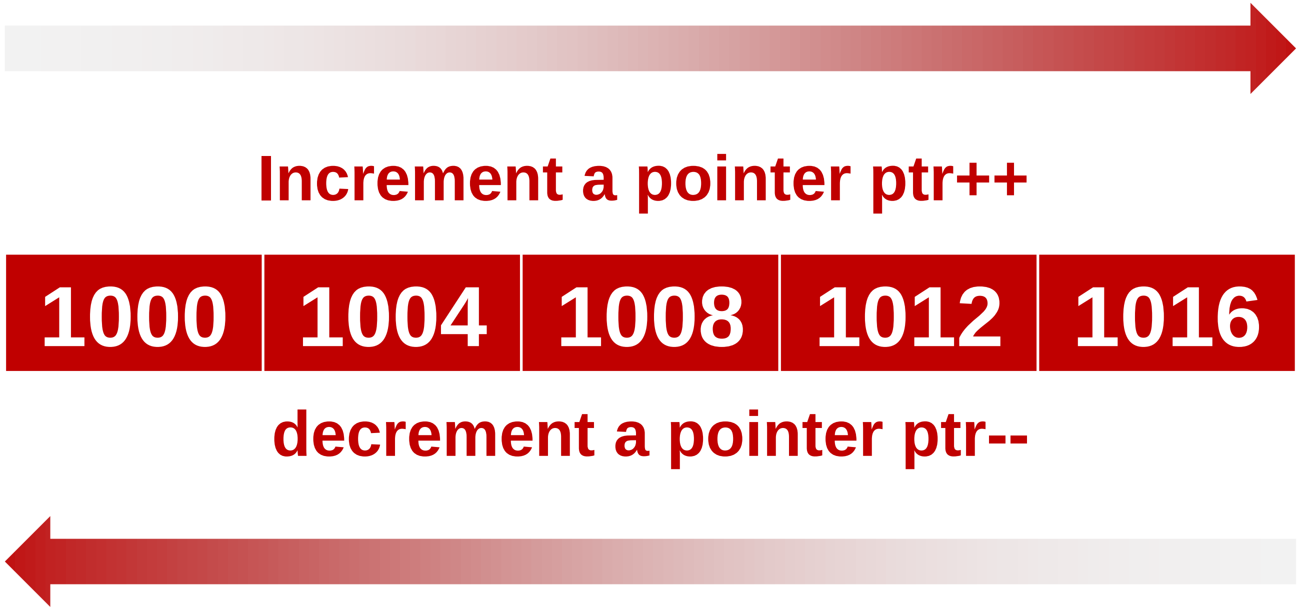
# Pointer points to array

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



| | | |
|---|---|---|
| data | 10 | 15 |
| | 4 byte | 4 byte |
| address | 1000 | 1004 |

int *prt;   prt++;



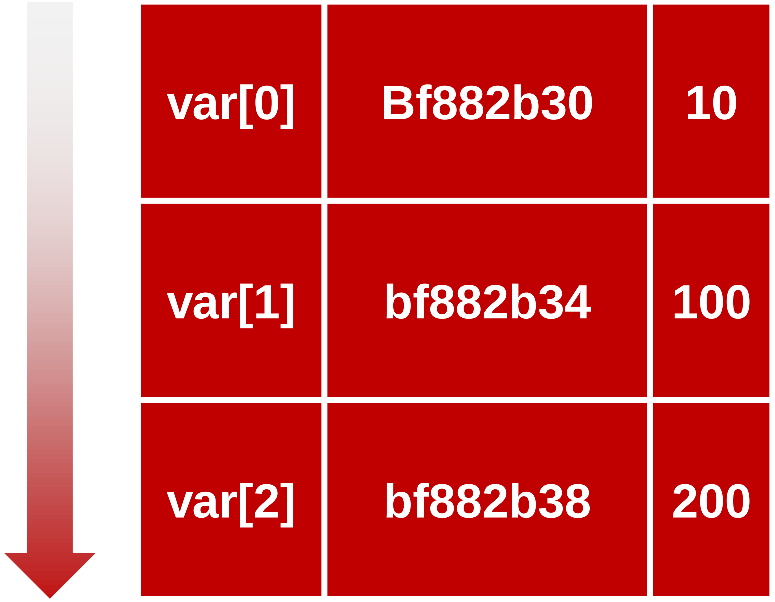
Increment a pointer ptr++

| | | | | |
|---|---|---|---|---|
| 1000 | 1004 | 1008 | 1012 | 1016 |

decrement a pointer ptr--

# Pointer points to array

```
#include <stdio.h>

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

    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 */
    }
}
```

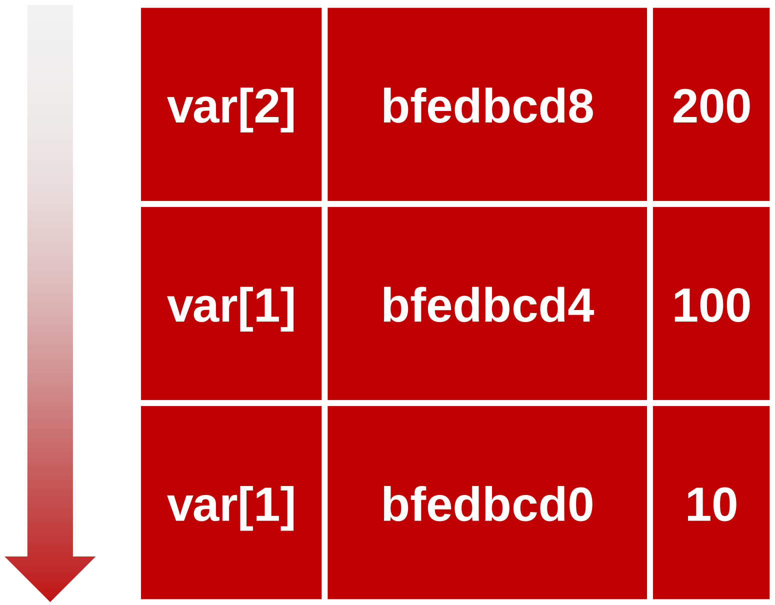


| | | |
|---|---|---|
| var[0] | Bf882b30 | 10 |
| var[1] | bf882b34 | 100 |
| var[2] | bf882b38 | 200 |

**Increments a pointer**

# Pointer points to array

```
#include <stdio.h>

main ()
{
    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 */
    }
}
```



