05506006 Data Structure

Lecture 6: Linked Lists (ต่อ) and Double Link Lists

ดร. รุ่งรัตน์ เวียงศรีพนาวัลย์

Outline

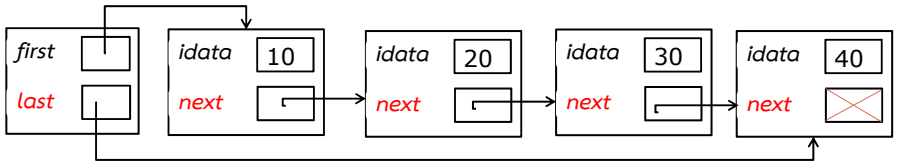
- LinkList
  - การสร้างลิสต์
  - การค้นหา
  - การเพิ่ม
    - การเพิ่มต้นลิสต์
    - การเพิ่มท้ายลิสต์
    - การเพิ่มระหว่างลิสต์
      - การเพิ่มแบบต่อท้ายโหนด
      - การเพิ่มแบบแทรกหน้าโหนด
  - การลบ
    - การลบต้นลิสต์
    - การลบท้ายลิสต์
    - การลบโหนดที่ตำแหน่งนั้นๆ
- Double LinkList
  - การสร้างลิสต์
  - การค้นหา
  - การเพิ่ม
    - การเพิ่มต้นลิสต์
    - การเพิ่มท้ายลิสต์
    - การเพิ่มระหว่างลิสต์
      - การเพิ่มแบบต่อท้ายโหนด
      - การเพิ่มแบบแทรกหน้าโหนด
  - การลบ
    - การลบต้นลิสต์
    - การลบท้ายลิสต์
    - การลบโหนดที่ตำแหน่งนั้นๆ

Operation พื้นฐานบน Link List

การสร้างลิสต์

- การท่องไปในลิสต์ (Traversing the list)
- การเพิ่มสมาชิกลงไปในลิสต์ (Inserting an item in the list)
  - การเพิ่มที่ต้นลิสต์
  - การเพิ่มไปที่ท้ายลิสต์
  - *การแทรกโหนดลงไปในลิสต์*
- การลบสมาชิกออกจากลิสต์ (Deleting an item from the list)
  - การลบสมาชิกตัวแรกในลิสต์
  - การลบสมาชิกตัวสุดท้ายในลิสต์
  - การลบ Node ใดๆ จากลิสต์
- การหาขนาดของลิสต์
- การค้นหาข้อมูลในลิสต์
- การวิเคราะห์ Big O

