



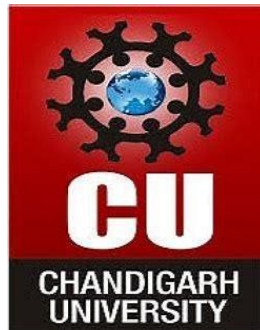
**CHANDIGARH  
UNIVERSITY**  
Discover. Learn. Empower.

**NAAC  
GRADE A+**  
Accredited University

## **UNIVERSITY INSTITUTE OF ENGINEERING**

**Department of Computer Science & Engineering**

**(BE-CSE)**



### **WINTER DOMAIN CAMP**

**Date : 19/12/2024**

**Submitted to:**

Faculty name: Er.Rajni Devi

**Submitted by:**

Name: ARYAN

UID: 22BCS15401

Section: IOT-620

## **PROBLEM 1**

**Objective:** Calculate the sum of all natural numbers from 1 to n, where n is a positive integer. Use the formula:  $\text{Sum} = n \times (n+1) / 2$ . Take n as input and output the sum of natural numbers from 1 to n.

**Task:** Write a program in C++ to print the Sum of all natural numbers upto n.

### **Code:**

```
#include<iostream>
using namespace std;
int main()
{
    int n,sum;
    cout<<"Enter the value = ";
    cin>>n;
    sum=n*(n+1)/2;
    cout<<"The sum of natural numbers upto "<<n <<" is = " << sum;
}
```

### **Output:**

```
Enter the value = 10
The sum of natural numbers upto 10 is = 55

=== Code Execution Successful ===
```

## **PROBLEM 2**

**Objective:** 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. To determine if a number is prime, iterate from 2 to  $\sqrt{n}$  and check if n is divisible by any number in this range. If it is divisible, it is not a prime number; otherwise, it is a prime.

**Task:** Write a program in C++ to find the entered number is prime or not

### **Code:**

```
#include <iostream>
using namespace std;
int main()
{
    int count=0,n;
    cout<<"Enter the number to be checked = ";
```

```

cin>>n;
for(int i=1;i<=n;i++)
{ if(n%i==0)
count++;
}
if(count==2)
cout<<"The entered number is a Prime Number";
else
cout<<"The entered number is not a Prime Number";
return 0; }

```

### **Output:**

```

Enter the number to be checked = 3
The entered number is a Prime Number

=== Code Execution Successful ===

```

## **PROBLEM 3**

**Objective:** 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. This problem is a simple introduction to loops and conditional checks. The goal is to use a loop to iterate over the numbers and check if they are odd using the condition  $i \% 2 \neq 0$ .

**Task:** Given an integer n, print all odd numbers from 1 to n, inclusive.

### **Code:**

```

#include <iostream>
using namespace std;
int main()
{
int i,n;
cout<<"Enter the value upto which number to be print = ";
cin>>n;
for (i=1;i<=n;i++)
{ if(i%2!=0)
cout<<i<<" ";
}
return 0;
}

```

### **Output:**

```
Enter the value upto which number to be print = 10
1,3,5,7,9,

=== Code Execution Successful ===
```

## **PROBLEM 4**

**Objective:** 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.

**Task:** Write a program in C++ to find the sum of odd number upto n.

### **Code:**

```
#include <iostream>
using namespace std;
int main()
{
    int n,i,sum=0;
    cout<<"Enter the number upto which sum is to be printed = ";
    cin>>n;
    for(i=1;i<=n;i++)
    { if(i%2!=0)
    {
        sum=sum+i;
    }
    }
    cout<<"The sum is "<<sum;
}
```

### **Output:**

```
Enter the number upto which sum is to be printed = 9
The sum is 25

=== Code Execution Successful ===
```

## **PROBLEM 5**

**Objective:** 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:  $3 \times 1 = 3, 3 \times 2 = 6, \dots, 3 \times 10 = 30$ .

**Aim:** Given an integer n, print the multiplication table of n from  $1 \times n$  to  $10 \times n$ .

### **Code:**

```
#include <iostream>
using namespace std;
int main()
{
    int n,i,mul;
    cout<<"Enter the number of which table you want to print = ";
    cin>>n;
    for(i=1;i<=10;i++)
    {
        mul=n*i;
        cout<<n<<"x"<<i<<"="<<mul<<endl;
    }
    return 0;
}
```

### **Output:**

```
Enter the number of which table you want to print = 13
13x1=13
13x2=26
13x3=39
13x4=52
13x5=65
13x6=78
13x7=91
13x8=104
13x9=117
13x10=130
```

```
=== Code Execution Successful ===
```

## **PROBLEM 6**

**Objective:** 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.

**Aim:** Given an integer n, print the total number of digits in n.

### **Code:**

```
#include <iostream>
using namespace std;
int main()
{
    int n, count;
    cout<<"Enter the number = ";
    cin>>n;
    do
    {
        n/=10;
        count++;
    }
    while(n!=0);
    cout<<"The number is a "<<count<<" digit number";
}
```

### **Output:**

```
Enter the number = 333
The number is a 3 digit number

=== Code Execution Successful ===
```

## **PROBLEM 7**

**Objective:** 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.

**Aim:** Given an integer n, print the numbers with its digits in reverse order.

### **Code:**

```
#include <iostream>
using namespace std;
int main() {
    int n;
    cout<< "Enter the number to be reversed = ";
    cin >> n;
    int reversedNumber = 0;
    while (n > 0) {
        int digit = n % 10;
        reversedNumber = reversedNumber * 10 + digit;
        n /= 10;
    }
    cout << "The reversed number is " << reversedNumber << endl;
    return 0;
}
```

### **Output:**

```
Enter the number to be reversed = 2584
The reversed number is 4852
```

```
=== Code Execution Successful ===|
```

## **PROBLEM 8**

**Objective:** 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.

**Aim:** Given an integer n, find and print the largest digit n.

### **Code:**

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

int main()
{ int n;
  cout<<"Enter the number = ";
  cin >> n;
  int largestDigit = 0;
  while (n > 0) {
```

