Chapter 4 鏈結串列
(Linked List)

鏈結串列(linked list)

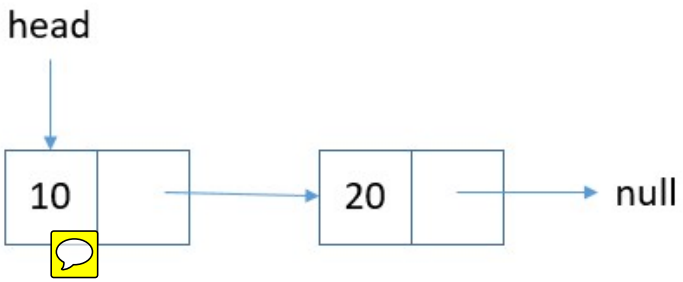
- 鏈結串列是一種線性串列，串列中的每一個節點裡除了存資料外，也存到下一個節點的指標(Pointer)，透過指標將節點連接起來。
- 為何使用鏈結串列(linked list)？
  - 為了避免以陣列方式來存放資料時，在插入(insert)或刪除(delete)某一節點所遇到的困難。
  - 節省配置的記憶體空間。
- 鏈結串列 vs. 陣列：
  - 在加入和刪除時利用指標(pointer)或參考(reference)，因此比陣列來得簡單。
  - 鏈結串列在搜尋上所花費的時間會比陣列來得久。

X[0]	X[1]
10	20

單向鏈結串列的節點

•假設鏈結串列中每個節點有data及指向下一個節點的指標(next)，若將節點結構定義為Node 型態，則宣告的方式如下：

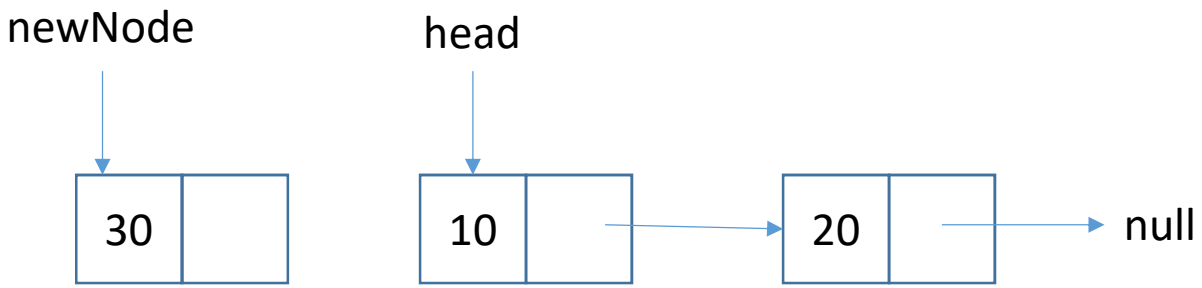
```
class Node{
        private int data;
        private Node next;
}
```

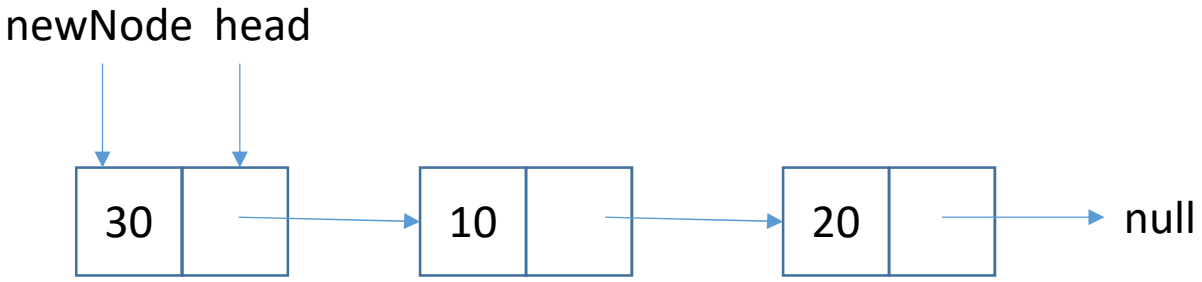


單向鏈結串列的ADT

```
public abstract class AbstractLinkedList {
    private Node head;
    public abstract Node findANode(int data);
    public abstract void displyAllNode();
    public abstract boolean isEmpty();
    public abstract void insertAtFirstNode(Node newNode);
    public abstract void removeFirstNode();
    public abstract void insertNode(Node curNode, Node
    newNode);//在curNode新增一newNode
    public abstract int removeNode(int data);
    public abstract void insertAtLastNode(Node newNode);
    public abstract void removeLastNode();
    public abstract void concatenate(IntLinkedList b); //結
    合兩個串列
    public abstract void inverse(); //反轉串列
    public abstract int length(); //串列的節點個數
}
```

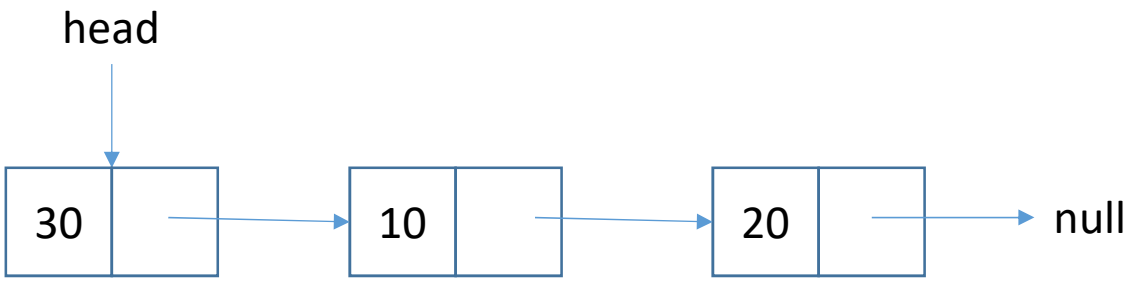
insertAtFirstNode(Node newNode)

