

204700

Data Structure and Programming Languages

Jakarin Chawachat

LINKED LIST

Linked List

- เป็นโครงสร้างข้อมูลแบบหนึ่งซึ่งประกอบด้วยสมาชิกหลายตัว โดยที่สมาชิกเหล่านั้นจะถูกเชื่อมต่อกันเป็นสายยาว
- ไม่ต้องกำหนดขนาดไว้ล่วงหน้า -> ทำให้รับข้อมูลได้เรื่อยๆ ยืดหยุ่นสูง
- มีส่วน **link** เชื่อมต่อระหว่างสมาชิก
- แต่ละสมาชิกจะเรียกว่า โหนด (**Node**)

Array Vs Linked List



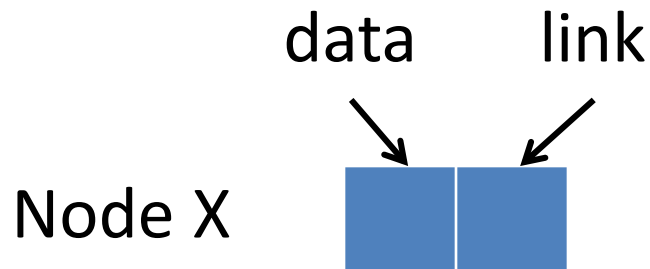
Array



Linked List

Node

- คือพื้นที่ในหน่วยความจำ ซึ่งประกอบด้วย อย่างน้อยสองฟิลด์คือ **data** และ **link**
- ส่วน **data** ทำหน้าที่เก็บข้อมูลต่างๆ
- ส่วน **link** ทำหน้าที่เก็บตำแหน่งที่อยู่ของโหนดถัดไป



```
public class Node {  
    public int data1;  
    public double data2;  
  
    public Node link;  
}
```

- การเข้าถึง **Linked List** จะต้องมี **pointer** ชี้ไปยังโหนดแรกเสมอ เรียกว่า **“Header”**
- โหนดสุดท้ายเป็นโหนดที่ไม่มีการชี้ไปที่โหนดอื่นใดอีก ซึ่งเป็นการบอกให้จบ **Linked List**
- **Linked List** ที่ไม่มีโหนดอยู่ภายใน เรียกว่า **“Empty List”** หรือ **“Null List”**

Singly Linked List

- แต่ละ node จะมี link เพียง 1 link เท่านั้น
- เป็น **Linked List** แบบทิศทางเดียว คือ การดำเนินการใดๆ กับ โหนดจะต้องเดินทางจากโหนดเริ่มต้นไปยังโหนดสุดท้าย เดินกลับไม่ได้



Create Linked List

```
public class LinkedList {  
    Node header;  
  
    public LinkedList() {  
        header = null;  
    }  
}
```

Operators

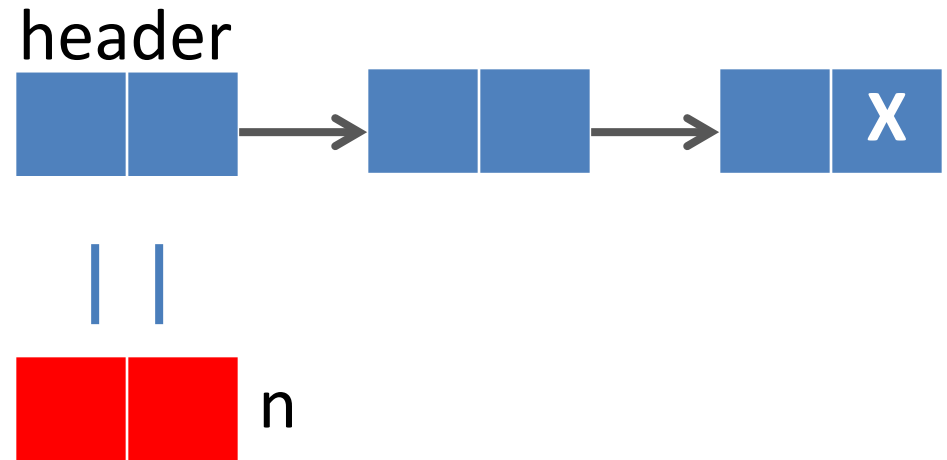
- isEmpty():boolean ตรวจสอบว่า **list** ว่างหรือไม่
- getLength():integer นับจำนวน **node** ใน **list**
- insertFirst(Data) แทรก **node** ตำแหน่งแรก
- find(i) ค้นหาข้อมูลของ **node** ลำดับที่ **i**
- insertLast(Data) แทรก **node** ตำแหน่งสุดท้าย
- insertPosition(i,Data) แทรก **node** ตำแหน่งที่ **i**
- removeFirst() ลบตำแหน่งแรก
- removePosition(i) ลบตำแหน่งที่ **i**

isEmpty()

```
public boolean isEmpty() {  
    if(header == null)  
        return true;  
    else  
        return false;  
}
```

