Module 2 – Overview of C Programming

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

ANS. **History and Evolution of C Programming:**

C was created in **1972 by Dennis Ritchie at Bell Labs** to develop the UNIX operating system. It evolved from earlier languages like **B and BCPL** and quickly became popular because of its **efficiency and portability**. In **1989**, ANSI standardized C (ANSI C), and later versions like **C99, C11, and C18** added modern features.

Importance and Why It's Still Used:

- **Fast and efficient** – close to machine code.
- **Portable** – runs on different hardware easily.
- **Foundation language** – influenced C++, Java, Python, and more.
- **Critical systems** – operating systems, embedded devices, and compilers are still written in C.

Q.2 - 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.

ANS. **Steps to Install GCC & Set Up IDE:**

1. Install GCC (Compiler):
   - **Download MinGW-w64 (Windows) or use built-in package managers (Linux/macOS).**
   - **Add the bin folder path to System PATH.**
     - ➢ **Test with:** **gcc --version**
2. Dev C++:
   - **Download Dev C++ (comes with GCC).**
   - **Install → Open → Create new project → Write code → Compile & Run.**
3. Code::Blocks:
   - **Download version with MinGW included.**
   - **Install → Open → New Console Project → Build & Run.**

4. **VS Code:**

- **Install VS Code.**
- **Add C/C++ extension.**
- **Install MinGW-w64 separately.**
- **Configure build tasks (tasks.json) to use GCC.**
- **Write code → Press Ctrl+Shift+B → Run.**

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

ANS.

Basic Structure of a C Program:

1. **Headers** – Include libraries using #include.
   - **Example:**

     ```
     #include <stdio.h>   // standard input-output library
     ```
2. **Main Function** – Entry point of every C program.
   - **Example:**

     ```
     int main() {
        // code goes here
        return 0;
     }
     ```
3. **Comments** – Notes for programmers, ignored by compiler.
   - **Example:**

     ```
     // This is a single-line comment

     /* This is a multi-line comment */
     ```
4. **Data Types** – Define type of data (e.g., int, float, char).
5. **Variables** – Named storage for data.
   - **Example:**

```
int age = 20;       // integer variable

float price = 99.5;  // floating-point variable

char grade = 'A';    // character variable
```

Example Program:

```
#include <stdio.h>   // header
int main() {
   // simple program
   int age = 20;
   float price = 99.5;
   char grade = 'A';
   printf("Age: %d, Price: %.2f, Grade: %c", age, price, grade);
   return 0;
}
```

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

ANS. Types of Operators in C:

1. **Arithmetic Operators:**
   - Perform basic math: +, -, *, /, %
   - Example: a + b adds two numbers.
2. **Relational Operators**
   - Compare values: ==, !=, >, <, >=, <=
   - Example: a > b checks if a is greater than b.
3. **Logical Operators:**
   - Combine conditions: && (AND), || (OR), ! (NOT)
   - Example: (a > b && b > c) is true if both conditions hold.
4. **Assignment Operators:**
   - Assign values: =, +=, -=, *=, /=, %=
   - Example: x += 5 means x = x + 5.
5. **Increment/Decrement Operators:**
   - Increase or decrease by 1: ++, --
   - Example: x++ adds 1 to x.
6. **Bitwise Operators:**
   - Work on bits: & (AND), | (OR), ^ (XOR), ~ (NOT), << (left shift), >> (right shift).
   - Example: a & b performs bitwise AND.
7. **Conditional (Ternary) Operator:**
   - Short form of if-else: condition ? expr1 : expr2
   - Example: (a > b ? a : b) returns the larger value.

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

ANS. **Decision-Making Statements in C:**

1. **if statement** – Executes code if condition is true.
   - **Example:**

```
if (x > 0) {
   printf("Positive");
}
```

2. **if-else statement** – Chooses between two options.
   - **Example:**

```
if (x % 2 == 0) {
   printf("Even");
} else {
   printf("Odd");
}
```

3. **nested if-else** – Multiple conditions checked inside each other.
   - **Example:**

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

4. **switch statement** – Selects one case from many options.
   - **Example:**

```
switch (day) {
   case 1: printf("Monday"); break;
   case 2: printf("Tuesday"); break;
   default: printf("Other day");
}
```

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

ANS. **Loops in C:**

1. **while loop:**

- Checks condition before running.
- Runs 0 or more times.

