4
Pointers

Pointers

- **Primitive Data Types, Variables and Address Operator**
- Pointer Variables
- Call by Reference

Variables of Primitive Data Types

```
#include  <stdio.h>
int main()
{
    int num = 5;

    printf("num = %d,                  \n", num           );


}
```

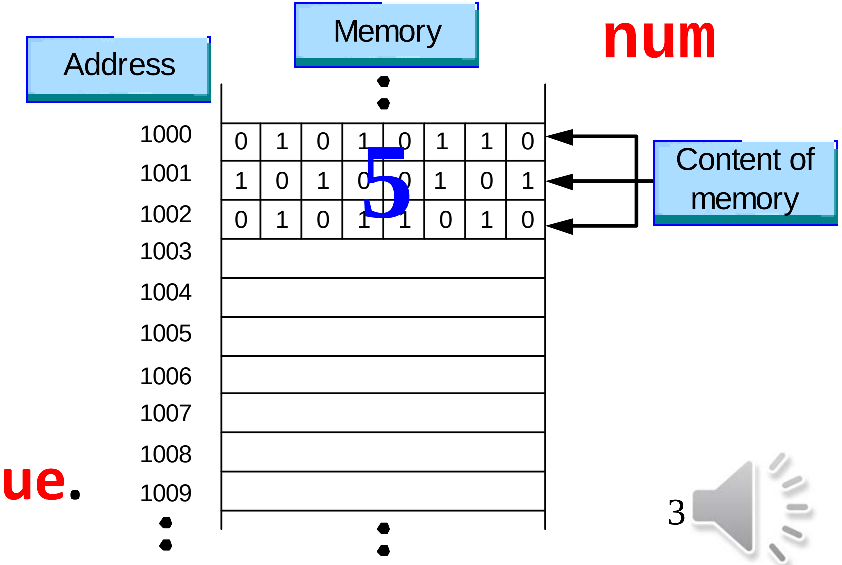
Printing the value of the variable

Variables of primitive data types: int, char, float, etc.

Output

```
num = 5,
```

Note: The variable num stores the value.



1000	0	1	0	1	0	1	1	0
1001	1	0	1	0	0	1	0	1
1002	0	1	0	1	1	0	1	0
1003								
1004								
1005								
1006								
1007								
1008								
1009								

Variables of Primitive Data Types

```
#include  <stdio.h>
int main()
{
     int num = 5;

     printf("num = %d,                    \n", num           );
→    scanf("%d", &num);
     printf("num = %d,                    \n", num           );
}
```

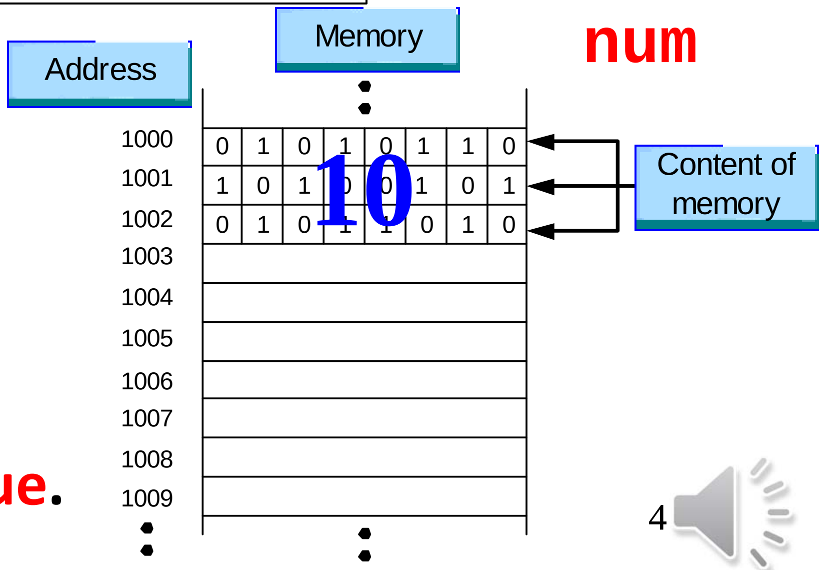
Printing the value of the variable

Variables of primitive data types: int, char, float, etc.

Output

```
num = 5,
→10
num = 10,
```

Note: The variable num stores the value.



Address Operator (&)

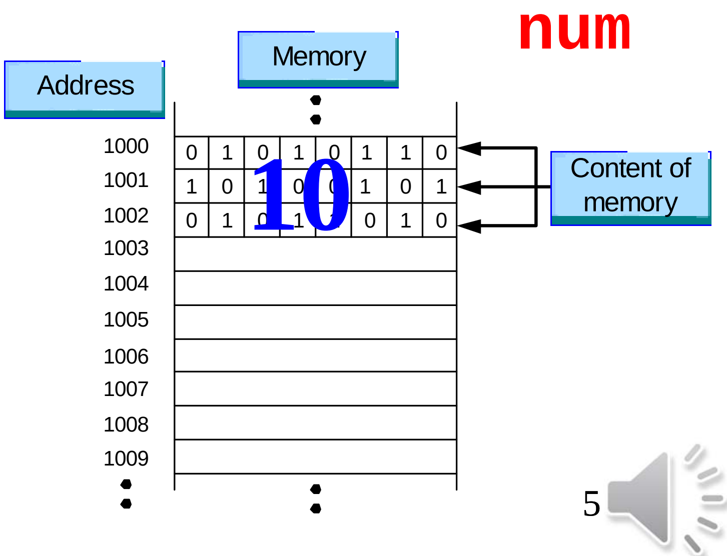
```
#include  <stdio.h>
int main()
{
     int num = 5;

     printf("num = %d, &num = %p\n", num, &num);
     scanf("%d", &num);
     printf("num = %d, &num = %p\n", num, &num);
}
```

Printing the memory address of the variable

Output

```
num = 5, &num = 1000 [address]
10
num = 10, &num = 1000
```

Note: The variable num stores the value.



Primitive Variables: Key Ideas

```
int num;
```

(1) num

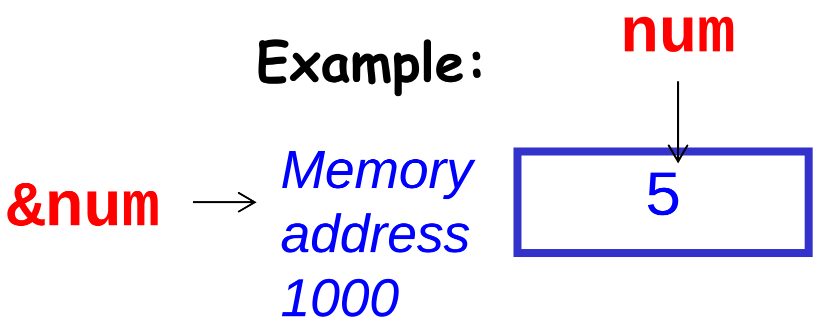
– it is a variable of data type int

– 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.

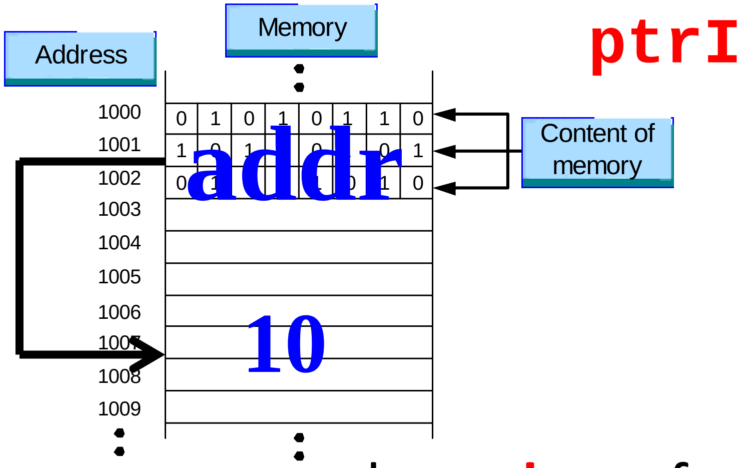
Pointers

- Primitive Data Types, Variables and Address Operator
- **Pointer Variables**
- Call by Reference

