DOMAIN WINTER CAMP WORKSHEET
DAY-1

Student Name: Riya Kungotra   UID: 22BCS14298
Branch: BE-CSE   Section/Group: FL_IOT_603 (B)
Semester: 5th   Date: 19/12/2024

Very Easy

1) Sum of Natural Numbers up to N

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:
```cpp
#include <iostream>
using namespace std;
int sum_of_natural_numbers(int n) {
    return n * (n + 1) / 2; // Using the formula
}
int main() {
    int n;
    cout << "Enter a positive integer: ";
    cin >> n;
    // Calculating the sum
    int result = sum_of_natural_numbers(n);
    // Printing the result
    cout << "The sum of all natural numbers from 1 to " << n << " is " << result << "." << endl;
    return 0;

}
```

Output:

```
Enter a positive integer: 5
The sum of all natural numbers from 1 to 5 is 15



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

2) Check if a Number is Prime

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:

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

bool isPrime(int n) {
    if (n <= 1) return false; // Handle edge cases
    for (int i = 2; i <= sqrt(n); i++) {
        if (n % i == 0) {
            return false; // Divisible by a number other than 1 and itself
        }
    }
    return true; // No divisors found
}

int main() {
    int n;
    cout << "Enter a number: ";
    cin >> n;

    if (n < 2) {
        cout << "Not Prime" << endl;
        return 0;
    }

    if (isPrime(n)) {
        cout << "Prime" << endl;
    } else {
        cout << "Not Prime" << endl;
    }

    return 0;
}
```

Output:

```
Enter a number: 56
Not Prime



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

3) Print Odd Numbers up to N

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.

Solution:

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

int main() {
    int n;
    cout << "Enter the upper limit: ";
    cin >> n;

    if (n < 1) {
        cout << "Invalid input!" << endl;
        return 1;
    }

    for (int i = 1; i <= n; i += 2) {
        cout << i;
        if (i + 2 <= n) {
            cout << " "; // Add a space if it's not the last number
        }
    }
    cout << endl;

    return 0;

}
```

Output:

```
Enter the upper limit: 15
1 3 5 7 9 11 13 15


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

4) Sum of Odd Numbers up to N

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.

Solution:
```cpp
#include <iostream>
using namespace std;

int main() {
    int n;
    cout << "Enter the upper limit: ";
    cin >> n;

    if (n < 1) {
        cout << "Invalid input!" << endl;
        return 1;
    }

    int sum = 0;
    for (int i = 1; i <= n; i += 2) {
        sum += i;
    }

    cout << sum << endl;

    return 0;
}
```
Output:

```
Enter the upper limit: 15
64



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

5) Print Multiplication Table of a Number

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: 3×1=3,3×2=6,…,3×10=30.

Solution:
```cpp
#include <iostream>
using namespace std;

int main() {
    int n;
    cout << "Enter a number: ";
    cin >> n;

    if (n < 1 || n > 100) {
        cout << "Invalid input! Please enter a number between 1 and 100." << endl;
        return 1;
    }

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

    return 0;
}
```

Output:

```
Enter a number: 5
5 x 1 = 5
5 x 2 = 10
5 x 3 = 15
5 x 4 = 20
5 x 5 = 25
5 x 6 = 30
5 x 7 = 35
5 x 8 = 40
5 x 9 = 45
5 x 10 = 50


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

Easy:

1) Count Digits in a Number

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.

Solution:
```cpp
#include <iostream>
#include <cmath>
using namespace std;
int countDigits(int n) {
    if (n == 0) {
        return 1; // Special case: 0 has 1 digit
    }
    int count = 0;
    while (n != 0) {
        n /= 10; // Remove the last digit
        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:

```
Enter an integer: 63731
Number of digits in 63731 is: 5


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

2) Reverse a Number

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.

Solution:
```
#include <iostream>
using namespace std;
int main() {
    int n, reverse = 0; cin >> n;
    while (n > 0) {
        reverse = reverse * 10 + n % 10;
        n /= 10;
    }
    cout << reverse << endl;
    return 0;
}
```

Output:

```
4568
8654



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

3) Find the Largest Digit in a Number

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.

