ECE 2574 ♦ Introduction to Data Structures and Algorithms
Homework 2

Overview: Polynomials are frequently encountered representations in engineering problem solving. For this assignment, you must write a C++ program that implements a set of polynomial operations, including addition, multiplication, and retrieving from a disk file, through the development of a pointer-based linked list.

Objective: The main goal of this assignment is to gain experience in using pointers, using linked lists, and their implementation. Additional goals include learning how to use file I/O and command-line arguments.

Program specifications: Define and implement an ADT version of the class `Poly` that uses a linked list to contain one polynomial. Note that this is a non-circular, singly linked list. An instance of the class `Poly` should contain a *polynomial*, represented by using a linked list in which each node is a *term* of the polynomial with three fields: one for the *coefficient*, one for the *exponent*, and one for the pointer to next term. As an example, consider the polynomial

$$P(x) = 4x^5 + 7x^3 - x^2 + 9$$

It has 4 *terms,* with *coefficients* 4, 7, -1, 9, and *exponents* 5, 3, 2, 0, respectively.

Memory must be allocated dynamically for the linked list. Note that the user is allowed to select different data types using the `typedef` for `CoefficientItem`. We will consider the type `int` to be the usual case for the coefficients, but your code should also work for `float` and `double`. The coefficients may be positive or negative numbers. The exponents of the polynomials are, however, either positive integer numbers or have a zero value. You should not make any assumption on the total number of terms that the polynomials will have.

Your ADT must implement addition and multiplication of polynomials. For example, given two polynomials, $3x^2 + 5x + 3$ and $2x + 3$, their addition and multiplication may be performed as follows:

(1) Addition:

$$\begin{array}{l} \left(3x^2 + 5x + 3\right) + \\ \qquad \left(2x + 3\right) \\ \hline \left(3x^2 + 7x + 6\right) \end{array}$$

(2) Multiplication:

$$\begin{aligned}&\left(3x^2+5x+3\right)\times\left(2x+3\right)\\&=6x^3+9x^2+10x^2+15x+6x+9\\&=6x^3+19x^2+21x+9\end{aligned}$$

Your C++ program must read two polynomials from a disk file, and then perform addition and multiplication on them. The name of the disk file needs to be passed through the command-line arguments. For this assignment, you must define and implement the ADT using a *linked list* (for storing and manipulating the polynomials). You should **not** use any of the ADT classes described in the Carrano text or the C++ STL libraries. You must write your own ADTs from "scratch."

Your program should be invoked on the command-line as follows:

```
compute-poly input_file output_file
```

Here, **`compute-poly`** is the name of your executable program, and `input_file` and `output_file` are the names of the input and output disk files, respectively. These two files contain ASCII text. The program starts by reading the terms of the polynomials from the *input_file*, stores them in the linked list, and then performs the addition and multiplication operations. After that, the program outputs the resulting polynomials to the *output_file*.

The format for describing the polynomials in the input file is as follows: Consider the following input file **testfile1** that represents the two polynomials $3x^2+5x+3$ and $2x+3$:

testfile1

```
3 2
5 1
3 0
XXX
2 1
3 0
```

testfile1.out

```
3 2
7 1
6 0
YYY
6 3
19 2
21 1
9 0
```

The file **testfile1** contains a description of two polynomials. The coefficient and the exponent of each term of the polynomial are written on a single line and are separated by a single space character. A line that contains string XXX separates the coefficients and exponents of the two polynomials. No text will appear after the line containing the last polynomial term in the file.

The file **testfile1.out** should contain the resulting polynomials of addition and multiplication. The coefficient and the exponent of each term of the polynomial should be written on a single line and should be separated by a single space character The two polynomials must be separated by a line of text that contain the special string YYY. No text should appear after the line containing the last polynomial term in the file.

The input polynomials need not appear (in the input file) in decreasing order of the exponents of their terms—the terms may appear in any order of their exponents in the input file. In contrast, your output polynomials must appear (in the output file) in decreasing order of the exponents of their terms. If the coefficient of a term equals zero, this term must not appear in the output file. For each polynomial, combine terms in your linked list and in the output file, so that there are no repeated exponents. For example, if a calculation results in $2x^2 + 3x^2$, then your implementation should combine those into the single term $5x^2$.

Exception Handling: Your implementation should use `try/throw/catch` to handle the following exceptions:

1. If input/output files are not provided on the command line, or the input file does not exist.
2. Bad input file, which includes the absence of string XXX.
3. If any coefficient of the input polynomials is zero.
4. If any exponent of the input polynomials is negative.
5. If `new` returns `NULL` when allocating memory.

Naming Conventions: Your source files should be named `PolytermsP.h` and `PolytermsP.cpp`, which are the specification and the implementation of an ADT, respectively. In addition, you should also create and submit a file, `computepoly.cpp`, which contains an example "client" code that demonstrates correct operation of your ADT.

What to hand in: Upload your 3 files (`computepoly.cpp`, `PolytermsP.h`, and `PolytermsP.cpp`) to Scholar. Be careful to use these file names, to test your files before uploading them, and to upload the correct files. The files that you upload to Scholar are the files that will be graded.

Partial Credit: If you could not get your program to run, also turn in a written description (MS Word or PDF file) that describes 1) the problems you encountered, 2) what parts of your program work correctly, and 3) suggestions to the grader as to how your program may be tested for partial credit consideration. If your program runs, you do not need to turn in such a file.

Grading: The assignment will be graded out of a maximum of 100 points: 90 points depend on correct program operation, and the remaining 10 points is based on the programming style.

- In order to test your program for correct operation, the grader will compile and link your code along with his/her copies of the test case files. We will try to automate part of this process. If your code does not compile, then the grader will be tempted to assign 0 points for correct operation, and then move on to the next student's submission. It is therefore very important that you upload the final, correct versions of your files, with the correct file names.

- Programming style includes clarity, comments, indentation, error handling, etc., as described in the *Programming Style Guidelines* document.

Honor Code: You must work independently on this assignment. Please review the statement in the syllabus.