การหาขนาดของ ลิสต์

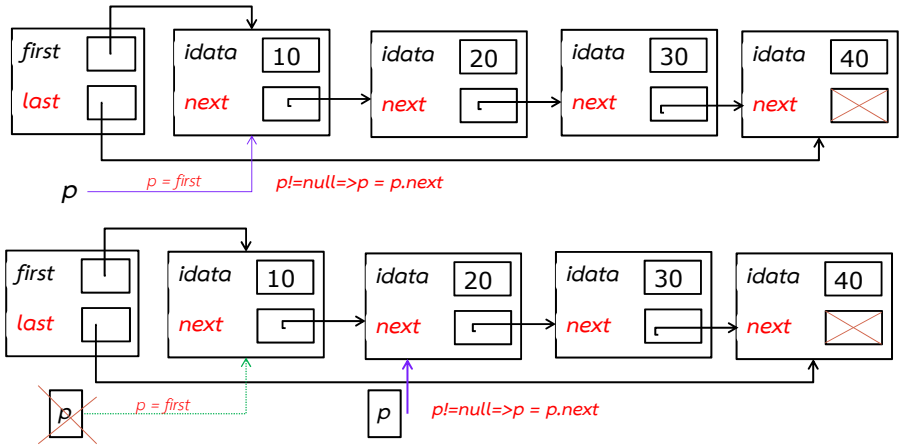


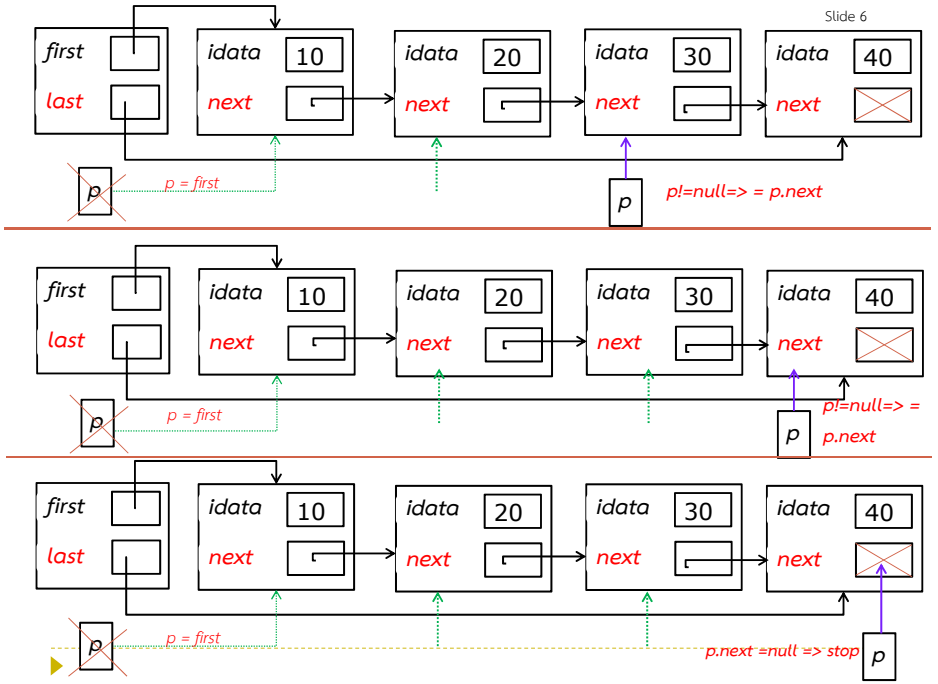
เหมือนกับการท่องไปในลิสต์แต่จะมีการเพิ่มการนับจำนวนเข้าไปทีละหนึ่งทุกครั้งที่วิ่งผ่านโหนดแต่ละโหนดในลิสต์

1. สร้างตัววิ่ง (p)ซึ่งมีชนิดข้อมูลเป็นชนิดข้อมูลเดียวกับโหนด (Node) และให้ค่าตัว นับ (count) เป็น 0
2. ให้ตัววิ่งเริ่มวิ่งจากต้นลิสต์
3. เมื่อวิ่งไปถึงโหนดใด

   ~~จะนำข้อมูลในโหนดนั้นมาปริ้นท์~~ => เพิ่มค่า count
4. ถ้ายังมีโหนดถัดไปจะเลื่อนไปที่โหนดถัดไป
5. วนลูปทำข้อ 3 จนกว่าตัววิ่งวิ่งไปครบทุกโหนดในลิสต์

การหาขนาดของลิสต์





Pseudo Code และ Code ของการหาขนาดของลิสต์

1. Check
   if list is empty
      ปรินท์ ลิสต์ว่าง ไม่มีข้อมูล
2. Else list ไม่ว่าง
   2.1 สร้างตัววิ่ง p มีชนิดเป็น Node
      ให้ค่า count = 0
   2.2 ให้ p = first;
   2.3 while p!=null
         count++;
      p = p.next
      end while
   2.5 print count

```
public void sizeList()
{
  if (first==null) {
     System.out.println("List is Empty");
  }
  else
  {  Node p;   p = first;
     count =0;
     while (p!=null)
     {
          count++;
          p = p.next;
     }
     System.out.println("This is has "+
count+" items.");
   }
}
```