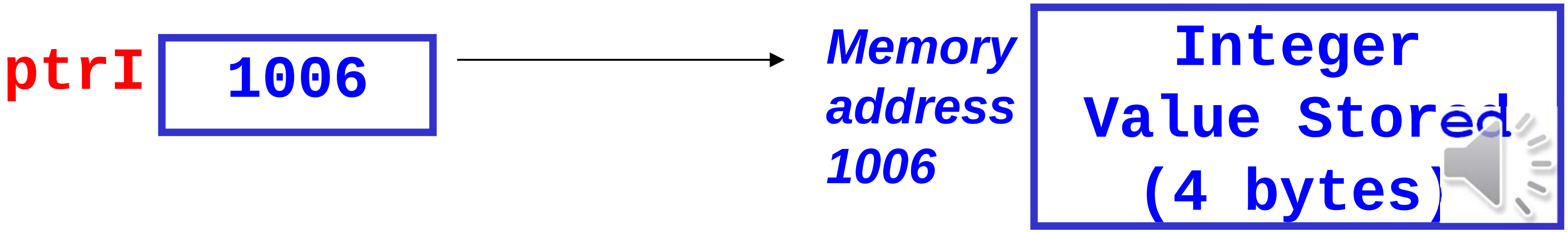
Pointer Variables: Declaration

- **Pointer variable** – different from the variable **num** (variable of primitive data type such as int, float, char) declared earlier, it stores the **address** of memory location of a data object.
- A **pointer variable** is declared by, for example:

```
      int *ptrl;
or    int * ptrl;
or    int* ptrl;
```



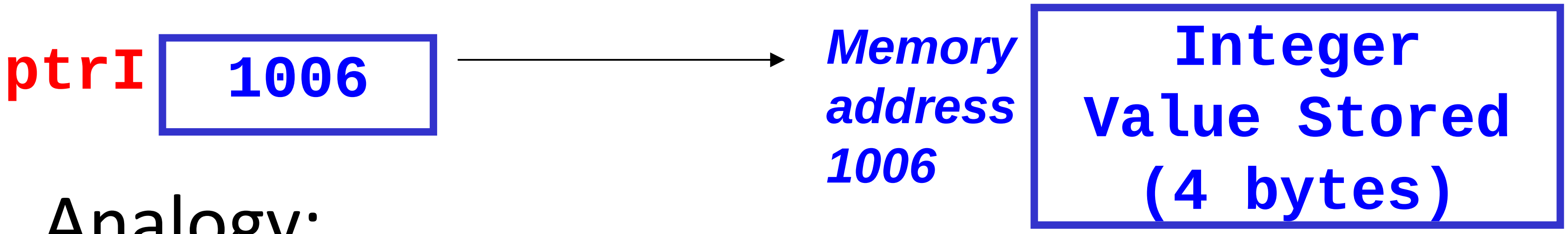
- **ptrl** is a pointer variable. It does not store the value of the variable. It stores the address of the memory which is used for storing an Int value. Diagrammatically,


ptrI 1006

Memory address 1006

Integer Value Stored (4 bytes)

Pointer Variables: Analogy



ptrI

1006

Memory address 1006

Integer Value Stored (4 bytes)

- Analogy:

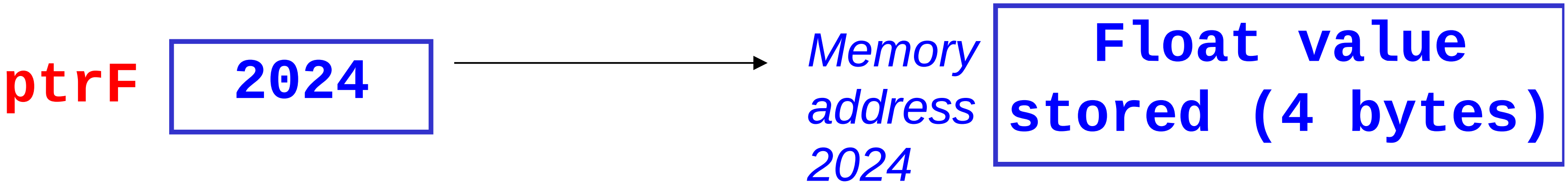
  (1) Address on envelope → your home

  (2) Bank account → your saving/money in the bank

Pointer Variables: Declaration Examples

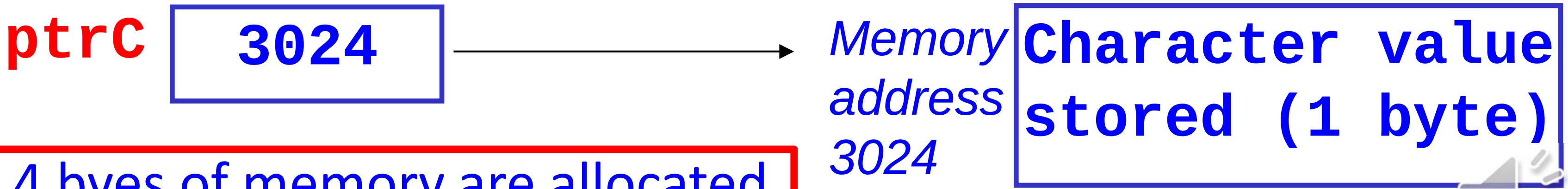
```
float *ptrF;
```

ptrF is a pointer variable. It stores the **address** of the memory which is used for storing a Float value.



```
char *prtC;
```

ptrC is a pointer variable. It stores the **address** of the memory which is used for storing a Character value.



Note: 4 byes of memory are allocated to each pointer variable.

Pointer Variables: Key Ideas

`int * ptrI;`

You need to understand the following 2 concepts:

(1) ptrI

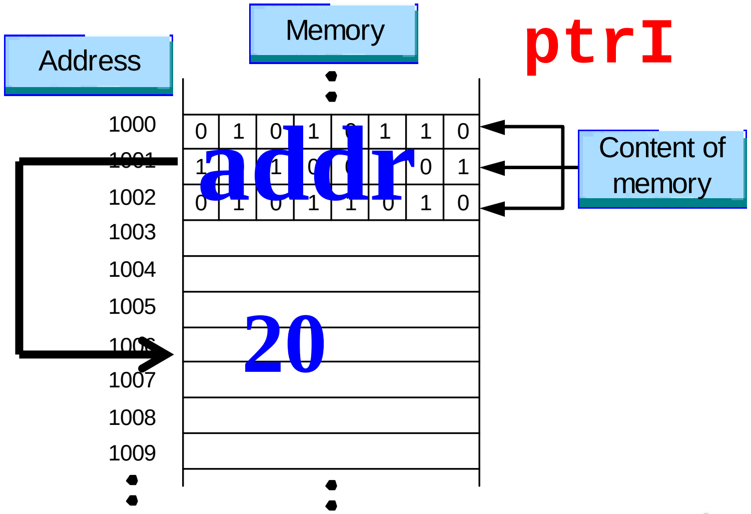
- pointer variable
- the value of the variable (i.e. stored in the variable) is an **address**

(2) *ptrI

- contains the **content** **(or value)** of the memory location pointed to by the pointer variable ptrI
- referred to by using the **indirection operator** **(*), i.e.** *ptrI, *ptrF, *ptrC.
- For example: we can assign

  ***ptrI = 20;**

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



How to use Pointer Variables?

