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 story of C programming starts in **1969** with a language called **B**, made by **Ken Thompson** at **Bell Labs**. B was created from an older language named **BCPL** (Basic Combined Programming Language).
But B had many limitations, especially for system programming. So, in **1972**, another developer at Bell Labs, **Dennis Ritchie**, 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>**

- It is fast and powerful.
- It is reliable and stable.
- Best for hardware programming like operating systems and drivers.
- Used in big systems and embedded systems.
- It is easy to shift to different types of computers.
- Great for learning programming basics.

(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 software used for writing and running C programs.
- It helps you compile your C code and see the output easily.

The step to install DevC++ compiler.

Step-1: Download DevC++

- Visit this website: https://sourceforge.net/projects/orwelldevcpp/
- Download the DevC++ setup file from there.

Step-2: install DevC++

- Double-click the downloaded file.
- Keep clicking **Next** until the installation is complete.

Step-3: Run your code

- Open DevC++.
- Click on **File > New > Source File**.
- Type your C code.
- Click **Execute > Compile & Run** to run your program.

(2) Vs Code (Visual Studio Code)

- VS Code is a free code editor made by Microsoft.
- It supports many programming languages like C, C++, Java, Python, etc.

The step to install vs Code compiler.

Step-1: Download Vs code

- Visit: https://code.visualstudio.com
- Download and install the setup.

Step 2: Install GCC Compiler (via MinGW)

1. Go to: https://www.mingw-w64.org
2. Download and install the 64-bit version of MinGW.
3. Note the installation folder (for example: `C:\mingw-w64`).
4. Add MinGW to your system PATH:
   - Right-click on **This PC > Properties > Advanced system settings > Environment Variables**.
   - In the "Path" section, add the MinGW bin folder.

Step 3: Install C Extension in VS Code

- Open VS Code.
- Go to the Extensions tab (on the left).
- Search for **C/C++ by Microsoft** and install it.
- (Optional: You can also install "Code Runner" extension for easier running.)

Step 4: Write and Run C Code

- Create a new file and save it with a .c extension (e.g., main.c).
- Write your C program.
- Open the terminal in VS Code.
- To compile, type: gcc main.c
- To run, type: ./a.exe

(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>   // Header file

void main()          // Main function
{
   // Body of the program

   getch();       // End of the program
}
```

For Example:

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

void main()           // 2. Main function
{
   // 3. Body of the program

   int age = 20;          // integer variable
   float marks = 85.78;    // float variable
   char grade = 'B';       // char variable

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

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


    getch();          // 4. End of the program
}
```

1. **Header file**

- Header files are like libraries.
- They give access to ready-made functions.
- Example: #include<stdio.h> → lets you use printf() and scanf().

2. **Main Function**

- Every C program starts from the main() function.
- Code inside main() gets executed.

- **Example:**

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

3. **Comments**

- Comments help explain your code.
- They are ignored by the computer.

➢ 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 store data in memory.
- You must declare them before using.
- Syntax:

```
int num;          // only declare
int num = 10;     // declare + assign value  Example:
```

(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

- Used to compare two values.
- Gives **1 (true)** or **0 (false)**.

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:

```
int a = 10, b = 20;
int max;

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

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

Output:

Maximum is: 20

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

Decision-making statements:

(1) **if Statement**

- Runs code **only if** the condition is **true**.

Syntax:

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

Example:

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

(2) if-else Statement

- Runs one block **if true**, and another **if 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 inside another if**.
- Used for checking **multiple conditions**.

.

Syntax:

```
if (condition1) {
if (condition2) {
            // code
} else {
            // code
}
} else {
          // code
}
```

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 there are **many choices** for one variable.

Syntax: switch(variable) {

```
    case 1:
        // code
        break;
    case 2:
        // code
        break;
    default:
        // code
}
```

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.

❖ What is a Loop?

- A loop runs the same block of code **again and again** until a condition becomes **false**.

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:

- Used when you **know how many times** to run.

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:

- Used when the **condition is checked first**.
- Runs **only if** condition is **true**.

Syntax:

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

Example:

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

```
        i++;
    }
}
```

3. Do-While Loop:

- Same as while, but **runs at least once** even if the condition is **false**.

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

- Used to **exit** from a loop or switch immediately.
- Control comes **out of the loop** when break is used.
  - **Syntax:**

    break**;**
  - **Example:**

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

Output: -

1 2

2. Continue statement

- Used to **skip** the current iteration of a loop.
- It goes to the **next loop iteration** without running remaining code inside the loop.

- **Syntax:**

  Continue;

- **Example:**

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

Output:-

1 2 4 5

3. Goto statement

- goto is used to **jump** to another part of the program.
- It uses a **label** to jump to a specific line.

- **Syntax:**

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

- **Example:**

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

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

printf("Sum of %d natural numbers = %d", n, sum);
}
```

(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 does a specific task.
- It helps to use the same code again and again, which makes the program shorter and easier to understand.

