GIRIJANANDA CHOWDHURY UNIVERSITY, GUWAHATI(ASSAM)

Programming using C++

LAB MANUAL

COURSE CODE: BCS23102P

B.TECH 2ND SEMESTER

Prepared by:-

MRIDUSMITA BARUAH
Laboratory Instructor

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEEING
GUWAHATI-781017

GIRIJANANDA CHOWDHURY UNIVERSITY, GUWAHATI(ASSAM)

PCC	PROGRAMMING USING C++	L	T	P	C
		1	0	2	2

Prerequisite: Mathematics-1, Programming for Problem Solving

Course Objectives: The objectives of this course are to:

1. To learn the fundamentals of C++ programming.
2. To learn the syntax and semantics of C++ programming language.
3. To obtain approximate solutions to common numerical methods.
4. To derive numerical methods for various mathematical operations and tasks.
5. To derive numerical methods for solutions of non-linear, algebraic and ordinary differential equations.
6. To analyze and evaluate the accuracy of common numerical methods. 7.

Course Outcome: After successful completion of this course, the students will be able to

- CO1: Describe the C++ concepts of streams, classes, functions, data and objects.
- CO2: Implement the use of OOPs concepts using C++ programs.
- CO3: Build programs for numerical solution of non-linear and algebraic equations.
- CO4: Build programs for numerical solution of ordinary differential equations.

Module 1: Introduction to C++	**6 hours**
Basic concepts of OOPs, structure of C++ program, tokens (identifiers, keywords, constants, operators, special characters), data types (basic, derived, user defined), console I/O statements	
Module 2: C++ Programming	**8 hours**
Defining a class, creating objects, accessing data members using objects, calling member functions using objects, scope resolution operator, access specifiers (private, public, protected)	
Module 3: Numerical solution of non-linear and algebraic equations	**8 hours**
Method of successive bisection, Regula-Falsi& Newton-Raphson method, Gauss-elimination method, Gauss-Seidel method	
Module 4: Numerical Solution of Ordinary Differential Systems	**8 hours**
Euler's method, Runge-Kutta – 4th order method, predictor-corrector method, two ordinary differential equation using numerical integration, Trapezoidal Rule of Numerical integration, elliptic boundary value problem using method of finite differences.	
Total hours	**30 hours**

Text Book(s)

1. Object Oriented Programming using C++by E. Balaguruswamy
2. Object Oriented Programming and C++ by Rajaraman, New Age International.
3. Numerical method: E. Balaguruswamy T.M.H

4.	Numerical Methods, Sukhendu Dey, Shishir Gupta, McGraw Hill Education (India) private Limited
5.	Numerical & Statistical Methods With Programming in c by Sujatha Sinha
Reference Books	
1.	Herbert Schildt, C++ The Complete Reference, Fourth Edition, Tata McGraw HillPublication.
2.	Joyce Farrell, Object-oriented programming using C++, Fourth Edition, Cengage Learning.
3.	Numerical Methods by B.S. Grewal
4.	Computer Programming & Numerical Analysis by N Dutta, University Press
5.	Numerical Algorithms. E. V. Krishnamurthy, S. K. Sen. Affilated East-West Press

GIRIJANANDA CHOWDHURY UNIVERSITY, GUWAHATI(ASSAM)

LIST OF EXPERIMENTS:

1. Write a program in c++ to perform addition of two numbers.
2. Write a program in c++ to perform swapping of two numbers.
3. Write a program in c++ to perform addition of two numbers using class and object.
4. Write a program in c++ to perform swapping of two numbers using class and object.
5. Write a program to find the roots of a quadratic equation in c++.
6. Write a program in c++ to calculate the relative error and absolute error.
7. Write a program in c++ to calculate the relative error and absolute error where exactValue = 3.14159 and approxValue = 3.14.
8. AIM: Write a program in c++ to calculate the Truncation error where integral = 0.0;deltaX = 0.01;
9. Write a program in c++ to implement Method of successive bisection.
10. Write a program in c++ to implement Regula-Falsi Method.
11. Write a program in c++ to implement Newton-Raphson Method.
12. Write a program in c++ to implement Gauss-elimination method.
13. Write a program in c++ to implement Gauss-Seidel method.
14. Write a program in c++ to implement Euler's method.
15. Write a program in c++ to implement Runge-Kutta – 4th order method.
16. Write a program in c++ to implement predictor-corrector method.
17. Write a program in c++ to implement two ordinary differential equation using numerical integration.
18. Write a program in c++ to implement Trapezoidal Rule of Numerical integration.
19. Write a program in c++ to implement elliptic boundary value problem using method of finite differences.

GIRIJANANDA CHOWDHURY UNIVERSITY, GUWAHATI(ASSAM)

AIM: Write a program in c++ to perform addition of two numbers

PROGRAM:

```
#include <iostream>
using namespace std;
int main()
{
   int num1, num2, sum;
   cout << "Enter first number: ";
   cin >> num1;
   cout << "Enter second number: ";
   cin >> num2;
   sum = num1 + num2;
   cout << "The sum of " << num1 << " and " << num2 << " is: " << sum << endl;
   return 0;
}
```

OUTPUT:

```
Enter first number: 5
Enter second number: 5
The sum of 5 and 5 is: 10
```

AIM: Write a program in c++ to perform swapping of two numbers.

