C++ LAB PROGRAMS :

Write a C++ program to find the area and circumference of a circle

```
#include <iostream>
#include <cmath>

using namespace std;

const double PI = 3.14159; // Value of PI

int main() {
   double radius;

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

   // Calculate area and circumference
   double area = PI * pow(radius, 2);
   double circumference = 2 * PI * radius;

   // Display the results
   cout << "Area of the circle: " << area << endl;
   cout << "Circumference of the circle: " << circumference << endl;

   return 0;
}
```

Copy and paste this code into a `.cpp` file, compile and run it. The program will prompt you to enter the radius of the circle, and then it will calculate and display the area and circumference based on that radius.

Write and execute a C++ Program to display names, roll no's, and grades of 3 students who have appeared in the examination. Create a class with data members as Name, Roll no and Marks for 3 subjects. Write a method to calculate the grade of a student

```
#include <iostream>
#include <string>

using namespace std;

class Student {
private:
    string name;
    int rollNo;
    double marks[3];

public:
    // Constructor
    Student(string n, int roll) : name(n), rollNo(roll) {
        for (int i = 0; i < 3; i++) {
            marks[i] = 0.0;
        }
    }

    // Method to set marks for each subject
    void setMarks(double m1, double m2, double m3) {
        marks[0] = m1;
        marks[1] = m2;
        marks[2] = m3;
    }

    // Method to calculate the average marks
    double calculateAverage() {
```

```
        return (marks[0] + marks[1] + marks[2]) / 3;
    }

    // Method to calculate the grade
    char calculateGrade() {
        double average = calculateAverage();
        if (average >= 90) {
            return 'A';
        } else if (average >= 80) {
            return 'B';
        } else if (average >= 70) {
            return 'C';
        } else if (average >= 60) {
            return 'D';
        } else {
            return 'F';
        }
    }

    // Method to display student information
    void displayInfo() {
        cout << "Name: " << name << endl;
        cout << "Roll No: " << rollNo << endl;
        cout << "Grade: " << calculateGrade() << endl;
        cout << endl;
    }
};

int main() {
    Student students[3] = {
        Student("Alice", 101),
```

```
    Student("Bob", 102),
    Student("Charlie", 103)
};

// Set marks for each student
students[0].setMarks(85, 92, 78);
students[1].setMarks(76, 88, 94);
students[2].setMarks(65, 70, 58);

// Display student information
for (int i = 0; i < 3; i++) {
    students[i].displayInfo();
}

return 0;
}
```

Copy and paste this code into a `.cpp` file, compile and run it. The program defines a `Student` class with methods to set marks, calculate averages, calculate grades, and display student information. It creates three instances of the `Student` class, sets their marks, calculates their grades, and displays their information.

Write a C++ program to find sum of all the elements, maximum and minimum element in an array

```
#include <iostream>

using namespace std;

int main() {
    const int size = 5; // You can change this value to match the size of your array
    int arr[size];
```

```
    cout << "Enter " << size << " integers:" << endl;
    for (int i = 0; i < size; i++) {
        cin >> arr[i];
    }

    int sum = 0;
    int maxElement = arr[0];
    int minElement = arr[0];

    for (int i = 0; i < size; i++) {
        sum += arr[i];

        if (arr[i] > maxElement) {
            maxElement = arr[i];
        }

        if (arr[i] < minElement) {
            minElement = arr[i];
        }
    }

    cout << "Sum of all elements: " << sum << endl;
    cout << "Maximum element: " << maxElement << endl;
    cout << "Minimum element: " << minElement << endl;

    return 0;
}
```

In this program, you first input integers into an array. Then, you iterate through the array to calculate the sum, find the maximum element, and find the minimum element. Finally, the program displays the calculated sum, maximum element, and minimum element. Just replace the `size` constant with the size of your desired array.