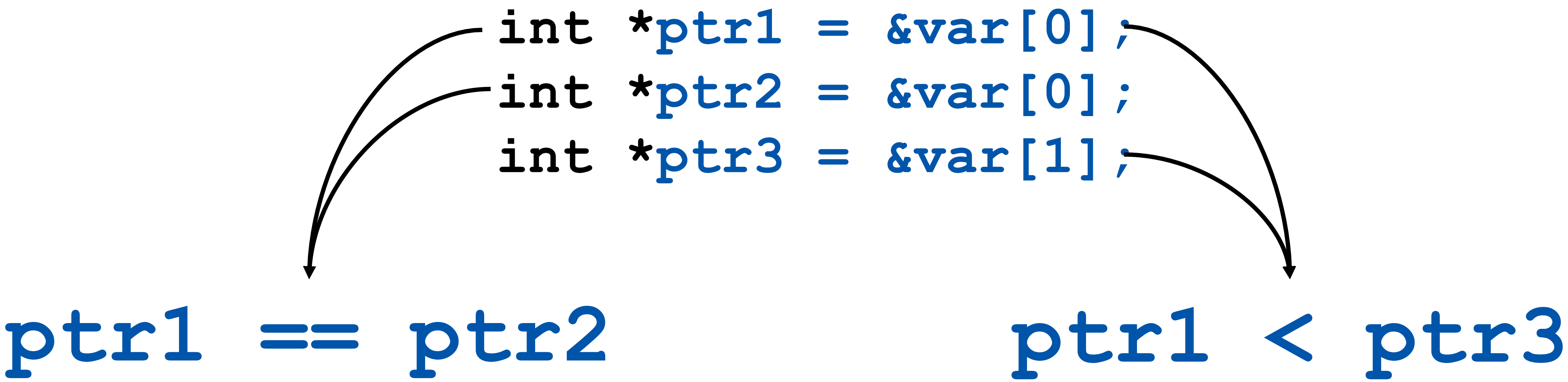
| | | |
|---|---|---|
| var[2] | bfedbcd8 | 200 |
| var[1] | bfedbcd4 | 100 |
| var[1] | bfedbcd0 | 10 |

**Decrements a pointer**

# Pointer points to array

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

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



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

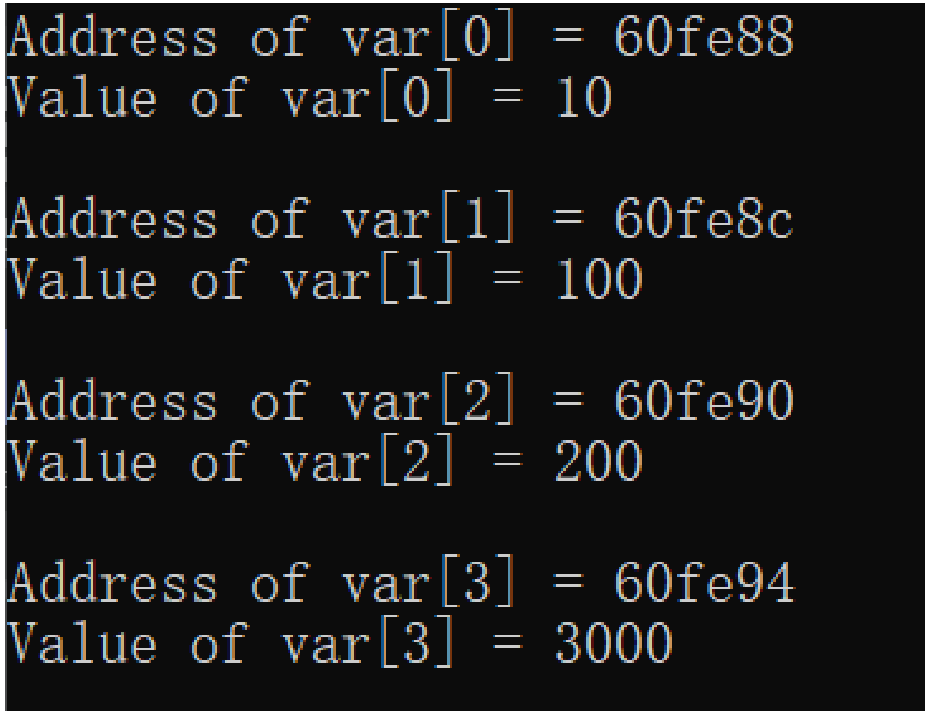
**ptr1 == ptr2**

**ptr1 < ptr3**

# Pointer points to array

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

```c
#include <stdio.h>

int main ()
{
   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;
}
```

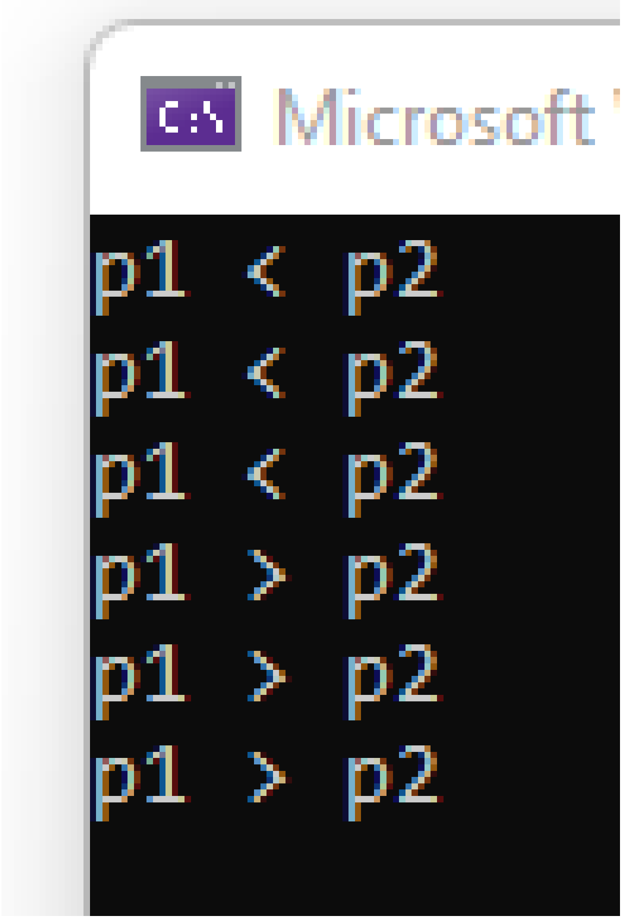


```
Address of var[0] = 60fe88
Value of var[0] = 10

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, c;
    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--;
    }
}
```



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

# Array of pointers

## An array to store pointers

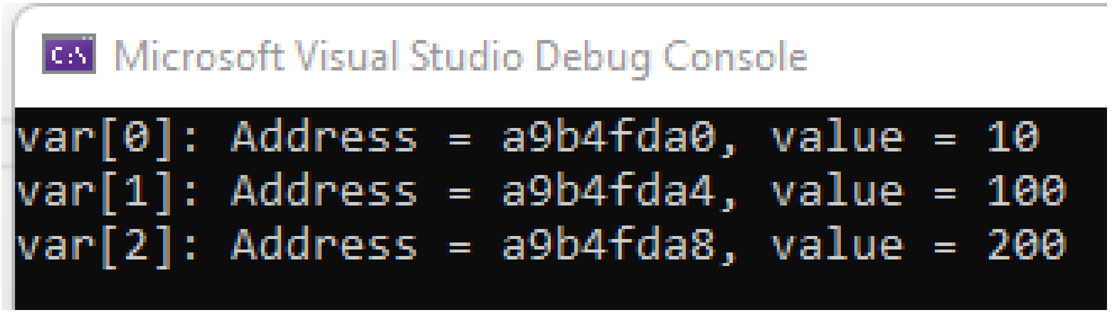
int val[3];

