**Module-2**

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

C is a general-purpose procedure-oriented programming language, c is a high-level programming language developed by donnish Ritchie in the early 1970. It is most popular and influential programming languages worldwide

C programming, though a classic language, remains a cornerstone of software development due to its power, efficiency, and portability. While newer languages have emerged, C continues to be vital in areas demanding low-level control, performance optimization, and system-level programming, as well as serving as a foundational language for countless other languages.

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

To Install a compiler, first download and install IDE like Dev C++, then install your chosen IDE , next configure the IDE to recognise the compiler. Finally verify the setup by building and running a simple c program.

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

Examples: -

#include<stdio.h> // Header File

main() // main function

{

Int age; // Data types & Variables

// Body

}

Data Types:- int, char, float, double, longdouble.

Variables:- In C programming, variables are named storage locations that hold data values, and each variable has a specific type that determines the kind of data it can store.

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

|  |  |
| --- | --- |
| **Operator** | **Examples** |
| **Assignment** | **"=, +=, -=, \*=, /=, %="** |
| **Arithmetic** | **"+ , - ,\* , / "** |
| **Relational** | **"==, !=, <, >, <=, >=, "** |
| **logical** | **&&, || , !** |
| **increment/Decrement** | **"++, --"** |
| **Conditional** | **" ?, : "** |

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

The C language programs presented until now follows a sequential form of execution of statements.

Many times, it is required to alter the flow of the sequence of instructions. C language provides statements that can alter the flow of a sequence of instructions.

These statements are called control statements. These statements help to jump from one part of the program to another. The control transfer may be conditional or unconditional.

If = When you have a only one condition in program then uses only if-else statement.

Nested if= when you have a more condition in one condition than use nested if statement.

Switch case: - When you have many conditions in some program then use switch case.

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

While, for, and do-while loops are fundamental control flow structures in programming, each serving different purposes based on the number of iterations and the condition for loop termination.

While and for loops are entry-controlled, meaning they check the condition before executing the loop body, while do-while is exit-controlled, checking the condition after the body is executed, guaranteeing at least one iteration.

While Loop:

* **Condition:** Checks the condition before each iteration.
* **Termination:** If the initial condition is false, the loop body is never executed.
* **Usage:** Ideal when the number of iterations is not known beforehand, and the loop continues as long as a dynamic condition remains true.
* **Example:** Reading user input until they enter "quit," as the number of iterations is dependent on the user's actions.

For Loop:

* **Condition:**

Involves initialization, condition check, and increment/decrement in a single header.

* **Termination:**

The loop iterates a predetermined number of times, often based on a counter or iterating over elements of a sequence.

* **Usage:**

Best when the number of iterations is known in advance, such as processing an array or performing a task a specific number of times.

* **Example:**

Iterating through the elements of an array or performing a task a fixed number of times.

**Do-While Loop:**

* **Condition:**

Checks the condition after each iteration (exit-controlled).

* **Termination:**

The loop body is executed at least once, even if the condition is initially false.

* **Usage:**

Useful when you need to guarantee a specific action within the loop is performed at least once, regardless of the condition, such as prompting the user for input at least once [1, 2, 8].

* **Example:**

Displaying a menu and allowing the user to choose an option, with the menu being shown at least once, even if the user doesn't enter a valid choice initially.

1. Explain the use of break, continue, and go to statements in C. Provide examples of each.

In C language break, continue, and go to are jump statements used to control the flow of execution within loops and functions. break terminates the loop, continue skips the current iteration and proceeds to the next, and go to jumps to a labelled statement within the same function.

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

functions are reusable blocks of code that perform specific tasks. They enable modularity and code organization. A function declaration informs the compiler about the function's structure (return type, name, parameters), while the function definition provides the actual code that the function executes. A function is called by its name, potentially passing arguments, and it can return a value.

Types of Functions:

* Built in function ( scanf, printf, etc…)
* User Define Function: -the function which are created by user for program are known as User define function.

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

array is a data structure that stores a collection of elements of the same data type in contiguous memory locations. One-dimensional arrays store data in a linear sequence, while multi-dimensional arrays (like two-dimensional arrays) arrange data in a table or matrix format with rows and columns.

1. Explain what pointers are in C and how they are declared and initialized. Why are pointers important in C?
2. Explain string handling functions like strlen(), strcpy(), strcat(), strcmp(), and strchr(). Provide examples of when these functions are useful
3. Explain the concept of structures in C. Describe how to declare, initialize, and access structure members.
4. Explain the importance of file handling in C. Discuss how to perform file operations like opening, closing, reading, and writing files.