Solution:
```cpp
#include <iostream>
using namespace std;
int main() {
    int n, maxDigit = 0;
    cout << "Enter a number: ";
    cin >> n;
    while (n > 0) {
        maxDigit = max(maxDigit, n % 10);
        n /= 10;
    }
    cout << "Largest digit: " << maxDigit << endl;
    return 0;
}
```

Output:

```
Enter a number: 46879
Largest digit: 9



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

4) Find the Sum of Digits of a Number

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.

Solution:
```
#include <iostream>
using namespace std;
int main() {
    int n, sum = 0;
    cout << "Enter a number: ";
    cin >> n;
    while (n > 0) {
        sum += n % 10;
        n /= 10;
    }
    cout << "Sum of digits: " << sum << endl;
    return 0;
}
```

Output:

```
Enter a number: 546
Sum of digits: 15



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

Medium:

1) **Function Overloading for Calculating Area.**

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.

Solution:
```
#include <iostream>
#include <cmath>
using namespace std;
const double PI = 3.14159;

double area(double radius) {
    return PI * radius * radius;
}

double area(double length, double breadth) {
```

```cpp
    return length * breadth;
}

float area(float base, float height) {
    return 0.5 * base * height;
}

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

    // Get input for circle
    cout << "Enter radius of the circle: ";
    cin >> radius;
    cout << "Area of the circle: " << area(radius) << endl;

    // Get input for rectangle
    cout << "Enter length and breadth of the rectangle: ";
    cin >> length >> breadth;
    cout << "Area of the rectangle: " << area(length, breadth) << endl;

    // Get input for triangle
    cout << "Enter base and height of the triangle: ";
    cin >> base >> height;
    cout << "Area of the triangle: " << area(base, height) << endl;

    return 0;
}
```

Output:

```
Enter radius of the circle: 5
Area of the circle: 78.5397
Enter length and breadth of the rectangle: 6 5
Area of the rectangle: 30
Enter base and height of the triangle: 7 8
Area of the triangle: 28


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

2) Function Overloading with Hierarchical Structure.
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).

Solution:

```cpp
#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, baseSalary, bonuses, incentives;
   cout << "Enter stipend for intern: ";
   cin >> stipend;
   cout << "Enter base salary and bonuses for employee: ";
   cin >> baseSalary >> bonuses;
   cout << "Enter base salary, bonuses, and incentives for manager: ";
   cin >> baseSalary >> bonuses >> incentives;

   cout << "Intern Salary: " << calculateSalary(stipend) << endl;
   cout << "Employee Salary: " << calculateSalary(baseSalary, bonuses) << endl;
   cout << "Manager Salary: " << calculateSalary(baseSalary, bonuses, incentives) << endl;

   return 0;
}
```

Output:

```
Enter stipend for intern: 5000
Enter base salary and bonuses for employee:
    45000 15000
Enter base salary, bonuses, and incentives for
    manager: 70000 20000 10000
Intern Salary: 5000
Employee Salary: 90000
Manager Salary: 100000


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

3) **Encapsulation with Employee Details**
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.

Solution:

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

class Employee {
private:
    int id;
    string name;
    float salary;

public:
    void setID(int empID) { id = empID; }
    void setName(string empName) { name = empName; }
    void setSalary(float empSalary) { salary = empSalary; }

    int getID() { return id; }
    string getName() { return name; }
    float getSalary() { return salary; }
```

```cpp
    void displayDetails() {
        cout << "Employee ID: " << id << endl;
        cout << "Employee Name: " << name << endl;
        cout << "Employee Salary: " << salary << endl;
    }
};

int main() {
    Employee emp;
    int id;
    string name;
    float salary;

    cout << "Enter Employee ID: ";
    cin >> id;
    cin.ignore(); // Clear the input buffer for name input
    cout << "Enter Employee Name: ";
    getline(cin, name);
    cout << "Enter Employee Salary: ";
    cin >> salary;

    emp.setID(id);
    emp.setName(name);
    emp.setSalary(salary);
    emp.displayDetails();

    return 0;

}
```

Output:

```
Enter Employee ID: 211
Enter Employee Name: Kethrin
Enter Employee Salary: 87000
Employee ID: 211
Employee Name: Kethrin
Employee Salary: 87000


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

4) Inheritance with Student and Result Classes.

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.