Write a C++ program to demonstrate multilevel inheritance for the following: Suppose we have three classes Vehicle, FourWheeler, and Car. The class Vehicle is the base class, the class FourWheeler is derived from it and the class Car is derived from the class FourWheeler. Class Vehicle has a method 'vehicle' that prints 'I am a vehicle', class FourWheeler has a method 'fourWheeler' that prints 'I have four wheels', and class Car has a method 'car' that prints 'I am a car'. So, as this is a multi-level inheritance; we can have access to all the other classes methods from the object of the class Car. We invoke all the methods from a Car object and print the corresponding outputs of the methods. So, if we invoke the methods in this order, car(), fourWheeler(), and vehicle(), then the output will be I am a car I have four wheels I am a vehicle

```
#include <iostream>

using namespace std;

class Vehicle {
public:
   void vehicle() {
      cout << "I am a vehicle" << endl;
   }
};

class FourWheeler : public Vehicle {
public:
   void fourWheeler() {
      cout << "I have four wheels" << endl;
   }
};

class Car : public FourWheeler {
public:
   void car() {
      cout << "I am a car" << endl;
   }
```

```
};

int main() {
    Car myCar;

    // Access methods from Car class
    myCar.car();
    myCar.fourWheeler();
    myCar.vehicle();

    return 0;
}
```

In this program, we define the `Vehicle` class with a method `vehicle`, the `FourWheeler` class derived from `Vehicle` with a method `fourWheeler`, and the `Car` class derived from `FourWheeler` with a method `car`.

In the `main` function, we create an instance of the `Car` class and then call the methods `car`, `fourWheeler`, and `vehicle` in order, which demonstrates the multilevel inheritance and the access to all the methods in the inheritance chain. The expected output will be:

```
I am a car
I have four wheels
I am a vehicle
```

Write a C++ program to search an element in an array using linear search

```
#include <iostream>

using namespace std;

int linearSearch(int arr[], int size, int target) {
```

```
    for (int i = 0; i < size; i++) {
        if (arr[i] == target) {
            return i; // Return the index where the element is found
        }
    }
    return -1; // Return -1 if the element is not found
}

int main() {
    const int size = 7; // You can change this value to match the size of your array
    int arr[size] = {12, 45, 67, 23, 9, 56, 31};

    int target;
    cout << "Enter the element you want to search: ";
    cin >> target;

    int index = linearSearch(arr, size, target);

    if (index != -1) {
        cout << "Element " << target << " found at index " << index << endl;
    } else {
        cout << "Element " << target << " not found in the array." << endl;
    }

    return 0;
}
```

In this program, the `linearSearch` function takes an array, its size, and the target element to search for. It iterates through the array and checks if the current element matches the target. If found, it returns the index where the element is located. If not found, it returns -1.

In the `main` function, you input the element you want to search for and then call the `linearSearch` function. The program will either display the index where the element is found or a message indicating that the element was not found.

Write a program that creates a Calculator class. The class contains two variables of integer type. Design a constructor that accepts two values as parameter and set those values

```
#include <iostream>

class Calculator {
private:
    int value1;
    int value2;

public:
    Calculator(int val1, int val2) : value1(val1), value2(val2) {
        // Constructor sets the values when an object is created
    }

    int getValue1() {
        return value1;
    }

    int getValue2() {
        return value2;
    }
};

int main() {
    int num1, num2;

    std::cout << "Enter two integers: ";
    std::cin >> num1 >> num2;
```

```
    Calculator calc(num1, num2);

    std::cout << "Value 1: " << calc.getValue1() << std::endl;
    std::cout << "Value 2: " << calc.getValue2() << std::endl;

    return 0;
}
```

In this program, the `Calculator` class has two private integer variables `value1` and `value2`. The constructor of the class takes two integer parameters and initializes these variables with the given values.

In the `main` function, you input two integers from the user and then create an instance of the `Calculator` class using the provided values. Finally, the program displays the values of `value1` and `value2` using the member functions `getValue1()` and `getValue2()` of the `Calculator` class.

Design four methods named Add (), Subtract (), multiply (), Division ( ) for performing addition, subtraction, multiplication and division of two numbers.

```
#include <iostream>

class Calculator {
public:
    int Add(int num1, int num2) {
        return num1 + num2;
    }

    int Subtract(int num1, int num2) {
        return num1 - num2;
    }

    int Multiply(int num1, int num2) {
        return num1 * num2;
```

```
    }

    double Divide(int num1, int num2) {
        if (num2 != 0) {
            return static_cast<double>(num1) / num2;
        } else {
            std::cout << "Error: Division by zero." << std::endl;
            return 0.0;
        }
    }
};

int main() {
    Calculator calc;

    int num1, num2;
    std::cout << "Enter two numbers: ";
    std::cin >> num1 >> num2;

    std::cout << "Sum: " << calc.Add(num1, num2) << std::endl;
    std::cout << "Difference: " << calc.Subtract(num1, num2) << std::endl;
    std::cout << "Product: " << calc.Multiply(num1, num2) << std::endl;
    std::cout << "Quotient: " << calc.Divide(num1, num2) << std::endl;

    return 0;
}
```

In this program, the `Calculator` class contains methods `Add`, `Subtract`, `Multiply`, and `Divide` that perform the corresponding arithmetic operations on two numbers. The `Divide` method also checks for division by zero to prevent errors.

