EECS2040 Data Structure Hw #3 Coding (Chapter 4 Linked List)
due date 5/1/2022
by 108011235 陳昭維

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.

```
*****testbench start*****

[test for InsertFront]:
9 8 7 6 5 4 3 2 1 0

[test for InsertBack]:
9 8 7 6 5 4 3 2 1 0 90 91 92 93 94 95 96 97 98 99 100 101 102 103 104

[test for DeleteBack]:
9 8 7 6 5 4 3 2 1 0 90 91 92 93 94 95 96 97 98 99

[test for Front and Back]:
Front of C1 is: 9
Back of C1 is: 99

[test for Get()]:
5th element of the chain is: 5
7th element of the chain is: 3
7th element of the chain is: 92

[test for Delete()]:
delete the 5th of the chain
9 8 7 6 4 3 2 1 0 90 91 92 93 94 95 96 97 98 99

[test for Insert()]:
Insert 55 as 5th element
9 8 7 6 55 4 3 2 1 0 90 91 92 93 94 95 96 97 98 99

[test for Size() to count the element number]:
Size of the chain is: 20

[test for ReplaceNode() to change the data field of given node]:
replace the 7th node's data to 777
9 8 7 6 55 4 777 2 1 0 90 91 92 93 94 95 96 97 98 99

[test for insertion before kth node]:
Insert 666 before the 7th node
9 8 7 6 55 4 666 777 2 1 0 90 91 92 93 94 95 96 97 98 99

[test for delete all the odd Node]:
8 6 4 777 1 90 92 94 96 98

[test for divideMid()]:
C1: 8 6 4 777 1
sublist: 90 92 94 96 98

[test for deconcatenate]:
C1: 8
C2: 6 4 777 1

[test for merge]:
C2: 6 4 777 1
sublist: 90 92 94 96 98
mergelist: 6 90 4 92 777 94 1 96 98


*****end*****
```

```cpp
int main() {
    Chain <int> C1;

    cout << "\n*****testbench start*****" << endl;

    cout << "\n[test for InsertFront]:" << endl;
    for(int i = 0; i < 10; i++)
        C1.InsertFront(i);
    C1.display();

    cout << "\n[test for InsertBack]:" << endl;
    for(int i = 90; i < 105; i++)
        C1.InsertBack(i);
    C1.display();

    cout << "\n[test for DeleteBack]:" << endl;
    for(int i = 0; i < 5; i++)
        C1.deleteBack();
    C1.display();

    cout << "\n[test for Front and Back]:" << endl;
    cout << "Front of C1 is: " << C1.Front()
    << endl << "Back of C1 is: " << C1.Back() << endl;

    cout << "\n[test for Get()]:" << endl;
    cout << "5th element of the chain is: " << C1.Get(5) << endl;
    cout << "7th element of the chain is: " << C1.Get(7) << endl;
    cout << "7th element of the chain is: " << C1.Get(13) << endl;

    cout << "\n[test for Delete()]:" << endl;
    cout << "delete the 5th of the chain" << endl;
    C1.Delete(5);
    C1.display();

    cout << "\n[test for Insert()]:" << endl;
    cout << "Insert 55 as 5th element" << endl;
    int k = 55;
    C1.Insert(5, k);
    C1.display();
```

```
cout << "\n[test for Size() to count the element number]:" << endl;
cout << "Size of the chain is: " << C1.Size() << endl;

cout << "\n[test for ReplaceNode() to change the data field of given node]:" << e
cout << "replace the 7th node's data to 777" << endl;
int j = 777;
C1.ReplaceNode(7, j);
C1.display();

cout << "\n[test for insertion before kth node]:" << endl;
cout << "Insert 666 before the 7th node" << endl;
int p = 666;
C1.insertBeforeK(7, p);
C1.display();

cout << "\n[test for delete all the odd Node]:" << endl;
C1.deleteAllOddNode();
C1.display();

cout << "\n[test for divideMid()]:" << endl;
Chain<int> sublist;
C1.divideMid(&sublist);
cout << "C1: ";
C1.display();
cout << "sublist: ";
sublist.display();


cout << "\n[test for deconcatenate]:" << endl;

Chain<int>* C2;
C2 = C1.deconcatenate(1);
cout << "C1: ";
C1.display();
cout << "C2: ";
C2->display();

cout << "\n[test for merge]:" << endl;
Chain<int>* mergelist;
cout << "C2: ";
C2->display();
cout << "sublist: ";
sublist.display();
mergelist = C2->merge(&sublist);
cout << "mergelist: ";
mergelist->display();
```

```
    cout << "\n\n*****end*****" << endl;
}
```

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.

