# Project 06: Polynomials

Covered Concepts: templated classes, STL list, use of multiple types for template, creating test driver program, private member function

A single-variable polynomial, p(x), of degree *n* over a numeric type, NumType, is of the form:

$$p(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0$$
$$\textit{where } n \in \text{nonnegative integers}, a_i \in \text{NumType, and } a_n \neq 0.$$

The $a_i$*'s* are called *coefficients*. Each $a_i x^i$ are called *monomials* (also called *terms*). The numeric type, NumType, can be any C++ built-in numeric type (short, int, long, float, double, long double) or a class type (e.g. rational, complex) that has the following defined operators: +, -, *, ==, !=, =, +=, -=, *=,  ≫, and ≪, where each operator definition has the "normal" semantics. The numeric type is the same for all coefficients in a polynomial instance.

## Polynomial Syntax

- Polynomial syntax for the input and output methods of your polynomial class is based upon finite sequences of pairs of numbers: coefficient, degree. The polynomials should be read in as a finite sequence of coefficient-degree pairs terminated by a 0. This means there should be an odd number of numbers read. The zero acts as a sentinel or end of sequence. Pairs with the same degree can be repeated (internally added) and the pairs cans be entered in any order with respect to the degree value.
- All polynomials must be written as a finite sequence of monomials starting with the monomial of highest degree, then next highest degree, and until the monomial with the smallest degree. Monomials must be represented in the form of cx^d where the coefficient, c, is in the numeric format of the numeric type T. The degree, d, is printed as a non-negative int.
- For output, no two distinct monomials in a polynomial should have the same degree.
- Monomials with coefficients equal to 0 are not written (and in your implementation should not be represented). The only polynomial with a 0 coefficient is the 0 polynomial. Monomials with coefficients equal to 1 are written with the 1 implicit, except for the monomial of degree 0.

## Polynomial Examples

| | Representation with Traditional Mathematical Typesetting | Input Syntax Specification | Output Syntax Specification |
|---|---|---|---|
| 1. | $x^2 + 5x – 1$ | 1 2 5 1 -1 0 0<br>or<br>5 1 -1 0 1 2 0<br>or … | x^2 + 5x -1 |
| 2. | $4x^{93} – 4x^{45} + x^2 – 500$ | 4 93 –4 45 1 2 -500 0 0<br>or -500 0 -4 45 1 2 4 93 0<br>or … | 4x^93 – 4x^45 + x^2 – 500 |
| 3. | $3x^2$ | 3 2 0<br>or 3 2 0 1 0 or … | 3x^2 |
| 4. | -5x | -5 1 0 | -5x |
| 5. | 5 | 5 0 0 | 5 |
| 6. | 1 | 1 0 0 | 1 |
| 7. | 0 | 0 | 0 |

Note that in the zero case, only 0 is needed for input and not 0 0. Note also that the output syntax does not match the input syntax. Ease of input and readable output are the two reasons for having a different syntax. To be truly well-designed the input and output should match. The reason for this is for persistent data and for command console piping '|' across processes. We will use the syntax above, but for a well-designed and useful polynomial class, there should at least be an option or additional operators to allow for matching syntax. For now, do not worry about exceptions for this project. However, do recognize the following error condition. If a negative degree is entered, print "Negative degree" and exit the input method.

## Implementation Specifications:

One could use a vector of monomials where the value of the $i^{th}$ component of the vector represents the coefficient of $i^{th}$ monomial of degree i. However, with sparse polynomials, ones with a majority of monomials with 0 coefficients, the vector implementation promotes an inefficient use of space. See, for example, the polynomial in Example 2 shown above.

An alternate design and implementation strategy, which ***must be used*** for this project, is the use of linked lists of monomials. For the list of monomials, you may use the list-implementation along with the associated list-iterator implementations supplied by your textbook or by the C++ STL <list>.

Each monomial is represented as a coefficient-degree pair of values. The coefficients are of a numeric type T, where T = short, int, long, float, double, or long double or a numeric class like rational or complex. For any numeric class type, one must provide defined operators +, -, *, ==, !=, =, +=, -=, *=, ≫, and ≪, where each operator definition has the "normal" semantics. The list of monomials must be stored in the order of decreasing degrees.

You must provide a monomial template class and a polynomial template class whose template variable, say NumType, will be instantiated with one of the numeric types listed earlier in this document. The polynomial class will have in object-oriented design and programming language terminology a "has a" relationship. In addition to the list of monomials, the polynomial class may have instance fields (data members in C++ terminology) that represent the number of monomials in the list of monomials and the highest degree term. Although this can be viewed as redundant information, the space needed for these extra fields are worth the gain in savings of time complexity for many of the methods.

