Ceng213 - Data Structures
Programming Assignment 1 : Polynomials via Linked Lists

Spring 2019

1 Objectives

In this assignment you are expected to implement a doubly linked list data structure, in which each node will have pointers to both the previous and the next node. The data structure will also include a head and a tail pointer that points to the first and the last nodes in the list. The details of the structure is explained further in the following sections. You will use this specialized linked list structure to represent polynomials and implement some functions on them.

Keywords: *Linked List, Polynomial*

2 Polynomials (Mathematical Background)

Most of the information about polynomials in this section is from the Wikipedia page "Polynomial" at the following url `https://en.wikipedia.org/wiki/Polynomial`.

In mathematics, a polynomial is an expression consisting of variables (also called indeterminates) and coefficients, that involves only the operations of addition, subtraction, multiplication, and non-negative integer exponents of variables. An example of a polynomial of a single indeterminate, x, is $x^2 - 4x + 7$. An example in three variables is $x^3 + 2xyz^2 - yz + 1$. The x occurring in a polynomial is commonly called either a variable or an indeterminate. When the polynomial is considered as an expression, x is a fixed symbol which does not have any value (its value is "indeterminate"). However, when one considers the function defined by the polynomial, then x represents the argument of the function, and is therefore called a "variable". These two words are used interchangeably. A polynomial P in the indeterminate x is commonly denoted either as P or as $P(x)$.

A polynomial in a single indeterminate x can always be written (or rewritten) in the form

$$a_n x^n + a_{n-1} x^{n-1} + ... + a_2 x^2 + a_1 x^1 + a_0 x^0$$

where $a_0, ..., a_n$ are constants and x is the indeterminate. A polynomial can either be zero or can be written as the sum of a finite number of non-zero terms. Each term consists of the product of a number—called the coefficient of the term—and a finite number of indeterminates, raised to nonnegative integer powers.

The exponent on an indeterminate in a term is called the degree of that indeterminate in that term; the degree of the term is the sum of the degrees of the indeterminates in that term, and the degree of a polynomial is the largest degree of any one term with nonzero coefficient. A term with no indeterminates and a polynomial with no indeterminates are called, respectively, a constant term and a constant polynomial. The degree of a constant term and of a nonzero constant polynomial is 0. The degree of the zero polynomial, 0, (which has no terms at all) is generally treated as not defined. For example: $-5x^2y$ is a term. The coefficient is -5, the indeterminates are x and y, the degree of x is two, while the degree of y is one. The degree of the entire term is the sum of the degrees of each indeterminate in it, so in this example the degree is $2 + 1 = 3$.

Forming a sum of several terms produces a polynomial. For example, the following is a polynomial: $3x^2 - 5x + 4$. It consists of three terms: the first is degree two, the second is degree one, and the third is degree zero. The first term has coefficient 3, indeterminate x, and exponent 2. In the second term, the coefficient is -5. The third term is a constant. Because the degree of a non-zero polynomial is the largest degree of any one term, this polynomial has degree two.

Two expressions that may be transformed, one to the other, by applying the usual properties of commutativity, associativity and distributivity of addition and multiplication are considered as defining the same polynomial. Two terms with the same indeterminates raised to the same powers are called "similar terms" or "like terms", and they can be combined, using the distributive law, into a single term whose coefficient is the sum of the coefficients of the terms that were combined. It may happen that this makes the coefficient 0.

A real polynomial is a polynomial with real coefficients. Similarly, an integer polynomial is a polynomial with integer coefficients. A polynomial in one indeterminate is called a univariate polynomial, a polynomial in more than one indeterminate is called a multivariate polynomial.

3 Linked List Implementation (80 pts)

The linked list data structure used in this assignment is implemented as the class template `LinkedList` with the template argument `T`, which is used as the type of the data stored in the nodes. The node of the linked list is implemented as the class template `Node` with the template argument `T`, which is the type of the data stored in nodes. `Node` class is the basic building block of the `LinkedList` class. `LinkedList` class has two `Node` pointers (`head` and `tail`) in its private data field, which point to the first and the last nodes of the linked list.

The `LinkedList` class has its definition and implementation in *LinkedList.hpp* file and the `Node` class has its in *Node.hpp* file.

