Module 2 – Introduction to Programming

1. Overview of C Programming:

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 a general-purpose, **procedure-oriented programming** language.
- C is a high-level programming language developed by **Dennis Ritchie** in the early **1970s**.
- It is now one of the most popular and influential programming languages worldwide.
- It has powerful features including low-level memory access, a rich set of operators, and a modular framework.

History Of C Language:

- By 1960, number of computer languages has come into existence. people started thinking for general-purpose language. therefor an international committee was formed to develop such a language.
- A new language called CPL(Combined Programming Language) developed at Cambridge.
- The C Language was derived from a language known as BCPL(Basic Combined Programming Language) which involved at USA in the 1967s by Martin Richards. BCPL is derived from CPL.
- Ken Thompson has developed language B as a further simplification of CPL.
- In 1972, Dennis M. Ritchie inherited the features of B and BCPL in c, added some of his own features and developed language C.
- The first major use of the C language was to write an operating system called UNIX.

Created By : Hensi vaghela

Various Stage in evolution of C is as below:

YEAR	LANGUAGE	DEVELOPED BY
1960	ALGOL	International committee
1963	CPL	Cambridge uni.
1967	BCPL	Martin Ricjards
1970	B	Ken Thompson
1972	C	Dennis Ritchie

Importance of c language:

1. It is **simple and powerful**.
2. It helps you **understand how computers work** (like memory and hardware).
3. C++, Java, and Python are many modern language **based on C**.
4. It's used to build **operating systems, device drivers, and fast software**.

C is still used today because:

1. C is still used because it's fast, efficient, and forms the backbone of many systems.
2. C programs can run on many different types of computers with small change because it is portable.
3. Operating systems, embedded systems, and device drivers are still written in C.
4. Many modern languages and tools are built using C.

2. Setting Up Environment:

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

Ans:-

Install a C Compiler (GCC):

- **Download MinGW Installer**

  From: https://sourceforge.net/projects/mingw/

- **Install GCC Packages**

  Run the installer.

  In MinGW Installation Manager, select: mingw32-gcc-g++

  Click **Installation > Apply Changes**.

- **Copy the Bin Path**

  Go to: C:\MinGW\bin

  Copy this path.

- **Add to Environment Variables**

  Right-click **This PC > Properties**

  Click **Advanced system settings > Environment Variables**

  Edit the **Path** variable → Click **New** → Paste: C:\MinGW\bin

  Click **OK**.

- **Verify Installation**

  Open **Command Prompt**

  Type: gcc –version

  You should see the GCC version displayed.

choose and setup IDE:

Option A: DevC++ (Simple and Beginner-Friendly)

- DevC++ is a lightweight and easy-to-use IDE for C/C++ beginners.
- **Step 1:** Download from DevC++ (SourceForge)
- **Step 2:** Install and open the application.
- **Step 3:** No extra setup required — it comes with a built-in compiler (GCC).
- **Best for:** Beginners who want to start coding quickly with less setup.

Option B: Code::Blocks (Good for C/C++ Development)

- Code::Blocks is a professional IDE specially designed for C and C++ programming.
- **Step 1:** Download from Code::Blocks Website
- **Step 2:** Select the version with **mingw-setup** (this includes the compiler).
- **Step 3:** Install and launch Code::Blocks.
- **Step 4:** Ready to use — GCC compiler is already included.
- **Best for:** Students and intermediate users working on C/C++ projects.

Option C: Visual Studio Code (VS Code - Powerful & Flexible)

- VS Code is a modern and powerful text editor that supports many languages including C/C++.
- **Step 1:** Download from VS Code Website
- **Step 2:** Install **GCC or MinGW** (compiler needed separately).
  - You can use MinGW installer to get GCC on Windows.
- **Step 3:** Open VS Code and go to the **Extensions** tab (left panel).
  - Search for and install **C/C++ extension** by Microsoft.
- **Step 4:** Create a .c file, write your code, and use the **Terminal** to compile:

  ```
  gcc filename.c -o filename
  ./filename
  ```

- **Best for:** Advanced users or developers who want full control and flexibility.

3. Basic Structure of a C Program:

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

Ans:-

1.stucture of c program:

1. **Preprocessor Directives**:-- instruction to the compiler to link function from the system library or run time library . a special source file need to include it is known as “Header File”. the preprocessor directive used #include.

2. **main() Function Section**:-- the main() is special function in every c program .it is the entry point in program. it contains the declaration part which declare all variable.

3. **statement & expression**:-- the body of the program containing logic and function which perform a specific task in the program.