Solution:

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

class Student {
protected:
    int rollNumber;
    string name;
public:
    void setDetails(int r, string n) {
        rollNumber = r;
        name = n;
    }
    void displayDetails() {
        cout << "Roll Number: " << rollNumber << endl;
        cout << "Name: " << name << endl;
    }
};

class Result : public Student {
private:
    int marks[3];
    int total;
    float percentage;
public:
    void setMarks(int m1, int m2, int m3) {
        marks[0] = m1;
        marks[1] = m2;
        marks[2] = m3;
    }
    void calculateResult() {
        total = marks[0] + marks[1] + marks[2];
        percentage = (total / 300.0) * 100;
    }
    void displayResult() {
        displayDetails();
        cout << "Marks: " << marks[0] << ", " << marks[1] << ", " << marks[2] << endl;
        cout << "Total: " << total << endl;
        cout << "Percentage: " << percentage << "%" << endl;
    }
```

```cpp
};

int main() {
    Result student;
    int rollNumber, marks1, marks2, marks3;
    string name;

    cout << "Enter Roll Number: ";
    cin >> rollNumber;
    cin.ignore();
    cout << "Enter Name: ";
    getline(cin, name);
    cout << "Enter marks for three subjects: ";
    cin >> marks1 >> marks2 >> marks3;

    student.setDetails(rollNumber, name);
    student.setMarks(marks1, marks2, marks3);
    student.calculateResult();
    student.displayResult();

    return 0;
}
```

Output:

```
Enter Roll Number: 41
Enter Name: Kethrin
Enter marks for three subjects: 89 99 79
Roll Number: 41
Name: Kethrin
Marks: 89, 99, 79
Total: 267
Percentage: 89%


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

Hard:

1) Implementing Polymorphism for Shape Hierarchies.

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.

Solution:

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

class Shape {
public:
    virtual void calculateArea() = 0; // Pure virtual function
};

class Circle : public Shape {
private:
    double radius;
public:
    Circle(double r) : radius(r) {}
    void calculateArea() override {
        cout << "Area of Circle: " << 3.14159 * radius * radius << endl;
    }
};

class Rectangle : public Shape {
private:
    double length, breadth;
public:
    Rectangle(double l, double b) : length(l), breadth(b) {}
    void calculateArea() override {
        cout << "Area of Rectangle: " << length * breadth << endl;
    }
};

class Triangle : public Shape {
private:
    double base, height;
public:
    Triangle(double b, double h) : base(b), height(h) {}
    void calculateArea() override {
        cout << "Area of Triangle: " << 0.5 * base * height << endl;
    }
};

int main() {
    double radius, length, breadth, base, height;
    cout << "Enter radius of the circle: ";
    cin >> radius;
    cout << "Enter length and breadth of the rectangle: ";
    cin >> length >> breadth;
```

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

    Circle circle(radius);
    Rectangle rectangle(length, breadth);
    Triangle triangle(base, height);

    Shape* shapes[] = {&circle, &rectangle, &triangle};
    for (Shape* shape : shapes) {
       shape->calculateArea();
    }

    return 0;
}
```

Output:

```
Enter radius of the circle: 5
Enter length and breadth of the rectangle: 4 5
Enter base and height of the triangle: 6 5
Area of Circle: 78.5397
Area of Rectangle: 20
Area of Triangle: 15



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

2) Matrix Multiplication Using Function Overloading
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.

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

class Matrix {
public:
    vector<vector<int>> operate(const vector<vector<int>>& A, const vector<vector<int>>& B,
int operation) {
```

```cpp
        if (operation == 1) return add(A, B);
        if (operation == 2) return multiply(A, B);
        return {{}}; // Default return for invalid operation
    }

private:
    vector<vector<int>> add(const vector<vector<int>>& A, const vector<vector<int>>& B) {
        int rows = A.size(), cols = A[0].size();
        if (B.size() != rows || B[0].size() != cols) {
            cout << "Invalid dimensions for operation." << endl;
            return {};
        }
        vector<vector<int>> result(rows, vector<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];
        return result;
    }

    vector<vector<int>> multiply(const vector<vector<int>>& A, const vector<vector<int>>& B)
{
        int rowsA = A.size(), colsA = A[0].size(), rowsB = B.size(), colsB = B[0].size();
        if (colsA != rowsB) {
            cout << "Invalid dimensions for operation." << endl;
            return {};
        }
        vector<vector<int>> result(rowsA, vector<int>(colsB, 0));
        for (int i = 0; i < rowsA; i++)
            for (int j = 0; j < colsB; j++)
                for (int k = 0; k < colsA; k++)
                    result[i][j] += A[i][k] * B[k][j];
        return result;
    }
};

void printMatrix(const vector<vector<int>>& matrix) {
    if (matrix.empty()) return;
    for (const auto& row : matrix) {
        for (int val : row) cout << val << " ";
        cout << endl;
    }
}

int main() {
    int operation, rowsA, colsA, rowsB, colsB;
    cout << "Enter operation (1 for Addition, 2 for Multiplication): ";
```

```
    cin >> operation;
    cout << "Enter dimensions of Matrix A (rows cols): ";
    cin >> rowsA >> colsA;
    vector<vector<int>> A(rowsA, vector<int>(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 dimensions of Matrix B (rows cols): ";
    cin >> rowsB >> colsB;
    vector<vector<int>> B(rowsB, vector<int>(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];

    Matrix matrix;
    vector<vector<int>> result = matrix.operate(A, B, operation);
    if (!result.empty()) printMatrix(result);

    return 0;
}
```

