Module-2: Introduction to programming

Theory exercise

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

Ans.

→Programming Overview

- **Created by:** Dennis Ritchie in the early 1970s at Bell Labs.
- **Purpose:** To develop the Unix operating system.
- **Based on:** Earlier languages like B and BCPL.
- **Standard Version:** ANSI C (1989), later C99, C11, and C18.

→Why C is Important:

- **Fast and efficient** – good for performance-critical tasks.
- **Low-level access** – allows control over memory and hardware.
- **Portable** – runs on many types of systems.
- **Foundation language** – base for C++, Java, Python, etc.
- **Still used in:**
  - Operating systems (e.g., Linux)
  - Embedded systems (e.g., cars, robots)
  - Game engines
  - Compilers

→in sort:
C is old but powerful, and still useful today for system-level programming and learning the basics of computer science.

Que-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.

Ans.

Option 1: Using Dev C++ (Windows)

1. **Download Dev C++**
    - Go to https://sourceforge.net/projects/orwelldevcpp/
2. **Install Dev C++**
    - Run the downloaded .exe file and follow the installation steps.
3. **Start Coding**
    - Open Dev C++, create a new C project or file, and start writing code.

Option 2: Using Code::Blocks (Windows/Linux/Mac)

1. **Download Code::Blocks with GCC**
    - Go to https://www.codeblocks.org/downloads/
    - Choose the version with "MinGW" (GCC compiler included).

2. **Install Code::Blocks**
    - Run the setup and follow the instructions.
3. **Start Coding**
    - Open Code::Blocks, create a new project, and start writing C code.

Option 3: Using VS Code (Windows/Linux/Mac)

1. **Download and Install VS Code**
    - Go to https://code.visualstudio.com/
2. **Install GCC Compiler**
    - **Windows:** Install MinGW and add bin folder to system PATH.
    - **Linux:** Run sudo apt install build-essential
    - **Mac:** Run xcode-select --install
3. **Install C/C++ Extension in VS Code**
    - Open VS Code → Extensions (Ctrl+Shift+X) → Search and install “C/C++” by Microsoft.
4. **Create and Run a C File**
    - Make a new .c file, write your code, and use terminal commands:

```
gcc filename.c -o output
```

→in sort:

- **Dev C++ or Code::Blocks** – Good for beginners, includes everything in one package.
- **VS Code** – More flexible, great for advanced users with extra setup.

Que-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. Header Files

- Used to include standard libraries.
- Example:

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

---

2. Main Function

- Entry point of every C program.
- Code runs from here.
- Example:

```
int main() {
```

```
    // code here
    return 0;
}
```

---

3. Comments

- Explain code (not executed).
- **Single-line:** // comment
- **Multi-line:**

```
/*
  this is
  a comment
*/
```

---

4. Data Types

- Tell the type of data a variable can store.
- Common types:
    - int – integers (e.g., 5)
    - float – decimal numbers (e.g., 3.14)
    - char – single character (e.g., 'A')

---

5. Variables

- Store data values.
- Example:

```
int age = 20;
float price = 99.5;
char grade = 'A';
```

→ **Full Simple Example:**

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

int main() {  // Main function
    printf("\n hello world");
   // Declare variables
   int age = 18;
    printf("\n %d",age);

}
```

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

Ans. **C Operators –**

---

1. Arithmetic Operators

Used for basic math.

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

---

2. Relational Operators

Compare two values.

Operator	Meaning	Example
==	Equal to	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 Operators

Used in conditions (true/false).

Operator	Meaning	Example

&&	AND (both true)	a > 5 && b < 10
||	OR(One true)	a>5||b<10
!	NOT (reverse)	!(a > 5)

---

4. Assignment Operators

Assign values to variables.

Operator	Meaning	Example
=	Assign	a = 5
+=	Add and assign	a += 2 (a = a + 2)
-=	Subtract and assign	a -= 1
*=	Multiply and assign	a *= 3
/=	Divide and assign	a /= 2

---

5. Increment / Decrement

Increase or decrease by 1.

Operator	Meaning	Example
++	Increment (+1)	a++ or ++a
--	Decrement (-1)	a-- or --a

---

6. Bitwise Operators

Work on bits (0s and 1s).

Operator	Meaning	Example
&	AND	a & b
|	OR	OR
~	NOT	~a

---

7. Conditional (Ternary) Operator

Shortcut for if-else.

Operator	Format	Example
?:	condition ? x : y	a > b ? a : b (returns bigger value)

8.Sizeof Operator

Que-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

- Runs code if condition is true.

Syntax:

```
if (condition) {
```

```
    // code
}
```

Example:

```
if (age >= 18) {
    printf("You are an adult.");
}
```

---

2. if-else Statement

- Runs one block if true, another if false.

Example:

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

---

3. Nested if-else

- if-else inside another if-else.

Example:

```
if (marks >= 90) {
    printf("Grade A");
```

```
} else if (marks >= 75) {
    printf("Grade B");
} else if (marks >= 50) {
    printf("Grade C");
} else {
    printf("Fail");
}
```

---

4. switch Statement

- Checks many possible values of a variable.

Syntax:

```
switch (value) {
    case x:
        // code
        break;
    case y:
        // code
        break;
    default:
        // code
```

```
}
```

Example:

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

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

Ans.

Comparison of Loops in c

---

1. while Loop

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

Syntax:

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

Best For:

- When number of repeats is **unknown**.
- Example: Repeat until user enters 0.

---

2. for Loop

- Has all parts (start, condition, update) in one line.
- Checks condition **before** running.
- Runs **0 or more times**.

Syntax:

```
for (int i = 0; i < 5; i++) {
   // code
}
```

Best For:

- When number of repeats is **known**.
- Example: Loop 10 times to print numbers.

---

3. do-while Loop

- Checks condition **after** running.
- Runs **at least once**.

Syntax:

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

Best For:

- When code must run **at least once**.
- Example: Show menu at least once before checking choice.

→ Quick Comparison Table

Loop Type	Condition Check	Runs At Least Once?	Best Use Case
while	Before loop	No	Unknown repetitions
for	Before loop	No	Known number of repeats
do-while	After loop	Yes	Must run at least once

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

Ans.

Control Statements in C

---

1. break Statement

- **Stops** a loop or switch early.

Use: Exit loop when a condition is met.

Example:

```
for (int i = 1; i <= 5; i++) {
    if (i == 3)
        break;
    printf("%d ", i);  // Output: 1 2
}
```

---

2. continue Statement

- **Skips** the rest of the loop for one time, and continues with next.

Use: Skip a part and move to next loop cycle.

Example:

```
for (int i = 1; i <= 5; i++) {
    if (i == 3)
        continue;
    printf("%d ", i);  // Output: 1 2 4 5
}
```

---

3. goto Statement

- **Jumps** to a labeled part of the code.

Use: Not recommended often. Use for special cases.

Example:

```
int num = 5;
if (num == 5)
    goto skip;

printf("This won't run.\n");

skip:
printf("Jumped to here.\n");  // Output: Jumped to here.
```

→in sort:

Statement	Use For	Action
break	Exit loop/switch early	Stops current block
continue	Skip one loop turn	Jumps to next iteration
goto	Jump to label	Moves to a specific location

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

Ans.

Functions in C

- Function: A block of code that performs a specific task.
- Purpose: Makes the program modular, reusable, and easy to understand.

---

1. Function Declaration (Prototype)

- Tells the compiler about the function name, return type, and parameters.
- Placed at the top of the program (before main()).

Example:

