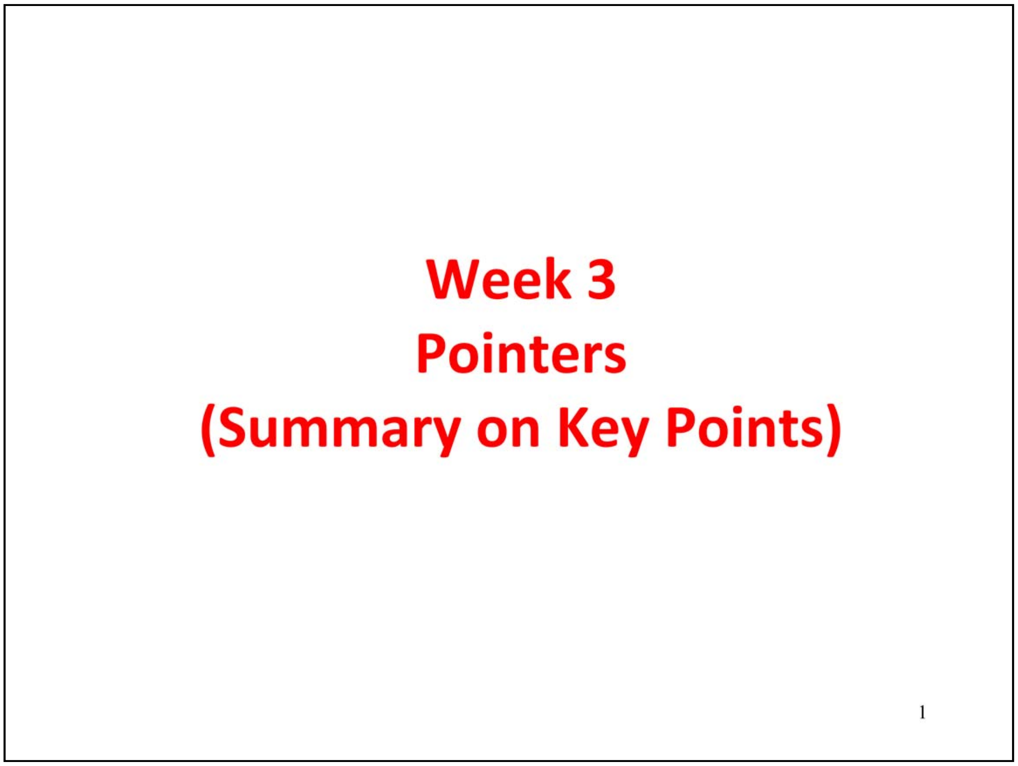
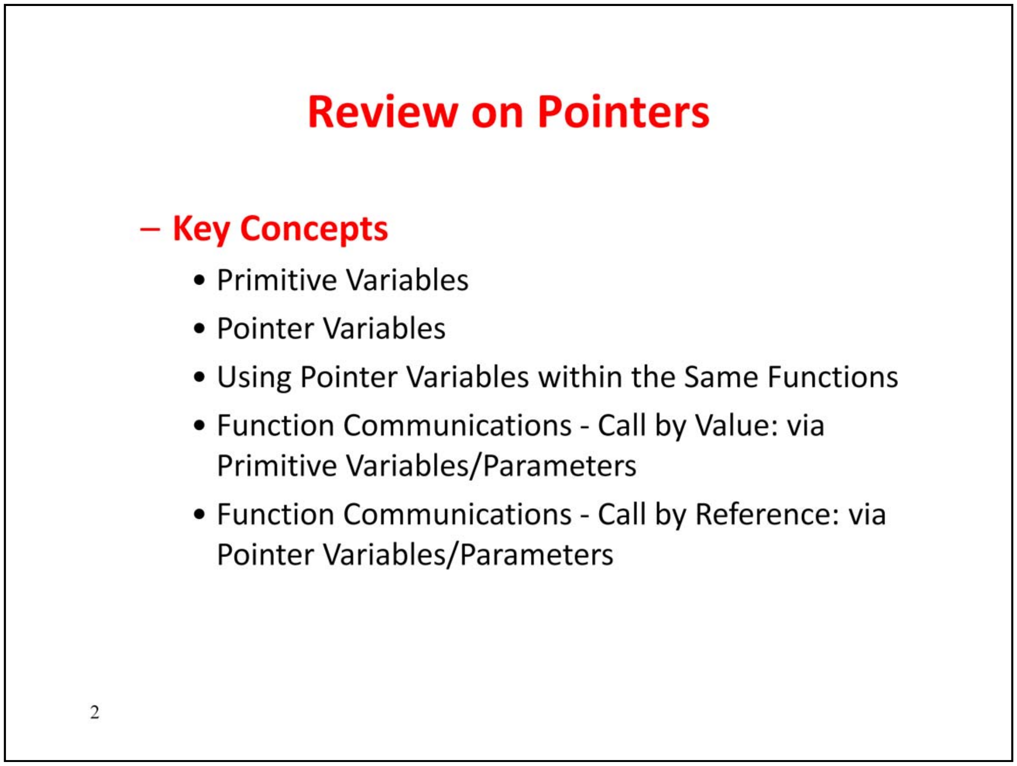


