Assignment
Module 2 – Introduction to Programming

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

- **History of C**
    - The history of C programming begins with another language called **B**, which was developed by **Ken Thompson** in 1969 at **Bell Labs**. B was itself based on an earlier language called **BCPL** (Basic Combined Programming Language).
    - However, B had limitations, especially when it came to system programming. So, in **1972**, **Dennis Ritchie**, also at Bell Labs, improved B and created a new language called **C**.
- **evolution**

Year	Version	Features
1966	BCPL	created by Martin Richards
1969	B	developed by Ken Thompson
1972	C	Created by Dennis Ritchie for UNIX
1978	K&R C	First book and guidelines published
1989	ANSI C	Standard rules (portable version)
1990	ISO C	International standardization
1999	C99	New data types, improved declarations

- **importance**
    - The ‘Mother’ of Languages
    - Direct Hardware Access
    - Memory Efficiency
    - Speed
    - Universality

- **<u>why it is still used today?</u>**
    - Fast
    - Reliable
    - Good for hardware
    - Used in big systems
    - Easy to move to other computers
    - Great for learning programming

(2) 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.

Describe the steps to install a C compiler.

Like **DevC++, VS Code**

(1) DevC++

- DevC++ is used to compile c programs.
- DevC++ is software used to writing and running the c programs.
- DevC++ also generates an output file.

The step to install DevC++ compiler.

Step-1: Download DevC++

- This website: https://sourceforge.net/projects/orwelldevcpp/
- Then after download.

Step-2: install DevC++

- Double click the Download file.
- Then after click Next until it`s done.

Step-3: Run your code

- Open DevC++
- Click the file then aften New and again click source file.
- Write the code.
- And then Run your code Click>Execute>compile & Run.

(2) Vs Code

- VS (Visual Stdio) code is free Code editor developed by Microsoft.

- It supports many programming languages like C, C++, Python, Java, JavaScript, and more.

The step to install vs Code compiler.

Step-1: Download Vs code

- This Website: https://code.visualstudio.com
- Then after download.

Step 2: Install GCC Compiler (via MinGW)

1. Download MinGW this website: https://www.mingw-w64.org
2. Install (choose 64-bit).
3. the install location (e.g., C:\mingw-w64).
4. Then after install MinGW add the path in system.
    - Right-click This PC >Properties >Advanced system settings >Environment Variables.

Step 3: Install C Extension in VS Code

- Open VS Code.
- Go to Extensions.
- Search for C/C++ by Microsoft (Optional: code Runner).
- Click install.

Step 4: Write and Run C Code

- Create a new file and save it .c extension (Like main.c).
- Type the code and then after Run.

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

Basic Structure Of c program

```
#include<stdio.h>   // 1. Header file
Void main ()        // 2. Main function
{
        //3. body of program

        getch();        // 4.end of program
}
```

For Example:

```
#include<stdio.h>                   // 1. Header file
#include<conio.h>

Void main ()                        // 2. Main function
{
        //3. body of program
        int age = 20;       // variable of int type
        float marks = 85.78;    // variable of float type
        char grade = 'B';   // variable of char type

        printf("\n Age: %d", age);
        printf("\n Marks: %f", marks);
```

```
    printf("\n Grade: %c", grade);


    getch (); // 4.end of program
}
```

1. Header file

- Header file is like a **library** in c.
- When you include a header file, you get access to functions without writing them yourself.
- For example:
- #include <stdio.h>
    - ✓ This includes the Standard Input Output library.
    - ✓ It uses functions like printf() and scanf().

2. Main Function

- Every C program **must have a main () function**.
- The execution of the program **starts from here**.
- **Example:**

```
Void main ()
{
        // code
}
```

3. Comments

- Comments are used to **explain the code**.
- They are ignored by the compiler.
- Two types:
  - ✓ **Single-line:** // This is a comment
  - ✓ **Multi-line:** /* This is a multi-line comment */

4. Data Types

- There are four type of data type.

