Introduction to OOPS Programming

1. Introduction to C++

THEORY EXERCISE:

Q1 . What are the key differences between Procedural Programming and ObjectOrientedProgramming (OOP)?

ANSWER :-

Aspect	(POP)	(OOP)
Paradigm	Focuses on **procedures** and functions.	Focuses on **objects** and classes.
Data Handling	Data is exposed; functions act on global data.	Data is encapsulated within objects (Encapsulation).
Modularity	Code is divided into functions or procedures.	Code is divided into classes and objects.
Reusability	Limited reusability (requires code duplication).	Promotes reusability through inheritance and polymorphism.
Security	Data is not protected; accessible globally.	Data is secured and accessed through methods
Approach	Top-down approach.	Bottom-up approach.

Q2. List and explain the main advantages of OOP over POP

ANSWER:-

Encapsulation:
OOP allows bundling data and functions into a single unit called a class. This protects data from unintended modification.
Example: Private attributes in a class can only be accessed via methods.
Reusability (Inheritance):
OOP allows code reuse through inheritance, where a new class can derive properties and behaviors from an existing class.
Flexibility (Polymorphism):
OOP supports polymorphism, allowing methods to behave differently based on objects or input.
Maintainability:
OOP structures the program into objects and classes, which makes code more modular and easier to manage or extend.
Abstraction:
Hides complex implementation details from the user and provides a clear interface to interact with.
Scalability:
OOP makes it easier to develop and scale large software systems by promoting modular and organized design.

Q3. Explain the steps involved in setting up a C++ development environment.

ANSWER:-

Step 1 .Install a C++ Compiler:
Install a C++ compiler like **GCC** (GNU Compiler Collection) for Linux or **MinGW** for Windows.

Step 2.Install an IDE (Optional):
Use an Integrated Development Environment (IDE) for better development experience:
Examples: Visual Studio, Code::Blocks, CLion, or Eclipse.

Step 3.Set Up the Environment Variables:
Configure system PATH to include the compiler's binary directory.

Step 4.Write Your First Program:
Create a simple C++ file (e.g., hello.cpp) using a text editor or IDE

Step 5. Compile the Program:
Use the terminal or IDE to compile the program:
Command: g++ hello.cpp -o hello

Step 6.Run the Executable:
Execute the compiled file:
Command: ./hello (Linux) or hello.exe (Windows).

Q4 . What are the main input/output operations in C++? Provide examples.

ANSWER:-

- C++ provides **cin** for input and **cout** for output, which are part of the iostream library.

A) Input Operations:
- The cin object is used to take input from the user.
- EX

```
#include <iostream> using namespace std;
  int main() { int age;
cout << "Enter your age: "; cin >> age; // Takes input from the user cout << "Your age is: " <<
age << endl; return 0; }
```

B) Output Operations:
-The cout object is used to print output to the console.
-EX

```
#include <iostream> using namespace std;
 int main()
{ cout << "Welcome to C++ Programming!" << endl; // Prints to the console return 0;
  }
```

C) File Input/Output:
- C++ also provides file input/output using fstream.
- #include <iostream>

```
 #include <fstream>
using namespace std;
int main() {
 // Writing to a file
 ofstream outputFile("example.txt");
```

```
outputFile << "Hello, File!" << endl;
    outputFile.close();

    // Reading from a file
    ifstream inputFile("example.txt");
    string content;
    getline(inputFile, content);
    cout << "File Content: " << content << endl;
    inputFile.close();

    return 0;
}
```

LAB EXERCISES:

Q1. First C++ Program: Hello World o Write a simple C++ program to display "Hello, World!". o Objective: Understand the basic structure of a C++ program, including #include, main(), and cout.

ANSWER:-

```
#include <iostream>
using namespace std;
 int main()  {
  cout << "Hello, World!" << endl;
 return 0;
 }
```

Q2. Basic Input/Output o Write a C++ program that accepts user input for their name and age and then displays a personalized greeting. o Objective: Practice input/output operations using cin and cout.

Answer :-

```
 #include <iostream>
 using namespace std;
 int main() {
   string name;
      int age;
    cout << "Enter your name: ";
    cin >> name;
   cout << "Enter your age: ";
    cin >> age
cout << "Hello, " << name << "! You are " << age << " years old." << endl;
    return 0;
}
```

Q3 . POP vs. OOP Comparison Program o Write two small programs: one using Procedural Programming (POP) to calculate the area of a rectangle, and another using Object-Oriented Programming (OOP) with a class and object for the same task. o Objective: Highlight the difference between POP and OOP approaches.