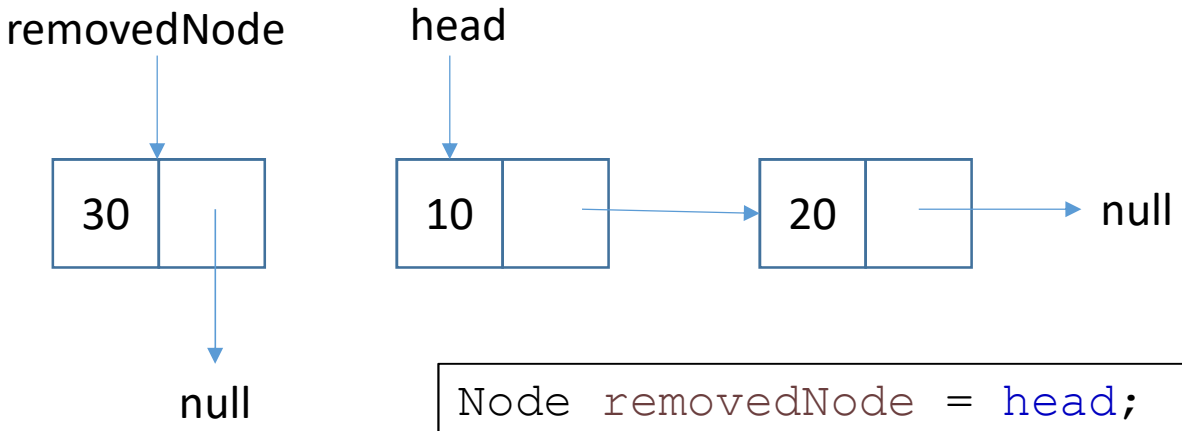





```
newNode.next = head;
head = newNode;
```

removeFirstNode()

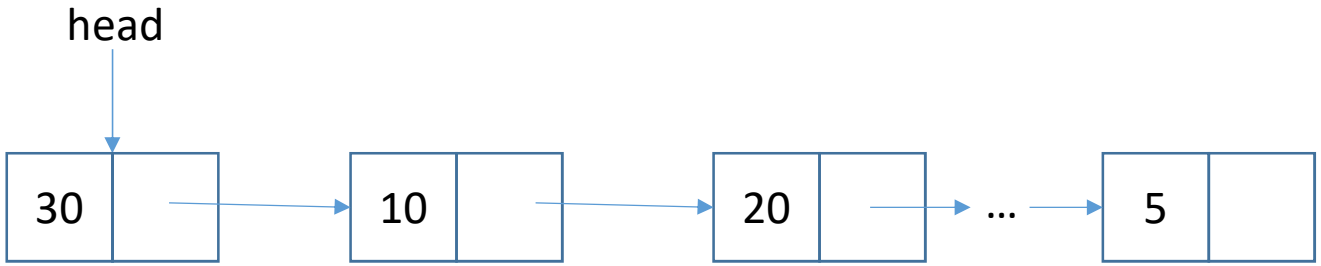





```
Node removedNode = head;
head = head.next;
removedNode.next = null;
```

insertAtLastNode(Node newNode);

- 由first找到最後一個Node，再insert在最後一個Node後
- 增加一個member last，指向最後一個Node



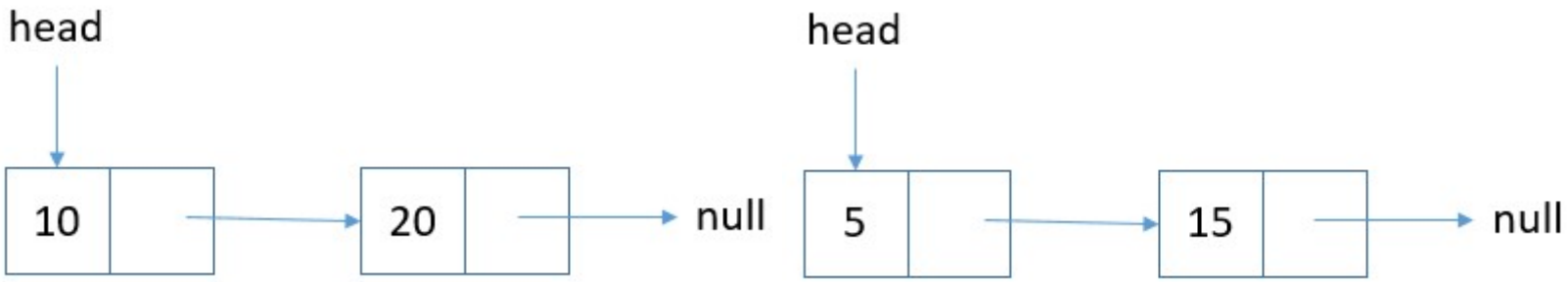

```
Node curNode = this.head;
while(curNode.next!=null){
        curNode = curNode.next;
}
curNode.next = newNode;
```

練習一

- 請實作單向鏈結串列的ADT
  - insertAtFirstNode(Node newNode)
  - removeFirstNode()
  - **`insertAtLastNode(Node newNode)`**

