**Day-1**

**C++ Basic & Input/Output**

**Student Name: Kamalpreet Singh UID: 22BCS11720**

**Branch: BE - CSE Section/Group: FL\_IOT-603-A**

**Semester: 5th Date of Performance:19-12-24**

## Problem 1 - Sum of Natural Numbers up to N

## Aim-Calculate the sum of all natural numbers from 1 to n, where n is a positive integer. Use the formula: Sum=n×(n+1)/2 .​ Take n as input and output the sum of natural numbers from 1 to n .

**Solution-**

#include <iostream>

using namespace std;

int sum\_of\_natural\_numbers(int n) {

return n \* (n + 1) / 2;

}

int main() {

int n;

cout << "Enter a positive integer: ";

cin >> n;

int result = sum\_of\_natural\_numbers(n);

cout << "The sum of all natural numbers from 1 to " << n << " is " << result << "." << endl;

return 0;

}  
**Output-**

p1

**Problem 2 - Count digits in a number**

## Aim-Count the total number of digits in a given number n. The number can be a positive integer. For example, for the number 12345, the count of digits is 5. For a number like 900000, the count of digits is 6.

**Given an integer n, your task is to determine how many digits are present in n. This task will help you practice working with loops, number manipulation, and conditional logic.**

**Solution-**

#include <iostream>

#include <cmath>

using namespace std;

int countDigits(int n) {

if (n == 0) {

return 1;

}

int count = 0;

while (n != 0) {

n /= 10;

count++;

}

return count;

}

int main() {

int n;

cout << "Enter an integer: ";

cin >> n;

int digitCount = countDigits(n);

cout << "Number of digits in " << n << " is: " << digitCount << endl;

return 0;

}

**Output-**

**p2**

## **Problem 3-** Function Overloading for Calculating Area

## **Aim-**Write a program to calculate the area of different shapes using function overloading. Implement overloaded functions to compute the area of a circle, a rectangle, and a triangle.

**Solution-**

#include <iostream>

#include <cmath>

using namespace std;

const double PI = 3.14159;

double area(double radius) {

return PI \* radius \* radius;

}

double areaRectangle(double length, double breadth) {

return length \* breadth;

}

double areaTriangle(double base, double height) {

return 0.5 \* base \* height;

}

int main() {

double radius, length, breadth, base, height;

cout << "Enter radius of the circle: ";

cin >> radius;

cout << "Area of the circle: " << area(radius) << endl;

cout << "Enter length and breadth of the rectangle: ";

cin >> length >> breadth;

cout << "Area of the rectangle: " << areaRectangle(length, breadth) << endl;

cout << "Enter base and height of the triangle: ";

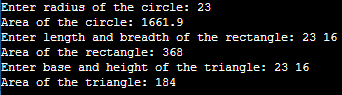
cin >> base >> height;

cout << "Area of the triangle: " << areaTriangle(base, height) << endl;

return 0;

}

**Output:**



## Problem 4: Polymorphism for Banking Transactions

## **Aim-** Design a C++ program to simulate a banking system using polymorphism. Create a base class Account with a virtual method calculateInterest(). Use the derived classes SavingsAccount and CurrentAccount to implement specific interest calculation logic:

## **SavingsAccount**: Interest = Balance × Rate × Time.

## **CurrentAccount**: No interest, but includes a maintenance fee deduction.

**Solution-**

#include <iostream>

using namespace std;

class Account {

public:

int balance;

Account(int b) : balance(b) {}

virtual double calculateInterest() {

return 0.0;

}

};

class SavingsAccount : public Account {

public:

double rate;

int time;

SavingsAccount(int b, double r, int t) : Account(b), rate(r), time(t) {}

double calculateInterest() override {

return (balance \* rate \* time) / 100.0;

}

};

class CurrentAccount : public Account {

public:

double maintenanceFee;

CurrentAccount(int b, double fee) : Account(b), maintenanceFee(fee) {}

double calculateInterest() override {

return -maintenanceFee;

}

};

int main() {

int accountType, balance, time;

double rate, maintenanceFee;

cout << "Enter Account Type (1 for Savings, 2 for Current): ";

cin >> accountType;

cout << "Enter Account Balance: ";

cin >> balance;

if (accountType == 1) {

cout << "Enter Interest Rate (%): ";

cin >> rate;

cout << "Enter Time (years): ";

cin >> time;