Data Type	Keyword	Value Type	Memory Size	Format Specifier
Integer	int	Numeric values (e.g., 1, 2, 3)	4 bytes	%d
Float	float	Decimal values (e.g., 7.5, 2.99)	4 bytes	%f
Character	char	Single characters (e.g., 'A', 'B')	1 byte	%c
Double	double	Decimal values (more precise float)	8 bytes	%lf

5. Variables

- Variables are **names** that store data.
- You must **declare** them before using.
- Syntax:

```
Data_type   var_name ;

Data_type   var_name = 10;  // assign value
```

- Example:

```
Int num=10;
```

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

Types of Operators in C

(1) Arithmetic Operator

- These operators are used to perform **basic mathematical calculations** like addition, subtraction, multiplication, division, and finding the remainder (modulus).

Operator	Meaning	Example
+	Addition	a + b
-	Subtraction	a - b
*	Multiplication	a * b
/	Division	a / b
%	Modulus (remainder)	a % b

(2) Relational Operators

- Relational operators **compare two values** and decide the relationship between them. The result of this comparison is either **true (1)** or **false (0).**

- These operators are commonly used in decision-making and loops.

Operator	Meaning	Example
==	Double Equal to (Comparison the value)	a == b
!=	Not equal to	a! = b
>	Greater than	a > b
<	Less than	a < b
>=	Greater or equal	a >= b
<=	Less or equal	a <= b

(3) Logical Operator

- Logical operators **combine multiple conditions** or expressions and return true or false based on their logical relation.

    - && means logical AND — both conditions are true.
    - || means logical OR — at least one condition is true.
    - ! means logical NOT — These conditions are not true.

Operator	Meaning	Example
&&	Logical AND	(a > 0) && (b > 0)
||	Logical OR	(a > 0) || (b > 0)
!	Logical NOT	!(a > 0)

(4) Assignment Operator (shorthand operators)

- Assignment operators are used to **assign values** to variables.
- The basic assignment operator is =, which assigns the value on the right to the variable on the left.
- There are also called **shorthand operators** which perform an operation and assign the result in one step.
- shorthand assignment operators work as **value replace** the variables.

Operator	Meaning	Example
=	Assign	a = 5
+=	Add and assign	a += 3 (same as a = a + 3)
-=	Subtract and assign	a -= 2 (same as a = a - 2)
*=	Multiply and assign	a *= 4
/=	Divide and assign	a /= 2
%=	Modulus and assign	a %= 3

(5) Increment and Decrement Operators

- These operators are special types of unary operators used to increase or decrease the value.
- Increment operator ++ increases the value by 1, and decrement operator -- decreases it by 1.
- They can be used before or after the variable (prefix or postfix).

Operator	Meaning	Example
++	Increment by 1	a++ or ++a
--	Decrement by 1	a-- or --a

(6) Bitwise Operators

- Bitwise operators work on the binary (bit-level) representation of integers.
- They perform operations like AND, OR, XOR, NOT, and shifting bits left or right.

Operator	Symbol	Description
AND	&	Bits that are 1 in both
OR	|	Bits that are 1 in either
XOR	^	Bits that are 1 in one, but not both
NOT	~	Inverts bits
Left Shift	<<	Shifts bits left
Right Shift	>>	Shifts bits right

(7) Conditional (Ternary) Operator

- The conditional operator, also called the ternary operator, is a shorthand way of writing an if-else statement in a single line.

Syntax:

condition ? expression1 : expression2;

- If the condition is true, expression1 is executed.
- If the condition is false, expression2 is executed.

Example:

```
#include <stdio.h>

int main()
{
    int a = 10, b = 20;
    int max;

    max = (a > b) ? a : b;

    printf("Maximum is: %d\n", max);

    return 0;
}
```

Output:

Maximum is: 20

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

Decision-making statements:

(1) if Statement

- Used to run code **only if a condition is true**.