```
int add(int, int); // function declaration
```

---

2. Function Definition

- The actual code of the function.
- Contains the statements to perform the task.

Example:

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

3. Calling a Function

- Use the function name followed by arguments.
- Usually done inside the main() function or other functions.

Example:

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

Summary Table

Step	What it Does	Example Code
Declaration	Introduces function to compiler	int add(int, int);
Definition	Contains actual logic	int add(int a, int b) { ... }
Calling	Uses function in code	sum = add(5, 3);

Que-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 type stored in contiguous memory locations.
It helps to store multiple values using a single variable name.

For example:

int marks[5]; // array to store 5 integers

---

1. One-Dimensional Array

- It stores data in a single row.
- Access elements with one index.

Example:

int numbers[4] = {10, 20, 30, 40};

printf("%d", numbers[2]); // prints 30

---

2. Multi-Dimensional Array

- Stores data in rows and columns (like tables).
- Access elements with multiple indices.

Example (2D Array):

int matrix[2][3] = { {1, 2, 3}, {4, 5, 6} };

printf("%d", matrix[1][2]); // prints 6

Difference in Short

Aspect	One-Dimensional	Multi-Dimensional
Shape	Line	Table (rows & columns)
Accessing elements	Single index (arr[i])	Multiple indices (arr[i][j], etc.)

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

Ans.

What are pointers in C?

Pointers are variables that store the memory address of another variable.

How are they declared and initialized?

- **Declaration:** data_type *pointer_name;
- **Initialization:** pointer_name = &variable;
- Example:

int a = 10;

int *p;

p = &a; // p now points to a

Why are pointers important in C?

- They allow direct memory access.
- Useful for dynamic memory allocation.
- Help in efficient array and string handling.
- Enable function arguments to be modified within the function.

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

Ans.

1. strlen() – Finds the length of a string

- **Use:** To count characters in a string (excluding \0).
- **Example:**

```
strlen("hello");  // returns 5
```

---

2. strcpy() – Copies one string to another

- **Use:** To copy contents of one string into another.
- **Example:**

```
strcpy(dest, "hello");
```

---

3. strcat() – Concatenates (joins) two strings

- **Use:** To add one string to the end of another.
- **Example:**

strcat(str1, str2);

---

4. strcmp() – Compares two strings

- **Use:** To check if strings are same or different.
- **Returns:**
  - 0 if equal,
  - <0 if first < second,
  - >0 if first > second.
- **Example:**

strcmp("abc", "abc"); // returns 0

---

5. strchr() – Finds a character in a string

- **Use:** To search a character in a string.
- **Returns:** Pointer to first match or NULL.
- **Example:**

strchr("hello", 'e'); // returns pointer to 'e'

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

Ans.

What is a Structure?

A **structure** in C is a user-defined data type that allows you to combine different types of data (like int, float, char) under one name.

Declaration of Structure:

```
struct Student {
    int id;
    char name[20];
    float marks;
};
```

Creating & Initializing a Structure Variable:

```
struct Student s1 = {1, "John", 85.5};
```

Accessing Structure Members:

```
printf("ID = %d\n", s1.id);
printf("Name = %s\n", s1.name);
```

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

Using Dot (.) Operator:

To access members: structureVariable.memberName

In sort:

- Structure groups variables of **different types**.
- Use struct keyword to define.
- Use **dot (.)** to access members.

Que-13.

Ans.

File Handling in C :-

Importance:

File handling allows a C program to **store data permanently** (unlike variables which lose data when the program ends). It helps in **reading/writing data** to files like .txt.

Basic File Operations:

1. **Open a File:**

```
FILE *fp;
```

```
fp = fopen("data.txt", "w");  // "w" = write, "r" = read, "a" = append
```

2. **Write to a File:**

```
fprintf(fp, "Hello, File!");
```

3. **Read from a File:**

```
char text[100];
fgets(text, 100, fp);    // reads a line
```

4. **Close a File:**

```
fclose(fp);
```

File Modes:

Mode	Meaning
"r"	Read only
"w"	Write (overwrite)
"a"	Append to file

IN sort:

- Use **FILE *** to handle files.
- Use **fopen()** and **fclose()** to open/close.
- Use **fprintf()** or **fscanf()**, fgets(), etc., to write/read.