IEEE RAS
ROBOTICS & AUTOMATION SOCIETY
Zagazig student chapter

TASK 4&5

Embedded Systems Track

DATE :

23/4/2025

PRESENTED BY :

MOHAMED KADRY

Table Of Content

1 POINTER

1. Declaring and Initializing Pointers in C

To create a pointer, use the * operator:

```
int x = 10;
int *ptr = &x;
```

- ptr holds the address of variable x.

2. Size of a Pointer Variable

- The size of a pointer depends on the architecture (commonly 4 bytes in 32-bit, 8 bytes in 64-bit systems).

```
printf("%lu", sizeof(int*));  // e.g., prints 8 on 64-bit
```

3. Referencing and Dereferencing Pointers

- Referencing (&): Gets the address of a variable.
- Dereferencing (*): Accesses the value at the address stored in the pointer.

```
int x = 20;
int *ptr = &x;
printf("%d", *ptr);  // prints 20
```

4. Performing Arithmetic Operations on Pointers

- Pointers can be incremented, decremented, or used in expressions.

```
int arr[3] = {1, 2, 3};
int *ptr = arr;
ptr++;  // moves to next element (arr[1])
```

1 POINTER

5. Access and Manipulate Values using Pointers

Pointers allow direct manipulation of memory:

```
int x = 5;
int *p = &x;
*p = 10;  // x becomes 10
```

6. Understand Passing by Reference vs Passing by Value in Functions

- Pass by value: A copy is passed; original variable is unchanged.
- Pass by reference: A pointer is passed; function can modify the original variable.

```
void modify(int *p) { *p = 20; }
```

7. Understanding Pointer Typecasting

Used to convert one pointer type to another:

```
void *ptr;
int a = 100;
ptr = &a;
printf("%d", *(int*)ptr);
```

1 POINTER

All Pointer Types in C (10 Essential Types)

Here are the 10 most important pointer types in C:

🔹 1. Null Pointer

Initialized to NULL; points to nothing.

```
int *ptr = NULL;
```

🔹 2. Void Pointer

Generic pointer to any type; needs typecasting.

```
void *ptr;
```

🔹 3. Wild Pointer

Uninitialized pointer; risky to use.

```
int *ptr;
```

🔹 4. Dangling Pointer

Points to freed or out-of-scope memory.

```
free(ptr);
```

🔹 5. Function Pointer

Stores the address of a function.

```
int (*fptr)(int, int);
```

1 POINTER

🔹 6. Pointer to Pointer

Points to another pointer.

```
int **pp;
```

🔹 7. Array of Pointers

Each element is a pointer.

```
char *arr[] = {"A", "B"};
```

🔹 8. Pointer to Array

Points to a whole array.

```
int (*ptr)[5];
```

🔹 9. Constant Pointer & Pointer to Constant

- const int *ptr – value constant
- int *const ptr – address constant
- const int *const ptr – both constant

🔹 10. Pointer to Structure

Access struct members using ->.

```
struct S { int x; };
struct S s = {10};
struct S *p = &s;
p->x;
```