3.1 Node

`Node` class represents nodes that constitute linked lists. A `Node` keeps two pointers (namely `prev` and `next`) to its previous and next nodes in the list, and the data variable of type T (namely `element`) to hold the data. The class has three constructors (including a copy constructor), and the overloaded output operator. They are already implemented for you. You should not change anything in file *Node.hpp*.

3.2 LinkedList

`LinkedList` class implements a doubly linked list data structure with the `head` and the `tail` pointers. Previously, data members of `LinkedList` class have been briefly described. Their use will be elaborated in the context of utility functions discussed in the following subsections. You must provide implementations for the following public interface methods that have been declared under indicated portions of *LinkedList.hpp* file.

3.2.1 `LinkedList();`

This is the default constructor. You should make necessary initializations in this function.

3.2.2 `LinkedList(const LinkedList &obj);`

This is the copy constructor. You should make necessary initializations, create new nodes by copying the nodes in given `obj` and insert new nodes into the linked list.

3.2.3 `~LinkedList();`

This is the destructor. You should deallocate all the memory that you were allocated before.

3.2.4 `Node<T> *getFirstNode() const;`

This function should return a pointer to the first node in the linked list. If the linked list is empty, it should return NULL.

3.2.5 `int getNumberOfNodes();`

This function should return an integer that is the number of nodes in the linked list.

3.2.6 `bool isEmpty();`

This function should return `true` if the linked list is empty (i.e. there exists no nodes in the linked list). If it is not empty, it should return `false`.

3.2.7 `void insertAtTheFront(const T &data);`

You should create a new node with given `data` and insert it at the front of the linked list as the first node. Do not forget to make necessary pointer and head-tail modifications.

3.2.8 `void insertAtTheEnd(const T &data);`

You should create a new node with given `data` and insert it at the end of the linked list as the last node. Do not forget to make necessary pointer and head-tail modifications.

3.2.9 `void insertAfterGivenNode(const T &data, Node<T> *prev);`

You should create a new node with given `data` and insert it after the given node `prev` as its next node. Do not forget to make necessary pointer and head-tail modifications. If the given node `prev` is not in the linked list, do nothing.

3.2.10 `void removeNode(Node<T> *node);`

You should remove the given node `node` from the linked list. Do not forget to make necessary pointer and head-tail modifications. If the given node `node` is not in the linked list, do nothing.

3.2.11 `void removeAllNodes();`

You should remove all nodes in the linked list.

3.2.12 `Node<T> *findNode(const T &data);`

You should search for the node in the linked list with the data same with the given `data` and return a pointer to that node. You can use the operator== to compare two T objects. If there exists no such node in the linked list, you should return NULL.

3.2.13 `void swapNodes(Node<T> *node1, Node<T> *node2);`

You should swap the two given nodes `node1` and `node2`. You are not allowed to just swap the data in the nodes. Also, you are not allowed to create new nodes in this function. Do not forget to make necessary pointer and head-tail modifications. If either of the given nodes is not in the linked list, do nothing.

3.2.14 `void printAllNodes();`

You should print all nodes in the linked list. Each node must be printed on a single line. You should directly use the overloaded operator<< of `Node` class. If the linked list is empty, you should do nothing.

3.2.15 `LinkedList &operator=(const LinkedList &rhs);`

This is the overloaded assignment operator. You should remove all nodes in the linked list and then create new nodes by copying the nodes in given `rhs` and insert new nodes into the linked list.

4 Polynomial Implementation (20 pts)

Some mathematical details of the polynomials were given in section 2. For this assignment, we will have some assumptions for the polynomials that will be worked on. First of all, the polynomials in this assignment will be univariate polynomials (i.e. there will be only one indeterminate, x). You will not be asked for multivariate polynomials. Second, the polynomials will be integer polynomials (i.e. all coefficients of the terms will be integers). You will not be asked for real polynomials. Third, you will not be asked for any ordering between terms of polynomials with respect to the powers of the indeterminates. And the last assumption is that you will not be given zero terms or zero polynomials.

The polynomials in this assignment is implemented as the class `Polynomial`. `Polynomial` class has a `LinkedList` object (namely `terms`) with the type `Term` in its private data field. `Term` class represents the terms of polynomials. It is the type of the data stored in nodes of the linked list.

The `Polynomial` class has its definition in *Polynomial.hpp* file and its implementation in *Polynomial.cpp* file. The `Term` class has its definition in *Term.hpp* file and its implementation in *Term.cpp* file.

