**EE2410 Data Structure Coding HW #1 (Chapter 1~4 of textbook)**

**110061212郭郁翔**

You should submit:

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

(b) Show the execution trace of your program, i.e., write a client main() to demonstrate all functions you designed using example data.

Linked Lists: **due date: 23:59, 5/14/2023 (Sun.)**

1. (30%)

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

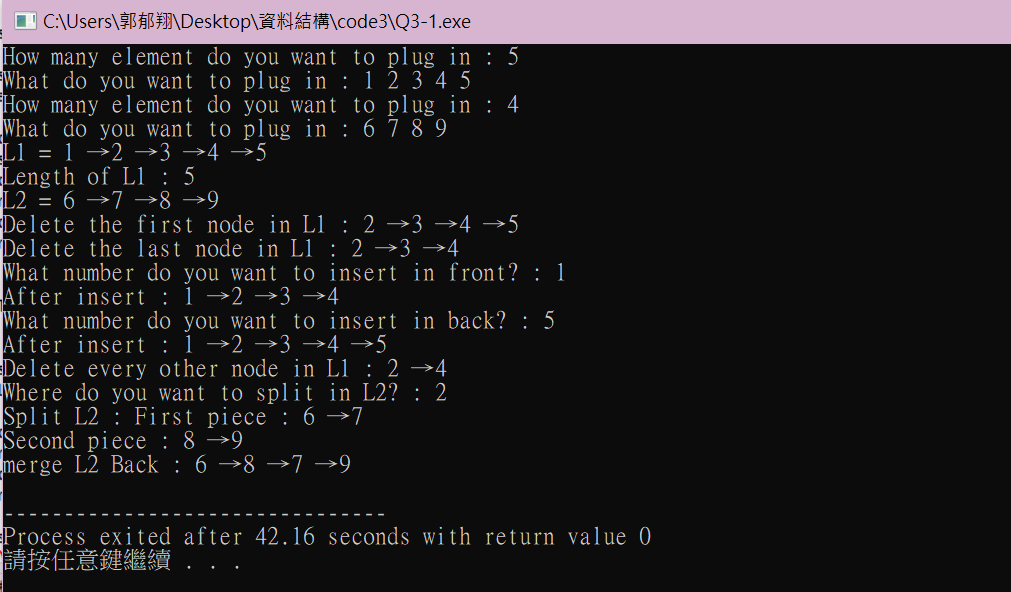
1. count the number of nodes in the circular list.
2. insert a new node at the front of the list InsertFront().
3. insert a new node at the back (right after the last node) of the list InsertBack().
4. delete the first node of the list DeleteFirst().
5. delete the last node of the list DeleteBack().
6. **delete every other node** of the list beginning with node first (i.e., the first, 3rd, 5th,…nodes of L are deleted).
7. **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.
8. Assume L1 and L2 are two circular lists: L1 = (x1,x2,..,xn) and L2 = (y1,y2,…,ym), respectively. Implement a member function 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.
9. (15%) Repeat (a) – (h) above if the circular list is modified as shown in Figure 4.16 below by introducing a dummy node, header.



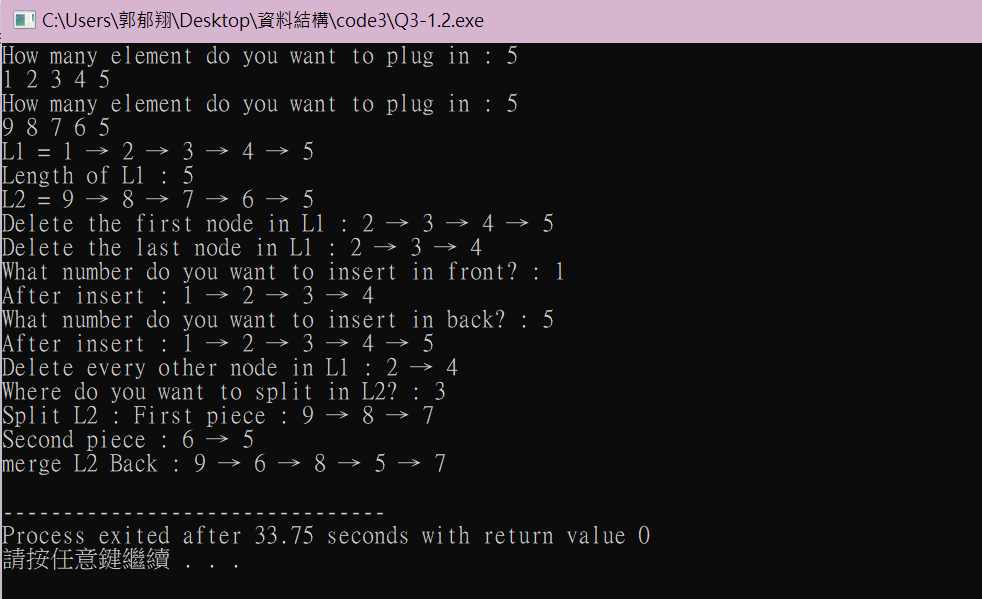
Figure 4.16 Circular list with a header node

