EECS2040 Data Structure Hw #3 Coding (Chapter 4 Linked List)
due date 5/1/2022

Format: Use a text editor to type your answers to the homework problem. You need to submit your HW in an HTML file or a DOC file named as **Hw3-SNo.doc** or **Hw3-SNo.html**, where SNo is your student number. Submit the **Hw3-SNo.doc or Hw3-SNo.html** file via eLearn. Inside the file, you need to put the **header and your student number, name (e.g., EECS2040 Data Structure Hw #3 (Chapter 4 of textbook) due date 5/1/2022 by SNo, name)** first, and then the **problem** itself followed by your **answer** to that problem, one by one. The grading will be based on the correctness of your answers to the problems, and the **format**. Fail to comply with the aforementioned format (file name, header, problem, answer, problem, answer,…), will certainly degrade your score. If you have any questions, please feel free to ask me.

Part 2 Coding (5% of final Grade, due 5/8)

You should submit:
(a) All your source codes (C++ file).
(b) Show the execution trace of your program.

1. (15%) Fully code and test the C++ template class Chain<T> shown below:

```
template < class T > class Chain;    // 前向宣告

template < class T >
class ChainNode {
friend class Chain <T>;
private:
      T data;
      ChainNode<T>* link;
};

template <class T>
class Chain {
public:
      Chain( ) {first = 0;} //  建構子將 first 初始化成 0
      //  鏈的處理運算
      .
      .
private:
      ChainNode<T> * first;
}
```

You must include:

(a) A destructor which deletes all nodes in the Chain.
(b) InsertFront() function to insert at the front of the Chain.
(c) DeleteFront() and DeleteBack() to delete from either end.
(d) Front() and Back() functions to return the first and last elements of the Chain, respectively.
(e) A function Get(int i) that returns the ith element in the Chain.
(f) Delete(int i) to delete the ith element.
(g) Insert(int i, T e) to insert as the ith element.
(h) Member function which will count the number of nodes in L.
(i) Member function that will change the data field of **the kth node** (the first 1st node start at index 0) of L to the value given by Y.
(j) Member function that will perform an insertion to the **immediate before of the kth node** in the list L.
(d) Member function that will **delete every other node** of L beginning with node first (i.e., the first, 3rd, 5th,…nodes of L are deleted).
(e) Member function divideMid that will divides the given list into two sublists of (almost) equal sizes.
(f) Member function that will **deconcatenate** (or **split**) a linked list L into two linked list. Assume the node denoted by the pointer variable split is to be the first node in the second linked list.
(g) Assume L_1 and L_2 are two chains: $L_1 = (x_1,x_2,..,x_n)$ and $L_2 = (y_1,y_2,\ldots,y_m)$, respectively. Member function that can **merge** the two chains together to obtain the chain $L_3 = (x_1,y_1,x_2,y_2,\ldots,x_m,y_m,x_{m+1},..,x_n)$ if n>m and $L_3 = (x_1,y_1,x_2,y_2,\ldots,x_n,y_n,y_{n+1},..,y_m)$ if n<m.

Write a client program (main()) to **demonstrate** those functions you developed.

2. (15%) Given a **circular linked list L** instantiated by **class** CircularList containing a private data member, **first** pointing to the first node in the circular list as shown in Figure 4.14.

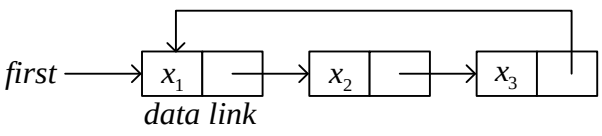

Fig. 4.14 A circular linked list

Write C++ codes to

(a) count the number of nodes in the circular list.

(b) insert a new node at the front of the list.

(c) insert a new node at the back (right after the last node) of the list.

(d) delete the first node of the list.

(e) delete the last node of the list.

(f) **delete every other node** of the list beginning with node first (i.e., the first, 3rd, 5th,…nodes of L are deleted).

(g) **deconcatenate** (or **split**) a linked circular list L into two circular lists. Assume the node denoted by the pointer variable split is to be the first node in the second circular list.

(h) Assume L_1 and L_2 are two circular lists: $L_1 = (x_1,x_2,..,x_n)$ and $L_2 = (y_1,y_2,\ldots,y_m)$, respectively. Implement a member function that can **merge** the two chains together to obtain the chain $L_3 = (x_1,y_1,x_2,y_2,\ldots,x_m,y_m,x_{m+1},..,x_n)$ if $n>m$ and $L_3 = (x_1,y_1,x_2,y_2,\ldots,x_n,y_n,y_{n+1},..,y_m)$ if $n<m$.

(i) Repeat (a) – (h) above if the circular list is modified as shown in Figure 4.16 below by introducing a dummy node, header.

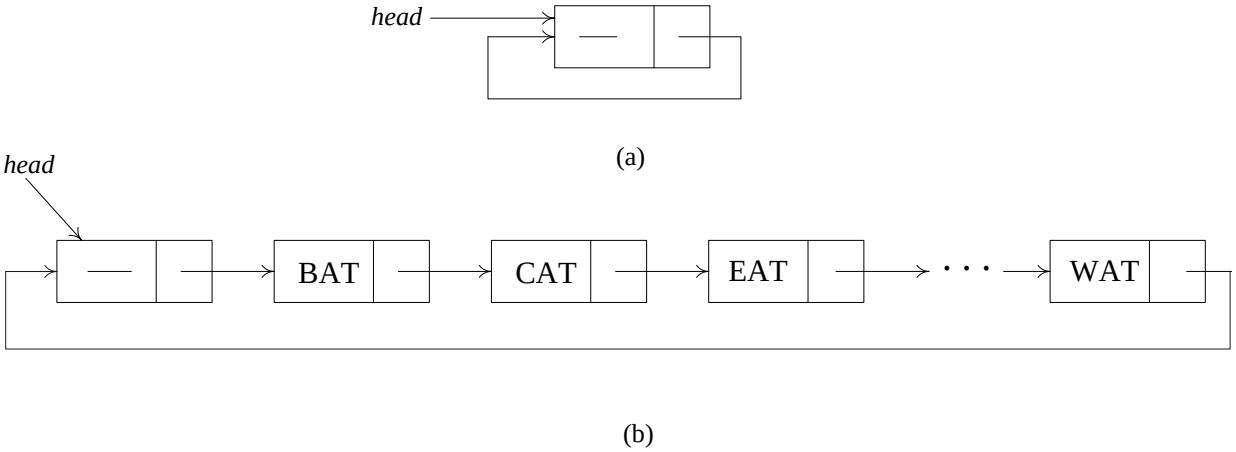

Figure 4.16 Circular list with a header node

Write a client program (main()) to **demonstrate** those functions you developed.

3. (20%) The class List<T> is shown below:

```
template <class T> class List;
template <class T>
class Node{
friend class List<T>;
private:    T data;
            Node* link;
};
```

```
template <class T>
class List{
public:
    List(){first = 0;}
    void InsertBack(const T& e);
    void Concatenate(List<T>& b);
    void Reverse();
    class Iterator{
    ….
    };
    Iterator Begin();
    Iterator End();
private:
    Node* first;
};
```

Fully code and test the C++ template class List<T> shown above. You must include:

(a) A destructor which deletes all nodes in the list.
(b) InsertFront() function to insert at the front of the list.
(c) DeleteFront() and DeleteBack() to delete from either end.
(d) Front() and Back() functions to return the first and last elements of the list, respectively.
(e) A function Get(int i) that returns the ith element in the list.
(f) Delete(int i) to delete the ith element
(g) Insert(int i, T e) to insert as the ith element
(h) Overload the output operator << to output all elements of the List object.
(i) As well as functions and forward iterator as shown above.
(j) Implement the stack data structure as a derived class of the class List<T>.
(k) Implement the queue data structure as a derived class of the class List<T>.
(l) Let $x_1, x_2, \ldots, x_n$ be the elements of a List<int> object. Each x_i is an integer. Write C++ code to compute the expression $\sum_{i=1}^{n-5}(x_i \times x_{i+5})$

Write a client program (main()) to **demonstrate** those functions you developed.

4. (25%) Develop a C++ class Polynomial to represent and manipulate univariate polynomials with double-type coefficients (use circular linked list with header nodes). Each term of the polynomial will be represented as a node. Thus a node in this system will have three data members as below.

coef	exp	link

Each polynomial is to be represented as a circular list with header node. To delete polynomials efficiently, we need to use an **available-space list** and associated functions GetNode() and RetNode() described in Section 4.5. The external (i.e., for input and output) representation of a univariate polynomial will be assumed to be a sequence of integers and doubles of the form: n, c_1, e_1, c_2, e_2, c_3, e_3,…, c_n, e_n, where e_i represents an integer exponent and c_i a double coefficient; n gives the number of terms in the polynomial. The exponents of the polynomial are in decreasing order.

Write and **test** the following functions:

(a) Istream& operator>>(istream& is, Polynomial& x): Read in an input polynomial and convert it to its circular list representation using a header node.
(b) Ostream& operator<<(ostream& os, Polynomial& x): Convert x from its linked list representation to its external representation and output it.
(c) Polynomila::Polynomial(const Polynomial& a): copy constructor
(d) Const Polynomila& Polynomial::operator=(const Polynomial& a) const[assignment operator]: assign polynomial a to *this.
(e) Polynomial::~ Polynomial(): desctructor, return all nodes to available-space list
(f) Polynomial operator+ (const Polynomial& b) const: Create and return the polynomial *this + b
(g) Polynomial operator- (const Polynomial& b) const: Create and return the polynomial *this – b
(h) Polynomial operator* (const Polynomial& b) const: Create and return the polynomial *this * b
(i) double Polynomial::Evaluate(double x) const: Evaluate the polynomial *this and return the result.

Write a client program (main()) to **demonstrate** those functions you developed.

5. (25%) The class definition for sparse matrix in Program 4.29 is shown below.

```
struct Triple{int row, col, value;};
class Matrix; // 前向宣告
class MatrixNode {
```

```
friend class Matrix;
friend istream& operator>>(istream&, Matrix&); //  為了能夠讀進矩陣
private:
    MatrixNode *down , *right;
    bool head;
    union { //  沒有名字的 union
        MatrixNode *next;
        Triple triple;
    };
    MatrixNode(bool, Triple*); //  建構子
}

MatrixNode::MatrixNode(bool b, Triple *t)    //  建構子
{
    head = b;
    if (b) {right = down = this;} //  列/行的標頭節點
    else triple = *t; //  標頭節點串列的元素節點或標頭節點
}

class Matrix{
friend istream& operator>>(istream&, Matrix&);
public:
    ~Matrix(); //  解構子
private:
    MatrixNode *headnode;
};
```

Based on this class, do the following tasks.

(a) Write the C++ function, **operator**+(**const** Matrix& b) **const**, which returns the matrix ***this** + b.

(b) Write the C++ function, **operator***(const Matrix& b) **const**, which returns the matrix ***this** * b.

(c) Write the C++ function, **operator**<<(), which outputs a sparse matrix as triples (i, j, a_{ij}).

(d) Write the C++ function, Transpose(), which transpose a sparse matrix.

(e) Write and test a **copy constructor** for sparse matrices. What is the computing time of your copy constructor?

Write a client program (main()) to **demonstrate** those functions you developed.