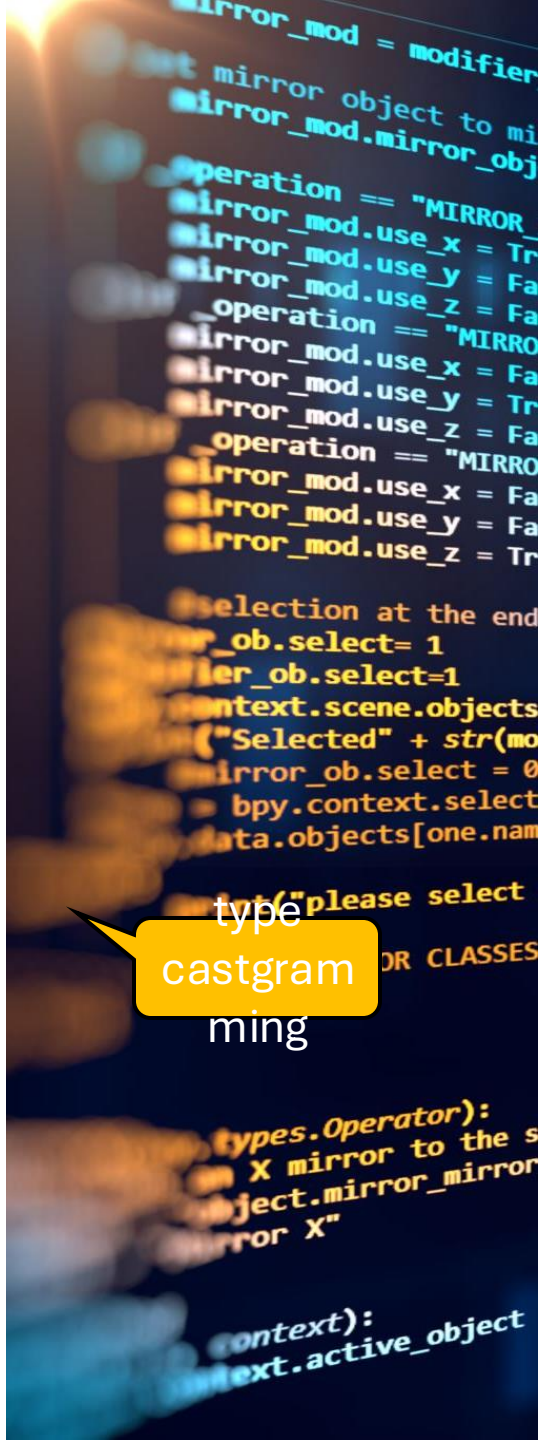
CS 1037
Fundamentals of Computer Science II

# C Programming Features

Ahmed Ibrahim



# Recall: Pass-by-Reference Example

```c
#include <stdio.h>

void swap(int *a, int *b) {
    int temp = *a;
    *a = *b;
    *b = temp;
}

int main() {
    int x = 5, y = 10;
    swap(&x, &y);  // Pass the addresses of x and y
    printf("x = %d, y = %d\n", x, y);
    return 0;
}
```

A pointer that refers to the memory location of a variable.

# Pointers

- A pointer is a **variable** that holds another variable's **memory address** at runtime.
- They are a unique and core feature of the C programming language.
- Pointers give C the power to perform memory address operations.

# Pointers (cont.)

- Declaring a Pointer
  - The syntax for declaring a pointer is **data_type *ptr_name**;
  - ***** indicates that **ptr_name** is a pointer variable.
  - **data_type** specifies the <u>type of data</u> the pointer will reference.
  - Ex. **int *ptr;**
    - Here, * tells the compiler that ptr is a pointer-type variable, and **data_type** specifies that it will store the address of **int** variable.
- Assign an Address
  - The syntax for assigning an address to a pointer is: **int x; ptr = &x;**
  - It is called that **ptr** references to x, or **ptr** points to x.

# Dereferencing a Pointer

- Dereferencing a pointer is an operation to get the value stored at the memory location pointed by the pointer.
- Ex.: **int x; ptr = &x; int y = *ptr;**
- The expression ***ptr_name** tells the compiler to generate instructions to:
  - get the address value stored in ***ptr_name**, then
  - get the value stored at the memory location at the address given in ***ptr_name**.
- Notation ***ptr** is called dereferencing **ptr**.
- Using Dereferenced Pointers:
  - Get the value at the memory location (**int** y = ***ptr**;).
  - Assign a value to the memory location (***ptr** = value;).