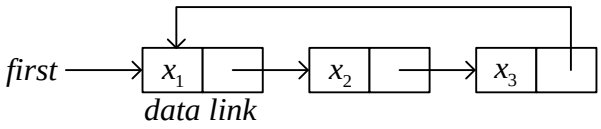


Fig. 4.14 A circular linked list

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.

```
*****testbench start*****
[test for InsertFront]
11 10 9 8 7 6 5 4 3 2 1 0

[test for InsertBack]
11 10 9 8 7 6 5 4 3 2 1 0 0 2 4 6 8 10 12 14 16 18 20 22

[test for Size() to count the node numbers]
Size of CL1 is: 24

[test for deleteFront]
CL1 before deletefront: 11 10 9 8 7 6 5 4 3 2 1 0 0 2 4 6 8 10 12 14 16 18 20 22
CL1 after deletefront 5 times: 6 5 4 3 2 1 0 0 2 4 6 8 10 12 14 16 18 20 22

[test for deleteBack]
CL1 before deleteback: 6 5 4 3 2 1 0 0 2 4 6 8 10 12 14 16 18 20 22
CL1 after deleteback 5 times: 6 5 4 3 2 1 0 0 2 4 6 8 10 12

[test for deconcatenate()]
CL1:
6 5 4 3
CL2:
2 1 0 0 2 4 6 8 10 12

[test for deleteAllOddNode()]
CL2 before delete odd node:
2 1 0 0 2 4 6 8 10 12
CL2 after delete odd node:
1 0 4 8 12

[test for merge()]
CL1: 6 5 4 3
CL2: 1 0 4 8 12
mergedlist: 6 1 5 0 4 4 3 8 12
*****test end*****
```

```cpp
int main() {
    CircularList<int> CL1;

    cout << "*****testbench start*****" << endl;

    cout << "[test for InsertFront]" << endl;
    for(int i = 0; i< 12; i++) {
        CL1.InsertFront(i);
    }
    CL1.display();

    cout << "\n[test for InsertBack]" << endl;
    for(int i = 0; i< 12; i++) {
        CL1.InsertBack(i*2);
    }
    CL1.display();

    cout << "\n[test for Size() to count the node numbers]" << endl;
    cout << "Size of CL1 is: " << CL1.Size() << endl;

    cout << "\n[test for deleteFront]" << endl;
    cout << "CL1 before deletefront: ";
    CL1.display();
    cout << "CL1 after deletefront 5 times: ";
    for(int i = 0 ; i < 5; i++) {
        CL1.DeleteFront();
    }
    CL1.display();

    cout << "\n[test for deleteBack]" << endl;
    cout << "CL1 before deleteback: ";
    CL1.display();
    cout << "CL1 after deleteback 5 times: ";
    for(int i = 0 ; i < 5; i++) {
        CL1.DeleteBack();
    }
    CL1.display();
```

```cpp
cout << "\n[test for deconcatenate()]" << endl;
CircularList<int>* CL2;
CL2 = CL1.deconcatenate(4);
cout << "CL1: " << endl;
CL1.display();
cout << "CL2: " << endl;
CL2->display();

cout << "\n[test for deleteAllOddNode()]" << endl;
cout << "CL2 before delete odd node: " << endl;
CL2->display();
CL2->deleteAllOddNode();
cout << "CL2 after delete odd node: " << endl;
CL2->display();

cout << "\n[test for merge()]" << endl;
CircularList<int>* mergeList;
cout << "CL1: ";
CL1.display();
cout << "CL2: ";
CL2->display();
mergeList = CL1.merge(CL2);
cout << "mergedlist: ";
mergeList->display();

cout << "*****test end*****";
```

