1)Callback Functions

In embedded C, callback functions are a powerful mechanism that enables a lower-level function (often a driver, library, or interrupt service routine) to indirectly invoke a function defined in a higher-level module (typically the application layer). This is achieved by passing the address (pointer) of the higher-level function as an argument to the lower-level function.

Implementation of Callback Function:

-Function Pointer Declaration: A function pointer is a variable that stores the memory address of a function. It has the same syntax as a function prototype, but instead of the function name, an asterisk (*) is used to indicate the pointer type.

-Passing the Callback Function: The application code defines the callback function, and its address (obtained using the & operator) is passed as an argument to the lower-level function

-Callback Invocation: When the lower-level function encounters a situation where the application's intervention is required (e.g., an event occurs, data is ready), it dereferences the stored function pointer and executes the callback function:

Advantages of Callback Functions:

-Modular Design: Callback functions promote modularity by separating the low-level event handling from the application-specific logic. This leads to cleaner, more maintainable code.

-Event-Driven Programming: They enable event-driven programming, where the application's response is triggered by specific events handled by the lower-level module. This makes the code more reactive and efficient in handling asynchronous operations.

-Flexibility: The application has control over how to handle events, allowing for customization and adaptation to various scenarios.

Disadvantages of Callback Functions:

-Complexity: Callback functions can introduce complexity, especially in larger projects with many interconnected modules. Careful design and documentation are crucial to avoid confusion.

-Debugging Challenges: Debugging can be trickier, as the flow of execution can jump between different modules when callbacks are invoked. Using print statements or debuggers strategically is essential.

-Potential for Errors: Improper pointer handling (e.g., passing a null pointer, dereferencing an invalid pointer) can lead to runtime errors or crashes.

2)Difference between wild and dangling pointers

Wild Pointers:

Definition:

A wild pointer is a pointer variable that has not been initialized to a valid memory address. It essentially contains garbage data that might point to anywhere in memory.

Cause:

Occurs when a pointer is declared but not assigned a proper memory location using mechanisms like malloc, calloc, or by pointing it to a variable.

Behavior:

Accessing or dereferencing a wild pointer results in undefined behavior. The program might crash, access invalid memory locations, or exhibit unpredictable results.

Dangling Pointers:

Definition:

A dangling pointer initially points to a valid memory location, but that memory has been deallocated using free or has gone out of scope (in the case of local variables).

Cause:

-Occurs when memory pointed to by the pointer is deallocated using free before the pointer itself is updated to point elsewhere.

-A pointer variable holding the address of a local variable continues to exist even after the local variable goes out of scope, making the pointer point to invalid memory.

Behavior:

Dereferencing a dangling pointer leads to undefined behavior similar to wild pointers. The program might crash, corrupt data, or exhibit unpredictable behavior.