Instructions

1. Create a program called Lab5main.cpp that meets the requirements identified below. □

2. Submit your Lab5main.cpp source code file to the Desire2Learn dropbox for this assignment or to the *turnin* directory.□

3. In addition to your program file, please submit a PDF file with a short summary of the results of your timing tests to the D2L dropbox.

In this assignment you will create a C++ program that performs 2 timing tests. One test will time a function that does a straight evaluation of a polynomial and the other will time a function that uses Horner's Rule (Algorithm 11.3.4 on page 750 of the Epp text) to evaluate a polynomial of the form:

$$a_n x^n + a_{n-1}x^{n-1} + \ldots + a_2 x^2 + a_1 x^1 + a_0$$

Empirical timing tests are a common way of assessing program performance. In this assignment, your program will generate 20 random polynomials (one each of degree 1 through 20), and evaluate these polynomials for a random real number x. Your program will then report the CPU time required to evaluate each polynomial.

The basic design of your program should be:

for n=1 to 20
{
Generate a random polynomial of degree n with integer coefficients between 1 and 99.

Generate a random real value x in the range from -20.00 to 20.00□

Display the polynomial and the x value.

Evaluate the polynomial using Horner's algorithm.

Report the time required to do this.

Print the result of the evaluation, P(x).

Evaluate the polynomial the usual way.

Report the time required to do this.

Print the result of the evaluation, P(x).

}

There should be a function that implements Horner's algorithm and another function to do a straight evaluation. For the straight evaluation, do not use any functions that evaluate exponents. Instead do repeated multiplies. In other words x^4 would be evaluated by implementing $x \cdot x \cdot x \cdot x$.

Set up the timing to start when you start the evaluations inside the functions. Don't time the part where you generate the polynomials. Wait until you call the functions.

See the example output below.

Example

Example from running the program on my laptop with degree limited to 4:

```
> main.exe
```

```
degree 1
 71x^1 + 55x^0
x = -1.85714
Horner 2.65e-07seconds.
P(x)= -76.8571
Not Horner 1.5e-07seconds.
P(x)= -76.8571

degree 2
 20x^2 + 94x^1 + 2x^0
x = -1.71429
Horner 1.13e-07seconds.
P(x)= -100.367
Not Horner 1.82e-07seconds.
P(x)= -100.367

degree 3
 16x^3 + 62x^2 + 72x^1 + 2x^0
x = -1.57143
Horner 1.12e-07seconds.
P(x)= -20.1283
Not Horner 2.1e-07seconds.
P(x)= -20.1283

degree 4
 68x^4 + 11x^3 + 40x^2 + 87x^1 + 23x^0
x = -1.42857
Horner 1.17e-07seconds.
P(x)= 231.492
Not Horner 2.3e-07seconds.
P(x)= 231.492
```

Representing Polynomials in C++

Polynomials of the form $a_n x^n + a_{n-1} x^{n-1} + ... + a_2 x^2 + a_1 x^1 + a_0$

can be represented as lists (arrays or vectors). The coefficients are int or long int data types, or mpz class objects.

In the example program above, polynomials were stored in an array. The subscript identified the coefficient, i.e. coeff[0] represented a_0, coeff[1] represented a_1, and so forth.

Random Numbers: The <random> Header

The C++ 11 standard greatly enhanced the ability to generate random numbers with the creation of the <random> header library. The steps needed to create a sequence of random numbers is:

1. Create a random device object

2. Create a random engine object from the random device

3. Create a distribution of random numbers

A sequence of random number is then extracted from the distribution.

For this assignment, a normal distribution with a mean of 50 is sufficient, although you may use other distributions and means if desired. (See lab 5 notes for more information on this.) The code to generate the normal sequence is:

```
random_device rd; // Random device

mt19937 engine(rd()); // Random engine

normal_distribution<> dist(50,10); //Normal distribution

// Generate 10 random numbers

for(int n = 0; n < 10; n++) {

   cout << round(dist(engine));
}
```

Date and Time: The <chrono> Header

The C++ 11 <chrono> header is used to access the system clocks in C++. The steps needed to calculate intervals is:

1. Create two time point<system clock> objects to store the begin and end clock times. ☐
2. Execute a system clock::now() command and store the result in one of the time points ☐
3. Execute your test code ☐
4. Execute a second system clock::now() command and store the result in the other time point ☐
5. Create a duration<long double> object to hold the difference between the time points ☐

This is a program that uses the code to time how long it take to print a bunch of stars :

```
// steady_clock example
#include <iostream>
#include <ctime>
#include <ratio>
#include <chrono>

int main ()
{
 using namespace std::chrono;

 steady_clock::time_point t1 = steady_clock::now();

 std::cout << "printing out 1000 stars...\n";    //time wasting loop
 for (int i=0; i<1000; ++i) std::cout << "*";
 std::cout << std::endl;

 steady_clock::time_point t2 = steady_clock::now();

 duration<double> time_span = duration_cast<duration<double>>(t2 - t1);

 std::cout << "It took me " << time_span.count() << " seconds.";
```

```
 std::cout << std::endl;

 return 0;
}
```