PROGRAM:

```
#include <iostream>
using namespace std;
int main()
{
   int a, b, temp;
   cout << "Enter first number (a): ";
   cin >> a;
   cout << "Enter second number (b): ";
   cin >> b;
   cout << "\nBefore swapping:\n";
   cout << "a = " << a << "\nb = " << b << endl;
   temp = a;
   a = b;
   b = temp;
   cout << "\nAfter swapping:\n";
   cout << "a = " << a << "\nb = " << b << endl;
   return 0;
}
```

OUTPUT:

```
Enter first number (a): 5
Enter second number (b): 10
Before swapping:
a = 5
b = 10
After swapping:
a = 10
b = 5
```

GIRIJANANDA CHOWDHURY UNIVERSITY, GUWAHATI(ASSAM)

AIM: Write a program in c++ to perform addition of two numbers using class and object.

PROGRAM:

```
#include <iostream>
using namespace std;
class Addition
{
   private:
      int num1;
      int num2;
  public:
      void setData(int a, int b);
      int add();
};
void Addition::setData(int a, int b)
{
   num1 = a;
   num2 = b;
}
int Addition::add()
{
   return num1 + num2;
}
int main()
{
   Addition obj;
   int a, b;
   cout << "Enter first number: ";
   cin >> a;
   cout << "Enter second number: ";
   cin >> b;
   obj.setData(a, b);
   int sum = obj.add();
   cout << "The sum is: " << sum << endl;
   return 0;
}
```

OUTPUT:

Enter first number: 10
Enter second number: 20
The sum is: 30

GIRIJANANDA CHOWDHURY UNIVERSITY, GUWAHATI(ASSAM)

AIM: Write a program in c++ to perform swapping of two numbers using class and object.

PROGRAM:

```
#include <iostream>
using namespace std;
class SwapNumbers
{
   private:
   int num1, num2;
   public:
   void getInput()
  {
      cout << "Enter first number: ";
      cin >> num1;
      cout << "Enter second number: ";
      cin >> num2;
   }
   void display() {
      cout << "First number: " << num1 << ", Second number: " << num2 << endl;
   }
   void swap();
};
void SwapNumbers::swap()
{
   int temp = num1;
   num1 = num2;
   num2 = temp;
}
int main()
{
   SwapNumbers obj;
   obj.getInput();
   cout << "\nBefore swapping:" << endl;
   obj.display();
   obj.swap();
   cout << "\nAfter swapping:" << endl;
   obj.display();
   return 0;
}
```

OUTPUT:

```
Enter first number: 5
Enter second number: 10
Before swapping:
First number: 5, Second number: 10
After swapping:
First number: 10, Second number: 5
```

GIRIJANANDA CHOWDHURY UNIVERSITY, GUWAHATI(ASSAM)

AIM: Write a program to find the roots of a quadratic equation in c++.

PROGRAM:

```
#include <iostream>
#include <cmath>
using namespace std;
int main()
{
   float a, b, c;
   float discriminant, root1, root2, realPart, imagPart;
   cout << "Enter coefficients a, b and c: ";
   cin >> a >> b >> c;
   discriminant = b * b - 4 * a * c;
   if (discriminant > 0)
  {
      // Two real and distinct roots
      root1 = (-b + sqrt(discriminant)) / (2 * a);
      root2 = (-b - sqrt(discriminant)) / (2 * a);
      cout << "Roots are real and different." << endl;
      cout << "Root 1 = " << root1 << endl;
      cout << "Root 2 = " << root2 << endl;
   }
   else if (discriminant == 0)
   {
      // Two real and equal roots
      root1 = -b / (2 * a);
      cout << "Roots are real and equal." << endl;
      cout << "Root = " << root1 << endl;
   }
   else
   {
      // Complex roots
      realPart = -b / (2 * a);
      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:

Enter coefficients a, b and c: 1 -3 2
Roots are real and different.
Root 1 = 2
Root 2 = 1

GIRIJANANDA CHOWDHURY UNIVERSITY, GUWAHATI(ASSAM)

AIM: Write a program in c++ to calculate the relative error and absolute error.

PROGRAM:

```
#include <iostream>
#include <cmath>
using namespace std;
int main()
{
   double trueValue, measuredValue, absoluteError, relativeError;
   cout << "Enter the true value: ";
   cin >> trueValue;
   cout << "Enter the measured value: ";
   cin >> measuredValue;
   // Calculate absolute error
   absoluteError = fabs(measuredValue - trueValue);
  // Calculate relative error
   relativeError = absoluteError / fabs(trueValue);
   cout << "\nAbsolute Error = " << absoluteError << endl;
   cout << "Relative Error = " << relativeError << endl;
   return 0;
}
```

OUTPUT:

```
Enter the true value: 50
Enter the measured value: 48

Absolute Error = 2
Relative Error = 0.04
```

GIRIJANANDA CHOWDHURY UNIVERSITY, GUWAHATI(ASSAM)

AIM: Write a program in c++ to calculate the relative error and absolute error where exactValue = 3.14159 and approxValue = 3.14.

PROGRAM:

```
#include <iostream>
#include <cmath>
using namespace std;
int main()
{
   // Define two numbers for comparison
   double exactValue = 3.14159;
   double approxValue = 3.14;

   // Absolute Error
   double absError = fabs(exactValue - approxValue);
   cout << "Absolute Error: " << absError << endl;

   // Relative Error
   double relError = fabs((exactValue - approxValue) / exactValue);
   cout << "Relative Error: " << relError << endl;
   return 0;
}
```

OUTPUT:

```
Absolute Error: 0.00159
Relative Error: 0.000506605
```

GIRIJANANDA CHOWDHURY UNIVERSITY, GUWAHATI(ASSAM)

AIM: Write a program in c++ to calculate the Truncation error where integral = 0.0; deltaX = 0.01;

PROGRAM:

```
#include <iostream>
#include <cmath>
using namespace std;
int main()
{
   // Define two numbers for comparison
   double exactValue = 3.14159;
   double approxValue = 3.14;

 // Absolute error
   double absError = fabs(exactValue - approxValue);
   cout << "Absolute Error: " << absError << endl;

   // Relative error
   double relError = fabs((exactValue - approxValue) / exactValue);
   cout << "Relative Error: " << relError << endl;

   // Round-off error demonstration
   double a = 1.0, b = 1e-15, c = 1e15;
   double result1 = (a + b) - c;
   double result2 = a - c + b;
   cout << "Result 1: " << result1 << endl;
   cout << "Result 2: " << result2 << endl;
   double roundOffError = fabs(result1 - result2);
   cout << "Round-off Error: " << roundOffError << endl;

   // Truncation error (Numerical Integration of sin(x) from 0 to pi)
   double integral = 0.0;
   double deltaX = 0.01;
   int numIntervals = static_cast<int>(M_PI / deltaX); // Integrating from 0 to pi
   for (int i = 0; i < numIntervals; ++i)
  {
      double x = i * deltaX;
      integral += sin(x) * deltaX;
   }
```

```
    double exactIntegral = 2.0; // Exact integral of sin(x) from 0 to pi is 2
    double truncationError = fabs(exactIntegral - integral);
    cout << "Truncation Error: " << truncationError << endl;
    return 0;
}
```

OUTPUT:

```
Absolute Error: 0.00159
Relative Error: 0.000506588
Result 1: -1e+15
Result 2: -1e+15
Round-off Error: 0
Truncation Error: 0.000166083
```

AIM: Write a program in c++ to implement Method of successive bisection.

PROGRAM:

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

// Define the function whose root we want to find
double f(double x) {
   return x * x * x - x - 2; // Example: f(x) = x^3 - x - 2
}

// Bisection Method
void bisection(double a, double b, double tol, int max_iter) {
   if (f(a) * f(b) >= 0) {
      cout << "Bisection method fails. f(a) and f(b) should have opposite signs." << endl;
      return;
   }
   double c = a;
   int iter = 0;
   cout << "Iteration\t a\t\t b\t\t c\t\t f(c)" << endl;
   while ((b - a) >= tol && iter < max_iter)
   {
      // Find midpoint
      c = (a + b) / 2;
     // Output iteration details
      cout << iter + 1 << "\t\t " << a << "\t " << b << "\t " << c << "\t " << f(c) << endl;
    // Check if c is root
      if (f(c) == 0.0)
         break;
      else if (f(c) * f(a) < 0)
         b = c;
      else
         a = c;
      iter++;
   }
   cout << "\nApproximate root: " << c << endl;
   cout << "Function value at root: f(" << c << ") = " << f(c) << endl;
   cout << "Total iterations: " << iter << endl;
```

}

OUTPUT:

Iteration	a	b	c	f(c)
1	1	2	1.5	0.375
2	1.5	2	1.75	1.60938

...
Approximate root: 1.52138
Function value at root: f(1.52138) = ~0
Total iterations: 20

AIM: Write a program in c++ to implement Regula-Falsi Method.

PROGRAM:

```
#include <iostream>
#include <cmath>
using namespace std;
// Define the function for which we want to find the root
double f(double x)
{
   return x*x*x - x - 2; // Example: x^3 - x - 2
}
int main()
{
   double a, b, c, fa, fb, fc;
   double tolerance;
   int maxIterations, iteration = 0;
   cout << "Enter interval [a, b]: ";
   cin >> a >> b;
   cout << "Enter tolerance: ";
   cin >> tolerance;
   cout << "Enter maximum iterations: ";
   cin >> maxIterations;
   fa = f(a);
   fb = f(b);
   // Check if initial guesses bracket the root
   if (fa * fb >= 0)
   {
      cout << "Invalid interval. f(a) and f(b) must have opposite signs.\n";
      return 1;
   }
   cout << "\nIter\t a\t\t b\t\t c\t\t f(c)\n";
   while (iteration < maxIterations)
   {
      // Regula-Falsi formula
      c = (a * fb - b * fa) / (fb - fa);
      fc = f(c);
      cout << iteration + 1 << "\t" << a << "\t" << b << "\t" << c << "\t" << fc << "\n";
     // Check convergence
      if (fabs(fc) < tolerance)
```

```
    {
        cout << "\nRoot found: " << c << " after " << iteration + 1 << " iterations.\n";
        return 0;
     }
    // Update interval
     if (fa * fc < 0)
    {
        b = c;
        fb = fc;
     }
     else
    {
        a = c;
        fa = fc;
     }
     iteration++;
  }
  cout << "\nMethod did not converge after " << maxIterations << " iterations.\n";
  return 0;
}
```