ANSWER:-

A) Procedural Programming (POP):

```
#include <iostream>
 using namespace std;
int calculateArea(int length, int width) {
                return length * width; }
int main() {
   int length, width;
   cout << "Enter length and width of the rectangle: ";
   cin >> length >> width;

  int area = calculateArea(length, width); cout << "The area of the   rectangle is: " << area <<
endl;

 return 0;
}
```

B) Object-Oriented Programming (OOP):

```
#include <iostream>
using namespace std;
class Rectangle {
 private:
      int length, width;

public:
       void setDimensions(int l, int w) {
             length = l;
             width = w;
          }
int calculateArea() {
    return length * width;
 }
};
int main() {
 Rectangle rect;
 int length,
width;

 cout << "Enter length and width of the rectangle: ";
 cin >> length >> width; rect.setDimensions(length, width); cout << "The area of the rectangle
is: " << rect.calculateArea() << endl;

 return 0;
}
```

Q4. Setting Up Development Environment o Write a program that asks for two numbers and displays their sum. Ensure this is done after setting up the IDE (like Dev C++ or CodeBlocks). o Objective: Help students understand how to install, configure, and run programs inan IDE**.**

ANSWER:-

```
-#include <iostream>
  using namespace std;
   int main() { int num1, num2, sum
   cout << "Enter the first number: ";
   cin >> num1;

  cout << "Enter the second number: ";
  cin >> num2;
  sum sum = num1 + num2;

  cout << "The sum of " << num1 << " and " << num2 << " is: "   << sum << endl;
   return 0;
}
```

- **Steps to Set Up and Run the Program:**

Install IDE:
Download and install an IDE like **Code::Blocks** or **Dev C++**.

Create a New Project/File:
Open the IDE and create a new C++ source file (e.g., sum_program.cpp).

Write the Code:
Copy the code above into the file.

Build/Compile the Program:
Press F9 in Code::Blocks or click "Build and Run" to compile the program.

Run the Executable:
If successful, the program will ask for input and display the result.

2. Variables, Data Types, and Operators

THEORY EXERCISE:

Q.1 What are the different data types available in C++? Explain with examples.

ANSWER:-

A) Fundamental Data Types

- These include basic data types like integers, floating points, characters, and Boolean values.

Data Type	Description	Example
int	Integer values	int a = 10;
float	Floating-point	float b = 5.75;
double	Double-precision float	double pi = 3.14159;
char	Single character	char grade = 'A';
bool	Boolean (true or false)	bool isTrue = true;
void	Represents "no value"	Used in function

```
#include <iostream>
using namespace std;

int main() {
   int age = 25;
   float weight = 65.5;
   double pi = 3.14159;
   char initial = 'J';
   bool isStudent = true;

   cout << "Age: " << age << ", Weight: " << weight << ", PI: " << pi << endl;
   cout << "Initial: " << initial << ", Is Student: " << isStudent << endl;

   return 0;
}
```

B). Derived Data Types

- These are built from fundamental types:

Arrays:

Collection of elements of the same type.

Example: int numbers[5] = {1, 2, 3, 4, 5};

Pointers:

Stores the memory address of another variable.

Example: int* ptr;

References:

Alias for another variable.

Example:

int &ref = age;

C) User-Defined Data Types

- These are defined by the programmer:

Structures:

Groups related variables together.

Example:

struct Student { string name; int age; };

Classes:

Blueprint for creating objects.

Enumerations (enum):

Used to define constants.

Example:

enum Day { Monday, Tuesday };

Q2 . Explain the difference between implicit and explicit type conversion in C++.

ANSWER:-

A) Implicit Type Conversion (Type Promotion):

- The compiler automatically converts one data type to another.
-It happens when mixing different data types in an operation.
-EX

```
 #include <iostream>
using namespace std;

int main() {
    int a = 10;
    float b = 2.5;

    float result = a + b;

    cout << "Result: " << result << endl; // Output: 12.5
    return 0;
}
```

B) Explicit Type Conversion (Type Casting):

-The user explicitly converts one data type to another using **type casting**.

Syntax :

```
 (data_type) value;
```

Example :

```
 #include <iostream>
using namespace std;

int main() {
    double pi = 3.14159;

    int int_pi = (int) pi;

    cout << "Original: " << pi << ", Converted: " << int_pi << endl;
    return 0;
}
```

-Explicit conversion requires the programmer to specify the type.

Q 3. What are the different types of operators in C++? Provide examples of each

ANSWER:-

A) Arithmetic Operators

- Perform mathematical operations.
- **Examples**:

+, -, *, /, %

```
int a = 10, b = 3;
cout << "Addition: " << a + b << endl;
cout << "Remainder: " << a % b << endl;
```

B) Relational Operators

- Compare two values and return a Boolean result.
- **Examples**:

==, !=, >, <, >=, <=

```
cout << (a > b) << endl
```

C) Logical Operators

- Perform logical operations.
- **Examples**:

&& (AND), || (OR), ! (NOT)

```
bool x = true, y = false; cout << (x && y) << endl;
cout << (!x) << endl;
```

D) Assignment Operators

- Assign values to variables.
- **Examples**:

=, +=, -=, *=, /=, %=

```
a += 5
cout << a
```

E) Bitwise Operators

- Operate on binary values.
- **Examples**:

& (AND), | (OR), ^ (XOR), << (left shift), >> (right shift)

```
int x = 5
cout << (x << 1);
```

F) Increment/Decrement Operators

- **Examples**: ++ (increment), -- (decrement)

```
int c = 5;
cout << ++c; // Pre-increment: Output 6
cout << c--; // Post-decrement: Output 6 (then decrements)
```

Q4 . Explain the purpose and use of constants and literals in C++.