4.1 `Term`

`Term` class represents terms that constitute polynomials. A `Term` keeps the coefficient and exponent variables of type int (namely `coefficient` and `exponent`) to hold the data related with the term. The class has three constructors (including a copy constructor), getter and setter functions, the overloaded output operator, and the overloaded equality check operator. They are already implemented for you. You should not change anything in files *Term.hpp* and *Term.cpp*.

4.2 `Polynomial`

In `Polynomial` class, there is a single member variable named as `terms`, which is a `LinkedList` object. Information of all terms will be stored in this linked list and no other information will be stored. All member functions should utilize this linked list to operate as described in the following subsections. Default constructor and the constructor that takes a polynomial expression string and populates the terms to the linked list have already been implemented for you. Do not change those implementation. In *polynomial.cpp* file, you need to provide implementations for following functions declared under *polynomial.hpp* header to complete the assignment. You should not change anything in file *Polynomial.hpp*.

4.2.1 `Polynomial();`

This is the default constructor. It is already implemented. It does nothing.

4.2.2 `Polynomial(std::string expression);`

This is the constructor that takes a polynomial expression string and populates the terms to the linked list. It is already implemented. You can track the required format for the expression strings from the code. A sample string is "$(3)x^2 + (-1)x^2 + (4)x^5 + (7)x^3$"

4.2.3 `void simplifyByExponents();`

You should simplify the polynomial by combining the "similar terms". Two terms are similar if they have same determinates raised to the same powers. Combine ~~them~~ similar terms into a single term whose coefficient is the sum of the coefficients of the terms that were combined. While combining ~~two~~ similar nodes in the linked list, update the coefficient of the first node (the one that comes first) and delete ~~the second node (the other one)~~ the rest of the nodes. If the combined coefficient is 0, then delete ~~both nodes~~ the first node too.

4.2.4 `void printPolynomial();`

This function prints the polynomial as a polynomial expression to the standard output. It is already implemented. It uses the same format with the already implemented constructor above.

4.2.5 `Polynomial operator+(const Polynomial &rhs) const;`

This is the overloaded + operator. You should create a new `Polynomial` object with the terms of the first polynomial with the nodes of second polynomial (`rhs`) appended.

5 Driver Programs

To enable you to test your LinkedList and Polynomial implementations, two driver programs, *main_linkedlist.cpp* and *main_polynomial.cpp* are provided. Their expected outputs are also provided in *output_linkedlist.txt* and *output_polynomial.txt* files, respectively.

6 Regulations

1. **Programming Language:** You will use C++.
2. **S**tandard Template Library is **not** allowed.
3. **E**xternal libraries other than those already included are **not** allowed.
4. **T**hose who do the operations (insert, remove, find, print) without utilizing the linked list will receive **0 grade**.
5. **T**hose who modify already implemented functions and those who insert other data variables or public functions and those who change the prototype of given functions will receive **0 grade**.
6. **T**hose who use STL vector or compile-time arrays or variable-size arrays (not existing in ANSI C++) will receive **0 grade**. Options used for g++ are "-ansi -Wall -pedantic-errors -O0". They are already included in the provided Makefile.
7. **Y**ou can add private member functions whenever it is explicitly allowed.
8. **Late Submission:** Each assignment will have a fixed duration of 10 days and every student has a total of 5 days for late submissions during which no points will be deducted. Your assignment will not be accepted if you used up all of the 5 late days.
9. **Cheating:** We have zero tolerance policy for cheating. In case of cheating, all parts involved (source(s) and receiver(s)) get zero. People involved in cheating will be punished according to the university regulations. Remember that students of this course are bounded to code of honor and its violation is subject to severe punishment.
10. **Newsgroup:** You must follow the Forum (odtuclass.metu.edu.tr) for discussions and possible updates on a daily basis.

7 Submission

- Submission will be done via CengClass (cengclass.ceng.metu.edu.tr).
- Do not write a main function in any of your source files.
- A test environment will be ready in CengClass.
  - You can submit your source files to CengClass and test your work with a subset of evaluation inputs and outputs.
  - Additional test cases will be used for evaluation of your final grade. So, your actual grades may be different than the ones you get in CengClass.
  - Only the last submission before the deadline will be graded.