OUTPUT:

```
Enter interval [a, b]: 1 2
Enter tolerance: 0.0001
Enter maximum iterations: 20
Iter    a         b         c         f(c)
1       1         2         1.33333    -0.962963
2       1.33333   2         1.46269    -0.333338
3       1.46269   2         1.50402    -0.054375
4       1.50402   2         1.51077    -0.008128
5       1.51077   2         1.51185    -0.001207

Root found: 1.51185 after 5 iterations.
```

AIM: Write a program in c++ to implement Newton-Raphson Method.

PROGRAM:

```
#include <iostream>
#include <cmath>
using namespace std;
// Define the function
double f(double x)
{
   return x*x*x - x - 2; // Example: x^3 - x - 2
}

// Define the derivative of the function
double f_prime(double x) {
   return 3*x*x - 1; // Derivative: 3x^2 - 1
}
int main()
{
   double x0, x1, tolerance;
   int maxIterations, iteration = 0;
   cout << "Enter initial guess: ";
   cin >> x0;
   cout << "Enter tolerance: ";
   cin >> tolerance;
   cout << "Enter maximum iterations: ";
   cin >> maxIterations;
   cout << "\nIter\t   x0\t\t   f(x0)\t   f'(x0)\t   x1\n";
   while (iteration < maxIterations)
   {
      double fx = f(x0);
      double fpx = f_prime(x0);

      if (fabs(fpx) < 1e-12) {
         cout << "Derivative near zero. Method fails.\n";
         return 1;
      }
     // Newton-Raphson formula
      x1 = x0 - fx / fpx;
```

```
        cout << iteration + 1 << "\t" << x0 << "\t" << fx << "\t" << fpx << "\t" << x1 << "\n";
        if (fabs(x1 - x0) < tolerance)
        {
            cout << "\nRoot found: " << x1 << " after " << iteration + 1 << " iterations.\n";
            return 0;
        }
        x0 = x1;
        iteration++;
    }
    cout << "\nMethod did not converge after " << maxIterations << " iterations.\n";
    return 0;
}
```

OUTPUT:

```
Enter initial guess: 1.5
Enter tolerance: 0.0001
Enter maximum iterations: 20
Iter    x0        f(x0)     f'(x0)     x1
1       1.5       -0.125    5.75       1.52174
2       1.52174   0.00213   5.94811    1.52138
3       1.52138   0         5.94800    1.52138

Root found: 1.52138 after 3 iterations.
```

AIM: Write a program in c++ to implement Gauss-elimination method.

PROGRAM:

```
#include <iostream>
#include <iomanip>
#include <cmath>
using namespace std;
int main()
{
   int n;
   cout << "Enter number of equations: ";
   cin >> n;
   double a[20][20]; // Augmented matrix
   cout << "Enter the augmented matrix (A|B):\n";
   for (int i = 0; i < n; i++)
  {
      for (int j = 0; j <= n; j++)
     {
         cin >> a[i][j];
      }
   }
   // Forward elimination
   for (int i = 0; i < n - 1; i++)
  {
      if (fabs(a[i][i]) < 1e-12)
     {
         cout << "Mathematical error: zero pivot.\n";
         return 1;
      }
      for (int j = i + 1; j < n; j++)
     {
         double ratio = a[j][i] / a[i][i];
         for (int k = i; k <= n; k++)
        {
            a[j][k] -= ratio * a[i][k];
         }
      }
   }
   // Back substitution
```

```
    double x[20];
    for (int i = n - 1; i >= 0; i--)
    {
        x[i] = a[i][n];
        for (int j = i + 1; j < n; j++)
        {
            x[i] -= a[i][j] * x[j];
        }
        if (fabs(a[i][i]) < 1e-12)
        {
            cout << "Mathematical error: zero pivot during back substitution.\n";
            return 1;
        }
        x[i] /= a[i][i];
    }
    cout << "\nSolution:\n";
    for (int i = 0; i < n; i++) {
        cout << "x[" << i + 1 << "] = " << x[i] << "\n";
    }
    return 0;
}
```

OUTPUT:

```
Enter number of equations: 3
Enter the augmented matrix (A|B):
2 1 -1 8
-3 -1 2 -11
-2 1 2 -3

Solution:
x1 = 2
x2 = 3
x3 = -1
```

AIM: Write a program in c++ to implement Gauss-Seidel method.