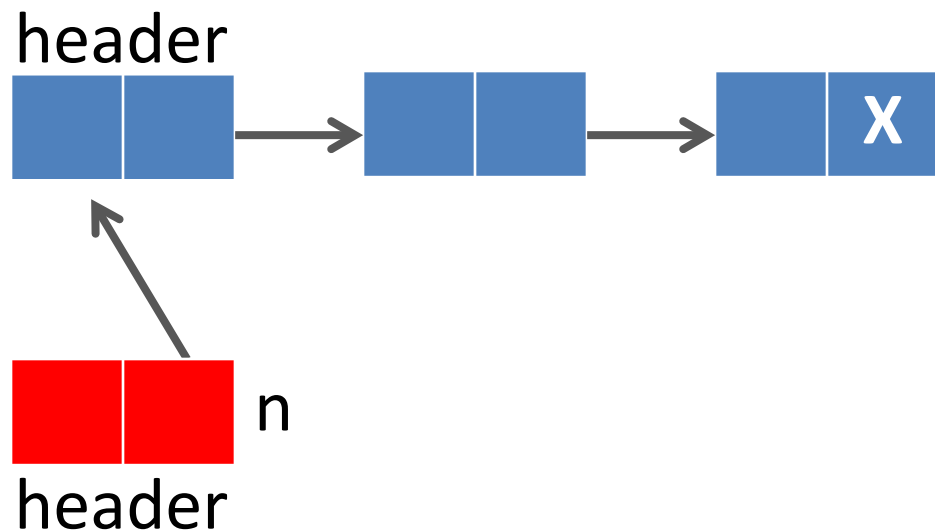
getLength()

```
public int getLength() {  
    int count;  
    Node n = new Node();  
    n = header;  
    if (header == null)  
        count = 0;  
    else{  
        count = 1;  
        while (n.link != null) {  
            n = n.link;  
            count++;  
        }  
    }  
    return count;  
}
```



insertFirst(Data)

```
public void insertFirst(int d1, double d2) {  
    Node n = new Node(d1, d2);  
    n.link = header;  
    header = n;  
}
```



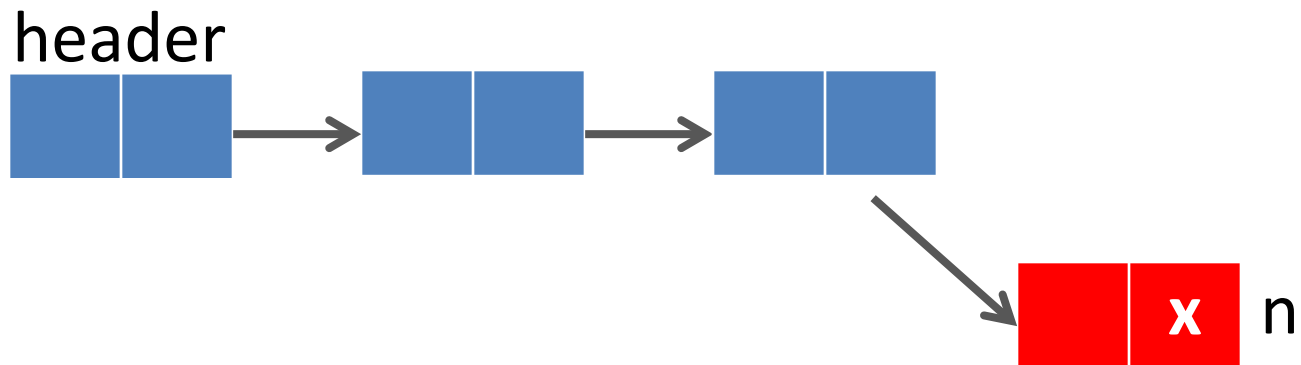
find(index)

```
public void find(int index){  
    if((index>=1)&&(index<=getLength())){  
        Node currentNode = new Node();  
        currentNode=header;  
        for(int i=1;i<index;i++){  
            currentNode=currentNode.link;  
        }  
        currentNode.showNodeInfo();  
    }else{  
        System.out.println("Index overflow");  
    }  
}
```