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

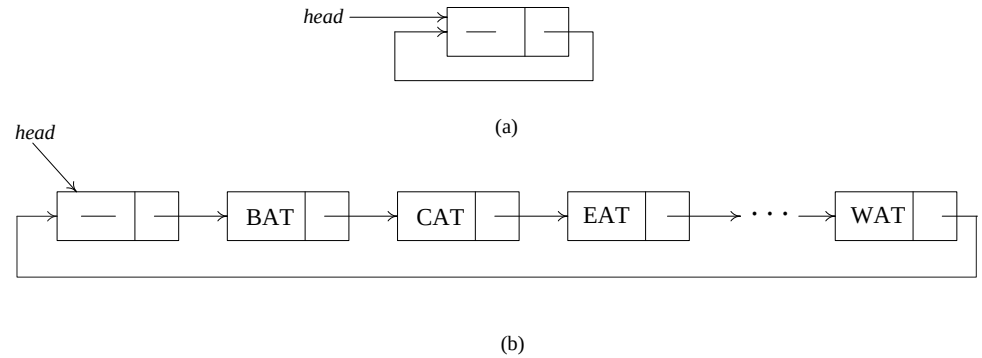


Figure 4.16 Circular list with a header node

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

(reference the code in the folder named part2_2_i where the header file is CL_with_head.h)

```
*****testbench start*****
[test for InsertFront]
11 10 9 8 7 6 5 4 3 2 1 0

[test for InsertBack]
11 10 9 8 7 6 5 4 3 2 1 0 0 2 4 6 8 10 12 14 16 18 20 22

[test for Size() to count the node numbers]
Size of CL1 is: 24

[test for deleteFront]
CL1 before deletefront: 11 10 9 8 7 6 5 4 3 2 1 0 0 2 4 6 8 10 12 14 16 18 20 22
CL1 after deletefront 5 times: 6 5 4 3 2 1 0 0 2 4 6 8 10 12 14 16 18 20 22

[test for deleteBack]
CL1 before deleteback: 6 5 4 3 2 1 0 0 2 4 6 8 10 12 14 16 18 20 22
CL1 after deleteback 5 times: 6 5 4 3 2 1 0 0 2 4 6 8 10 12

[test for deconcatenate()]
CL1:
6 5 4 3
CL2:
2 1 0 0 2 4 6 8 10 12

[test for deleteAllOddNode()]
CL2 before delete odd node:
2 1 0 0 2 4 6 8 10 12
CL2 after delete odd node:
1 0 4 8 12

[test for merge()]
CL1: 6 5 4 3
CL2: 1 0 4 8 12
mergedlist: 6 1 5 0 4 4 3 8 12
*****test end*****
```

The result is identical to the previous without head dummy node.

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.

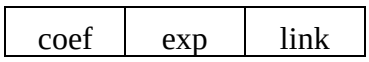
Reference the header file List.h, Queue.h, Stack.h in the folder part2_3

```
*****testBench Start*****
[test for InsertFront for List and Push for Queue and Stack
List: 19 18 17 16 15 14 13 12 11 10 -10 -9 -8 -7 -6 -5 -4 -3 -2 -1
Queue: 0 1 2 3 4 5 6 7 8 9
Stack: 9 8 7 6 5 4 3 2 1 0

[test for DeleteFront and DeleteBack for list and Pop for Queue and Stack]
List: 14 13 12 11 10 -10 -9 -8 -7 -6
Queue: 5 6 7 8 9
Stack: 4 3 2 1 0

[test for Front and Back for list and Top for Stack and Front for Queue]
Front of List: 14 Back of List: -6
Top of stack: 4
Front of Queue: 5

[test for Get for list]
L114 13 12 11 10 -10 -9 -8 -7 -6
The 5th element of the list is: 10
The 7th element of the list is: -9

[test for Delete for list]
L1 before delete the 5th element 14 13 12 11 10 -10 -9 -8 -7 -6
L1 after delete the 5th element 14 13 12 11 -10 -9 -8 -7 -6

[test for Insert for list]
L1 before Insert 666 as 8th element 14 13 12 11 -10 -9 -8 -7 -6
L1 after Insert 666 as 8th element 14 13 12 11 -10 -9 -8 666 -7 -6

[test of summation of speical mult]:
L1: 14 13 12 11 -10 -9 -8 666 -7 -6
14 * -9 = -126
13 * -8 = -104
12 * 666 = 7992
11 * -7 = -77
-10 * -6 = 60
Summation of special mult: 7745

*****end of testbench*****
```

```cpp
#include <iostream>
using namespace std;

int main() {

    List<int> L1;
    Queue<int> Q1; // derived from List
    Stack<int> S1; // derived from List

    cout << "\n*****testBench Start*****" << endl;
    cout << "[test for InsertFront for List and Push for Queue and Stack" << endl;

    for(int i = 0; i < 10; i++) {
        L1.InsertBack(i-10);
        L1.InsertFront(i+10);
        Q1.Push(i);
        S1.Push(i);
    }


    cout << "List: " << L1 << endl;
    cout << "Queue: " << Q1 << endl;
    cout << "Stack: " << S1 << endl;


    cout << "\n[test for DeleteFront and DeleteBack for list "
     << "and Pop for Queue and Stack]" << endl;
    for(int i = 0; i < 5; i++) {
        L1.DeleteFront();
        L1.DeleteBack();
        Q1.Pop();
        S1.Pop();
    }

    cout << "List: " << L1 << endl;
    cout << "Queue: " << Q1 << endl;
    cout << "Stack: " << S1 << endl;

    cout << "\n[test for Front and Back for list "
         << "and Top for Stack and Front for Queue]" << endl;

    cout << "Front of List: " << L1.Front() << " Back of List: " << L1.Back() << endl;
    cout << "Top of stack: " << S1.Top() << endl;
    cout << "Front of Queue: " << Q1.Front() << endl;

    cout << "\n[test for Get for list] " << endl;
    cout << "L1" << L1 << endl;
    cout << "The 5th element of the list is: " << L1.Get(5) << endl;
    cout << "The 7th element of the list is: " << L1.Get(7) << endl;
```

```cpp
cout << "\n[test for Delete for list] " << endl;
cout << "L1 before delete the 5th element " << L1 << endl;
L1.Delete(5);
cout << "L1 after delete the 5th element " << L1 << endl;

cout << "\n[test for Insert for list] "  << endl;
cout << "L1 before Insert 666 as 8th element " << L1 << endl;
L1.Insert(8, 666);
cout << "L1 after Insert 666 as 8th element " << L1 << endl;

cout << "\n[test of summation of speical mult]: " << endl;
cout << "L1: " << L1 << endl;
cout << L1.specialMult() << endl;




cout << "\n*****end of testbench*****" << endl;
```

