**EECS2040 Data Structure Hw #3 (Chapter 4 Linked List)**

**due date 4/25/2021**

***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. Send the **Hw3-SNo.doc or Hw3-SNo.html** file to me ([tljong@mx.nthu.edu.tw](mailto:tljong@mx.nthu.edu.tw)) via e-mail. Inside the file, you need to put the **header and your student number, name (e.g., EE2410 Data Structure Hw #3 (Chapter 4 of textbook) due date 4/25/2021 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 1 (due 4/25/2021)**

1. (30%) Given a template linked list **L** instantiated by the Chain class with a pointer **first** to the first node of the list as shown in Program 4.6 in textbook. The node is a ChainNode object consisting of a template data and link field.

|  |
| --- |
| **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***;** |
| **}** |

Program 4.6

1. **Formulate an algorithm** (pseudo code OK, C++ code not necessary) which will count the number of nodes in L. Explain your algorithm properly (using either text or graphs).
2. **Formulate an algorithm** that will change the data field of **the kth node** of L to the value given by Y. Explain your algorithm properly (using either text or graphs).
3. **Formulate an algorithm** that will perform an insertion to the immediate **before of the kth node** in the list L. Explain your algorithm properly (using either text or graphs).
4. **Formulate an algorithm** that will **delete every other node** of L beginning with node first (i.e., the first, 3th, 5th,…nodes of L are deleted). Explain your algorithm properly (using either text or graphs).
5. **Formulate an algorithm** 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. Formulate a step-by-step algorithm to perform this task. Explain your algorithm properly (using either text or graphs).
6. Assume L1 and L2 are two chains: L1 = (x1,x2,..,xn) and L2 = (y1,y2,…,ym), respectively. **Formulate an algorithm** that can merge the two chains together to obtain the chain L3 = (x1,y1,x2,y2,…,xm,ym,xm+1,..,xn) if n>m and L3 = (x1,y1,x2,y2,…,xn,yn,yn+1,..,ym) if n<m. Explain your algorithm properly (using either text or graphs).
7. (55%) 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

**formulate algorithms** (pseudo code OK, C++ code not necessary) to

1. count the number of nodes in the circular list. Explain your algorithm properly (using either text or graphs)
2. insert a new node at the front of the list. Discuss the time complexity of your algorithm. Explain your algorithm properly (using either text or graphs)
3. insert a new node at the back (right after the last node) of the list. Discuss the time complexity of your algorithm. Explain your algorithm properly (using either text or graphs)
4. delete the first node of the list. Discuss the time complexity of your algorithm. Explain your algorithm properly (using either text or graphs)
5. delete the last node of the list. Discuss the time complexity of your algorithm. Explain your algorithm properly (using either text or graphs).
6. Repeat (a) – (f) in Problem 1 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

1. (15%) 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;

};

1. Implement the stack data structure as a derived class of the class List<T>.
2. Implement the queue data structure as a derived class of the class List<T>.
3. Let x1, x2,…, xn be the elements of a List<int> object. Each xi is an integer. Formulate an algorithm (pseudo code OK, C++ code not necessary) to compute the expression

**Part 2 Coding (due 5/8)**

You should submit:

(a) All your source codes (C++ file).

(b) Show the execution trace of your program.

1. (30%) Fully code and test the C++ template class List<T> shown in Part 1 Problem 3 above. You must include:
2. A constructor which constructs an initially empty list.
3. A destructor which deletes all nodes in the list.
4. InsertFront() function to insert at the front of the list.
5. DeleteFront() and DeleteBack() to delete from either end.
6. Front() and Back() functions to return the first and last elements of the list, respectively.
7. A function Get(int i) that returns the ith element in the list.
8. Delete(int i) to delete the ith element
9. Insert(int i, T e) to insert as the ith element
10. Overload the output operator << to out put the List object.
11. As well as functions and forward iterator as shown above.

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

1. (35%) Develop a C++ class Polynomial to represent and manipulate univariate polynomials with double 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, c1, e1, c2, e2, c3, e3,…, cn, en, where ei represents an integer exponent and ci 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:

1. Istream& operator>>(istream& is, Polynomial& x): Read in an input polynomial and convert it to its circular list representation using a header node.
2. Ostream& operator<<(ostream& os, Polynomial& x): Convert x from its linked list representation to its external representation and output it.
3. Polynomila::Polynomial(const Polynomial& a): copy constructor
4. Const Polynomila& Polynomial::operator=(const Polynomial& a) const[assignment operator]: assign polynomial a to \*this.
5. Polynomial::~ Polynomial(): desctructor, return all nodes to available-space list
6. Polynomial operator+ (const Polynomial& b) const: Create and return the polynomial \*this + b
7. Polynomial operator- (const Polynomial& b) const: Create and return the polynomial \*this – b
8. Polynomial operator\* (const Polynomial& b) const: Create and return the polynomial \*this \* b
9. double Polynomial::Evaluate(double x) const: Evaluate the polynomial \*this and return the result.
10. (35%) 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.

1. Write the C++ function, **operator**+(**const** Matrix& b) **const**, which returns the matrix \***this** + b.
2. Write the C++ function, **operator**\*(const Matrix& b) **const**, which returns the matrix \***this** \* b.
3. Write the C++ function, **operator**<<(), which outputs a sparse matrix as triples (i, j, aij).
4. Write the C++ function, Transpose(), which transpose a sparse matrix.
5. 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.