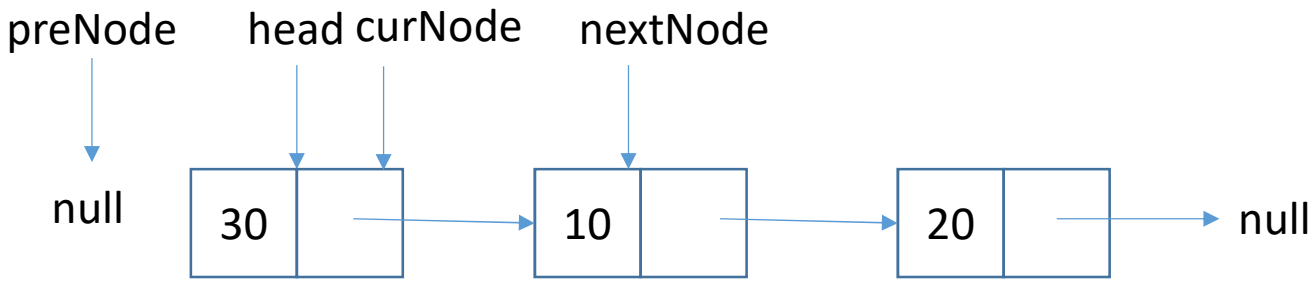
concatenate(LinkedList b)

```
public void concatenate(LinkedList b) {
    if(isEmpty()){
        this.head=b.head;
        return;
    }
    if(!b.isEmpty())
    {
        Node curNode = head;
        while(curNode.next!=null){
                curNode = curNode.next;
        }
        curNode.next = b.head;
    }
}
```



inverse()

```
public void inverse() {
    if(!isEmpty()){
        Node preNode = null;
        Node nextNode = head.next;
        Node curNode = head;
        while(nextNode!=null){
            curNode.next = preNode;
            preNode = curNode;
            curNode = nextNode;
            nextNode = nextNode.next;
        }
        curNode.next = preNode
    }
}
```



練習二

- 請實作單向鏈結串列的ADT
  - **`concatenate(LinkedList b)`**
  - **`inverse()`**

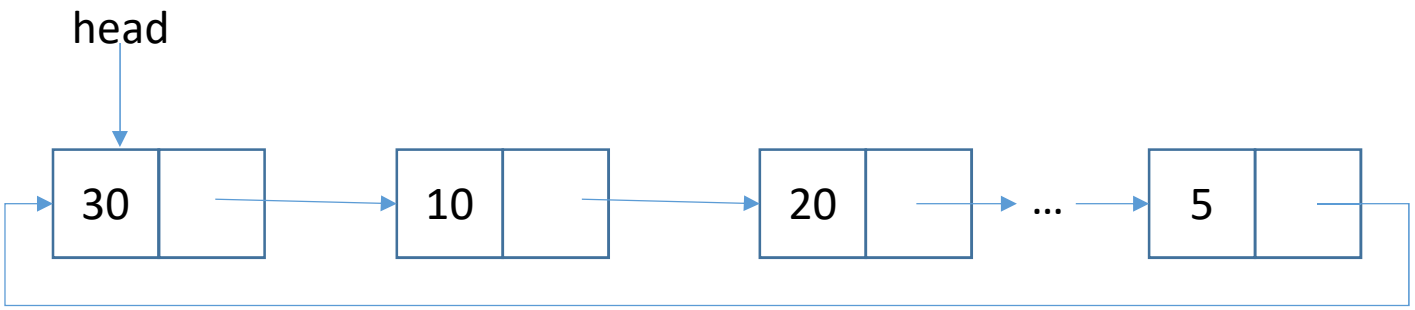
作業04-01

- 請實作單向鏈結串列ADT的所有方法

```
public abstract class CircularLinkedList {
    private Node head;
    public abstract void displyAllNode(); //依串列順序印出結點值
    public abstract boolean isEmpty();
    public abstract void insertAtFirstNode(Node newNode);
    public abstract void removeFirstNode();
    public abstract void insertNode(Node curNode, Node
    newNode);//在curNode新增一newNode
    public abstract void insertAtLastNode(Node newNode);
    public abstract void removeLastNode();
    public abstract void concatenate(IntLinkedList b); //結合兩個
    串列
    public abstract void inverse(); //反轉串列
    public abstract int length(); //串列的節點個數
}
```

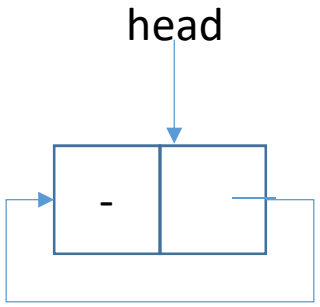
環狀鏈結串列 (circular list)(1)

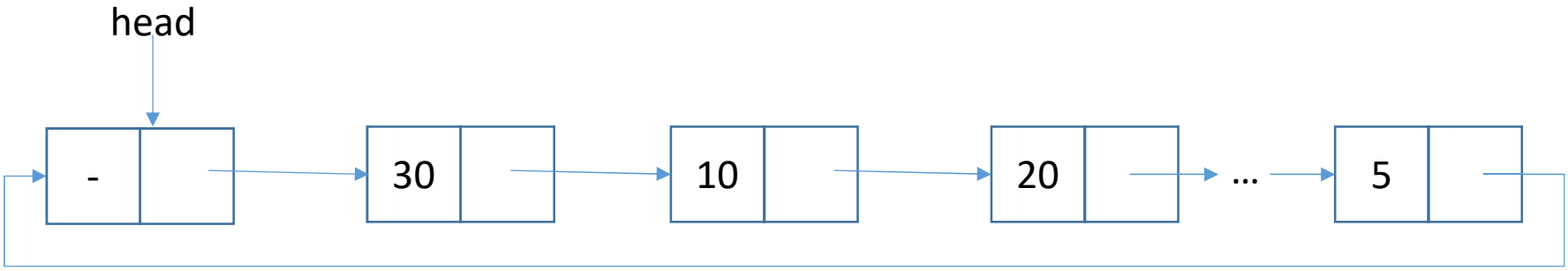
- 若將單向鏈結串列的最後一個節點的next指標，指向第一個節點時，則稱此串列為環狀串列
  - 本來`curNode.`next = null代表已到最後一個節點，但circular list改為`curNode.next`= head代表已到最後一個節點