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.



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.

```
*****Start testbench:*****

[Initializing polynomial P1 and P2 using overloading >> ]
P1:
3
2 3
4 5
-5 3
P2:
4
2 3
2 8
9 2
5 1

[test for copy constructor]:
P1: 4X^5 - 3X^3
copyPolynomial: 4X^5 - 3X^3

[test for assignment operator =]:
copyPolynomial: 4X^5 - 3X^3
P2: 2X^8 + 2X^3 + 9X^2 + 5X^1
~~~~~~copyPolynomial = P2~~~~~~
copyPolynomial: 2X^8 + 2X^3 + 9X^2 + 5X^1

[test for operator +]:
sumPoly: { 0, zero polynomial }
P1: 4X^5 - 3X^3
P2: 2X^8 + 2X^3 + 9X^2 + 5X^1
sumPoly = P1 + P2
sumPoly: 2X^8 + 4X^5 - 1X^3 + 9X^2 + 5X^1

[test for operator -]:
resPoly: { 0, zero polynomial }
P1: 4X^5 - 3X^3
P2: 2X^8 + 2X^3 + 9X^2 + 5X^1
resPoly = P1 - P2
resPoly:  - 2X^8 + 4X^5 - 5X^3 - 9X^2 - 5X^1

[test for operator *]:
multPoly: { 0, zero polynomial }
P1: 4X^5 - 3X^3
P2: 2X^8 + 2X^3 + 9X^2 + 5X^1
multPoly = P1 + P2
multPoly: 8X^40 - 6X^24 + 8X^15 + 36X^10 - 6X^9 - 27X^6 + 20X^5 - 15X^3

[test for evaluate]:

input a number for evaluation for multPoly m(x), x: 0.43
m(0.43) =  -1.06447

***** end testbench *****
```

Solution

Keep Practicing >

Show Steps

$8x^{40} - 6x^{24} + 8x^{15} + 36x^{10} - 6x^9 - 27x^6 + 20x^5 - 15x^3, x = 0.43: \quad -1.06447\ldots$

Steps

For $8x^{40} - 6x^{24} + 8x^{15} + 36x^{10} - 6x^9 - 27x^6 + 20x^5 - 15x^3$ substitute x with 0.43

$= 8 \cdot 0.43^{40} - 6 \cdot 0.43^{24} + 8 \cdot 0.43^{15} + 36 \cdot 0.43^{10} - 6 \cdot 0.43^9 - 27 \cdot 0.43^6 + 20 \cdot 0.43^5 - 15 \cdot ($

Show Steps

$8 \cdot 0.43^{40} - 6 \cdot 0.43^{24} + 8 \cdot 0.43^{15} + 36 \cdot 0.43^{10} - 6 \cdot 0.43^9 - 27 \cdot 0.43^6 + 20 \cdot ($

$= -1.06447\ldots$

```cpp
int main() {

    cout << "*****Start testbench:*****" << endl;

    cout << "\n[Initializing polynomial P1 and P2 using overloading << ]" << endl;
    Polynomial P1;
    cout << "P1:" << endl;
    cin >> P1;