  Syntax:

```
if (condition) {
        // statement
}
```

  Example:

```
#include <stdio.h>
int main () {
        int age = 18;
        if (age >= 18)
        {
                printf("You are an adult.\n");
        }
return 0;
}
```

(2) if-else Statement

- Used when you want to run one block **if condition is true**, and another **if it is false**.

  Syntax:

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

Example:

```
#include <stdio.h>
int main () {
        int age = 16;
        if (age >= 18)
         {
                printf("You can vote.\n");
        }
        else
         {
                 printf("You cannot vote.\n");
        }
Return 0;
}
```

(3) Nested if-else

- If condition inside if condition is called nested if condition. Used for **multiple conditions**.

  Syntax:

```
if ()
{
        if ()
        {
        }
}
```

Example:

```
#include <stdio.h>
int main () {
        int num;
        printf("Enter a number: ");
        scanf("%d", &num);

        if (num > 0)
        {
                if (num % 2 == 0)
                {
                        printf("The number is positive and even.\n");
                }
                else
                {
                        printf("The number is positive but odd.\n");
                }
        }
        else
```

```
        {
                printf("The number is not positive.\n");
        }
    return 0;
    }
```

(4) switch Statement

- Used when you have **many options(choice)** for one variable.

Syntax:

```
switch(condition)
{
        case 1:
        case 2:
        case 3:
}
```

Example:

```
#include <stdio.h>
int main()
 {
        int day;
        printf("Enter day number (1-3): ");
        scanf("%d", &day);

       switch(day)
       {
```

```
            case 1:
                printf("Monday\n");
                break;
            case 2:
                printf("Tuesday\n");
                break;
            case 3:
                printf("Wednesday\n");
                break;
            default:
                printf("Invalid day\n");
            }

    return 0;
}
```

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

Compare (for loop, while loop, do. While loop)

Feature	for loop	while loop	do-while loop
Syntax	Compact (init; condition; update)	Simple (while (condition))	Similar to a while, but runs first
Entry/Exit Check	Entry-controlled	Entry-controlled	Exit-controlled
When Condition Checked	Before loop starts	Before loop starts	After first execution
Minimum Execution	0 times	0 times	1 time

Loop:

- A **loop** is used to repeat a block of code again and again, until a certain condition is true or false.

- Types Of Loops:

1. Entry Control Loop : for loop, while loop
2. Exit Control Loop : do while loop

1. **For Loop:**

    - Runs a block of code a fixed number of times.

Syntax:

```
for(initialization;condition;incre/decre)
{
}
```

Example:

```
#include <stdio.h>
void main() {
        int i;
        for(i = 1; i <= 5; i++) {
        printf("%d\n", i);
        }
}
```

2. While Loop:

- While loop first check the condition and then execute the block of code**.**

Syntax:

```
initialization;
while(condition)
{
        incre/decre;
}
```

Example:

```
#include <stdio.h>
void main() {
         int i = 1;
         while(i <= 5) {
```

```
        printf("%d", i);
        i++;
        }
}
```

3. Do-While Loop:

- Do while loop is the same as while loop, only one difference is there.
- While loop first check the condition and then execute the block of code, where do while loop first execute the block of code and then check the condition.

Syntax:

```
initialization;
do
{
    incre/decre;
}
while(condition);
```

Example:

```
#include <stdio.h>
void main() {
    int i = 1;
    do {
        printf("%d\n", i);
        i++;
    } while(i <= 5);
}
```

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

There are three Type of Jumping statement:

- Break statement
- Continue statement
- Goto statement

1. Break statement

- The break statement in C is a loop control statement that is used to **immediately terminate** a loop (for, while, or do-while) or to exit a switch statement.
- **Syntax:**

```
break;
```

- **Example:**

```
#include <stdio.h>
void main() {
   int i;
   for(i = 1; i <= 5; i++)
  {
      if(i == 3)
     {
         break;
      }
      printf("%d ", i);
   }
}
```

Output: -

1 2

2. Continue statement

- The continue statement in C is a loop control statement that is used to **skip the current iteration** of a loop and **jump to the next iteration** immediately.