PROGRAM:

```
#include <iostream>
#include <cmath>
using namespace std;
int main()
{
   int n;
   cout << "Enter number of equations: ";
   cin >> n;
   double a[20][20], b[20], x[20], old_x[20];
   cout << "Enter the coefficients matrix (A):\n";
   for (int i = 0; i < n; i++)
   {
      for (int j = 0; j < n; j++)
      {
         cin >> a[i][j];
      }
   }
   cout << "Enter the constant terms (B):\n";
   for (int i = 0; i < n; i++)
  {
      cin >> b[i];
   }
   cout << "Enter initial guesses for variables:\n";
   for (int i = 0; i < n; i++)
  {
      cin >> x[i];
   }
   double tol;
   int maxIter;
   cout << "Enter tolerance (e.g., 0.0001): ";
   cin >> tol;
   cout << "Enter maximum iterations: ";
   cin >> maxIter;
   cout << "\nIteration process:\n";
   for (int iter = 1; iter <= maxIter; iter++)
  {
```

```
    for (int i = 0; i < n; i++)
    {
        old_x[i] = x[i];
    }
    for (int i = 0; i < n; i++)
    {
        double sum = b[i];
        for (int j = 0; j < n; j++)
        {
            if (j != i)
            {
                sum -= a[i][j] * x[j];
            }
        }
        x[i] = sum / a[i][i];
    }

    // Display current iteration
    cout << "Iter " << iter << ": ";
    for (int i = 0; i < n; i++)
    {
        cout << x[i] << " ";
    }
    cout << endl;
   // Check convergence
    bool converged = true;
    for (int i = 0; i < n; i++)
    {
        if (fabs(x[i] - old_x[i]) > tol)
        {
            converged = false;
            break;
        }
    }
    if (converged) {
        cout << "\nConverged in " << iter << " iterations.\n";
        break;
    }
}
cout << "\nSolution:\n";
```

```
    for (int i = 0; i < n; i++) {
        cout << "x" << i + 1 << " = " << x[i] << endl;
    }
    return 0;
}
```

OUTPUT:

```
Enter number of equations: 3
Enter the coefficients matrix (A):
4 -1 0
-1 4 -1
0 -1 3
Enter the constant terms (B):
15 10 10
Enter initial guesses for variables:
0 0 0
Enter tolerance (e.g., 0.0001): 0.0001
Enter maximum iterations: 25

Iteration process:
Iter 1: 3.75 3.4375 4.47917
Iter 2: 4.60938 4.27214 4.75738
Iter 3: 4.81804 4.89386 4.96462
Iter 4: 4.97347 4.98452 4.99484
Iter 5: 4.99613 4.99824 4.99941

Converged in 5 iterations.

Solution:
x1 = 4.99613
x2 = 4.99824
x3 = 4.99941
```

GIRIJANANDA CHOWDHURY UNIVERSITY, GUWAHATI(ASSAM)

AIM: Write a program in c++ to implement Euler's method.

PROGRAM:

```
#include <iostream>
#include <iomanip>
using namespace std;
// Define the derivative function f(x, y) = dy/dx
double f(double x, double y)
{
   // Example: dy/dx = x + y
   return x + y;
}
int main()
{
   double x0, y0, h, xn, x, y;
   int n;
   cout << "Enter initial value of x (x0): ";
   cin >> x0;
   cout << "Enter initial value of y (y0): ";
   cin >> y0;
   cout << "Enter step size (h): ";
   cin >> h;
   cout << "Enter the value of x at which y is required (xn): ";
   cin >> xn;
   // Number of steps
   n = (int)((xn - x0) / h);
   // Initialize variables
   x = x0;
   y = y0;
  // Display table header
   cout << fixed << setprecision(4);
   cout << "\nStep\tX\t\tY\t\tf(x,y)\n";
   cout << "-----------------------------------------\n";
 // Euler's method iteration
   for (int i = 0; i < n; i++)
   {
      double slope = f(x, y);
      cout << i + 1 << "\t" << x << "\t" << y << "\t" << slope << endl;
      y = y + h * slope;  // y_{n+1} = y_n + h*f(x_n,y_n)
```

```
        x = x + h;          // x_{n+1} = x_n + h
    }

    cout << "\nApproximate value of y at x = " << xn << " is: " << y << endl;
    return 0;
}
```

OUTPUT:

```
Enter initial value of x (x0): 0
Enter initial value of y (y0): 1
Enter step size (h): 0.1
Enter the value of x at which y is required (xn): 0.5

Step    X       Y       f(x,y)
-----------------------------------------
1       0.0000  1.0000  1.0000
2       0.1000  1.1000  1.2000
3       0.2000  1.2200  1.4200
4       0.3000  1.3620  1.6620
5       0.4000  1.5282  1.9282

Approximate value of y at x = 0.5 is: 1.7210
```

GIRIJANANDA CHOWDHURY UNIVERSITY, GUWAHATI(ASSAM)

AIM: Write a program in c++ to implement Runge-Kutta – 4th order method.

