CMP 210 – Data Structures and Algorithms
Special Section – Spring 2023
Assignment - 03

Issue Date: **Tuesday – December 5th, 2023**
Submission Deadline: **Saturday – December 16th, 2023 (Till 12:00 am)**

Instructions!

1. You are required to do this assignment on your own. Absolutely **NO** collaboration is allowed, if you face any difficulty feel free to discuss with me.
2. Cheating will result in a **ZERO** for the assignment. (Finding solutions online is cheating, copying someone else's solution is cheating). Also, do not hand your work over to another student to read/copy. If you allow anyone to copy your work, in part or in whole, you are liable as well.
3. Hard **DEADLINE** of this assignment is **Saturday, November 16th, 2023**. No late submissions will be accepted after due date and time so manage emergencies beforehand.

Memory-Efficient Doubly Linked List: [25 Marks]

Recall that an ordinary Doubly Linked List requires space for two address fields to store the addresses of previous and next nodes. A memory-efficient version of Doubly Linked List can be created using only one space for the address field with every node. This memory-efficient Doubly Linked List is called XOR Linked List or Memory Efficient as the list uses bitwise XOR operation to save space for one address.

An ordinary doubly linked list stores addresses of the previous and next list items in each node, requiring two address fields. An XOR linked list compresses the same information into one address field by storing the bitwise XOR of the address for the previous and the address for the next in one field. Thus, in the XOR linked list, instead of storing actual memory addresses, every node stores the XOR of addresses of previous and next nodes.

```
// Node structure of a memory efficient doubly linked list
struct Node
{
      int data;
      Node* npx; /* XOR of next and previous node */
};
```

Here, we store the XOR of the next and previous nodes with every node and we call it *npx*, which is the only address member we have with every node. When we insert a new node at the beginning, *npx* of new node will always be XOR of NULL and the current head. And *npx* of the current head must be changed to XOR of new node and node next to the current head. For a better understanding, you may visit wikipedia. Your task is to implement the following functions for XOR linked list.

```
      void insertATHead(int val);
      void insertAtTail(int val);
      void insertAfter(int val);
      void deletetBefore(int val)
      int deleteAtHead();
      int DeleteAtTail();
      void printList();
```

Implement a driver program to test the functions of your class.

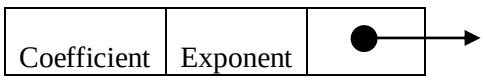
Polynomials [25 Marks]

An important application of linked list is to represent polynomials and their manipulations. The main advantage for storing polynomials in linked list is that it can accommodate number of polynomials of growing sizes, so that their combined size does not exceed the total memory available. A polynomial is an equation of the form:

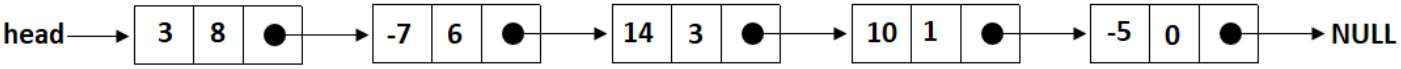
$$P(x) = a^1x\ e^1 + a^2x\ e^2 + a^3x\ e^3 + \ldots + a^nx\ e^n$$

Where, a^1, a^2, a^3 . . . a^n are coefficients, and e^1, e^2, e^3 . . . . e^n are exponents.

The structure of a node to represent a term of a polynomial can be as shown below:



Number of nodes to represent a polynomial is the same as the number of terms in the polynomial. Let's consider an example of a polynomial with five terms. $P(x) = 3x^8 - 7x^6 + 14x^3 + 10x - 5$. The linked list representation of this polynomial is as shown below:



Implement the following functions of the polynomial class using linked list.

1. **Constructors and destructor.**
2. **addTerm(Coefficient, power)**
   Add a new term to your polynomial having coefficient and power.
3. **getDegree( )**
   Returns the degree of the polynomial, for example the degree of polynomial $4x^5+3x^2+1$ is 5.
4. **getCoefficient(power)**
   Returns the coefficient of term x^{power} , For example if polynomial is $4x^5+3x^2+1$ then getCoefficient(2) should return 3.
5. **evaluate (value)**
   This function should evaluate the polynomial assuming x = value and returns the final answer.
6. **Overload + operator**
   Adds two polynomials. For example p1 = $4x^5+3x^2+1$ and p2 = $x^6+3x^3+9x^2+3$ then the answer of p1+p2 should be $x^6+4x^5+3x^3+12x^2+4$
7. **Overload – operator**
   Subtracts two polynomials. For example p1= (x3 + 3x2 + 5x – 4) and p2 = (3x3 – 8x2 – 5x + 6) then the answer of p1- p2 should be –2x3 + 11x2 + 10x –10.
8. **Clear ( )**
   Set the coefficients of all terms in the polynomial to zero.
9. **addToCoefficient(coefficient, power)**
   Adds the c to the coefficient of term having power p. For example, if polynomial is $4x^5+3x^2+1$ then addToCoefficient (10, 5) should changes the polynomial to $14x^5+3x^2+1$
10. **derivative ( )**
    This function should return the first derivative of the given polynomial. For example, our polynomial is $4x^2 + 6x$.
    The first step is to take any exponent and bring it down, multiplying it times the coefficient. In other words, bring the 2 down from the top and multiply it by the 4. Then reduce the exponent by 1. The final derivative of that term is 8x. The second term is 6x. Since the exponent is assumed to be 1, we can bring that down and multiply, which does not change the coefficient. Reducing the exponent by 1 makes it 0, so the derivative of 6x is just 6x0, or the number 6. For any linear term like 6x, the derivative will simply be that coefficient.

    You may use following class definitions.

```
Class Polynomial;   //forward declaration
Class  Term
{
   friend Polynomial;
   private:
        int exp;
        int coef;
        Term* next;
};
```

```
Class  Polynomial
{
   private:
        Term* head;
public:
        // implement required functions here.
};
```

Write a main program providing menu to make it easy to use and test polynomial class functionalities. No marks shall be given without this driver program.

Bad programmers worry about the code. Good programmers worry about data structures and their relationships.

--Linus Torvalds