4. **Return statement**:-- Ends the main() function, usually returns 0.

5. **comment section**:-- a set of comment lines giving the name of the program ,the author name, and many more details as programmer would like to use later. compiler ignore comment in the program. there are two types of comment:

1. Single line comment: use // to single line comment.

2. Multi-line comment: use start the comment /* and end with */ comment to multi line comment.

2.Datatype:

Datatype defines the **type of data** a variable can hold.

It tells the compiler how much **memory to allocate** for the variable.

Basic data type:

C language support **5** fundamental data type.

1.**Interger**: integer is number with a range of values supported by a computer.

2.**Float**: Float data type is used to store floating point numbers.

3.**Double**: Double data type is also used to store floating point number but it provides 14 digit of accuracy.

4.**Character**: A single character can be defined as a character data type.

5.**Void**: void data type has no values. this is used to specific the type of function.

Data type	Size	Format Specifier	Keyword
Integer	2 Byte	%d	Int
Float	4 Byte	%f	float
Double	8 Byte	%lf	Double
Character	1 Byte	%c	char
Void	--	--	Void

Derived data type:

Type	Description
array	Collection of elements of same type
pointer	Stores address of another variable
structure	Group of different data types
union	Like structure, but memory efficient
function	A block of code to perform task

3.variable:

Defition:-

variable is a **named storage location** in memory that holds a value which can be modified during program execution.

Stores data like numbers, characters, etc. Must be **declared before use**. Value can **change** during program execution.

Declaration does two thing:

1. it tells the compiler what variable name is,
2. it specifies what type of data the variable will hold.

Syntax:

Datatype variable_name;

Many variable of same datatype can also be created using following format:

Datatype v1,v2,v3,….,vn.

Assiging values to variables:

Value can be assign to variable using the assignment operator(=).it takes following format.

Variable_name = Value;

Example:

```
#include<stdio.h>  // Include standard input-output library
#include<conio.h>  //Include console input-output library
void main()
{
    // Variable declarations
    int age = 18; // Integer variable
    float height = 5.9; // Floating-point variable
    char grade = 'A';  // Character variable

    // Output using printf
    printf("Age: %d\n", age);
    printf("Height: %f\n", height);
    printf("Grade: %c\n", grade);

    getch();  // terminate the program
}
```

4. Operators in C:

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

Ans:-

An operator is a **symbol** that tells the compiler to perform specific mathematical or logical operations on data.

1. **Arithmetic Operators:**
    - It is used for **mathematical operation.**

Operator	Meaning
+	Adds two numbers (e.g., 3 + 2 = 5)
-	Subtracts one number from another
*	Multiplies two numbers
/	Divides one number by another
%	Gives the **remainder** after division

2. Assignment Operators:
- It is used to **assign values in variables**.

Operator	Meaning
=	Assigns a value (e.g., a = 10)
+=	Adds and stores result (e.g., a += 5 is a = a + 5)
-=	Subtracts and stores
*=	Multiplies and stores
/=	Divides and stores
%=	Modulus and stores

3. Relational (Comparison) Operators:
- It is used to **compare** values (true/false).

Operator	Meaning
==	Is equal to
!=	Not equal to
>	Greater than
<	Less than

>=	Greater than or equal to
<=	Less than or equal to

4. Logical Operators :

- It is used to **check conditions** (true or false).

Operator	Meaning
&&	AND – true if **both** sides are true
||	OR – true if **least one** condition is true
!	NOT – **opposite** of the condition (true becomes false)

5. Increment and Decrement Operators :

- **it is** used to **increase or decrease** a number by 1.

Operator	Meaning
++	Increases by 1
--	Decreases by 1

6. Bitwise Operators :

- **it** Work on **binary (0s and 1s)**.

Operator	Meaning (Simple)
&	Bitwise AND
|	Bitwise OR
^	Bitwise NOT
~	1's complements
<<	Left shift – moves bits to the left
>>	Right shift – moves bits to the right

7. Ternary (Conditional) Operator :

- Used to **make decisions in one line**.

Operator	Meaning
? :	If-else in short form. Example: a > b ? x : y means: if a > b, return x; else y

5. Control Flow Statements in C:

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

Ans:-

Defition:-

Real life situations where we have to take condition based decisions by asking 'if' questions.

1.if Statement:

the if statement is a control statement that test a particular condition.

Syntax:

```
if (condition)
{
   // code runs if condition is true
}
```

Example:

```
if (age >= 18) {
 printf("You can vote.");
}
```

2. if-else Statement:

The if statement is evaluates the statement when condition is true but it can not provide any way when condition is false. so that if-else statement is used.