SavingsAccount savingsAccount(balance, rate, time);

cout << "Interest earned: " << savingsAccount.calculateInterest() << endl;

} else if (accountType == 2) {

cout << "Enter Monthly Maintenance Fee: ";

cin >> maintenanceFee;

CurrentAccount currentAccount(balance, maintenanceFee);

cout << "Maintenance Fee deduction: " << currentAccount.calculateInterest() << endl;

} else {

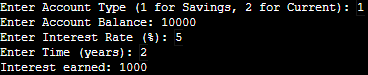
cout << "Invalid Account Type." << endl;

}

return 0;

}

**Output-**

****

**p4-2**

## Problem 5:Hierarchical Inheritance for Employee Management System

## Aim:Create a C++ program to simulate an employee management system using hierarchical inheritance. Design a base class Employee that stores basic details (name, ID, and salary). Create two derived classes:

## **Manager**: Add and calculate bonuses based on performance ratings.

## **Developer**: Add and calculate overtime compensation based on extra hours worked.

## The program should allow input for both types of employees and display their total earnings.

## Solution:

#include <iostream>

#include <string>

using namespace std;

class Employee {

public:

string name;

int ID;

int salary;

Employee(string n, int i, int s) : name(n), ID(i), salary(s) {

if (salary < 10000 || salary > 1000000) {

cout << "Invalid salary. Salary must be between 10000 and 1000000." << endl;

exit(1);

}

}

virtual int getTotalEarnings() {

return salary;

}

};

class Manager : public Employee {

public:

int performanceRating;

Manager(string n, int i, int s, int pr) : Employee(n, i, s), performanceRating(pr) {

if (performanceRating < 1 || performanceRating > 5) {

cout << "Invalid performance rating. Rating must be between 1 and 5." << endl;

exit(1);

}

}

int getTotalEarnings() override {

int bonus = 0;

switch (performanceRating) {

case 1:

bonus = 0;

break;

case 2:

bonus = salary \* 0.05;

break;

case 3:

bonus = salary \* 0.10;

break;

case 4:

bonus = salary \* 0.15;

break;

case 5:

bonus = salary \* 0.20;

break;

}

return salary + bonus;

}

};

class Developer : public Employee {

public:

int extraHours;

Developer(string n, int i, int s, int eh) : Employee(n, i, s), extraHours(eh) {}

int getTotalEarnings() override {

int overtimePay = extraHours \* (salary / 200); // Assuming 200 working hours per month

return salary + overtimePay;

}

};

int main() {

int employeeType, ID, salary, performanceRating, extraHours;

string name;

cout << "Enter Employee Type (1 for Manager, 2 for Developer): ";

cin >> employeeType;

cout << "Enter Name: ";

cin >> name;

cout << "Enter ID: ";

cin >> ID;

cout << "Enter Salary: ";

cin >> salary;

if (employeeType == 1) {

cout << "Enter Performance Rating (1-5): ";

cin >> performanceRating;

Manager manager(name, ID, salary, performanceRating);

cout << "Total Earnings for Manager " << manager.name << ": " << manager.getTotalEarnings() << endl;

} else if (employeeType == 2) {

cout << "Enter Extra Hours Worked: ";

cin >> extraHours;

Developer developer(name, ID, salary, extraHours);

cout << "Total Earnings for Developer " << developer.name << ": " << developer.getTotalEarnings() << endl;

} else {

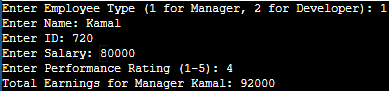
cout << "Invalid Employee Type." << endl;

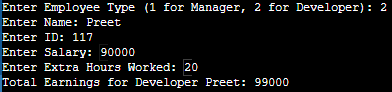
}

return 0;

}

**Output-**





**Problem 6-Check if a Number is Prime**

# **Aim-**Check if a given number n is a prime number. A prime number is a natural number greater than 1 that has no positive divisors other than 1 and itself.