PROGRAM:

```
#include <iostream>
#include <iomanip>
using namespace std;
// Define the derivative function f(x, y) = dy/dx
double f(double x, double y)
{
   // Example: dy/dx = x + y
   return x + y;
}
int main()
{
   double x0, y0, xn, h, x, y, k1, k2, k3, k4;
   int n;
   cout << "Enter initial value of x (x0): ";
   cin >> x0;
   cout << "Enter initial value of y (y0): ";
   cin >> y0;
   cout << "Enter step size (h): ";
   cin >> h;
   cout << "Enter the value of x at which y is required (xn): ";
   cin >> xn;
   // Number of steps
   n = (int)((xn - x0) / h)
   // Initialize variables
   x = x0;
   y = y0;
   cout << fixed << setprecision(4);
   cout << "\nStep\tX\t\tY\t\tk1\t\tk2\t\tk3\t\tk4\n";
   cout << "-------------------------------------------------------------------------------\n";
   // Runge-Kutta iterations
   for (int i = 0; i < n; i++)
  {
      k1 = h * f(x, y);
      k2 = h * f(x + h / 2, y + k1 / 2);
      k3 = h * f(x + h / 2, y + k2 / 2);
      k4 = h * f(x + h, y + k3);
```

```
        cout << i + 1 << "\t" << x << "\t" << y << "\t" << k1 << "\t" << k2 << "\t" << k3 << "\t"
<< k4 << endl;

        y = y + (k1 + 2 * k2 + 2 * k3 + k4) / 6;
        x = x + h;
    }
    cout << "\nApproximate value of y at x = " << xn << " is: " << y << endl;
    return 0;
}
```

OUTPUT:

```
Enter initial value of x (x0): 0
Enter initial value of y (y0): 1
Enter step size (h): 0.1
Enter the value of x at which y is required (xn): 0.5

Step    X       Y       k1      k2      k3      k4
-------------------------------------------------------------------------------
1       0.0000  1.0000  0.1000  0.1050  0.1053  0.1205
2       0.1000  1.1103  0.1210  0.1270  0.1274  0.1454
3       0.2000  1.2428  0.1443  0.1515  0.1522  0.1739
4       0.3000  1.3997  0.1699  0.1784  0.1793  0.2038
5       0.4000  1.5836  0.1984  0.2084  0.2095  0.2356

Approximate value of y at x = 0.5 is: 1.7974
```

AIM: Write a program in c++ to implement predictor-corrector method.

PROGRAM:

```
#include <iostream>
#include <iomanip>
using namespace std;
// Define the derivative function f(x,y) = dy/dx
double f(double x, double y)
{
   // Example: dy/dx = x + y
   return x + y;
}
int main()
{
   double x0, y0, xn, h, x, y, yp;
   int n;
   cout << "Enter initial value of x (x0): ";
   cin >> x0;
   cout << "Enter initial value of y (y0): ";
   cin >> y0;
   cout << "Enter step size (h): ";
   cin >> h;
   cout << "Enter the value of x at which y is required (xn): ";
   cin >> xn;
   // Number of steps
   n = (int)((xn - x0) / h);
   // Initialize variables
   x = x0;
   y = y0;
   cout << fixed << setprecision(4);
   cout << "\nStep\tX\t\tY\t\tf(x,y)\t\tPredictor(y_p)\tCorrected(y)\n";
   cout << "-------------------------------------------------------------------------\n";
   // Predictor-Corrector iteration
   for (int i = 0; i < n; i++)
  {
      double slope1 = f(x, y);
     // Predictor using Euler's method
      yp = y + h * slope1;
    // Corrector using improved formula
```

```
        double slope2 = f(x + h, yp);
        y = y + (h / 2) * (slope1 + slope2);

        cout << i + 1 << "\t" << x << "\t" << y << "\t" << slope1 << "\t\t" << yp << "\t\t" << y
<< endl;
        x = x + h;
    }
    cout << "\nApproximate value of y at x = " << xn << " is: " << y << endl;
    return 0;
}
```

OUTPUT:

```
Enter initial value of x (x0): 0
Enter initial value of y (y0): 1
Enter step size (h): 0.1
Enter the value of x at which y is required (xn): 0.5

Step    X       Y       f(x,y)   Predictor(y_p)   Corrected(y)
------------------------------------------------------------------------
1       0.0000  1.1100  1.0000   1.1000           1.1100
2       0.1000  1.2426  1.2100   1.2310           1.2426
3       0.2000  1.3995  1.4426   1.3869           1.3995
4       0.3000  1.5833  1.6995   1.5695           1.5833
5       0.4000  1.7971  1.9833   1.7817           1.7971

Approximate value of y at x = 0.5 is: 1.7971
```

GIRIJANANDA CHOWDHURY UNIVERSITY, GUWAHATI(ASSAM)

AIM: Write a program in c++ to implement two ordinary differential equation using numerical integration.

PROGRAM:

```
#include <iostream>
#include <iomanip>
#include <cmath>
using namespace std;
// Define the system of two ODEs
double f1(double x, double y, double z)
{
   return z;   // dy/dx = z
}
double f2(double x, double y, double z)
{
   return -y;  // dz/dx = -y
}
int main()
{
   double x0, y0, z0, h, xn;
   int n;
   cout << "Enter initial value of x (x0): ";
   cin >> x0;
   cout << "Enter initial value of y (y0): ";
   cin >> y0;
   cout << "Enter initial value of z (z0): ";
   cin >> z0;
   cout << "Enter step size (h): ";
   cin >> h;
   cout << "Enter final value of x (xn): ";
   cin >> xn;
   // Number of steps
   n = (int)((xn - x0) / h);
   // Print table header
   cout << fixed << setprecision(6);
   cout << "\n----------------------------------------------------------\n";
   cout << "Step\t   x\t\t   y\t\t   z\n";
   cout << "----------------------------------------------------------\n";
```

```
    // Euler's method iteration
    for (int i = 0; i <= n; i++) {
    cout << i << "\t" << x0 << "\t" << y0 << "\t" << z0 << endl;
       // Update using Euler's formula
        double y_next = y0 + h * f1(x0, y0, z0);
        double z_next = z0 + h * f2(x0, y0, z0);
        x0 += h;
        y0 = y_next;
        z0 = z_next;
    }
    cout << "-------------------------------------------------------\n";
    cout << "Approximate solution at x = " << xn << ":\n";
    cout << "y ≈ " << y0 << ",  z ≈ " << z0 << endl;
    return 0;
}
```