    Polynomial P2;
    cout << "P2:" << endl;
    cin >> P2;

    cout << "\n[test for copy constructor]: " << endl;
    cout << "P1: " << P1 << endl;
    Polynomial copyPoly(P1);
    cout << "copyPolynomial: " << copyPoly << endl;

    cout << "\n[test for assignment operator =]: " << endl;
    cout << "copyPolynomial: " << copyPoly << endl;
    cout << "P2: " << P2 << endl;
    cout << "~~~~~~copyPolynomial = P2~~~~~~" << endl;
    copyPoly = P2;
    cout << "copyPolynomial: " << copyPoly << endl;


    cout << "\n[test for operator +]: " << endl;
    Polynomial sumPoly;
    cout << "sumPoly: " << sumPoly << endl;
    cout << "P1: " << P1 << endl;
    cout << "P2: " << P2 << endl;
    cout << "sumPoly = P1 + P2 " << endl;
    sumPoly = P1 + P2;
    cout << "sumPoly: " << sumPoly << endl;
```

```cpp
    cout << "\n[test for operator -]: " << endl;
    Polynomial resPoly;
    cout << "resPoly: " << resPoly << endl;
    cout << "P1: " << P1 << endl;
    cout << "P2: " << P2 << endl;
    cout << "resPoly = P1 - P2 " << endl;
    resPoly = P1 - P2;
    cout << "resPoly: " << resPoly << endl;


    cout << "\n[test for operator *]: " << endl;
    Polynomial multPoly;
    cout << "multPoly: " << multPoly << endl;
    cout << "P1: " << P1 << endl;
    cout << "P2: " << P2 << endl;
    cout << "multPoly = P1 + P2 " << endl;
    multPoly = P1 * P2;
    cout << "multPoly: " << multPoly << endl;

    cout << "\n[test for evaluate]: " << endl;
    cout << "\ninput a number for evaluation for multPoly m(x), x: ";
    double x;
    cin >> x;
    cout << "m(" << x << ") =  " << multPoly.evaluate(x) << endl;

    cout << "\n***** end testbench *****" << endl;
```

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.

```
input for M1:
3 3 4
0 0 2
1 0 5
1 2 4
2 1 6

input for M2:
3 4 4
0 1 1
1 2 3
2 1 2
2 2 2

input for M3:
3 3 4
0 1 1
1 0 4
1 1 2
2 2 8
M1:
  2  0  0
  5  0  4
  0  6  0

M2:
  0  1  0  0
  0  0  3  0
  0  2  2  0

M3:
  0  1  0
  4  2  0
  0  0  8

SumMatrix = M1 + M3:
  2  1  0
  9  2  4
  0  6  8

Multmatrix = M1 * M2:
  0  2  0  0
  0 13  8  0
  0  0 18  0

[test for time of copy constructor of multiplication of Summatrix and Multmatrix]

the testing time result is: 3.03767e-06s

[test for transpose of the MultMatrix]
before transpose:
  0  2  0  0
  0 13  8  0
  0  0 18  0

after transpose:
  0  0  0
  2 13  0
  0  8 18
  0  0  0

end
```

```cpp
#include "Matrix.h"
#include <time.h>

int main() {
    Matrix M1;
    Matrix M2;
    Matrix M3;

    cout << "input for M1:" << endl;
    cin >> M1;
    cout << "\ninput for M2:" << endl;
    cin >> M2;
    cout << "\ninput for M3:" << endl;
    cin >> M3;


    cout << "M1:\n" << M1 << endl;
    cout << "M2:\n" << M2 << endl;
    cout << "M3:\n" << M3 << endl;

    Matrix SumMatrix(M1+M3);
    Matrix MultMatrix(M1*M2);
    cout << "SumMatrix = M1 + M3:\n" << SumMatrix << endl;
    cout << "Multmatrix = M1 * M2:\n" << MultMatrix << endl;

    cout << "[test for time of copy constructor of multiplication of Summatrix and Multmatrix]" << endl;
    clock_t start, stop;
    double duration = 0;
    int counter = 0;
    do {
        counter++;
        start = clock();
        Matrix CopyMatrix(SumMatrix*MultMatrix);
        stop = clock();
        duration += stop - start;
    }while (duration < 10);
    cout << "\nthe testing time result is: " << duration / counter / double(CLOCKS_PER_SEC) << "s" << endl << endl;

    cout << "[test for transpose of the MultMatrix]" << endl;
    cout << "before transpose:\n" << MultMatrix << endl;
    Matrix TransposeM(MultMatrix.transpose());
    cout << "after transpose:\n" << TransposeM << endl;


    cout << "end" << endl;
}
```