Example:

```
while (x < 5) {
   printf("%d", x);
   x++;
}
```

2. **for loop:**

- Has initialization, condition, and update in one line.
- Runs 0 or more times.

Example:

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

3. **do-while loop:**

- Executes code first, then checks condition.
- Runs at least once.

Example:

```
do {
   printf("%d", x);
   x++;
} while (x < 5);
```

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

ANS. **Control Statements in C:**

1. break – Exits from a loop or switch immediately.

Example:

```
for (int i = 1; i <= 5; i++) {
   if (i == 3) break;   // stops loop at 3
   printf("%d ", i);
}
```

2. continue – Skips the current iteration and moves to the next.

Example:

```
for (int i = 1; i <= 5; i++) {
   if (i == 3) continue;   // skips printing 3
   printf("%d ", i);
}
```

3. goto – Jumps to a labelled statement in the program.

   Example:

```
int x = 1;
goto label;
printf("This won't run");
label:
printf("Jumped here using goto");
```

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

ANS. **Functions in C:**

A function is a block of code that performs a specific task, making programs modular and reusable.

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

   Example:

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

2. **Function Definition:**
   - Actual body of the function where logic is written.

   Example:

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

3. **Function Call:**
   - Executes the function from main() or another function.

   Example:

```
int main() {
   int sum = add(5, 3);   // call
   printf("Sum = %d", sum);
   return 0;
}
```

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

ANS. **Arrays in C:**

An array is a collection of elements of the same data type stored in contiguous memory locations. It allows storing and accessing multiple values using a single variable name with an index.

1. One-Dimensional Array:

- **Stores elements in a single row (like a list).**

Example:

```
int arr[5] = {10, 20, 30, 40, 50};
printf("%d", arr[2]);   // prints 30
```

2. Multi-Dimensional Array:

- **Stores elements in rows and columns (like a table).**

Example:

```
int matrix[2][3] = {{1,2,3}, {4,5,6}};
printf("%d", matrix[1][2]);   // prints 6
```

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

ANS.

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

Declaration: Use the * operator with a data type.

Example:

```
int *p;   // pointer to int

char *c;  // pointer to char
```

Initialization: Assign it the address of a variable using &.

Example:

```
int x = 10;
int *p = &x;  // p holds address of x
```

Importance:

- **Enable dynamic memory management (malloc, free).**
- **Allow efficient array and string handling.**
- **Support function arguments by reference (modify values directly).**
- **Crucial for data structures like linked lists, trees, etc.**

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

ANS. **Common String Handling Functions in C:**

- **strlen(str)** → Returns length of string (excluding \0).
  Use: Count characters in "Hello" → strlen("Hello") = 5.
- **strcpy(dest, src)** → Copies string src into dest.
  Use: Copy "World" into another buffer.
- **strcat(dest, src)** → Concatenates src to the end of dest.
  Use: Join "Hello" + " World" → "Hello World".
- **strcmp(s1, s2)** → Compares two strings; returns 0 if equal, 0 otherwise.
  Use: Check if user input matches a password.
- **strchr(str, ch)** → Finds first occurrence of character ch in str.
  Use: Locate '@' in an email string.

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

ANS.

- **Concept:** **A structure (struct) is a user-defined data type that groups related variables (of different types) under one name.**
- **Declaration:**

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

- **Initialization:**

```
struct Student s1 = {1, "Alice", 95.5};
```

- **Access Members:** **Use the dot (.) operator.**

```
printf("%s", s1.name); // Access name
s1.marks = 98.0; // Modify marks
```

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

ANS.

- **Importance:** File handling allows programs to store, retrieve, and manipulate data permanently (beyond program execution).

❖ **Basic Operations:**

- **Open a file:**

```
FILE *fp = fopen("data.txt", "r");  // open for reading
FILE *fp = fopen("data.txt", "w");  // open for writing
```

- **Read/Write:**

```
fscanf(fp, "%s", str);   // read from file
fprintf(fp, "%s", str);  // write to file
```

- **Close a file:**

```
fclose(fp);
```