```

int digit = n % 10;
if (digit > largestDigit)
{ largestDigit = digit;
}
n /= 10;
}
cout << "The largest digit from the number "<<n<<" is "<<largestDigit << endl;
return 0;
}

```

### **Output:**

```

Enter the number = 7256
The largest digit from the number is 7

=== Code Execution Successful ===

```

## **PROBLEM 9**

**Objective:** 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.

**Task:** Write a program in C++ to check whether a given number is palindrome or not.

### **Code:**

```

#include <iostream>
using namespace std;
int main()
{
int n,num,digit,add=0;
cout<<"Enter the number = ";
cin>>num;
n=num;
do
{
digit=num%10;
add=(add*10)+digit;
num=num/10;
}while(num!=0);
cout << "The reverse of the number is: " <<add<< endl;
if (n == add && n > 0)
cout << "The number is a palindrome.";
}

```



```
else
cout << "The number is not a palindrome.";
return 0;
}
```

### **Output:**

```
Enter the number = 363
The reverse of the number is: 363
The number is a palindrome.
```

```
=== Code Execution Successful ===
```

## **PROBLEM 10**

**Objective:** 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.

**Task:** Given an integer n, find and print the sum of its digits.

### **Code:**

```
#include <iostream>
using namespace std;
int main() {
int n;
cout<<"Enter the number = ";
cin >> n;
int sumOfDigits = 0;
while (n > 0) {
int digit = n % 10;
sumOfDigits += digit;
n /= 10;
}
cout << "The sum of the digits of the number is "<<sumOfDigits << endl;
return 0;
}
```

### **Output:**

```
Enter the number = 12345
The sum of the digits of the number is 15
```

```
=== Code Execution Successful ===
```

## **PROBLEM 11**

**Objective:** 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.

### **Code:**

```
#include <iostream>
#include <cmath>
using namespace std;
double area(double radius)
{
    return M_PI * radius * radius;
}
double area(double length, double breadth)
{
    return length * breadth;
}
double area(double base, double height)
{
    return 0.5 * base * height;
}
int main()
{
    double circleRadius = 5.0;
    cout << "Area of the circle with radius " << circleRadius << " : " << area(circleRadius) << endl;
    double rectLength = 10.0, rectBreadth = 5.0;

    cout << "Area of the rectangle with length " << rectLength << " and breadth " << rectBreadth <<
    " : " << area(rectLength, rectBreadth) << endl;

    double triangleBase = 6.0, triangleHeight = 4.0;
    cout << "Area of the triangle with base " << triangleBase << " and height " << triangleHeight <<
    " : " << area(triangleBase, triangleHeight, 0) << endl;

    return 0;
}
```

### **Output:**

```
Area of the circle with radius 5 : 78.5398
Area of the rectangle with length 10 and breadth 5 : 50
Area of the triangle with base 6 and height 4 : 12
```

```
=== Code Execution Successful ===
```

## **PROBLEM 12**

**Objective:** 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).

### **Code:**

```
#include <iostream>
using namespace std;
int calculateSalary(int stipend)
{ return stipend;
}
int calculateSalary(int baseSalary, int bonuses)
{ return baseSalary + bonuses;
}
int calculateSalary(int baseSalary, int bonuses, int incentives)
{ return baseSalary + bonuses + incentives;
}
int main()
{ int stipend;
cout<<"Enter the stipend = ";
cin >> stipend;
int baseSalaryEmployee, bonusesEmployee;
cout<<"Enter the baseSalary = ";
cin >> baseSalaryEmployee;
cout<<"Enter the bonus = ";
cin >> bonusesEmployee;
int baseSalaryManager, bonusesManager, incentivesManager;
cout<<"Enter the baseSalary = ";
cin >> baseSalaryEmployee;
cout<<"Enter the bonus = ";
cin >> bonusesEmployee;
cout<<"Enter the incentive = ";
cin >> incentivesManager;
cout << "Intern Salary : " << calculateSalary(stipend) << endl;
cout << "Employee Salary : " << calculateSalary(baseSalaryEmployee, bonusesEmployee) <<
endl;
cout << "Manager Salary : " << calculateSalary(baseSalaryManager, bonusesManager,
incentivesManager) << endl;
return 0;
}
```

### **Output:**

```
Enter the stipend = 2000
Enter the baseSalary = 100005
Enter the bonus = 500
Enter the baseSalary = 500000
Enter the bonus = 6050
Enter the incentive = 500
Intern Salary : 2000
Employee Salary : 506050
Manager Salary : 500
```

```
...Program finished with exit code 0
Press ENTER to exit console.[]
```

### **PROBLEM 13**

**Objective:** 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.

### **Code:**

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

class Employee
{ private:
    int employeeID;
    string employeeName;
    double employeeSalary;

public:
    // Constructor to initialize Employee with default values
    Employee() : employeeID(0), employeeName(""), employeeSalary(0.0) {}

    // Setter for Employee ID
    void setEmployeeID(int id) {
        employeeID = id;
    }
}
```

```

// Getter for Employee ID
int getEmployeeID() const {
    return employeeID;
}

// Setter for Employee Name
void setEmployeeName(const string& name) {
    employeeName = name;
}

// Getter for Employee Name
string getEmployeeName() const {
    return employeeName;
}

// Setter for Employee Salary
void setEmployeeSalary(double salary)
    { employeeSalary = salary;
}

// Getter for Employee Salary
double getEmployeeSalary() const {
    return employeeSalary;
}

// Method to display employee details
void displayDetails() const {
    cout << "Employee ID: " << employeeID << endl;
    cout << "Employee Name: " << employeeName << endl;
    cout << "Employee Salary: $" << employeeSalary << endl;
}
};

int main()
{ Employee emp;

    // Setting employee details
    emp.setEmployeeID(101);
    emp.setEmployeeName("John Doe");
    emp.setEmployeeSalary(75000.00);

    // Displaying employee details
    cout << "Employee Details:" << endl;
    emp.displayDetails();

    return 0;
}