環狀鏈結串列 (circular list)(2)

- 使用第一種方式製作circular list，當為空串列會出現問題，head是null, head.next會出現錯誤，所以增加一個dummy Node，稱為head Node

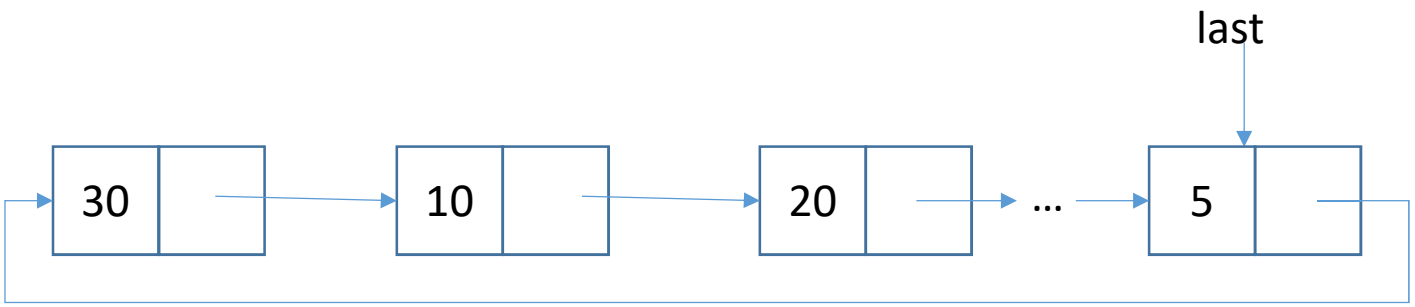




環狀鏈結串列 (circular list)(3)

- 使用一個last指標，指向最後的節點，last.next指向第一個節點
  - 新增一個節點在串列的最後變得容易

```
newNode.next = last.next;
last.next = newNode;
last = newNode;
```



作業04-02

- 請實作環狀鏈結串列ADT的所有方法

```
public abstract class CircularLinkedList {
    private Node head; // or last
    public abstract void displyAllNode(); //依串列順序印出結點值
    public abstract boolean isEmpty();
    public abstract void insertAtFirstNode(Node newNode);
    public abstract void removeFirstNode();
    public abstract void insertNode(Node curNode, Node
    newNode);//在curNode新增一newNode
    public abstract void insertAtLastNode(Node newNode);
    public abstract void removeLastNode();
    public abstract void concatenate(IntLinkedList b); //結合兩個
    串列
    public abstract void inverse(); //反轉串列
    public abstract int length(); //串列的節點個數
}
```

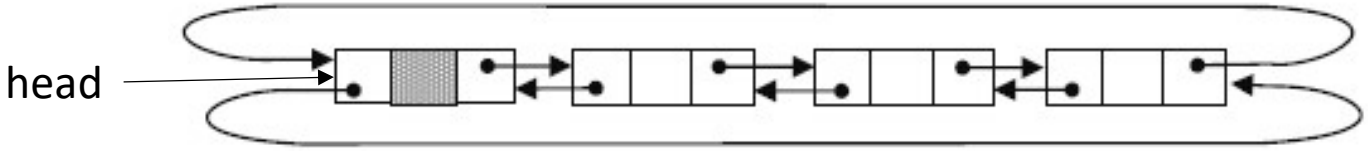
雙向鏈結串列(1)

- 雙向鏈結串列(doubly linked list)乃是每個節點皆具有三個欄位，一為左鏈結(LLINK)，二為資料(DATA)，三為右鏈結(RLINK)，其資料結構如下：


```
public class DbListNode{
    public DbListNode llink;
    public DbListNode rlink;
    public int data;
    DbListNode(int data){
        this.data = data;
        this.llink = this;
        this.rlink = this;
    }
}
```

雙向鏈結串列(2)

- 其中LLINK 指向前一個節點，而RLINK 指向後一個節點。通常在雙向鏈結串列加上一個串列首，此串列首的資料欄不存放資料。如下圖所示：

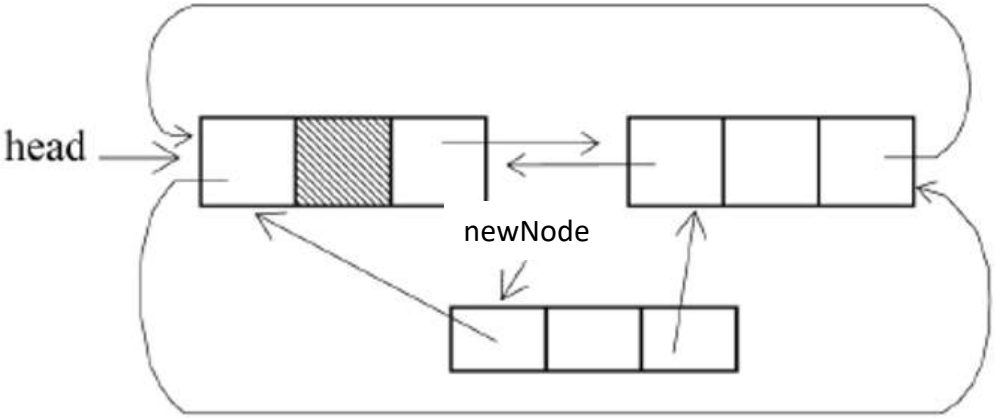


雙向鏈結串列具有下列兩點特性：

1. 假設curNode 是任何節點的指標，則：
   - curNode = curNode.llink.rlink = curNode.rlink.llink;
2. 若此雙向鏈結串列是空串列，則只有一個串列首。

