Problem Statement 1: WAP to find the roots of non-linear equation using Bisection method.

Code:

```
#include<iostream>
using namespace std;
#define EPSILON 0.01
double func(double x)
{
   return x*x*x - x*x + 2;
}
void bisection(double a, double b)
{
   if (func(a) * func(b) >= 0)
   {
      cout << "You have not assumed right a and b\n";
      return;
   }
   double c = a;
   while ((b-a) >= EPSILON)
   {
      c = (a+b)/2;
      if (func(c) == 0.0)
         break;
      else if (func(c)*func(a) < 0)
         b = c;
      else
         a = c;
   }
   cout << "The value of root is : " << c;
}
int main()
{
   double a, b;
   cout << "Enter values of a and b: ";
   cin >> a >> b;
   bisection(a, b);
   return 0;
}
```

Output:

```
Enter values of a and b: -100 250
The value of root is : -1.00174
```

Problem Statement 2: WAP to find the roots of non-linear equation using False position method.

Code:

```
#include<iostream>
using namespace std;
#define MAX_ITER 1000000
double func(double x)
{
   return x*x*x - x*x + 2;
}
void regulaFalsi(double a, double b)
{
   if (func(a) * func(b) >= 0)
   {
      cout << "You have not assumed right a and b\n";
      return;
   }
   double c = a;
   for (int i=0; i < MAX_ITER; i++)
   {
      c = (a*func(b) - b*func(a))/ (func(b) - func(a));
      if (func(c)==0)
         break;
      else if (func(c)*func(a) < 0)
         b = c;
      else
         a = c;
   }
   cout << "The value of root is : " << c;
}
int main()
{
   double a,b;
   cout << "Enter values of a and b: ";
   cin >> a >> b;
   regulaFalsi(a, b);
   return 0;
}
```

Output:

```
Enter values of a and b: 1
2
You have not assumed right a and b
```

Problem Statement 3: WAP to find the roots of non-linear equation using Newton's Raphson method.

Code:

```
#include<iostream>
#include<cmath>
#define EPSILON 0.001
using namespace std;
double func(double x)
{
   return x*x*x - x*x + 2;
}
double derivFunc(double x)
{
   return 3*x*x - 2*x;
}
void newtonRaphson(double x)
{
   double h = func(x) / derivFunc(x);
   while (abs(h) >= EPSILON)
   {
      h = func(x)/derivFunc(x);
      x = x - h;
   }
   cout << "The value of the root is : " << x;
}
int main()
{
   double x0;
   cout << "Enter the value of x0: ";
   cin >> x0;
   newtonRaphson(x0);
   return 0;
}
```

Output:

```
Enter the value of x0: -20
The value of the root is : -1
```

Problem Statement 4: WAP to find the roots of non-linear equation using Iteration method.

Code:

```
#include <stdio.h>
#include <math.h>
double g(double x)
{
   return (x * x + 2) / 3;
}
void iterationMethod(double x0, double tolerance, int maxIterations){
   double x1;
   int iteration = 0;
   printf("Iteration\t x0\t\t g(x0)\t\t Error\n");
   while (iteration < maxIterations){
      x1 = g(x0);
      printf("%d\t\t %lf\t %lf\t %lf\n", iteration + 1, x0, x1, fabs(x1 - x0));
      if (fabs(x1 - x0) < tolerance){
         printf("\nThe root is approximately: %lf\n", x1);
         return;
      }
      x0 = x1;
      iteration++;
   }
   printf("\nMax iterations reached. The root is approximately: %lf\n", x1);
}
int main(){
   double x0, tolerance;
   int maxIterations;
   printf("Enter the initial guess (x0):\n");
   scanf("%lf", &x0);
   printf("Enter the tolerance value:\n");
   scanf("%lf", &tolerance);
   printf("Enter the maximum number of iterations:\n");
   scanf("%d", &maxIterations);
   iterationMethod(x0, tolerance, maxIterations);
}
```

Output:

```
Enter the initial guess (x0):
2.5
Enter the tolerance value:
0.0001
Enter the maximum number of iterations:
5
Iteration        x0              g(x0)           Error
1                2.500000        2.750000        0.250000
2                2.750000        3.187500        0.437500
3                3.187500        4.053385        0.865885
4                4.053385        6.143311        2.089926
5                6.143311        13.246757       7.103446

Max iterations reached. The root is approximately: 13.246757
```

Problem Statement 5: WAP to interpolate numerically using Newton's Forward Difference method.

