Sameera-Upadhyaya

Q1. What is the purpose of the `main()` function in a C program? Explain its significance.
The `main()` function in a C program serves as the **starting point** for execution. When a C program runs, the operating system looks for the `main()` function to begin execution. It is essential and must be present in every C program, as it defines the control flow and may call other functions. Typically, it returns an integer value (usually 0) to indicate successful execution to the operating system. It can also accept arguments like `int argc, char *argv[]` to handle command-line inputs. Overall, the `main()` function provides structure to the program and determines what operations are performed when the program runs.

Q2. Explain the difference between a variable declaration and a variable initialization in C.
In C, **variable declaration** is the process of telling the compiler the name and type of a variable, without necessarily assigning it a value. For example, `int x;` is a declaration that creates a variable `x` of type `int`. On the other hand, **variable initialization** is when a value is assigned to a variable for the first time, either at the time of declaration or afterward. For instance, `x = 10;` is initialization. You can also declare and initialize in one line like `int x = 10;`. Declaration reserves memory, while initialization stores data in that memory.

Q3. What are the different data types available in C? Provide examples of each data type.
C provides several fundamental data types used to define variables and manage data. The **basic data types** include `int` for integers (e.g., `int age = 25;`), `float` for single-precision floating-point numbers (e.g., `float pi = 3.14;`), `double` for double-precision floating-point numbers (e.g., `double salary = 12345.67;`), and `char` for single characters (e.g., `char grade = 'A';`). Additionally, there is the `void` type, which signifies the absence of value, often used in function declarations (e.g., `void display();`). C also supports **derived types** such as arrays (`int arr[5];`), pointers (`int *ptr;`), structures (`struct Person { ... };`), unions, and enumerations (`enum Days {MON, TUE};`). These types allow for more complex data organization and manipulation.

Q4. Explain the concept of type conversions in C. Provide examples of implicit and explicit type conversions.
In C, **type conversion** refers to converting a variable from one data type to another. This can happen either **implicitly** or **explicitly**. **Implicit type conversion**, also known as **type promotion**, is performed automatically by the compiler when it evaluates expressions containing mixed data types. For example, in `int a = 5; float b = a;`, the integer `a` is automatically promoted to float during assignment. On the other hand, **explicit type conversion**, or **type casting**, is done manually by the programmer to convert a variable to a specific type using cast syntax. For example, `float x = 10.75; int y = (int)x;` will convert the float value `x` to an integer, storing `10` in `y`. Type conversions are important for data precision, compatibility, and correct program behavior.

Q5. What is the role of the `scanf()` function in C? Provide an example of its usage.
The `scanf()` function in C is used for **input**—it allows the user to enter data from the keyboard, which is then stored in variables. It is part of the standard input/output library (`stdio.h`). The function uses **format specifiers** to determine the type of input, such as `%d` for integers, `%f` for floats, and `%c` for characters. Importantly, the variables passed to `scanf()` must be provided using the address-of operator (`&`) so the function can store the input values at the correct memory location. For example, in the code `int age; scanf("%d", &age);`, the user enters an integer which is stored in the `age` variable. This function is essential for making interactive programs that accept user input.