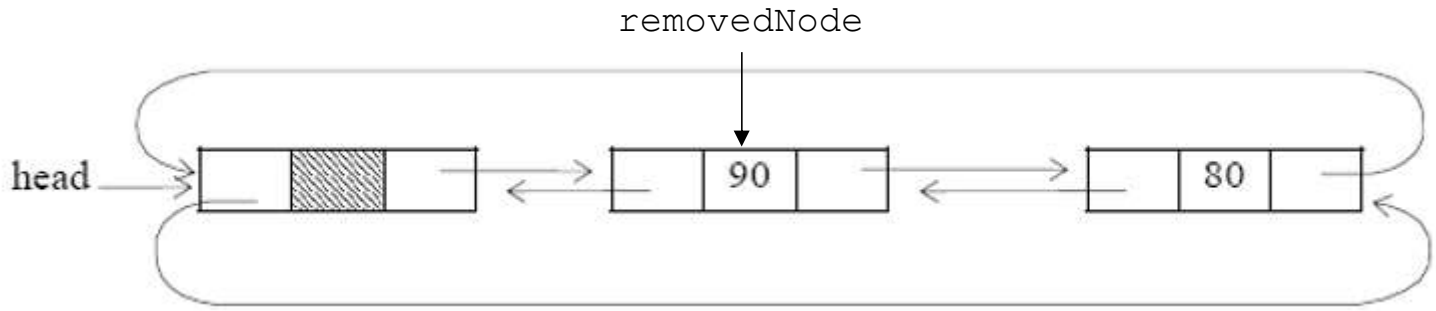
雙向鏈結串列- 新增節點在前端

```
public void insertAtFirstNode(DbListNode newNode){
    newNode.rlink = head.rlink;
    newNode.llink = head;
    head.rlink.llink = newNode;
    head.rlink = newNode;
}
```



雙向鏈結串列- 刪除前端節點

```
public void removeFirstNode(){
    if(isEmpty()){
        System.out.println("此串列為空串列，沒有節點可刪除!");
    }
    else
    {
        DbListNode removedNode = head.rlink;
        head.rlink = removedNode.rlink;
        removedNode.rlink.llink = head;
        removedNode.rlink=removedNode;
        removedNode.llink = removedNode;
    }
}
```



作業04-03

- 請實作雙向鏈結串列的以下方法：

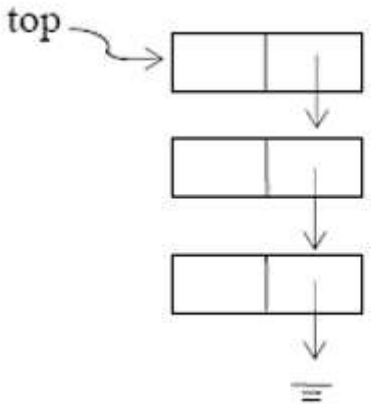
```
public abstract class DBLinkedList {
    private DbListNode head;
    public abstract void displyAllNode();
    public abstract boolean isEmpty();
    public abstract void insertAtFirstNode(DbListNode newNode);
    public abstract void removeFirstNode();
}
```

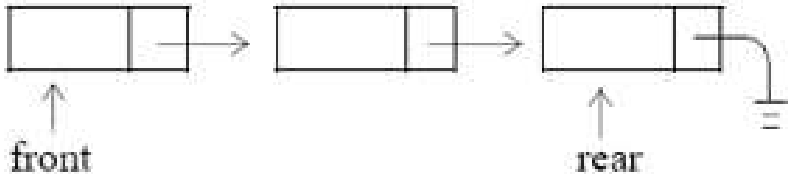
進階作業04-01

- 請實作雙向鏈結串列的所有方法

```
public abstract class DBLinkedList {
    private DbListNode head;
    public abstract void displyAllNode();
    public abstract boolean isEmpty();
    public abstract void insertAtFirstNode(DbListNode newNode);
    public abstract void removeFirstNode();
    public abstract void insertNode(DbListNode curNode,
    DbListNode newNode);//在curNode新增一newNode
    public abstract void insertAtLastNode(DbListNode newNode);
    public abstract void removeLastNode();
    public abstract void concatenate(DbListNode b); //結合兩個串列
    public abstract void inverse(); //反轉串列
    public abstract int length(); //串列的節點個數
}
```

鏈結串列的應用 –
以鏈結串列表示堆疊及佇列

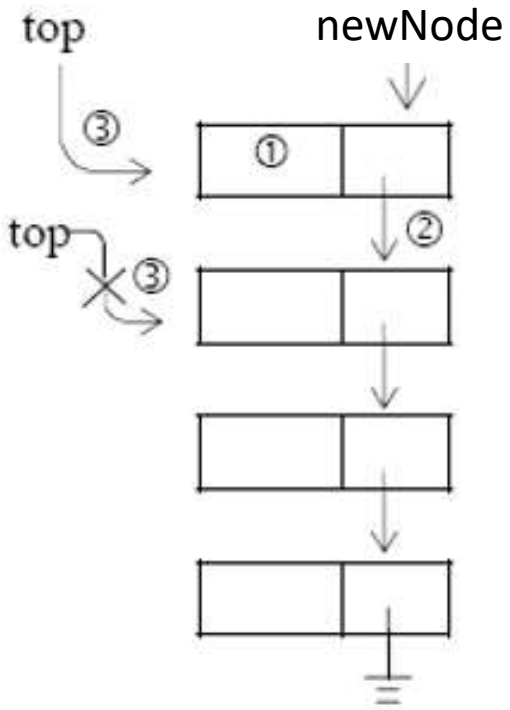





```
public class ChainNode {
    public int data;
    public ChainNode next;
    ChainNode(int data){
        this.data = data;
        this.next = null;
    }
}
```