การค้นหาข้อมูลในลิสต์ว่า โหนดใดมีข้อมูลเท่ากับ ค่าที่ต้องการหา

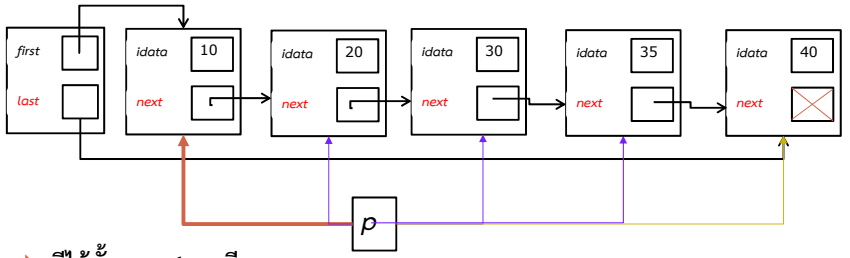
- การวิเคราะห์
- ผลลัพธ์ที่ได้จะมีอยู่ 3 กรณี
  - 1. ลิสต์ว่างไม่มีสมาชิก
  - 2. ลิสต์ไม่ว่าง
    - 2.1 พบข้อมูลที่ต้องการหา
    - 2.2 ไม่พบข้อมูลที่ต้องการหา
  - ใช้วิธีการคล้ายกับการหาข้อมูลของโหนดที่อยู่ตำแหน่งที่ i ในลิสต์ แต่แทนที่จะใช้ค่า count จะทำการตรวจสอบว่า idata ของโหนดนั้นเท่ากับค่าที่ต้องการหาหรือไม่ เมื่อเท่าคือพบแล้วไม่ต้องวน loop อีก การออกจากลูปทำได้สองรูปแบบ
    - จะมีการใช้คำสั่ง break ออกมาจาก loop ถ้าพบค่านั้นแล้ว
    - มีการเซตตัวแปรว่า พบแล้ว
- ผลลัพธ์ที่ได้ ขึ้นอยู่กับคำถามว่าจะ ให้ตอบเป็นโหนด หรือ แค่ว่าพบแล้วหรือไม่
  - ถ้าตอบเป็น node return เป็น node
  - ถ้าตอบเป็น พบหรือไม่พบ จะ return เป็น boolean
- หมายเหตุ คีย์ต้องมีชนิดข้อมูลเป็นชนิดข้อมูลเดียวกับ idata

Pseudo Code และ Code ของการหาว่าโหนดใดมีข้อมูลเท่ากับ ค่าที่ต้องการหา ใช้ break

```
1.      Check
2.      if  list is empty
        ปรินท์ ลิสต์ว่าง ไม่มีข้อมูล
        return null กรณีให้ retrun ตำแหน่งของโหนด
    Else list ไม่ว่าง
    2.1 สร้างตัววิ่ง p มีชนิดเป็น     Node
        ให้ค่าที่ต้องการหาคือคีย์ key
    2.2 ให้ p = first;
    2.3  ถ้ายังไม่หมดลิสต์ while p!=null
            ทำการเปรียบเทียบว่าค่าในโหนดนั้นเท่ากับ key หรือไม่
            ถ้าเท่าออกจากลูปเลย
            ถ้าไม่เท่าเลื่อนไปที่โหนดถัดไป
        end while
    2.5 ถ้า p ชี้ไปที่ null (p==null)     (แสดงว่าไม่พบ)
            ปรินท์ (The list item is less than i)
            return null
        Else
            return(p)  กรณีให้ return ตำแหน่งของโหนด
จบการทำงาน
```

