Lab 3 - Implementation of Polynomial Using Linked List

Due Date: Lab sessions February 27 – March 10, 2023
Assessment: 5% of the total course mark.

Instructions:

- You can complete this lab with one lab partner.
- By the end of the lab session you must demonstrate to your TA the `C++` code. You will also be asked to answer questions regarding your implementation. Both partners can demonstrate together, but the TA can pick the student to domonstarte certain parts.
- Both partners have to submit the source code separately.
- NO REPORT IS NEEDED. Submit the source code for each class in a separate TEXT file. Include your student number in the name of each file. The electronic submission on Avenue to Learn of all `C++` source code must be done by the end of your designated lab session.

Description:

In this lab you will implement polynomials and operations on polynomials using a **sorted singly linked list with a dummy header**. To implement the linked list nodes you must use the `C++` calss `PolyNode` provided in this lab. Additionally, you need to implement the `C++` class `Poly` (to represent polynomials). In your implementation you are **not allowed** to use `std::list`! Please compute the asymptotic run time and space complexity of all methods and be ready to present them to the TA with explanations at your demo.

Definitions and Specifications:

A polynomial is an expression of the form:

$$P(X) = \sum_{i=0}^{n} a_i X^i$$

Where a_i are real numbers. Each $a_i X^i$ is a term in the polynomial, where i is the **degree** of the term and a_i is the **coefficient**. The degree of a polynomial is the highest value of i such that $a_i \neq 0$. For example, the polynomial $P_1(X) = 10X^{1000} + 2X + 1$ has degree 1000. Notice that even if the degree is very high the number of non-zero terms of $P_1(X)$ (terms with non-zero coefficient) is very low. Such a polynomial is called sparse. Sparse polynomials can be efficiently represented using linked lists. Each node in the linked list represents a non-zero term, and must contain a **field to store the coefficient** and a **field to store the degree** of the term.

You are required to represent the polynomial using a **sorted singly linked list with a dummy header**, in other words, where the nodes are sorted in **decreasing order of**

their degrees. The linked list should **NOT** contain two nodes with the same degree. It also should **NOT** contain nodes to represent terms with the coefficient equal to 0. For example, for polynomial $P_1(X)$ specified above, the sorted linked list representation should contain three nodes, one for the term $10X^{1000}$, one for $2X$ and one for the term 1 (notice that the degree of the last term is 0). This node count does **NOT** include the dummy header. These requirements on the linked list have to be satisfied **at all times**, in other words each time a constructor or a method of the class `Poly` finishes execution.

The **zero** polynomial should be represented by an empty linked list. Notice that the degree of a zero polynomial equals -1 by convention.

Class `PolyNode`: You must use the `C++` class `PolyNode` given below, to implement the linked list nodes.

```
class PolyNode {
public:
    int deg;
    double coeff;
    PolyNode* next;

    PolyNode(int d, double c, PolyNode* n)
    { deg = d; coeff = c; next = n;}
};
```

Class `Poly`: All its fields have to be `private`, and it must contain one `private` field named `head`, which is a pointer to the dummy header of the linked list. The class should contain at least the following `public` constructors and destructor:

- `Poly()`: creates the zero polynomial with degree equals -1.
- `Poly(const std::vector<int>& deg, const std::vector<double>& coeff)`: creates a polynomial out of the two input vectors as follows: vector `deg` contains the degrees of **non-negative** terms, vector `coeff` contains the coefficients of **non-zero** terms. The two vectors have the same size, moreover vector `deg` is sorted in strictly decreasing order (thus any two elements are different), and it does not contain negative values. For any index `j` in vector `coeff`, the vector element `coeff[j]` represents the coefficient of the term of degree equal to `deg[j]`. When writing your code you may assume that the requirements on the input are satisfied (hence no error checking is needed).
- ∼`Poly()`: frees the memory allocated for the linked list nodes in a polynomial.

Class `Poly` should contain at least the following `public` methods:

- `void addMono(int i, double c)`: adds the monomial cX^i to `this` polynomial. You may assume that i is non-negative (thus no error checking is needed), `c` can be zero. The method modifies `this` polynomial. You may need to add/delete/update one node to your linked list in different cases.

- `void addPoly(const Poly& p)`: adds polynomial `p` to `this` polynomial. The method modifies `this` polynomial, but it does not modify polynomial `p`.
- `void multiplyMono(int i, double c)`: multiplies `this` polynomial by the monomial cX^i. You may assume that i is non-negative, `c` can be zero. The method modifies `this` polynomial.
- `void multiplyPoly(const Poly& p)`: multiplies `this` polynomial by polynomial `p`. The method modifies `this` polynomial, but it does not modify polynomial `p`.
- `void duplicate(Poly& outputPoly)`: copies `this` polynomial to `outputPoly`. The incoming `outputPoly` can be zero polynomial or non-zero polynomial. The linked list corresponding to the outgoing `outputPoly` must be a duplicate of `this` linked list.
- `int getDegree()`: returns the degree of the polynomial.
- `int getTermsNo()`: returns the number of non-zero terms of the polynomial.
- `double evaluate(double x)`: evaluates the polynomial expression for the value `x` of the variable.
  Example: If `this` polynomial is $P(X) = 4X^3 + 5X + 2$, then the invocation `evaluate(2.0)` returns the value $4 * 2.0^3 + 5 * 2.0 + 2 = 44.0$.
- `std::string toString()`: returns a `string` representation of the polynomial. The string should specify the degree of the polynomial and the coefficients of all non-zero terms, listed in decreasing order of terms degrees.
  Example: If `this` polynomial is $P(X) = 4X^3 + 5X + 2$, then the string representation could be:
  "degree=3; a(3)=4.0; a(1)=5.0; a(0)=2.0"

NOTES:

- Compute the asymptotic run time and space complexity of the following methods:
  - `addMono`, `addPoly`, `multiplyMono`, `multiplyPoly`, `getDegree`, `getTermsNo`, `duplicate`.

  Be prepared to present your derivations to the TA at demo time and provide verbal justification for your analysis.
- You are permitted to implement other `private` methods inside class `Poly`. However, you are not permitted to modify the class `PolyNode`. Additionally, class `Poly` must contain one field which is an `PolyNode` pointer named `head`.
- You may use the code from the lecture notes on linked list, or from the tutorial. However, you may need to adapt the code and provide complete references.
- To get credit for the lab you must demonstrate your code (i.e., class `Poly`) in front of a TA during your lab session. A 50% penalty will be applied for either a late demo or if the electronic submission of the code is late.

SUBMISSION INSTRUCTIONS:

NO REPORT IS NEEDED. Submit the source code for the classes `Poly` in a separte text file. Include your student number in the name of each file. For instance, if your student number is 12345 then the files should be named as follows: Poly_12345.h.txt, Poly_12345.cpp.txt. Submit the files in the Assignment folder on Avenue by the end of your designated lab session.