CSS 343 Data Structures, Algorithms, and Discrete Mathematics II
2017 Winter

Assignment 1, 100 possible points (7% of final grade)

Submission Due by: January 14th (Saturday) 11:59 pm

Assignment Description:

This project is to **create the abstract data type (ADT) for a polynomial called Poly**.

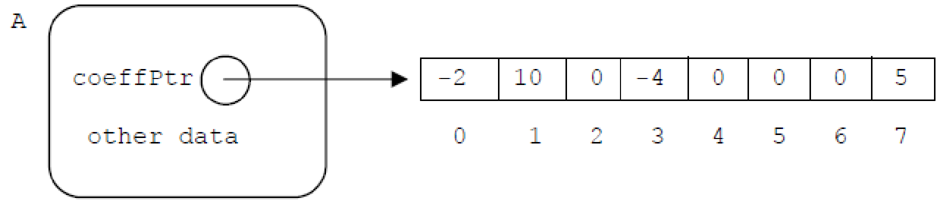
Implement the internal representation of a polynomial (nonnegative exponents) as an array of terms. One array is sufficient to store one polynomial. One term is stored in one array element; each term contains an exponent and a coefficient. The array element subscript is the exponent and the content of the element is the coefficient value. A constructor receives at most two values, the coefficient of the term and the largest exponent currently expected. (The coefficient may be changed later using functions.) Dynamically allocate initial memory for your array using only as many elements as needed. (After the Poly object instantiation, the size of the array need not be tied to the largest exponent. i.e., you may want to make the array larger when needed, but typically an array is not made smaller as it is inefficient.)

Develop a full class:

1. All proper constructor and destructor functions, e.g.,

```
Poly A(5,7);        // Memory is reserved for polynomial A, with a largest power of 7
                    // and the coefficient is set to 5: +5x^7.
Poly B(2);          // B has one element, initialized to +2x^0
Poly X;             // Initialized to 0x^0 (zero polynomial)
Poly C(A);          // C is an exact copy (deep copy) of A.
```

To make sure you understand memory, when Poly A is +5x^7 -4x^3 +10x -2 memory for the object A looks like



2. Overload common operators:

(a). The addition, subtraction, and multiplication operators (+, -, *) to add, subtract, multiply two polynomials

(b). The assignment operator (=) to assign one polynomial to another

(c). The addition, subtraction, multiplication assignment operators (+=, -=, *=)

(d). The equality and inequality operators (==, !=)

3. Accessors:

(a). Overload << to output an entire polynomial, print " +5x^3 -2x^2 +4" to represent the Poly 5 x cubed minus 2 x squared plus 4. Use the following exact formatting. Must-do rules for displaying polynomials:

- Use lowercase x

- A nonzero coefficient and its sign are always displayed (display exactly one blank before every sign)
- Do not display a new line in operator<< at the end of the polynomial
- Do not display extra blanks at the end or anywhere else in the polynomial
- A power of 1 is not displayed and x is not shown at all for a power of 0, e.g., +1x^4 +2x -10
- Don't print a term if the coefficient is zero, except if all the coefficients are zero, then print " 0"

(b). Get one term's coefficient, called **getCoeff**, e.g.,

```
int coeff = P.getCoeff(2);
cout << "coeff of power 2 of P is: " << coeff << endl;
```

4. Mutators:

(a). Overload >> to input all coeffs (sets all terms of one Poly, in a loop; it is expected that a user enters one coefficient and an exponent repeatedly, Enters -1 for both when done; no data type-checking). e.g.,

to set A, Poly A(2,7); so that A is `+5x^7 -4x^3 +10x –2`
the user enters the pairs in any order: `5 7 10 1 -4 3 -2 0 -1 -1` on one line or multiple lines.

(b). Set one coefficient, called **setCoeff**, e.g.,

P.setCoeff(50,0); // set x^0 coeff to 50

Miscellaneous Rules

- Test your code thoroughly. You may find **lab1.cpp** file on canvas to help clarify the functional requirements. HOWEVER, it doesn't thoroughly test your Poly class. We will run our own main to test your code.
- Use valgrind often to find memory leaks (as you develop).
- You must handle errors such as trying to access a non-existent term. **All operations must work correctly and not crash under any circumstances,** so thoroughly test your program. Do **NOT** print error messages; deal with errors appropriately. For example, for the coefficient of a non-existent term, you could return zero.
- You do **NOT** need to handle data type errors, e.g., entering a char instead of int. **You are NOT allowed to use anything from the STL or 342's Array class.** You may not fix the array size at some large constant. The point of this assignment is to review memory management; you will be allocating/deallocating memory often.
- Do **NOT** discuss implementation (arrays) on Canvas. Any questions having to do with arrays come to only me. Any questions posted on Canvas would have to do with the polynomial ADT. Of course, you can always discuss class related stuff, software, etc.
- If your program does not compile with the sample main given, you will receive a maximum of 50% of the points. It will not be fixed! Note that this sample main typically does not test your program thoroughly.
- **NEVER** make us enter characters at the terminal. Remove all "cin >> ..." or system("pause") from your code before you turn it in.

Submission Requirements:

The **softcopy of well-commented code (.cpp, .h)** needed to be submitted to Canvas by the due time. You may **optionally** submit a typescript file to record your program in action (this is often helpful in resolving possible grading discrepancy).

Though you may develop and test your code using any compiler, any IDE, you will electronically turn in your code from the linux machines in UW1-320. See **Linux guide** on Canvas (Pages/For Assignments) for directions on how to do this.

As always expected when programming, comment clearly and thoroughly. Clearly state any assumptions you make in the beginning comment block of the appropriate place, e.g., the class definition. Comments in the class definition file should describe the ADT, all functionality, and assumptions. Pre and post conditions are recommended, but not required. Functions in the .cpp file must be separated using comment separator lines. See from the 342 sample code (http://courses.washington.edu/css342/zander/code.html) the **Array class** for an example of commenting and separator lines (in .cpp file).

Evaluation Criteria: - 100 points

Completeness and Correctness:	70 pts; - We will run our own main to test your code.
Coding style and quality	30 pts

See Files/Assignments/gradingRubric.pdf on canvas for grading information.
See Files/Assignments/codingStandards.pdf for coding, documentation, and style guidelines.