CSCI 310 – 02 (Fall 2019)

Programming Foundations

Programming Assignment 4

DUE: Sun, Oct 20, 11:59 PM (`turnin` time)

Overview

In this programming assignment, we return to our `Polynomial` class and use it to gain experience with the C++ concept of *operator overloading*.

Specifications

You will complete the definition for a `Polynomial` class based on the following incomplete `Polynomial.h` header file:

```
class Polynomial
{
  private:

    //YOUR CODE GOES IN HERE!!!

  public:

  //CONSTRUCTOR(S)//////////////////////////////////////////////////////////////

  /** Creates a zero polynomial.
      @post  Polynomial is set to the zero polynomial. */
      Polynomial();
  /** Creates a constant polynomial.
      @post  Polynomial is set to the constant c. */
      Polynomial(double c);
  /** Copy constructor.
      @post  Polynomial is set to the other polynomial. */
      Polynomial(const Polynomial& other);

  //DESTRUCTOR(S)///////////////////////////////////////////////////////////////

  /** Destructor
      @post  All dynamic memory allocated for the object is released. */
      ~Polynomial();

  //MODIFICATION MEMBER FUNCTIONS///////////////////////////////////////////////

  /** Resets the polynomial to the zero polynomial.
      @post  Polynomial is set to the zero polynomial. */
      void clear();
  /** Sets a term in the polynomial.
      @pre   exponent >= 0
      @post  If successful, a new term is added to the polynomial
          and the degree of the polynomial is updated, if needed.
          If a term with the given exponent already exists, its
          coefficient is replaced by the coefficient parameter.
      @param coefficient The coefficient of the new term.
      @param exponent The exponent of the new term.
      @return  True if the set was successful, or false if not. */
      bool setTerm(unsigned exponent, double coefficient);
  /** Adds to a term in the polynomial.
      @post  If successful, the term with the matching exponent in
          the polynomial will have the coefficient parameter value
          added to its coefficient value. If no such term exists,
```

```
          then this operation is just like setTerm().
      @param coefficient The value to add to the coefficient of
          the term with the given exponent.
      @param exponent The exponent of the term to update.
      @return  True if addition was successful, or false if not. */
      bool addTerm(unsigned exponent, double coefficient);

  //CONSTANT MEMBER FUNCTIONS////////////////////////////////////////////////////

  /** Determines if this is the zero polynomial.
      @return true if polynomial is the constant zero. */
      bool isZero() const;
  /** Determines the degree of the polynomial.
      @return Degree of the polynomial. */
      unsigned getDegree() const;
  /** Determines the coefficient of the term with the
      given exponent.
      @param exponent The exponent of a term.
      @pre  exponent >= 0
      @post Returns the coefficient at the specified
            exponent of this Polynomial */
      double coefficient(unsigned exponent) const;
  /** Evaluates the polynomial based on a given value for x.
      @param x The value to evaluate the polynomial with.
      @return The value f(x) of the polynomial. */
      double evaluate(double x) const;
  /** Returns the composition of this polynomial where
      every occurrence of x is substituted with the
      other polynomial.
      @param other The other polynomial to substitute for x.
      @return The polynomial f(g) where g is the other polynomial. */
      Polynomial sub(const Polynomial& other) const;
};
```

In addition to the above functionality, you also need to overload the following operators where `f` is declared to be a `Polynomial` object:

1. `f(x)` where `x` is a `double`, evaluates the polynomial `f` with the given `x` value; this is basically `evaluate()`.
2. `f(g)` where `g` is another `Polynomial` object, returns the `Polynomial` object representing the polynomial composition of `f` and `g` where `g` is substituted for every occurrence of `x` in `f`.
3. `-f` returns the negative of `f`.
4. `+f` returns `f`.
5. `f=g` where `g` is another `Polynomial` object, assigns `g` into `f`.
6. `f+=g` where `g` is another `Polynomial` object, adds `g` into `f`.
7. `f-=g` where `g` is another `Polynomial` object, subtracts `g` from `f`.
8. `f*=g` where `g` is another `Polynomial` object, multiplies `g` into `f`.
9. `f^=n` where `n` is an `unsigned` value, raises `f` to the `n`th power.
10. `f+g` where `g` is another `Polynomial` object, returns the sum of `f` and `g`.
11. `f-g` where `g` is another `Polynomial` object, returns the difference of `f` and `g`.
12. `f*g` where `g` is another `Polynomial` object,
13. `f^n` where `n` is an `unsigned` value, returns the value of `f` raised to the `n`th power.
14. `outs << f` where `outs` is an `ostream` object, "displays" `f` in the traditional way with terms of decreasing exponents (see sample output below).

Note that you will have to determine various aspects of the class implementation on your own. You are allowed to use whatever built–in C++ container class you determine will best help you manage the terms in a polynomial. You also need to determine how you will implement all the overloaded operators — as a member function or as a non–member function.

