```
// Vahe Karamian - CS 301 - Project 1 - Dr. H.K. Liu
// Filename: Taylor.cpp

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

float factorial( float m );         // Function prototype
void  T1( float x, int m, int s );  // Function prototype
void  T2( float x, int m, int s );  // Function prototype
void  T3( float x, int m, int s );  // Function prototype

int main( )
{
  int s = 1;
  int n = 0;
  int m = 0;

  float x;
  float Et;
  float original = 0.0;
  float PI = (float) 3.14159;

  ifstream dataFile( "taylor.dat", ios::in );

  if( !dataFile )
  {
    cout<<"\n Data file not found! \n"<<endl;
    return 0;
  }

  // Step 1 - Get input
  while( dataFile >> x )
  {
    original = x;

    // Step 2 - Reduce input x to -pi/2 to pi/2
    // A) sin(-x) = -sin(x), if x<0 then x=|x| but save the sign!

    if(x<0)
    {
      x = x * -1;
      s = -1;
    }
    else
    {
      s = 1;
    }

    // B) sin(x) = sin(x - 2*n*pi)
    //    if x > 2*pi then x = x - n*2*pi
    //                n = int( x/(2*pi) )
    //                x = x - 2*pi*n
    //                this will make sure that 0 <= x<= 2*pi


    if( 2*PI < x )
    {
      n = (int)( x/(2*PI) );  // compute n turn into integer
      x = x - 2*n*PI;     // compute new x value
    }

    // C) if 0 <= x <= pi/2 no change
    //    if pi/2 <= x <= 3*pi/2  x = pi - x
    //    if 3*pi/2 <= x <= 2*pi  x = x - 2*pi

    if( x >= PI/2 && x <= 3*PI/2 )
    {
      x = PI - x;
    }
    else if( x >= 3*PI/2 && x <= 2*PI )
    {
      x = x - 2*PI;
    }

    // Step 3 - Determine how many m terms such that Et <= 10^(-7)
    //        using try ad error method.
    //        Et <= (x^(2*(m+1)))/(2*(m+1))! <= 10^(-7)
```

```
    while( 1 )
    {
      Et = (float)( pow( x, (float)2*m+1 ) / factorial( (float)2*m+1 ) );
      if( Et <= (float)pow( 10, -7.0) )
      {
        break;
      }
      m++;
    }

    cout<<"x: "<<original<<endl;
    cout<<"m: "<<m<<endl;
    T1( x, m, s );
    T2( x, m, s );
    T3( x, m, s );
    cout<<endl;
  }

  return 0;
}

// Return factorial of m
float factorial( float m )
{
  float z = 1.0;
  for( int i=1; i<=m; i++ )
    z = z*i;
  return z;
}

void T1( float x, int m, int s )
{
  float sin_x = 0.0;
  float error = 0.0;

  // Traylor Series in Forward Order
  for( int i=0; i<=m; i++ )
    sin_x = (float) (sin_x + pow( -1, i ) * (pow( x, 2*i+1 ) / factorial( (float) 2*i+1 ) ));

  if( sin_x - sin(x) < 0 )
    error = (float) (-1 * (sin_x - sin(x)));
  else
    error = (float) (sin_x - sin(x));

  cout<<setiosflags( ios::left )
     <<"Sin(x) = "
     <<setw(6)
     <<setiosflags( ios::scientific )
     <<setw(16)
     <<setprecision( 6 )
     <<s*sin_x
     <<setw(10)
     <<"Error: "<< error <<endl;
}

void T2( float x, int m, int s )
{
  float sin_x = 0.0;
  float error = 0.0;

  // Taylor Series in Backward Order
  for( int i=m; i>=0; i-- )
    sin_x = (float) (sin_x + pow( -1, i ) * (pow( x, 2*i+1 ) / factorial( (float) 2*i+1 ) ));

  if( sin_x - sin(x) < 0 )
    error = (float) (-1 * (sin_x - sin(x)));
  else
    error = (float) (sin_x - sin(x));

  cout<<setiosflags( ios::left )
     <<"Sin(x) = "
     <<setw(6)
     <<setiosflags( ios::scientific )
     <<setw(16)
     <<setprecision( 6 )
     <<s*sin_x
     <<setw(10)
     <<"Error: "<< error <<endl;
```

```
}

void T3( float x, int m, int s )
{
  float sin_x = 0.0;
  float v = 1.0;
  float error = 0.0;

  // Taylor Series using Hrner's Approach
  for( int i=m; i>=2; i-- )
  {
    v = (float)(1 - ( pow(x, 2) / ((2*i+1)*(2*i)) ) * v);
  }

  sin_x = (float) (x - pow(x, 3)/factorial(3) * v);

  if( sin_x - sin(x) < 0 )
    error = (float) (-1 * (sin_x - sin(x)));
  else
    error = (float) (sin_x - sin(x));

  cout<<setiosflags( ios::left )
     <<"Sin(x) = "
     <<setw(6)
     <<setiosflags( ios::scientific )
     <<setw(16)
     <<setprecision( 6 )
     <<s*sin_x
     <<setw(10)
     <<"Error: "<< error <<endl;
}
```