In the `main` function, you input two numbers from the user and then create an instance of the `Calculator` class. You then use the class methods to perform the desired calculations and display the results.

For addition and subtraction, two numbers should be positive. If any negative number is entered then throw an exception in respective methods. So design an exception handler (ArithmeticException) in Add () and Subtract () methods respectively to check whether any number is negative or not.

```
#include <iostream>
#include <stdexcept>

class ArithmeticException : public std::exception {
public:
    const char* what() const throw() {
        return "Negative number not allowed.";
    }
};

class Calculator {
public:
    int Add(int num1, int num2) {
        if (num1 < 0 || num2 < 0) {
            throw ArithmeticException();
        }
        return num1 + num2;
    }

    int Subtract(int num1, int num2) {
        if (num1 < 0 || num2 < 0) {
            throw ArithmeticException();
        }
        return num1 - num2;
    }
```

```
    int Multiply(int num1, int num2) {
        return num1 * num2;
    }

    double Divide(int num1, int num2) {
        if (num2 != 0) {
            return static_cast<double>(num1) / num2;
        } else {
            std::cout << "Error: Division by zero." << std::endl;
            return 0.0;
        }
    }
};

int main() {
    Calculator calc;

    int num1, num2;
    std::cout << "Enter two numbers: ";
    std::cin >> num1 >> num2;

    try {
        std::cout << "Sum: " << calc.Add(num1, num2) << std::endl;
    } catch (ArithmeticException& e) {
        std::cout << "Exception: " << e.what() << std::endl;
    }

    try {
        std::cout << "Difference: " << calc.Subtract(num1, num2) << std::endl;
    } catch (ArithmeticException& e) {
```

```
        std::cout << "Exception: " << e.what() << std::endl;
    }

    std::cout << "Product: " << calc.Multiply(num1, num2) << std::endl;
    std::cout << "Quotient: " << calc.Divide(num1, num2) << std::endl;

    return 0;
}
```

In this program, I've defined a custom exception class `ArithmeticException` that inherits from `std::exception`. The `Add()` and `Subtract()` methods now check if the input numbers are negative, and if so, they throw an instance of the `ArithmeticException`. In the `main()` function, I've wrapped the calls to these methods in try-catch blocks to catch and handle the exceptions.

For division and multiplication two numbers should not be zero. If zero is entered for any number then throw an exception in respective methods. So design an exception handler (ArithmeticException) in multiply () and Division () methods respectively to check whether any number is zero or not.

```
#include <iostream>
#include <stdexcept>

class ArithmeticException : public std::exception {
public:
    const char* what() const throw() {
        return "ArithmeticException: Operation not allowed.";
    }
};

class Calculator {
public:
    int Add(int num1, int num2) {
        if (num1 < 0 || num2 < 0) {
            throw ArithmeticException();
```

```
        }
        return num1 + num2;
    }

    int Subtract(int num1, int num2) {
        if (num1 < 0 || num2 < 0) {
            throw ArithmeticException();
        }
        return num1 - num2;
    }

    int Multiply(int num1, int num2) {
        if (num1 == 0 || num2 == 0) {
            throw ArithmeticException();
        }
        return num1 * num2;
    }

    double Divide(int num1, int num2) {
        if (num2 == 0) {
            throw ArithmeticException();
        }
        return static_cast<double>(num1) / num2;
    }
};

int main() {
    Calculator calc;

    int num1, num2;
    std::cout << "Enter two numbers: ";
```

```
    std::cin >> num1 >> num2;

    try {
        std::cout << "Sum: " << calc.Add(num1, num2) << std::endl;
    } catch (ArithmeticException& e) {
        std::cout << "Exception: " << e.what() << std::endl;
    }

    try {
        std::cout << "Difference: " << calc.Subtract(num1, num2) << std::endl;
    } catch (ArithmeticException& e) {
        std::cout << "Exception: " << e.what() << std::endl;
    }

    try {
        std::cout << "Product: " << calc.Multiply(num1, num2) << std::endl;
    } catch (ArithmeticException& e) {
        std::cout << "Exception: " << e.what() << std::endl;
    }

    try {
        std::cout << "Quotient: " << calc.Divide(num1, num2) << std::endl;
    } catch (ArithmeticException& e) {
        std::cout << "Exception: " << e.what() << std::endl;
    }

    return 0;
}
```

In this version, I've added checks in the `Multiply()` and `Divide()` methods to throw an instance of the `ArithmeticException` if one of the numbers is zero (for multiplication) or if the second number is zero (for division). The `main()` function now handles these exceptions using try-catch blocks.