Code:

```
#include <iostream>
#include <vector>
#include <iomanip>
using namespace std;
float u_cal(float u, int n) {
    float temp = u;
    for (int i = 1; i < n; i++) {
        temp = temp * (u - i);
    }
    return temp;
}
int fact(int n) {
    int f = 1;
    for (int i = 2; i <= n; i++) {
        f *= i;
    }
    return f;
}
int main() {
    int n;
    cout << "Enter the number of data points: ";
    cin >> n;
    vector<float> x(n);
    vector<vector<float>> y(n, vector<float>(n));
    cout << "Enter the x values: \n";
    for (int i = 0; i < n; i++) {
        cin >> x[i];
    }
    cout << "Enter the y values (function values at x): \n";
    for (int i = 0; i < n; i++) {
        cin >> y[i][0];
    }
    for (int i = 1; i < n; i++) {
        for (int j = 0; j < n - i; j++) {
            y[j][i] = y[j + 1][i - 1] - y[j][i - 1];
        }
    }
    cout << "\nForward Difference Table:\n";
    for (int i = 0; i < n; i++) {
        cout << setw(10) << x[i] << "\t";
        for (int j = 0; j < n - i; j++) {
            cout << setw(10) << y[i][j] << "\t";
        }
```

```
        cout << endl;
    }
    float value;
    cout << "Enter the value to interpolate at: ";
    cin >> value;
    float sum = y[0][0];
    float u = (value - x[0]) / (x[1] - x[0]);
    for (int i = 1; i < n; i++) {
        sum = sum + (u_cal(u, i) * y[0][i]) / fact(i);
    }
    cout << "\nThe interpolated value at x = " << value << " is: " << sum << endl;
    return 0;
}
```

Output:

```
Enter the number of data points: 4
Enter the x values:
45
50 55 60
Enter the y values (function values at x):
0.7071 0.7660 0.8192 0.8660

Forward Difference Table:
        45          0.7071          0.0589      -0.00569999     -0.000699997
        50           0.766          0.0532      -0.00639999
        55          0.8192          0.0468
        60           0.866
Enter the value to interpolate at: 52

The interpolated value at x = 52 is: 0.788003
```

Problem Statement 6: WAP to interpolate numerically using Newton's Backward Difference method.

Code:

```
#include <iostream>
#include <vector>
#include <iomanip>
using namespace std;
float u_cal(float u, int n) {
    float temp = u;
    for (int i = 1; i < n; i++) {
        temp = temp * (u + i);
    }
    return temp;
}
int fact(int n) {
    int f = 1;
    for (int i = 2; i <= n; i++) {
        f *= i;
    }
    return f;
}
int main() {
    int n;
    cout << "Enter the number of data points: ";
    cin >> n;
    vector<float> x(n);
    vector<vector<float>> y(n, vector<float>(n));
    cout << "Enter the x values: \n";
    for (int i = 0; i < n; i++) {
        cin >> x[i];
    }
    cout << "Enter the y values (function values at x): \n";
    for (int i = 0; i < n; i++) {
        cin >> y[i][0];
    }
    for (int i = 1; i < n; i++) {
        for (int j = n - 1; j >= i; j--) {
            y[j][i] = y[j][i - 1] - y[j - 1][i - 1];
        }
    }
    cout << "\nBackward Difference Table:\n";
    for (int i = 0; i < n; i++) {
        cout << setw(10) << x[i] << "\t";
        for (int j = 0; j <= i; j++) {
            cout << setw(10) << y[i][j] << "\t";
        }
```

```
        cout << endl;
    }
    float value;
    cout << "Enter the value to interpolate at: ";
    cin >> value;
    float sum = y[n - 1][0];
    float u = (value - x[n - 1]) / (x[1] - x[0]);
    for (int i = 1; i < n; i++) {
        sum = sum + (u_cal(u, i) * y[n - 1][i]) / fact(i);
    }
    cout << "\nThe interpolated value at x = " << value << " is: " << sum << endl;
    return 0;
}
```

Output:

```
Enter the number of data points: 5
Enter the x values:
1891 1901 1911 1921 1931
Enter the y values (function values at x):
46 66 81 93 101

Backward Difference Table:
      1891              46
      1901              66              20
      1911              81              15              -5
      1921              93              12              -3               2
      1931             101               8              -4              -1
-3
Enter the value to interpolate at: 1925

The interpolated value at x = 1925 is: 96.8368
```

Problem Statement 7: WAP to interpolate numerically using Lagrange's method.

