```
// Vahe Karamian - Project 2 - CS 301 - Dr. H. K. Liu
// Filename: Root.cpp

#include <iostream.h>
#include <fstream.h>
#include <iomanip.h>
#include <math.h>

float valueAtPoint( float x, float coefficients[], int N );
void  bisectionMethod( float coefficients[], int N );
void  quadraticEquation( float coefficients[] );
void  newtonMethod( float coefficients[], int N, float c );

int main( )
{
  ifstream dataFile( "Root.dat", ios::in );
  if( !dataFile )
  {
    cout<<"\n Data file not found! \n"<<endl;
    return 0;
  }

  // Step 1 - get the polynomial from the data file
  // Format
  // ======
  // LINE 1: N               1 integer, the number of polynomial
  // LINE 2: An An-1 An-2 ... A1 A0 N+1 floats separated by one space,
  //                        coefficients of polynomial

  int N = 0;    // The number of polynomial, initialize to zero

  dataFile >> N;  // Get the number of polynomial

  // Create coefficients data array
  float *coefficients = new float[N+1];
  int  index = 0;// Index of the coefficient data array

  // Initialize coefficients to zero
  for( int z=0; z<N+1; z++ )
    coefficients[z] = 0;

  // Get the N+1 ceofficients
  while( dataFile >> coefficients[index] ) { index++; }

  // Print the polynomial for testing
  cout<<"The root(s) to the polynomial ";
  for( int i=0; i<N+1; i++ )
  {
    cout<<coefficients[i]<<"x^"<<N-i<<" ";
  }
  cout<<"are: \n";

  // Check to see if polynomial is of degree 2
  if( N == 2 )
  {
    quadraticEquation( coefficients );
  }
  else
  {
    // Step 2 - starting from 0.0 use step size 0.5 to find a and b such that
    //          sign( f(a) ) <> sign( f(b) )
    float a = 0.0;
    float b = (float)(a + 0.5);

    while( N > 2 )
    {
      bisectionMethod( coefficients, N );
      N--;
    }
  }

  quadraticEquation( coefficients );

  return 0;
}

void quadraticEquation( float coefficients[] )
{
  // We can use the quadratic equation
```

```
  float r1 = (float)( ( -1*coefficients[1] + sqrt( pow(coefficients[1],2) - 4*coefficients[0]*coefficients[2] ) ) / ( 2*
coefficients[0] ) );
  float r2 = (float)( ( -1*coefficients[1] - sqrt( pow(coefficients[1],2) - 4*coefficients[0]*coefficients[2] ) ) / ( 2*
coefficients[0] ) );

  cout<< r1 <<" "<<r2<<endl;
}

// Calculate the value of the function at point x
float valueAtPoint( float x, float coefficients[], int N )
{
  float value = 0.0;          // initialize value to 0.0
  for( int i=0; i<N+1; i++ )
    value = (float)(value + (float)( coefficients[i]*pow(x, N-i) ));
  return value;               // return value of polynomial at x
}

// Perfom Bisection Method 3 Times
void bisectionMethod( float coefficients[], int N )
{
  float a1, a2, b1, b2, c, u, v;
  float a, b;

  b1 = 0.0;
  b2 = 0.0;

  while( true )
  {
    // Check to see if sign(u) <> sign(v), in other words
    // sign( f(a) ) <> sign( f(b) )

    a1 = b1;
    b1 = (float)(a1 + 0.5);

    u = valueAtPoint( a1, coefficients, N ); // compute f(a)
    v = valueAtPoint( b1, coefficients, N ); // compute f(b)

    cout<<"a1 = "<<a1<<" f(a1) = "<< u <<endl;
    cout<<"b1 = "<<b1<<" f(b1) = "<< v <<endl;

    if( (u < 0 && v > 0) || (u > 0 && v < 0) )
    {
      a = a1;
      b = b1;
      break;
    }

    a2 = b2;
    b2 = (float)(a2 - 0.5);

    u = valueAtPoint( a2, coefficients, N ); // compute f(a)
    v = valueAtPoint( b2, coefficients, N ); // compute f(b)

    cout<<"a2 = "<<a2<<" f(a2) = "<< u <<endl;
    cout<<"b2 = "<<b2<<" f(b2) = "<< v <<endl;

    if( (u < 0 && v > 0) || (u > 0 && v < 0) )
    {
      a = a2;
      b = b2;
      break;
    }
  }

  // Do 3 itirations of Bisection Method
  for( int i=0; i<3; i++ )
  {
    c = (float) ( (a+b)/2 );
    if( c == 0 )
    {
      // we found a root print and break
      cout<<c;
      return;
    }
    else
    {
      u = valueAtPoint( a, coefficients, N );
      v = valueAtPoint( b, coefficients, N );
```

```
      // Check to see if sign(u) <> sign(v), in other words
      // sign( f(a) ) <> sign( f(b) )
      if( (u < 0 && v > 0) || (u > 0 && v < 0) )
      {
        b = c;
      }
      else
      {
        a = c;
      }
    }
  }

  c = (float)( (a+b)/2 );

  // cout<<"a = "<<a<<"and b = "<<b<<" which makes c = "<< c <<endl;
  // cout<<"One of the roots is "<< c <<" using Bisection Method with 3 itirations"<<endl;

  // Use Newton's Method to find the root
  newtonMethod( coefficients, N, c );
}

void newtonMethod( float coefficients[], int N, float c )
{
  // code here ...
}
```