```

### **Output:**

```
Employee Details:  
Employee ID: 101  
Employee Name: John Doe  
Employee Salary: $75000  
  
...Program finished with exit code 0  
Press ENTER to exit console.□
```

### **PROBLEM 14**

**Objective:** 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.

### **Code:**

```
#include <iostream>  
#include <string>  
using namespace std;  
  
class Student  
{ protected:  
    int rollNumber;  
    string name;  
  
public:  
    Student() : rollNumber(0), name("") {}  
  
    void setRollNumber(int roll)  
    { rollNumber = roll;  
    }  
  
    int getRollNumber() const  
    { return rollNumber;  
    }  
  
    void setName(const string& studentName)  
    { name = studentName;  
    }
```

```

string getName() const
{
    return name;
}

void displayStudentDetails() const {
    cout << "Roll Number: " << rollNumber << endl;
    cout << "Name: " << name << endl;
}
};

class Result : public Student
{ private:
    float marks[3];

public:
    Result() {
        for (int i = 0; i < 3; ++i)
            { marks[i] = 0.0;
            }
    }

    void setMarks(float mark1, float mark2, float mark3)
    { marks[0] = mark1;
      marks[1] = mark2;
      marks[2] = mark3;
    }

    float calculateTotal() const {
        return marks[0] + marks[1] + marks[2];
    }

    float calculatePercentage() const
    { return calculateTotal() / 3;
    }

    void displayResultDetails() const {
        cout << "Marks: " << marks[0] << ", " << marks[1] << ", " << marks[2] << endl;
        cout << "Total Marks: " << calculateTotal() << endl;
        cout << "Percentage: " << calculatePercentage() << "%" << endl;
    }
};

int main()
{
    Result
    student;

    student.setRollNumber(101);
    student.setName("Alice Smith");
    student.setMarks(85.5, 90.0, 78.0);

    cout << "Student Details:" << endl;

```

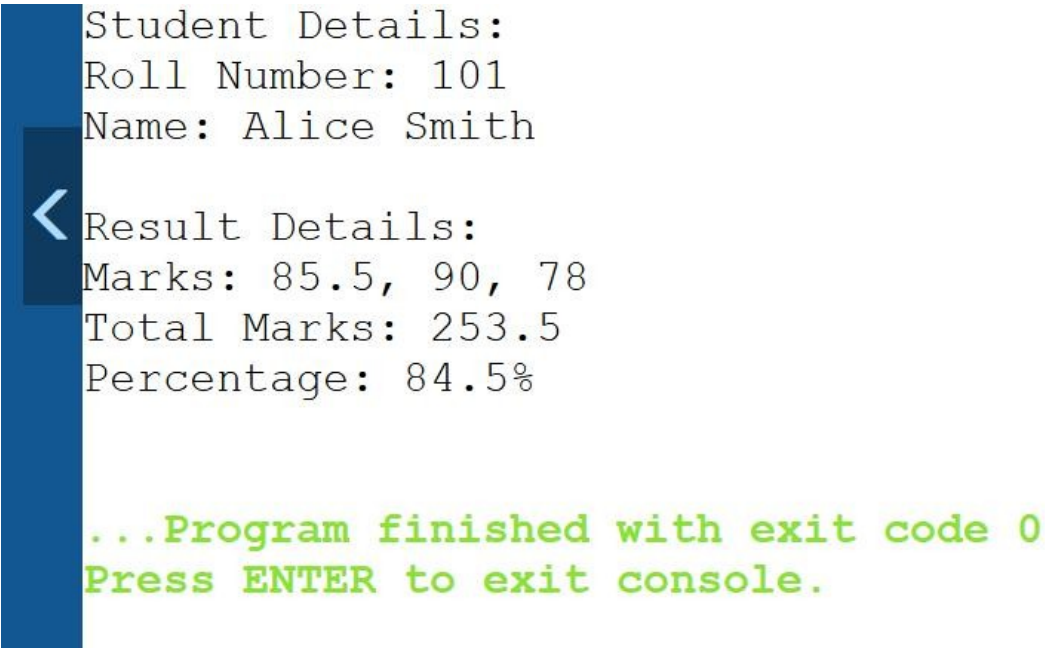
```
student.displayStudentDetails();
```



```
cout << "\nResult Details:" << endl;
student.displayResultDetails();

return 0;
}
```

### **Output:**



```
Student Details:
Roll Number: 101
Name: Alice Smith

< Result Details:
Marks: 85.5, 90, 78
Total Marks: 253.5
Percentage: 84.5%

...Program finished with exit code 0
Press ENTER to exit console.
```

## **PROBLEM 15**

**Objective:** 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.

### **Code:**

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

class Shape
{ public:
    virtual double calculateArea() const = 0;
    virtual ~Shape() {}
};

class Circle : public Shape
{ private:
    double radius;
```

```

public:
    Circle(double r) : radius(r) {}
    double calculateArea() const override
    { return M_PI * radius * radius;
    }
};

class Rectangle : public Shape
{ private:
    double length, breadth;

public:
    Rectangle(double l, double b) : length(l), breadth(b) {}
    double calculateArea() const override {
        return length * breadth;
    }
};

class Triangle : public Shape
{ private:
    double base, height;

public:
    Triangle(double b, double h) : base(b), height(h) {}
    double calculateArea() const override {
        return 0.5 * base * height;
    }
};

int main() {
    double radius, length, breadth, base, height;

    cout << "Enter radius of the circle: ";
    cin >> radius;
    Circle circle(radius);

    cout << "Enter length and breadth of the rectangle: ";
    cin >> length >> breadth;
    Rectangle rectangle(length, breadth);

    cout << "Enter base and height of the triangle: ";
    cin >> base >> height;
    Triangle triangle(base, height);

    cout << "\nArea of Circle: " << circle.calculateArea() << endl;
    cout << "Area of Rectangle: " << rectangle.calculateArea() << endl;
    cout << "Area of Triangle: " << triangle.calculateArea() << endl;

    return 0;
}

```

### **Output:**

```
Enter radius of the circle: 3
Enter length and breadth of the rectangle: 4
2
Enter base and height of the triangle: 3 6
Area of Circle: 28.2743
Area of Rectangle: 8
Area of Triangle: 9