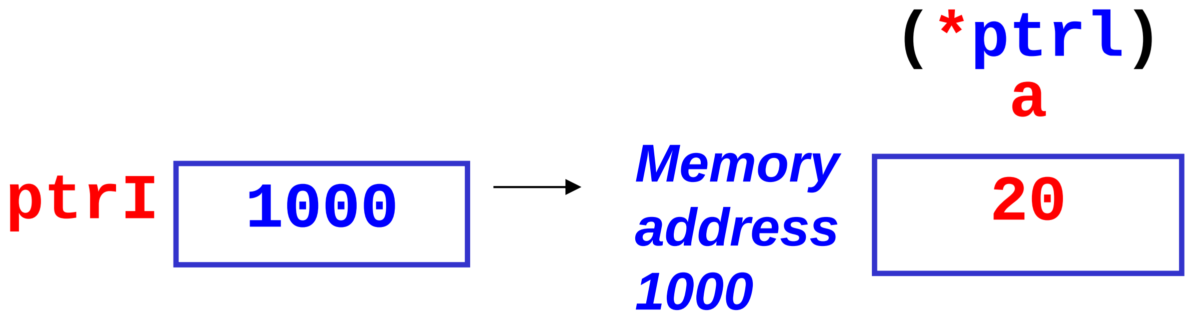
(1) Declare variables

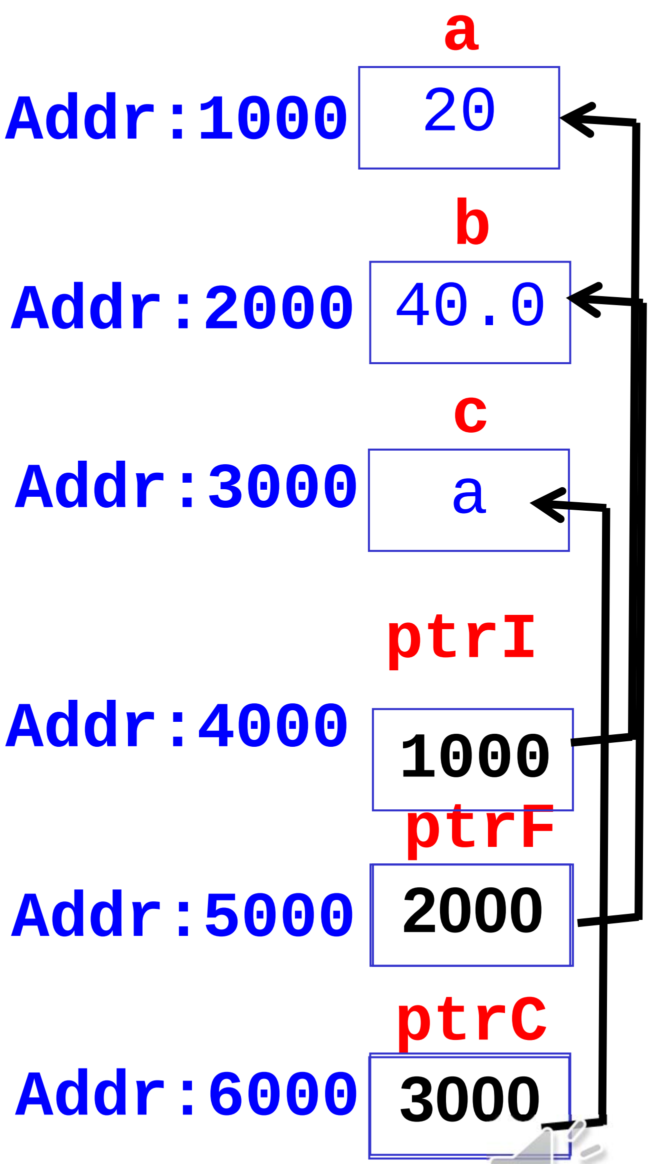
int a=20; float b=40.0; char c='a';
int *ptrl; float *ptrF; char *ptrC;

(2) Assign variable address to pointer variable

ptrl = &a;
ptrF = &b;
ptrC = &c;

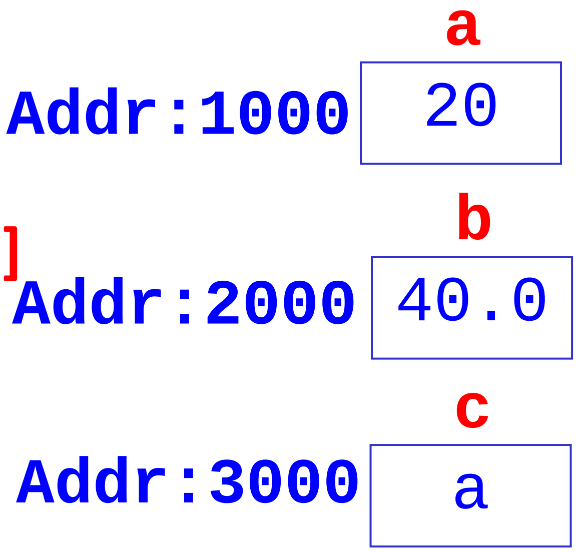
After establishing the relationship, *ptrl and a – are now refer to the same memory content

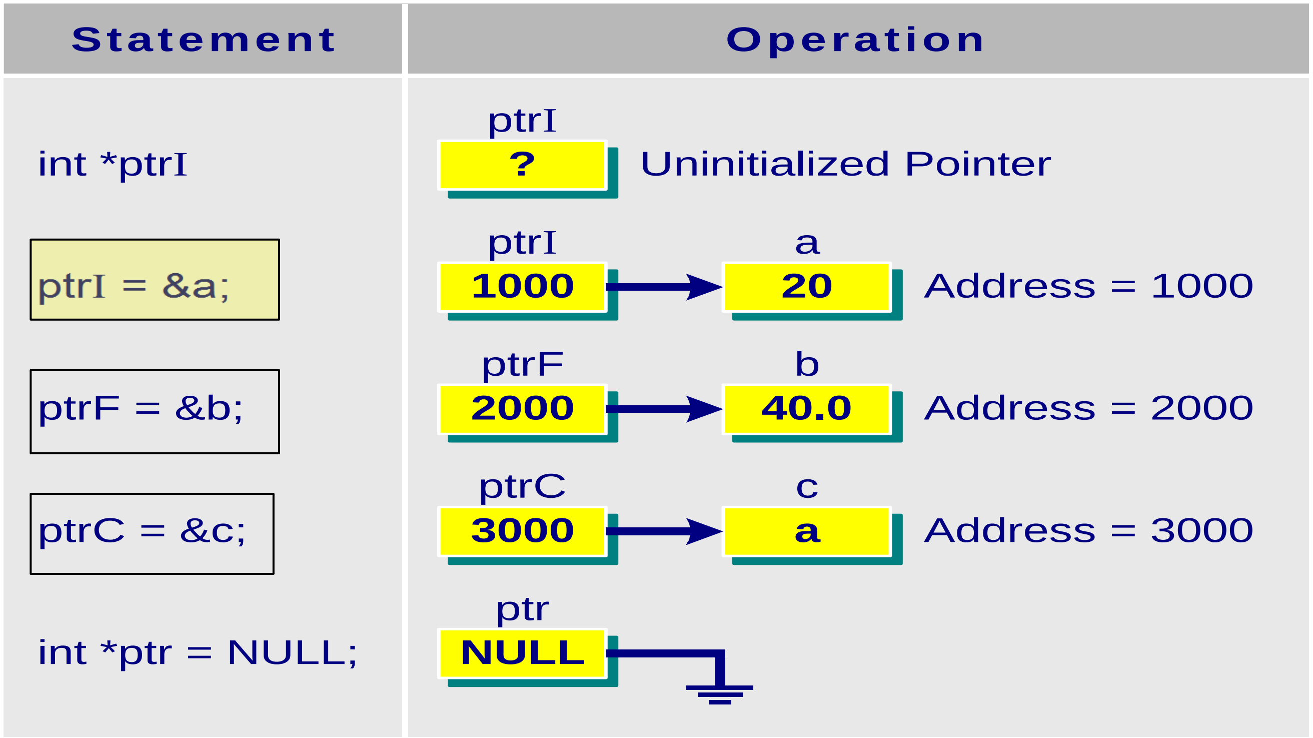




How to use Pointer Variables? (Cont’d.)

```
int a=20;      float b=40.0;    char c='a';
int *ptrI;     float *ptrF;     char *ptrC;
ptrI = &a;          => *ptrI == 20 [same as variable a]
ptrF = &b;          => *ptrF == 40.0  [same as b]
ptrC = &c;          => *ptrC == ‘a’  [same as c]
```



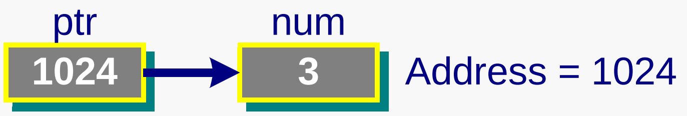
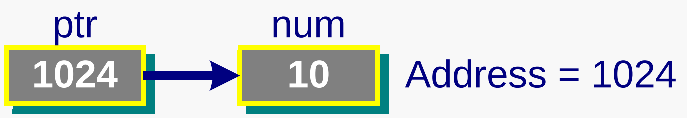


Statement	Operation
int *ptrI	ptrI: ? — Uninitialized Pointer
ptrI = &a;	ptrI: 1000 → a: 20 — Address = 1000
ptrF = &b;	ptrF: 2000 → b: 40.0 — Address = 2000
ptrC = &c;	ptrC: 3000 → c: a — Address = 3000
int *ptr = NULL;	ptr: NULL

*ptrI and a – now refer to the same memory content

Pointer Variables – Example 1

```
#include  <stdio.h>
int main()
{
   int num = 3; // integer var
   int * ptr;    // pointer var

   ptr = &num;  // assignment

  // Question: what will be ptr, *ptr, num?
  printf("num = %d, &num = %p\n", num, &num);

  printf("ptr = %p, *ptr = %d\n", ptr, *ptr);



 }
```

Statement	Operation
ptr = &num;	 ptr 1024 → num 3 Address = 1024
*ptr = 10;	 ptr 1024 → num 10 Address = 1024