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

Test a-h



Test i



1. (30%)

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;

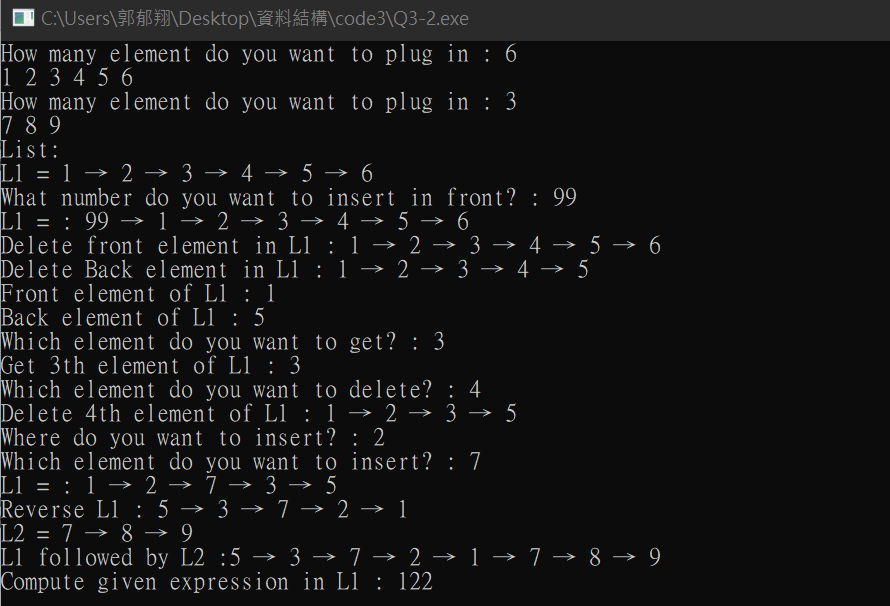
};

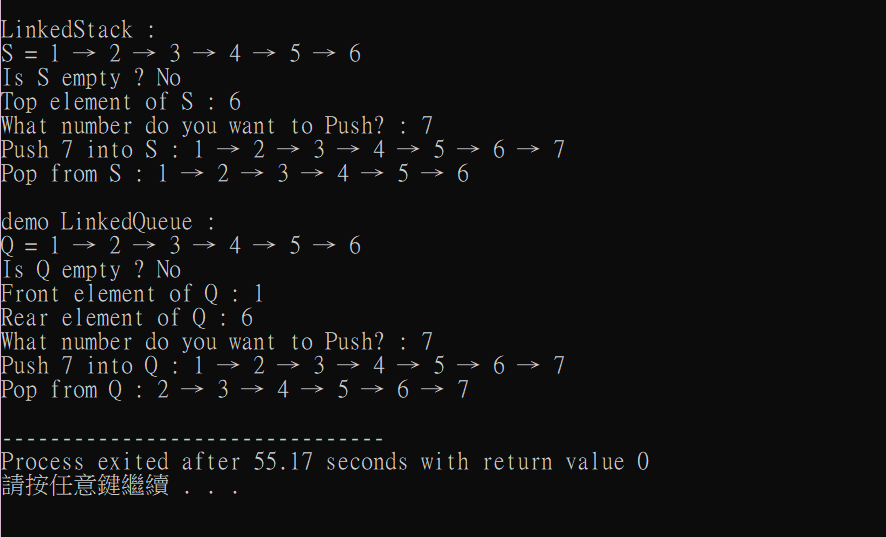
Fully code and test the C++ template class List<T> shown above. You must include:

1. A destructor which deletes all nodes in the list.
2. InsertFront() function to insert at the front of the list.
3. DeleteFront() and DeleteBack() to delete from either end.
4. Front() and Back() functions to return the first and last elements of the list, respectively.
5. A function Get(int i) that returns the ith element in the list.
6. Delete(int i) to delete the ith element
7. Insert(int i, T e) to insert as the ith element
8. Overload the output operator << to output all elements of the List object.
9. As well as functions and forward iterator as shown above.
10. Implement the stack data structure as a derived class of the class List<T>.
11. Implement the queue data structure as a derived class of the class List<T>.
12. Let x1, x2,…, xn be the elements of a List<int> object. Each xi is an integer. Write C++ code to compute the expression

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

Test





1. (20%)

