CSE 06131223 ♦ CSE 06131224

# Structured Programming

## Lecture 21

### Pointers in C (2)


*Prepared by*

**Md. Mijanur Rahman, Prof. Dr.**

Dept. of Computer Science and Engineering

**Jatiya Kabi Kazi Nazrul Islam University, Bangladesh**

www.mijanrahman.com

# Contents

## Pointers in C

# Accessing a Variable Through Its Pointer

- How to access the value of the variable using the pointer?
- This is done by using another another unary operator * (asterisk), usually known as the **indirection operator.** Consider the following statements:

```
int x, *ptr, y;
x = 10;
ptr = &x;
y = *ptr;
*ptr = 25;
```

- The first line declares the variables, the second line assigns the value to a variable x, the third line assigns the address of x to the pointer variable p, and in fourth line, *p returns the value of the variable x, then assigns to the variable y. The fifth line puts the value of 25 at the address pointed by ptr.

# Accessing a Variable…

- Consider the following statements:

```
int x, *ptr, y;
x = 10;
ptr = &x;
y = *ptr;
*ptr = 25;
```

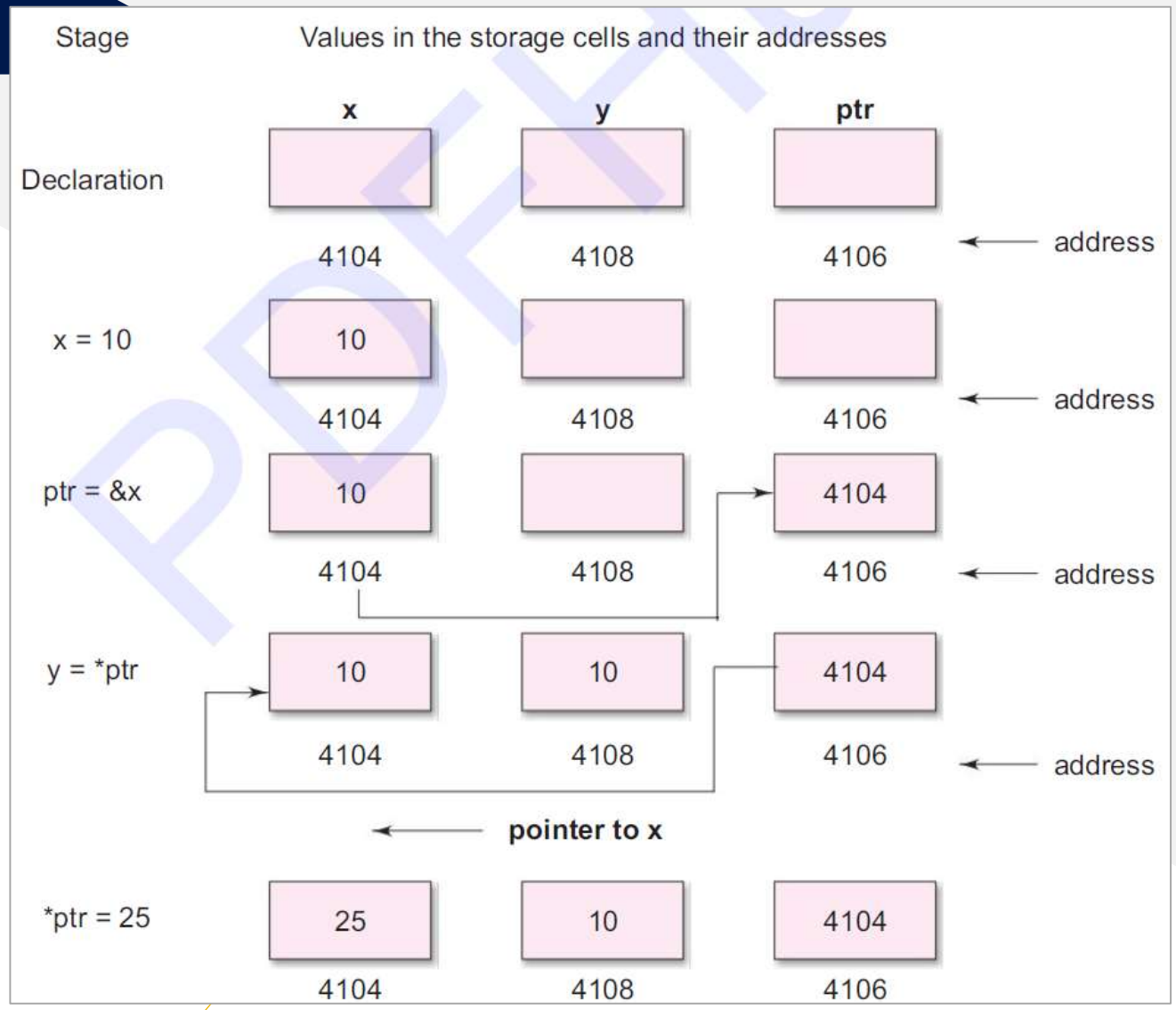



Fig: Illustration of pointer assignments

# Accessing a Variable…

- C Program for illustrating pointer assignments:

```c
#include <stdio.h>
int main() {
    int num1 = 10;
    int num2 = 20;

    int *ptr1, *ptr2;
    // Assign the addresses of num1 and num2
    ptr1 = &num1;
    ptr2 = &num2;
     // Print the values of num1, num2, ptr1, and ptr2
    printf("Num1 = %d, Num2 = %d\n", num1, num2);
    printf("ptr1 = %p, ptr2 = %p\n",ptr1, ptr2);
    printf("*ptr1 = %d, *ptr2 = %d\n",*ptr1, *ptr2);

    // Assign the values of num1 and num2
    *ptr1 = num2;
    *ptr2 = num1;
    // Print the updated values
    printf("After pointer assignments:\n");
    printf("num1 = %d, num2 = %d\n", num1, num2);
    printf("ptr1 = %p, ptr2 = %p\n", (void *)ptr1, (void *)ptr2);