# Solution:

#include <iostream>

#include <cmath>

using namespace std;

bool isPrime(int n) {

if (n <= 1)

return false;

for (int i = 2; i <= sqrt(n); ++i) {

if (n % i == 0)

return false;

}

return true;

}

int main() {

int n;

cout << "Enter a number to check if it is prime: ";

cin >> n;

if (isPrime(n)) {

cout << n << " is a prime number." << endl;

} else {

cout << n << " is not a prime number." << endl;

}

return 0;

}

Output-  


**Problem 7- Print Odd Numbers up to N**

# **Aim-**Print all odd numbers between 1 and n, inclusive. Odd numbers are integers that are not divisible by 2. These numbers should be printed in ascending order, separated by spaces.

**Solution-**

#include <iostream>

using namespace std;

void printOddNumbers(int n) {

for (int i = 1; i <= n; i += 2) {

cout << i << " ";

}

cout << endl;

}

int main() {

int n;

cout << "Enter a number to print all odd numbers up to n: ";

cin >> n;

if (n >= 1) {

cout << "Odd numbers up to " << n << ": ";

printOddNumbers(n);

} else {

cout << "Enter a number greater than or equal to 1." << endl;

}

return 0;

}

**Output:**



**Problem 8-Sum of Odd Numbers up to N**

# **Aim-** Calculate the sum of all odd numbers from 1 to n. An odd number is an integer that is not divisible by 2.The sum of odd numbers, iterate through all the numbers from 1 to n, check if each number is odd, and accumulate the sum.

**Solution:**

#include <iostream>

using namespace std;

int sumOfOddNumbers(int n) {

int sum = 0;

for (int i = 1; i <= n; i += 2) {

sum += i;

}

return sum;

}

int main() {

int n;

cout << "Enter a number to calculate the sum of odd numbers up to n: ";

cin >> n;

if (n >= 1) {

int sum = sumOfOddNumbers(n);

cout << "The sum of odd numbers up to " << n << " is: " << sum << endl;

} else {

cout << "Enter a number greater than or equal to 1." << endl;

}

return 0;

}

**Output:**



**Problem 9- Print Multiplication Table of a Number**

## **Aim:** Print the multiplication table of a given number n. A multiplication table for a number n is a list of products of n with integers from 1 to 10. For example, the multiplication table for 3 is: 3×1=3,3×2=6,…,3×10=30.

**Solution:**

#include <iostream>

using namespace std;

void printMultiplicationTable(int n) {

for (int i = 1; i <= 10; ++i) {

cout << n << " × " << i << " = " << n \* i << endl;

}

}

int main() {

int n;

cout << "Enter a number to print its multiplication table: ";

cin >> n;

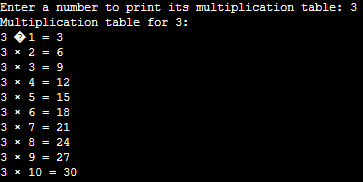
cout << "Multiplication table for " << n << ":\n";

printMultiplicationTable(n);

return 0;

}

**Output:**



**Problem 10 -Reverse a Number**

## **Aim:** Reverse the digits of a given number n. For example, if the input number is 12345, the output should be 54321. The task involves using loops and modulus operators to extract the digits and construct the reversed number.

**Solution:**

#include <iostream>

using namespace std;

int reverseNumber(int n) {

int reversed = 0;

while (n != 0) {

int digit = n % 10;

reversed = reversed \* 10 + digit;

n /= 10;

}

return reversed;

}

int main() {

int n;

cout << "Enter a number to reverse: ";

cin >> n;

cout << "The reversed number is: " << reverseNumber(n) << endl;

return 0;

}

**Output:**



**Problem 11: Find the Largest Digit in a Number**

## **Aim:** Find the largest digit in a given number n. For example, for the number 2734, the largest digit is 7. You need to extract each digit from the number and determine the largest one. The task will involve using loops and modulus operations to isolate the digits.