...Program finished with exit code 0
Press ENTER to exit console.
```

### **PROBLEM 16**

**Objective:** 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.

### **Code:**

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

class Shape
{ public:
    virtual double calculateArea() const = 0;
    virtual ~Shape() {}
};

class Circle : public Shape
{ private:
    double radius;

public:
    Circle(double r) : radius(r) {}
    double calculateArea() const override
    { return M_PI * radius * radius;
    }
};
```

```

class Rectangle : public Shape
{ private:
    double length, breadth;

public:
    Rectangle(double l, double b) : length(l), breadth(b) {}
    double calculateArea() const override {
        return length * breadth;
    }
};

```

```

class Triangle : public Shape
{ private:
    double base, height;

public:
    Triangle(double b, double h) : base(b), height(h) {}
    double calculateArea() const override {
        return 0.5 * base * height;
    }
};

```

```

int main() {
    Shape* shape = nullptr;
    int choice;
    cout << "Choose a shape to calculate area:\n1. Circle\n2. Rectangle\n3. Triangle\n";
    cin >> choice;
    if(choice == 1)
    { double radius;
      cout << "Enter radius of the circle: ";
      cin >> radius;
      shape = new Circle(radius);
    } else if (choice == 2)
    { double length, breadth;
      cout << "Enter length and breadth of the rectangle: ";
      cin >> length >> breadth;
      shape = new Rectangle(length, breadth);
    } else if (choice == 3)
    { double base, height;
      cout << "Enter base and height of the triangle: ";
      cin >> base >> height;
      shape = new Triangle(base, height);
    } else {
      cout << "Invalid choice." << endl;
      return 1;
    }
    cout << "Area: " << shape->calculateArea() << endl;
    delete shape;
    return 0;
}

```



### **Output:**

```
Choose a shape to calculate area:
1. Circle
2. Rectangle
3. Triangle
< 2
Enter length and breadth of the rectangle: 3 6
Area: 18

...Program finished with exit code 0
Press ENTER to exit console.
```

### **PROBLEM 17**

**Objective:** 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.

### **Code:**

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

class Matrix
{ private:
    vector<vector<int>>> mat;
    int rows, cols;

public:
    Matrix(int r, int c) : rows(r), cols(c)
    { mat.resize(rows, vector<int>(cols, 0));
    }

    void input() {
        for (int i = 0; i < rows; ++i)
            { for (int j = 0; j < cols; ++j)
                {
                    cin >> mat[i][j];
                }
            }
    }
}
```



```

    }

void display() const {
    for (int i = 0; i < rows; ++i)
        { for (int j = 0; j < cols; ++j)
            {
                cout << mat[i][j] << " ";
            }
            cout << endl;
        }
}

Matrix operate(const Matrix& other, char op) const
{ if (op == '+') {
    if (rows != other.rows || cols != other.cols) {
        throw invalid_argument("Matrix dimensions must match for addition.");
    }
    Matrix result(rows, cols);
    for (int i = 0; i < rows; ++i)
        { for (int j = 0; j < cols; ++j)
            {
                result.mat[i][j] = mat[i][j] + other.mat[i][j];
            }
        }
    return result;
} else if (op == '*') {
    if (cols != other.rows) {
        throw invalid_argument("Number of columns of the first matrix must equal number of
rows of the second matrix.");
    }
    Matrix result(rows, other.cols);
    for (int i = 0; i < rows; ++i) {
        for (int j = 0; j < other.cols; ++j)
            { for (int k = 0; k < cols; ++k) {
                result.mat[i][j] += mat[i][k] * other.mat[k][j];
            }
        }
    }
    return result;
} else {
    throw invalid_argument("Invalid operation.");
}
}
};

int main() {
    int r1, c1, r2, c2;
    cout << "Enter rows and columns for first matrix: ";
    cin >> r1 >> c1;
    Matrix mat1(r1, c1);

```



```
cout << "Enter elements of the first matrix:" << endl;  
mat1.input();
```

```

cout << "Enter rows and columns for second matrix: ";
cin >> r2 >> c2;
Matrix mat2(r2, c2);

cout << "Enter elements of the second matrix:" << endl;
mat2.input();

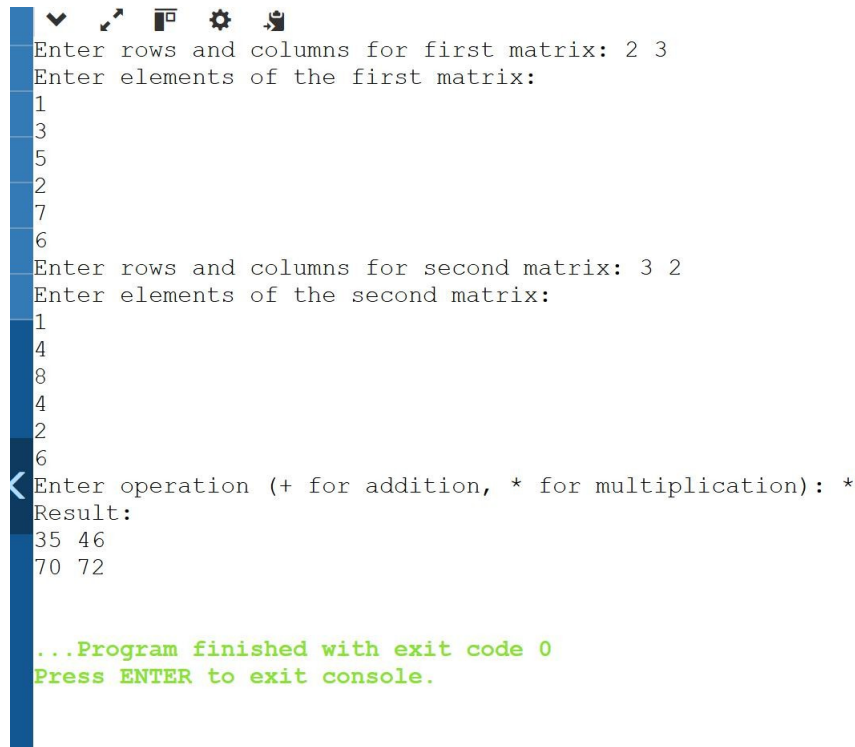
char op;
cout << "Enter operation (+ for addition, * for multiplication): ";
cin >> op;