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

int *ptr[3];

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

# Array of pointers



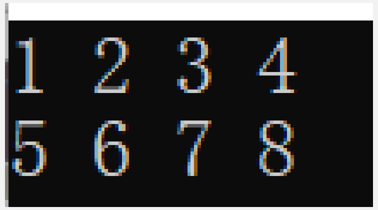
```
var[0]: Address = a9b4fda0, value = 10
var[1]: Address = a9b4fda4, value = 100
var[2]: Address = a9b4fda8, value = 200
```

```
#include <stdio.h>

main ()
{
    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]);
    }
}
```

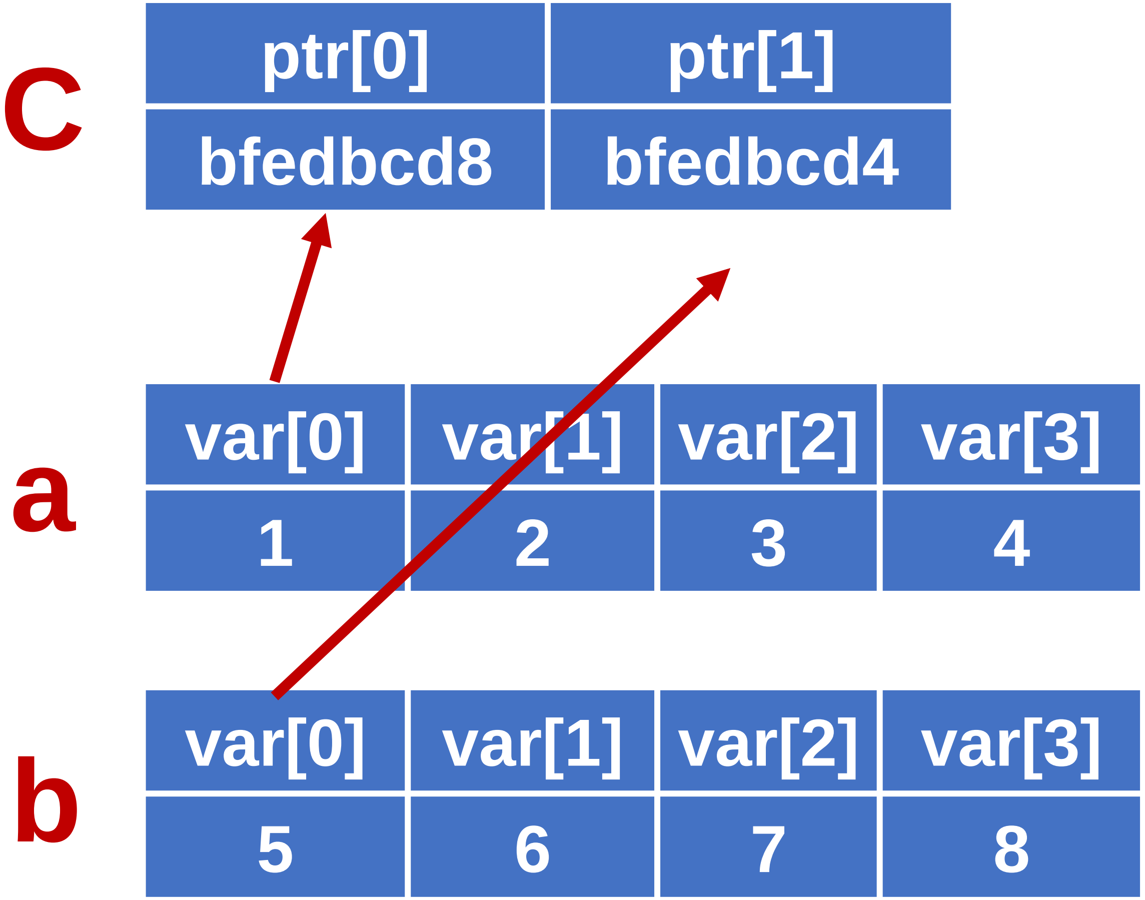
# Array of pointers

**We can use a 1-D array which is full of points of 1-D int arrays to store a 2-D int array**

```c
#include<stdio.h>

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

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
```



c

| ptr[0] | ptr[1] |
|---|---|
| bfedbcd8 | bfedbcd4 |

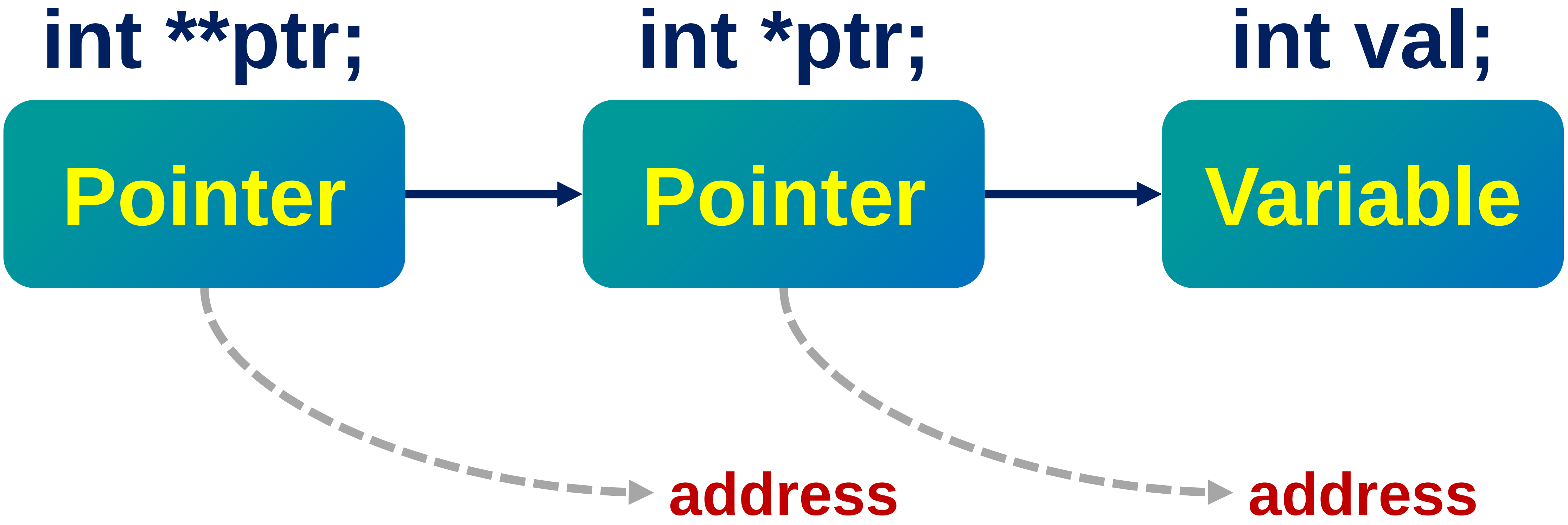
a

| var[0] | var[1] | var[2] | var[3] |
|---|---|---|---|
| 1 | 2 | 3 | 4 |

b

| var[0] | var[1] | var[2] | var[3] |
|---|---|---|---|
| 5 | 6 | 7 | 8 |

# Pointer to pointer



int **ptr;