OUTPUT:

```
-------------------------------------------------------
Step      x          y          z
-------------------------------------------------------
0     0.000000   0.000000   1.000000
1     0.100000   0.100000   1.000000
2     0.200000   0.200000   0.990000
3     0.300000   0.299000   0.970000
4     0.400000   0.396000   0.940100
5     0.500000   0.490010   0.900500
6     0.600000   0.580060   0.851499
7     0.700000   0.665210   0.793493
8     0.800000   0.744559   0.726972
9     0.900000   0.817256   0.652516
10    1.000000   0.882508   0.570791
-------------------------------------------------------
Approximate solution at x = 1:
y ≈ 0.882508,  z ≈ 0.570791
```

GIRIJANANDA CHOWDHURY UNIVERSITY, GUWAHATI(ASSAM)

AIM: Write a program in c++ to implement Trapezoidal Rule of Numerical integration.

PROGRAM:

```
#include <iostream>
#include <iomanip>
#include <cmath>
using namespace std;
// Define the function f(x)
double f(double x)
{
   return 1 / (1 + x * x);  // Example: f(x) = 1 / (1 + x^2)
}
int main()
{
   double a, b;   // Lower and upper limits
   int n;         // Number of sub-intervals
   cout << "Enter lower limit (a): ";
   cin >> a;
   cout << "Enter upper limit (b): ";
   cin >> b;
   cout << "Enter number of sub-intervals (n): ";
   cin >> n;
   double h = (b - a) / n;   // Step size
   double sum = f(a) + f(b); // First and last terms
   // Summing the middle terms
   for (int i = 1; i < n; i++)
  {
      double x = a + i * h;
      sum += 2 * f(x);
   }
   // Applying trapezoidal rule formula
   double result = (h / 2) * sum;
   cout << fixed << setprecision(6);
   cout << "\nApproximate value of the integral = " << result << endl;
   return 0;
}
```

OUTPUT:

```
Enter lower limit (a): 1
Enter upper limit (b): 2
Enter number of sub-intervals (n): 4
Approximate value of the integral = 0.321750
```

GIRIJANANDA CHOWDHURY UNIVERSITY, GUWAHATI(ASSAM)

AIM: Write a program in c++ to implement elliptic boundary value problem using method of finite differences.

