Q9. Linked List

For this problem, you need to know how to implement a linked list.

We will upload our new test case on 5/20, you must download it on before demo.

We will provide you a file (Lab9_template.cpp) and a test case (test.txt). You need to do all TODO parts in the file.

A linked list consists of nodes. Each node contains an element, and a pointer which points to its next neighbor.

We provide a structure called Node:

```
struct Node
{
    int value;
    Node* next;

    Node() : value(0), next(NULL) {};
    Node(int x) : value(x), next(NULL) {};
};
```

value is the value stored in the node.
next is the pointer point to its neighbor.

We also provide a class called LinkedList:

```
class LinkedList
{
private:
    Node* Head;
public:
    LinkedList();
    ~LinkedList();
    void Push_back(int x);
    void Push_front(int x);
    void Insert(int index, int x);
    void Delete(int index);
    void Reverse();
    void Print();
    void Clear();
};
```

Class explanation:
Members:
Node* head: the node in the first place.

Methods:

- void Push_back(int *x*) : Insert a node to the end of the linked list with value *x*.
  If the list is empty, you need to assign the node to be a head
  - E.g.:
    List1: 2 -> 4 -> 6 -> 8 -> null
    List1.Push_back(10)

List1: 2 -> 4 -> 6 -> 8 -> 10 -> null

- void Push_front(int *x*) : Insert a node to the front of the linked list with value *x*.
  If the list is empty, you need to assign the node to be a head
  - E.g.:
    List1: 3 -> 5 -> 7 -> 9 -> null
    List1.Push_front(1)
    List1: 1 -> 3 -> 5 -> 7 -> 9 -> null

- void Insert(int *index* ,int *x*) : Insert a node to the linked list at position *index* with value *x*.
  Note: The index of the first node is 0. If the list is empty, you need to assign the node to be a head
  - E.g.:
    List1: 2 -> 4 -> 8 -> 10 -> null
    List1. Insert(2,6)
    List1: 2 -> 4 -> 6 -> 8 -> 10 -> null
    List1. Insert(0,0)
    List1: 0 -> 2 -> 4 -> 6 -> 8 -> 10 -> null

- void Delete(int index) : Remove the node with index "*index*" in the linked list.
  - E.g.:
    List1: 1 -> 5 -> 9 -> 13 -> 20 -> null
    List1.Delete(3)
    List1: 1 -> 5 -> 9 -> 20 -> null
    List1.Delete(0)
    List1: 5 -> 9 -> 20 -> null

- void Reverse() : Reverse all elements in the linked list.
  - E.g.:
    List1: 1 -> 3 -> 5 -> 7 -> 9 -> null
    List1. Reverse()
    List1: 9 -> 7 -> 5 -> 3 -> 1 -> null

- void Print() : Print all the elements in the linked list in order.
  If the list is empty, you need to print "The list is empty."
  - E.g.:
    List1: 9 -> 7 -> 5 -> 3 -> 1 -> null
    List1.Print()
    List: 9 7 5 3 1

- Void Clear() : Clear all nodes in the linked list:
  - E.g.:
    List1: 1 -> 3 -> 5 -> 7 -> 9 -> null
    List1.Clear()
    List1: (The list is empty)

Note

You can assume that Insert() and Delete() will only do legal calculations. That is, the index must be an integer greater than or equal to 0, and it won't exceed the length of the list
You must use template to do this lab.

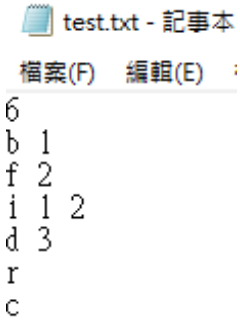
Hint

Linked list:
Teacher's slide "20_LinkedList_PriorityQueue_voice_Eng" p.2 ~ p.70

Input Format

We will provide a text file. The file includes several commands

The first number represents numbers of operations.
The rest are commands, please refer to the table 1.



Commands	Explanation
b {value}	Push_back(value)
f {value}	Push_front(value)
i {index} {value}	Insert(index, value)
d {index}	Delete(index)
r	Reverse()
c	Clear()

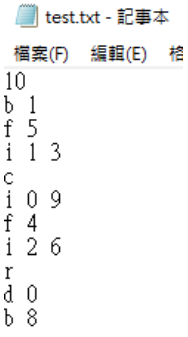
Table 1

Output Format

You must print all operation and the content of linked list in order after doing each command.
See more detail from Sample output.

Sample Input & Output.

Input:



Output: