**Project 3**

**Polynomials**

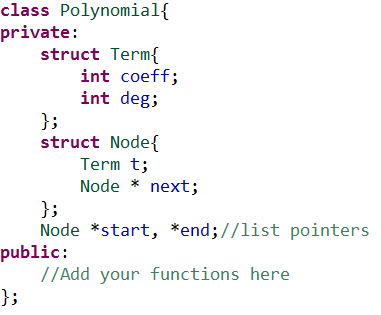
|  |  |
| --- | --- |
| **Release Date** | Thursday, April 16, 2015, 11:55PM |
|  |  |
| **Due Date** | Thursday, April 30, 2015, Before Class, on SLATE |
|  |  |
| **Absolute Weight** | 5% |
|  |  |

**Statement**

The objective of this programming project is to learn the concept of classes in C++, pointers to classes, dynamic objects, and operator overloading etc. You will be implementing a class called Polynomials to store and operate upon polynomials. You will be using lists (as described in class) to store the polynomials.

**What do you have to code?**

Simple. We need you to submit the code for the class Polynomial (you can submit the main program, however we will replace it with our own during the evaluation), whose skeleton is defined in the following snapshot.



While you add to this skeleton, you cannot subtract from it.

Polynomial()

As described in class, the advantage of storing a polynomial in a list, rather than an array is that, for polynomials of large degree, we shall be saving a lot of space by not storing the zero coefficients, which are inevitable in the array format. Instead, every node in the list will contain a term, with both its coefficient and degree. The degree of the polynomial is simply the highest degree among the terms.

Your [JOB[http://cdncache-a.akamaihd.net/items/it/img/arrow-10x10.png](http://www.sarimbaig.com/cp/projects/p3.html)](http://www.sarimbaig.com/cp/projects/p3.html) is to add the following functionalities to the class.

1. **Constructors**
   * Constructor with string parameter, accepts polynomials as character strings, e.g., "-2x^50+3x^2-4" which represents the polynomial -2x50+3x2-4. Let us call this format the "standard format".
   * Copy constructor, which does the usual by making a deep copy of the entire list.
2. **Operators**
   * operator = which does the usual by making a deep copy of the entire list.
   * operators +, -, \* for adding, subtracting and multiplying two polynomials, or a polynomial and a number. In case one of the operands is a number it could be the first or the second operand.
   * operator '<<' for outputing a polymonial in the standard format described above
   * operator '>>' for reading a polymonial in the standard format described above.
   * operator ++ for both pre and post increments.
   * operator - for the uniary minus operator to negate the entire polynomial.
   * operator == to compare if two polynomials are the same.
3. **Other Methods**
   * evaluate(int): evaluates the value of a polynomial at a particlar value of x.
   * clear: deletes the entire polynomial.
   * derivate(): returns the first derivative of a polynomial w.r.t. x.
   * deleteNegativeTerms(int): deletes all those terms which are negative at the value of x passed in parameter.
   * maxima(int,int): returns the value of x, from the range passed in parameters, where the polynomial has max value.
   * minima(int,int): returns the value of x, from the range passed in parameters, where the polynomial has min value.
   * whichX(int): returns the value of x where the polynomial has the value passed as parameter (inverse function of evaluate). In case no value of x evaluates to exactly the passed value, return the x which produces the nearest value.
4. **Extra Credit**
   * operator / for dividing one polymonial by another.
   * A method called plot which prints a four quadrant cartersian plot for the polynomial, for -50<=x<=+50, with origin at the center of the screen. In cases where for a particular value of x, the value of the polynomial is too large or two small to fit into the screen, simply indicate this with a dot on top, or bottom of the screen.

**Plagiarism Policy**

We will check the codes for plagiarism with each other, and codes from last year's class. Where detected, plagiarism is punishable by awarding zero in all assignments [25% absolute] at least.

**Good Luck! **