Programming Assignment #3

DEADLINE: October 19, 2018 at Midnight-1min (11:59 pm)
POINTS: 200

General Rules:

- **Students are responsible for ensuring that their files were submitted correctly on Canvas.** This means making sure their files were submitted **without error, on time, and also submitting the correct files**. Students will not be permitted to "submit the correct file" after the due date.
- **Backup your files.** Make frequent backups of your files to either email or the cloud to prevent losing your homework due to unforeseen events such as laptop malfunction. If you are using CLion to write your programs on your own computer or laptop, consider setting up automatic SFTP synchronization so that your files are **backed up automatically on linprog**. You can follow the instructions outlined in the following link https://www.jetbrains.com/help/clion/settings-deployment.html
- **Turn in all assignments on time!** Late assignments will be accepted only one day after the due date, with a deduction of 10% of the grade. **Assignments more than a day late will not be accepted.** Students are not permitted to "re-do" assignments. Assignment deadlines are STRICTLY enforced. If you have a ***compelling and documented reason*** (see Syllabus for accepted reasons) for not being able to meet the deadline, you must inform the instructor and provide documentation ***at least 24 hours before*** the due date.
- **Compiling**. Programs that do not compile show a lack of testing, which is a large part of programming. There will be a 5% point penalty for each compile error in a student's code that has to be fixed in the grading process. **Make sure your code compiles on linprog before you submit it!**

Objective: Upon completion of this program, you should have more experience with friend functions and overloading basic operators for use with a C++ class.

Task: Create a class called **Polynomial**. Objects of type **Polynomial** will store and manage univariate polynomials. The class, along with the required operator overloads, should be written in the files "***Polynomial.h***" and "***Polynomial.cpp***".

Class Details and Requirements

A univariate polynomial is an arithmetic expression of the form:

$$\boldsymbol{a_n x^n + a_{n-1} x^{n-1} + \cdots + a_2 x^2 + a_1 x^1 + a_0}$$

Where *x* is a lowercase letter variable that can take on different numeric values and $a_n, \ldots, a_2, a_1,$ *and* a_0 are constants called the *coefficients* of the polynomial. The highest exponent with non-zero coefficient, *n*, is called the degree of the polynomial. For example, $0x^2 + 2x + 3$ is normally written as $2x + 3$ and has degree 1. Note that x^1 is the same as *x*. A non-zero constant (e.g., 5, 39, 1005) has the degree zero. A polynomial whose coefficients are all zero (i.e., the value 0) has degree -1.

Some examples of univariate polynomials are:

- $-2.3x^{23} - 1$ (degree 23)
- $-2y^3 + 3y^2 + 1.5y$ (degree 3)
- 5 (degree 0)
- 0 (degree -1)

1. Your class must allow for storage of polynomials of any degree up to and including MAX_DEGREE (a global constant defined within the class) and should also store the name of the univariate letter used for the polynomial (e.g., 'x'). **Note:** Make sure that your program works when the value of the MAX_DEGREE constant is changed.

2. **Constructor(s)**
   - There should be a default constructor that creates a polynomial of degree -1. Namely, a polynomial where all coefficients are zero. The univariate letter used for the polynomial should be 'x'.

   - A second constructor should expect a single int parameter. This constructor allows an integer to be passed in and used to determine the coefficient values of the polynomial. The parameter received will determine the value of a_0 and each consequent coefficient will take the value of the previous coefficient decreased by one (e.g., $a_1 = a_0-1$, $a_2 = a_1-1$, etc.) until all the coefficients up to MAX_DEGREE are assigned. The univariate letter used for the polynomial should be 'x'.
     Example: Given MAX_DEGREE = 5,
     Polynomial p1(4) would result in the following polynomial

     $$-1x^5 + 0x^4 + 1x^3 + 2x^2 + 3x + 4$$

     Note that this constructor will also act as a "conversion constructor" allowing automatic type conversions from type int to type Polynomial.

3. The Polynomial class should have at least the public member functions described below. You may write any other member functions you feel necessary, but the public interface must include all the functionality described here.

   - **clear**
     This function should set all the coefficients of the polynomial to zero and the degree to -1.

