CS100
Introduction to Programming

Lecture 6. Pointers

Learning Objectives

- At the end of this lecture, you should be able to understand and use the following:
  - Address Operator
  - Pointer Variables
  - Indirection Operators
  - Call by Reference

Address Operator (&)

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

   printf("num = %d, &num = %p\n", num, &num);
   scanf("%d", &num);
   printf("num = %d, &num = %p\n", num, &num);
   return 0;
}
```

This value is just for **illustration**, and may be different for another run.

Output:

num = 5, &num = 1024

10

num = 10, &num = 1024

Pointer Variables

- We may have variables which store the addresses of memory locations of some data objects. These variables are called **pointers**.
- A **pointer variable** is declared by **`dataType *pointerName`**, for example:

`int *ptrI;`	/* Variable `ptrI` is a pointer. It stores the address of a memory location for an **integer** */
`float *ptrF;`	/* Variable `ptrF` is a pointer. It stores the address of a memory location for a **float** */
`char *ptrC;`	/* Variable `ptrC` is a pointer. It stores the address of a memory location for a **char** */

- The **value** of a pointer variable is an **address**.

Pointer Variables

Example:

```
int a = 20; float b = 40.0; char c = ‘a’;
int *ptrI; float *ptrF; char *ptrC;
ptrI = &a; ptrF = &b; ptrC = &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

Indirection Operators (*)

- The **content** of the memory location pointed to by a pointer variable is referred to by using the **indirection operator *** .
- If a pointer variable is defined as **ptr**, we use the expression ***ptr** to *dereference* the pointer to obtain the value stored at the address pointed to by the pointer **ptr**.

Indirection Operator – Example 1

```
#include <stdio.h>
int main(void)
{
   int num = 3;
   int *ptr;

   ptr = &num;
   printf("num = %d, &num = %p\n", num, &num);
   printf("ptr = %p, *ptr = %d\n", ptr, *ptr);
   *ptr = 10;
   printf("num = %d, &num = %p\n", num, &num);
   return 0;
}
```

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

Output:

num = 3, &num = 1024

ptr = 1024, *ptr = 3

num = 10, &num = 1024

Indirection Operator – Example 2

```
/* example to show the use of pointers */
#include <stdio.h>

int main(void)
{
    int num1 = 3, num2 = 5;
    int *ptr1, *ptr2;

    ptr1 = &num1; // put the address of num1 into ptr1
    printf("num1 = %d, *ptr1 = %d\n", num1, *ptr1);

    (*ptr1)++; /* increment by 1 the content of the
                  memory location pointed to by ptr1 */
    printf("num1 = %d, *ptr1 = %d\n", num1, *ptr1);

    ptr2 = &num2; // put the address of num2 into ptr2
    printf("num2 = %d, *ptr2 = %d\n", num2, *ptr2);
```

Code continues in next slide …

Output:

num1 = 3, *ptr1 = 3
num1 = 4, *ptr1 = 4
num2 = 5, *ptr2 = 5

```
    *ptr2 = *ptr1; /* copy the content of the location
                      pointed to by ptr1 into the
                      location pointed to by ptr2 */
    printf("num2 = %d, *ptr2 = %d\n", num2, *ptr2);

    *ptr2 = 10;  /* 10 is copied into the location
                      pointed to by ptr2 */
    num1 = *ptr2; /* copy the content of the memory
                      location pointed to by ptr2
                      into num1 */
    printf("num1 = %d, *ptr1 = %d\n", num1, *ptr1);

    *ptr1 = *ptr1 * 5;
    printf("num1 = %d, *ptr1 = %d\n", num1, *ptr1);

    ptr2 = ptr1; // address in ptr1 copied into ptr2
    printf("num2 = %d, *ptr2 = %d\n", num2, *ptr2);

    return 0;
}
```

Output:

num2 = 4, *ptr2 = 4
num1 = 10, *ptr1 = 10
num1 = 50, *ptr1 = 50
num2 = 10, *ptr2 = 50

Indirection Operator – Example 2

Statement	num1 (addr = 1024)	num2 (addr = 2048)	ptr1	ptr2
int num1 = 3, num2 = 5;	3	5		
int *ptr1, *ptr2;	3	5	?	?
ptr1 = &num1;	3	5	1024	?
(*ptr1)++;	4	5	1024	?
ptr2 = &num2;	4	5	1024	2048
*ptr2 = *ptr1;	4	4	1024	2048
*ptr2 = 10;	4	10	1024	2048
num1 = *ptr2;	10	10	1024	2048
*ptr1 = *ptr1 * 5;	50	10	1024	2048
ptr2 = ptr1;	50	10	1024	1024

