Data Structures — Homework 5 — Polynomial Class

Overview

This assignment is due Tuesday, March 29 at 11:59:59pm and is worth 75 points toward your homework grade. Up to 5 points of extra credit may be earned, as discussed below.

Your problem is to implement a Polynomial class. This class should be able to represent, evaluate, add, subtract, multiply and, perhaps, factor polynomials. The polynomials should have integer coefficients and non-negative integer exponents. For example,

\[ f(x) = 5x^{13} - 3x^6 + 4x - 1 \]

is a degree-13 polynomial, with coefficients 5, -3, 4 and -1, and with exponents 13, 6, 1 and 0. The variable \( x \) can be any real number. It is assumed that you know enough about polynomials to complete the assignment.

A main program has been provided for you on the course web site. We will test with this main program and with an additional one that we have not provided, so be careful with your implementation! Use our main program to figure out the exact format of the function calls described below.

You must use a \texttt{std::list} to store the information associated with the polynomial inside your \texttt{Polynomial} class, and you must only store terms with non-zero coefficients.

Operations Required

Your class should have the following operations:

- Construct an empty polynomial.
- Construct a polynomial from two equal-length vectors of integers. The first vector specifies the non-zero coefficients and the second vector specifies the associated exponents. If the two vectors are of different lengths, only form the terms up to the shorter length. For example, if the input coefficient vector contains 5, -3, 4 and 1 and the input exponential vector contains 13, 6 and 0, then the resulting polynomial should be
  \[ f(x) = 5x^{13} - 3x^6 + 4x \]

  Any terms with negative exponents or coefficient values of 0 should be ignored. Finally, if the same exponent appears more than once, the coefficient associated with the last appearance of the exponent should be used. You may not assume the input exponent values are ordered, even though the above example is (and this is the typical case).
- A copy constructor.
- \texttt{degree}: Provide the degree of the polynomial. This is the value of the highest exponent with non-zero coefficient.
- \texttt{coefficient}: Provide the coefficient associated with the given exponent.
- \texttt{set_coefficient}: Set the coefficient of a term in the \texttt{Polynomial}. If the coefficient is 0 then the term should not be represented in the polynomial after the function call is made. If the exponent provided is negative, do nothing.
• **add**: Add two polynomials to create a third polynomial.

• **subtract**: Subtract two polynomials to create a third polynomial.

• **multiply**: Multiply two polynomials to create a third polynomial.

• **eval**: Evaluate a polynomial given a particular value of \( x \), returning a real number.

• **write**: Write the polynomial to an output stream. The output should all be on one line, with terms in decreasing order of exponent. See the example posted on-line for the format.

• **integer_factors**: (5 points extra credit) Return a list of all integers that are factors of the polynomial. Recall that \( n \) is a factor of polynomial \( p \) if \( p(n) = 0 \). Your function should run in time no worse than \( O(m\sqrt{n}) \), where \( m \) is the number of non-zero coefficients in the polynomial and \( n \) is the coefficient of the term with lowest degree exponent. The `main.cpp` function we provided has calls to this function, but they are commented out.

In all cases if the coefficient of a term becomes 0, then it should be removed from the polynomial. If all terms of a polynomial have a coefficient value of 0 then the value of the polynomial is 0.

**Notes**

• You will need to evaluate \( x^n \), where \( n \) is an integer. One simple way to do this is the `pow` function (look it up), but there are others.

• Part of your assignment grade will depend on the efficiency of your implementations. For example, if there are \( N \) terms with non-zero coefficients in one polynomial and \( M \) terms with non-zero coefficients in the second, then the time required for the `add` operation should be at most \( O(M + N) \). Unfortunately, the multiplication operation can not be performed in better than \( O(MN) \) time and your algorithm is likely to be slower than this.

• You will need to submit four files: `main.cpp`, `Polynomial.h`, `Polynomial.cpp` and `readme.txt`. The `main.cpp` function should be the one we provided, perhaps modified to comment out functions (such as multiplication) that do not work and to uncomment functions like `integer_factors` that do.