try {
    Matrix result = mat1.operate(mat2, op);
    cout << "Result:" << endl;
    result.display();
} catch (const exception& e)
    { cout << e.what() << endl;
    }

return 0;
}

```

## **Output:**



```

Enter rows and columns for first matrix: 2 3
Enter elements of the first matrix:
1
3
5
2
7
6
Enter rows and columns for second matrix: 3 2
Enter elements of the second matrix:
1
4
8
4
2
6
Enter operation (+ for addition, * for multiplication): *
Result:
35 46
70 72

...Program finished with exit code 0
Press ENTER to exit console.

```

## **PROBLEM 18**

**Objective:** Design a C++ program using polymorphism to calculate the area of different shapes: A Rectangle (Area = Length  $\times$  Breadth), A Circle (Area =  $\pi \times \text{Radius}^2$ ), A Triangle (Area =  $\frac{1}{2} \times \text{Base} \times \text{Height}$ ). Create a base class Shape with a pure virtual function getArea(). Use derived classes Rectangle, Circle, and Triangle to override this function.

### **Code:**

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

class Shape
{ public:
    virtual double getArea() const = 0;
    virtual ~Shape() {}
};

class Rectangle : public Shape
{ private:
    double length, breadth;

public:
    Rectangle(double l, double b) : length(l), breadth(b) {}
    double getArea() const override {
        return length * breadth;
    }
};

class Circle : public Shape
{ private:
    double radius;

public:
    Circle(double r) : radius(r) {}
    double getArea() const 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() const override {
        return 0.5 * base * height;
    }
}
```

```

};

int main() {
    Shape* shape = nullptr;

    cout << "Choose a shape to calculate area:\n1. Rectangle\n2. Circle\n3. Triangle\n";
    int choice;
    cin >> choice;

    if (choice == 1) {
        double length, breadth;
        cout << "Enter length and breadth of the rectangle: ";
        cin >> length >> breadth;
        shape = new Rectangle(length, breadth);
    } else if (choice == 2)
    { double radius;
        cout << "Enter radius of the circle: ";
        cin >> radius;
        shape = new Circle(radius);
    } else if (choice == 3)
    { double base, height;
        cout << "Enter base and height of the triangle: ";
        cin >> base >> height;
        shape = new Triangle(base, height);
    } else {
        cout << "Invalid choice." << endl;
        return 1;
    }

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

    return 0;
}

```

### **Output:**

```

Choose a shape to calculate area:
1. Rectangle
2. Circle
3. Triangle
< 1
Enter length and breadth of the rectangle: 2 4
Area: 8

```

```

...Program finished with exit code 0
Press ENTER to exit console.

```

## **PROBLEM 19**

**Objective:** Create a C++ program using multiple inheritance to simulate a library system. Design two base classes: Book to store book details (title, author, and ISBN), Borrower to store borrower details (name, ID, and borrowed book). Create a derived class Library that inherits from both Book and Borrower. Use this class to track the borrowing and returning of books.

### **Code:**

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

class Book
{ protected:
    string title;
    string author;
    string ISBN;

public:
    void setBookDetails(const string& t, const string& a, const string& isbn)
    { title = t;
      author = a;
      ISBN = isbn;
    }

    void displayBookDetails() const {
        cout << "Book Title: " << title << endl;
        cout << "Author: " << author << endl;
        cout << "ISBN: " << ISBN << endl;
    }
};

class Borrower
{ protected:
    string name;
    int ID;
    string borrowedBook;

public:
    void setBorrowerDetails(const string& n, int id)
    { name = n;
      ID = id;
    }

    void borrowBook(const string& book)
    { borrowedBook = book;
    }

    void returnBook() {
```

```

        borrowedBook = "None";
    }

    void displayBorrowerDetails() const {
        cout << "Borrower Name: " << name << endl;
        cout << "Borrower ID: " << ID << endl;
        cout << "Borrowed Book: " << borrowedBook << endl;
    }
};

class Library : public Book, public Borrower
{ public:
    void borrow(const string& t, const string& a, const string& isbn)
        { setBookDetails(t, a, isbn);
          borrowBook(t);
        }

    void returnBorrowedBook()
        { returnBook();
        }

    void displayLibraryDetails() const
        { displayBookDetails();
          displayBorrowerDetails();
        }
};

int main()
{ Library
  library;

  string title, author, isbn, borrowerName;
  int borrowerID;

  cout << "Enter book details (Title, Author, ISBN): ";
  cin.ignore();
  getline(cin, title);
  getline(cin, author);
  getline(cin, isbn);

  cout << "Enter borrower details (Name, ID): ";
  getline(cin, borrowerName);
  cin >> borrowerID;

  library.setBorrowerDetails(borrowerName, borrowerID);
  library.borrow(title, author, isbn);

  cout << "\nBorrowing Details:\n";
  library.displayLibraryDetails();

  cout << "\nReturning Book...\n";

```

```
library.returnBorrowedBook();
```

```

cout << "\nUpdated Borrowing Details:\n";
library.displayLibraryDetails();

return 0;
}

```

### **Output:**



```

input
Enter book details (Title, Author, ISBN): The Gulliver's Travel
Jonathan Swift
1272
Enter borrower details (Name, ID): Mukesh Arya
50149

Borrowing Details:
Book Title: he Gulliver's Travel
Author: Jonathan Swift
ISBN: 1272
Borrower Name: Mukesh Arya
Borrower ID: 50149
Borrowed Book: he Gulliver's Travel

Returning Book...