Write a C++ program to overload function for computing the area triangle, circle and square

```
#include <iostream>
#include <cmath>

class AreaCalculator {
public:
    double calculateArea(double base, double height) {
        return 0.5 * base * height; // Area of a triangle
    }

    double calculateArea(double radius) {
        return 3.14159 * radius * radius; // Area of a circle
    }

    double calculateArea(int side) {
        return side * side; // Area of a square
    }
};

int main() {
    AreaCalculator calculator;

    double base, height, radius;
    int side;

    std::cout << "Enter the base and height of the triangle: ";
    std::cin >> base >> height;
    std::cout << "Area of the triangle: " << calculator.calculateArea(base, height) << std::endl;

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

```
    std::cin >> radius;
    std::cout << "Area of the circle: " << calculator.calculateArea(radius) << std::endl;

    std::cout << "Enter the side of the square: ";
    std::cin >> side;
    std::cout << "Area of the square: " << calculator.calculateArea(side) << std::endl;

    return 0;
}
```

In this program, the `AreaCalculator` class contains three overloaded functions named `calculateArea`:

1. `calculateArea(double base, double height)` calculates the area of a triangle.
2. `calculateArea(double radius)` calculates the area of a circle.
3. `calculateArea(int side)` calculates the area of a square.

In the `main()` function, you input the required dimensions (base and height for the triangle, radius for the circle, and side for the square) and then call the appropriate `calculateArea` function based on the shape. The program then displays the calculated area for each shape.

Create a C++ class that includes constructors to do the following. • Create an uninitialized string. • Initialize an object with a string constant at the time of creation. • Create an object and initialize with another object. Also write a function to concatenate two strings

```
#include <iostream>
#include <string>

class StringManipulator {
private:
    std::string str;

public:
    // Constructor to create an uninitialized string
    StringManipulator() {}
```

```
    // Constructor to initialize with a string constant
    StringManipulator(const std::string& s) : str(s) {}

    // Constructor to initialize with another object
    StringManipulator(const StringManipulator& other) : str(other.str) {}

    // Function to concatenate two strings
    std::string concatenate(const std::string& other) {
        return str + other;
    }

    // Function to display the string
    void display() {
        std::cout << "String: " << str << std::endl;
    }
};

int main() {
    StringManipulator uninitializedStr;
    StringManipulator strWithConstant("Hello, ");
    StringManipulator strWithObject(strWithConstant);

    uninitializedStr.display();
    strWithConstant.display();
    strWithObject.display();

    std::string otherString = "World!";
    std::cout << "Concatenated: " << strWithObject.concatenate(otherString) << std::endl;

    return 0;
```

}

In this program, the `StringManipulator` class has three constructors:

1. A default constructor to create an uninitialized string.
2. A constructor that initializes the object with a string constant.
3. A constructor that initializes the object with another `StringManipulator` object.

The class also includes a `concatenate` function that takes another string as input and concatenates it with the object's string. The `display` function is used to display the stored string.

In the `main()` function, instances of the `StringManipulator` class are created using different constructors, and the `display` function is used to show the contents of the strings. Additionally, the `concatenate` function is demonstrated by concatenating another string with an object's string.

Write a program that creates a Calculator class. The class contains two variables of integer type. Design a constructor that accepts two values as parameter and set those values. • Design four methods named Add (), Subtract (), multiply (), Division ( ) for performing addition, subtraction, multiplication and division of two numbers. • For addition and subtraction, two numbers should be positive. If any negative number is entered then throw an exception in respective methods. So design an exception handler (ArithmeticException) in Add () and Subtract () methods respectively to check whether any number is negative or not. • For division and multiplication two numbers should not be zero. If zero is entered for any number then throw an exception in respective methods. So design an exception handler (ArithmeticException) in multiply () and Division () methods respectively to check whether any number is zero or not.