**Solution:**

#include <iostream>

using namespace std;

int findLargestDigit(int n) {

n = abs(n);

int largestDigit = 0;

while (n != 0) {

int digit = n % 10;

if (digit > largestDigit)

largestDigit = digit;

n /= 10;

}

return largestDigit;

}

int main() {

int n;

cout << "Enter a number to find its largest digit: ";

cin >> n;

cout << "The largest digit in " << n << " is: " << findLargestDigit(n) << endl;

return 0;

}

**Output:**



**Problem 12: Check if a Number is a Palindrome**

## **Aim:** Check whether a given number is a palindrome or not. A number is called a palindrome if it reads the same backward as forward. For example, 121 is a palindrome because reading it from left to right is the same as reading it from right to left. Similarly, 12321 is also a palindrome, but 12345 is not.

**Solution:**

#include <iostream>

using namespace std;

bool isPalindrome(int n) {

int original = n, reversed = 0, remainder;

n = abs(n);

while (n != 0) {

remainder = n % 10;

reversed = reversed \* 10 + remainder;

n /= 10;

}

return original == reversed;

}

int main() {

int n;

cout << "Enter a number to check if it's a palindrome: ";

cin >> n;

if (isPalindrome(n)) {

cout << n << " is a palindrome." << endl;

} else {

cout << n << " is not a palindrome." << endl;

}

return 0;

}

**Output:**

**Problem 13:Find the Sum of Digits of a Number**

## **Aim:** Calculate the sum of the digits of a given number n. For example, for the number 12345, the sum of the digits is 1+2+3+4+5=15. To solve this, you will need to extract each digit from the number and calculate the total sum.

**Solution:**

#include <iostream>

using namespace std;

int sumOfDigits(int n) {

int sum = 0;

n = abs(n);

while (n != 0) {

sum += n % 10;

n /= 10;

}

return sum;

}

int main() {

int n;

cout << "Enter a number to find the sum of its digits: ";

cin >> n;

cout << "The sum of the digits of " << n << " is: " << sumOfDigits(n) << endl;

return 0;

}

**Output:**



### Problem 13: **Function Overloading with Hierarchical Structure.**

## **Aim:** Write a program that demonstrates function overloading to calculate the salary of employees at different levels in a company hierarchy. Implement overloaded functions to compute salary for:

## Intern (basic stipend).

## Regular employee (base salary + bonuses).

## Manager (base salary + bonuses + performance incentives).

**Solution:**

#include <iostream>

using namespace std;

class Employee {

public:

string name;

int baseSalary;

Employee(string n, int b) : name(n), baseSalary(b) {}

virtual double calculateSalary() {

return baseSalary;

}

};

class Intern : public Employee {

public:

int stipend;

Intern(string n, int b, int s) : Employee(n, b), stipend(s) {}

double calculateSalary() override {

return stipend;

}

};

class RegularEmployee : public Employee {

public:

int bonus;

RegularEmployee(string n, int b, int bon) : Employee(n, b), bonus(bon) {}

double calculateSalary() override {

return baseSalary + bonus;

}

};

class Manager : public RegularEmployee {

public:

int performanceIncentive;

Manager(string n, int b, int bon, int pi) : RegularEmployee(n, b, bon), performanceIncentive(pi) {}

double calculateSalary() override {

return baseSalary + bonus + performanceIncentive;

}

};

int main() {

string name;

int baseSalary, stipend, bonus, performanceIncentive;

int employeeType;

cout << "Enter Employee Type (1 for Intern, 2 for Regular Employee, 3 for Manager): ";

cin >> employeeType;

cout << "Enter Name: ";

cin >> name;

cout << "Enter Base Salary: ";

cin >> baseSalary;

if (employeeType == 1) {

cout << "Enter Stipend for Intern: ";

cin >> stipend;

Intern intern(name, baseSalary, stipend);

cout << "Salary of Intern " << intern.name << ": " << intern.calculateSalary() << endl;

}

else if (employeeType == 2) {

cout << "Enter Bonus for Regular Employee: ";

cin >> bonus;

RegularEmployee regularEmployee(name, baseSalary, bonus);

cout << "Salary of Regular Employee " << regularEmployee.name << ": " << regularEmployee.calculateSalary() << endl;

}

else if (employeeType == 3) {

cout << "Enter Bonus for Manager: ";

cin >> bonus;

cout << "Enter Performance Incentive for Manager: ";

cin >> performanceIncentive;

Manager manager(name, baseSalary, bonus, performanceIncentive);

cout << "Salary of Manager " << manager.name << ": " << manager.calculateSalary() << endl;