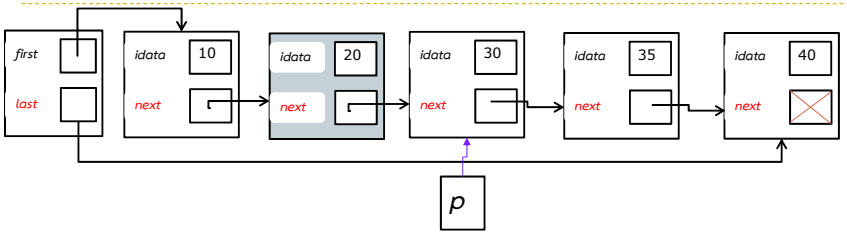
```
public Node findNode (int key) // public int findData(int key)
{    if (front==null)
     {  System.out.println("List Empty");
          return null;  }
     else
     {       Node p;

             p  = first;
             while (p!=null)
             {    if (p.idata==key)
                       break;
                  else p = p.next;
             }
             if (p==null)
             {  System.out.println("Cannot Find" + key);
                return null; return -1; }
             else
             {    return p;  return p.idata;}
     }
}
```


การลบข้อมูลที่ตำแหน่ง p



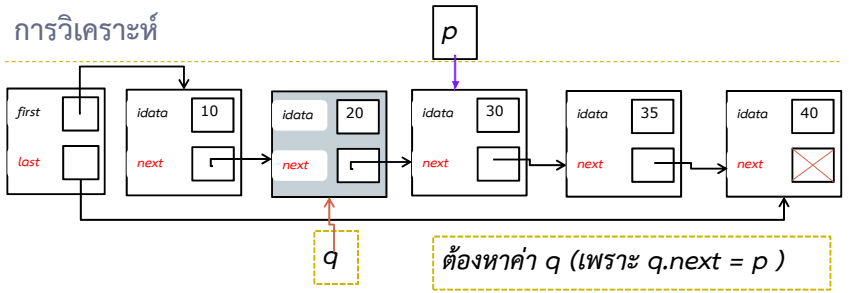
- มีได้ทั้งหมด 4 กรณี
  - 1. ลิสต์ว่าง
  - 2. p คือ โหนดตัวแรก - เมื่อเช็คได้ว่า p ชี้ไปที่โหนดแรกสามารถทำ deleteFirst ได้เลย
  - 3. p คือ โหนดตัวสุดท้าย - เมื่อเช็คได้ว่า p ชี้ไปที่โหนดสุดท้ายสามารถทำ deleteLast ได้เลย
  - 4. p คือ โหนดใดๆ ที่อยู่ระหว่าง ตัวแรกและตัวสุดท้าย
    กรณีนี้ต้องคิดเพิ่ม

การลบข้อมูลที่ตำแหน่ง p



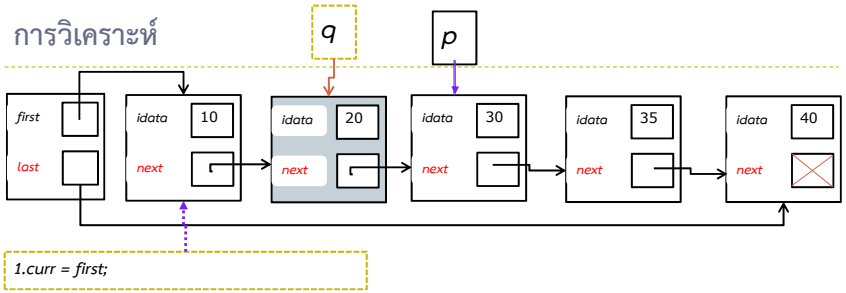
- ให้ p ชี้ไปที่ตำแหน่ง ที่ idata = 30
- การลบ p จาก ลิสต์จะต้องทราบค่า Node ก่อนหน้า p (คล้ายกับการลบโหนด Last ออกจาก ลิสต์ ต้องทราบตำแหน่งของโหนดก่อนโหนด Last)
- <u>คำถาม</u> ทำอย่างไร
- <u>Hint</u> ให้ q เป็น node ก่อนหน้า p => q.next = p
- ทำเหมือนกับการหา node ก่อน last