```
#include <iostream>
#include <stdexcept>

class ArithmeticException : public std::exception {
public:
    const char* what() const throw() {
        return "ArithmeticException: Operation not allowed.";
    }
};

class Calculator {
```

```
private:
    int num1, num2;

public:
    Calculator(int val1, int val2) : num1(val1), num2(val2) {}

    int Add() {
        if (num1 < 0 || num2 < 0) {
            throw ArithmeticException();
        }
        return num1 + num2;
    }

    int Subtract() {
        if (num1 < 0 || num2 < 0) {
            throw ArithmeticException();
        }
        return num1 - num2;
    }

    int Multiply() {
        if (num1 == 0 || num2 == 0) {
            throw ArithmeticException();
        }
        return num1 * num2;
    }

    double Divide() {
        if (num2 == 0) {
            throw ArithmeticException();
        }
```

```
        return static_cast<double>(num1) / num2;
    }
};

int main() {
    int num1, num2;
    std::cout << "Enter two numbers: ";
    std::cin >> num1 >> num2;

    Calculator calc(num1, num2);

    try {
        std::cout << "Sum: " << calc.Add() << std::endl;
    } catch (ArithmeticException& e) {
        std::cout << "Exception: " << e.what() << std::endl;
    }

    try {
        std::cout << "Difference: " << calc.Subtract() << std::endl;
    } catch (ArithmeticException& e) {
        std::cout << "Exception: " << e.what() << std::endl;
    }

    try {
        std::cout << "Product: " << calc.Multiply() << std::endl;
    } catch (ArithmeticException& e) {
        std::cout << "Exception: " << e.what() << std::endl;
    }

    try {
        std::cout << "Quotient: " << calc.Divide() << std::endl;
```

```
    } catch (ArithmeticException& e) {
        std::cout << "Exception: " << e.what() << std::endl;
    }

    return 0;
}
```

This program defines a `Calculator` class with constructors and methods for addition, subtraction, multiplication, and division. Exception handling has been added to each method according to the specified conditions. The `main()` function takes user input, creates an instance of the `Calculator` class, and demonstrates each operation while handling potential exceptions.

---

b) Write a C++ program to find the sum of all the natural numbers from 1 to n.

```
#include <iostream>

int main() {
    int n;
    std::cout << "Enter a positive integer n: ";
    std::cin >> n;

    if (n < 1) {
        std::cout << "Please enter a positive integer." << std::endl;
        return 1; // Exit with an error code
    }

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

    std::cout << "Sum of natural numbers from 1 to " << n << " is: " << sum << std::endl;
```

```
    return 0;
}
```

In this program, the user is prompted to enter a positive integer 'n'. If 'n' is not positive, the program displays an error message and exits. Otherwise, it calculates the sum of natural numbers from 1 to 'n' using a loop and displays the result.

Write a C++ program to find the simple interest

```
#include <iostream>

int main() {
    double principal, rate, time;

    std::cout << "Enter the principal amount: ";
    std::cin >> principal;

    std::cout << "Enter the rate of interest (in percentage): ";
    std::cin >> rate;

    std::cout << "Enter the time (in years): ";
    std::cin >> time;

    // Calculate simple interest
    double interest = (principal * rate * time) / 100;

    std::cout << "Simple Interest: " << interest << std::endl;

    return 0;
}
```

In this program, the user is prompted to input the principal amount, rate of interest (in percentage), and time (in years). The simple interest is then calculated using the formula `(principal * rate * time) / 100` and displayed.

Write a C++ program to implement the following inheritance. Person Student Teacher Marks Assume suitable data members and member functions for all the classes. • Display the number of publications for a teacher and percentage marks for a student

```
#include <iostream>
#include <string>

class Person {
protected:
    std::string name;

public:
    Person(const std::string& n) : name(n) {}

    void displayInfo() {
        std::cout << "Name: " << name << std::endl;
    }
};

class Marks {
protected:
    double percentage;

public:
    Marks(double p) : percentage(p) {}

    void displayMarks() {
        std::cout << "Percentage Marks: " << percentage << "%" << std::endl;
    }
};
```