C (C17 + GNU extensions)
known limitations

```c
int main() {
  int x = 10;
  int *ptr = &x;

  int y = *ptr;
  *ptr = 50;

  return 0;
}
```

DEMO

Remember: In C programming, ***ptr_name** represents the value or data pointed by **ptr_name**.

```c
#include <stdio.h>

int main() {
    int x = 10;          // Declare a variable
    int *ptr = &x;       // Declare a pointer and store the address of 'x'

    // Display the value of 'x' and the address stored in the pointer
    printf("Value of x: %d\n", x);
    printf("Address of x: %p\n", &x);

    // Display the address stored in the pointer and the value it points to
    printf("Pointer ptr holds the address: %p\n", ptr);
    printf("Value pointed by ptr: %d\n", *ptr);

    return 0;
}
```

To display an *address* value

Output:

```
Value of x: 10
Address of x: 0x7ff7bfeff218
Pointer ptr holds the address: 0x7ff7bfeff218
Value pointed by ptr: 10
```

```c
#include<stdio.h>
int main() {
    int x = 1890259661;
    printf("Value of x is %d\n", x);
    printf("Runtime memory address of x in Hex is %p\n", &x);
    printf("Runtime memory address of x in decimal is %lu\n", &x);
    printf("Value stored at address %lu is %d\n", &x, *(&x));
    return 0;
}
```

To display an *address* value in decimal

Output:

```
Value of x is 1890259661
Runtime memory address of x in Hex is 0065FE9C
Runtime memory address of x in decimal is 6684316
Value stored at address 6684316 is 1890259661
```

# Notes

1. A pointer must reference a **valid** memory location before **dereferencing**, or it will cause a runtime error.
   - Example of incorrect use: `int *ptr; *ptr = 20; // causes a runtime error as ptr holds no valid address.`
2. Dereferencing is less efficient than using the variable name directly because it involves additional step.
3. The <u>pointer size is the same for all pointer types</u>, but the type ensures proper data retrieval during dereferencing.
   - Example of type mismatch: `float x = 10.0; int *ptr = &x; // will fail to compile.`
4. Pointer values are not part of the program's application data but are runtime intermediate data, which can differ across executions and systems

# Pointer Operations

# Dereferencing Operation

- The dereferencing operation is to get the value pointed by a pointer.
- The assignment operation assigns the value of one pointer or address of a variable to another pointer of the same type.

- The following code shows examples of dereferencing and assignment operations:

```c
#include <stdio.h>

int main() {
    int num1=2, num2= 3, sum=0, mul=0;
    int *ptr1, *ptr2, *ptr3;
    ptr1 = &num1;          // address assignment, ptr1 is pointing to num1
    ptr2 = &num2;          // address assignment, ptr2 is pointing to num2
    ptr3 = ptr1;           // assign ptr1 to to ptr2, ptr3 is pointing to num1
    sum = *ptr1 + *ptr2; // dereferencing ptr1 and ptr2, the value of sum will be 5 at runtime
    mul = sum * *ptr3;   // dereferencing ptr3, the value of mul will be 10 at runtime
    }
```

# Assignment Operation

- The dereferencing operation can also be used to set values at the memory location pointed by a pointer.

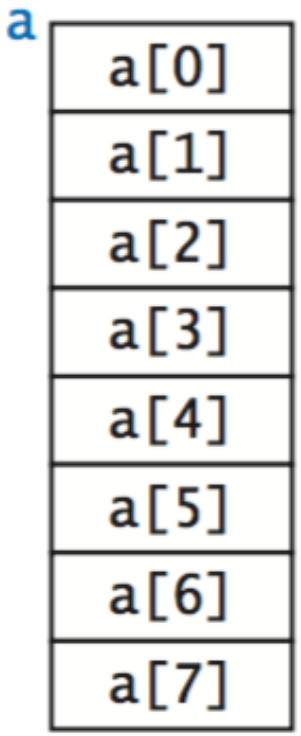
```c
#include <stdio.h>

int main() {
    int num;
    int *ptr;
    ptr = &num;      // address assignment, ptr is pointing num
    *ptr = 2;        // num will have value 2 at runtime
    }
```