# Week 3
# Pointers
# (Summary on Key Points)

1

**Why Learning Pointers**

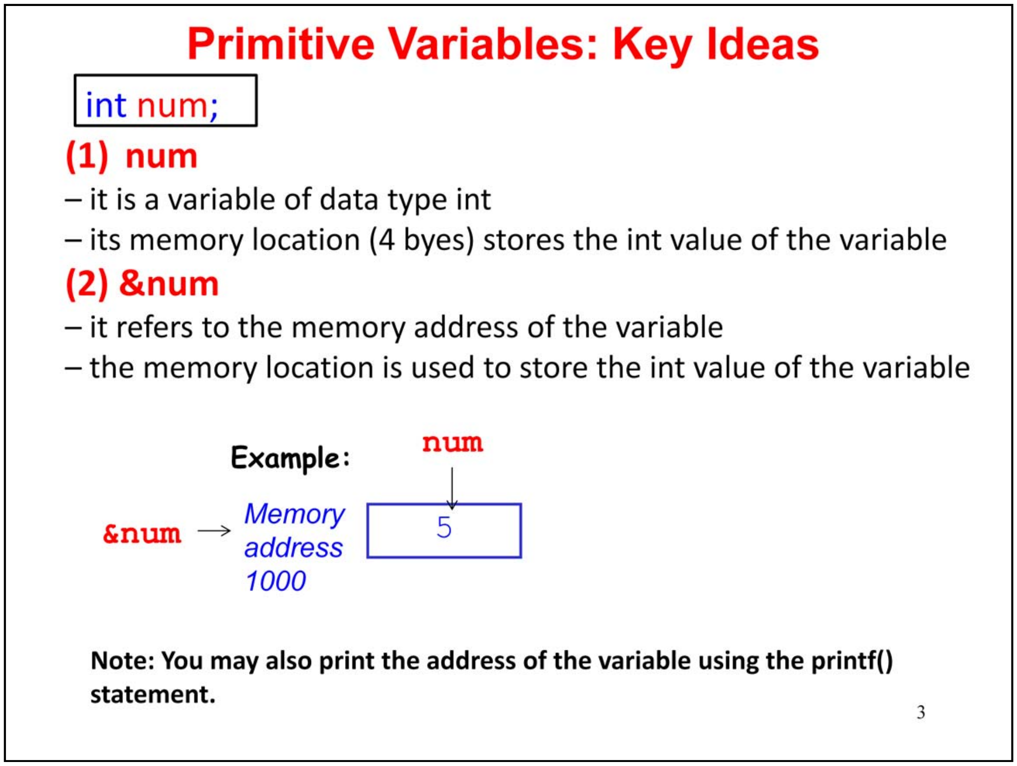
1. Pointer is a very powerful tool for the design of C programs. A pointer is a variable that holds the value of the address or memory location of another data object.
2. In C, pointers can be used in many ways. This includes the passing of variable’s address to functions to support call by reference, and the use of pointers for the processing of arrays and strings.
3. In this lecture, we discuss the concepts of pointers including address operator, pointer variables and call by reference.