   - ***evaluate***
     This function should return the double resulting from evaluating the polynomial at the double value given as parameter. If no parameter is given, the function should return the result of evaluating the polynomial at 0.
     Example: Given the polynomial p1 as $-3x^2 + 4x + 1$
     p1.evaluate() // returns 1
     p1.evaluate(2) // returns -3
     p1.evaluate(5.5) // returns -67.75

   - ***getCoefficient***
     This function should receive an integer parameter k. If 0 <= k <= the degree of the polynomial, the return value is the coefficient of x^k. Otherwise, the return value is zero.
     Example: Given the polynomial p1 as $3x^5 + x - 2$
     p1.getCoefficient(5) // returns 3
     p1.getCoefficient(0) // returns 2
     p1.getCoefficient(-1) // returns 0

   - ***getDegree***
     This function should return the degree of the polynomial.
     Example: Given the polynomial p1 as $x^2 + 25x - 3$
     p1. getDegree () // returns 2

Example: Given the polynomial p2 as $x^3 + x - 1$
p2.getDegree() // returns 3

- ***setCoefficient***
  This function should set a coefficient of the polynomial. This function should receive two parameters: an integer indicating the exponent of the coefficient to be set and a value for the coefficient. If the exponent 0<= k<= MAX_DEGREE, the coefficient of the x^k term has to be set to the new coefficient value. No change should be done otherwise. Moreover, if no value is provided for the new coefficient value, the corresponding coefficient should be set to zero.
  The function should return true if the coefficient is updated and false otherwise.

  Example: Given the polynomial p1 as $x^3 + x - 1$ and MAX_DEGREE = 5
  After a call to p1.setCoefficient (2,3), p1 becomes $x^3 + 3x^2 + x - 1$ *//returns true*
  Next, after a call to p1. setCoefficient (1), p1 becomes $x^3 + 3x^2 - 1$ *//returns true*
  Next, after a call to p1. setCoefficient (11), p1 remains $x^3 + 3x^2 - 1$ *//returns false*

- ***setLetter***
  This function changes the letter used to represent the letter variable in the polynomial. The function should accept both uppercase and lowercase entries. However, the value of the letter variable in the polynomial object should always be stored as a lowercase alphabetic character. No change should be made if the user provides a character other than a letter. The function should return true upon a successful update and false otherwise.

  Example: Given the polynomial p1 as $x^3 + x - 1$
  A call to *p1.setLetter('m')* would change the polynomial to $m^3 + m - 1$
  A call to *p1.setLetter('D')* would change the polynomial to $d^3 + d - 1$
  // The previous calls will return true
  A call to *p1.setLetter('1')* would not change the polynomial and return false

4. **Extraction operator >>**

Create an overload of the extraction operator >> for reading polynomials from an input stream. The input format for a polynomial object will be the letter variable, a semicolon, followed by the coefficient values separated by a comma (MAX_DEGREE + 1 values in total).

You may assume that this will always be the format that is entered (i.e., your function does not have to handle incorrect entries that would violate this format). However, this function should check the values that come in. In the case of an incorrect entry, just set the Polynomial object to represent a polynomial of degree -1, as a default. An incorrect entry occurs if the value for the letter variable is not a letter.

Examples: Given MAX_DEGREE = 4

Input	Corresponding Polynomial
x;4,3,0,0,0	$3x + 4$
x;2,8.7,0,1.3,9	$9x^4 + 1.3x^3 - 8.7x + 2$
x;-1,3,5,0,0	$5x^2 + 3x - 1$
m;-2,-8.7,0,0,-7	$-7m^4 - 8.7m - 2$

Invalid inputs:

- 2;4,0,0,0,0

5. **Insertion operator <<**

Create an overload of the insertion operator << for output of Polynomials. This should output the polynomial expression in the following format:

$$P(x) = a_n x\text{^}n + a_{n-1} x\text{^}(n-1) + \cdots + a_1 x + a_0$$

The output should follow the format as above, with the following exceptions:

- If a coefficient is zero, then the corresponding term should not be displayed.
- The letter variable with exponent 1 should be displayed without exponent (e.g., x^1 should be displayed as *x NOT as x^1)*
- If the decimal part of an exponent is zero (e.g., 4.0, -23.0) only the integer part of the coefficient should be printed (e.g., 4, -23). Otherwise, the coefficients should be displayed with exactly one decimal value.
- If the coefficient is a negative number, the operator preceding it should be a minus (–) rather than a plus (+) symbol.
- If the Polynomial has a of degree -1, print "Polynomial of degree -1"

Examples:

- $9x^5 + 1.3x^3 - 8.7x + 2$
- $-8.7y$
- $-9x^3 - 3.2x^2 - 8.7x + 265$
- Polynomial of degree -1

6. **Comparison operators**

Create overloads for all 6 of the comparison operators (< , > , <= , >= , == , !=). Each of these operations should test two objects of type Polynomial and return the result of the comparison as true or false. You are testing the Polynomials for order and/or equality based on their degree and the value of their coefficients using the usual meaning of order and equality for numbers (these functions should not do comparisons by evaluating the Polynomials).

Two polynomials P, Q are equal if they have the same degree and all their coefficients are equal.