ANSWER:-

A) Constants:

- Constants are values that cannot be changed during program execution.
- Declared using the const keyword.
-EX

```
const double PI = 3.14159;
cout << "Value of PI: " << PI << endl;
```

B) **Literals:**

- Literals are **fixed values** used directly in code. They can be:

1) Integer literals:
10, -5

2) Floating-point literals:
3.14, 2.5e3

3) Character literals:
'A', 'B'

4) String literals:
"Hello, World!"

5) Boolean literals:
true, false

-EX

```
-#include <iostream>
-using namespace std;
-int main() {
-   int age = 20;
-   double pi = 3.14159
-   char grade = 'A';
-   string message = "Hi!";
-   cout << "Age: " << age << ", Pi: " << pi << ", Grade: " << grade << endl;
-   cout << "Message: " << message << endl;
-   return 0;
-}
```

Purpose of Constants and Literals:

Constants:
Ensure values remain unchanged, improving code safety.

Literals:
Represent fixed values for easy program readability and simplicity.

LAB EXERCISES:

Q1 .Variables and Constants o Write a C++ program that demonstrates the use of variables and constants. Create variables of different data types and perform operations on them. o Objective: Understand the difference between variables and constants

ANSWER :-

```
#include <iostream>
using namespace std;

int main() {
 int age = 20;
   double pi = 3.14159
   char grade = 'A';
   bool isStudent = true;
   string name = “HARRY POTTER”


   const double TAX_RATE = 0.05;

   double price = 100.0
   double tax = price * TAX_RATE;
   double totalPrice = price + tax;

   cout << "Name: " << name << endl;
   cout << "Age: " << age << ", Grade: " << grade << endl;
   cout << "Is Student: " << isStudent << endl;
   cout << "Price: $" << price << ", Tax: $" << tax << ", Total Price: $" << totalPrice << endl;
   cout << "Constant Tax Rate: " << TAX_RATE << endl;

   return 0;
}
```

Q2. Type Conversion o Write a C++ program that performs both implicit and explicit type conversions and prints the results. o Objective: Practice type casting in C++
ANSWER:-

```
#include <iostream>
using namespace std;

int main() {
int intVar = 10;
   double doubleVar = intVar

   double pi = 3.14159;
   int intPi = (int)pi

cout << "Implicit Type Conversion:" << endl;
   cout << "Original int value: " << intVar << endl;
   cout << "Converted to double: " << doubleVar << endl;

   cout << "\nExplicit Type Conversion:" << endl;
   cout << "Original double value: " << pi << endl;
   cout << "Converted to int: " << intPi << endl;

   return 0;
}
```

Q3. Operator Demonstration o Write a C++ program that demonstrates arithmetic, relational, logical, and bitwise operators. Perform operations using each type of operator and display the results. o Objective: Reinforce understanding of different types of operatorsin C++

ANSWER :-

```
#include <iostream>
using namespace std;
int main() {
   int a = 10, b = 3;
   cout << "Arithmetic Operators:" << endl;
   cout << "a + b = " << a + b << endl;
   cout << "a - b = " << a - b << endl;
   cout << "a * b = " << a * b << endl;
   cout << "a / b = " << a / b << endl;
   cout << "a % b = " << a % b << endl;
   cout << "\nRelational Operators:" << endl;
   cout << "(a > b): " << (a > b) << endl;
   cout << "(a < b): " << (a < b) << endl;
   cout << "(a == b): " << (a == b) << endl;
```

```
 bool x = true, y = false;
    cout << "\nLogical Operators:" << endl;
    cout << "(x && y): " << (x && y) << endl;
    cout << "(x || y): " << (x || y) << endl;
    cout << "(!x): " << (!x) << endl;
    int p = 5, q = 3; // Binary: p = 0101, q = 0011
    cout << "\nBitwise Operators:" << endl;
    cout << "(p & q): " << (p & q) << endl; // AND
    cout << "(p | q): " << (p | q) << endl; // OR
    cout << "(p ^ q): " << (p ^ q) << endl; // XOR
    cout << "(p << 1): " << (p << 1) << endl; // Left Shift
    cout << "(p >> 1): " << (p >> 1) << endl; // Right Shift
    return 0;
}
```

3. Functions and Scope

THEORY EXERCISE:

Q1 . What is a function in C++? Explain the concept of function declaration, definition, and calling.

ANSWER:-

- A **function** in C++ is a block of code designed to perform a specific task. Functions improve modularity, reusability, and organization of code.

