# OO Polynomial Handling

Assigned: Wednesday, November 11
Due: Thursday, November 19, before midnight
Value: 20 points (for successfully submitting a functionally correct program to your Assembla repo before the deadline). Late submissions will receive a 0.

## Purpose

- Learn more about OOD, create an Abstract Data Type (ADT), and C++ programming.
- Learn real pair programming.

## Pair Programming

Please work with one other person on this assignment using pair programming technique. You may choose your teammate. Do not forget to put both names and EIDs on the submitted program.

Please sign up your group on Canvas for this assignment. The submission is per team. The two persons in the team will get the same grade. Only one of the team members has to submit the code to their repository.

## Deliverables

The `poly.h`, and `assignment-7.cpp` program source files implementing the functionality described below.

## Background

It is assumed that you have some background in understanding simple polynomials.

You are to design and write a program that can perform the following operations on polynomials over a single variable X, i.e. each polynomial is a f (x).

1. Symbolically add 2 polynomials together,
2. Symbolically multiply 2 polynomials together,
3. Evaluate a polynomial for a given value of x.

**For maximum flexibility we require that the internal representation for a polynomial will be a linked list of terms.**

### Input Requirements

A set of polynomials will be stored in a text file whose name and contents must be read in by the program. All other user inputs will come from the keyboard. See the sample user dialog below for more details. A text file "ptest1.txt" will contain the polynomials. The first line of the ptest1.txt will contain the number of polynomials existing in the file. Each subsequent line will contain a polynomial.

For example, let us say the polynomial ptest1.txt contains the following

```
2
4x^5 - 2x^3 + 2x + 3
8.3x^4 + 4x^3 - .3x + 9
```

The first line indicates that the file has two polynomials and the subsequent lines represent the 2 polynomials. Each polynomial is on a line of text. The terms and operators in the polynomial are internally delimited by one space character. The symbol '^' is used throughout this assignment to mean 'raised to the power of'. Note how the coefficients may or may not have a decimal point. The coefficient will always be explicit even if it has value 1; for example, you will not be given "`x`" as a polynomial, but "`1x`" or "`1.0x`".

**You may assume that you will not be given an invalid polynomial input file, e.g. every polynomial that your program will be tested with will be properly formed and have at least one term in it; and every file will have at least 1 polynomial in it. Term coefficients can either by integer or real. Exponents are integer.**

## Output Requirements

Output the user prompts and the user inputs to the screen, as well as, the results of each operation. Output of all polynomials will be in descending order of the terms by degree. See the sample dialog below for more details. Floating point numbers should be formatted with 1 decimal places.

## Process Requirements

This is a menu driven application. Repeatedly prompt the user and process the selected operation until the user selects the quit option.

## Representation requirements

Each polynomial must be represented inside of your program as a linked list as follows: for every term in the polynomial there is one entry (i.e. node) in the linked list consisting of the term's coefficient and degree. Zero-coefficient terms are NOT stored. Subtracted terms in the polynomial can be represented by a negative coefficient.

For example, the following polynomial is made up of 4 terms:

```
4x^5 - 2x^3 + 2x +3
```

and can be represented as a linked list of nodes in descending degree order (degree order is useful but not required) as follows:

```
poly1 -> (4,5) -> (-2,3) -> (2,1) -> (3,0)
```

where each node holds a coefficient, a degree and a pointer to the next node.

Notes about representation:

- An EMPTY (zero) polynomial is represented by a linked list with NO NODES in it, i.e. referenced by a NULL pointer.
- Coefficients are floating point numbers, degrees are positive integers.
- There must not be more than one term in the same degree of each polynomial.
- When you add or multiply 2 polynomials you will be dynamically creating a new linked list to hold the resultant polynomial.

**Display**

Display all the polynomials and prompt the user to choose the operation and the polynomials that they want the operations to be performed on. Output the final polynomial used. For example, for the files ptest1.txt and ptest2.txt the display will be as shown:

**Sample User Dialog Box:** *(user input is in red)*

```
Enter the file that you would like to work with: ptest1.txt
The polynomials available for operation are:
1. 4.0x^5 + -2.0x^3 + 2.0x + 3.0
2. 8.0x^4 + 4.0x^3 + -3.0x + 9.0
What operation would you like to perform?
1. ADD polynomials
2. MULTIPLY polynomials
3. EVALUATE polynomial

Enter choice #: 1
Enter the two polynomials that you would like to work with: 1,2
The symbolic sum of the 2 polynomials is:
4.0x^5 + 8.0x^4 + 2.0x^3 + -1.0x + 12.0

Do you want to perform additional operations on the existing file (Y/N)? Y
What operation would you like to perform?
1. ADD polynomials
2. MULTIPLY polynomials
3. EVALUATE polynomial

Enter choice #: 2
Enter the two polynomials that you would like to work with: 1,2
The symbolic product of the 2 polynomials is:
32.0x^9 + 16.0x^8 + -16.0x^7 + -20.0x^6 + 52.0x^5 + 38.0x^4 + -6.0x^3
+ -6.0x^2 + 9.0x + 27.0

Do you want to perform additional operations on the existing file (Y/N)? N
Do you want to work with another file (Y/N)? Y
Enter the file that you would like to work with: ptest2.txt
The polynomials available for operation are:
1. 2.0x^5 + -1.0x^3 + 2.0x + 3.0
2. 3.0x^4 + 4.0x^3 + -3.0x + 9.0
3. 1.0x^6 + 4.0x^3 + -1.0x + 9.0
What operation would you like to perform?
1. ADD polynomials
2. MULTIPLY polynomials
3. EVALUATE polynomial

Enter choice #: 3
Enter the polynomial that you would like to work with: 2
Enter the evaluation point (the value of x): 2
```

```
Value of that polynomial at 2.0 is: 83.0

Do you want to perform additional operations on the existing file (Y/N)? N
Do you want to work with another file (Y/N)? N
Thank you for using this program!
```

## OO Design Requirements

You are to create a **Poly** class that represents a new data type for declaring, creating and manipulating polynomials. The linked list representation of the polynomial and all of the required functions for manipulating them should be encapsulated inside of the Poly class.

You will also need a **Node** struct to represent each term of the polynomial. It will contain the following members: coefficient, degree (exponent) and a pointer to the next term in the linked list.

Your main function will contain the menu driven interaction with the user, will use the Poly constructor to create objects of that type, and will call the other functions as needed to output, add, multiply and evaluate the polynomials.

## Poly Class design

You may place your Poly and Node class definitions in a separate header file and #include it into your main function program file or you may define Poly and Node inline before the main function.

Each polynomial must be represented as a linked list of node objects (terms). The data structure used to represent a polynomial is to be a *private* data member of the Poly class. Polynomials are only accessible by the driver program (main function) through the PUBLIC member functions provided in the Poly class. You will need to pass Poly references (pointers) to the functions that operate on the polynomials.

The set of polynomials available to be operated on for a given file can be represented as a dynamically allocated array of pointers to Poly objects. p1 and p2 in the examples below are parameters which would receive the arguments p[1] and p[2] for example.

Required public member functions in the Poly class will be needed to:

1. Construct a polynomial from a given string (use a constructor function to do this).

2. Symbolically add 2 polynomials together, the function header will look like:
   Poly *addPolys (Poly *p2);
   Poly *p1 would be the implicit parameter and the first operand of the add operation
3. Symbolically multiply 2 polynomials together, the function header will look like:
   Poly *multiplyPolys (Poly *p2);
   Poly *p1 would be the implicit parameter and the first operand of the multiply operation
4. Evaluate a polynomial for a given value of x. The function header will look like:
   double evalPoly ( );
5. Output a given polynomial - Output of all polynomials will be in descending order by degree. Terms with a zero coefficient should be skipped. For example, the following output is incorrect

```
2x^2 + 0x + -1
```

Coefficients should be formatted with one decimal place. For the previous example, the correct output would be:

```
2.0x^2 + -1.0
```

The output should be terminated with a new line character.

6. Checks whether the polynomial is equal to another polynomial. The function header is:
   bool equals(Poly *p2);
   Poly *p1 would be the implicit parameter and the first operand of the equals operation
7. Destruct all memory associated with a Poly object *via the destructor*.

**Optional:** extra credit – max score will be capped at 20.

Overload operators +, *, (), and == operators to be able to execute code like

```
Poly p1("1x + 2");
Poly p2("3x^2 + 2.3x + 1");
Poly p3 = p1 + p2;
Poly p4 = p1 * p2;
Poly p5 = p2 * p1;
double x = p2(5.0); // evaluates p2 at value 5.0
if (p4 == p5) { cout<<"multiplication is commutative\n"; }
```

## Setup

1. You need to create a Project named Lab7 in Visual Studio for this lab. Follow the same setup as in previous labs:
   The project location should be your <repo> folder. Be sure to uncheck the "Create directory for solution" option. In the project wizard, make sure to check "Empty project" and uncheck "Security Development Lifecycle (SDL) checks"
2. From the Solution Explorer, add all the files to your Lab7 project, so your file structure will be <repo>/Lab7/<individual .cpp or .h files>