Output

num = 3,

&num = 1024

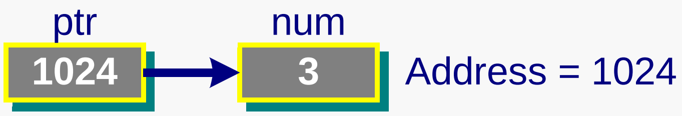
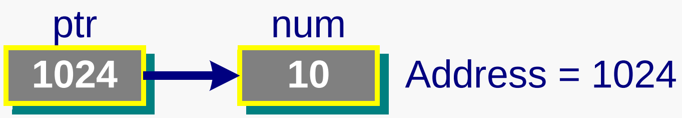
ptr = 1024,

***ptr = 3**

[num and *ptr have the same value]

Pointer Variables – Example 1 (Cont’d.)

```
#include  <stdio.h>
int main()
{
    int num = 3; // integer var
    int *ptr;       // pointer var

    ptr = &num;

   printf("num = %d &num = %p\n", num, &num);

   printf("ptr = %p *ptr = %d\n", ptr, *ptr);

    *ptr = 10;
    // What will be the values for *ptr, num, &num?
    printf("num = %d &num = %p\n", num, &num);
    return 0;
 }
```

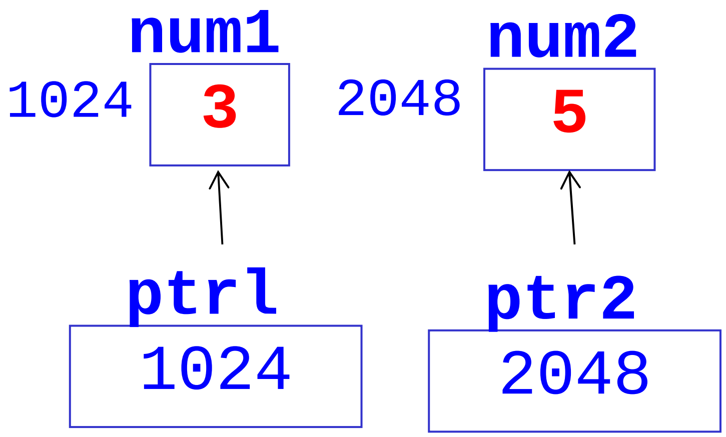
Statement	Operation
ptr = &num;	
*ptr = 10;	

Output

```
num = 3
&num = 1024

ptr = 1024
*ptr = 3


num = 10
[*ptr = 10]
&num = 1024
```

Pointer Variables – Example 2

```
/* Example to show the use of pointers */
 #include  <stdio.h>
int  main()
{
    int num1 = 3, num2 = 5;   // integer variables
    int *ptr1, *ptr2;         // pointer variables

    ptr1 = &num1;   /* put the address of num1 into ptr1 */
    // What are the values for num1, *ptr1?
    printf(“num1 = %d, *ptr1 = %d\n”, num1, *ptr1);

    ptr2 = &num2;   /* put the address of num2 into ptr2 */
    // What are the values for num2, *ptr2?
    printf(“num2 = %d, *ptr2 = %d\n”, num2, *ptr2);
```



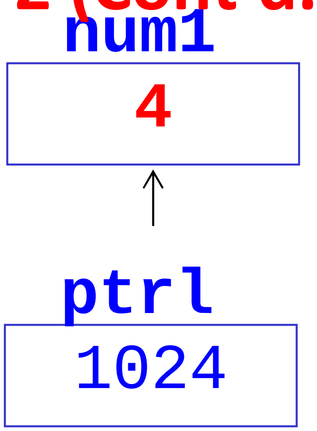
Output

```
num1 = 3, *ptr1 = 3
num2 = 5, *ptr2 = 5
```

Pointer Variables – Example 2 (Cont’d.)

/* increment by 1 the content of the memory location pointed by ptr1 */

```
(*ptr1)++;
```



// What are the values for num1, *ptr1?

printf(“num1 = %d, *ptr1 = %d\n”, num1, ***ptr1**);

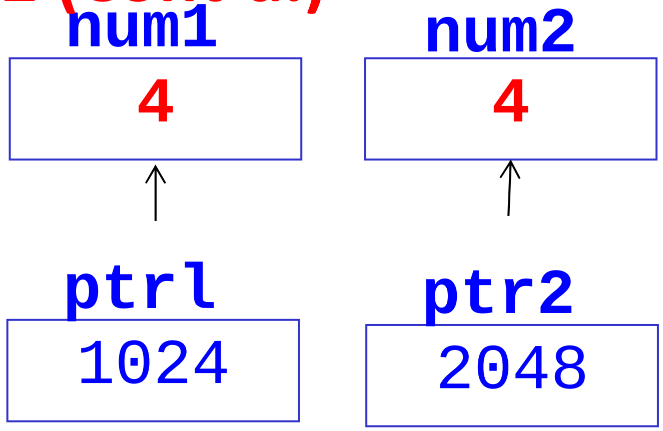
Output

```
num1 = 4, *ptr1 = 4
```

Pointer Variables – Example 2 (Cont’d.)

/* copy the content of the location pointed by ptr1 into the location pointed by ptr2*/

```
*ptr2 = *ptr1;
```



// What are the values for num2, *ptr2?

```
printf(“num2 = %d,*ptr2 = %d\n”,num2, *ptr2);
```

Output

```
num2 = 4, *ptr2 = 4
```

Pointer Variables – Example 2 (Cont'd.)

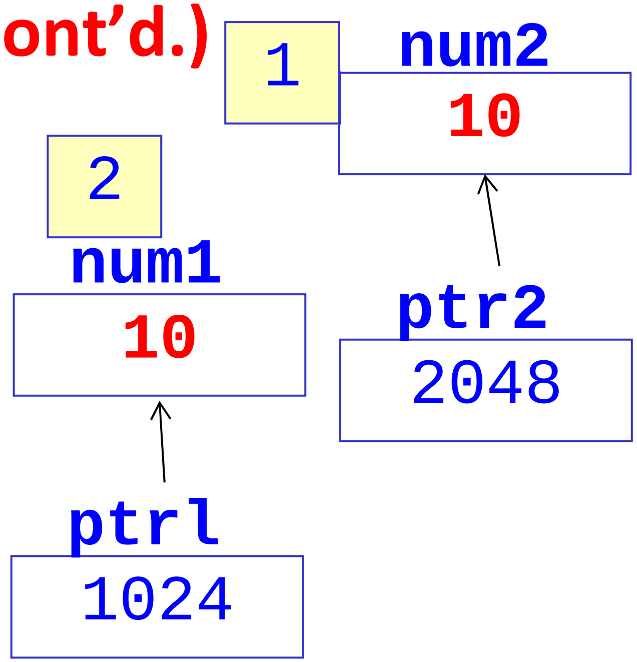
```
*ptr2 = 10;
```

1 /*10 copied into the location pointed by ptr2*/

```
num1 = *ptr2;
```

2 /* copy the content of the memory location pointed by ptr2 into num1 */

```
printf("num1 = %d,*ptr1 = %d\n",num1, *ptr1);
```



Output

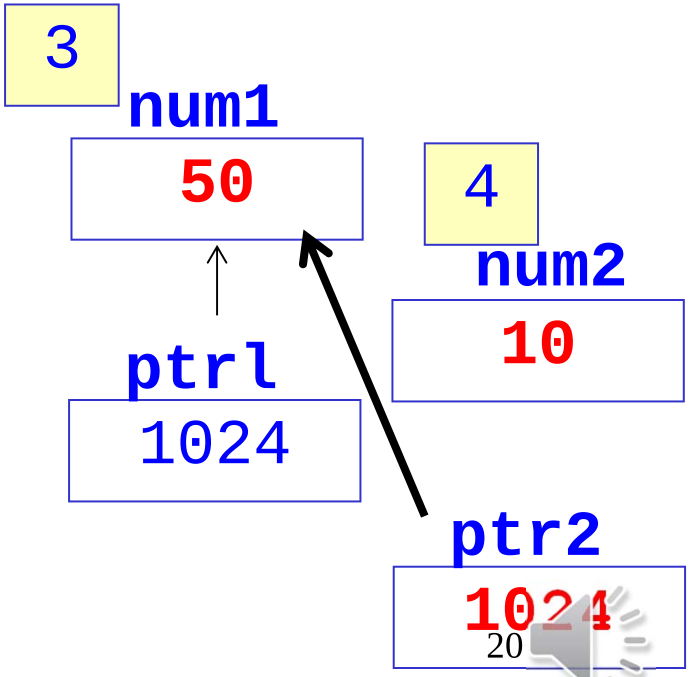
```
num1 = 10, *ptr1 = 10
```

Pointer Variables – Example 2 (Cont’d.)