- **Syntax:**

  Continue;

- **Example:**

```
#include <stdio.h>
void main()
{
 int i;
 for(i = 1; i <= 5; i++)
 {
          if(i == 3)
         {
                 continue; // skip printing 3
         }
         printf("%d ", i);
 }
}
```

Output:-

1 2 4 5

3. Goto statement

- The goto statement in C is used to **transfer control** to another part of the program. It jumps to a labelled statement unconditionally.
- **Syntax:**

```
goto label;
...
label:
  statement;
```

- **Example:**

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int n,sum=0,i=0;
    printf("\n enter any number :");
    scanf("%d",&n);

    jump:
        i++;
        sum+=i;
    if(i<n)
        goto jump;

    printf ("\n sum of %d natural numbers = %d", n, sum);

    getch();
}
```

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

Function:

- A function in C is a **block of code** that performs a **specific task**, grouped together under a name.
- It is used to multiple time call.

Function declaration:

- A function declaration tells the compiler:
    - The name of the function
    - The return type
    - The number and type of parameters
- It is written before the main () function.
- So, the compiler knows about the function before it is called.

Syntax:

return_type function_name(parameter_type1, parameter_type2, ...);

Example:

int add (int, int);

- int -> return type (function returns an integer)
- add -> function name
- (int, int) -> two integer parameters

Function definition:

- A function definition provides the actual code of the function.

  It matches the function declaration in:

  - Name
  - Return type
  - Number and type of parameters

Syntax:

```
return_type function_name(parameter_list) {

// Body of the function

// Statements to perform task

}
```

Example:

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

 {

        return a + b;

}
```

- int → return type
- add → function name
- (int a, int b) → parameters
- return a + b; → returns the sum

Function Call

- A function call is the statement that executes a function.

Syntax:

```
function_name(arguments);
```

Example:

```
#include <stdio.h>
int add(int a, int b);          // Declaration
void main() {
    int result;
    result = add(5, 10);        // Function Call
    printf("Sum = %d", result);
}
int add(int a, int b) {         // Definition
return a + b;
}
```

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

Array:

- array is a **collection of elements** of the same data type.
- Array index starts from **0**.
- Array is storing **multiple values** in a single variable.
- There are two types of arrays:
  - one-dimensional array
  - multi-dimensional array

Differentiate:

Feature	One-Dimensional Array	Multi-Dimensional Array
Structure	Linear (like a list)	Tabular (like a matrix or table)
Declaration	int a[5];	int b[3][4];
Access	a[2]	b[1][3]
Use Case	Store single list of items (e.g. marks)	Store matrix-like data (e.g. tables, grids)

One-Dimensional Array

- A **single row** of elements.
- Syntax: data_type array_name[size of array];
- Example:

```
int marks[3] = {85, 90, 78};
printf("%d", marks[1]);     // Output: 90
```

Multi-Dimensional Array (two-Dimensional Array)

- **Rows and columns** (like a table).
- Syntax: int a[2][3];
- Example:

```
int matrix[2][2] = {{1, 2}, {3, 4}};
printf("%d", matrix[1][0]);          // Output: 3
```

```
#include<stdio.h>

void main()
{
        int a[2][3] = {
        {1, 2, 3},
        {4, 5, 6}
        };
        printf("%d",a[1][2]);
}
```

Output: 6

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

Pointer:

- A **pointer** is a variable that **stores the memory address** of another variable.

Symbol	Meaning
*	Value at address
&	Address of a variable

How to Declare a Pointer?

int *ptr;

- int → Type of data the pointer will point to.
- *ptr → Declares ptr as a pointer to an integer.

How to Initialize a Pointer?

int a = 10;

int *ptr = &a;

- a is a normal variable.
- &a gives the address of a.
- ptr stores the address of a.

Why are Pointers Important in C?

1. **Direct Memory Access**: You can access and modify memory directly, which makes programs faster and more efficient.
2. **Function Arguments (Call by Reference)**: Pointers allow functions to modify arguments passed to them.