Output:

```
Enter operation (1 for Addition, 2 for
    Multiplication): 1
Enter dimensions of Matrix A (rows cols): 2 2
Enter elements of Matrix A:
3 5
6 7
Enter dimensions of Matrix B (rows cols): 2 2
Enter elements of Matrix B:
4 5
7 8
7 10
13 15


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

4) Implement Polymorphism for Banking Transactions
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.

Solution:

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

class Account {
protected:
    int balance;
public:
    virtual void calculateInterest() = 0; // Pure virtual function
    virtual ~Account() {}
};

class SavingsAccount : public Account {
    int rate, time;
public:
    SavingsAccount(int b, int r, int t) {
        balance = b;
        rate = r;
        time = t;
    }
    void calculateInterest() override {
        int interest = (balance * rate * time) / 100;
        cout << "Savings Account Interest: " << interest << endl;
    }
};

class CurrentAccount : public Account {
    int fee;
public:
    CurrentAccount(int b, int f) {
        balance = b;
        fee = f;
    }
    void calculateInterest() override {
        balance -= fee;
```

```cpp
        cout << "Balance after fee deduction: " << balance << endl;
    }
};

int main() {
    int accountType;
    cout << "Enter Account Type (1 for Savings, 2 for Current): ";
    cin >> accountType;

    if (accountType == 1) {
        int balance, rate, time;
        cout << "Enter Balance, Interest Rate, and Time: ";
        cin >> balance >> rate >> time;
        SavingsAccount sa(balance, rate, time);
        sa.calculateInterest();
    } else if (accountType == 2) {
        int balance, fee;
        cout << "Enter Balance and Maintenance Fee: ";
        cin >> balance >> fee;
        CurrentAccount ca(balance, fee);
        ca.calculateInterest();
    } else {
        cout << "Invalid account type." << endl;
    }

    return 0;
}
```

Output:

```
Enter Account Type (1 for Savings, 2 for Current
    ): 1
Enter Balance, Interest Rate, and Time: 5000 5 1
Savings Account Interest: 250


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

Very Hard

1) Hierarchical Inheritance for Employee Management System
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.

Solution:

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

class Employee {
protected:
    string name;
    int id;
    int salary;
public:
    Employee(string n, int i, int s) : name(n), id(i), salary(s) {}
    virtual void calculateEarnings() = 0; // Pure virtual function
    virtual ~Employee() {}
};

class Manager : public Employee {
    int rating;
public:
    Manager(string n, int i, int s, int r) : Employee(n, i, s), rating(r) {}
    void calculateEarnings() override {
        int bonus = salary * rating * 0.10;
        cout << "Employee: " << name << " (ID: " << id << ")" << endl;
        cout << "Role: Manager" << endl;
        cout << "Base Salary: " << salary << endl;
        cout << "Bonus: " << bonus << endl;
        cout << "Total Earnings: " << salary + bonus << endl;
    }
};

class Developer : public Employee {
    int extraHours;
public:
    Developer(string n, int i, int s, int e) : Employee(n, i, s), extraHours(e) {}
    void calculateEarnings() override {
        int overtime = extraHours * 500;
        cout << "Employee: " << name << " (ID: " << id << ")" << endl;
        cout << "Role: Developer" << endl;
        cout << "Base Salary: " << salary << endl;
```

```cpp
        cout << "Overtime Compensation: " << overtime << endl;
        cout << "Total Earnings: " << salary + overtime << endl;
    }
};

int main() {
    int type;
    cout << "Enter Employee Type (1 for Manager, 2 for Developer): ";
    cin >> type;

    if (type == 1) {
        string name;
        int id, salary, rating;
        cout << "Enter Name, ID, Salary, and Rating: ";
        cin >> name >> id >> salary >> rating;
        Manager m(name, id, salary, rating);
        m.calculateEarnings();
    } else if (type == 2) {
        string name;
        int id, salary, extraHours;
        cout << "Enter Name, ID, Salary, and Extra Hours: ";
        cin >> name >> id >> salary >> extraHours;
        Developer d(name, id, salary, extraHours);
        d.calculateEarnings();
    } else {
        cout << "Invalid employee type." << endl;
    }

    return 0;
}
```