Output

```
num1 = 50, *ptr1 = 50
num2 = 10, *ptr2 = 50
```

```
*ptr1 = *ptr1 * 5;    3

  printf(“num1 =%d, *ptr1 = %d\n”,num1, *ptr1);

ptr2 = ptr1;    4   /*address in ptr1 copied into ptr2 */

  printf(“num2 = %d, *ptr2 = %d\n”, num2, *ptr2);
  return 0;
}
```



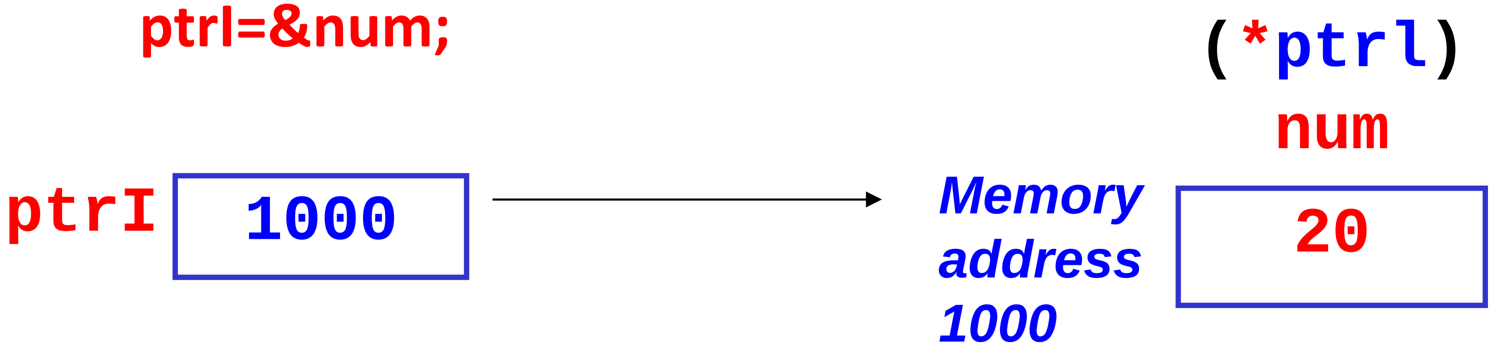
Using Pointer Variables (within the Same Function): Key Steps

1. Declare variables and pointer variables:

```
int num=20;
int *ptrl;
```

2. Assign the address of variable to pointer variable:

```
ptrl=&num;
```



(*ptrl)

num

ptrI

1000

Memory address 1000

20

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

Pointers

- Primitive Data Types, Variables and Address Operator
- Pointer Variables
- **Call by Reference**

Call by Reference

- Parameter passing between functions has two modes:
  - **call by value** [in the last lecture on Function]
  - **call by reference** [to be discussed in this lecture]

- **Call by reference**: the parameter in the function holds the address of the argument variables, i.e. the **parameter** is a **pointer variable**.
  - In a **function call**, the **arguments** must be **pointers** (or using address operator as the prefix).

```
E.g.  double x1,y1;
           ....
      distance(&x1, &y1);
```

  - In the **function header**'s parameter declaration list, the **parameters** must be prefixed by the **indirection operator ***.

```
E.g. void distance(double *x, double *y)
```

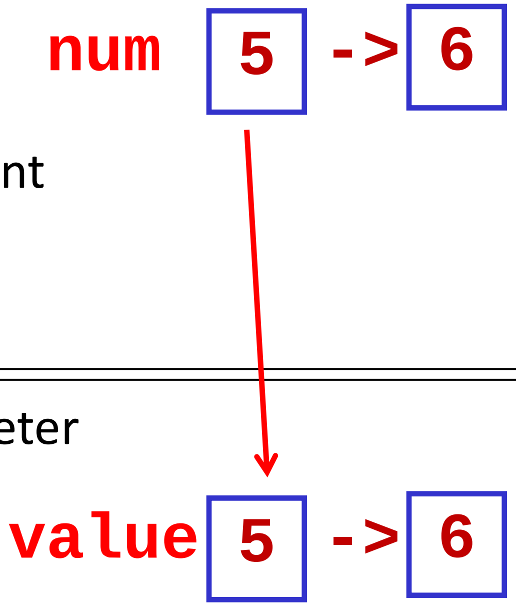
Recap: Call by Value

- **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

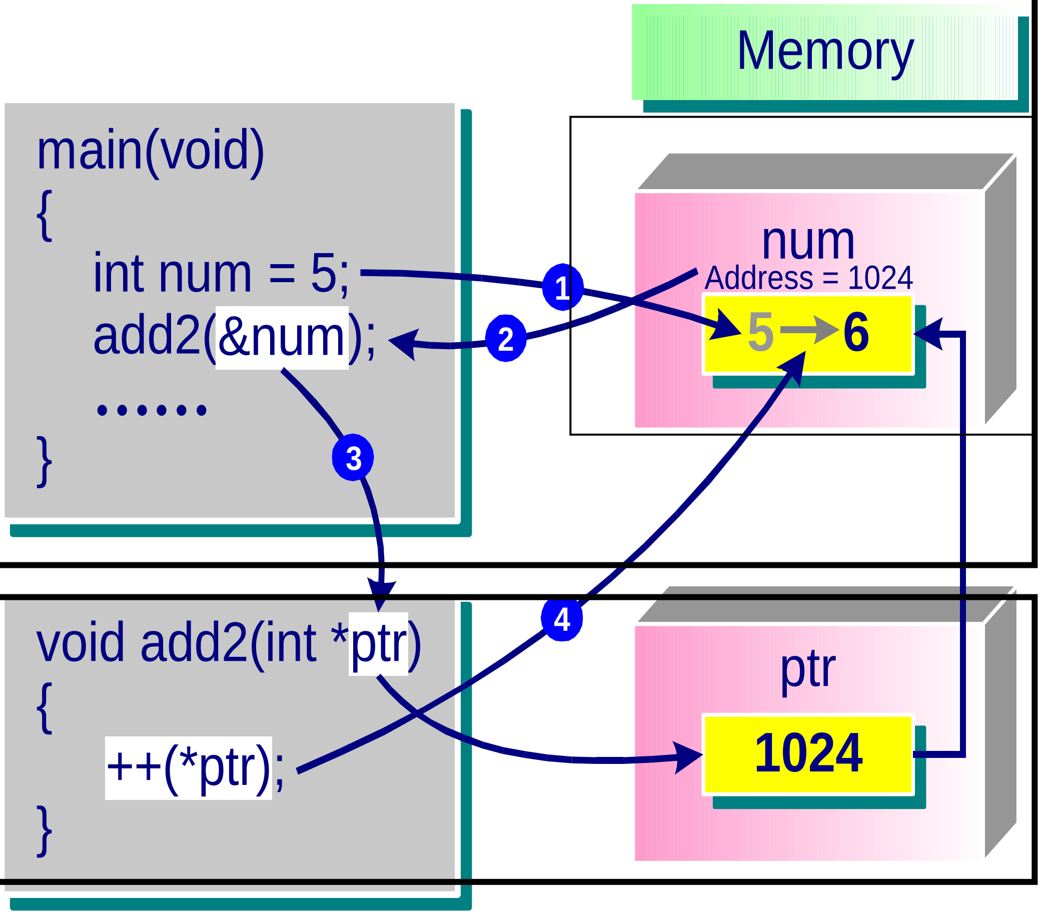
```
The value of num is: 6
```



Call by Reference: Example 1

```
#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);
}
```