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

1. strlen() – Returns the **length** of the string.

Example:

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
void main()
{
        char name[50];
        printf("\n enter Name :");
        scanf("%s",&name);
        printf("\n length is %d",strlen(ch));
        getch();
}
```

Output: 6

2. strcpy() – Copies one string into another.

Example:

```
#include <stdio.h>
#include <string.h>
int main()
{
```

```
    char name1[20], name2[] = "Dhiraj";
    strcpy(name1, name2);
    printf("Copied String = %s", name1);
    return 0;
}
```

Output: Dhiraj

3. strcat() – Appends one string to the end of another.

Example:

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
void main()
{
        char name[20],surname[50];

        printf("\n enter name :");
        scanf("%s",name);
        printf("\n enter surname :\n");
        scanf("%s",surname);

        printf("%s",strcat(name,surname));
        getch();
}
```

Outut: divyaparmar

4. strcmp() –Compares two strings alphabetical order).

Example:

```
//string function(strcmp)
#include<stdio.h>
#include<conio.h>
#include<string.h>

void main()
{
        char name[30]="divya",name1[50];
        do
        {
                printf("\n enter name :");
                scanf("%s",&name1);
        }while(strcmp(name,name1)!=0);

        printf("\n Answer is Correct!!");
        getch();
}
```

Output: Answer is Correct!!

5. strchr() – Find Character in String

Example:

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

```c
#include <string.h>

int main()
{
   char str[] = "hello";
   char *ptr = strchr(str, 'l');
   if (ptr != NULL)
      printf("First 'l' at position: %ld", ptr - str);
   return 0;
}

// Output: 2
```

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

- **Structure**
  - Structure is a user defined data type.
  - structure is define using the struct keyword.

- **Structure Declaration**

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

  - struct is the keyword.
  - Student is the structure name.
  - roll, name, and marks are members.

- **Structure Variable Declaration**

```
struct Student
{
        int roll;
        char name[50];
        float marks;
} s1, s2;        // Declaring s1 and s2 variable.
```

- **Initializing Structure Members**

  You can initialize members in two ways:

  Method 1: Direct Initialization

  ```
  struct Student s1 = {101, "Rahul", 89.5};
  ```

  Method 2: Assigning Values Later

  ```
  s1.roll = 101;
  strcpy(s1.name, "Rahul");  // Use strcpy for strings
  s1.marks = 89.5;
  ```

- **Accessing Structure Members**

  Use the **dot operator (.)** with the structure variable:

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

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

Importance of file handling in C

- Permanent Storage
- Large Data Handling
- Data Sharing
- Input/Output Operations

Operation	Function	Purpose
Open	fopen()	Opens file in specified mode
Write	fprintf() fputs() fputc()	Writes data to file
Read	fscanf() fgets() fgetc()	Reads data from file
Close	fclose()	Closes the file

1. Opening a File

- To perform any operation (read, write, or append), the file must first be opened using the fopen () function.

```
FILE *fp;

fp = fopen("filename.txt", "mode");// Opens file in read mode
```

Modes of Opening a File:

Mode	Description
"r"	Read only.
"w"	Write only.
"a"	Append.
"r+"	Read and write.
"w+"	Read and write.
"a+"	Read and append.

2.Writing to a File

- Used to store data into a file using functions like:
  - fprintf() – Like printf() but writes to a file.
  - fputs() – Writes a string to a file.
  - fputc() – Writes a single character.

```
FILE *fp = fopen("output.txt", "w");
fprintf(fp, "Hello, world!\n");
fputs("This is a line.\n", fp);
fclose(fp);
```

3. Reading from a File

- Used to retrieve data from a file using:
  - fscanf() – Like scanf() but reads from a file.
  - fgets() – Reads a line (string) from a file.
  - fgetc() – Reads a single character.

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

4. Closing a File

- Always close a file using fclose():

```
fclose(fp);
```