}

else {

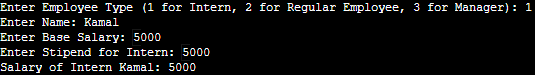
cout << "Invalid Employee Type." << endl;

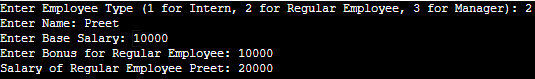
}

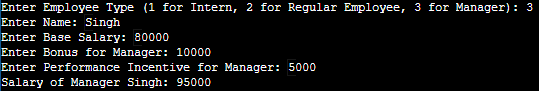
return 0;

}

**Output:**







**Problem 14 : Encapsulation with Employee Details**

## **Aim:**Write a program that demonstrates encapsulation by creating a class Employee. The class should have private attributes to store:

## Employee ID.

## Employee Name.

## Employee Salary.

## Provide public methods to set and get these attributes, and a method to display all details of the employee.

**Solution:**

#include <iostream>

#include <string>

using namespace std;

class Employee {

private:

int employeeID;

string employeeName;

double employeeSalary;

public:

void setEmployeeID(int id) {

employeeID = id;

}

int getEmployeeID() {

return employeeID;

}

void setEmployeeName(string name) {

employeeName = name;

}

string getEmployeeName() {

return employeeName;

}

void setEmployeeSalary(double salary) {

employeeSalary = salary;

}

double getEmployeeSalary() {

return employeeSalary;

}

void displayEmployeeDetails() {

cout << "Employee ID: " << employeeID << endl;

cout << "Employee Name: " << employeeName << endl;

cout << "Employee Salary: $" << employeeSalary << endl;

}

};

int main() {

Employee emp;

int id;

string name;

double salary;

cout << "Enter Employee ID: ";

cin >> id;

emp.setEmployeeID(id);

cout << "Enter Employee Name: ";

cin.ignore();

getline(cin, name);

emp.setEmployeeName(name);

cout << "Enter Employee Salary: ";

cin >> salary;

emp.setEmployeeSalary(salary);

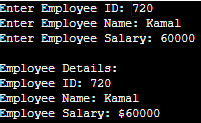
cout << endl << "Employee Details: " << endl;

emp.displayEmployeeDetails();

return 0;

}

**Output:**



**Problem 15: Inheritance with Student and Result Classes.**

## **Aim:**Create a program that demonstrates inheritance by defining:

## A base class Student to store details like Roll Number and Name.

## A derived class Result to store marks for three subjects and calculate the total and percentage.

**Solution:**

#include <iostream>

#include <string>

using namespace std;

class Student {

protected:

int rollNumber;

string name;

public:

void setRollNumber(int r) {

rollNumber = r;

}

int getRollNumber() {

return rollNumber;

}

void setName(string n) {

name = n;

}

string getName() {

return name;

}

};

class Result : public Student {

private:

float marks1, marks2, marks3;

float total, percentage;

public:

void setMarks(float m1, float m2, float m3) {

marks1 = m1;

marks2 = m2;

marks3 = m3;

}

void calculateResult() {

total = marks1 + marks2 + marks3;

percentage = (total / 300.0) \* 100;

}

void displayResult() {

cout << "Roll Number: " << getRollNumber() << endl;

cout << "Name: " << getName() << endl;

cout << "Marks in Subject 1: " << marks1 << endl;

cout << "Marks in Subject 2: " << marks2 << endl;

cout << "Marks in Subject 3: " << marks3 << endl;

cout << "Total Marks: " << total << endl;

cout << "Percentage: " << percentage << "%" << endl;

}

};

int main() {

Result student;

int rollNo;

string name;

float marks1, marks2, marks3;

cout << "Enter Roll Number: ";

cin >> rollNo;

student.setRollNumber(rollNo);

cout << "Enter Student Name: ";

cin.ignore();

getline(cin, name);

student.setName(name);

cout << "Enter Marks for Subject 1: ";

cin >> marks1;

cout << "Enter Marks for Subject 2: ";

cin >> marks2;

cout << "Enter Marks for Subject 3: ";

cin >> marks3;

student.setMarks(marks1, marks2, marks3);