Pointer

int *ptr;

Pointer

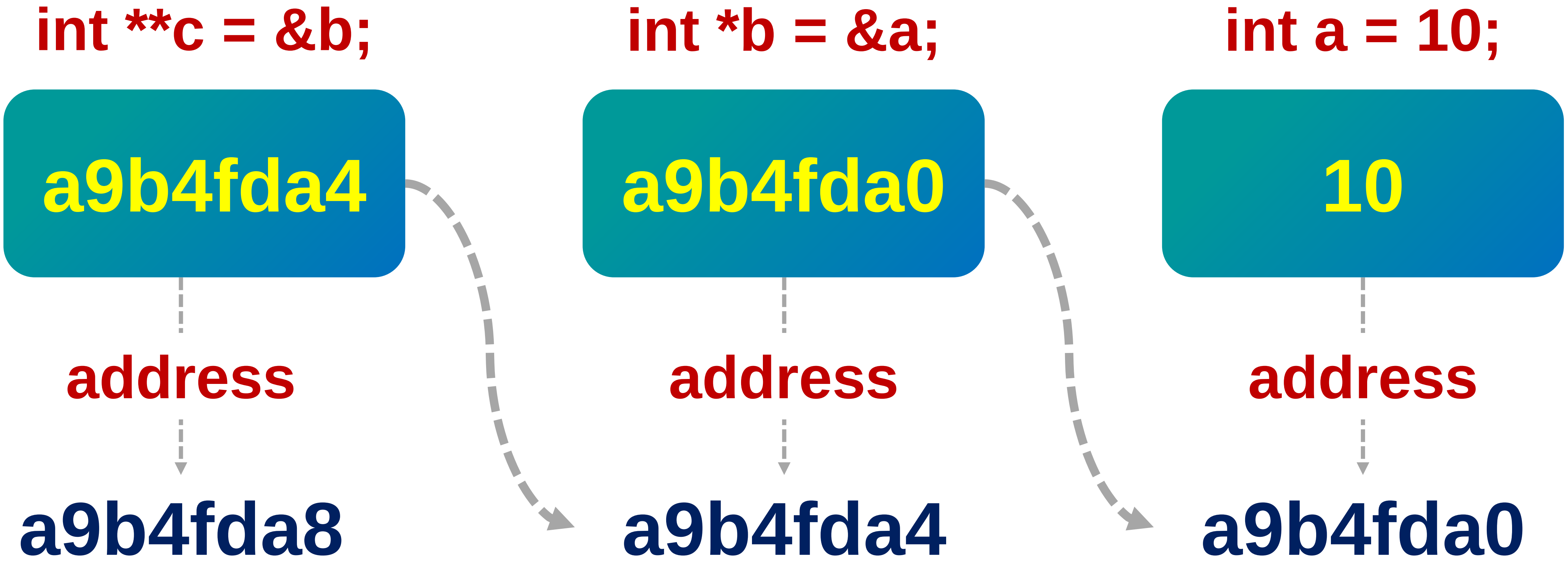
int val;

Variable

address

address

# Pointer to pointer



int **c = &b;

a9b4fda4

address

a9b4fda8

int *b = &a;

a9b4fda0

address

a9b4fda4

int a = 10;

10

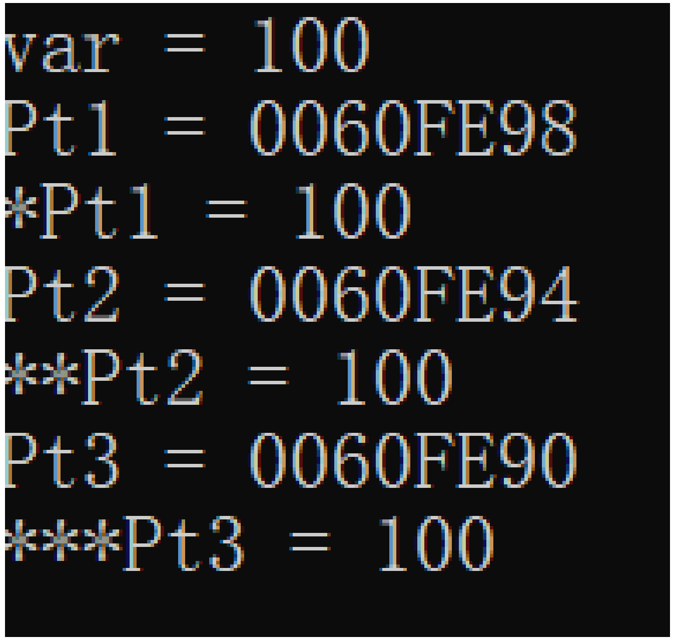
address

a9b4fda0

# 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. **Memory management**

# 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 |

# Memory management

You must use this library to use memory management function

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

# 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)

# calloc() function

Fixed array size, fixed memory

```
int name[100];
```

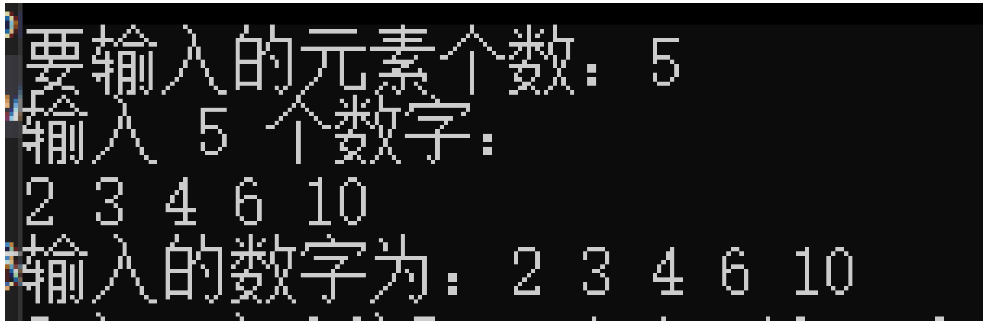
```
int *name;
name = (int*)calloc(100, sizeof(int));
```

How many bytes in total???