pointer



```
main(void)
{
    int num = 5;
    add2(&num);
    ......
}
```

```
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).**

Call by Reference: Key Steps

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

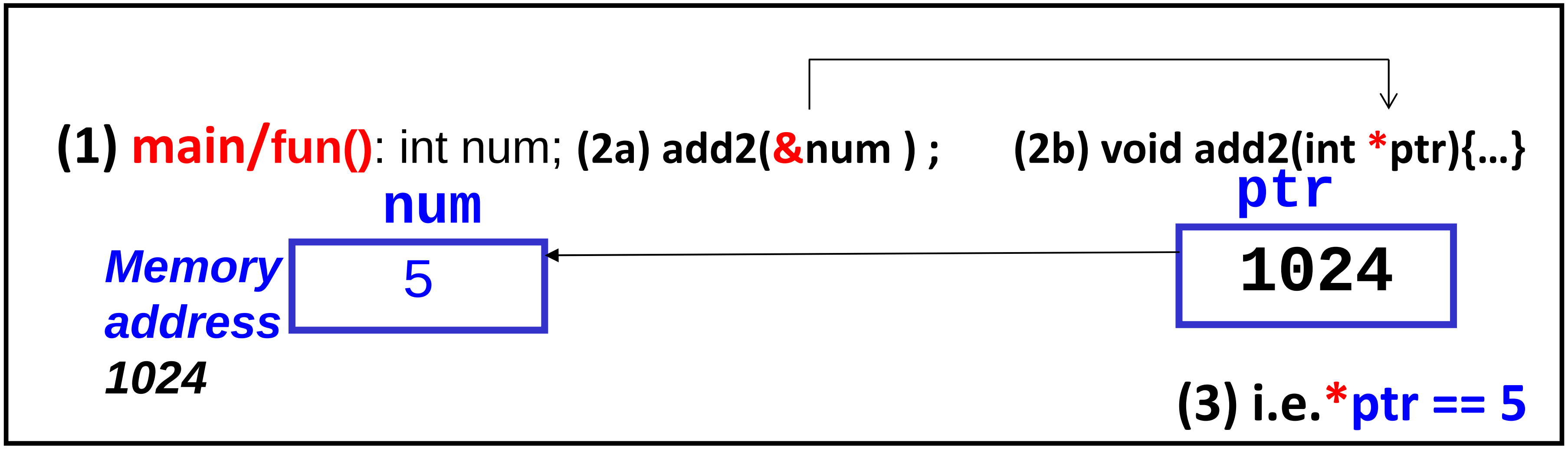
add2( ) function header: void add2(int *ptr) { …}

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

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

Call by Reference: Analogy

Communications between **2 functions**:

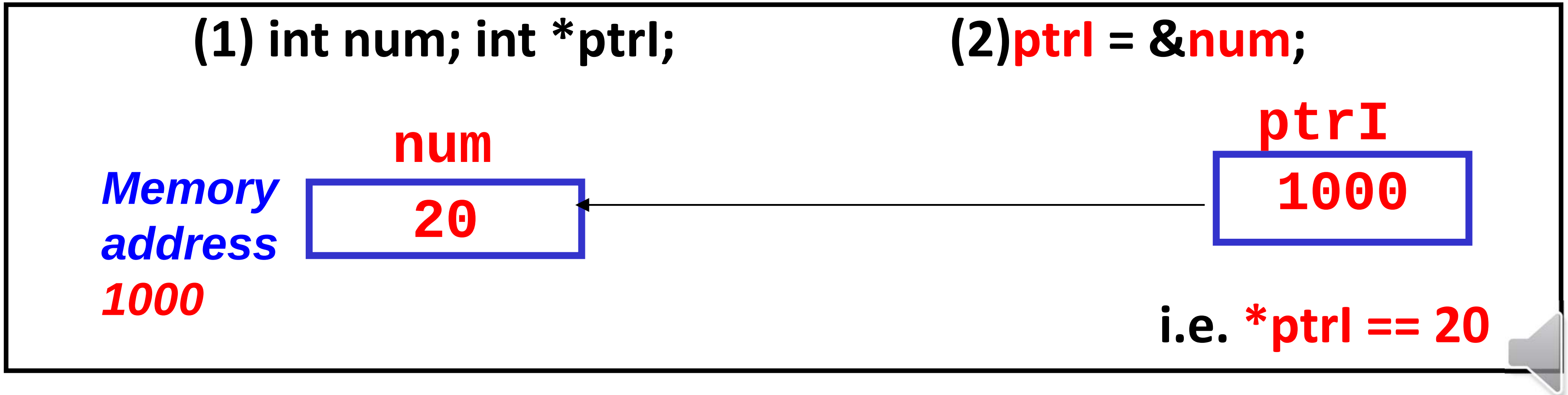


(1) main/fun(): int num; **(2a) add2(&num) ;**

(2b) void add2(int *ptr){…}

(3) i.e.*ptr == 5

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



(1) int num; int *ptrl;

(2)ptrl = &num;

i.e. *ptrl == 20

Call by Reference – Example 2

```
#include<stdio.h>
void function1 (int a, int *b); void function2 (int c, int *d);
void function3 (int h, int *k);
int main() {
    int x, y;
    x = 5;  y = 5;                               /* (i) */
    function1(x, &y);                            /* (x) */
    return 0;
}
void function1(int a, int *b)  {                 /* (ii) */
    *b = *b + a;                                 /* (iii) */
    function2(a, b);                             /* (ix) */
}
void function2(int c, int *d) {                  /* (iv) */
    *d = *d * c;                                 /* (v) */
    function3(c, d);                             /* (viii) */
}
void function3(int h, int *k) {                  /* (vi) */
    *k = *k - h;                                 /* (vii) */
}
```

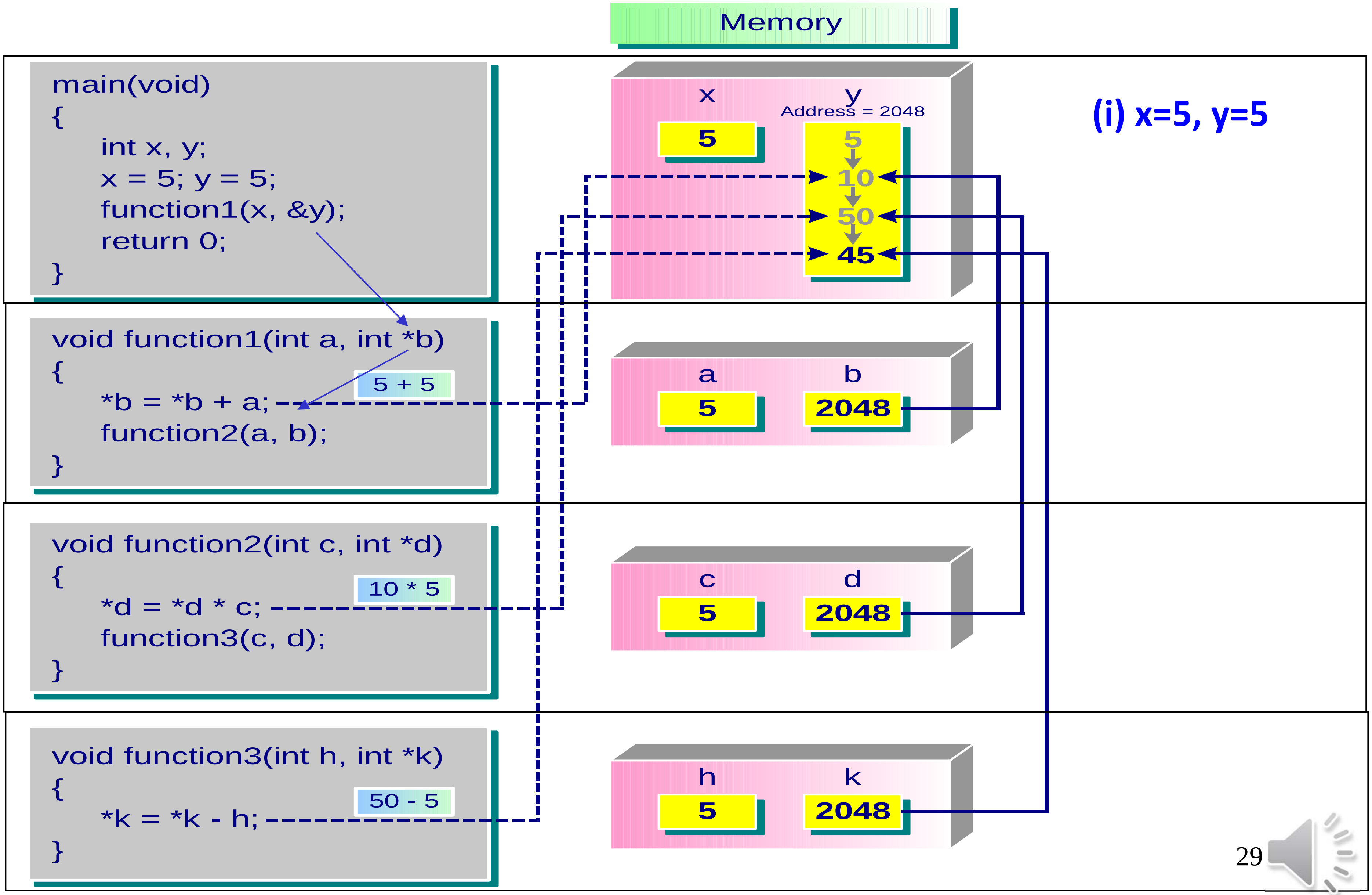
address

pointer

pointer

pointer

Call by Reference – Example 2 (i)