Function declaration:

- A function declaration tells the compiler:
- The function's name
- What it will return
- What type and how many inputs (parameters) it takes
- It is written **before the main() function**, so the compiler knows about the function in advance.

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:

- Function definition means writing the full code of the function — what it will do.
- 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

- To run the function, we need to call it by using its name and giving it the required values.

Syntax:

```
function_name(arguments);
```

Example:

```
#include <stdio.h>

int add(int a, int b);  // Function Declaration

void main() {
   int result;
   result = add(5, 10);  // Function Call
   printf("Sum = %d", result);
}

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

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

Array:

- An **array** is a group of elements that all have the **same data type** (like all integers or all floats).
- In C, **array index starts from 0**.
- It is used to **store many values using a single variable** name.
- There are two main 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

- It is like a single row of elements.
- Used when you need to store data in a list form.

- **Syntax:**

  data_type array_name[size];

- Example:

```
int marks[3] = {85, 90, 78};

printf("%d", marks[1]);  // Output: 90
```

- **Multi-Dimensional Array (Two-Dimensional):**
- **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 special 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 that stores value 10.
- &a gives the **address** of variable a.
- ptr stores that **address**.

Now *ptr will give the **value of a**, and ptr holds the **address of a**.

Why are Pointers Important in C?

1. **Direct Memory Access:**

- You can access or change values directly from memory.
- It makes programs **faster and more powerful**.

2. **Function Arguments (Call by Reference):**

- With pointers, a function can change the value of variables passed to it.
- Useful when working with arrays, strings, and large data.

Example:

```
#include <stdio.h>


void main()
{
   int a = 20;
   int *ptr = &a;
   printf("Value of a: %d\n", a);      // Output: 20
   printf("Address of a: %p\n", &a);    // Shows address
   printf("Value at ptr: %d\n", *ptr);  // Output: 20
}
```

(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.

   - This function returns the **number of characters** in a string (excluding \0).
   - Useful when you want to know the size of a string.

Example:

```
#include <stdio.h>

#include <string.h>


int main() {
   char name[] = "Mukhtar";
   printf("Length is: %d", strlen(name));  // Output: 7
   return 0;
}
```

Output: 7

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

   - It **copies the content** of one string into another.
   - Useful when you want to copy a string without using loops.

Example:

```
#include <stdio.h>

#include <string.h>


int main() {
```

```
    char name1[20];
    char name2[] = "mukhtar";

    strcpy(name1, name2);
    printf("Copied String = %s", name1);  // Output: mukhtar

    return 0;
}
```

Output: mukhtar

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

- This function **joins** two strings together.
- Useful for making full names, sentences, or file paths.

Example:

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

int main() {
    char name[50] = "mukhtar";
    char surname[] = "badi";

    strcat(name, surname);
    printf("Full name = %s", name);  // Output: mukhtarbadi

    return 0;
}
```

Outut: mukhtarbadi

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

- Compares two strings character by character.
- Returns 0 if both strings are equal.
- Useful in login systems, password checks, etc.

Example:

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

int main() {
   char correct[] = "mukhtar";
   char input[50];

   do {
      printf("Enter name: ");
      scanf("%s", input);
   }
  while (strcmp(correct, input) != 0);

   printf("Answer is Correct!!");  // Output when user types "mukhtar"
   return 0;
}
```

Output: Answer is Correct!!

5. strchr() – Find Character in String

- Finds the first occurrence of a character in a string.
- Useful when searching in strings.

Example:

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

int main() {
    char str[] = "hello";
    char *ptr = strchr(str, 'l');

    if (ptr != NULL) {
        printf("First 'l' at position: %ld", ptr - str);  // Output: 2
    }

    return 0;
}
// Output: 2
```

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

Structure

- A **structure** is a **user-defined data type** in C.
- It is used to **group different types of data** together under one name.
- It is declared 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, "mukhtar", 89.5};

Method 2: Assigning Values Later s1.roll

= 101;

strcpy(s1.name, "mukhtar"); // Use strcpy for strings

s1.marks = 89.5;

- **Accessing Structure Members**

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

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

- **Example:**

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

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

int main() {
   struct Student s1;

   s1.roll = 101;
   strcpy(s1.name, "mukhtar");
   s1.marks = 89.5;

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

```
    printf("Marks : %.2f\n", s1.marks);

    return 0;
}
```

(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

- Before doing any read/write, you must open the file using fopen().

```
FILE *fp;
```

fp = fopen("filename.txt", "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

- Use these functions to write data:

```
FILE *fp = fopen("output.txt", "w");


fprintf(fp, "Hello, world!\n");      // write formatted text
fputs("This is a line.\n", fp);     // write a string
fputc('A', fp);                     // write a single character


fclose(fp);  // Always close after writing
```

- 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

- Use these functions to read data:

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

fgets(buffer, 100, fp);   // read a line from file
printf("%s", buffer);

fclose(fp);
```

4. Closing a File

- Always close a file using fclose():

```
fclose(fp);
```