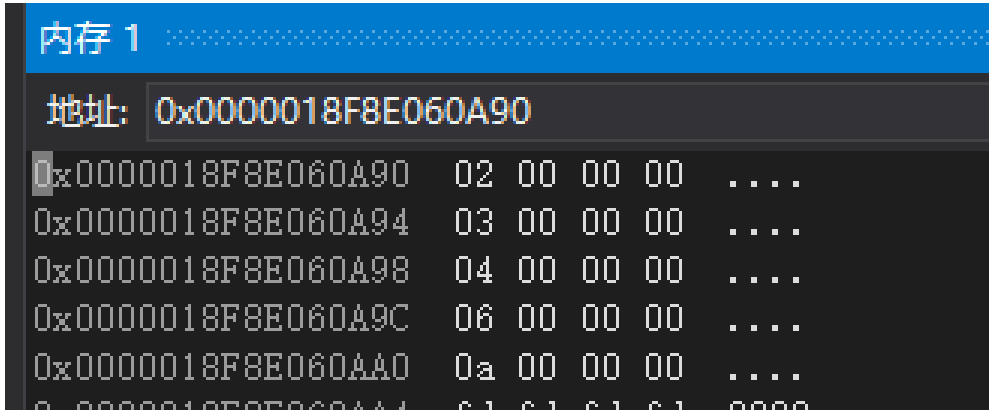
# calloc() function

## How to use calloc() to allocate memory for a pointer?

```
#include <stdio.h>
#include <stdlib.h>
main()
{
        int n;
        printf("要输入的元素个数： ");
        scanf("%d", &n);
        int* test_array = (int*)calloc(n, sizeof(int));
        printf("输入 %d 个数字： \n", n);
        for (int i = 0; i < n; i++){
                scanf("%d", &test_array[i]);
        }
        printf("输入的数字为： ");
        for (int i = 0; i < n; i++) {
                printf("%d ", test_array[i]);
        }
        free(test_array);
}
```



```
要输入的元素个数: 5
输入 5 个数字:
2 3 4 6 10
输入的数字为: 2 3 4 6 10
```



地址: 0x0000018F8E060A90

```
0x0000018F8E060A90  02 00 00 00  ....
0x0000018F8E060A94  03 00 00 00  ....
0x0000018F8E060A98  04 00 00 00  ....
0x0000018F8E060A9C  06 00 00 00  ....
0x0000018F8E060AA0  0a 00 00 00  ....
```

# 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)

# malloc() function

Fixed array size, fixed memory

```
float name[30];
```

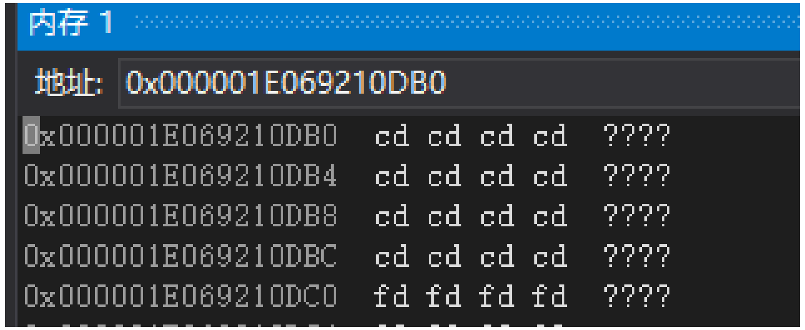
```
float *name;
name = (float*)malloc(30*sizeof(float));
```

How many bytes in total?

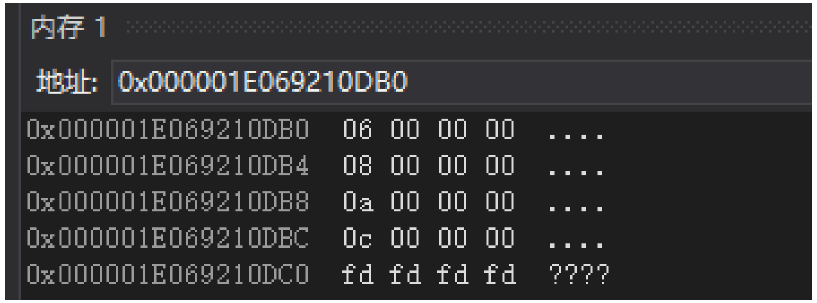
# malloc() function

## How to use malloc() to allocate memory for a pointer?

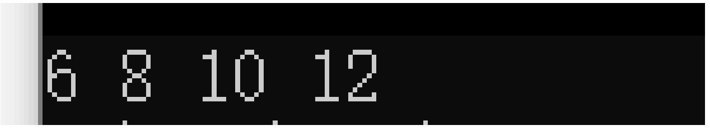
```
int* add_mats(int A[], int B[],int n)
{
    int* C = (int*)malloc(sizeof(int) * n);
    for (int i = 0; i < n; i++)
        C[i] = B[i] + A[i];
    return C;
}

main(void)
{
   int A[4] = { 1,2,3,4 };
   int B[4] = {5,6,7,8};
   int *C = add_mats(A, B, 4);
   for (int i = 0; i < 4; i++){
        printf("%d ",C[i]);
   }
}
```



```
地址: 0x000001E069210DB0
0x000001E069210DB0  cd cd cd cd  ????
0x000001E069210DB4  cd cd cd cd  ????
0x000001E069210DB8  cd cd cd cd  ????
0x000001E069210DBC  cd cd cd cd  ????
0x000001E069210DC0  fd fd fd fd  ????
```



```
地址: 0x000001E069210DB0
0x000001E069210DB0  06 00 00 00  ....
0x000001E069210DB4  08 00 00 00  ....
0x000001E069210DB8  0a 00 00 00  ....
0x000001E069210DBC  0c 00 00 00  ....
0x000001E069210DC0  fd fd fd fd  ????
```



```
6 8 10 12
```

# calloc() & malloc()

```
char *name;

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

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

# calloc() & malloc()

## calloc()

contiguous/连续的
allocation

↓

allocates memory and
initializes all bits to zero

## malloc()

memory
allocation

↓

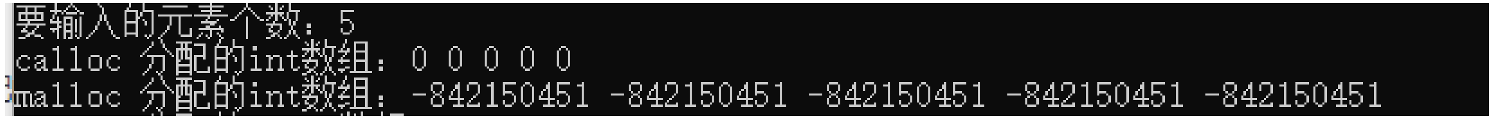
allocates memory and leaves
the memory uninitialized

# Comparison on malloc() & calloc()

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
    int n;
    printf("要输入的元素个数：");
    scanf_s("%d", &n);
    int *test_array = (int*)calloc(n, sizeof(int));
    int *test_array1 = (int*)malloc(sizeof(int)*n);
    printf("calloc 分配的int数组：");
    for (int i = 0; i < n; i++) {
        printf("%d ", test_array[i]);
    }
    printf("\nmalloc 分配的int数组：");
    for (int i = 0; i < n; i++) {
        printf("%d ", test_array1[i]);
    }
}
```

