Module 2 – Introduction to Programming: C Programming Questions

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

- C was created by Dennis Ritchie at Bell Labs in the early 1970s to build the UNIX operating system.

Importance: It's a powerful and efficient "middle-level" language, meaning it combines high-level readability with low-level control over computer hardware.

Why it's still used:- It's extremely fast and is the foundation for modern operating systems (like Windows and Linux), device drivers, and performance-critical software. Many modern languages like Python and JavaScript have their core interpreters written in C.

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.

- **To write and run C code, you need two things:**

A Compiler (like GCC) to translate your C code into machine code that the computer can execute.

An IDE (Integrated Development Environment) like VS Code or Code::Blocks, which is a text editor with helpful features for programming.

Brief Steps (for VS Code):-

Step 1:- Install a C Compiler.
Step 2:- Install and Set Up VS Code.
Step 3:- Install Necessary Extensions in VS Code.
Step 4:- Create and Run Your First C Program.

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

- A C program has a standard structure that includes preprocessor directives, the main function, and statements.

#include <stdio.h>: A header that includes the standard input/output library, giving you access to functions like printf().

int main(): The main function where every C program begins execution.

int age = 25;: A variable age of data type int (integer) is declared and initialized.

printf(...): A function call to print formatted text to the console.

return 0;: Indicates that the program has finished successfully.

Input:-

```
#include <stdio.h> // Header for input/output functions

// This is a single-line comment
int main() {
```

```
    // Variable declaration
    int year = 2025;

    // Print the value of the variable to the console
    printf("Hello from the year %d!\n", year);

    return 0; // End of the program
}
```

Output:-

Hello from the year 2025!

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

- Operators are symbols that perform operations on variables and values.

Arithmetic:- + (add), **-** (subtract), ***** (multiply), **/** (divide), **%** (modulus/remainder).

Relational:- == (equal to), **!=** (not equal to), **>** (greater than), **<** (less than).

Logical:- && (logical AND), || (logical OR), ! (logical NOT).

Assignment:- = (assign), += (add and assign, e.g., x += 5 is x = x + 5).

Increment/Decrement:- ++ (increment by 1), -- (decrement by 1).

Input:-

```
#include <stdio.h>

int main() {
    int a = 10, b = 4;
    int sum = a + b;      // Arithmetic operator
    int remainder = a % b;  // Arithmetic operator (modulus)

    printf("Sum: %d\n", sum);
    printf("Remainder: %d\n", remainder);

    // Relational and Logical operators
    if (a > b && b != 0) {
        printf("a is greater than b, and b is not zero.\n");
    }

    return 0;
}
```

Output:-

Sum: 14
Remainder: 2
a is greater than b, and b is not zero.

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

- These statements control the flow of a program based on certain conditions.

if:- Executes a block of code if a condition is true.

if-else:- Executes the if block if the condition is true, otherwise executes the else block.

switch:- A cleaner alternative to a long if-else if chain for checking a variable against multiple constant values.

Input:-

```
#include <stdio.h>

int main() {
   int day = 4;
// Switch Case
   switch (day) {
      case 1:
         printf("Monday\n");
         break;
      case 2:
         printf("Tuesday\n");
         break;
      case 3:
         printf("Wednesday\n");
         break;
      case 4:
         printf("Thursday\n");
         break;
      default:
         printf("Some other day\n");
   }
   return 0;
}
```

Output:-

Thursday

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

- Loops are used to execute a block of code repeatedly.

for loop:- Best when you know exactly how many times you want to loop.

while loop:- Best when you want to loop as long as a condition is true, but you don't know the number of iterations beforehand. It checks the condition before each iteration.

do-while loop:- Similar to while, but it checks the condition after each iteration, meaning it will always execute at least once.

Input:-

```
#include <stdio.h>

int main() {
```

```
    printf("For loop:\n");
    // Use a 'for' loop to print numbers 1 to 3
    for (int i = 1; i <= 3; i++) {
        printf("%d ", i);
    }
    printf("\n");

    printf("\nWhile loop:\n");
    int j = 1;
    // Use a 'while' loop for the same task
    while (j <= 3) {
        printf("%d ", j);
        j++;
    }
    printf("\n");

    return 0;
}
```

Output:

For loop:
1 2 3

While loop:
1 2 3

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

- These are jump statements that alter the normal flow of control.

break:- Immediately terminates the loop or switch statement it is in.

continue:- Skips the current iteration of a loop and moves to the next one.

goto:- Jumps to a labeled statement. (Use is heavily discouraged as it can make code confusing and hard to maintain).