PROGRAM:

```
#include <bits/stdc++.h>
using namespace std;
/*
  Elliptic BVP via Finite Differences (5-point stencil) + Gauss-Seidel SOR

  Problem:
    - Solve:  u_xx + u_yy = f(x, y)   on (a,b) x (c,d)
    - Dirichlet BC: u = g on the boundary

  Customize:
    - Edit f(x,y) below for Poisson RHS (set to 0.0 for Laplace)
    - Edit boundary functions g_left, g_right, g_bottom, g_top
*/
static inline double f_rhs(double x, double y)
{
    // RHS f(x,y). Set 0.0 for Laplace's equation.
    return 0.0;
}
// Dirichlet boundary data:
static inline double g_left(double y)   { return 0.0; }                    // u(a,y)
static inline double g_right(double y)  { return 0.0; }                     // u(b,y)
static inline double g_bottom(double x) { return 0.0; }                       // u(x,c)
static inline double g_top(double x)    { return sin(M_PI * x); }              // u(x,d)
/*
  Optional exact solution for the default Laplace test:
  u(x,y) = sinh(pi y)/sinh(pi) * sin(pi x)
*/
static inline double u_exact(double x, double y)
{
    return (sinh(M_PI * y) / sinh(M_PI)) * sin(M_PI * x);
}
int main()
{
    ios::sync_with_stdio(false);
    cin.tie(nullptr);
```

```
// Domain and grid
double a = 0.0, b = 1.0;   // x in [a,b]
double c = 0.0, d = 1.0;   // y in [c,d]
int Nx = 50;               // number of intervals in x (Nx+1 points)
int Ny = 50;               // number of intervals in y (Ny+1 points)
double tol = 1e-8;         // stopping tolerance on infinity-norm change
int maxIter = 100000;      // safety cap on iterations
double omega = 0.0;        // SOR parameter; 0 or <=1 means plain Gauss-Seidel
// ---- Optional: read parameters from stdin (uncomment if desired) ----
// cout << "Enter Nx Ny tol maxIter omega (omega in (1,2) for SOR, else <=1 for GS):\n";
// if (!(cin >> Nx >> Ny >> tol >> maxIter >> omega)) return 0;
double hx = (b - a) / Nx;
double hy = (d - c) / Ny;
double hx2 = hx * hx, hy2 = hy * hy;
// If omega not provided, pick a decent guess (near-optimal for Laplace on rectangle)
if (omega <= 0.0 || omega > 2.0)
{
    double rho = max(sin(M_PI * hx / 2.0), sin(M_PI * hy / 2.0));
    omega = 2.0 / (1.0 + sqrt(1.0 - rho * rho)); // commonly used heuristic
}
// Allocate grid: u[i][j] corresponds to (x_i, y_j) = (a + i*hx, c + j*hy)
vector<vector<double>> u(Nx + 1, vector<double>(Ny + 1, 0.0));
// Apply Dirichlet boundary conditions
for (int j = 0; j <= Ny; ++j)
{
    double y = c + j * hy;
    u[0][j]    = g_left(y);    // x = a
    u[Nx][j]   = g_right(y);   // x = b
}
for (int i = 0; i <= Nx; ++i)
{
    double x = a + i * hx;
    u[i][0]    = g_bottom(x);  // y = c
    u[i][Ny]   = g_top(x);     // y = d
}
// Iterative solve: Gauss-Seidel with SOR
const double denom = 2.0 * (1.0 / hx2 + 1.0 / hy2);
int it = 0;
double maxChange = 0.0;
```

```
   do
   {
      maxChange = 0.0;
      for (int i = 1; i < Nx; ++i)
      {
         double xi = a + i * hx;
         for (int j = 1; j < Ny; ++j)
         {
            double yj = c + j * hy;
             // 5-point Poisson stencil:
            // (-u_{i-1,j} + 2u_{i,j} - u_{i+1,j})/hx^2 + (-u_{i,j-1} + 2u_{i,j} - u_{i,j+1})/hy^2
= f
            // => u_{i,j} = [ (u_{i+1,j}+u_{i-1,j})/hx^2 + (u_{i,j+1}+u_{i,j-1})/hy^2 - f ] /
denom
            double rhs = (u[i+1][j] + u[i-1][j]) / hx2
                     + (u[i][j+1] + u[i][j-1]) / hy2
                     - f_rhs(xi, yj);
            double u_new = rhs / denom;

            // SOR relaxation
            double updated = (1.0 - omega) * u[i][j] + omega * u_new;
            maxChange = max(maxChange, fabs(updated - u[i][j]));
            u[i][j] = updated;
         }
      }
      ++it;
      if (it % 1000 == 0)
      {
         cerr << "Iter " << it << " maxChange=" << maxChange << "\n";
      }
   } while (maxChange > tol && it < maxIter);
   cout.setf(std::ios::fixed); cout << setprecision(8);
   cout << "Converged in " << it << " iterations with maxChange = " << maxChange
        << ", omega = " << omega << "\n";

   // Print a small sample of the solution: centerline y ~ 0.5
   cout << "\nSample: u(x, 0.5) for x in [0,1] (every 5th node)\n";
   int jmid = int(round(0.5 / hy));
   jmid = std::min(std::max(jmid, 0), Ny);
   for (int i = 0; i <= Nx; i += max(1, Nx/10)) {
```

```
        double x = a + i * hx;
        cout << "x=" << setw(7) << setprecision(4) << x
             << "  u=" << setw(12) << setprecision(8) << u[i][jmid];
        // Compare with exact (only valid for default BC and f=0)
        cout << "  (u_exact=" << setw(12) << setprecision(8) << u_exact(x, 0.5) << ")";
        cout << "\n";
    }

    // Optional: write full grid to CSV (uncomment if needed)
    // {
    //     ofstream fout("solution.csv");
    //     fout.setf(std::ios::fixed);
    //     fout << setprecision(12);
    //     fout << "x,y,u\n";
    //     for (int j = 0; j <= Ny; ++j) {
    //         double y = c + j * hy;
    //         for (int i = 0; i <= Nx; ++i) {
    //             double x = a + i * hx;
    //             fout << x << "," << y << "," << u[i][j] << "\n";
    //         }
    //     }
    //     fout.close();
    //     cout << "\nWrote solution to solution.csv\n";
    // }
      return 0;
}
```

OUTPUT:

Converged in 5421 iterations with maxChange = 0.00000001, omega = 1.9045

Sample: u(x, 0.5) for x in [0,1] (every 5th node)

```
x=  0.0000  u=   0.00000000  (u_exact=   0.00000000)
x=  0.1000  u=   0.13536782  (u_exact=   0.13533528)
x=  0.2000  u=   0.24983645  (u_exact=   0.24970928)
x=  0.3000  u=   0.33303197  (u_exact=   0.33298379)
x=  0.4000  u=   0.37793216  (u_exact=   0.37782942)
x=  0.5000  u=   0.38266639  (u_exact=   0.38257311)
x=  0.6000  u=   0.34795121  (u_exact=   0.34789148)
x=  0.7000  u=   0.27761043  (u_exact=   0.27759692)
```

x= 0.8000 u= 0.17825877 (u_exact= 0.17825999)
x= 0.9000 u= 0.05860325 (u_exact= 0.05860550)
x= 1.0000 u= 0.00000000 (u_exact= 0.00000000)