**UNIT-1**

**Experiment-1**

WEEK-1

1.**Aim**: **To find the roots of a quadratic equation using C++.**

**Description**: Write a C++ program to find the roots of a quadratic equation of the form

ax2+bx+c=0ax^2 + bx + c = 0ax2+bx+c=0

**The program should:**

1. Accept values for coefficients a, b, and c.
2. Calculate the discriminant D=b2−4acD = b^2 - 4acD=b2−4ac.
3. Determine the nature of the roots based on the discriminant:
   * **D > 0** → Two distinct real roots.
   * **D = 0** → Two equal real roots.
   * **D < 0** → Two complex roots.
4. Display the roots accordingly.

Program: #include <iostream>

#include <cmath> // for sqrt() function

using namespace std;

int main()

{

float a, b, c;

cout << "Enter coefficients a, b and c: ";

cin >> a >> b >> c;

float discriminant = b\*b - 4\*a\*c;

float root1, root2;

if (a == 0) {

cout << "This is not a quadratic equation (a cannot be 0)." << endl;}

else if (discriminant > 0) {

// Real and distinct roots

root1 = (-b + sqrt(discriminant)) / (2\*a);

root2 = (-b - sqrt(discriminant)) / (2\*a);

cout << "Roots are real and distinct." << endl;

cout << "Root 1 = " << root1 << endl;

cout << "Root 2 = " << root2 << endl;

}else if (discriminant == 0) {

// Real and equal roots

root1 = root2 = -b / (2\*a);

cout << "Roots are real and equal." << endl;

cout << "Root = " << root1 << endl;

}else {

// Complex roots

float realPart = -b / (2\*a);

float imagPart = sqrt(-discriminant) / (2\*a);

cout << "Roots are complex and imaginary." << endl;

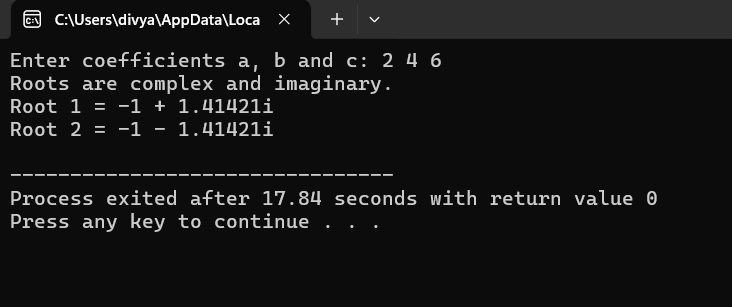
cout << "Root 1 = " << realPart << " + " << imagPart << "i" << endl;

cout << "Root 2 = " << realPart << " - " << imagPart << "i" << endl;

} return 0;

}

Output:



**Experiment-2**

2.**Aim**: **Find factorial of a given number using recursion.**

**Description**: Write a C++ program to calculate the factorial of a given positive integer using recursion.  
**The program should**:

1. Accept an integer n from the user.
2. Use a recursive function factorial(int n) to compute:

n!=n×(n−1)×(n−2)×⋯×1n! = n \times (n-1) \times (n-2) \times \dots \times 1n!=n×(n−1)×(n−2)×⋯×1

with the base case:

factorial(0)=1factorial(0) = 1factorial(0)=1

1. Display the factorial result for the given number.

**Program**: #include <iostream>

using namespace std;

long factorial(int n)

{

if (n == 0 || n==1)

return 1;

else

return n \* factorial(n - 1);

}

int main()

{

int number;

cout << "Enter a positive integer: ";

cin >> number;

if (number < 0) {

cout << "Factorial is not defined for negative numbers." << endl;

} else

{

long result = factorial(number);

cout << "Factorial of " << number << " = " << result << endl;

}

return 0;

}

Output:

A screenshot of a computer

AI-generated content may be incorrect.

**Experiment-3**

3.**Aim**: **Implement scope resolution and namespaces.**

**Description**: Write a C++ program to demonstrate the use of **scope resolution operator (::)** and **namespaces**.  
**The program should**:

1. Show how the **scope resolution operator** can be used to:
   * Access a global variable when a local variable has the same name.
   * Define a function outside its class.
2. Create one or more **namespaces** containing variables or functions.
3. Demonstrate accessing namespace members using:
   * The namespace name with the scope resolution operator (namespace\_name::member).
   * The using directive to simplify access.
4. Display outputs to clearly indicate how scope resolution and namespaces work.

Program: #include <iostream>

using namespace std; // using standard namespace

int x = 100; // Global variable

namespace MyNamespace

{

int value = 50;

void show()

{

cout << "Inside MyNamespace, value = " << value << endl;

}

}

int main()

{ int x = 10; // Local variable

cout << "Local x = " << x << endl;

cout << "Global x = " << ::x << endl; // Using scope resolution to access global x

cout << "Namespace value = " << MyNamespace::value << endl;

MyNamespace::show();

return 0;

}

Output: A screenshot of a computer

AI-generated content may be incorrect.

**Experiment-4**

4.Aim: **Illustrate the use of default arguments and access specifiers**

Description: Write a C++ program to demonstrate the use of **default arguments** in functions and **access specifiers** in classes.  
The program should:

1. Create a function with one or more **default arguments**, allowing it to be called with fewer parameters.
2. Create a class containing data members and member functions, using different **access specifiers**:
   * public
   * private
   * protected
3. Show how access specifiers control the visibility and accessibility of class members.
4. Display outputs that clearly illustrate the effect of default arguments and different access levels.

Program: #include <iostream>

using namespace std;

class Student

{

private: // it is the default access specifier

int sno; // Private: accessible only inside the class

protected:

string sgrade; // Protected: accessible in derived classes

public:

string sname; // Public: accessible from outside

// Constructor

Student(int sn, string sna, string sg)

{

sno = sn;

sname = sna;

sgrade = sg;

}

// Function with default arguments

void disp(string sgrade = "Pass")

{

cout << " Name: " << sname << endl;

cout << "Roll Number: " << sno << endl;

cout << "Grade: " << sgrade << endl;

}

};

int main()

{

Student s1(100, "Rajesh", "A+"); // Creating object

cout << "Name of the Student ="<< s1.sname << endl; // accessing public variable

s1.disp();

s1.disp("Distinction");

s1.disp();

return 0;

}

Output:

A screenshot of a computer program

AI-generated content may be incorrect.