การวิเคราะห์

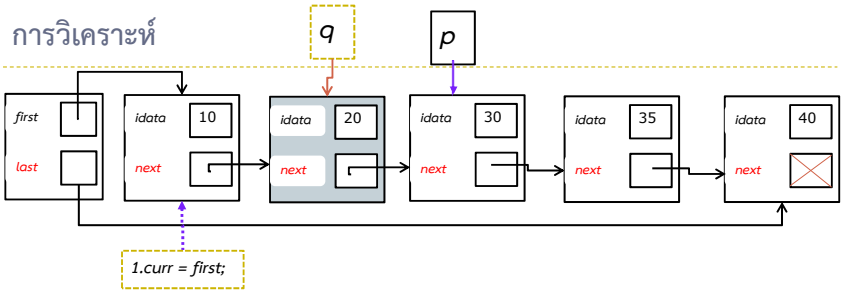


- หลักการ
  - ให้สร้างโหนด โหนดหนึ่งขึ้นมาเป็นตัววิ่ง ให้โหนดนั้นชื่อ curr
  - โดย curr จะเริ่มวิ่งจาก ต้น list (จาก first) *(curr = first)*
  - ให้สร้างโหนดอีกโหนดชื่อ q ไว้วิ่งตาม curr (curr จะอยู่ตำแหน่งถัดจาก q เสมอ)
  - curr จะวิ่งไปจนกว่าจะเจอ p while
  - เมื่อ curr เจอ p จะได้ node ก่อนหน้า p คือ q

การวิเคราะห์

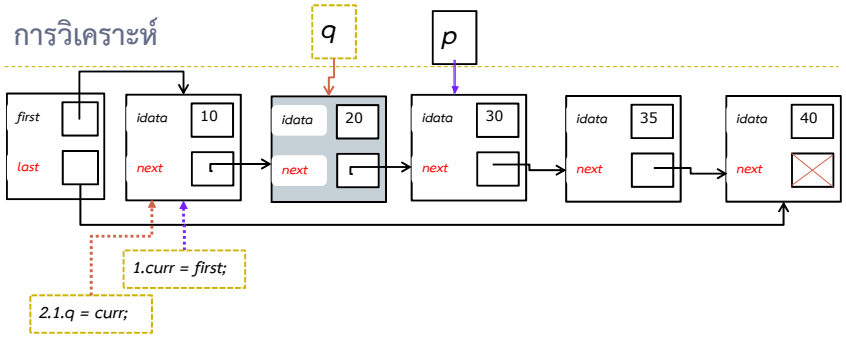


การวิเคราะห์



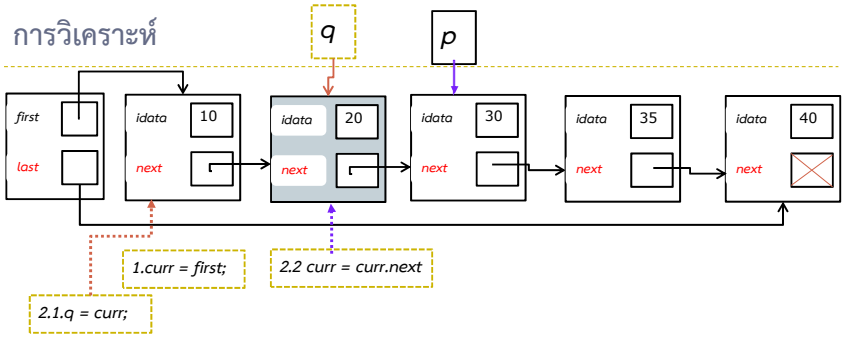
2.curr!=p
=> 2.1 q = curr
2.2 curr=curr.next

การวิเคราะห์



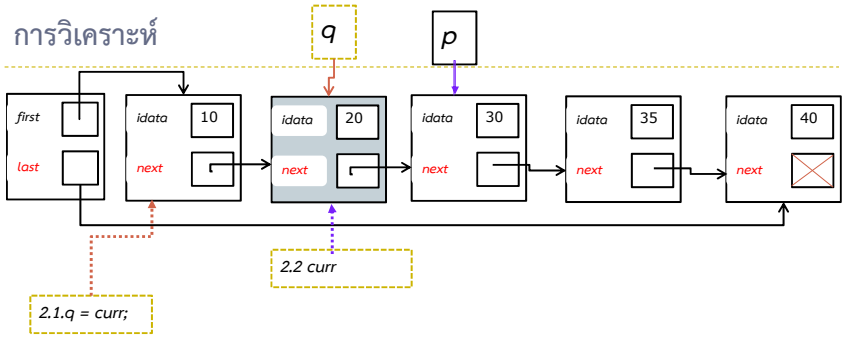
2.curr!=p
=> 2.1 q = curr
2.2 curr=curr.next