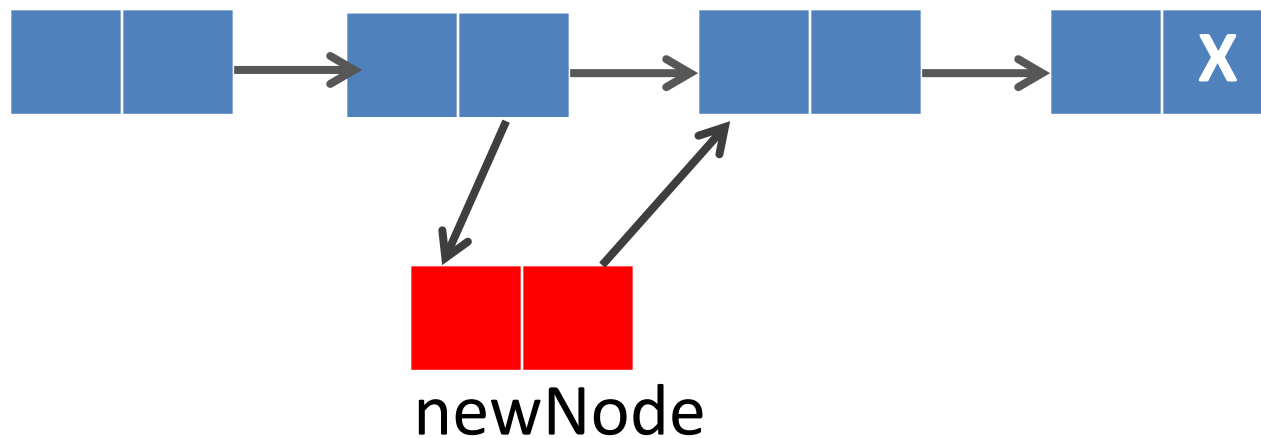
insertLast(Data)

- TRY!!



insertPosition(i,Data)

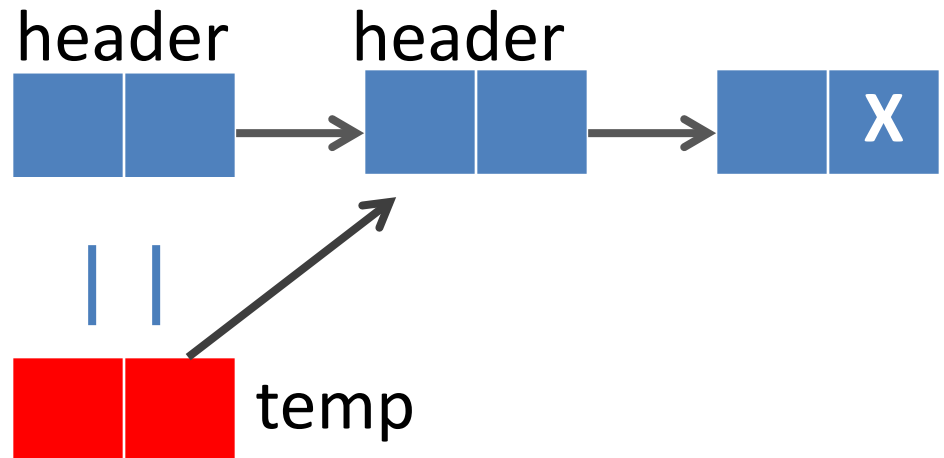
- TRY!!
- Hint
- Find ก่อน แล้วค่อยแทรก




```
public void insertPosition(int index,int d1, double d2){
    if((index>=1)&&(index<=getLength()+1)){
        Node currentNode = new Node();
        currentNode= this.find(index-1);
        Node newNode = new Node(d1,d2);
        newNode.link=currentNode.link;
        currentNode.link=newNode;
    }else{
        System.out.print("Cannot insert.");
    }
}
```

removeFirst()

```
public void removeFirst(){  
    Node temp = new Node();  
    temp=header;  
    header = temp.link;  
}
```



removePosition()