```
main(void)
{
     int x, y;
     x = 5; y = 5;
     function1(x, &y);
     return 0;
}
```

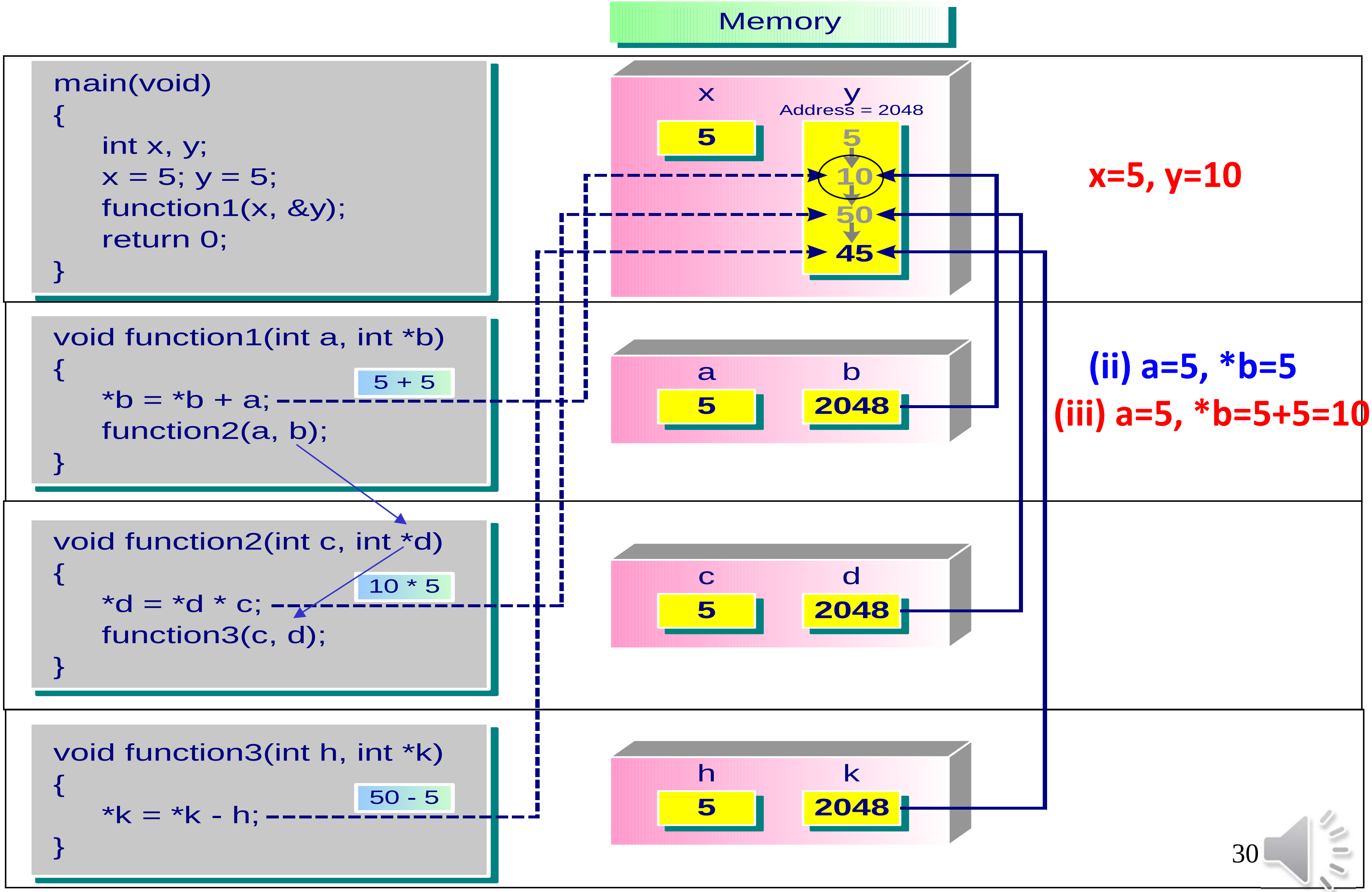
```
void function1(int a, int *b)
{
     *b = *b + a;
     function2(a, b);
}
```

```
void function2(int c, int *d)
{
     *d = *d * c;
     function3(c, d);
}
```

```
void function3(int h, int *k)
{
     *k = *k - h;
}
```

(i) x=5, y=5

Call by Reference – Example 2 (ii, iii)



```
main(void)
{
      int x, y;
      x = 5; y = 5;
      function1(x, &y);
      return 0;
}
```

x=5, y=10

```
void function1(int a, int *b)
{
      *b = *b + a;
      function2(a, b);
}
```

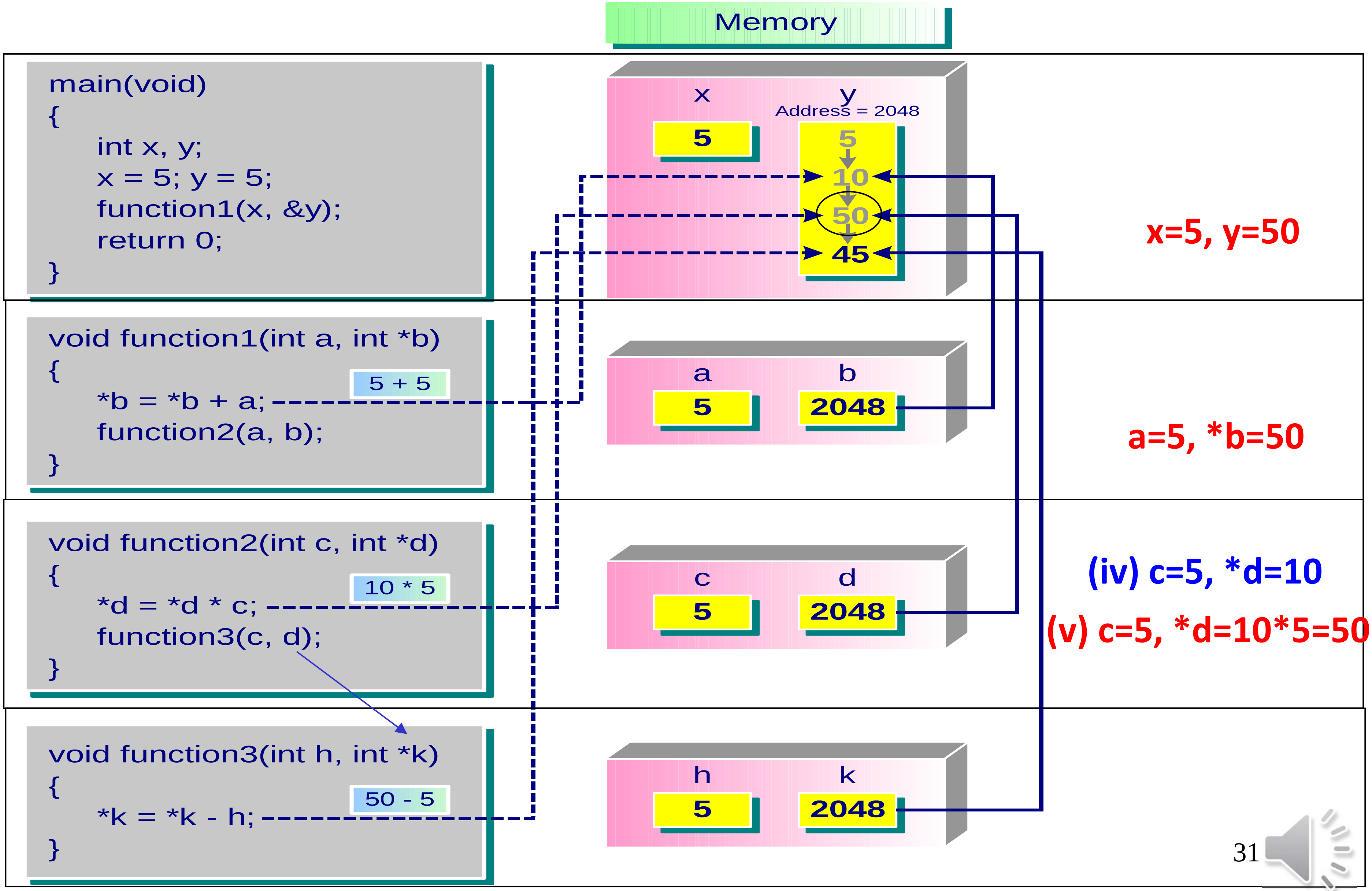
(ii) a=5, *b=5

(iii) a=5, *b=5+5=10

