Deakin University

Object Oriented Development

OnTrack Submission

The MyPolynomial class

Submitted By:
Peter Stacey
pstacey
2020/03/31 11:02

Tutor:
Dipto Pratyaksa

Outcome	Weight
Evaluate Code	♦♦♦◊◊
Principles	♦♦♦◊◊
Build Programs	♦♦♦♦◊
Design	♦♦♦◊◊
Justify	♦♦♦◊◊

This task is far more complex than all earlier tasks, requiring close attention to the use of correct conventions to be able to read and understand the code, significant time (2-3 hours) to design the solution before writing any code, the research of additional concepts to translate them into code (eg. adding and multiplying polynomials), and the addition of comments that add explanation for some coding choices. Overall, a really challenging and good task

March 31, 2020

```
using System;

namespace Task_3._3C
{
    class TestMyPolynomial
    {
        public static String CoeffsToString(MyPolynomial poly)
        {
            String result = "[";
            for (int i = 0; i < poly.GetDegree(); i++)
            {
                result += poly.CoeffAt(i) + ", ";
            }
            result += poly.CoeffAt(poly.GetDegree()) + "]:  ";
            return result;
        }

        public static MyPolynomial RandomPolynomial(Random rnd, int min, int max)
        {
            double[] coeffs = new double[rnd.Next(1, max)];
            int num = 0;
            for (int j = 0; j < coeffs.Length; j++)
            {
                num = rnd.Next(min, max);
                if (num % 7 == 0)
                    num = 0;
                coeffs[j] = Convert.ToDouble(num);
            }
            if (coeffs[coeffs.Length - 1] == 0)
                coeffs[coeffs.Length - 1] = 7;
            return new MyPolynomial(coeffs);
        }
        public static void Main(string[] args)
        {
            int numberOfTests = 50;

            // Test ToString by creating 100 random sets of arrays and
            // creating polynomials for each one and printing via
            // ToString
            Console.WriteLine("\n\nTesting ToString:\n");
            Random rnd = new Random();

            for (int i = 0; i < numberOfTests; i++)
            {
                MyPolynomial poly = RandomPolynomial(rnd, -20, 20);
                Console.WriteLine("\n" + CoeffsToString(poly));
                Console.WriteLine(poly.ToString());
            }

            // Test evaluate
            Console.WriteLine("\n\nTesting evaluating a polynomial");

            for (int i = 0; i < numberOfTests; i++)
```

```
54              {
55
56                  double x = rnd.Next(-10, 10);
57                  MyPolynomial poly2 = RandomPolynomial(rnd, -10, 10);
58                  double result = poly2.Evaluate(x);
59                  Console.WriteLine("\n({0}), {1} = {2}", poly2.ToString(), x,
                    ↪  result);
60              }
61
62              // Test adding polynomials
63              Console.WriteLine("\n\nTesting adding polynomials:\n");
64
65              for (int i = 0; i < numberOfTests; i++)
66              {
67                  MyPolynomial poly5 = RandomPolynomial(rnd, -5, 5);
68                  MyPolynomial poly6 = RandomPolynomial(rnd, -5, 5);
69                  Console.Write("\n({0}) + ({1}) = ", poly5.ToString(),
                    ↪  poly6.ToString());
70                  poly5.Add(poly6);
71                  Console.WriteLine(poly5.ToString());
72              }
73
74              // Test multiplying polynomials
75              Console.WriteLine("\n\nTesting multiplying polynomials:\n");
76
77              for (int i = 0; i < numberOfTests; i++)
78              {
79                  MyPolynomial poly5 = RandomPolynomial(rnd, -5, 5);
80                  MyPolynomial poly6 = RandomPolynomial(rnd, -5, 5);
81                  Console.Write("\n({0}) x ({1}) = ", poly5.ToString(),
                    ↪  poly6.ToString());
82                  poly5.Multiply(poly6);
83                  Console.WriteLine(poly5.ToString());
84              }
85          }
86      }
87  }
```

```cs
using System;

namespace Task_3._3C
{
    class MyPolynomial
    {
        // Instance variables
        private double[] _coeffs;

        /// <summary>
        /// Constructor for a polynomial
        /// </summary>
        /// <param name="coeffs">Double array of coefficients</summary>
        public MyPolynomial(double[] coeffs)
        {
            _coeffs = coeffs;
        }

        /// <summary>
        /// Returns the degree of the polynomial
        /// </summary>
        /// <returns>
        /// The degree of the polynomial as an integer
        /// </returns>
        public int GetDegree()
        {
            return _coeffs.Length - 1;
        }

        /// <summary>
        /// Returns a common string representation of a polynomial
        /// </summary>
        /// <returns>
        /// String representation of an expanded polynomial
        /// </returns>
        public override String ToString()
        {
            String result = "";
            String op, exp;
            double num;
            for (int i = _coeffs.Length - 1; i >= 0; i--)
