**Module – 2**

**OVERVIEW OF C**

**PROGRAMMING**

1. **Theory:**

**1). Write an essay covering the history and evolution of C programming. Explain its importance and why it is still used today?**

**Ans. C Programming is one of the most influential and widely used programming languages in computing history. Developed in the early 1970s, C has played a crucial role in shaping modern software development. It serves as the foundation for many contemporary programming languages & remains relevant due to its efficiency, flexibility, and powerful features. This essay explorers the history, evolution, and enduring importance of C programming.**

**Evolution: As C gained popularity, variations of the languages emerged, leading to compatibility issues. To address this, the American National Standards Institutes (ANSI) established a standardization version of C in 1989, known as ANSI C (or C89). This standardization ensured that C programs could be compiled and executed across different systems without significant modifications.**

**In 1990, the International Organization for Standardization (ISO) adopted C89, leading to further refinements in later versions, including:**

1. **C99 (1999): Introduced new data types, inline functions, and improved support for floating-point arithmetic.**
2. **C11 (2011): Added features like multithreading support, improved memory handling, and better security features.**
3. **C18 (2018): Primarily focused on bug fixes and minor refinements rather than major languages changes**

**Importance and Modern Usage**

**C remains widely used today for several reasons:**

**Performance and Efficiency: C provides direct memory access and minimal runtime overhead, making it ideal for performance-critical applications such as operating systems, embedded systems, and game development.**

**Portability: Since C is a compiled language, it can run on virtually any hardware platform with minimal modifications, making it a preferred choice for cross-platform development.**

**Foundation for Other Languages: Many modern programming languages, such as C++, Java, Python, and C#, are influenced by C. Understanding C provides a solid foundation for learning these languages.**

**System-Level Programming: C remains the dominant language for writing operating systems (such as Linux and Windows components), device drivers, and embedded software in areas like robotics, telecommunications, and automotive systems.**

**Embedded Systems: Many microcontrollers and embedded devices run software written in C due to its efficiency and close-to-hardware capabilities.**

**Lab Exe:**

* **Research and provide three real-world applications where C programming is extensively used, such as in embedded systems, operating systems, or game development.**

**Ans. C Programming is extensively used in various domains due to its efficiency, performance, and low-level access to memory. Here are three real-world applications where C is widely used are as follows:**

1. **Embedded Systems:**

* **C is the dominant language for programming embedded systems, such as microcontrollers, automotive systems, medical devices, and IOT devices.**
* **e.g.: The firmware in an automobiles ECU (Engine Control Unit) is often written in C to ensure real-time performance & hardware interactions.**

1. **Operating Systems:**

* **Most operating system are either written in C or have significant portions of their kernel system utilities implemented in C.**
* **e.g.: The Linux kernel is predominantly written in C, allowing direct hardware manipulation, process management & memory handling.**

1. **Game Development:**

* **C is used in game engines for its performance & control over system resources.**
* **E.g.: The Unreal Engine (one of the most powerful game engines) has its core components implemented in C++, but C is often used for performance –critical parts.**

1. **Theory**

* **Describe the steps to install a C compiler (e.g., GCC) and set up an Integrated Development Environment (IDE) like DevC++, VS Code, or Code Blocks.**

**Ans. Install a C Complier (GCC)**

**GCC (GNU Complier Collection) is a popular C compiler. You can install it based on your operating system.**

**Windows:**

1. **Download MinGW (Minimalist GNU for windows)**

* **Go to MinGW -64 or install via MSYS2.**
* **If using MSYS2, download it from msys2.org.**
* **Run the installer & follow the setup instructions.**

1. **Add GCC to System Path**

* **After installation, add c:\MinGW\bin (or the relevant directory) to the system environment variable PATH.**
* **Open Command Prompt (cmd) & type gcc –version to verify installation.**

1. **Choose & Set Up an IDE**

**An IDE provides tools like code editing, debugging, & compiling.**

**Option 1: Dev-C++**

* **Download Dev-C++ from Source Forge.**
* **Install & launch it.**
* **Go to Tools > Compiler Options, ensure that MinGW/GCC is selected.**
* **Create a new project & start coding.**

**Option 2: Code::Blocks**

* **Download Code::Blocks from codebloks.org.**
* **Choose the version with MinGW (if not installed separately).**
* **Install & open Code::Blocks.**
* **Configure the compiler.**
  + **Go to Settings > Compiler**
  + **Ensure GNU GCC Compiler is selected.**