- **Concepts:**

A) Function Declaration:

  - It tells the compiler about the function name, return type, and parameters before the function is defined.
  - Syntax:

```
return_type function_name(parameter_list);
```

  - Example:

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

- **B) Function Definition**:
  - It provides the actual implementation of the function.
  - Syntax:

```
return_type function_name(parameter_list) {
// Function body }
```

  - Example:

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

Function Calling:

- A function is executed when it is called by its name.
- Syntax:

  function_name(arguments);
- Example:

  int result = add(5, 3); // Function call

COMPLETE EXAMPLE :

```
#include <iostream>
using namespace std;

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

int main() {
    int x = 5, y = 10;
    int result = add(x, y); // Function call
    cout << "The sum is: " << result << endl;
    return 0;
}

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

Q2 .What is the scope of variables in C++? Differentiate between local and global scope.

Answer :-

-**Scope** determines where a variable can be accessed or modified within a program

-Types of Scope:

A) **Local Scope**:

- Variables declared inside a function or block are **local variables**.
- They can only be accessed within the function or block where they are declared.
- These variables are created when the function is called and destroyed when the function exits.

-EX

```
#include <iostream>
using namespace std;
void display() {
int localVar = 10;
cout << "Local variable: " << localVar << endl;
}

int main() {
display();
// cout << localVar; // Error: localVar is not accessible here
return 0;
}
```

B) **Global Scope**:

- Variables declared outside all functions are **global variables**.
- They can be accessed and modified from any function in the program.

- EX

```
#include <iostream>
using namespace std;
int globalVar = 20;
void display() {
```

```
 cout << "Global variable: " << globalVar << endl;
}
int main() {
    display();
    globalVar = 30; // Modify the global variable
    cout << "Modified global variable: " << globalVar << endl;
    return 0;
}
```

Q3. Explain recursion in C++ with an example.

Answer :-

Recursion is a programming technique where a function calls itself to solve a smaller instance of the same problem. It continues until a **base condition** is met.

Key Concepts:

Base Case:

The condition under which recursion stops.

Recursive Case:

The condition under which the function calls itself.

```
#include <iostream>
using namespace std;
int factorial(int n) {
    if (n == 0)
        return 1;
    else
        return n * factorial(n - 1);
}
int main() {
    int num = 5;
    cout << "Factorial of " << num << " is: " << factorial(num) << endl;
    return 0;
}
```

Q4. What are function prototypes in C++? Why are they used?

Answer:-

- A **function prototype** is a declaration of a function that specifies its name, return type, and parameters, but does not provide the function body.

-Syntax:

return_type function_name(parameter_list);

-**Purpose of Function Prototypes**:

A) **Allows Function Calls Before Definition**:

Prototypes enable the compiler to recognize a function even before its definition appears

B) Type Checking

The compiler checks that the function is called with the correct number and types of arguments.

-EX

```
#include <iostream>
using namespace std;

Int multiply(int a, int b);
int main() {
   int x = 4, y = 5;
   cout << "Product: " << multiply(x, y) << endl; // Function call
   return 0;
```

```
}
int multiply(int a, int b) {
    return a * b;
}
```

Why Use Prototypes?

They make it possible to organize functions in a program in any order.

They enable separate compilation, where the prototype appears in a header file, and the function definition appears in a source file.

LAB EXERCISES

Q1. Simple Calculator Using Functions o Write a C++ program that defines functions for basic arithmetic operations (add, subtract, multiply, divide). The main function should call these based on user input. o Objective: Practice defining and using functions in C++.

Annswer:-

```
#include <iostream>
using namespace std;

double add(double a, double b);
double subtract(double a, double b);
double multiply(double a, double b);
double divide(double a, double b);

int main() {
   double num1, num2;
   char op;
   cout << "Enter first number: ";
   cin >> num1;
   cout << "Enter an operator (+, -, *, /): ";
   cin >> op;
   cout << "Enter second number: ";
   cin >> num2;
   switch (op) {
      case '+‘:
```

```
            cout << "Result: " << add(num1, num2) << endl;
            break;
        case '-':
            cout << "Result: " << subtract(num1, num2) << endl;
            break;
        case '*':
            cout << "Result: " << multiply(num1, num2) << endl;
            break;
        case '/':
            if (num2 != 0)
                cout << "Result: " << divide(num1, num2) << endl;
            else
                cout << "Error: Division by zero!" << endl;
            break;
        default:
            cout << "Invalid operator!" << endl;
    }

    return 0;
}
double add(double a, double b) {
    return a + b;
}
double subtract(double a, double b) {
    return a - b;
}
double multiply(double a, double b) {
    return a * b;
}
double divide(double a, double b) {
    return a / b;
}
```

Q2. Factorial Calculation Using Recursion o Write a C++ program that calculates the factorial of a number using recursion. o Objective: Understand recursion in functions

Answer :-