    printf("*ptr1 = %d, *ptr2 = %d\n",*ptr1, *ptr2);
    return 0;
}
```

Terminal

```
Num1 = 10, Num2 = 20
ptr1 = 0x7fff20bda650, ptr2 = 0x7fff20bda654
*ptr1 = 10, *ptr2 = 20
After pointer assignments:
num1 = 20, num2 = 20
ptr1 = 0x7fff20bda650, ptr2 = 0x7fff20bda654
*ptr1 = 20, *ptr2 = 20
```

# Pointer Expressions

- Like other variable, pointer variables can be used in expressions. For example if p1 and p2 are declared and initialized pointers, then the following statements are valid.

```
y = *p1 *  *p2;
sum = sum +  *p1;
z = 5 * -  *p2/ *p1;
*p2 =  *p2  + 10;
```

- Not that there is a blank space between / and * in line 3. But the following is wrong:

```
z = 5* - *p2  /*p1;
```

-

# Pointer Expressions

- C Program for illustrating pointer expressions:

```c
#include <stdio.h>
int main() {
    int num1 = 10;
    int num2 = 5;
    int *ptr = &num1;
    int *ptr2 = &num2;

    // Pointer arithmetic expressions
    printf("*ptr: %d\n", *ptr);
    printf("*ptr + 1: %d\n", (*ptr + 1));
    printf("*ptr - 1: %d\n", (*ptr - 1));
    printf("*ptr * 2: %d\n", (*ptr * 2));
    printf("*ptr / 2: %d\n", (*ptr / 2));
    printf("*ptr %% 3: %d\n", (*ptr % 3));
    printf("*ptr + *ptr2: %d\n", (*ptr + *ptr2));
    printf("*ptr / *ptr2: %d\n", (*ptr / *ptr2));
    printf("*ptr * *ptr2: %d\n", (*ptr * *ptr2));
    printf("*ptr - *ptr2: %d\n", (*ptr - *ptr2));
    return 0;
}
```

Terminal

```
*ptr: 10
*ptr + 1: 11
*ptr - 1: 9
*ptr * 2: 20
*ptr / 2: 5
*ptr % 3: 1
*ptr + *ptr2: 15
*ptr / *ptr2: 2
*ptr * *ptr2: 50
*ptr - *ptr2: 5
```

# Pointer Increment and Scale Factor

- The pointers can be incremented as:

```
p1 = p2 + 2;
p1 = p1 + 1;
```

- An expression like:

```
p1++;
```

will cause the pointer ptr to point the next value of its type. If ptr is an interger with an initial value, 28000, then after the operation p1=p1 + 1, the value of p1 will be 28002, but not 28001.