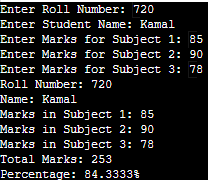
student.calculateResult();

student.displayResult();

return 0;

}

**Output:**



### **Problem 16: Polymorphism with Shape Area Calculation.**

**Aim:Create a program that demonstrates polymorphism by calculating the area of different shapes using a base class Shape and derived classes for Circle, Rectangle, and Triangle. Each derived class should override a virtual function to compute the area of the respective shape.**

**Solution:**

#include <iostream>

#include <cmath>

using namespace std;

class Shape {

public:

virtual double calculateArea() = 0;

virtual ~Shape() {}

};

class Circle : public Shape {

private:

double radius;

public:

Circle(double r) : radius(r) {}

double calculateArea() override {

return M\_PI \* radius \* radius;

}

};

class Rectangle : public Shape {

private:

double length, width;

public:

Rectangle(double l, double w) : length(l), width(w) {}

double calculateArea() override {

return length \* width;

}

};

class Triangle : public Shape {

private:

double base, height;

public:

Triangle(double b, double h) : base(b), height(h) {}

double calculateArea() override {

return 0.5 \* base \* height;

}

};

int main() {

Shape\* shape1 = new Circle(5);

Shape\* shape2 = new Rectangle(4, 6);

Shape\* shape3 = new Triangle(4, 5);

cout << "Area of Circle: " << shape1->calculateArea() << endl;

cout << "Area of Rectangle: " << shape2->calculateArea() << endl;

cout << "Area of Triangle: " << shape3->calculateArea() << endl;

delete shape1;

delete shape2;

delete shape3;

return 0;

} **Output:**



**Problem 17 : Implementing Polymorphism for Shape Hierarchies.**

## **Aim:**Write a program to demonstrate runtime polymorphism in C++ using a base class Shape and derived classes Circle, Rectangle, and Triangle. The program should use virtual functions to calculate and print the area of each shape based on user input.

**Solution:**

#include <iostream>

#include <cmath>

using namespace std;

class Shape {

public:

virtual double calculateArea() = 0;

virtual ~Shape() {}

};

class Circle : public Shape {

private:

double radius;

public:

Circle(double r) : radius(r) {}

double calculateArea() override {

return M\_PI \* radius \* radius;

}

};

class Rectangle : public Shape {

private:

double length, width;

public:

Rectangle(double l, double w) : length(l), width(w) {}

double calculateArea() override {

return length \* width;

}

};

class Triangle : public Shape {

private:

double base, height;

public:

Triangle(double b, double h) : base(b), height(h) {}

double calculateArea() override {

return 0.5 \* base \* height;

}

};

int main() {

int choice;

cout << "Choose a shape to calculate area:" << endl;

cout << "1. Circle\n2. Rectangle\n3. Triangle\n";

cout << "Enter your choice (1/2/3): ";

cin >> choice;

Shape\* shape = nullptr;

if (choice == 1) {

double radius;

cout << "Enter the radius of the circle: ";

cin >> radius;

shape = new Circle(radius);

} else if (choice == 2) {

double length, width;

cout << "Enter the length and width of the rectangle: ";

cin >> length >> width;

shape = new Rectangle(length, width);

} else if (choice == 3) {

double base, height;

cout << "Enter the base and height of the triangle: ";

cin >> base >> height;

shape = new Triangle(base, height);

} else {

cout << "Invalid choice!" << endl;

return 0;

}

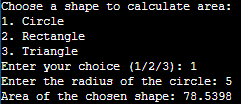
cout << "Area of the chosen shape: " << shape->calculateArea() << endl;

delete shape;

return 0;

}

**Output:**



**Problem 18: Matrix Multiplication Using Function Overloading**

## **Aim:**Implement matrix operations in C++ using function overloading. Write a function operate() that can perform:

## **Matrix Addition** for matrices of the same dimensions.

## **Matrix Multiplication** where the number of columns of the first matrix equals the number of rows of the second matrix.