For the purpose of this assignment, given two Polynomials P and Q, P is considered to be larger than Q if either:

a) The degree of P > the degree of Q
b) They have the same degree and the sum of the coefficients of P is larger than the sum of the coefficients of Q

7. **Arithmetic Operations**

Create operator overloads for the following arithmetic operations (+ , - , *) , to perform addition, subtraction, and multiplication of two polynomials. Each of these operators will perform its task on two Polynomial objects as operands and will return a Polynomial object as a result using the usual meaning of arithmetic operations on polynomials with the same letter variable.
Note that the two Polynomial operands will be treated as having the same letter variable even if their letter variables are different. The resulting Polynomial of these operations will have the letter variable of the first polynomial operand.

In the multiplication operator, if the polynomial resulting from the multiplication has a degree larger than MAX_DEGREE, the Polynomial object returned by the function will be cut to contain only the elements up to MAX_DEGREE.

Examples: Given MAX_DEGREE = 5

Operand 1	Operand 2	Operation	Result
$x^3 + x + 5$	$2x^2 + 3x - 2$	+	$x^3 + 2x^2 + 4x + 3$
$x^3 + x + 5$	$6y^3 + y$	+	$7x^3 + 2x + 5$
$x^3 + x + 5$	$2x^2 + 3x - 2$	-	$x^3 - 2x^2 - 2x + 7$
$y^3 + y + 5$	$6x^3 + x$	-	$-5y^3 + 5$
$-2z^3$	$z^3 + z - 1$	*	$-2z^4 + 2z^3$
$y^2 + y + 5$	$6x^2 + x$	*	$6y^4+7y^3+31y^2 + 5x$

8. **Unary Operators**

Create overloads for the increment and decrement operators (++ and --). You need to handle both the pre- and post- forms (pre-increment, post-increment, pre-decrement, post-decrement). These operators should have their usual meaning -- increment will add 1 to the Polynomial, whereas decrement will subtract 1 from the polynomial.

Examples:
Given Polynomial p1 = $2x^2 + 3x - 2$

cout << p1++; // *prints* $2x^2 + 3x - 2$ p1 is now $2x^2 + 3x - 1$
cout << ++p1; // *prints* $2x^2 + 3x$ p1 is now $2x^2 + 3x$
cout << p1--; // *prints* $2x^2 + 3x$ p1 is now $2x^2 + 3x - 1$
cout << --p1; // *prints* $2x^2 + 3x - 2$ p1 is now $2x^2 + 3x - 2$

Polynomial p2(); // *All the coefficients are zero (degree -1 polynomial)*
cout << p2++; // *prints Polynomial of degree -1* p2 is now 1
cout << --p2; // *prints Polynomial of degree -1* p2 is now of degree -1

General Requirements

1. Make sure that your program works with g++ on linprog before you hand it in.
2. Include a comment header with your NAME and SECTION at the top of your submitted files.
3. No global variables, other than constants.
4. No goto statements.
5. No system calls.
6. Declare all member data of the class as private.
7. Use **const** whenever is appropriate in the functions' declarations and definitions.
8. You may use the **iostream**, **iomanip**, **cstdlib, cstdio, cstring**, **cctype**, and **cmath** libraries.
9. Do not use language or library features that are C++11-only.
10. No cin or cout statements should happen in your definition of the classes.
11. When you write source code, it should be readable and well-documented.
    - Refer to the general notes on style guidelines posted on Canvas
12. Your ***Polynomial.h*** file should contain the class declaration only, whereas the ***Polynomial.cpp*** file should contain the member function definitions.

Testing Your Class

You will need to test your class, which means you will need to write one or more main programs that will call upon the functionality (i.e., the public member functions) of the class and exercise all of the different functions and cases for each function. You do **NOT** need to turn any test programs in, but you should write them to verify your class' features.

You will find on Canvas a sample main program (***main.cpp***) and one sample test run of ***main.cpp***. These files are just to get you started, illustrating some sample calls. **Keep in mind**, this is just a **basic sample, NOT a comprehensive test**. It is your job to test your class thoroughly to ensure that it meets the requirements listed above in the specification.

Deliverables

Submit your ***Polynomial.h*** and ***Polynomial.cpp*** files through Canvas. Do **NOT** send program submissions through e-mail. E-mail attachments will **NOT** be accepted as valid submissions.

Make sure your filenames are these exact names. Do **NOT** submit the ***main.cpp*** file. Do **NOT** submit any test main files you wrote.

Grading Rubric

Item	Points
Class storage	15
Constructor(s) • Default constructor [5 points] • Conversion constructor [10 points]	15
Mutator functions • clear [5 points] • evaluate [8 points] • getCoefficient [5 points] • getDegree [5 points] • setCoefficient [8 points] • setLetter [5 points]	36
Extraction operator >>	15
Insertion operator <<	15
Arithmetic operators [12 points each *3]	36
Unary operators [8 points each * 2]	16
Comparison operators [7 points each * 6]	42
Code quality and documentation (i.e., comments / header / style / readability)	10
Total Points	200