Syntax:

```
if (condition){
 // runs if condition is true
} else {
   // runs if condition is false
}
```

Example:

```
if (marks >= 40) {
   printf("Pass");
}
else {
   printf("Fail");
}
```

3. Nested if-else Statement:

When a series of decision are involved, there is a need to use more than one if…else statement, this is called Nested if-else statement.

Syntax:

```
If(test condition1){
        If(test condition2){
                //statement1;
        }
        else{
                //statement2;
        }
}
else{
        //statement3;
}
Statementx;
```

Example:

```
if (number > 0) {
   printf("Positive");
} else {
   if (number < 0) {
      printf("Negative");
   } else {
      printf("Zero");
   }
}
```

4. switch Statement:

Conditional Branching with long if…else statement can often be more efficiency performed using a switch statement.

Syntax:

```
switch (condition)
 {
   case value1:
      // code
      break;
   case value2:
      // code
      break;
   default:
      // code if no case matches
}
```

Example:

```
switch (day)
{
   case 1:
      printf("Monday");
      break;

   case 2:
      printf("Tuesday");
      break;

   default:
      printf("Invalid day");
}
```

6. Looping in C:

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

Ans:-

Defition:-

A loop statement allows us to execute a statement or group of statements multiple times based on a condition.

- The c language provide following loop construct.

1.Entry Controll Loop:

- For Loop
- While Loop

2.Exit Controll Loop:

- do…while Loop

1.For Loop:

- Entry-controlled loop with initialization, condition-checking, and increment /decrement control all in one line.
- Best when the **number of iterations is known** beforehand.

Syntax:

```
for (initialization; condition; increment)
{
   // code block
}
```

Example:

```
 for (int i = 0; i < 5; i++)
{
        printf("%d\n", i);
 }
```

2.While Loop:

- Entry-controlled loop: the condition is evaluated **before** executing the loop body.
- If the condition is false initially, the loop body **may not execute at all**.
- Typically used when the **number of iterations is not known** in advance.

Syntax:

```
while (condition) {
   // code block
}
```

Example:

```
int i = 0;
while (i < 5) {
     printf("%d\n", i);
     i++;
  }
```

3.do…while Loop:

- Exit-controlled loop: the loop body is executed **at least once**, regardless of the condition.
- The condition is evaluated **after** the loop body.

Syntax:

```
do {
   // code block
} while (condition);
```

Example:

```
int i = 0;
do {
     printf("%d\n", i);
      i++;
  } while (i < 5);
```

Scenario	Recommended Loop
Reading input until a sentinel value is entered	while or do-while
Iterating through a fixed list or array	for loop
Running a menu at least once and repeating user choice	do-while loop

7.Loop Control Statements:

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

Ans:-

1. break Statement:

- The Break Statement cases an **immediate exit** for the inner most loop.

Use:

- To stop a loop when a specific condition is met.
- Commonly used in switch-case to prevent fall-through.

Example:

```
#include <stdio.h>
#include <conio.h>

int main()
{
   for (int i = 0; i < 10; i++) {
      if (i == 5) {
         break;  // Exit loop when i is 5
      }
      printf("%d ", i);
   }
   return 0;
}
```

2. continue Statement:

- **Skips the current iteration** of a loop and moves to the next iteration.

Use:

- To skip over specific values or conditions in a loop without exiting the loop entirely.

Example:

```
#include <stdio.h>
#include <conio.h>

int main()
{
   for (int i = 0; i < 10; i++) {
      if (i == 5) {
         continue;  // Skip 5 number
      }
      printf("%d ", i);
   }
   return 0;
}
```

3. goto Statement:

- Transfers control to a **labeled statement** elsewhere in the function.

Use:

- Jumping out of nested loops or skipping to error recovery code.

Example:

```
#include <stdio.h>
#include <conio.h>

int main()
{
   printf("Start of the program.\n");

   goto skip;  // Jump to the label 'skip'

   // This line will be skipped
   printf("This line will be skipped.\n");

skip:  // Label to jump to
   printf("This line is after the goto.\n");

   return 0;
}
```

8.Functions of C:

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.
- A function is a set of code that can be called from any other function.
- Functions are small module of a program which is used to reduce size of main program.

Elements of a Function:

1. Function Declaration (Prototype)
2. Function Definition (Body)
3. Function Call

1.Function Declaration (Prototype):

It tells the compiler about the function's **name**, **return type**, and **parameters before it is used**.