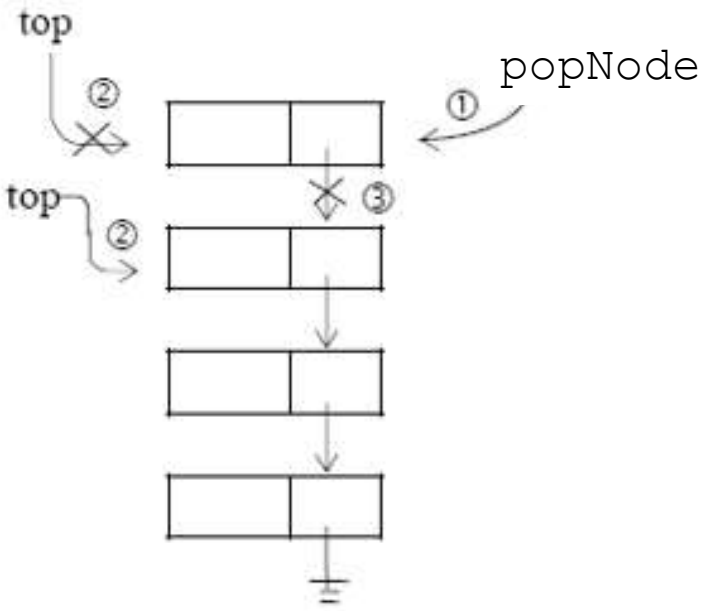
```
public class Stack {
    private ChainNode top;
    public boolean isEmpty();
    public void push(ChainNode newNode);
    public ChainNode pop();
    public ChainNode top();
    //public ChainNode front();
    //public ChainNode rear();
}
```

鏈結串列的應用 –
加入物件至堆疊 push(ChainNode newNode)




```
public void push(ChainNode newNode){
    newNode.next = top;
    top = newNode;
}
```

鏈結串列的應用 –
由堆疊取出物件 ChainNode pop()



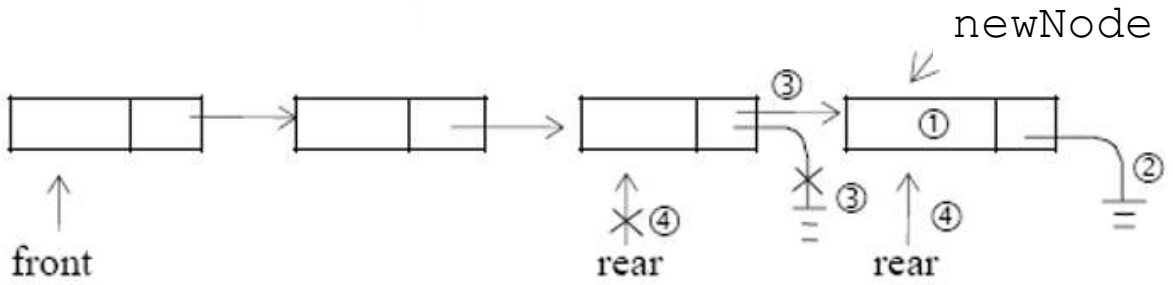

```
public ChainNode pop(){
    if(isEmpty())
        return null;
    ChainNode popNode = top;
    top = top.next;
    popNode.next=null;
    return popNode;
}
```

作業04-04

- 請使用單向鏈結串列實作Stack Abstract data type

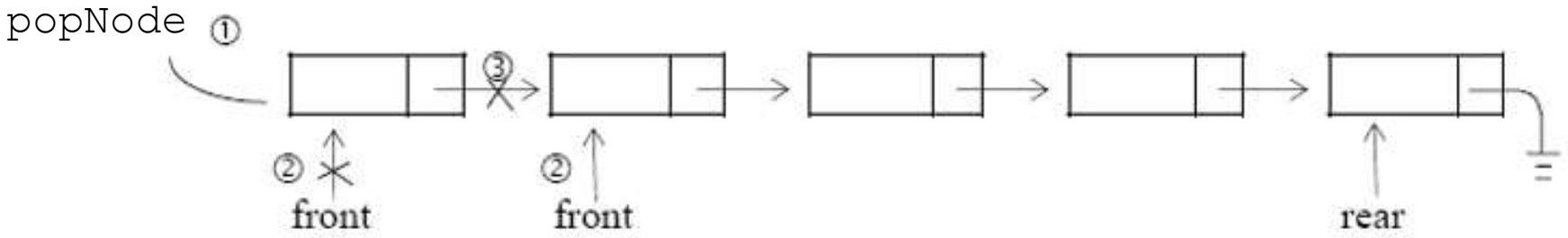
```
public class Stack {
    private ChainNode top;
    public boolean isEmpty();
    public void push(ChainNode newNode);
    public ChainNode pop();
    public ChainNode top();
}
```

鏈結串列的應用 –
加入物件至佇列 push(ChainNode newNode)




```
public void push(ChainNode newNode){
    if(front==null){
        front = newNode;
        rear=newNode;
    }
    else
    {
        rear.next = newNode;
        rear = newNode;
    }
}
```

鏈結串列的應用 –
由佇列取出物件 ChainNode pop()




```
public ChainNode pop()
{
    if(isEmpty())
        return null;
    ChainNode popNode = front;
    if(front==rear){ //The link just has one node.
        front = null;
        rear = null;
    }
    else
    {
        front = front.next;
        popNode.next = null;
    }
    return popNode;
```