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 |

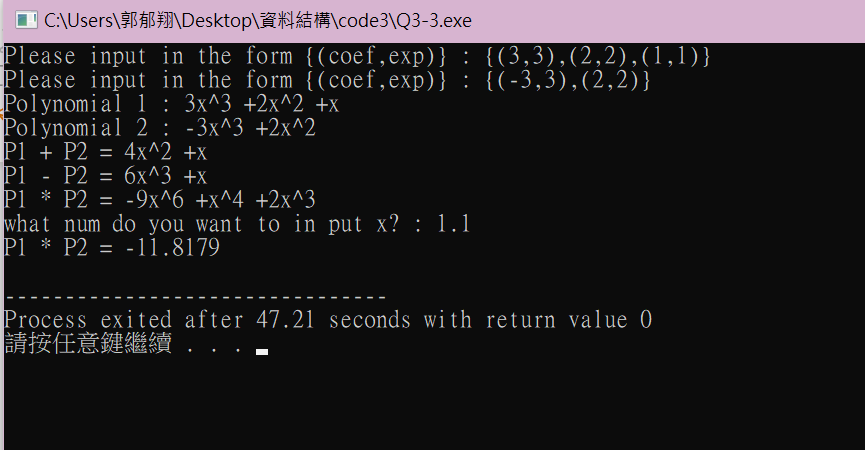
Each polynomial is to be represented as a circular list with header node. To delete polynomials efficiently, we need to use an **available-space list** and associated functions GetNode() and RetNode() described in Section 4.5. The external (i.e., for input and output) representation of a univariate polynomial will be assumed to be a sequence of integers and doubles of the form: n, 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.

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

Test



1. (20%)

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***;**

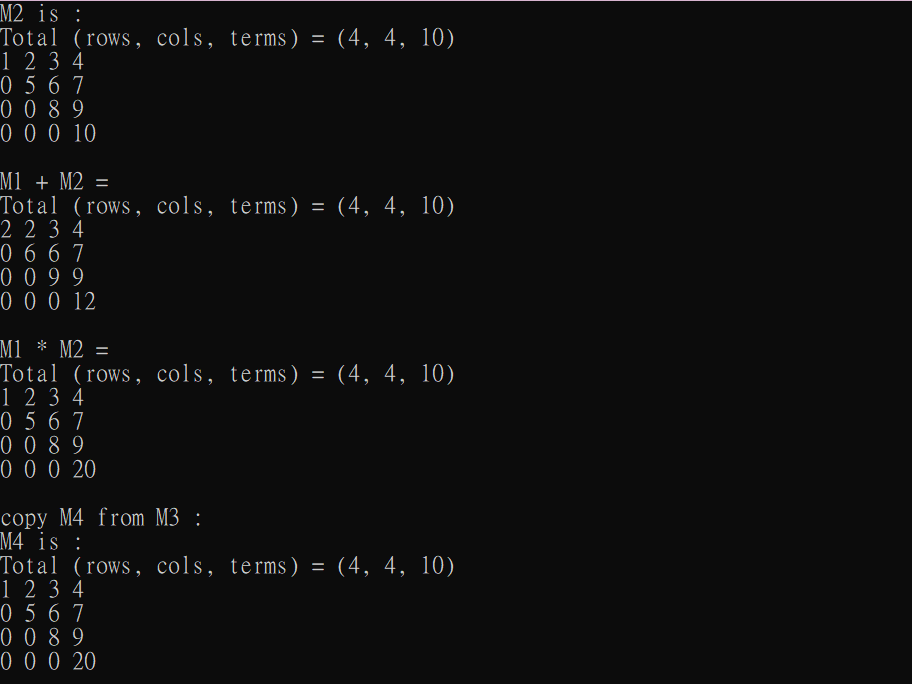
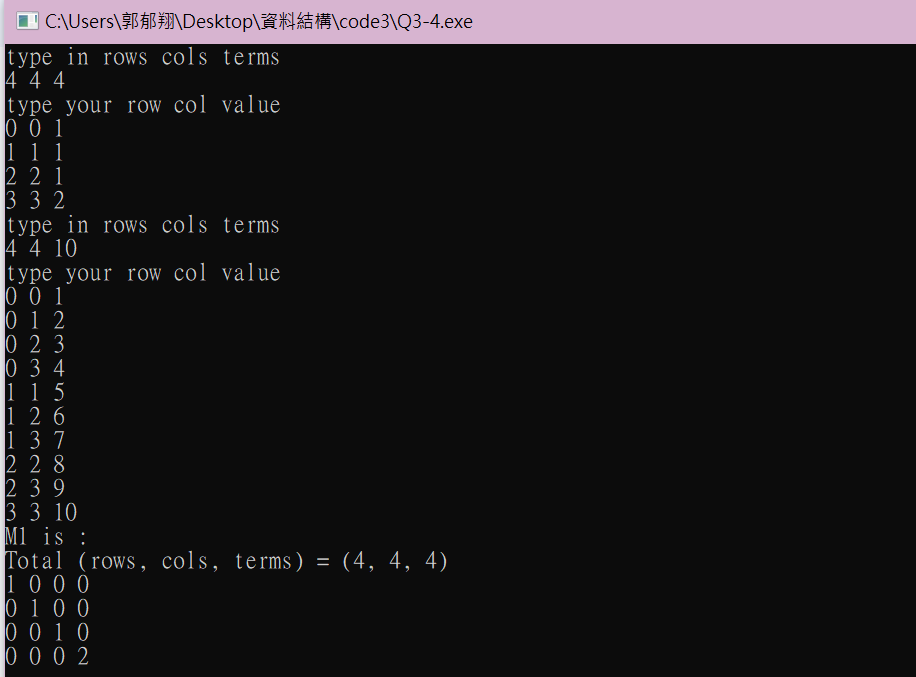
**};**

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.

Test1



Test2