            {
                num = _coeffs[i];
                // If num is less than 0, add " - " to the string, otherwise
                // add " + " as long as this isn't the start of the string
                // Example:
                // - 4x^2 + 2x - 2 (print minus sign at start)
                // 4x^2 + 2x - 2 (don't print plus sign at start)
                op = num < 0 ? "- " : (i < _coeffs.Length - 1 && num > 0) ? "+ " :
                ↪  "";
                num = Math.Abs(num);
                exp = (i == 0 && num != 0) ? num.ToString()             // If
                ↪  constant and not 0, then concatenate to string
```

```
52                      : (i == 1 && num == 1) ? "x "                           // if 1x^1,
                        ↪ don't print 1 or ^1, just x
53                      : (i == 1 && num != 0) ? num.ToString() + "x "          // if ax^1,
                        ↪ don't print ^1, just ax (a not 0)
54                      : (i > 1 && num == 1) ? $"x^{i} "                        // don't
                        ↪ print 1 before any number, but include coefficient if > 1
55                      : (i > 1 && num != 0) ? num.ToString() + $"x^{i} "  //
                        ↪ otherwise if not 0, print ax^coeff
56                      : "";                                                   // if 0,
                        ↪ add nothing to the string
57                  result += (op + exp);                                       // add the
                    ↪ operator and the exponent part
58              }
59              return result;
60          }
61
62          /// <summary>
63          /// Calculates the value of f(x) for a given value of x
64          /// </summary>
65          /// <returns>
66          /// Double value of substituting x into the polynomial
67          /// </returns>
68          public double Evaluate(double x)
69          {
70              double result = _coeffs[0];
71              for (int i = 1; i < _coeffs.Length; i++)
72              {
73                  result += _coeffs[i] * Math.Pow(x, i);
74              }
75              return result;
76          }
77
78          /// <summary>
79          /// Returns the value of the coefficent at the given index position
80          /// </summary>
81          /// <returns>
82          /// Double value of the coefficient at the given index
83          /// </returns>
84          /// <exception cref="System.IndexOutOfRangeException">Thrown when index
85          /// is larger than the size of the coefficients array</exception>
86          public double CoeffAt(int index)
87          {
88              try
89              {
90                  return _coeffs[index];
91              }
92              catch (IndexOutOfRangeException)
93              {
94                  throw new IndexOutOfRangeException(
95                          "Index " + index
96                          + " not valid for a polynomial of length "
97                          + _coeffs.Length);
98              }
```

```
99          }
100
101         /// <summary>
102         /// Adds another polynomial to this
103         /// </summary>
104         /// <returns>
105         /// this polynomial
106         /// </returns>
107         /// <param name="other">The polynomial to add</param>
108         public MyPolynomial Add(MyPolynomial other)
109         {
110             double[] result;
111             bool longer = this.GetDegree() >= other.GetDegree();
112             if (longer)
113                 result = new double[_coeffs.Length];
114             else
115                 result = new double[other.GetDegree() + 1];
116             for (int i = 0; i < result.Length; i++)
117             {
118                 try
119                 {
120                     result[i] = _coeffs[i] += other.CoeffAt(i);
121                 }
122                 catch (IndexOutOfRangeException)
123                 {
124                     if (longer)
125                         result[i] = _coeffs[i];
126                     else
127                         result[i] = other.CoeffAt(i);
128                 }
129             }
130             _coeffs = result;
131             return this;
132         }
133
134         /// <summary>
135         /// Multiplies this by another polynomial
136         /// </summary>
137         /// <returns>
138         /// this polynomial
139         /// </returns>
140         /// <param name="other">The polynomial to multiply by</param>
141         public MyPolynomial Multiply(MyPolynomial other)
142         {
143             double[] product = new double[_coeffs.Length + other.GetDegree()];
144             for (int i = 0; i < _coeffs.Length; i++)
145             {
146                 for (int j = 0; j < other.GetDegree() + 1; j++)
147                 {
148                     product[i + j] += _coeffs[i] * other.CoeffAt(j);
149                 }
150             }
151             _coeffs = product;
```

```
152                 return this;
153             }
154         }
155 }
```