Input

Your program lines of text that are in groups of three. Each group of three lines represent one test case. The first two lines will contain a list of $n+1$ integers $a_n, a_{n-1}, \ldots, a_1, a_0$ which represent a set of coefficients of a polynomial of degree n. The coefficients are paired with the terms of the polynomial in the following manner:

$$a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x^1 + a_0 x^0$$

The third line of text are the $m+1$ values $x_0, x_1, \ldots, x_m$ to be used for x when evaluating the first polynomial provided.

Input will be redirected from standard input via redirection on the *nix command prompt. As in the previous assignment, feel free to use the `split()` function provided at: `http://www.ecst.csuchico.edu/~bjuliano/csci310/Code/split.cpp`.

Sample Input

```
-2
1 -1
5 0 1 6
7 6 -1
-1 1
7 6 -1
3 0 0 -4 0 2
-5 0 3 -11 4
1 2 3 4 5
```

How many test cases are provided in the input above? What are two polynomials given for each test case? How many x values are given for each test case?

Output

For each test case, your program must display the following:

1. the test case number (starts at `1`) in the form "`TEST CASE #`n" where n is the test case number;
2. the first polynomial in the test case and denote this as `f`;
3. the second polynomial in the test case and denote this as `g`;
4. evaluate `f` for all the $m+1$ given values of x (x_0 through x_m) in the third line of input;
5. the polynomial `f+g`;
6. the polynomial `f-g`;
7. the polynomial `f*g`;
8. the polynomial `-g`;
9. the polynomial `f^2`;
10. the polynomial `g^3`;
11. the polynomial `f^0`;
12. the polynomial `f(g)`; and
13. the value of `g(f(-1))`.

Refer to the sample output below for appropriate prompts/labels for each output.

Output should be sent to standard output and must exactly follow the format in the sample below.

Sample Output

```
TEST CASE #1
f: -2
g: x - 1
-2 -2 -2 -2
f+g: x - 3
f-g: -x - 1
f*g: -2x + 2
 -g: -x + 1
f^2: 4
g^3: x^3 - 3x^2 + 3x - 1
f^0: 1
f(g): -2
g(f(-1)): -3

TEST CASE #2
f: 7x^2 + 6x - 1
g: -x + 1
384 287 0
f+g: 7x^2 + 5x
f-g: 7x^2 + 7x - 2
f*g: -7x^3 + x^2 + 7x - 1
 -g: x - 1
f^2: 49x^4 + 84x^3 + 22x^2 - 12x + 1
g^3: -x^3 + 3x^2 - 3x + 1
f^0: 1
f(g): 7x^2 - 20x + 12
g(f(-1)): 1

TEST CASE #3
f: 3x^5 - 4x^2 + 2
g: -5x^4 + 3x^2 - 11x + 4
1 82 695 3010 9277
f+g: 3x^5 - 5x^4 - x^2 - 11x + 6
f-g: 3x^5 + 5x^4 - 7x^2 + 11x - 2
f*g: -15x^9 + 9x^7 - 13x^6 + 12x^5 - 22x^4 + 44x^3 - 10x^2 - 22x + 8
 -g: 5x^4 - 3x^2 + 11x - 4
f^2: 9x^10 - 24x^7 + 12x^5 + 16x^4 - 16x^2 + 4
g^3: -125x^12 + 225x^10 - 825x^9 + 165x^8 + 990x^7 - 2148x^6 + 1023x^5 + 957x^4 - 2123x^3 + 1596x^2
- 528x + 64
f^0: 1
f(g): -9375x^20 + 28125x^18 - 103125x^17 + 3750x^16 + 247500x^15 - 523500x^14 + 107250x^13 + 831675x^12
- 1.50315e+06x^11 + 675279x^10 + 1.14494e+06x^9 - 2.3189e+06x^8 + 1.64175e+06x^7 + 135165x^6
- 1.37327e+06x^5 + 1.39938e+06x^4 - 765336x^3 + 243260x^2 - 41888x + 3010
g(f(-1)): -2991
```

Additional Requirements

Although this project does not require you use all the overloaded operators identified above, for full credit you are expected to implement all overloaded operators specified. Note that some operators can be defined in terms of other operators, so carefully plan your implementation before writing code to avoid duplication of effort.

Deliverables

Your submission will consist of the following files, submitted using the Department of Computer Science's `turnin` facility:

- `Polynomial.h` – specification/header file for `Polynomial` class
- `Polynomial.cpp` – implementation file for `Polynomial` class
- `TestPoly.cpp` – driver code containing `main()` function

The following file(s) will be available in `turnin`:

- `split.cpp` containing the code for the `split()` function

We want you to develop good code documentation habits. Source code solutions submitted without any meaningful documentation will receive a total score of zero (0). You may refer to the *Google C++ Style Guide* section on source code comments as a guide.

Be sure to also review and adhere to the **Coding Standards** for this course.