Output:

```
Enter Employee Type (1 for Manager, 2 for
    Developer): 1
Enter Name, ID, Salary, and Rating: Kethrin 202
    89000 5
Employee: Kethrin (ID: 202)
Role: Manager
Base Salary: 89000
Bonus: 44500
Total Earnings: 133500


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

2) Multi-Level Inheritance for Vehicle Simulation
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.

Solution:

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

class Vehicle {
protected:
    string brand;
    string model;
    double mileage;
public:
    Vehicle(string b, string m, double mil) : brand(b), model(m), mileage(mil) {}
    virtual void displayDetails() = 0; // Pure virtual function
    virtual ~Vehicle() {}
};

class Car : public Vehicle {
protected:
    double fuel;
    double distance;
public:
    Car(string b, string m, double mil, double f, double d) : Vehicle(b, m, mil), fuel(f), distance(d)
{}
    void displayDetails() override {
        double fuelEfficiency = distance / fuel;
        cout << "Vehicle: " << brand << " " << model << endl;
        cout << "Mileage: " << mileage << endl;
        cout << "Fuel Efficiency: " << fuelEfficiency << " miles/gallon" << endl;
    }
};

class ElectricCar : public Car {
private:
    double batteryCapacity;
    double efficiency;
public:
    ElectricCar(string b, string m, double mil, double f, double d, double bc, double e)
        : Car(b, m, mil, f, d), batteryCapacity(bc), efficiency(e) {}
```

```cpp
    void displayDetails() override {
        double range = batteryCapacity * efficiency;
        cout << "Vehicle: " << brand << " " << model << endl;
        cout << "Mileage: " << mileage << endl;
        cout << "Range: " << range << " miles" << endl;
    }
};

int main() {
    int type;
    cout << "Enter Vehicle Type (1 for Car, 2 for Electric Car): ";
    cin >> type;

    if (type == 1) {
        string brand, model;
        double mileage, fuel, distance;
        cout << "Enter Brand, Model, Mileage, Fuel, Distance: ";
        cin >> brand >> model >> mileage >> fuel >> distance;
        Car c(brand, model, mileage, fuel, distance);
        c.displayDetails();
    } else if (type == 2) {
        string brand, model;
        double mileage, fuel, distance, batteryCapacity, efficiency;
        cout << "Enter Brand, Model, Mileage, Fuel, Distance, Battery Capacity, Efficiency: ";
        cin >> brand >> model >> mileage >> fuel >> distance >> batteryCapacity >> efficiency;
        ElectricCar ec(brand, model, mileage, fuel, distance, batteryCapacity, efficiency);
        ec.displayDetails();
    } else {
        cout << "Invalid vehicle type." << endl;
    }

    return 0;
}
```

Output:

```
Enter Vehicle Type (1 for Car, 2 for Electric
    Car): 1
Enter Brand, Model, Mileage, Fuel, Distance:
    Toyota
Corolla
30000
15
300
Vehicle: Toyota Corolla
Mileage: 30000
Fuel Efficiency: 20 miles/gallon


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

4) Polymorphism for Shape Area Calculations
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.

Solution:
```cpp
#include <iostream>
#include <cmath>
using namespace std;
class Shape {
public:
    virtual void calculateArea() = 0;  // Pure virtual function
    virtual ~Shape() {}
};
class Rectangle : public Shape {
private:
    float length, width;
public:
    Rectangle(float l, float w) : length(l), width(w) {}
    void calculateArea() override {
        float area = length * width;
        cout << "Shape: Rectangle" << endl;
```