**Option 3: Visual Studio Code (VS Code)**

* **Download VS Code from code.visualstudio.com.**
* **Install the C/C++ extensions from the market place.**
* **Install the Code Runner extension (optional).**
* **Configure the compiler:**
  + **Open settings.json & set up task to run gcc for compiling & running.**

**Option 4: Debugging & Running**

* **Use the built-in debugger in the IDE.**
* **For Command line debugging, use GDB.**

**3. Basic Structure of C Program**

**Theory Exe:**

* **Explain the basic structure of C program, including headers, main function, comments, data types, and variables. Provide examples**

**Ans: A C program consists of several fundamental components that provide structure and functionality. This includes headers, the main function, comments, data types, and variables. Below is an explanation of each with examples.**

1. **Headers**

* **Headers contain preprocessor directives, typically starting with #include, to include libraries.**
* **Common headers:**
  + **#include <stdio.h> (for input/output operations)**
  + **#include <stdlib.h> (for memory allocation, exit functions)**
  + **#include <math.h> (for mathematical operations)**

1. **Main Functions**

* **Every C program must have a main() functions, which serves as the entry point.**
* **Its return type is usually int, and it may take command-line arguments (int argc, char \*argv[ ]).**

1. **Comments**

* **Single line comments: // This is a comment**
* **Multi line comments:** 
  + **/\* This is a multi-line comment\*/**

1. **Data Type**

**C provides several built-in data types:**

* **int (integer)**
* **float (floating-point number)**
* **double (double-precision floating-point number)**
* **char (single character)**
* **void (used for functions returning nothing)**

1. **Variables**

* **Variables store values and must be declared before use.**
* **Declaration syntax: <data\_types> <variable\_name>;**
* **Example: int age;**

1. **Operators in C**

**Theory Exe:**

* **Write notes explaining each type of operator in C: arithmetic, relational, logical, assignment, increment/decrement, bitwise, and conditional operators.**

**Ans: Types of operators in c are as follows:**

1. **Arithmetic Operators**

**These operators are used for basic mathematical operations:**

* **+ (Addition): Add two numbers.**
* **- (Subtraction): Subtracts one number from another.**
* **\* (Multiplication): multiplies two numbers.**
* **/ (Division): Divides one from another.**
* **% (Modulus): Returns the remainder of a division.**

1. **Relational Operators**

**These are used to compare two values & return a Boolean result (true or false).**

* **== (Equal to): Checks if two values are equal.**
* **!= (Not equal to): Checks if two values are not equal.**
* **> (Greater than): Checks if the left value is greater.**
* **< (Less than): Checks if the left values is smaller.**
* **>=(Greater than or equal to): Checks if the value is greater than or equal to the right.**
* **<=(Less than or equal to): Checks if the value is left value is less than or equal to the right.**

1. **Logical operator**

**These are used to combine or modify Boolean expressions:**

* **&& (logical AND): True if both conditions are true (eg: a && b).**
* **|| (Logical OR): True if at least one condition is true (eg: a || b).**
* **! (Logical NOT): Negates a condition (eg: !a).**

1. **Assignments Operators**

**These assign value to variables.**

* **= (Assign): Assigns a value to variable (eg: a=b).**
* **+= (Add & Assign): Adds & then assigns (eg: a += b is a=a+b).**
* **-= (Subtract & Assign): Subtract & Assign (eg: a -=b).**
* **\*= (Multiply & Assign): Multiplies & Assigns (eg: a\*= b).**
* **/= (Divides & Assign): Divides & Assigns (eg: a /= b).**
* **%= (Modulo & Assign): Modulo & Assigns (eg: a %= b).**

1. **Increment & Decrement Operators**

**Used to increase & decrease a value by 1.**

* **++ (Increment): Increase a value by 1. Can be prefix (++a) or postfix (a++).**
* **-- (Decrement): Decreases a value by 1. Similarly, it can be prefix (--a ) or postfix (a--).**

1. **Bitwise Operators**

**Operate on the binary representation of numbers:**

* **& (Bitwise AND): Performs AND operation bit by bit.**
* **| (Bitwise OR): Performs OR operation bit by bit.**
* **^ (Bitwise XOR): Performs XOR operation bit by bit.**
* **~ (Bitwise NOT): Flips bits (complement).**
* **<< (Left shift): Shifts bits to the left.**
* **>> (Right shift): Shifts bits to the right.**