```
class Student : public Person, public Marks {
public:
    Student(const std::string& n, double p) : Person(n), Marks(p) {}

    void displayStudentInfo() {
        displayInfo();
        displayMarks();
    }
};

class Teacher : public Person {
private:
    int publications;

public:
    Teacher(const std::string& n, int pubs) : Person(n), publications(pubs) {}

    void displayTeacherInfo() {
        displayInfo();
        std::cout << "Number of Publications: " << publications << std::endl;
    }
};

int main() {
    Student student("Alice", 85.5);
    Teacher teacher("Mr. Smith", 10);

    std::cout << "Student Information:" << std::endl;
    student.displayStudentInfo();

    std::cout << "\nTeacher Information:" << std::endl;
```

```
    teacher.displayTeacherInfo();


    return 0;
}
```

n this program, there are four classes:

1. `Person` class with a protected data member `name` and a member function `displayInfo()` to display the name.
2. `Marks` class with a protected data member `percentage` and a member function `displayMarks()` to display the percentage marks.
3. `Student` class which is derived from both `Person` and `Marks` classes. It has a constructor to initialize name and percentage, and a member function `displayStudentInfo()` to display student's name and percentage marks.
4. `Teacher` class which is derived from `Person` class. It has a constructor to initialize name and number of publications, and a member function `displayTeacherInfo()` to display teacher's name and number of publications.

The `main()` function creates instances of `Student` and `Teacher` classes and demonstrates the functionality by calling the respective member functions.

Write a C++ program to read the contents from a text file, count and display the number of alphabets present in it. b) Write a C++ program to find the factorial of a number

```
#include <iostream>
#include <fstream>
#include <cctype>


int main() {
    std::ifstream inputFile("input.txt");
    if (!inputFile.is_open()) {
        std::cout << "Could not open the file." << std::endl;
        return 1;
    }


    char ch;
    int alphabetCount = 0;
```

```
    while (inputFile.get(ch)) {
        if (isalpha(ch)) {
            alphabetCount++;
        }
    }

    inputFile.close();

    std::cout << "Number of alphabets in the file: " << alphabetCount << std::endl;

    return 0;
}
```

In this program, the content of the file named "input.txt" is read character by character. The `isalpha()` function from the `<cctype>` header is used to check if a character is an alphabet. The count of alphabets is maintained, and the result is displayed.

---

b) Find Factorial of a Number:

```
#include <iostream>

unsigned long long factorial(int n) {
    if (n == 0 || n == 1) {
        return 1;
    } else {
        return n * factorial(n - 1);
    }
}

int main() {
    int num;
```

```
    std::cout << "Enter a positive integer: ";
    std::cin >> num;

    if (num < 0) {
        std::cout << "Please enter a positive integer." << std::endl;
        return 1;
    }

    unsigned long long result = factorial(num);
    std::cout << "Factorial of " << num << " is: " << result << std::endl;

    return 0;
}
```

In this program, the `factorial()` function recursively calculates the factorial of a number. The user inputs a positive integer, and the program displays the factorial of that number.

Write a C++ program to create a text file, check file created or not, if created it will write some text into the file and then read the text from the file.

```
#include <iostream>
#include <fstream>

int main() {
    std::ofstream outputFile("output.txt");
    if (!outputFile.is_open()) {
        std::cout << "Could not create/open the file." << std::endl;
        return 1;
    }

    outputFile << "Hello, this is some text written to the file." << std::endl;
    outputFile.close();

    std::ifstream inputFile("output.txt");
```

```
    if (!inputFile.is_open()) {
        std::cout << "Could not open the file." << std::endl;
        return 1;
    }

    std::string content;
    while (std::getline(inputFile, content)) {
        std::cout << content << std::endl;
    }

    inputFile.close();

    return 0;
}
```

In this program, it first creates a file named "output.txt" and writes a line of text into it. Then, it reads the content of the file and displays it.

---

b) Check Prime Number using a Function:

```
#include <iostream>

bool isPrime(int num) {
    if (num <= 1) {
        return false;
    }

    for (int i = 2; i * i <= num; i++) {
        if (num % i == 0) {
            return false;
        }
    }
```

```
    return true;
}

int main() {
    int num;
    std::cout << "Enter a positive integer: ";
    std::cin >> num;

    if (isPrime(num)) {
        std::cout << num << " is a prime number." << std::endl;
    } else {
        std::cout << num << " is not a prime number." << std::endl;
    }

    return 0;
}
```

In this program, the `isPrime()` function checks whether a given number is prime or not. The `main()` function takes an input number and uses the `isPrime()` function to determine if it's prime or not, then displays the result.