Syntax:

```
return_type  function_name(parameter1_type, parameter2_type, ...);
```

Example:

```
int add(int a, int b);
```

2. Function Definition:

This contains the actual body of the function—**the code that runs when the function is called**.

Example:

```
int add(int a,int b)
{
   return a+b;
}
```

3. Function Call:

To **use** or **execute** the function, you call it from the main() or another function.

Example:

```
int result = add(10, 5);
```

Example:

```
#include <stdio.h>
#include <conio.h>

// Function Declaration (also called Prototype)
int add(int, int);

// Function Definition
int add(int x, int y) {
   return x + y;  // Returns the sum of x and y
}

void main() {
   int a = 3, b = 5, result;

   // Function Call
   result = add(a, b);

   // Output the result
   printf("Sum = %d", result);

   getch();  // Waits for a key press
}
```

Types of Functions:

1. No argument, no return
2. Argument, no return
3. No argument, with return
4. Argument and return

9.Array in C:

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

Ans:-

- In short word,An Array is a collection of element.
- An **array** is a collection of **similar data types** stored in **contiguous memory locations**.
- C support a derived datatype named array for such application.
- An array is a group of related data items that share a common name. for ex, Group of marks of ten student. If name group is marks, then marks of 6^{TH} student is accessed by mentioning the position 6 in the group marks.
- Individual value of array are called **elements**. an array is defined as a set of **homogeneous data item**.
- The element are individually identified by its subscript. the subscript of array is also called an **index.**
- Indexing will be starts from **0**.
- Arrays are **fixed-size** and must be declared before use.
- Efficient for storing and accessing large data of the same type.

Types of Arrays in C:

1. One-Dimensional Array or 1-D Array

2. Multi-Dimensional Array or 2-D Array(3D arrays)

1. One-Dimensional Array (1D Array):

A 1D array is like a list. It's a simple linear structure.

Example:

```
#include<stdio.h>
#include<conio.h>

void main()
{
   int  i;
   int marks[5] = {80, 90, 70, 85, 60};
```

```
   for(i=0;i<5;i++)
   {
      printf("marks[%d] = %d\n", i, marks[i]);
   }
   getch();
}
```

2. Multi-Dimensional Array (2D Array):

A 2D array is like a matrix with rows and columns.

Example:

```
#include<stdio.h>
#include<conio.h>

void main()
{
   int  i, j;
   int matrix[2][3] = {{1, 2, 3},{4, 5, 6}};

   for(i=0;i<2;i++)
   {
      for(j = 0; j < 3; j++)
      {
         printf("matrix[%d][%d] = %d\n", i, j, matrix[i][j]);
      }
   }

   getch();
}
```

Difference Between 1D and Multi-Dimensional Arrays:

Feature	One-Dimensional Array	Multi-Dimensional Array
Structure	Linear list	Table/matrix (rows × columns)
Declaration	int a[5];	int b[2][3];
Access	a[2]	b[1][2]
Use case	Storing list of values	Storing grid/matrix (e.g., marks, games)

10.Pointers in C:

Explain what pointers are in C and how they are declared and initialized. Why are pointers important in C?

Ans:-

- Pointer is a derived data type in c.
- A pointer is a variable that points to or reference a memory location in which data is stored.
- A **pointer** is a **variable that stores the address of another variable**.
- Use * to declare and get value, use & to get address.

Why are pointers important in C?

- C uses **memory addresses** a lot.
  - Access and change values directly in memory
  - Use arrays and strings easily
  - Work with functions and pass data efficiently
  - Create dynamic memory (malloc, calloc, etc.)

Important Pointer Symbols:

***** => Value at the address

& => Address of a variable

Syntax:

```
data_type  *pointer_name
```

1) Pointer Declaration:

```
int *ptr;   // pointer to int
char *p;    // pointer to char
float *f;   // pointer to float
```

2) Pointer Initialization:

```
int a = 5;
int *ptr;
ptr = &a;  // &a means "address of a"
```

3) Accessing Value Through Pointer:

```
printf("%d", *ptr);  // shows the value of 'a'
```

Example:

```
#include <stdio.h>

#include <conio.h>

int main()

{

   int num = 5;

   int *ptr;

   ptr = &num;     // pointer stores the address of num

   printf("Before: num = %d\n", num);

   *ptr = 20;      // change value of num using pointer

   printf("After: num = %d\n", num);

   return 0;

}
```

11.Strings in C:

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

Ans:-

1. strlen() – String Length:

- Returns the **length** of the string (excluding the null character \0).