# Pointer Increment and Scale Factor

- Rules of Pointer Operations:

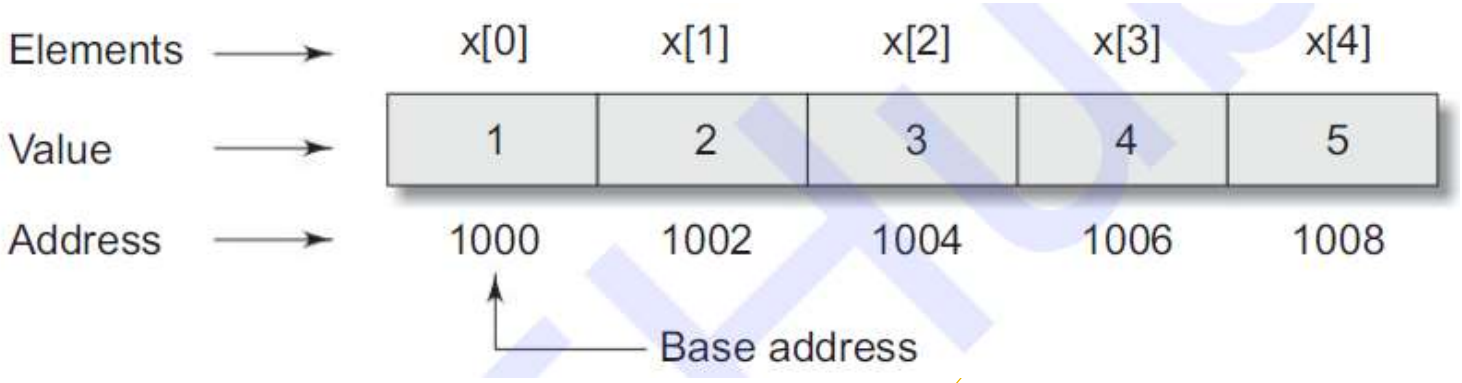
The following rules apply when performing operations on pointer variables.

1. A pointer variable can be assigned the address of another variable.
2. A pointer variable can be assigned the values of another pointer variable.
3. A pointer variable can be initialized with NULL or zero value.
4. A pointer variable can be pre-fixed or post-fixed with increment or decrement operators.
5. An integer value may be added or subtracted from a pointer variable.
6. When two pointers point to the same array, one pointer variable can be subtracted from another.
7. When two pointers point to the objects of the same data types, they can be compared using relational operators.
8. A pointer variable cannot be multiplied by a constant.
9. Two pointer variables cannot be added.
10. A value cannot be assigned to an arbitrary address (i.e., &x = 10; is illegal).

# Pointers and Arrays

- When an array is declared, the compiler allocates a base address and sufficient amount of storage to contain all elements of the array in contiguous locations.
- The base address is the location of the first element (index 0) of the array. The compiler also defines the array name as a constant pointer to the first element. Suppose we declare an array x as follows:

```
int x[5] = {1, 2, 3, 4, 5};
```



# Pointers and Arrays

- The name p is defined as a constant pointer pointing the first element, x[0]. That is,

  p = &x[0] (= 1000)

- This can be assigned as:

  p = x;

- The relationship between p and x is shown as:

```
p = &x[0] (= 1000)
p+1 = &x[1] (= 1002)
p+2 = &x[2] (= 1004)
p+3 = &x[3] (= 1006)
p+4 = &x[4] (= 1008)
```

- The address of element is calculated using its index and the scale factor of the data type. For instance,

  address of **x[3]** = based address + (3 x scale factor of **int**)

  = 1000 + (3 x 2) = 1006

# Pointers and Arrays

- Here's a C program demonstrating pointer increment and scale factor using array of elements:

```
Terminal
Initial array elements:
array[0] = 10
array[1] = 20
array[2] = 30
array[3] = 40
array[4] = 50

Pointer increment and accessing elements:
Value at ptr = 10
Address of ptr = 0x7fff7f341b30
Value at ptr = 20
Address of ptr = 0x7fff7f341b34
Value at ptr = 30
Address of ptr = 0x7fff7f341b38
Value at ptr = 40
Address of ptr = 0x7fff7f341b3c
Value at ptr = 50
Address of ptr = 0x7fff7f341b40
```

```c
#include <stdio.h>
int main() {
    int array[] = {10, 20, 30, 40, 50};
    // Pointer to the first element of the array
    int *ptr = array;
    printf("Initial array elements:\n");
    for (int i = 0; i < 5; i++) {
        printf("array[%d] = %d\n", i, array[i]);
    }

    printf("\nPointer increment and accessing elements:\n");
    for (int i = 0; i < 5; i++) {
        printf("Value at ptr = %d\n", *ptr);
        printf("Address of ptr = %p\n", ptr);
        ptr++;
    }
    return 0;
}
```

