```
// Vahe Karamian - CS 301 - Numerical Methods - Dr. H. K. Liu
// Filename: Gaussian.cpp


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

// Global variables
float solution0[10][11];
float solution1[10][11];
float *NaiveSolution; // [10];
float *NotNaiveSolut; // [10];

// Function prototypes
void   NotNaive( int degree );
void   Naive( int degree );
void   Pivot( int x, int degree );
float* ReverseSub( float x[][11], int y );
float  RMS( float x[][11], float y[], int degree );
void   Print( float x[], int degree );

int main( )
{
  bool FLAG = false;

  int   N = 0;    // The number of degree, initialize to zero
  int   c = 0;    // counter for reading in the linear equation

  // Create a filestream
  ifstream dataFile( "Gaussian.dat", ios::in );
  if( !dataFile )
  {
    cout<<"\n Data file not found! \n"<<endl;
    return 0;
  }
  else
  {
    /*
    Step 1 - get the polynomial from the data file
    Format
    ======
    input file format:
        LINE 1:   N                              intger, the degree of the system, maximum 10
        LINE 2:   A11 A12 A13........A1n A1n+1  n+1 floats separated by one space
        LINE 3:   A21 A22 A23........A2n A2n+1  n+1 floats separated by one space
          .
          .
          .
        LINE n+1: An1 An2 An3........Ann Ann+1  n+1 floats separated by one space
    */

    float value;

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

    // Get the N+1 ceofficients
    // while( dataFile >> coefficients[index] ) { index++; }
    for( int i=0; i<N; i++ )
    {
      // read content of each line
      c = 0;
      while( c<N+1 )
      {
        dataFile >> value;
        solution0[i][c] = value;
        solution1[i][c] = value;
        c++;
      }
    }
    FLAG = true;
  }

  if( FLAG )
  {
    float t = 0.0;

    cout<<"Values with pivoting"<<endl;
```

```
    NotNaive( N );
    NaiveSolution = ReverseSub( solution1, N );
    Print( NaiveSolution, N );
    t = RMS( solution1, NaiveSolution, N );
    cout<<"RMS Error = "<<t<<endl;

    cout<<"Value withouth pivoting"<<endl;
    Naive( N );
    NotNaiveSolut = ReverseSub( solution0, N );
    Print( NotNaiveSolut, N );
    t = RMS( solution0, NotNaiveSolut, N );
    cout<<"RMS Error = "<<t<<endl;
  }
  else
  {
    cout<<"There was a problem reading in the file!"<<endl;
  }
  return( FLAG );
}

void Print( float x[], int degree )
{
  for( int i=0; i<degree; i++ )
  {
    int j = i+1;
    cout<<"X"<<j<<" = "<<x[i]<<endl;
  }
}

void NotNaive( int degree )
{
  for( int k=0; k<degree; k++ )
  {
    if( k<degree-1 )
      Pivot( k, degree );
    for( int i=k+1; i<=degree; i++ )
    {
      float m = solution1[i][k]/solution1[k][k];
      for( int j=k; j<=degree+1; j++ )
      {
        solution1[i][j] = solution1[i][j] - ( solution1[k][j] * m );
      }
    }
  }
}

void Naive( int degree )
{
  for( int k=0; k<degree; k++ )
  {
    for( int i=k+1; i<=degree; i++ )
    {
      float m = solution0[i][k]/solution0[k][k];
      for( int j=k; j<=degree+1; j++ )
      {
        solution0[i][j] = solution0[i][j] - ( solution0[i][j] * m );
      }
    }
  }
}

void Pivot( int x, int degree )
{
  float temp = 0.0;
  int   max  = x;

  for( int i=x; i<degree-1; i++ )
  {
    if( abs(solution1[i+1][x]) > abs(solution1[max][x]) )
      max = i+1;
  }

  for( int j=0; j<=degree; j++ )
  {
    temp = solution1[max][j];
    solution1[max][j] = solution1[x][j];
    solution1[x][j] = temp;
  }
}
```

```
float* ReverseSub( float x[][11], int y )
{
  static float S[10];
  for( int k=y-1; k>=0; k-- )
  {
    float sum = 0.0;
    for( int j=k+1; j<=y-1; j++ )
    {
      sum = sum + (S[j] * x[k][j]);
    }
    S[k] = (1/x[k][k]) * (x[k][y] - sum);
  }
  return S;
}

float RMS( float x[][11], float y[], int degree )
{
  float *temp = new float[degree];
  float value = 0.0;

  for( int i=0; i<degree; i++ )
  {
    for( int j=0; j<degree; j++ )
    {
      temp[i]+=x[i][j]*y[j];
    }
  }

  for( int k=0; k<degree; k++ )
  {
    value += (x[k][degree]-temp[k]) * (x[k][degree]-temp[k]);
  }

  value = value / degree;
  value = (float) sqrt(value);

  return value;
}
```