**Solution:**#include <iostream>

using namespace std;

void operate(int a[10][10], int b[10][10], int result[10][10], int rows, int cols) {

for (int i = 0; i < rows; i++) {

for (int j = 0; j < cols; j++) {

result[i][j] = a[i][j] + b[i][j];

}

}

}

void operate(int a[10][10], int b[10][10], int result[10][10], int rowsA, int colsA, int rowsB, int colsB) {

if (colsA != rowsB) {

cout << "Matrix multiplication is not possible. Columns of A must equal rows of B." << endl;

return;

}

for (int i = 0; i < rowsA; i++) {

for (int j = 0; j < colsB; j++) {

result[i][j] = 0;

for (int k = 0; k < colsA; k++) {

result[i][j] += a[i][k] \* b[k][j];

}

}

}

}

void displayMatrix(int matrix[10][10], int rows, int cols) {

for (int i = 0; i < rows; i++) {

for (int j = 0; j < cols; j++) {

cout << matrix[i][j] << " ";

}

cout << endl;

}

}

int main() {

int a[10][10], b[10][10], result[10][10];

int rowsA, colsA, rowsB, colsB;

int choice;

cout << "Enter rows and columns for matrix A: ";

cin >> rowsA >> colsA;

cout << "Enter elements of matrix A:" << endl;

for (int i = 0; i < rowsA; i++) {

for (int j = 0; j < colsA; j++) {

cin >> a[i][j];

}

}

cout << "Enter rows and columns for matrix B: ";

cin >> rowsB >> colsB;

cout << "Enter elements of matrix B:" << endl;

for (int i = 0; i < rowsB; i++) {

for (int j = 0; j < colsB; j++) {

cin >> b[i][j];

}

}

cout << "Choose operation: 1 for Matrix Addition, 2 for Matrix Multiplication: ";

cin >> choice;

if (choice == 1) {

if (rowsA != rowsB || colsA != colsB) {

cout << "Matrix addition is not possible. Matrices must have the same dimensions." << endl;

} else {

operate(a, b, result, rowsA, colsA);

cout << "Matrix A + Matrix B = " << endl;

displayMatrix(result, rowsA, colsA);

}

} else if (choice == 2) {

operate(a, b, result, rowsA, colsA, rowsB, colsB);

cout << "Matrix A \* Matrix B = " << endl;

displayMatrix(result, rowsA, colsB);

} else {

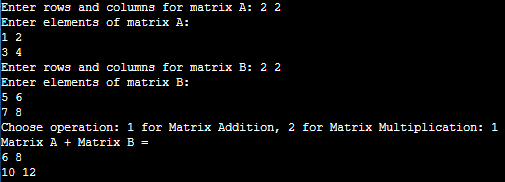
cout << "Invalid choice!" << endl;

}

return 0;

}

**Output:**



## **Problem 19: Polymorphism in Shape Classes Aim:**Design a C++ program using polymorphism to calculate the area of different shapes:

## A **Rectangle** (Area = Length × Breadth).

## A **Circle** (Area = π × Radius²).

## A **Triangle** (Area = ½ × Base × Height).

## Create a base class Shape with a pure virtual function getArea(). Use derived classes Rectangle, Circle, and Triangle to override this function.

**Solution:**#include <iostream>

#include <cmath>

using namespace std;

class Shape {

public:

virtual double getArea() = 0;

virtual ~Shape() {}

};

class Rectangle : public Shape {

private:

double length, breadth;

public:

Rectangle(double l, double b) : length(l), breadth(b) {}

double getArea() override {

return length \* breadth;

}

};

class Circle : public Shape {

private:

double radius;

public:

Circle(double r) : radius(r) {}

double getArea() override {

return M\_PI \* radius \* radius;

}

};

class Triangle : public Shape {

private:

double base, height;

public:

Triangle(double b, double h) : base(b), height(h) {}

double getArea() override {

return 0.5 \* base \* height;

}

};

void printArea(Shape\* shape) {

cout << "Area: " << shape->getArea() << endl;

}

int main() {

Rectangle rect(5.0, 3.0);

Circle circ(4.0);

Triangle tri(6.0, 4.0);

cout << "Rectangle:" << endl;

printArea(&rect);

cout << "Circle:" << endl;

printArea(&circ);

cout << "Triangle:" << endl;

printArea(&tri);

return 0;

}

**Output:**