```
#include <iostream>
using namespace std;
int factorial(int n) {
    if (n <= 1)
        return 1;
    else
        return n * factorial(n - 1);
}

int main() {
    int number;

    cout << "Enter a positive integer: ";
    cin >> number;

    if (number < 0)
        cout << "Factorial is not defined for negative numbers!"
<< endl;
    else
        cout << "Factorial of " << number << " is " <<
factorial(number) << endl;

    return 0;
}
```

3. Variable Scope o Write a program that demonstrates the difference between local and global variables in C++. Use functions to show scope. o Objective: Reinforce the concept of variable scope.

Answer :-

```
#include <iostream>
using namespace std;

int globalVar = 10;

void displayScope() {
    int localVar = 20;
    cout << "Inside function: " << endl;
    cout << "Global variable: " << globalVar << endl;
    cout << "Local variable: " << localVar << endl;
}

int main() {
    cout << "In main function: " << endl;
    cout << "Global variable: " << globalVar << endl;
    // cout << "Local variable: " << localVar; // Error:
localVar is not in scope here

    displayScope();

    return 0;
}
```

Control Flow Statements

THEORY EXERCISE

Q1. What are conditional statements in C++? Explain the if-else and switch statements

Answer:-

- Conditional statements are used to perform different actions based on different conditions. They control the flow of execution in a program by making decisions.

A) if-else Statement

The if-else statement is used to execute a block of code if a specified condition is true and another block of code if the condition is false.

Syntax:

```
if (condition) {
  // Code to execute if condition is true
} else {
  // Code to execute if condition is false
}
```

EXAMPLE :-

```
int x = 10;
if (x > 5) {
  cout << "x is greater than 5";
} else {
  cout << "x is not greater than 5";
}
```

B) switch Statement

The switch statement is used to perform different actions based on the value of a variable or expression. It is useful when there are multiple conditions to evaluate.

Syntax:

```
switch (expression) {
  case value1:
break;
  case value2:
        break;
  default:
}
```

EX :- int day = 3;

```
switch (day) {
  case 1:
    cout << "Monday";
    break;
   case 2:
    cout << "Wednesday";
    break;
  default:
    cout << "Invalid day";
}
```

Q2. What is the difference between for, while, and do-while loops in C++?

Answer :-

Feature	for Loop	while Loop	do-while Loop
Usage	Used when the number of iterations is known beforehand.	Used when the number of iterations is not known but depends on a condition.	Similar to while, but executes at least once.
Syntax	for(initialization; condition; increment)	while(condition)	do { } while(condition);
Execution	Condition is checked before each iteration.	Condition is checked before each iteration.	Condition is checked after each iteration.

Example	for (int i = 0; i < 5; i++) {
	  cout << i;
	}
		int i = 0;
		while (i < 5) {
		  cout << i;
		  i++;
		}
			int i = 0;
			do {
			  cout << i;
			  i++;
			} while (i < 5);

Q3. How are break and continue statements used in loops? Provide examples

Answer :-

A) **Break Statement**

The break statement is used to terminate the loop immediately, regardless of the loop's condition.

EX:-

```
for (int i = 0; i < 10; i++) {
   if (i == 5) {
      break; // Exits the loop when i == 5
   }
   cout << i << " ";
}
```

B) Continue Statement

The continue statement skips the current iteration and moves to the next iteration of the loop.

EX:-

```
for (int i = 0; i < 10; i++) {
   if (i == 5) {
      continue; // Skips the iteration when i == 5
   }
   cout << i << " ";
}
```

Q4 4. Explain nested control structures with an example

Anser:-

Nested control structures refer to control structures within other control structures, such as a loop inside another loop or an if-else inside a loop.

Example NESTED FOR LOOP:-

```
for (int i = 1; i <= 3; i++) { // Outer loop
   for (int j = 1; j <= 3; j++) { // Inner loop
      cout << "i = " << i << ", j = " << j << endl;
   }
}
```

EXAMple OF if-else :-

```
int x = 10, y = 20;
if (x > y) {
   cout << "x is greater than y";
} else {
   if (x == y) {
      cout << "x is equal to y";
   } else {
      cout << "x is less than y";
   }
}
```

LAB EXERCISES

Q1. Grade Calculator o Write a C++ program that takes a student's marks as input and calculates the grade based on if-else conditions. o Objective: Practice conditional statements(if-else).

Answer:-

```
#include <iostream>
using namespace std;
int main() {

   int marks;
   cout << "Enter your marks (0-100): ";
   cin >> marks;

   if (marks >= 90) {
      cout << "Grade: A" << endl;
   } else if (marks >= 80) {
      cout << "Grade: B" << endl;
   } else if (marks >= 70) {
      cout << "Grade: C" << endl;
   } else if (marks >= 60) {
      cout << "Grade: D" << endl;
   } else if (marks >= 50) {
      cout << "Grade: E" << endl;
   } else {
      cout << "Grade: F (Fail)" << endl;
   }   return 0;
}
```

Q2. 2. Number Guessing Game o Write a C++ program that asks the user to guess a number between 1 and 100. The program should provide hints if the guess is too high or too low. Use loops to allow the user multiple attempts. o Objective: Understand while loops and conditional logic

Answer:-