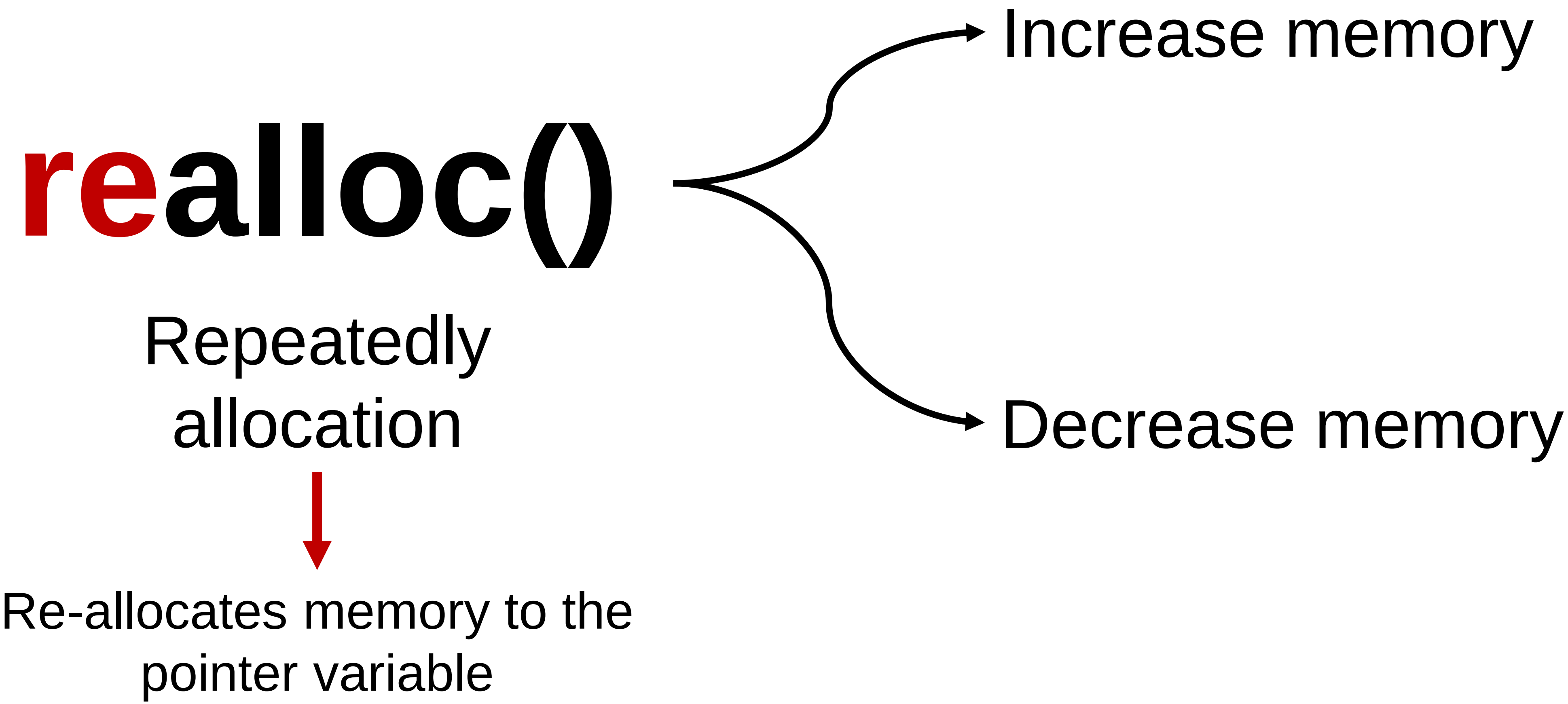
**calloc()：The space is initialized to zero.**

**malloc()：The space is randomly initialized.**



```
要输入的元素个数：5
calloc 分配的int数组：0 0 0 0 0
malloc 分配的int数组：-842150451 -842150451 -842150451 -842150451 -842150451
```

# 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)

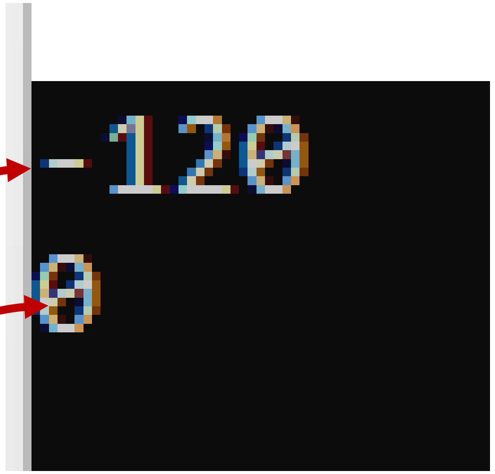
```
char *name;
name = (char*)malloc(200*sizeof(char));


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

Resize the merry at address of name (100 bytes)

# realloc() function

## Why using realloc()?

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

int main()
{
    int *ptr;
    ptr = malloc(5 * sizeof(int));
    ptr[3] = -120;

    ptr = realloc(ptr, 10 * sizeof(int));
    printf("%d", ptr[3]);

    ptr = malloc(ptr, 5 * sizeof(int));
    printf("%d", ptr[3]);

    return 0;
}
```