1. **Conditional (Ternary) Operator**

**A concise way to write an if-else statement:**

* **Syntax: condition ? value\_if\_true : value\_if\_false;**
* **Example: a > b ? a : b (returns a if a > b, otherwise returns b).**

1. **Control Flow Statements in C**

**Theory Exe:**

* **Explain decision-making statements in C (if, else, nested if-else, switch). Provide examples of each.**

**Ans: Decision making statements in C allows a program to execute different code blocks based on certain conditions. The main decision-making statement includes:**

1. **If statement**
2. **If-else statement**
3. **Nested if-else statement**
4. **Switch statement**
5. **If statement**

**The if statement evaluates a condition and executes a block of code only if the condition is true.**

**Example:**

**#include <stdio.h>**

**main()**

**{**

**int n;**

**printf("Please Enter a number:");**

**scanf("%d",&n);**

**if (n>=0)**

**{**

**printf("The number is positive.\n");**

**}**

**}**

1. **If-else statement**

**The if-else statement provides an alternative path execution when the if condition is false.**

**Example:**

**#include <stdio.h>**

**main()**

**{**

**int n;**

**printf("Please Enter a number:");**

**scanf("%d",&n);**

**if (n>=0)**

**{**

**printf("The number is Positive.\n");**

**}**

**else**

**{**

**printf("The number is Negative.\n");**

**}**

**}**

1. **Nested if-else statement**

**A nested if-else statement is an if-else statement inside another if or else block.**

**Example:**

**#include <stdio.h>**

**main()**

**{**

**int n;**

**printf("Please Enter a number:");**

**scanf("%d",&n);**

**if (n>0)**

**{**

**printf("The number is Positive.\n");**

**}**

**else**

**{**

**if(n<0)**

**{**

**printf("The number is Negative.\n");**

**}**

**else**

**{**

**printf("The number is ZERO.\n");**

**}**

**}**

**}**

1. **Switch Case**

**The switch case statement allows selecting one of many code blocks to execute based on variables value.**

**Example:**

**#include<stdio.h>**

**main()**

**{**

**char choice;**

**float num1, num2, result;**

**printf("Add (+)");**

**printf("\nSub (-)");**

**printf("\nMul (\*)");**

**printf("\nDiv (/)");**

**printf("\nSelect Your Choice(1-4):");**

**scanf("%c",&choice);**

**printf("\nEnter Number 1:");**

**scanf("%f",&num1);**

**printf("Enter Number 2:");**

**scanf("%f",&num2);**

**switch (choice)**

**{**

**case '+':**

**result=num1+num2;**

**printf("\nAdd:%.2f",result);**

**break;**

**case '-':**

**result=num1-num2;**

**printf("\nSub:%.2f",result);**

**break;**

**case '\*':**

**result=num1\*num2;**

**printf("\nMul:%.2f",result);**

**break;**

**case '/':**

**result=num1/num2;**

**printf("\nDiv:%.2f",result);**

**break;**

**default:**

**printf("\nINVALID CHOICE! Please Select Between (1-4)");**

**}**

**}**

1. **Looping in C**

**Theory Exe**

* **Compare and contrast while loops, for loops, and do-while loops. Explain the scenarios in which each loop is most appropriate.**

**Ans.**

1. **While loops**

* **The condition is checked before the loop executes**
* **If the condition is true, the loop executes; otherwise , it terminates**
* **It is possible for the loop body to never execute if the condition is initially false.**

1. **For loop**

* **The initialization runs once before the loop starts.**
* **The condition is checked before each iteration.**
* **The update expression execute after each iteration.**
* **If the condition is false, the loop terminates.**

1. **Do while loop**

* **The loop body executes at least once, even if the condition is false.**
* **After the first iteration, the condition is checked.**
* **If the condition is true, the loop repeats.**

1. **Use a for loop when the number of iteration is known in advance.**
2. **Use a while loop when the number of iteration is not known and depends on condition.**
3. **Use a do while loop when the loop must execute at least once.**
4. **Loop Control Statements**

**Theory Exe**

* **Explain the use of break, continue, and goto statements in C. Provide examples of each.**

**Ans.**

1. **Break Statement: The break statement is used to immediately exit a switch statement or a loop (for, while, do while). Execution resumes at the first statement after the loop or switch.**

