4th Task in Embedded Systems

1. Declaring and Initializing Pointers

Pointers are variables that store memory addresses of other variables.

```
int num = 10;                 // Integer variable
int *ptr = &num;              // Pointer to integer initialized with address of num
printf ("Address stored in ptr: %p\n", ptr);  // Prints memory address of num
printf ("Value pointed by ptr: %d\n", *ptr);  // Prints 10 (value of num)
```

2. Size of a Pointer Variable

Pointer size depends on the system architecture (typically 4 bytes on 32-bit, 8 bytes on 64-bit systems).

```
printf ("Size of int pointer: %zu bytes\n", sizeof(int*));     // Usually 4 or 8
printf ("Size of char pointer: %zu bytes\n", sizeof(char*));   // Same size
printf ("Size of double pointer: %zu bytes\n", sizeof(double*)); // Same size
```

3. Referencing and Dereferencing

& (address-of) gets a variable's memory address

* (dereference) accesses the value at a pointer's address

```
int value = 50;
int *p = &value;                        // Referencing (getting address)
printf ("Original value: %d\n", value);  // 50
printf ("Pointer value: %d\n", *p);      // 50 (dereferencing)
*p = 100;                               // Changing value through pointer
printf ("New value: %d\n", value);      // Now 100
```

4. Pointer Arithmetic

Pointers can be incremented/decremented based on their data type size.

```
int arr[3] = {10, 20, 30};
int *ptr = arr;                          // Points to first element
printf ("First element: %d\n", *ptr);    // 10
ptr++;                                   // Moves to next int (4 bytes typically)
printf ("Second element: %d\n", *ptr);   // 20
ptr++;
printf ("Third element: %d\n", *ptr);    // 30
```

5. Access and Manipulate Values

Pointers allow indirect access to memory locations.

```
float temperature = 36.5;
float *tempPtr = &temperature;
printf ("Current temp: %.1f\n", *tempPtr);  // 36.5
*tempPtr = 37.2;                            // Change value via pointer
printf ("New temp: %.1f\n", temperature);   // 37.2
```

6. Pass by Value vs Pass by Reference

By passing the address of a variable to a function, you allow that function to access and modify the actual variable stored at that address. This is very powerful, as it gives the function direct access to the original data in memory.

```
void update(int *x) {
     *x = 100;                  // Modifies the original variable
      }
int main() {
    int num = 50;
    update(&num);               // Passes the address of 'num'
    printf ("%d", num);         // Output: 100 (changed)
    }
```