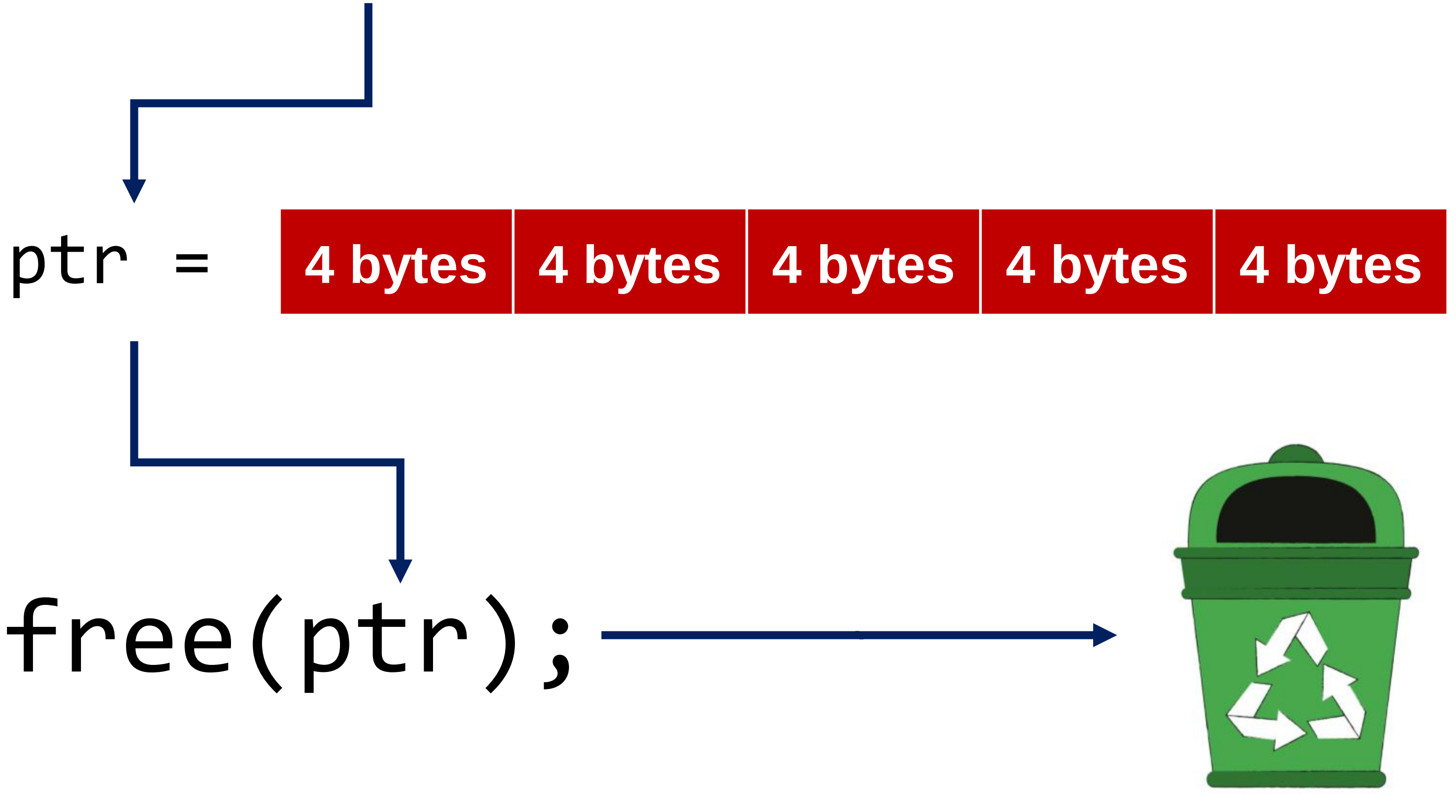
```
-120
0
```

**If using malloc, the value will be erased!**

# free() function

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

4 bytes



ptr =

| 4 bytes | 4 bytes | 4 bytes | 4 bytes | 4 bytes |
|---|---|---|---|---|

```
free(ptr);
```

# Summary

1. **Memory address**
2. **Pointer**
3. **Pointer operations**

# 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 can be modified, typically useful for **functions** in transferring values.
- Pointer has **arithmetic and logical operations** (++, --, ==, >, <) on manipulating the memory address.
- We can **manage the memory** using C provided functions in **stdlib.h**.
- Time to use the pointer!!!

# Assignment

1. Write a function which has three arguments . The function can exchange the first 2 arguments and make the third argument equal to the sum of the first two arguments

a) Call this function in main , a = 5.5, b = 10.1 , print the value of a, b ,c in main

2. Write a function that put elements in the array in the reverse order and return the pointer of the array (e.g. 1,2,3,4,5,6 -> 6,5,4,3,2,1)

a) You are not allowed to use “malloc()” to create a new memory space
b) Use int a[8] = {0,10,20,30,40,50,60,70} as the argument of the function and print the array in main
c) Hint: use pointer increment (p++) or decrement (p--)

# Assignment

3. You have used “strlen()” before. This function can return the length of a string. Now write a “strlen()” function by yourself

a) A char occupies one byte, you can access each char in the string using pointer +1
b) The last element of a string is ‘\0’, you can use this this property to know if the element is the end of the string
c) Use pointer to finish this assignment
d) Use “IloveCHINA” as the argument of the function
e) Hint: use pointer increment (p++) or decrement (p--)

4. Write a strcat() function yourself to concatenate two strings

a) Use “Ilove” “CHINA” as the argument of the function and print the concatenated string in main
b) Hint: use relloc()

