6th Task in Embedded Systems

- **Pointers and Arrays in Structures:**

  A pointer in a structure is used to reference memory addresses instead of storing data directly. This is useful when working with dynamic memory (malloc) or large data.

```
struct Student {
char *name;                    // pointer to name string
};
```

  An array in a structure holds multiple values of the same type directly in memory.

```
struct Student {
char name[50];                 // array of characters for name
};
```

- **Passing Structures to Functions:**
  - By value: Makes a copy ,safe but uses more memory.
  - By pointer: Sends the address , more efficient & allows modification of original data.

```
void printAnimal(struct Animal a);        // by value

void modifyAnimal(struct Animal *a);      // by pointer
```

- **Size of Structure:**

  The size of a struct is not always equal to the sum of the size of its members because of memory padding.

```
struct variable {
char a;              // 1 byte
int b;               // 4 bytes
};
```

- **Memory Padding, Aligned Memory, and Unaligned Memory:**
  -Memory Padding: Extra bytes added by the compiler between members to align data for faster access.
  -Aligned Memory: Data is stored at addresses that match the CPU's word boundary.
  -Unaligned Memory: Data is packed tightly without gaps may cause slow access or crash on some hardware.

- **Difference Between Structure and Object**

	Structure	Object
Contains	Only data (variables)	Data + Functions
Encapsulation	Not supported	Supported
Inheritance	Not supported	Supported
Polymorphism	Not supported	Supported
Memory Model	Simple procedural	OOP uses class-based model

- A struct is just a container for data.
- An object (from a class) is a combination of **data** and the **functions** that operate on it.