```cpp
        cout << "Area: " << area << endl;
    }
};

class Circle : public Shape {
private:
    float radius;
public:
    Circle(float r) : radius(r) { }
    void calculateArea() override {
        float area = 3.14159 * radius * radius;
        cout << "Shape: Circle" << endl;
        cout << "Area: " << area << endl;
    }
};
class Triangle : public Shape {
private:
    float base, height;
public:
    Triangle(float b, float h) : base(b), height(h) { }
    void calculateArea() override {
        float area = 0.5 * base * height;
        cout << "Shape: Triangle" << endl;
        cout << "Area: " << area << endl;
    }
};
int main() {
    int type;
    cout << "Enter Shape Type (1 for Rectangle, 2 for Circle, 3 for Triangle): ";
    cin >> type;
    Shape* shape = nullptr;
    if (type == 1) {
        float length, width;
        cout << "Enter Length and Width: ";
        cin >> length >> width;
        shape = new Rectangle(length, width);
    } else if (type == 2) {
        float radius;
        cout << "Enter Radius: ";
        cin >> radius;
        shape = new Circle(radius);
    } else if (type == 3) {
        float base, height;
        cout << "Enter Base and Height: ";
        cin >> base >> height;
        shape = new Triangle(base, height);
```

```
    } else {
        cout << "Invalid shape type" << endl;
        return 0;
    }

    shape->calculateArea();
    delete shape;
    return 0;
}
```

Output:

```
Enter Shape Type (1 for Rectangle, 2 for Circle,
    3 for Triangle): 1
Enter Length and Width: 50 31
Shape: Rectangle
Area: 1550

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

5) Advanced Function Overloading for Geometric Shapes
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.
Solution:
```cpp
#include <iostream>
#include <cmath>
using namespace std;

// Function to calculate the area of a circle
double calculateArea(double radius) {
    return 3.14159 * radius * radius;
}
```

```cpp
// Function to calculate the area of a rectangle
double calculateArea(double length, double breadth) {
    return length * breadth;
}

// Function to calculate the area of a triangle
float calculateArea(float base, float height) {
    return 0.5 * base * height;
}

int main() {
    int choice;
    cout << "Choose a shape (1 for Circle, 2 for Rectangle, 3 for Triangle): ";
    cin >> choice;

    if (choice == 1) {
        double radius;
        cout << "Enter radius of the circle: ";
        cin >> radius;
        if (radius <= 0) {
            cout << "Invalid input. Radius must be positive." << endl;
            return 0;
        }
        double area = calculateArea(radius);
        cout << "Shape: Circle" << endl;
        cout << "Radius: " << radius << endl;
        cout << "Area: " << area << endl;

    } else if (choice == 2) {
        double length, breadth;
        cout << "Enter length and breadth of the rectangle: ";
        cin >> length >> breadth;
        if (length <= 0 || breadth <= 0) {
            cout << "Invalid input. Length and breadth must be positive." << endl;
            return 0;
        }
        double area = calculateArea(length, breadth);
        cout << "Shape: Rectangle" << endl;
        cout << "Length: " << length << endl;
        cout << "Breadth: " << breadth << endl;
        cout << "Area: " << area << endl;

    } else if (choice == 3) {
        float base, height;
        cout << "Enter base and height of the triangle: ";
        cin >> base >> height;
```

```cpp
        if (base <= 0 || height <= 0) {
            cout << "Invalid input. Base and height must be positive." << endl;
            return 0;
        }
        double area = calculateArea(base, height);
        cout << "Shape: Triangle" << endl;
        cout << "Base: " << base << endl;
        cout << "Height: " << height << endl;
        cout << "Area: " << area << endl;

    } else {
        cout << "Invalid choice. Please select a valid shape type." << endl;
    }

    return 0;

}
```

Output:

```
Choose a shape (1 for Circle, 2 for Rectangle, 3
    for Triangle): 3
Enter base and height of the triangle: 67 22
Shape: Triangle
Base: 67
Height: 22
Area: 737


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