```
#include <iostream>
#include <cstdlib>
#include <ctime>
using namespace std;

int main() {
   srand(time(0));
   int number = rand() % 100 + 1;
   int guess;
   int attempts = 0;
   cout << "Guess the number (between 1 and 100): " << endl;

   while (true) {
      cout << "Enter your guess: ";
      cin >> guess;
      attempts++;
```

```
if (guess == number) {
        cout << "Congratulations! You guessed the correct number in " << attempts << " attempts." << endl;
        break;
    } else if (guess > number) {
        cout << "Too high! Try again." << endl;
    } else {
        cout << "Too low! Try again." << endl;
    }
  }

  return 0;
}
```

Q3 3. Multiplication Table o Write a C++ program to display the multiplication table of a given number using a for loop. o Objective: Practice using loops

ANSWER:-

```
#include <iostream>
using namespace std;

int main() {
  int num;
  cout << "Enter a number to display its multiplication table: ";
  cin >> num;

  for (int i = 1; i <= 10; i++) {
    cout << num << " x " << i << " = " << num * i << endl;
  }

  return 0;
}
```

Q4 4. Nested Control Structures o Write a program that prints a right-angled triangle using stars(*) with a nested loop. o Objective: Learn nested control structures

Answer:-

```
#include <iostream>
using namespace std;

int main() {
  int rows;
  cout << "Enter the number of rows for the triangle: ";
  cin >> rows;

  for (int i = 1; i <= rows; i++) { // Outer loop for rows
    for (int j = 1; j <= i; j++) { // Inner loop for columns
      cout << "*"; // Print star
    }
    cout << endl; // New line after each row
  }

  return 0;
}
```

5. Arrays and Strings

THEORY EXERCISE:

Q1 .What are arrays in C++? Explain the difference between single-dimensional and multi-dimensional arrays.

Answer :-

- Arrays are collections of elements of the same data type stored in contiguous memory locations. They allow easy access and manipulation of data using indices.

- A) **Single-dimensional Arrays:**
  - These are linear arrays, representing a sequence of elements accessed by a single index
  - EX :- int arr[5] = {1, 2, 3, 4, 5};
- B) **Multi-dimensional Arrays**:
  - These are arrays of arrays, where data is accessed using multiple indices. They represent tabular or grid-like structures.
  - EX :- int matrix[3][3] = { {1, 2, 3}, {4, 5, 6}, {7, 8, 9} };

Key Difference:
Single-dimensional arrays have one index for accessing elements, while multi-dimensional arrays require multiple indices.

Q2 Explain string handling in C++ with examples.

ANSWER:-

- C++ provides various ways to handle strings, primarily using **C-style strings** and the **std::string class.**
- **A) C-Style Strings**:
  - A character array terminated by a null character ('\0').
  - EX:- char name[10] = "Alice"; cout << "Name: " << name << endl;
- B) **std::string Class**:
  - Provides more functionality and is easier to use than C-style strings.
  - EX:-

```
#include <iostream>
#include <string> using namespace std;
 int main() {
 string str = "Hello, World!";
 cout << "String: " << str << endl; return 0;
 }
```

- **String Operations**:
- **Concatenation**: + operator or append() method.
- **Length**: length() or size() method.
- **Substrings**: substr(start, length) method.
- **Comparison**: compare() method or relational operators.

EX:-

```
string s1 = "Hello";
string s2 = "World";
string s3 = s1 + " " + s2;
cout << "Concatenated String: " << s3 << endl;
cout << "Length of String: " << s3.length() << endl;
```

3. How are arrays initialized in C++? Provide examples of both 1D and 2D arrays.

ANSWER :-

A) 1D Array Initialization:

- **Default Initialization**: All elements are initialized to default values (garbage for local variables).

-**Explicit Initialization**:

Example:

```
- int arr[5] = {1, 2, 3, 4, 5};
-int arr2[5] = {1, 2};
```

B) **2D Array Initialization**:

- **Default Initialization**: Similar to 1D arrays.

-**Explicit Initialization**:

Example:

```
 int matrix[3][3] = {
  {1, 2, 3},
   {4, 5, 6},
   {7, 8, 9}
};
int matrix2[3][3] = {0};
```

Q4. Explain string operations and functions in C++.

-The **std::string** class provides several built-in functions for string manipulation:

-A) **Concatenation**:

Example:

```
- string str1 = "Hello";
-string str2 = "World";
-string result = str1 + " " + str2;  // Using '+' operator
-str1.append(" World!");           // Using `append()`
```

B) **Length**:

```
 string str = "Hello";
cout << "Length: " << str.length() << endl;
```

C) **Substring Extraction**:

```
 string str = "Hello, World!";
cout << "Substring: " << str.substr(7, 5) << endl;
```

D) **Comparison**:

```
string s1 = "Apple";
string s2 = "Banana";
if (s1.compare(s2) < 0)
    cout << "s1 comes before s2" << endl;
```

E) **Finding Substring**:

```
string str = "Hello, World!";
size_t pos = str.find("World");
if (pos != string::npos)
    cout << "'World' found at position " << pos << endl;
```

F) **Replace**:

```
string str = "I love programming";
str.replace(2, 4, "adore");
cout << str << endl;
```

LAB EXERCISES:

Q1. Array Sum and Average o Write a C++ program that accepts an array of integers, calculates the sum and average, and displays the results. o Objective: Understand basic array manipulation

ANSWER :-

```
#include <iostream>
using namespace std;

int main() {
   int n, sum = 0;
   float average;

   cout << "Enter the number of elements: ";
   cin >> n;

   int arr[n];
   cout << "Enter " << n << " elements: ";
   for (int i = 0; i < n; i++) {
      cin >> arr[i];
      sum += arr[i];
   }

   average = static_cast<float>(sum) / n;

   cout << "Sum: " << sum << endl;
   cout << "Average: " << average << endl;

   return 0;
}
```

Q2 . Matrix Addition o Write a C++ program to perform matrix addition on two 2x2 matrices. o Objective: Practice multi-dimensional arrays.

ANSER :-

```
#include <iostream>
using namespace std;

int main() {
   int matrix1[2][2], matrix2[2][2], result[2][2];

   cout << "Enter elements of the first 2x2 matrix: " << endl;
   for (int i = 0; i < 2; i++) {
      for (int j = 0; j < 2; j++) {
         cin >> matrix1[i][j];
      }
   }

   cout << "Enter elements of the second 2x2 matrix: " << endl;
   for (int i = 0; i < 2; i++) {
      for (int j = 0; j < 2; j++) {
         cin >> matrix2[i][j];
      }
   }

   // Perform matrix addition
   for (int i = 0; i < 2; i++) {
      for (int j = 0; j < 2; j++) {
         result[i][j] = matrix1[i][j] + matrix2[i][j];
      }
   }

   cout << "Resultant matrix after addition: " << endl;
   for (int i = 0; i < 2; i++) {
      for (int j = 0; j < 2; j++) {
         cout << result[i][j] << " ";
      }
      cout << endl;
   }

   return 0;
}
```

Q3 . 3. String Palindrome Check o Write a C++ program to check if a given string is a palindrome (reads the same forwards and backwards). o Objective: Practice string operations.

Answer :-

```
#include <iostream>
#include <string>
using namespace std;

int main() {
   string str, reversedStr;
   bool isPalindrome = true;

   cout << "Enter a string: ";
   cin >> str;

   int n = str.length();

   // Check for palindrome
   for (int i = 0; i < n / 2; i++) {
      if (str[i] != str[n - i - 1]) {
         isPalindrome = false;
         break;
      }
   }

   if (isPalindrome) {
      cout << str << " is a palindrome." << endl;
   } else {
      cout << str << " is not a palindrome." << endl;
   }

   return 0;
}
```

6. Introduction to Object-Oriented Programming

THEORY EXERCISE:

Q1 Explain the key concepts of Object-Oriented Programming (OOP)

Answer :-

Object-Oriented Programming (OOP) is a programming paradigm that uses objects and classes to design and structure programs. The key concepts are:

A) Class and Object:

- A **class** is a blueprint for creating objects. It defines the properties (attributes) and behaviors (methods) of objects.
- An **object** is an instance of a class.

B) Encapsulation:

- Encapsulation is the bundling of data (attributes) and methods (functions) operating on the data into a single unit (class).
- It also involves restricting direct access to some components using access specifiers like private, protected, and public.

C)Inheritance:

- Inheritance allows a class (child class) to inherit properties and methods from another class (parent class), promoting code reuse.

D) Polymorphism:

- Polymorphism allows entities like functions or operators to operate in multiple ways depending on their input.
- It is achieved via function overloading, operator overloading, and virtual functions.

E) Abstraction:

- Abstraction hides unnecessary details and exposes only essential features of an object.
- It can be achieved using abstract classes and interfaces.

Q 2. What are classes and objectsin C++? Provide an example.
ANSWER :-

A) **Classes**: A class is a user-defined data type that acts as a blueprint for objects. It defines attributes (data members) and methods (member functions).

B) Objects: An object is an instance of a class that holds specific values for the attributes.

EXAMPLE :-

```
#include <iostream>
using namespace std;
class Car {
public:
    string brand;
    int year;

    void display() {
        cout << "Brand: " << brand << ", Year: " << year << endl;
    }
};
int main() {
    Car car1;
    car1.brand = "Toyota";
    car1.year = 2022;

    car1.display();
    return 0;
}
```

3. What is inheritance in C++? Explain with an example
ANSWer :-

Inheritance is a mechanism where one class (child or derived class) inherits attributes and methods from another class (parent or base class). It allows code reuse and establishes a relationship between classes.

EXAMPLE :-