Updated Borrowing Details:
Book Title: he Gulliver's Travel
Author: Jonathan Swift
ISBN: 1272
Borrower Name: Mukesh Arya
Borrower ID: 50149
Borrowed Book: None

...Program finished with exit code 0
Press ENTER to exit console.

```

## **PROBLEM 20**

**Objective:** 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.

### **Code:**

```

#include <iostream>
using namespace std;

class Account
{ protected:
    double balance;

public:
    Account(double initialBalance) : balance(initialBalance) {}

    virtual void calculateInterest() = 0;

```



```

virtual void displayBalance() const {
    cout << "Current Balance: " << balance << endl;
}

virtual ~Account() {}
};

class SavingsAccount : public Account
{ private:
    double rate;
    int time;

public:
    SavingsAccount(double initialBalance, double interestRate, int timePeriod)
        : Account(initialBalance), rate(interestRate), time(timePeriod) {}

    void calculateInterest() override {
        double interest = balance * rate * time;
        balance += interest;
        cout << "Interest Added: " << interest << endl;
    }
};

class CurrentAccount : public Account
{ private:
    double maintenanceFee;

public:
    CurrentAccount(double initialBalance, double fee)
        : Account(initialBalance), maintenanceFee(fee) {}

    void calculateInterest() override
    { balance -= maintenanceFee;
      cout << "Maintenance Fee Deducted: " << maintenanceFee << endl;
    }
};

int main() {
    Account* account = nullptr;

    cout << "Choose account type:\n1. Savings Account\n2. Current Account\n";
    int choice;
    cin >> choice;

    if (choice == 1) {
        double initialBalance, rate;
        int time;
        cout << "Enter initial balance, interest rate, and time period (in years): ";
        cin >> initialBalance >> rate >> time;
        account = new SavingsAccount(initialBalance, rate, time);
    } else if (choice == 2) {

```



```

        double initialBalance, fee;
        cout << "Enter initial balance and maintenance fee: ";
        cin >> initialBalance >> fee;
        account = new CurrentAccount(initialBalance, fee);
    } else {
        cout << "Invalid choice." << endl;
        return 1;
    }

    account->calculateInterest();
    account->displayBalance();

    delete account;
    return 0;
}

```

### **Output:**

```

Choose account type:
1. Savings Account
2. Current Account
2
Enter initial balance and maintenance fee: 12000
2000
Maintenance Fee Deducted: 2000
Current Balance: 10000

...Program finished with exit code 0
Press ENTER to exit console.

```

## **PROBLEM 21**

**Objective:** 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.

### **Code:**

```

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

class Employee
{ protected:
    string name;
    int id;
    double salary;

```

```

public:
    Employee(string n, int i, double s) : name(n), id(i), salary(s) {}

    virtual void calculateEarnings() = 0;

    void displayDetails() const {
        cout << "Name: " << name << endl;
        cout << "ID: " << id << endl;
        cout << "Base Salary: " << salary << endl;
    }

    virtual ~Employee() {}
};

class Manager : public Employee
{
private:
    double performanceRating;

public:
    Manager(string n, int i, double s, double rating)
        : Employee(n, i, s), performanceRating(rating) {}

    void calculateEarnings() override {
        double bonus = salary * (performanceRating / 100);
        double totalEarnings = salary + bonus;
        cout << "Performance Rating: " << performanceRating << "%" << endl;
        cout << "Bonus: " << bonus << endl;
        cout << "Total Earnings: " << totalEarnings << endl;
    }
};

class Developer : public Employee
{
private:
    int extraHoursWorked;
    double overtimeRate;

public:
    Developer(string n, int i, double s, int hours, double rate)
        : Employee(n, i, s), extraHoursWorked(hours), overtimeRate(rate) {}

    void calculateEarnings() override
    {
        double overtimeCompensation = extraHoursWorked * overtimeRate;
        double totalEarnings = salary + overtimeCompensation;
        cout << "Extra Hours Worked: " << extraHoursWorked << endl;
        cout << "Overtime Compensation: " << overtimeCompensation << endl;
        cout << "Total Earnings: " << totalEarnings << endl;
    }
};

```



```

int main()
{
    string name;
    int id;
    double salary, performanceRating, overtimeRate;
    int extraHoursWorked;

    cout << "Enter Manager details:\n";
    cout << "Name: ";
    cin >> name;
    cout << "ID: ";
    cin >> id;
    cout << "Salary: ";
    cin >> salary;
    cout << "Performance Rating (%): ";
    cin >> performanceRating;
    Manager manager(name, id, salary, performanceRating);

    cout << "\nEnter Developer details:\n";

    cout << "Name: ";
    cin >> name;
    cout << "ID: ";
    cin >> id;
    cout << "Salary: ";
    cin >> salary;
    cout << "Extra Hours Worked: ";
    cin >> extraHoursWorked;
    cout << "Overtime Rate: ";
    cin >> overtimeRate;
    Developer developer(name, id, salary, extraHoursWorked, overtimeRate);

    cout << "\nManager Details and Earnings:\n";

    manager.displayDetails();
    manager.calculateEarnings();

    cout << "\nDeveloper Details and Earnings:\n";
    developer.displayDetails();
    developer.calculateEarnings();

    return 0;
}

```

## **Output:**

```
Enter Manager details:
Name: John
ID: 101
Salary: 50000
Performance Rating (%): 12

Enter Developer details:
Name: Shreya
ID: 103
Salary: 45000
Extra Hours Worked: 3
Overtime Rate: 10

Manager Details and Earnings:
Name: John
ID: 101
< Base Salary: 50000
Performance Rating: 12%
Bonus: 6000
Total Earnings: 56000