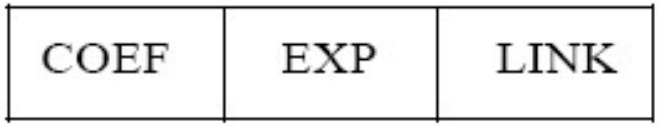
作業04-05

- 請使用單向鏈結串列實作Queue Abstract Data Type

```
public class Queue {
    private ChainNode top;
    public boolean isEmpty();
    public void push(ChainNode newNode);
    public ChainNode pop();
    public ChainNode front();
    public ChainNode rear();
}
```

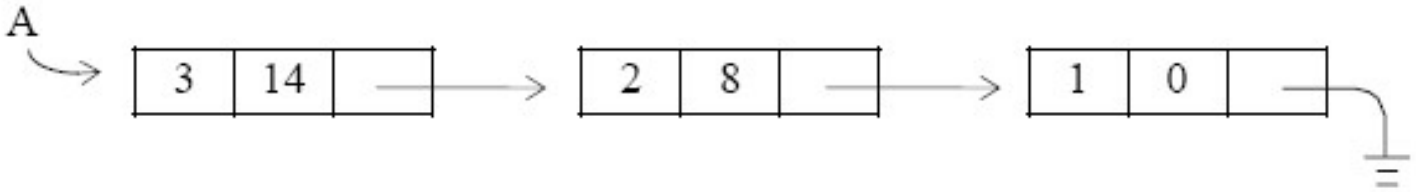
鏈結串列的應用 -以鏈結串列表示多項式(1)

多項式相加可以利用鏈結串列來完成。多項式以鏈結串列的資料結構如下：



COEF 是變數的係數，EXP 為變數的指數，而 LINK 為指向下一節點的指標。

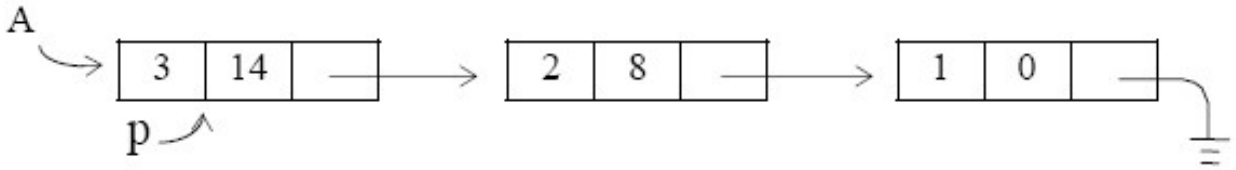
假設有一多項式 $A = 3x^{14} + 2x^{8} + 1$，以鏈結串列表示如下：

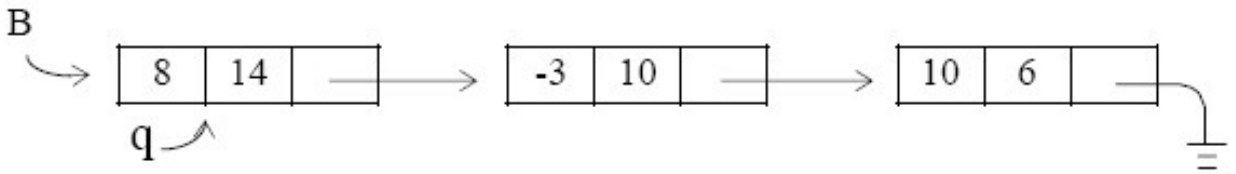



```
public class TermNode{
    int coef;
    int exp;
    TermNode next;
    TermNode (int ncoef,int nexp){
        this.coef=ncoef;
        this.exp=nexp;
        this.next=null;
    }
}
```

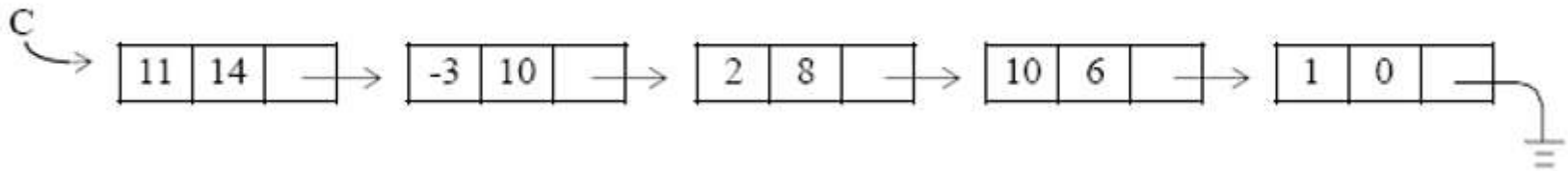
鏈結串列的應用 -以鏈結串列表示多項式(2)

$A = 3x^{14} + 2x^{8} + 1$ ， $B = 8x^{14} - 3x^{10} + 10x^{6}$





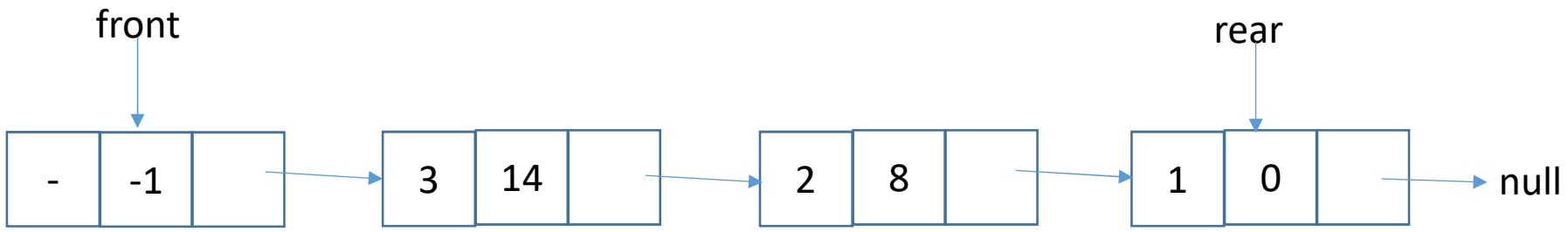
$C = 11x^{14} - 3x^{10} + 2x^{8} + 10x^{6} + 1$