## Review on Pointers

– **Key Concepts**

- Primitive Variables
- Pointer Variables
- Using Pointer Variables within the Same Functions
- Function Communications - Call by Value: via Primitive Variables/Parameters
- Function Communications - Call by Reference: via Pointer Variables/Parameters



## Primitive Variables: Key Ideas

```
int num;
```

**(1) num**

– it is a variable of data type int

– its memory location (4 byes) stores the int value of the variable

**(2) &num**

– it refers to the memory address of the variable

– the memory location is used to store the int value of the variable

**Example:**

num

&num → *Memory address 1000*

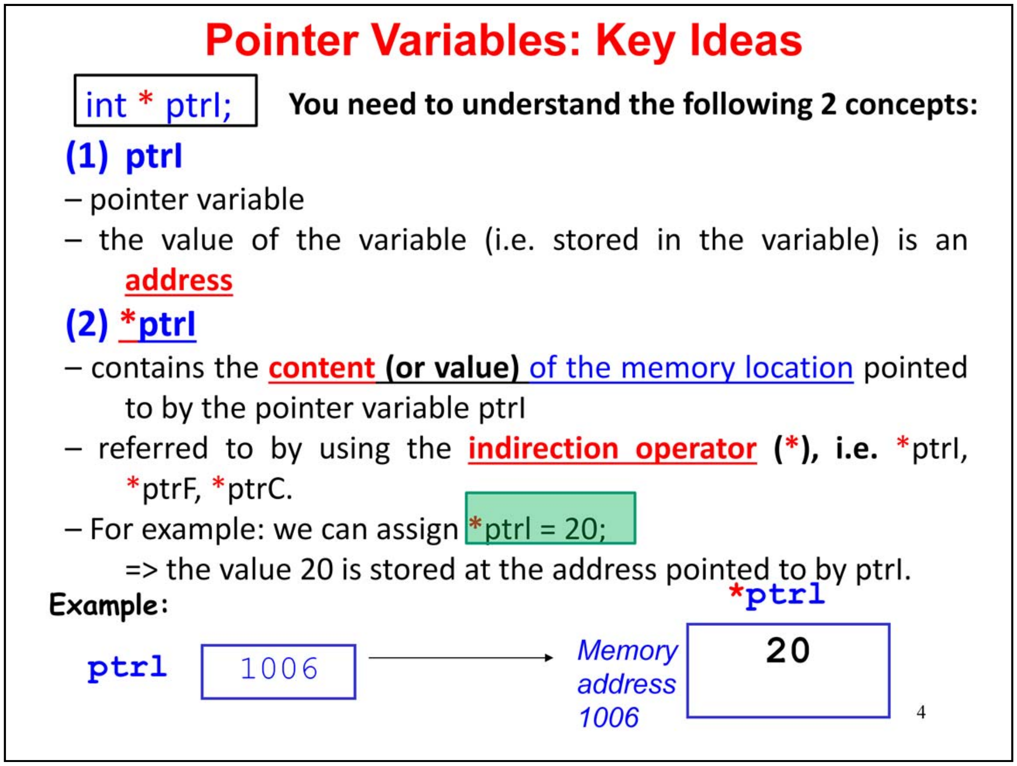
5

**Note: You may also print the address of the variable using the printf() statement.**

3

**<u>Primitive Variables: Key Ideas</u>**

1. There are two important ideas related to primitive variables in the above example:
   a) The primitive variable (**num**) stores a variable value of data type int.
   b) After applying the address operator to the variable **num** (i.e., **&num**), it refers to the memory address of the variable. The memory location is used to store the variable value.



## Pointer Variables: Key Ideas

int * ptrl; **You need to understand the following 2 concepts:**

**(1) ptrl**

– pointer variable

– the value of the variable (i.e. stored in the variable) is an **address**

**(2) *ptrl**

– contains the **content (or value)** of the memory location pointed to by the pointer variable ptrl

– referred to by using the **indirection operator (*), i.e.** *ptrl, *ptrF, *ptrC.