Developer Details and Earnings:
Name: Shreya
ID: 103
Base Salary: 45000
Extra Hours Worked: 3
Overtime Compensation: 30
Total Earnings: 45030

...Program finished with exit code 0
Press ENTER to exit console.□
```

## **PROBLEM 22**

**Objective:** Create a C++ program to simulate a vehicle hierarchy using multi-level inheritance. Design a base class Vehicle that stores basic details (brand, model, and mileage). Extend it into the Car class to add attributes like fuel efficiency and speed. Further extend it into ElectricCar to include battery capacity and charging time. Implement methods to calculate:

Fuel Efficiency: Miles per gallon (for Car).

Range: Total distance the electric car can travel with a full charge.

## **Code:**

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

// Base class: Vehicle
class Vehicle
{ protected:
    string brand;
    string model;
    double mileage; // Miles driven

public:
    Vehicle(string b, string m, double ml) : brand(b), model(m), mileage(ml) {}

    void displayBasicDetails() const {
        cout << "Brand: " << brand << ", Model: " << model << ", Mileage: " << mileage << "
miles\n";
    }
};

// Derived class: Car
class Car : public Vehicle
{ protected:
    double fuelEfficiency; // Miles per gallon
    double speed;          // Average speed in mph

public:
    Car(string b, string m, double ml, double fe, double sp)
        : Vehicle(b, m, ml), fuelEfficiency(fe), speed(sp) {}

    void displayCarDetails() const
    { displayBasicDetails();
      cout << "Fuel Efficiency: " << fuelEfficiency << " mpg, Speed: " << speed << " mph\n";
    }

    double calculateFuelEfficiency() const
    { return fuelEfficiency;
    }
};

// Further derived class: ElectricCar
class ElectricCar : public Car
{ private:
    double batteryCapacity; // Battery capacity in kWh
    double chargingTime;    // Charging time in hours

public:
    ElectricCar(string b, string m, double ml, double fe, double sp, double bc, double ct)
        : Car(b, m, ml, fe, sp), batteryCapacity(bc), chargingTime(ct) {}
};
```





```

void displayElectricCarDetails() const
{ displayCarDetails();
  cout << "Battery Capacity: " << batteryCapacity << " kWh, Charging Time: " <<
chargingTime << " hours\n";
}

double calculateRange() const {
  return batteryCapacity * fuelEfficiency; // Approximation
}
};

int main() {
  // Create an ElectricCar object
  ElectricCar myCar("Tesla", "Model S", 12000, 100, 120, 85, 1.5);

  // Display details
  cout << "Electric Car Details:\n";
  myCar.displayElectricCarDetails();

  // Calculate and display fuel efficiency and range
  cout << "Calculated Fuel Efficiency: " << myCar.calculateFuelEfficiency() << " mpg\n";
  cout << "Calculated Range: " << myCar.calculateRange() << " miles\n";

  return 0;
}

```

### **Output:**

```

Electric Car Details:
Brand: Tesla, Model: Model S, Mileage: 12000 miles
Fuel Efficiency: 100 mpg, Speed: 120 mph
Battery Capacity: 85 kWh, Charging Time: 1.5 hours
Calculated Fuel Efficiency: 100 mpg
Calculated Range: 8500 miles

=== Code Execution Successful ===

```

## **PROBLEM 23**

**Objective:** Design a C++ program using function overloading to perform arithmetic operations on complex numbers. Define a Complex class with real and imaginary parts. Overload functions to handle the following operations:

Addition: Sum of two complex numbers.

Multiplication: Product of two complex numbers.

Magnitude: Calculate the magnitude of a single complex number.

The program should allow the user to select an operation, input complex numbers, and display results in the format  $a + bi$  or  $a - bi$  (where  $b$  is the imaginary part).

### **Code:**

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

class Complex
{ double real, imag;

public:
    // Constructor
    Complex(double r = 0, double i = 0) : real(r), imag(i) {}

    // Overloaded function to add two complex numbers
    Complex add(const Complex& c) const {
        return Complex(real + c.real, imag + c.imag);
    }

    // Overloaded function to multiply two complex numbers
    Complex multiply(const Complex& c) const {
        return Complex(real * c.real - imag * c.imag, real * c.imag + imag * c.real);
    }

    // Function to calculate magnitude
    double magnitude() const {
        return sqrt(real * real + imag * imag);
    }

    // Display function
    void display() const {
        if (imag >= 0)
            cout << real << " + " << imag << "i";
        else
            cout << real << " - " << -imag << "i";
    }
};

int main()
{ int
  choice;
  cout << "Select an operation:\n";
  cout << "1. Addition of two complex numbers\n";
  cout << "2. Multiplication of two complex numbers\n";
  cout << "3. Magnitude of a complex number\n";
```



```

cin >> choice;

if (choice == 1)
{ double r1, i1, r2,
  i2;
  cout << "Enter real and imaginary parts of the first complex number: ";
  cin >> r1 >> i1;
  cout << "Enter real and imaginary parts of the second complex number: ";
  cin >> r2 >> i2;

  Complex c1(r1, i1), c2(r2, i2);
  Complex result = c1.add(c2);

  cout << "Sum: ";
  result.display();
  cout << endl;
} else if (choice == 2)
{ double r1, i1, r2, i2;
  cout << "Enter real and imaginary parts of the first complex number: ";
  cin >> r1 >> i1;
  cout << "Enter real and imaginary parts of the second complex number: ";
  cin >> r2 >> i2;

  Complex c1(r1, i1), c2(r2, i2);
  Complex result = c1.multiply(c2);

  cout << "Product: ";
  result.display();
  cout << endl;
} else if (choice == 3)
{ double r, i;
  cout << "Enter real and imaginary parts of the complex number: ";
  cin >> r >> i;