# Pointers and Arrays

- C Program using pointers to compute the sum of all elements stored in an array.

```
Terminal
Elements in the array:
10
20
30
40
50
Sum of all elements in the array: 150
```

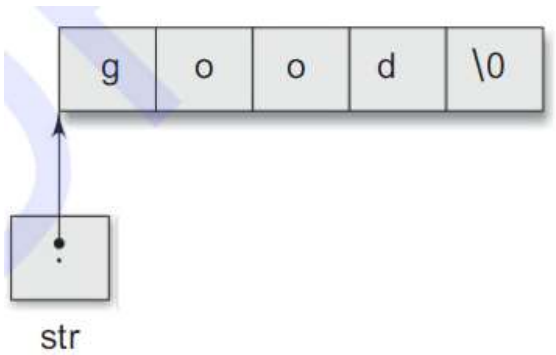
```c
#include <stdio.h>
int main() {
    int array[] = {10, 20, 30, 40, 50};
    int *ptr = array; // Pointer to the first element of the array
    int sum = 0;
    printf("Elements in the array:\n");
    for (int i = 0; i < 5; i++) {
        printf("%d\n", *ptr);
        sum += *ptr;
        ptr++;
    }
    printf("Sum of all elements in the array: %d\n", sum);

    return 0;
}
```

# Pointers and Character Strings

- In C, strings are represented as arrays of characters, terminated by a null character \0. Pointers are commonly used to work with strings, allowing efficient access to individual characters and enabling various string manipulation operations.
- Consider the following character strings:

  char str[5] = “good”’
- C supports an alternative method to create strings using pointer variables of type **char.** For example:

  char *str = “good”;



# Pointers and Strings

- Here's a simple example that demonstrates the usage of pointers with character strings:

```c
#include <stdio.h>
int main() {
    char str_array[] = "Hello, Array String!";
    char *str_ptr = "Hello, Pointer String!";

    // Printing the strings using array indexing
    printf("String using array indexing:\n");
    for (int i = 0; str_array[i] != '\0'; i++) {
        printf("%c", str_array[i]);
    }

    // Printing the strings using pointer arithmetic
    printf("\nString using pointer arithmetic:\n");
    while (*str_ptr != '\0') {
        printf("%c", *str_ptr);
        str_ptr++;
    }
    return 0;
}
```

Terminal

```
String using array indexing:
Hello, Array String!
String using pointer arithmetic:
Hello, Pointer String!
```

# Array of Pointers

- An array of pointers in C is an array where each element is a pointer to another data type. It's commonly used to store addresses of other variables or to create arrays of strings.

- Here's a simple example demonstrating the concept:

```c
#include <stdio.h>
int main() {
    int *ptrArray[3];
    int num1 = 10, num2 = 20, num3 = 30;

    // Assign the addresses of variables
    ptrArray[0] = &num1;
    ptrArray[1] = &num2;
    ptrArray[2] = &num3;

    printf("Values using pointers:\n");
    for (int i = 0; i < 3; i++) {
        printf("Value %d: %d\n", i + 1, *ptrArray[i]);
    }
    return 0;
}
```

Terminal

```
Values using pointers:
Value 1: 10
Value 2: 20
Value 3: 30
```

# Array of Pointers

- Pointers are commonly used in handling tables of strings in C, particularly when dealing with arrays of character pointers.
- This approach allows us to manage and manipulate strings efficiently.

- Here's an example demonstrating the concept:

```c
#include <stdio.h>
int main() {
    // Table of strings
    char *strings[] = {"Hello", "world", "from", "C", "programming"};

    // Calculate the number of strings in the array
    int num_strings = sizeof(strings) / sizeof(strings[0]);

    printf("Strings in the array:\n");
    for (int i = 0; i < num_strings; i++) {
        printf("%s\n", strings[i]);
    }
    return 0;
}
```

Terminal

```
Strings in the array:
Hello
world
from
C
programming
```

# THE END