```
 #include <iostream>
using namespace std;

class Animal {
public:
   void eat() {
      cout << "I can eat." << endl;
   }
};

class Dog : public Animal {
public:
   void bark() {
      cout << "I can bark." << endl;
   }
};

int main() {
   Dog dog1;
   dog1.eat();
   dog1.bark();
   return 0;
}
```

4. What is encapsulation in C++? How is it achieved in classes?
ANWER :-

-Encapsulation is the process of bundling data and methods into a single unit (class) and restricting direct access to some components to protect the integrity of the data.
-**Achieved by**:
private: Members are accessible only within the class.
protected: Members are accessible within the class and derived classes.
public: Members are accessible from anywhere.

Getter and Setter Methods:
Provide controlled access to private data members.

EXAMPLE :-

```
#include <iostream>
using namespace std;

class Employee {
private:
```

```
 int salary; r
public:
   void setSalary(int s) {
      salary = s;
   }
   int getSalary() {
      return salary;
   }
};

int main() {
   Employee emp1;
   emp1.setSalary(50000);
   cout << "Salary: " << emp1.getSalary() << endl;}
```

Advantages of Encapsulation:

-- Enhances code maintainability and readability.
-- Provides data security by restricting unauthorized access.
-- Helps in achieving abstraction.

LAB EXERCISES:

Q 1. Class for a Simple Calculator o Write a C++ program that defines a class Calculator with functions for addition, subtraction, multiplication, and division. Create objects to use these functions. o Objective: Introduce basic classstructure.

ANSWER :-

```
#include <iostream>
using namespace std;

class Calculator {
public:
    int add(int a, int b) {
        return a + b;
    }

    int subtract(int a, int b) {
        return a - b;
    }

    int multiply(int a, int b) {
        return a * b;
    }

    double divide(int a, int b) {
        if (b == 0) {
            cout << "Error: Division by zero!" << endl;
            return 0;
        }
        return static_cast<double>(a) / b;
    }
};

int main() {
    Calculator calc;
    int a, b;

    cout << "Enter two numbers: ";
    cin >> a >> b;

    cout << "Addition: " << calc.add(a, b) << endl;
    cout << "Subtraction: " << calc.subtract(a, b) << endl;
    cout << "Multiplication: " << calc.multiply(a, b) << endl;
    cout << "Division: " << calc.divide(a, b) << endl;

    return 0;
}
```

Q2. Class for Bank Account o Create a class BankAccount with data members like balance and member functions like deposit and withdraw. Implement encapsulation by keeping the data members private. o Objective: Understand encapsulation in classes.

Answer :-

```
#include <iostream>
using namespace std;

class BankAccount {
private:
   double balance;

public:
   BankAccount() {
      balance = 0.0;
   }

   void deposit(double amount) {
      if (amount > 0) {
         balance += amount;
         cout << "Deposited: " << amount << endl;
      } else {
         cout << "Invalid deposit amount!" << endl;
      }
   }

   void withdraw(double amount) {
      if (amount > 0 && amount <= balance) {
         balance -= amount;
         cout << "Withdrew: " << amount << endl;
      } else {
         cout << "Invalid withdrawal amount or insufficient balance!" << endl;
      }
   }

   double getBalance() const {
      return balance;
   }
};

int main() {
   BankAccount account;
   account.deposit(500);
   account.withdraw(200);
   cout << "Current Balance: " << account.getBalance() << endl;

   return 0;
}
```

Q3. Inheritance Example o Write a program that implements inheritance using a base class Person and derived classes Student and Teacher. Demonstrate reusability through inheritance. o Objective: Learn the concept of inheritance.

Answer :-

```
#include <iostream>
using namespace std;

class Person {
protected:
   string name;
   int age;

public:
   void setDetails(string n, int a) {
      name = n;
      age = a;
   }

   void displayDetails() {
      cout << "Name: " << name << ", Age: " << age << endl;
   }
};

class Student : public Person {
private:
   string course;

public:
   void setCourse(string c) {
      course = c;
   }

   void displayStudent() {
      displayDetails();
      cout << "Course: " << course << endl;
   }
};
class Teacher : public Person {
private:
   string subject;

public:
   void setSubject(string s) {
      subject = s;
   }

   void displayTeacher() {
      displayDetails();
      cout << "Subject: " << subject << endl;
   }
};

int main() {
   Student student;
   Teacher teacher;

   student.setDetails("Alice", 20);
   student.setCourse("Computer Science");
   cout << "Student Details:" << endl;
   student.displayStudent();

   teacher.setDetails("Mr. Smith", 40);
   teacher.setSubject("Mathematics");
   cout << "\nTeacher Details:" << endl;
   teacher.displayTeacher();

   return 0;
}
```