  Complex c(r, i);
  cout << "Magnitude: " << c.magnitude() << endl;
} else {
  cout << "Invalid choice!" << endl;
}

return 0;
}

```

## **Output:**

```

Select an operation:
1. Addition of two complex numbers
2. Multiplication of two complex numbers
3. Magnitude of a complex number
1
Enter real and imaginary parts of the first complex number: 2 5
Enter real and imaginary parts of the second complex number: 2 3
Sum: 4 + 8i

=== Code Execution Successful ===

```

## **PROBLEM 24**

**Objective:** Create a C++ program that uses polymorphism to calculate the area of various shapes. Define a base class Shape with a virtual method calculateArea(). Extend this base class into the following derived classes:

Rectangle: Calculates the area based on length and width.

Circle: Calculates the area based on the radius.

Triangle: Calculates the area using base and height.

The program should use dynamic polymorphism to handle these shapes and display the area of each.

### **Code:**

```

#include <iostream>
#include <cmath>
#include <vector>
using namespace std;

// Base class
class Shape
{ public:
    virtual void calculateArea() const = 0; // Pure virtual function
    virtual ~Shape() {} // Virtual destructor
};

// Derived class for Rectangle
class Rectangle : public Shape {
    double length, width;
public:
    Rectangle(double l, double w) : length(l), width(w) {}
    void calculateArea() const override {
        if (length <= 0 || width <= 0) {
            cout << "Invalid dimensions. Length and width must be greater than 0." << endl;
            return;
        }
        cout << "Area of Rectangle: " << length * width << endl;
    }
}

```

```
};
```

```
// Derived class for Circle
```

```
class Circle : public Shape {
```

```
    double radius;
```

```
public:
```

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

```
    void calculateArea() const override
```

```
    { if (radius <= 0) {
```

```

        cout << "Invalid radius. Must be greater than 0." << endl;
        return;
    }
    cout << "Area of Circle: " << M_PI * radius * radius << endl;
}
};

// Derived class for Triangle
class Triangle : public Shape {
    double base, height;
public:
    Triangle(double b, double h) : base(b), height(h) {}
    void calculateArea() const override {
        if (base <= 0 || height <= 0) {
            cout << "Invalid dimensions. Base and height must be greater than 0." << endl;
            return;
        }
        cout << "Area of Triangle: " << 0.5 * base * height << endl;
    }
};

int main() {
    vector<Shape*> shapes; // Store shapes dynamically

    // Add shapes to the vector
    shapes.push_back(new Rectangle(4, 5));
    shapes.push_back(new Circle(3));
    shapes.push_back(new Triangle(6, 7));

    // Calculate the area of each shape
    for (const Shape* shape : shapes) {
        shape->calculateArea();
    }
    for (Shape* shape : shapes) {
        delete shape;
    }
    return 0;
}

```

### **Output:**

```

Area of Rectangle: 20
Area of Circle: 28.2743
Area of Triangle: 21

```

```

=== Code Execution Successful ===

```



## **PROBLEM 25**

**Objective:** Create a C++ program that demonstrates function overloading to calculate the area of different geometric shapes. Implement three overloaded functions named calculateArea that compute the area for the following shapes:

Circle: Accepts the radius.

Rectangle: Accepts the length and breadth.

Triangle: Accepts the base and height.

Additionally, use a menu-driven program to let the user choose the type of shape and input the respective parameters. Perform necessary validations on the input values.

### **Code:**

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

// Function to calculate the area of a circle
double calculateArea(double radius) {
    if (radius <= 0) {
        cout << "Invalid radius. Must be greater than 0." << endl;
        return -1;
    }
    return M_PI * radius * radius;
}

// Function to calculate the area of a rectangle
double calculateArea(double length, double breadth)
{ if (length <= 0 || breadth <= 0) {
    cout << "Invalid dimensions. Length and breadth must be greater than 0." << endl;
    return -1;
}
    return length * breadth;
}

// Function to calculate the area of a triangle
double calculateArea(double base, double height, int dummy)
{ if (base <= 0 || height <= 0) {
    cout << "Invalid dimensions. Base and height must be greater than 0." << endl;
    return -1;
}
    return 0.5 * base * height;
}

int main()
{ int
  choice; do
  {
```



```

cout << "\nChoose a shape to calculate the area:" << endl;
cout << "1. Circle" << endl;
cout << "2. Rectangle" << endl;
cout << "3. Triangle" << endl;
cout << "4. Exit" << endl;
cout << "Enter your choice: ";
cin >> choice;

switch (choice)
{
    case 1: {
        double radius;
        cout << "Enter the radius of the circle: ";
        cin >> radius;
        double area = calculateArea(radius);
        if (area != -1) {
            cout << "Area of the circle: " << area << endl;
        }
        break;
    }
    case 2: {
        double length, breadth;
        cout << "Enter the length and breadth of the rectangle: ";
        cin >> length >> breadth;
        double area = calculateArea(length, breadth);
        if (area != -1) {
            cout << "Area of the rectangle: " << area << endl;
        }
        break;
    }
    case 3: {
        double base, height;
        cout << "Enter the base and height of the triangle: ";
        cin >> base >> height;
        double area = calculateArea(base, height, 0); // The dummy argument is used to
differentiate the function
        if (area != -1) {
            cout << "Area of the triangle: " << area << endl;
        }
        break;
    }
    case 4:
        cout << "Exiting the program." << endl;
        break;
    default:
        cout << "Invalid choice. Please choose a valid option." << endl;
}
} while (choice != 4);

return 0;
}

```

### **Output:**

Choose a shape to calculate the area:

1. Circle
2. Rectangle
3. Triangle
4. Exit

Enter your choice: 1

Enter the radius of the circle: 5

Area of the circle: 78.5398

Choose a shape to calculate the area:

1. Circle
2. Rectangle
3. Triangle
4. Exit

Enter your choice: 2

Enter the length and breadth of the rectangle: 10 9

Area of the rectangle: 90

Choose a shape to calculate the area:

1. Circle
2. Rectangle
3. Triangle
4. Exit

Enter your choice: 3

Enter the base and height of the triangle: 22 21

Area of the triangle: 231