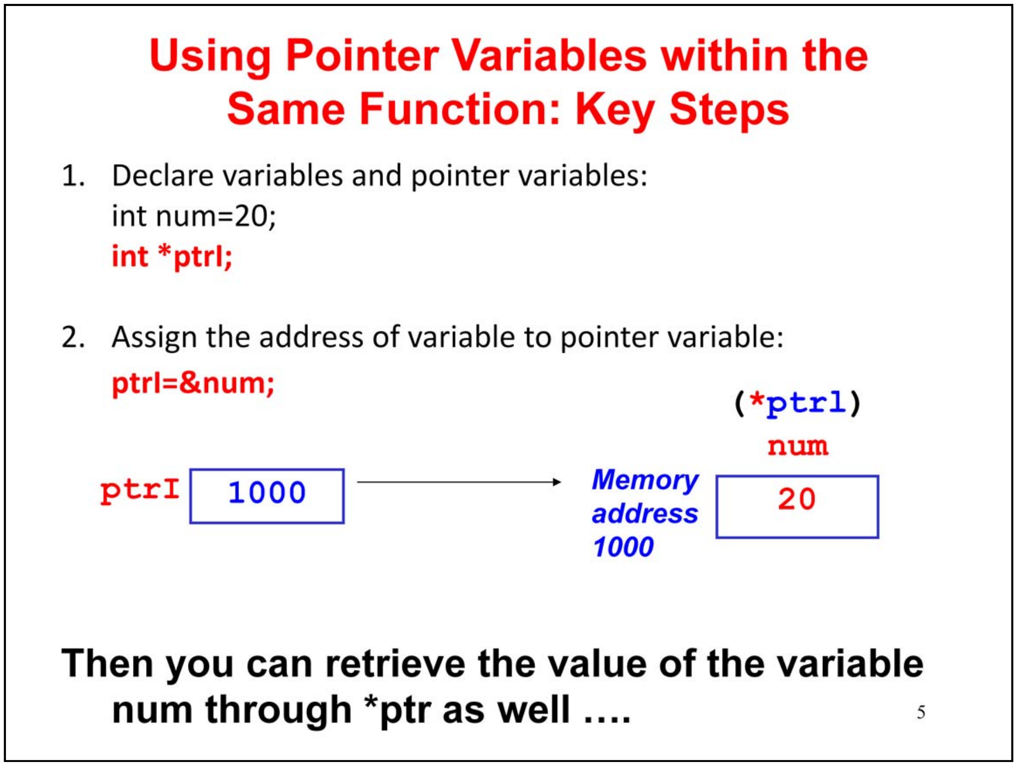
– For example: we can assign *ptrl = 20;

=> the value 20 is stored at the address pointed to by ptrl.

**Example:**

**Pointer Variables: Key Ideas**

1. There are two important concepts related to pointers:
   a) The pointer variable (**ptr**) is used to store an address which refers to the location that stores the actual data of the specified data type.
   b) The indirection operator (***ptr**) can be used to retrieve the actual value pointed to by the pointer variable.
2. For example, after declaring the pointer variable, the following assignment statement, ***ptrl = 20;** will update the memory location pointed to by the pointer variable to 20.
3. As such, we can retrieve the actual integer value of 20 referred to by the pointer variable **ptrl** using indirection operator ***ptrl** which will give the value of 20.



## Using Pointer Variables within the Same Function: Key Steps

1. Declare variables and pointer variables:
   int num=20;
   **int *ptrl;**

2. Assign the address of variable to pointer variable:
   **ptrl=&num;**

**Then you can retrieve the value of the variable num through *ptr as well ….**

5

**Using Pointer Variables within the Same Function: Key Steps**

1.There are two key steps on using pointer variables:

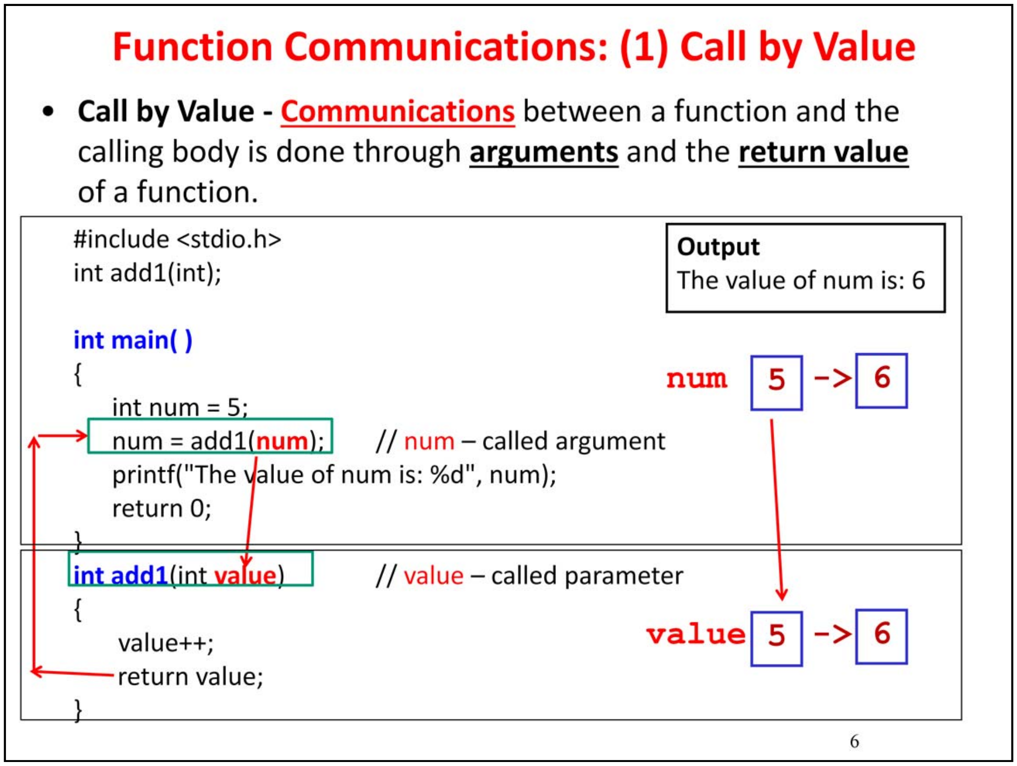
a) Declare variables and pointer variables:

**int num=20;**

**int *ptrl;**

a) Assign the address of variable to pointer variable:

**ptrl=&num;**



## Function Communications: (1) Call by Value

- **Call by Value - Communications** between a function and the calling body is done through **arguments** and the **return value** of a function.

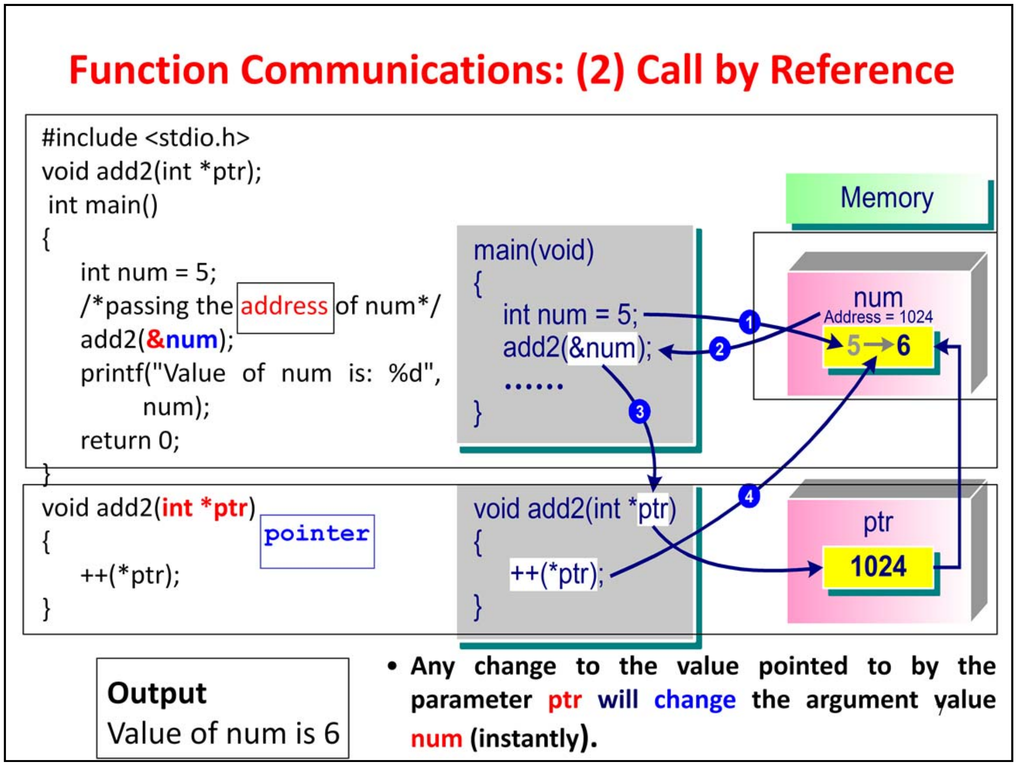
```
#include <stdio.h>
int add1(int);

int main( )
{
     int num = 5;
     num = add1(num);      // num – called argument
     printf("The value of num is: %d", num);
     return 0;
}
int add1(int value)        // value – called parameter
{
      value++;
      return value;
}
```

**Output**

```
The value of num is: 6
```

**Function Communications: Call by Value**

1. This example illustrated call by value was used in the chapter on Functions.
2. In this example, we pass in the value of the variable **num** as argument from the **main()** function to the parameter **value** of the **add1()** function.



