

The simplest polynomial implementation

☒

```
/*
A Programmer's Introduction to C# (Second Edition)
by Eric Gunnerson
```

```
Publisher: Apress L.P.
ISBN: 1-893115-62-3
*/
```

```
//Compile:
//csc /debug- /o+ /out:polynomial.exe /r:system.dll Counter.cs
Driver.cs PolySimple.cs polynomial.cs ipoly.cs
```

```
//File:PolySimple.cs
```

```
namespace Polynomial
{
using System;
///

/// The simplest polynomial implementation
///
///
/// This implementation loops through the coefficients and
evaluates each
/// term of the polynomial.
///
class PolySimple: Polynomial
{
public PolySimple(params double[] coefficients):
base(coefficients)
{ }
}
```

```
public override double Evaluate(double value)
{
double retval = coefficients[0];

double f = value;

for (int i = 1; i < coefficients.Length; i++) { retval += coefficients[i] * f; f *= value; } return(retval); } } }
//File:Polynomial.cs namespace Polynomial { using System;
using PolyInterface; /**
/// The abstract class all implementations inherit from
///

public abstract class Polynomial
{
public Polynomial(params double[] coefficients)
{
this.coefficients = new double[coefficients.Length];

for (int i = 0; i < coefficients.Length; i++)
this.coefficients[i] = coefficients[i]; } public abstract
double Evaluate(double value); protected double[] coefficients
= null; } } //File:IPoly.cs namespace PolyInterface { /**
/// The interface that implementations will implement
///

public interface IPolynomial
{
double Eval(double value);
}

//File:Driver.cs
namespace Polynomial
{
using System;
using System.Diagnostics;

///
```

```
/// Driver class for the project
///
public class Driver
{
///
/// Times the evaluation of a polynomial
///
/// The polynomial to evaluate public static double
TimeEvaluate(Polynomial p)
{
double value = 2.0;

Console.WriteLine("{0}", p.GetType().Name);

// Time the first iteration. This one is done
// separately so that we can figure out the startup
// overhead separately...
long start = Counter.Value;

p.Evaluate(0.0); // do the first iteration.
long delta = Counter.Value - start;
Console.WriteLine("Overhead = {0:f2} seconds", (double)
delta/Counter.Frequency);
Console.WriteLine("Eval({0}) = {1}", value,
p.Evaluate(value));

int limit = 100000;
start = Counter.Value;

// Evaluate the polynomial the required number of
// times.
double result = 0;
for (int i = 0; i < limit; i++) { result += p.Evaluate(value);
} delta = Counter.Value - start; double ips = (double) limit *
((double)Counter.Frequency / (double) delta);
Console.WriteLine("Evaluations/Second = {0:f0}", ips);
Console.WriteLine(); return(ips); } ///
/// Run all implementations for a given set of coefficients
```

```
///  
  
///    public static void Eval(double[] coeff)  
{  
Polynomial[] imps = new Polynomial []  
{  
new PolySimple(coeff),  
};  
  
double[] results = new double[imps.Length];  
for (int index = 0; index < imps.Length; index++) {  
results[index] = TimeEvaluate(imps[index]); }  
Console.WriteLine("Results for length = {0}", coeff.Length);  
for (int index = 0; index < imps.Length; index++) {  
Console.WriteLine("{0} = {1:f0}", imps[index],  
results[index]); } Console.WriteLine(); } ///  
/// Main function.  
///  
  
public static void Main()  
{  
Eval(new Double[] {5.5});  
  
// Evaluate the first polynomial, with 7 elements  
double[] coeff =  
new double[] {5.5, 7.0, 15, 30, 500, 100, 1};  
  
Eval(coeff);  
  
// Evaluate the second polynomial, with 50 elements  
coeff = new double[50];  
for (int index = 0; index < 50; index++) { coeff[index] =  
index; } Eval(coeff); } } //File:Counter.cs using System;  
namespace Polynomial { class Counter { public static long  
Frequency { get { long freq = 0; QueryPerformanceFrequency(ref  
freq); return freq; } } public static long Value { get { long  
count = 0; QueryPerformanceCounter(ref count); return count; } }  
[System.Runtime.InteropServices.DllImport("KERNEL32")]  
private static extern bool QueryPerformanceCounter( ref long
```

```
lpPerformanceCount);  
[System.Runtime.InteropServices.DllImport("KERNEL32")] private  
static extern bool QueryPerformanceFrequency( ref long  
lpFrequency); } } Polynomial.zip( 2 k)[/csharp]
```