演算法與array方式相同

鏈結串列的應用 -多項式相加實作(1)

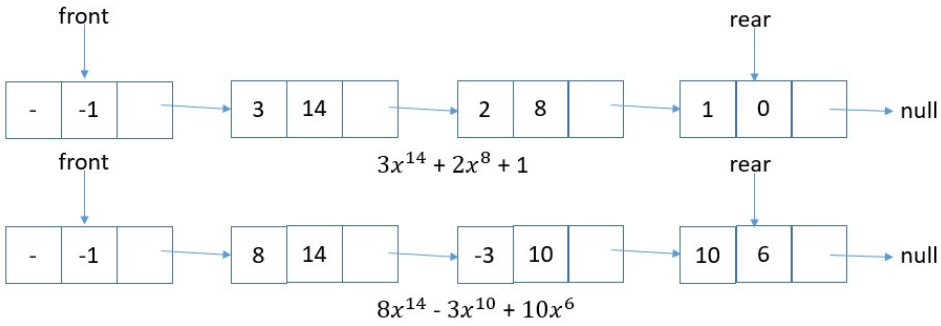
```
public class Polynomial {
    private TermNode front;
    private TermNode rear;
    public Polynomial(){
        TermNode dummyNode = new TermNode (-1, -1);
        front = dummyNode;
        rear = dummyNode;
    }
    public void addATermNode(TermNode newNode){
        this.rear.next = newNode;
        this.rear = newNode;
    }
    public static Polynomial add(TermNode A, TermNode B){
          …
    }
}
```



$3x^{14} + 2x^{8} + 1$

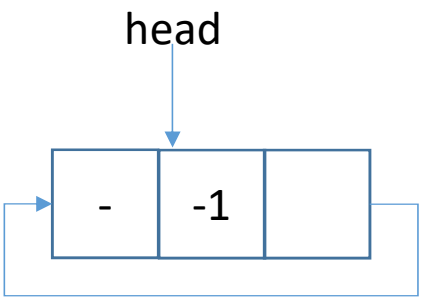
鏈結串列的應用 -多項式相加實作(2)

```
public static Polynomial add(TermNode A, TermNode B){
      TermNode newNode;
      Polynomial C = new Polynomial();
      A = A.next;
      B = B.next;
      while(A!=null && B!=null){
            if(A.exp==B.exp){
                  int t = A.coef+B.coef;
                  newNode = new TermNode(t, A.exp);
                  C.addATermNode(newNode);
                  A = A.next;
                  B = B.next;
            }
            else if(A.exp<B.exp)
            {
                  int t = B.coef;
                  newNode = new TermNode(t, B.exp);
                  C.addATermNode(newNode);
                  B = B.next;
            }
            else {
                  int t = A.coef;
                  newNode = new TermNode(t, A.exp);
                  C.addATermNode(newNode);
                  A = A.next;
            }
      }
```

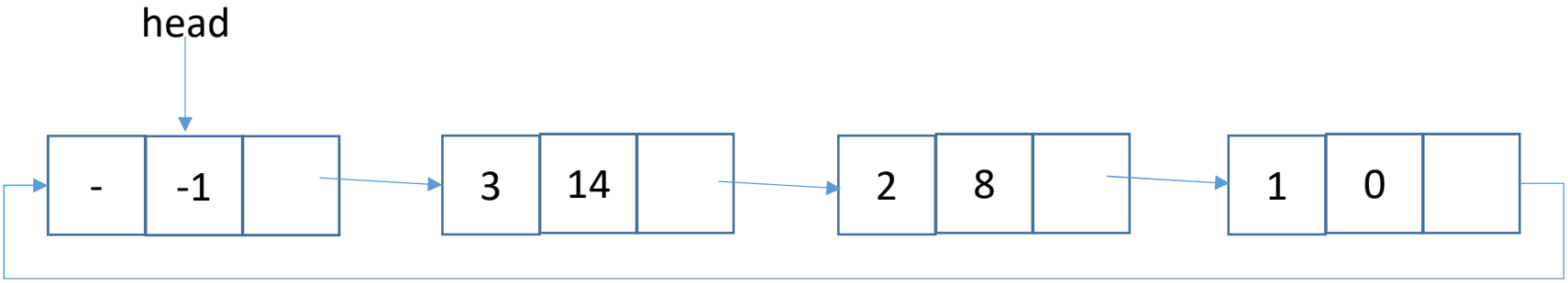



```
      for(;A!=null;A=A.next){
            int t = A.coef;
            newNode = new TermNode(t, A.exp);
            C.addATermNode(newNode);
      }
      for(;B!=null;B=B.next){
            int t = B.coef;
            newNode = new TermNode(t, B.exp);
            C.addATermNode(newNode);
      }
      return C;
}
```

鏈結串列的應用 -多項式的環形串列表示



Zero polynomial



$3x^{14} + 2x^{8} + 1$

作業04-06

- 請使用環形鏈結串列實作多項式相加