```
void function2(int c, int *d)
{
      *d = *d * c;
      function3(c, d);
}
```

```
void function3(int h, int *k)
{
      *k = *k - h;
}
```

Call by Reference – Example 2 (iv, v)



```
main(void)
{
      int x, y;
      x = 5; y = 5;
      function1(x, &y);
      return 0;
}
```

```
void function1(int a, int *b)
{
      *b = *b + a;
      function2(a, b);
}
```

```
void function2(int c, int *d)
{
      *d = *d * c;
      function3(c, d);
}
```

```
void function3(int h, int *k)
{
      *k = *k - h;
}
```

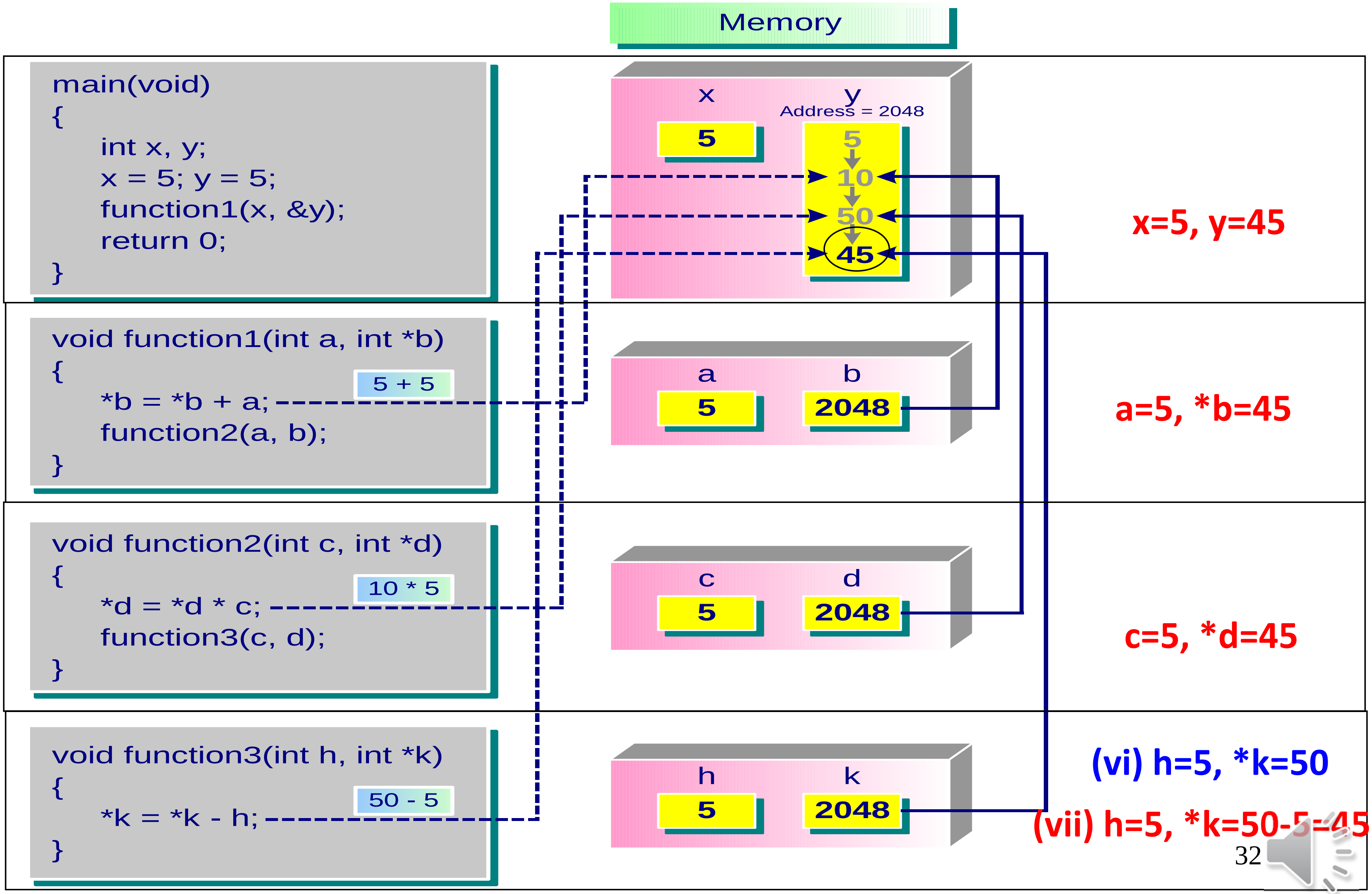
x=5, y=50

a=5, *b=50

(iv) c=5, *d=10

(v) c=5, *d=10*5=50

Call by Reference – Example 2 (vi, vii, etc.)



```
main(void)
{
     int x, y;
     x = 5; y = 5;
     function1(x, &y);
     return 0;
}
```

```
void function1(int a, int *b)
{
     *b = *b + a;
     function2(a, b);
}
```

```
void function2(int c, int *d)
{
     *d = *d * c;
     function3(c, d);
}
```

```
void function3(int h, int *k)
{
     *k = *k - h;
}
```

x=5, y=45

a=5, *b=45

c=5, *d=45

(vi) h=5, *k=50

(vii) h=5, *k=50-5=45

Call by Reference – Example 2

	x	y	a	*b	c	*d	h	*k	remarks
(i)	5	5	-	-	-	-	-	-	in main
(ii)	5	5	5	5	-	-	-	-	in fn 1
(iii)	5	10	5	10	-	-	-	-	in fn 1
(iv)	5	10	5	10	5	10	-	-	in fn 2
(v)	5	50	5	50	5	50	-	-	in fn 2
(vi)	5	50	5	50	5	50	5	50	in fn 3
(vii)	5	45	5	45	5	45	5	45	in fn 3
(viii)	5	45	5	45	5	45	-	-	return to fn 2
(ix)	5	45	5	45	-	-	-	-	return to fn 1
(x)	5	45	-	-	-	-	-	-	return to main

When to Use Call by Reference

When to use call by reference:

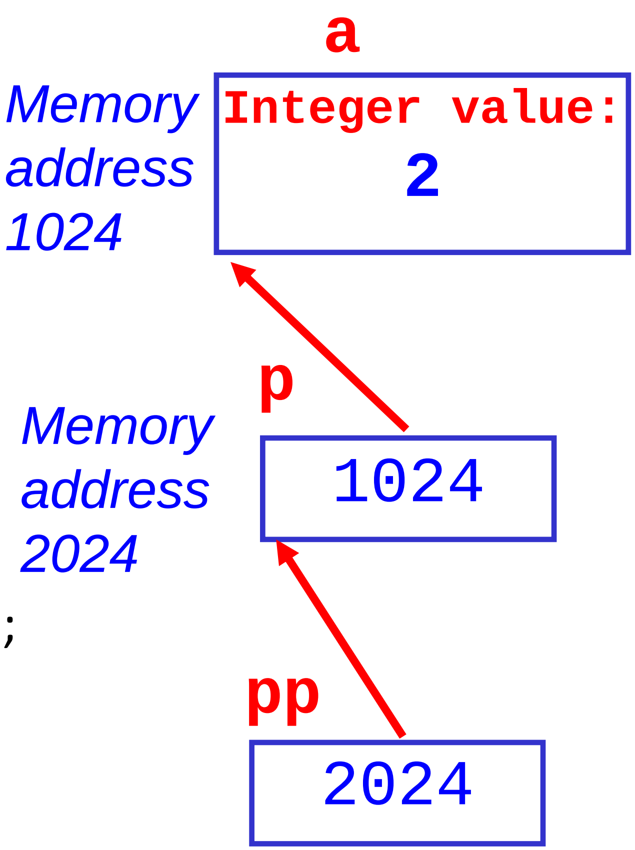
(1) When you need to pass **more than one value** back from a function.

(2) When using call by value will result in a **large piece of information** being copied to the formal parameter, for efficiency reason, for example, passing large arrays or structure records.

Double Indirection

```
#include  <stdio.h>
  int main()
  {
    int  a=2;
    int *p;
    int **pp;
    
    p = &a;
    pp = &p;
    a++;
    printf("a = %d, *p = %d, **pp = %d\n", a, *p, **pp);
    return 0;
  }
```

double indirection



a

Memory address 1024

Integer value: **2**

p

Memory address 2024

1024

pp

2024

Output

```
a = 2
*p = 2
**pp = 2
```

Note: it could also be ***ppp, etc. The idea remains the same.

Thank you !!!