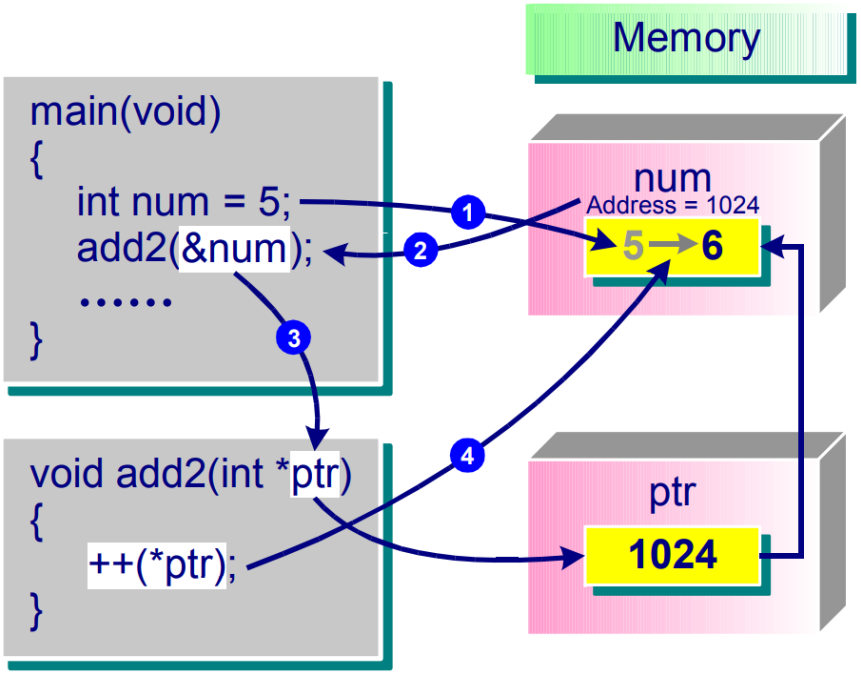
Call by Reference

- Parameter passing between functions has two modes:
  - Call by value
  - Call by reference
- **Call by reference**: The parameter of a function holds the **address** of the argument variable, i.e. the parameter is a **pointer**.
- Therefore, a change to the value pointed to by the parameter **changes** the argument value (instantly).

Call by Reference – Example 1

```
#include <stdio.h>
void add2(int *ptr);
int main(void)
{
    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

Call by Reference – Example 2

```
#include <stdio.h>
void function1(int, int);
void function2(int, int);
void function3(int, int);
int main(void) {
   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) */
}
```

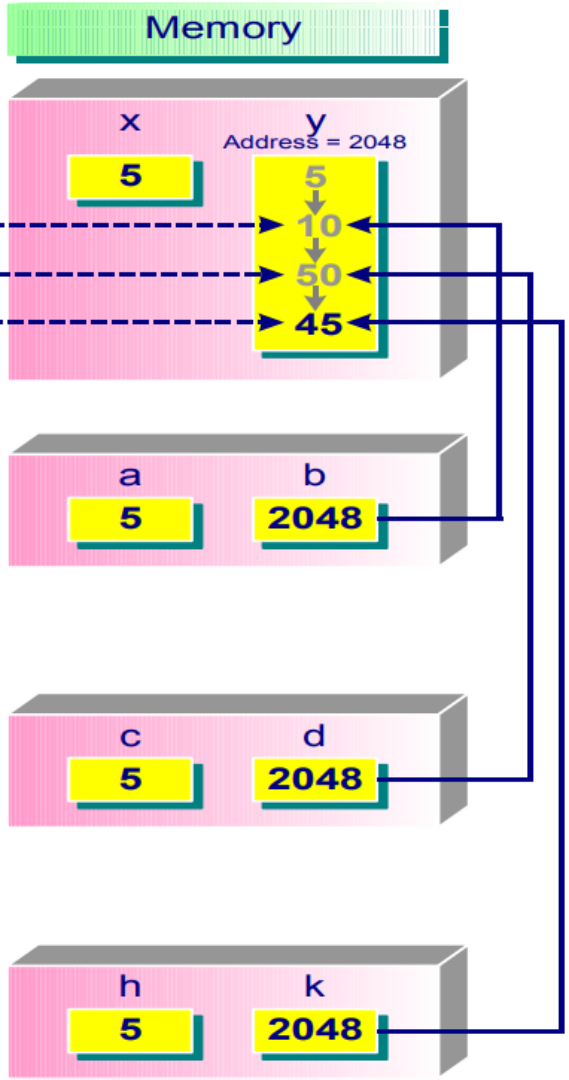
Call by Reference – Example 2

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

```
void function1(int a, int *b)
{
    *b = *b + a;        5 + 5
    function2(a, b);
}
```

```
void function2(int c, int *d)
{
    *d = *d * c;        10 * 5
    function3(c, d);
}
```

```
void function3(int h, int *k)
{
    *k = *k - h;        50 - 5
}
```



Call by Reference – Example 2

	x	y	a	*b	c	*d	h	*k	remarks
(i)	5	5	-	-	-	-	-	-	
(ii)	5	5	5	5	-	-	-	-	
(iii)	5	10	5	10	-	-	-	-	
(iv)	5	10	5	10	5	10	-	-	
(v)	5	50	5	50	5	50	-	-	
(vi)	5	50	5	50	5	50	5	50	
(vii)	5	45	5	45	5	45	5	45	
(viii)	5	45	5	45	5	45	-	-	
(ix)	5	45	5	45	-	-	-	-	
(x)	5	45	-	-	-	-	-	-	

When to Use Call by Reference

- When you need to pass more than one value **back** from a function.

- When using *call by value* will result in a large piece of information being **copied** to the parameter, e.g. passing large arrays. In such cases, for the sake of **efficiency**, we'd better use call by reference.

Recap

- The following concepts have been covered in this lecture:
  - Address Operator
  - Pointer variables
  - Indirection Operators
  - Call by Reference