## Function Communications: (2) Call by Reference

```
#include <stdio.h>
void add2(int *ptr);
 int main()
{
     int num = 5;
     /*passing the address of num*/
     add2(&num);
     printf("Value  of  num  is:  %d",
             num);
     return 0;
}
void add2(int *ptr)
{
     ++(*ptr);
}
```

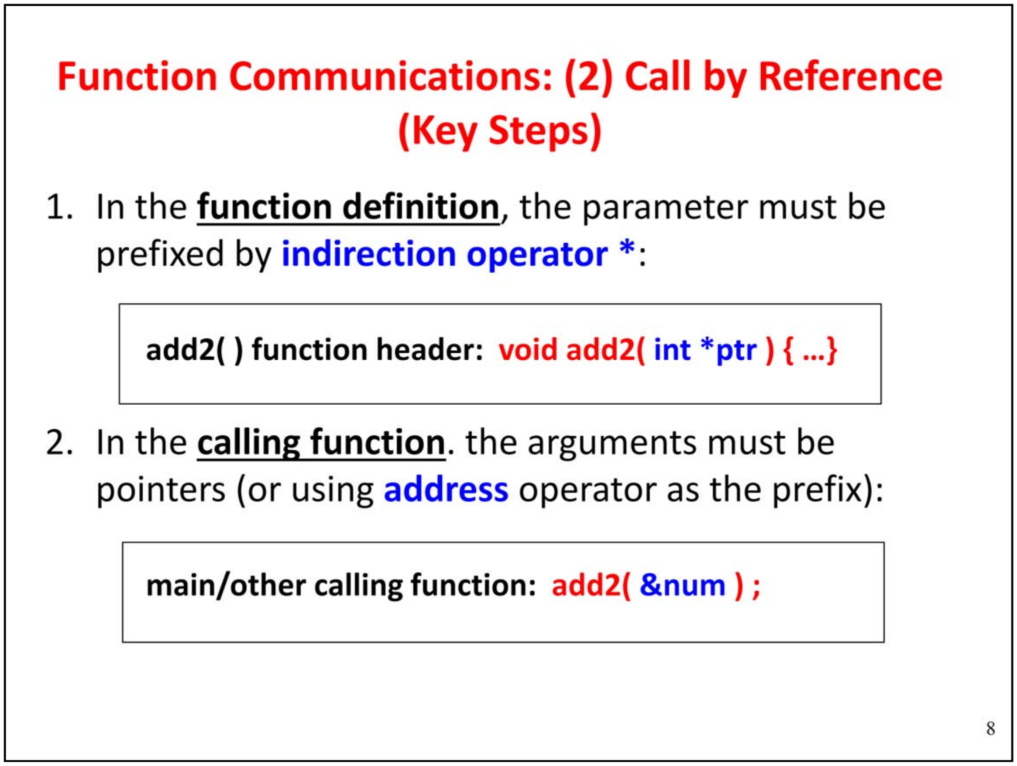
**Output**

```
Value of num is 6
```

- **Any change to the value pointed to by the parameter ptr will change the argument value num (instantly).**

**Function Communications: Call by Reference**

1. In the program, the variable **num** is initially assigned with a value 5 in **main()**.
2. The address of the variable **num** is then passed as an argument to the function **add2()** (step 2) and stored in the parameter **ptr** in the function (step 3).
3. In the function **add2()**, the value of the memory location pointed to by the variable **ptr** (i.e. **num**) is then incremented by 1 (step 4). It implies that the value stored in the variable **num** becomes 6. When the function ends, the control is then returned to the calling **main()** function. Therefore, when **num** is printed, the value 6 is displayed on the screen.
4. In this example, note that the parameter variable **ptr** in **add2()** is used to store the address of the variable **num** in **main()**. After passing the variable address of **num** into the parameter variable **ptr**, all the operations on **ptr** in the function **add2()** will update the content of the variable **num** indirectly.