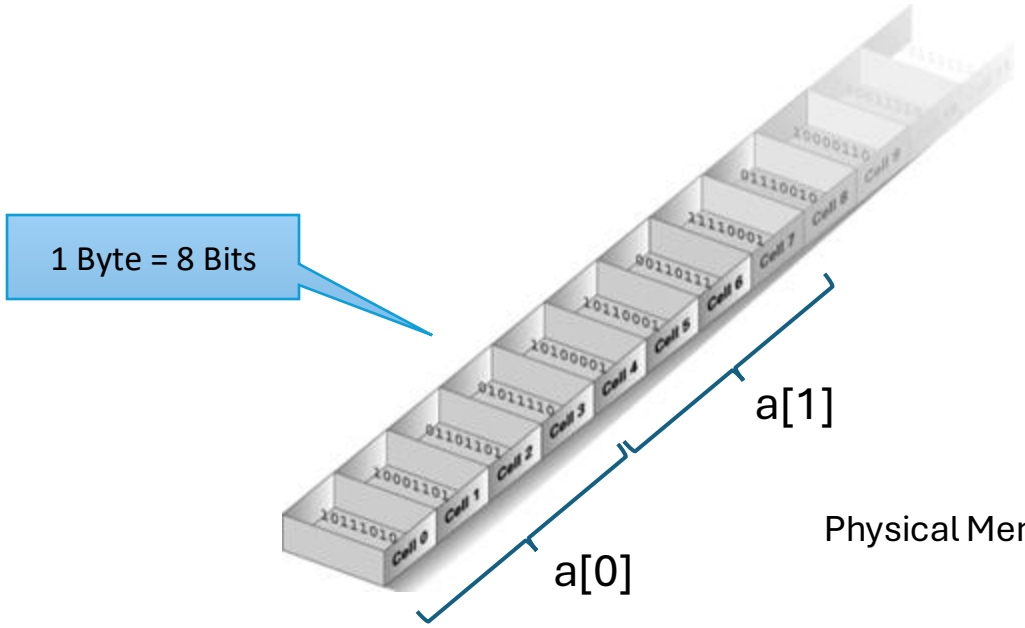
# Arrays

- Arrays are fundamental data structures that store a collection of the same type of data values.
- Pointers provide an efficient tool for array operations.

# Arrays (cont.)




Array of Integers




Physical Memory Layout

# Increase and Decrease Operations

- Pointers support **addition** and **subtraction** operations by adding or subtracting an integer value **k**, adjusting the address by **k** times the size of the data type.
  - Example: **ptr1 - 1** subtracts 4 (size of an int) from ptr1, so **ptr1 - 1** points to a previous variable (e.g., num2).
- Unary increment (++) and decrement (--) operators are supported for pointers.
  - Example: ptr1++ is equivalent to ptr1 = ptr1 + 1.
  - **The amount of increment or decrement is determined by the size of the pointer type**.
  - Example: char *p; p++ increases p by 1 (size of char), while float *p; p++ increases p by 4 (size of float)
- Unary increment (++) and decrement (--) operators have higher precedence than the dereference operator (*).
  - Example: **ptr++** is equivalent to *(**ptr++**), meaning the pointer is incremented before dereferencing.

```c
#include <stdio.h>

int main() {
    // Declare an array of integers
    int arr[5] = {10, 20, 30, 40, 50};

    // Declare a pointer to an integer
    int *ptr;

    // Initialize the pointer to point to the first element of the array
    ptr = arr;

    // Use pointer to access and manipulate array elements
    for(int i = 0; i < 5; i++) {
        printf("Element %d: %d\n", i, *(ptr + i));
    }

    return 0;
}
```

To print *Int* value on display

Output:

```
Element 0: 10
Element 1: 20
Element 2: 30
Element 3: 40
Element 4: 50
```

# Comparison operations

- Pointer comparisons are supported by using relational operators.
  - For example, ptr1 > ptr2, ptr1 == ptr2 and ptr1 != ptr2 are all valid in C.


thank you

# References


- Data Structures Using C, second edition, by Reema Thareja, Oxford University Press, 2014.