CS2810 OOAIA: A3

Design Deadline: January 27 at 16:40 on Moodle
Code Deadline: January 29 at 23:45 on Hackerrank

Contest Link

https://www.hackerrank.com/cs2810-a3

Objective

- To learn about polymorphism by using function overloading and operator overloading in C++

Data Structure

Create a class in C++ to implement `Polynomials`. This class should support functionality to add, subtract, and multiply using both functions as well as by overloading these operators: `+, -, *`. The operands for these operations can be two `Polynomials` or a `Polynomial` and a `double`. Also include the functionality to evaluate the result of a `Polynomial` at a given `x`.

The internal details of the implementation of the class are your choice.

Input Format

Each testcase will consist of `n` operations. Each operation is either an addition (`'a'`), subtraction (`'s'`), multiplication (`'m'`), or evaluation (`'e'`). The first three operations take two `Polynomials` as input and output a `Polynomial`. These operations can also take a `Polynomial` and a `double` as input, however the `double` will also be given in the polynomial format with one constant term. Polynomial evaluation takes a `Polynomial` and a `double` as input and outputs a `double`.

The testcase format is as follows:

```
<number of operations>                    // → "n"
<inputs for op_1>
<inputs for op_2>
.
.
.
<inputs for op_n>
```

A Polynomial is represented in the following form in the testcase:

<number of terms> // → "m"
<exponent_1> <coefficient_1>
<exponent_2> <coefficient_2>
.
.
.
<exponent_m> <coefficient_m>

The exponent will be a non-negative integer. The coefficient will be a `double`. Note that the exponents can be in *any* order.

Output Format

The output for addition, subtraction, multiplication is a `Polynomial`, while for evaluation it is a `double`.

`Polynomial`:

- Each term of the Polynomial has to be printed in the format: `<coefficient>`x^`<exponent>`
- The terms of the `Polynomial` have to be printed in increasing order of their exponents.
- If there exists a constant term, print that with "x^0".
- The format for the whole Polynomial is `<term><space><sign><space><term><space><sign><space><term>`...
- The sign will be '+' or '-' depending on the sign of the coefficient following it.
- Only `terms` with non-zero coefficients have to be printed.
- All coefficients should be printed with a fixed precision of 3 decimal digits (see note at the end).
- An "empty" `Polynomial` should be printed as a blank line

Incorrect way to print Polynomial	Corresponding correct representation
6x^4 + 3x^2	3.000x^2 + 6.000x^4
5.00 - 10.0x^3	5.000x^0 - 10.000x^3
5.000 + -6x^2	5.000x^0 - 6.000x^2
4x + 0x^2	4.000x^1

`Double`:

- Each `double` has to be printed with a fixed precision of 3 decimal digits (see note at the the end).

Constraints

- There will be a maximum of 100 operations per testcase.
- The maximum degree of any `Polynomial` taken as input will be 10.

Sample Testcase

Input:

```
2       → Number of operations
a       → Addition (first operation)
2       → Number of terms in the first polynomial
1 2     → 2x^1
3 4     → 4x^3
3       → Number of terms in the next (second) polynomial
5 -10   → -10x^5
2 7     → 7x^2
3 -6    → -6x^3
e       → Evaluation (second operation)
3       → Number of terms in the polynomial
1 2     → 2x^1
3 4     → 4x^3
5 6     → 6x^5
10      → Value of x at which polynomial has to be evaluated
```

Output:

```
2.000x^1 + 7.000x^2 - 2.000x^3 - 10.000x^5
604020.000
```

Design Submission Format

For the design submission on Moodle, please submit a .tar.gz file named as your roll number.

Note

All `doubles` have to be printed with a fixed precision of 3 decimal digits. This includes the coefficients of the `Polynomials` as well the result of the evaluation operation. Take a look at the example below.

Number	How it should be displayed
2	2.000
3.4	3.400
3.14159	3.142

This style of printing can be set by using the following statements before any “cout”. You only need to write these statements once.

```
std::cout.precision(3);
std::cout << std::fixed;
```

Fun Challenge (not evaluated)

Extend the polynomial to support complex numbers.