Example:

```
#include <stdio.h>
#include <conio.h>
void main()
{
        char name[20];
        printf("\n Enter your name : ");
        scanf(“%s", &name);

        printf("Length of name = %d" ,strlen(name));
        getch();
}
```

Use Case:
To validate input length or allocate memory for string operations.

2. strcpy() – String Copy:

- Copies the content of one string into another.

Example:

```
#include <stdio.h>
#include <conio.h>
#include <string.h>
void main()
{
        char a[20]="apple";
         char b[20];
```

```
            printf("\n Enter name: ");
            scanf("%s", b);

             printf("\n You entered: %s",b);
             strcpy(b, a);
             printf("\n After strcpy, b=%s",b);
            getch();
}
```

Use Case:
When you want to duplicate a string into another variable.

3. strcat() – String Concatenation:

- Appends one string to the end of another.

Example:

```
include <stdio.h>
#include <string.h>
int main() {
              char str1[50] = "Good ";
              char str2[] = "Morning";

              strcat(str1, str2);
             printf("Concatenated String: %s\n", str1);  // Output: Good Morning
      return 0;
}
```

Use Case:
To merge strings such as first name and last name or build full messages.

4. strcmp() – String Compare:

- Compares two strings **lexicographically**.

Example:

```
#include <stdio.h>
#include <conio.h>
#include <string.h>
void main()
{
            char name1[20]="Hensi";
            char name2[50];
```

```
        do
        {
                printf("\n enter any name : ");
                scanf("%s",&name2);
        }while(strcmp(name1,name2)!=0);
        printf("\n correct name");
        getch();
}
```

Use Case:
To sort strings alphabetically or compare user input with expected text.

5. strchr() – Search Character in String:

- Finds the **first occurrence** of a character in a string.

Example:

```
#include <stdio.h>
#include <string.h>
int main() {
   char str[] = "computer";

   char *ptr = strchr(str, 'p');
   if(ptr)
      printf("Found at position: %ld\n", ptr - str);  // Output: Found at position: 3
   else
      printf("Character not found.\n");

   return 0;
}
```

Use Case:
To locate specific characters (e.g., @ in email addresses).

12.Strctures in C:

Explain the concept of structures in C. Describe how to declare, initialize, and access structure members.

Ans:-

- In C, a **structure** is a **user-defined data type** that allows you to group variables of **different types** under a single name.
- It is used to represent a **record** or a complex data entity, like a student, employee, book, etc.
- Each data element in a structure is called a member, and they can be of different types, such as integers, floats, or arrays.
- They are defined using the `struct` keyword and can be accessed using the dot operator (`.`).

Declaring a Structure:

```
struct Student {
   char name[50];
   int roll;
   float marks;
};
```

Here, Student is a structure with three members:

- name – string (character array)
- roll – integer
- marks – float

Initializing a Structure:

```
struct Student s1 = {"Hensi", 101, 88.5};
```

Accessing Structure Members:

Use the **dot operator (.)** for direct variables:

```
printf("Name: %s\n", s1.name);
printf("Roll: %d\n", s1.roll);
printf("Marks: %.2f\n", s1.marks);
```

13. File Handling in C:

Explain the importance of file handling in C. Discuss how to perform file operations like opening, closing, reading, and writing files.

Ans:-

Importance of File Handling in C:

- **Data Persistence:**
  Keeps data even after the program ends.
- **Large Data Storage:**
  Can handle more data than memory (RAM) allows.
- **Data Sharing:**
  Enables data to be shared between programs.
- **Reuse & Access:**
  Allows reading/modifying stored data easily.

Basic File Operations in C:

- C uses the **FILE** structure (from <stdio.h>) to handle files.

Task	Function
Open	fopen()
Write	fprintf(), fputs()
Read	fscanf(), fgets()
Close	fclose()

1. Opening a File:

```
FILE  *fp;
fp = fopen("filename.txt", "mode");
```

Mode	Description
"r"	Open for reading (must exist)
"w"	Open for writing (create/overwrite)

"a"	Append to file (create if not exist)
"r+"	Read + Write
"w+"	Write + Read (overwrites)
"a+"	Read + Append

2. Closing a File:

Always close the file after work is done.

fclose(fp);

3. Writing to a File:

- fprintf() => formatted output
- fputs() or putw() => string or integer output

```
FILE  *fp = fopen("file.txt", "w");
fprintf(fp, "Hello, World!");
fclose(fp);
```

4. Reading from a File:

- fscanf() => formatted input
- fgets() or getw() => string or integer input

```
char data[100];
FILE *fp = fopen("file.txt", "r");
fgets(data, 100, fp);
printf("%s", data);
fclose(fp);
```