Input:-

```
#include <stdio.h>

int main() {
    for (int i = 1; i <= 5; i++) {
        if (i == 3) {
            continue; // Skip printing the number 3
        }
        if (i == 5) {
            break; // Stop the loop when i is 5
        }
        printf("%d ", i);
    }
    printf("\n");
    return 0;
}
```

Output:-

1 2 4

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

- A function is a reusable block of code that performs a specific task.

Declaration (Prototype):- Tells the compiler about the function's name, return type, and parameters.

Definition:- The actual code that makes up the function.

Call:- Invoking the function to execute its code.

Input:-

```
#include <stdio.h>

// Function declaration (prototype)
int add(int a, int b);

int main() {
   int result = add(5, 10); // Function call
   printf("The result is: %d\n", result);
   return 0;
}

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

Output:-

The result is: 15

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

-

Feature	One-Dimensional (1D) Array	Multi-Dimensional (2D) Array
Concept	A simple list of elements.	An array of arrays, forming a grid or table.
Analogy	A single row of lockers or a shopping list.	A grid of seats in a classroom or a calendar.
Declaration	dataTypearrayName[size];	dataTypearrayName[rows][columns];
Accessing Elements	Uses one index: arrayName[index]	Uses two indices: arrayName[rowIndex][colIndex]
Use Case	Storing a list of student scores, temperatures, or names.	Representing a game board (like Tic-Tac-Toe), a matrix in mathematics, or an image (grid of pixels).

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

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

Why they are important: Pointers allow for dynamic memory allocation, efficient array manipulation, and passing variables to functions "by reference," which means the function can modify the original variable.

Input:-

```
#include <stdio.h>

int main() {
   int age = 30;
   int *pAge = &age; // pAge is a pointer that stores the address of 'age'

   printf("Value of age: %d\n", age);
   printf("Memory address of age: %p\n", (void *)&age);
   printf("Value stored in pointer pAge: %p\n", (void *)pAge);
   printf("Value at the address stored in pAge: %d\n", *pAge); // Dereferencing

   return 0;
}
```

Output:- (The memory address will vary on your machine)

```
Value of age: 30
Memory address of age: 0x7ffc1234abcd
Value stored in pointer pAge: 0x7ffc1234abcd
Value at the address stored in pAge: 30
```

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

- In C, strings are arrays of characters ending with a null character (\0). The <string.h> library provides functions to work with them.

strlen(str):- Returns the length of the string str.

strcpy(dest, src):- Copies the string src into dest.

strcat(dest, src):- Concatenates (appends) the string src to the end of dest.

strcmp(str1, str2):- Compares two strings. Returns 0 if they are equal.

Input:-

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

int main() {
   char greeting[20] = "Hello";
   char name[] = " World";

   strcat(greeting, name); // Appends " World" to "Hello"

   printf("Combined string: %s\n", greeting);
```

```
    printf("Length of new string: %zu\n", strlen(greeting));

    return 0;
}
```

Output:-

```
Combined string: Hello World
Length of new string: 11
```

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

- A structure is a user-defined data type that allows you to group together related variables of different data types. It's like creating a template for a record.

Input:-

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

// Declare a structure named 'Student'
struct Student {
    int rollNo;
    char name[50];
    float gpa;
};

int main() {
    // Create a structure variable and initialize it
    struct Student s1;
    s1.rollNo = 101;
    strcpy(s1.name, "Raj");
    s1.gpa = 3.8;

    // Access and print the members
    printf("Student Name: %s\n", s1.name);
    printf("Roll No: %d\n", s1.rollNo);
    printf("GPA: %.1f\n", s1.gpa);

    return 0;
}
```

Output:-

```
Student Name: Raj
Roll No: 101
GPA: 8.8
```

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

- File handling in C allows you to create, read, update, and delete files. This is essential for data persistence, which means saving data even after your program has closed.

Key Functions:

fopen():- Opens or creates a file.

fprintf() / fputs():- Writes data to a file.

fscanf() / fgets():- Reads data from a file.

fclose():- Closes a file.

Input:-

```
#include <stdio.h>

int main() {
   FILE *filePointer;

   // Open a file named "log.txt" in write mode ("w")
   filePointer = fopen("log.txt", "w");

   if (filePointer == NULL) {
      printf("File could not be opened.\n");
      return 1;
   }

   // Write text to the file
   fprintf(filePointer, "This is a test log entry.\n");

   // Close the file
   fclose(filePointer);

   printf("Successfully wrote to the file 'log.txt'.\n");

   return 0;
}
```

Output:-

```
Successfully wrote to the file 'log.txt'.
```