Code:

```
#include  <iostream>
using namespace std;
struct Data{
   int x, y;
};
double interpolate(Data f[], int xi, int n){
   double result = 0;
   for (int i = 0; i < n; i++){
      double term =  f[i].y;
      for (int j = 0; j < n; j++)
      {
         if (j != i) term = term * (xi - f[j].x) / double(f[i].x - f[j].x);
      }
      result += term;
   }
   return result;
}
int main(){
   int n;
   cout << "Enter the number of data points: ";
   cin >> n;
   Data *f = new Data[n];
   cout << "Enter the data points (x y):" << endl;
   for (int i = 0; i < n; i++){
      cout << "Point " << i + 1 << ": ";
      cin >> f[i].x >> f[i].y;
   }
   int xi;
   cout << "Enter the value of x at which you want to interpolate: ";
   cin >> xi;
   cout << "Value of f(" << xi << ") is : " << interpolate(f, xi, n) << endl;
   delete[] f;
   return 0;
}
```

Output:

```
Enter the number of data points: 4
Enter the data points (x y):
Point 1: 0 2
Point 2: 1 3
Point 3: 2 12
Point 4: 5 100
Enter the value of x at which you want to interpolate: 3
Value of f(3) is : 30.3
```

Problem Statement 8: WAP to Integrate numerically using Trapezoidal rule.

Code:

```
#include<stdio.h>
using namespace std;
float y(float x)
{
   return 1 / (1 + x * x);
}
float trapezoidal(float a, float b, int n)
{
   float h = (b - a) / n;
   float s = y(a) + y(b);
   for (int i = 1; i < n; i++)
      s += 2 * y(a + i * h);
   return (h / 2) * s;
}
int main()
{
   float x0, xn;
   int n;
   printf("Enter the lower limit of integration (x0): ");
   scanf("%f", &x0);
   printf("Enter the upper limit of integration (xn): ");
   scanf("%f", &xn);
   printf("Enter the number of grid points (n): ");
   scanf("%d", &n);
   if (n <= 0) {
      printf("Number of grid points must be positive.\n");
      return 1;
   }
   printf("Value of integral is: %6.4f\n", trapezoidal(x0, xn, n));
   return 0;
}
```

Output:

```
Enter the lower limit of integration (x0): 0
Enter the upper limit of integration (xn): 1
Enter the number of grid points (n): 6
Value of integral is: 0.7842
```

Problem Statement 9: WAP to Integrate numerically using Simpson's 1/3 rule.

Code:

```
#include <iostream>
#include <cmath>
using namespace std;
float func(float x){
   return log(x);
}
float simpsons_(float ll, float ul, int n){
   if (n % 2 != 0) {
      cout << "Number of intervals must be even for Simpson's Rule.\n";
      return -1;
   }
   float h = (ul - ll) / n;
   float res = func(ll) + func(ul);
   for (int i = 1; i < n; i++) {
      float x = ll + i * h;
      if (i % 2 == 0)
         res += 2 * func(x);
      else
         res += 4 * func(x);
   }
   res = res * (h / 3);
   return res;
}
int main(){
   float lower_limit, upper_limit;
   int n;
   cout << "Enter the lower limit of integration: ";
   cin >> lower_limit;
   cout << "Enter the upper limit of integration: ";
   cin >> upper_limit;
   cout << "Enter the number of intervals (must be even): ";
   cin >> n;
   float result = simpsons_(lower_limit, upper_limit, n);
   if (result != -1)
      cout << "The approximate value of the integral is: " << result << endl;
   return 0;
}
```

Output:

```
Enter the lower limit of integration: 4
Enter the upper limit of integration: 5
Enter the number of intervals (must be even): 6
The approximate value of the integral is: 1.50201
```

Problem Statement 10: WAP to Integrate numerically using Simpson's 3/8 rule.

Code:

```
#include  <iostream>
using namespace std;
float func(float x){
   return (1 / (1 + x * x));
}
float calculate(float lower_limit, float upper_limit, int interval_limit){
   if (interval_limit % 3 != 0) {
      cout << "Number of intervals must be a multiple of 3 for Simpson's 3/8 Rule.\n";
      return -1;
   }
   float interval_size = (upper_limit - lower_limit) / interval_limit;
   float sum = func(lower_limit) + func(upper_limit);
   for (int i = 1; i < interval_limit; i++){
      if (i % 3 == 0)
         sum += 2 * func(lower_limit + i * interval_size);
      else
         sum += 3 * func(lower_limit + i * interval_size);
   }
   return (3 * interval_size / 8) * sum;
}
int main(){
   float lower_limit, upper_limit;
   int interval_limit;
   cout << "Enter the lower limit of integration: ";
   cin >> lower_limit;
   cout << "Enter the upper limit of integration: ";
   cin >> upper_limit;
   cout << "Enter the number of intervals (must be a multiple of 3): ";
   cin >> interval_limit;
   float integral_res = calculate(lower_limit, upper_limit, interval_limit);
   if (integral_res != -1) {
      cout << "The approximate value of the integral is: " << integral_res << endl;
   }
   return 0;
}
```

Output:

```
Enter the lower limit of integration: 1
Enter the upper limit of integration: 10
Enter the number of intervals (must be a multiple of 3): 9
The approximate value of the integral is: 0.692532
```