## Function Communications: (2) Call by Reference (Key Steps)

1. In the **function definition**, the parameter must be prefixed by **indirection operator ***:

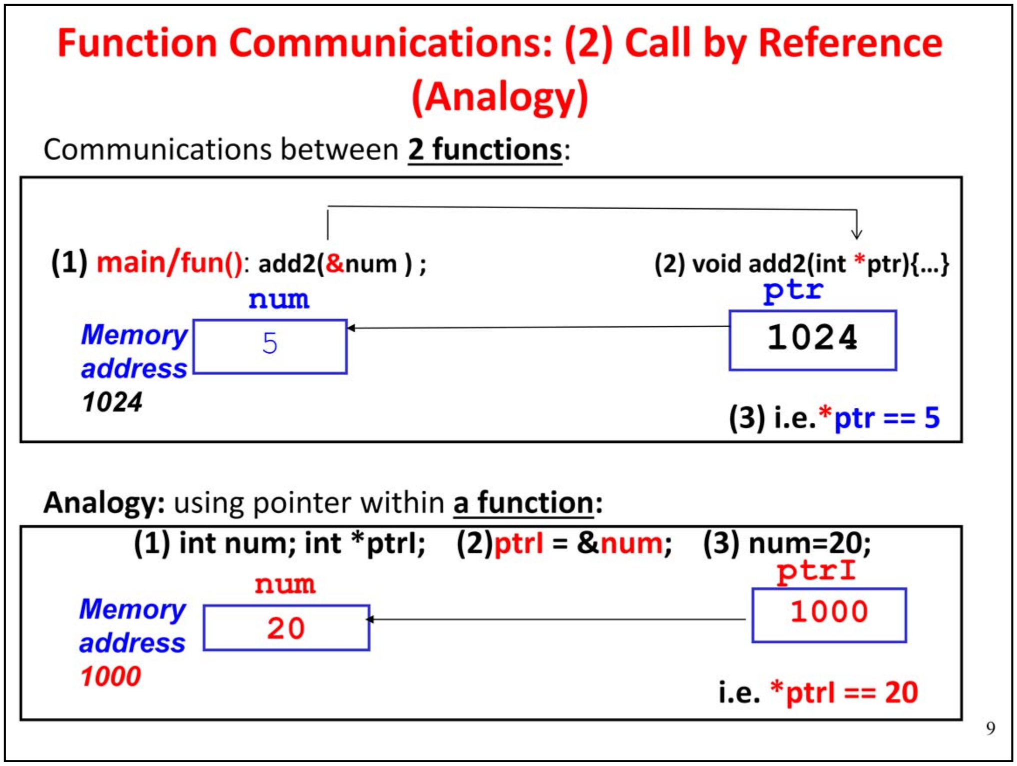
   **add2( ) function header: void add2( int *ptr ) { …}**

2. In the **calling function**. the arguments must be pointers (or using **address** operator as the prefix):

   **main/other calling function: add2( &num ) ;**

**Call by Reference: Key Steps**

1. There are two key steps when using call by reference:
   a) In the function, the parameter must be prefixed by the indirection operator: **void add2(int *ptr);**
   b) In the calling function (e.g. **main()**), the arguments must be pointers (or using address operator as the prefix): **add2(&num);**



## Function Communications: (2) Call by Reference (Analogy)

Communications between **2 functions**:

(1) main/fun(): add2(&num ) ;
(2) void add2(int *ptr){…}
num
ptr
Memory address 1024
5
1024
(3) i.e.*ptr == 5

**Analogy:** using pointer within **a function**:

(1) int num; int *ptrI; (2)ptrI = &num; (3) num=20;
num
ptrI
Memory address 1000
20
1000
i.e. *ptrI == 20

**Call by Reference: Analogy**

1. Using call by reference via pointer is very similar to that of using pointers in a function. When using pointers in a function:
   a) We first declare the variable and pointer variable: **int num; int *ptrI;**
   b) Then, assign the address of the variable **num** to **ptrI**: **ptrI = &num;**
2. Therefore, when the variable **num** is updated to 20: **num** is 20**; *ptrI** is also **20;**