Call your classes **Monomial** and **Polynomial**. Note the use of capital letters for the first letter of each of these class names.

You must implement the two template classes (monomial and polynomial) in file **Polynomial.h.** You may code the implementation for the various functions in the .h file. This will allow easier compilation.

If you opt to put the definitions (with the exception of the constructors) in **Polynomial.cpp** file. The **Polynomial.h** file will need #include "Polynomial.cpp." The reason for this is the ease of compiling the template classes.

You are given the public interfaces for each. Your main assignment is to understand and to define the operations over polynomials and monomials and to utilize the C++ STL <list>. See the included **"Polynomial.h" for the starting** source code.

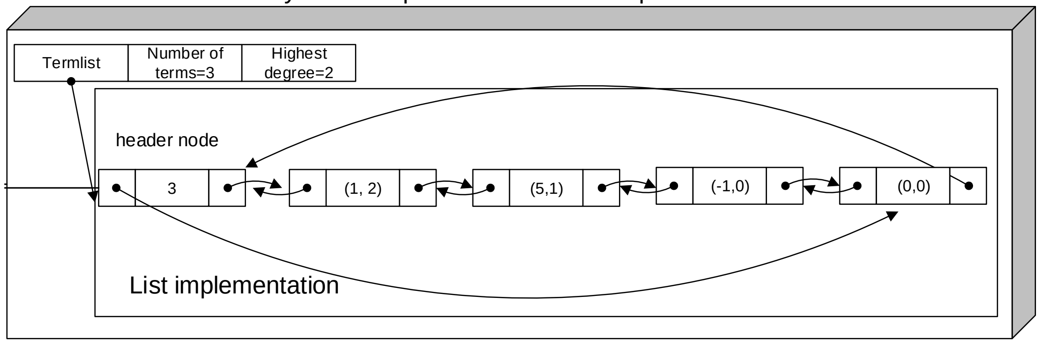
Create the **P06PolynomialDriver.cpp** file to test your work. See the **"P06 Polynomials Sample Run.pdf"** for format and various tests of input and function calls.

## Examples of Graphical Depiction of Polynomials:

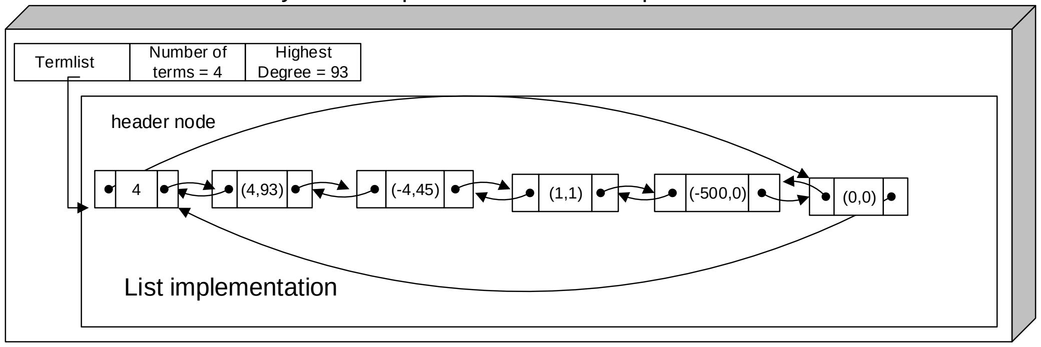
You can use the C++ STL class <list>.

If you were to implement the list class, you would have to implement it as shown in the “List implementation” box.

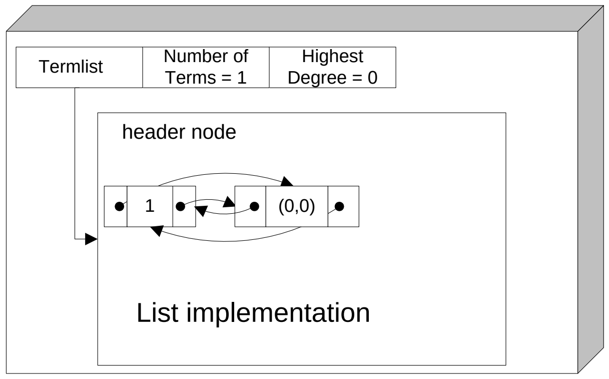
Polynomial Implementation of Example 1



Polynomial Implementation of Example 2



Polynomial Implementation of Example 7



## Submission Requirements:

A zip file that must contain the **<u>entire</u>** project folder containing your Visual Studio 2022 project for Project 7 and .txt files representing test runs.