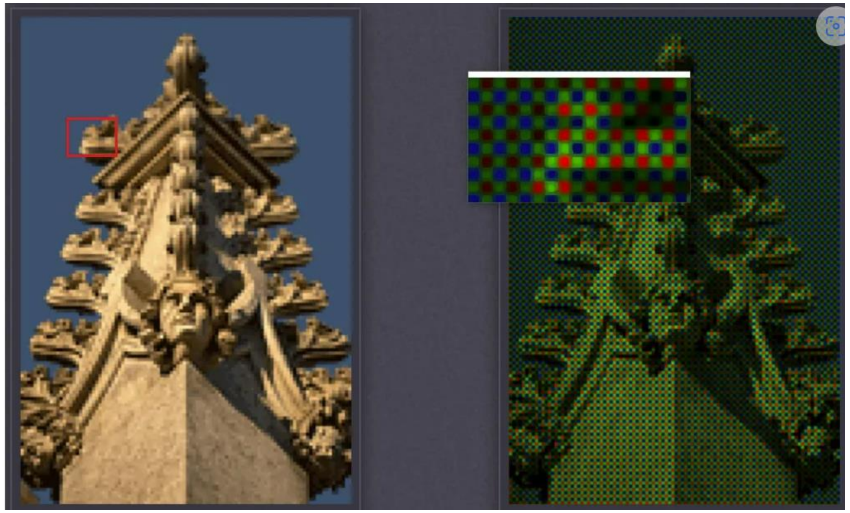
# Assignment

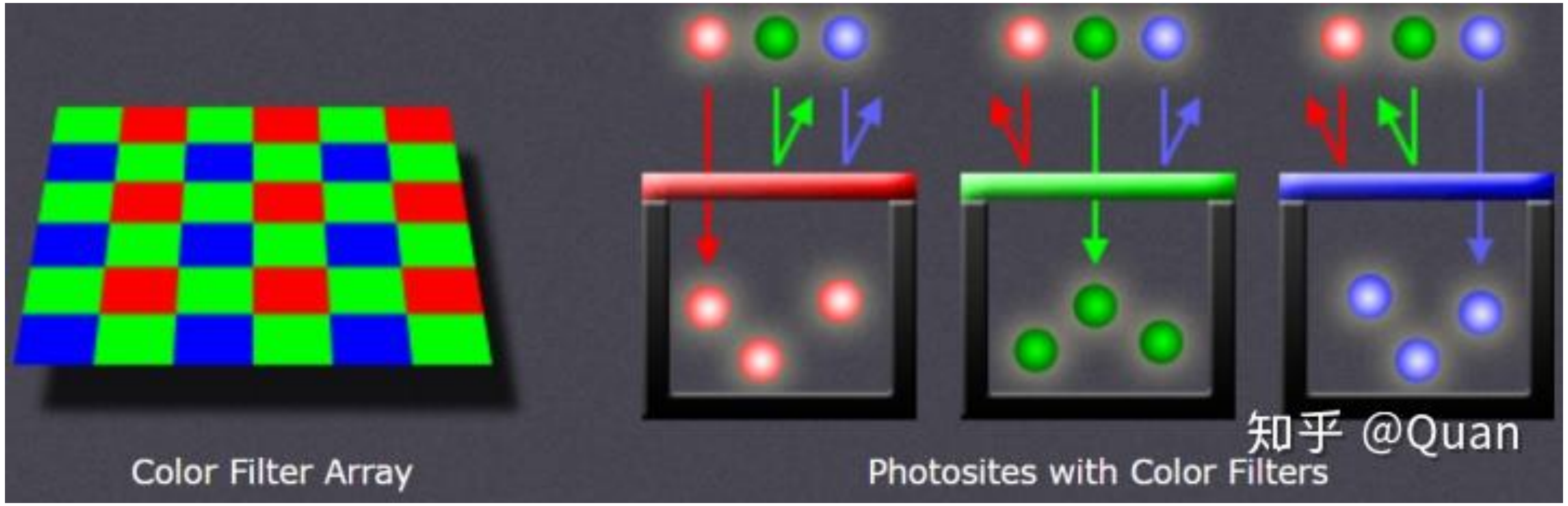
5. Write a function that input the nature number as parameter and returns Chinese phonetic alphabet of the number (e.g. input 1234, return yi er san si)

a) The Chinese phonetic alphabet of the number is a string ,so the function must return char*
b) A string is a 1-D array, you can use 2-D array to store the Chinese phonetic alphabet of 0-9 (create a dictionary)
c) Call this function in main and print the Chinese phonetic alphabet of the number in main, you are not allowed to print in the function
d) You are not allowed to use if/else or switch/case
e) The nature number used as input is 12345
f) Hint: use malloc() and free(), use exercise 4 as part of program

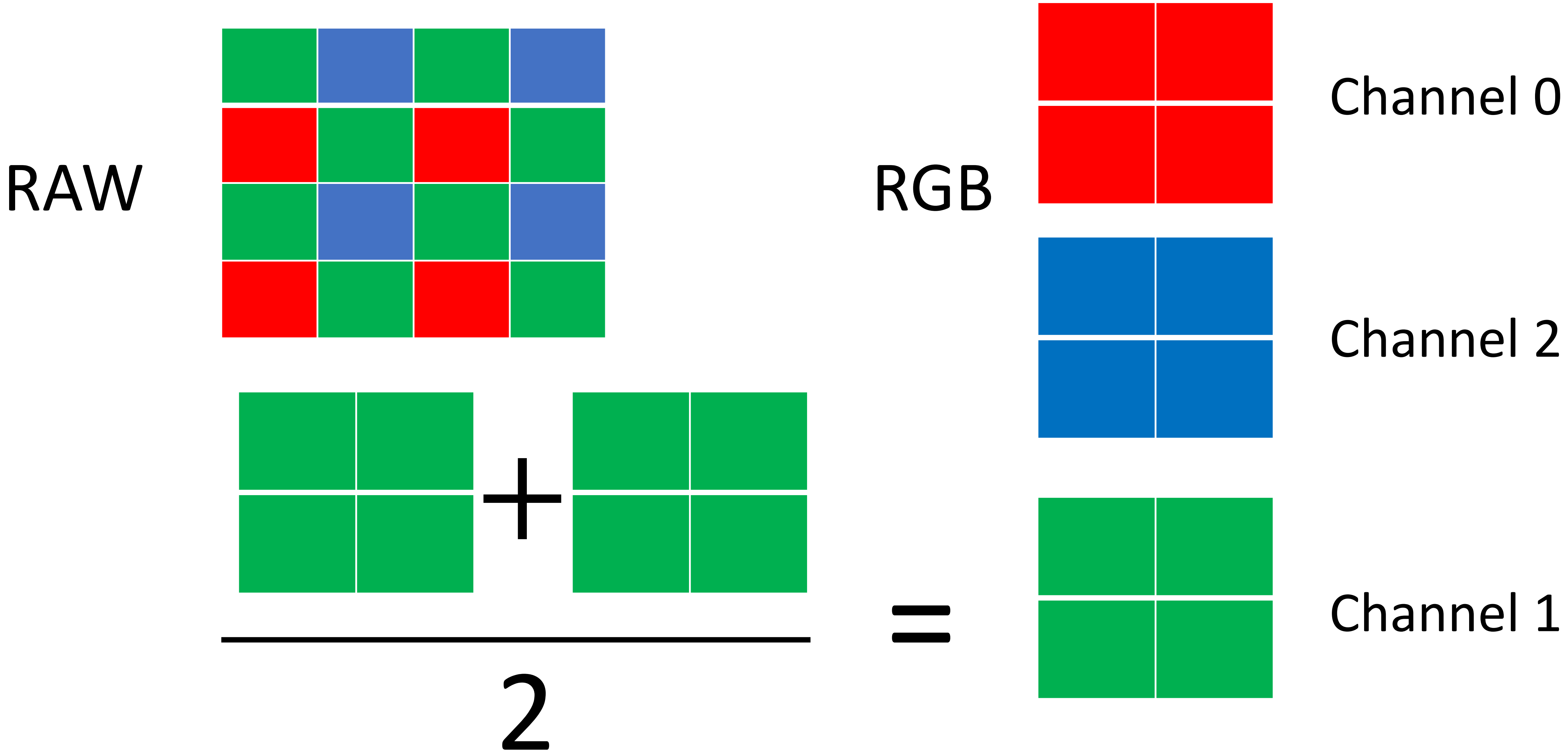
# Assignment

6. (**bonus**) RAW data is the original signal you get from the sensor, however the algorithms usually use RGB data as input ,so you need to convert RAW data to RGB data.  RAW data is a 2-D array, RGB data is a 3-D array which has less rows and cols but has more dimensions(channels) than RAW data . The conversion from RAW data to RGB data is called debayer.





# Assignment



# Assignment

Write a function to convert RAW data (2-D matrix) to RGB data (3-D matrix)

a) Red is the first channel of RGB, green is the second channel of RGB and blue is the third channel of RGB
b) The data type of each element in RAW and RGB is unsigned char
c) You need to free the memory if you have used malloc()
d) You can use 1-D array to represent 2-D matrix or 3-D matrix
e) Use the following 2-D matrix (RAW data) as the argument of the function and print each channel of the 3-D matrix (RGB data)

$$\begin{matrix} 23 & 45 & 37 & 78 \\ 6 & 29 & 23 & 57 \\ 67 & 39 & 47 & 76 \\ 0 & 78 & 39 & 56 \end{matrix}$$

# Assignment