**Example:**

**#include<stdio.h>**

**main()**

**{**

**int i=1;**

**for(i=1;i<=10;i++)**

**{**

**if(i==6)**

**{**

**break;**

**}**

**printf("\n%d",i);**

**}**

**}**

1. **Continue Statement: The continue statement is used to skip the remaining code in the current iteration of a loop and move to the next iteration.**

**Example:**

**#include<stdio.h>**

**main()**

**{**

**int i=1;**

**for(i=1;i<=10;i++)**

**{**

**if(i==4)**

**{**

**continue;**

**}**

**printf("\n%d",i);**

**}**

**}**

1. **Go to Statement: The go to statement transfers control to a labeled statement within the same function. It is generally discouraged as it can make code harder to understand & maintain.**

**Example:**

**#include<stdio.h>**

**main()**

**{**

**int i=1;**

**Bhagirath:// Label**

**printf("\n%d",i);**

**i++;**

**if(i<=10)**

**{**

**goto Bhagirath;**

**}**

**}**

1. **Function in C**

**Theory Exe**

* **What are functions in C? Explain function declaration, definition, and how to call a function. Provide examples.**

**Ans. A function in C is a block of code that performs a specific task. Functions allow for modular programming, making the code easier to read, maintain, and reuse.**

1. **Function Declaration: A function must be declared before it is used. The declaration (also called a prototype) tells the compiler about the function name, return type, and parameters.**

**Example: int add(int, int); // Function declaration**

1. **Function Definition: The function definition contains the actual implementation of the function.**

**Example: return\_type function\_name(parameter)**

**{**

**// Function body**

**return value; // If return type is not void**

**}**

1. **Function Call: A function is executed when it is called from the main() function or another function.**

**Example: int result = add(5, 3); // Function call**

1. **Array in C**

**Theory Exe**

* **Explain the concept of arrays in C. Differentiate between one-dimensional and multi-dimensional arrays with examples.**

**Ans. Concepts of Array in C: An array in C is a collection of elements of the same data type stored in contiguous memory locations. Array allows easy manipulation of large amounts of data using a single name and index values.**

**Difference between one dimensional & multi-dimensional array is as follows:**

**One Dimensional Array:**

* **Stores elements in a single row & column.**
* **Using a single index to access elements.**
* **Linear structure (list).**
* **Allocates memory in a single continuous block.**
* **Used for storing simple lists like marks, prices, temperature, etc.**

**Multi-Dimensional Array:**

* **Stores elements in multiple rows & columns (matrices, table, etc.).**
* **Uses multiple indices (e.g. two for 2D, three for 3D).**
* **Tabular or matrix-like structure.**
* **Allocates memory in a multi-layered block structure.**
* **Used for matrices, game boards, image processing, etc.**

**One dimensional array**

**Example:**

**A one dimensional (1D) array is a simple list of elements of the same type.**

**Multi-dimensional array**

**Example:**

**A multi-dimensional array is an array, where data is stored in a structured format (rows and columns).**

1. **String in C**

**Theory Exe**

* **Explain string handling functions like strlen(), strcpy(), strcat(), strcmp(), and strchr(). Provide examples of when these functions are useful.**

**Ans. C does not have a built-in string data type like other high-level languages. Instead, strings are handled as arrays of characters terminated by a null character (\0). The <string.h> library provides several functions for string manipulation.**

1. **strlen() – String Length**

**Computes and returns the length of the string (excluding the null terminator \0).**

**Example: You need to determine the length of a string e.g. for buffer allocation or validation.**

1. **strcpy() – String Copy**

**Copies the source string into the destination string, including the null terminator (\0).**

**Example: You need to duplicate a string e.g. copying user input into a buffer.**

1. **strcat() – string concatenation: Appends the source string to the destination (overwriting \0 in destination).**

**Example: When you need to combine two strings, e.g. formatting output messages or constructing file path.**

1. **strcmp() – Compares two strings lexicographically (alphabetical Order).**

* **Returns 0 if str1==str2.**
* **Returns <0 if str1 is less than str2.**
* **Returns >0 if str1 is greater than str2.**

**Example: When you need to compare user input, sort strings, or check for string equality.**

1. **strchr() – Find character in string**

* **Find the first occurrence of the character ch in str.**
* **Returns a pointer to the found character or NULL if not found.**

**Example: When you need to search for a character in a string, e.g. parsing input data.**