Computer Science 2 — CSCI-142
Polynomials — Lab 1

5/12/2020

1 Introduction

We will use an `Native Array` to represent a single variable polynomial, whose coefficients are integers. The index of a coefficient in the native array indicates the power of the term. The first coefficient in the native array is a constant term (power of 0), the second coefficient in the native array is a term with a power of 1, and so on. If there is no term of a particular power, less than the degree of the polynomial, the native array will store a coefficient value of 0.

Here are a few examples of polynomials in `native array` form, followed by their regular notation:

Native Array	**Polynomial**
`[1]`	1
`[3, -1]`	$-x + 3$
`[0, 3]`	$3x$
`[2, -1, -2, 1]`	$x^3 - 2x^2 - x + 2$
`[-5, 0, 0, 3, 3, 1]`	$x^5 + 3x^4 + 3x^3 - 5$

2 Problem Solving

Work in a team of three to four students as determined by the instructor.

1. Show the polynomial for the following `native array`: `[5, -3, 0, 2, 1]`.

2. Show the `native array` for the following polynomial: $-4x^5 + 2x^4 - x^2 + 9x$.

3. A program was run with a polynomial specified on the *command line* as:

   ```
   -1 2 0 3
   ```

   Because of the `String` array parameter in the `main` method, this is equivalent to creating a *native array* of strings in code as:

   ```
   String[] args = {"-1", "2", "0", "3"};
   ```

   Given an initially empty `native array` of integers whose declaration should begin as follows,

   ```
   int[] poly = . . .
   ```

   Write a code snippet that completes the above initialization of `poly`, loops over the elements in `args`, the native array of `String`s, converts them, and adds them to `poly`.

4. Write a public static method, `evaluate`, that evaluates a polynomial in an `native array` of integers for a certain integer value for `x`. For example, each call below should give the following result:

```
evaluate(poly=[1], x=0) = 1
evaluate(poly=[3, -1], x=5) = -2
evaluate(poly=[-5, 0, 0, 3, 3, 1], x=2) = 99
```

5. Write a public static method, `computeDerivative`, that takes a polynomial, and computes and returns a new polynomial that is its *derivative*. For example, each call below should give the following result:

```
computeDerivative(poly=[1]) = [0]
computeDerivative(poly=[0, 3]) = [3]
computeDerivative(poly=[-5, 0, 0, 3, 3, 1]) = [0, 0, 9, 12, 5]
```