การวิเคราะห์



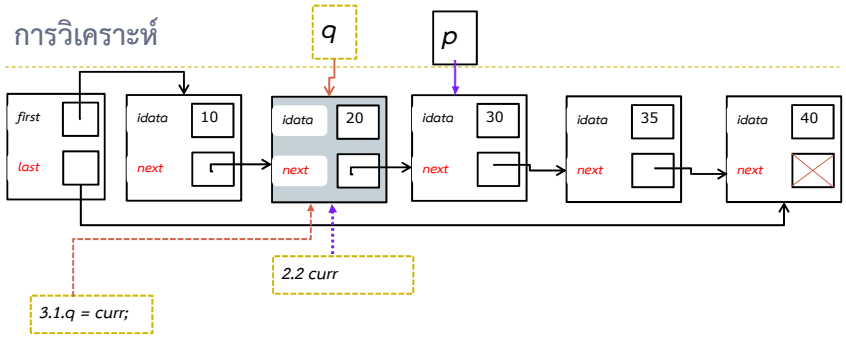
2.curr!=p
=> 2.1 q = curr
2.2 curr=curr.next

การวิเคราะห์



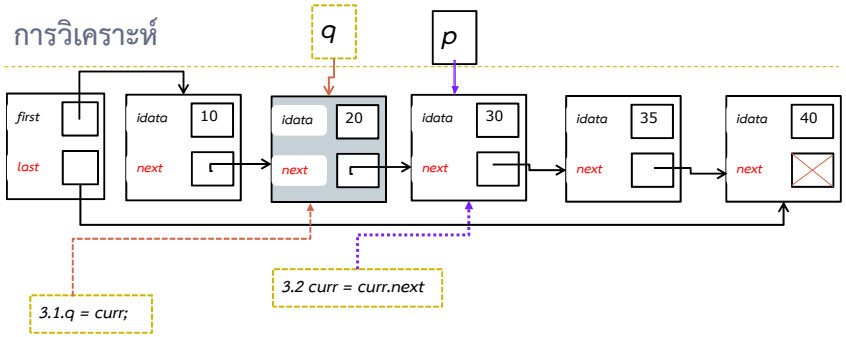
3.curr!=p
=> 3.1 q = curr
3.2 curr=curr.next

การวิเคราะห์



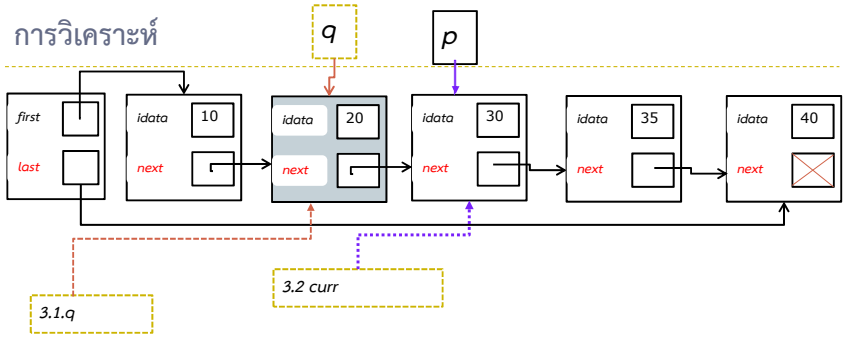
3.curr!=p
=> 3.1 q = curr
3.2 curr=curr.next

การวิเคราะห์



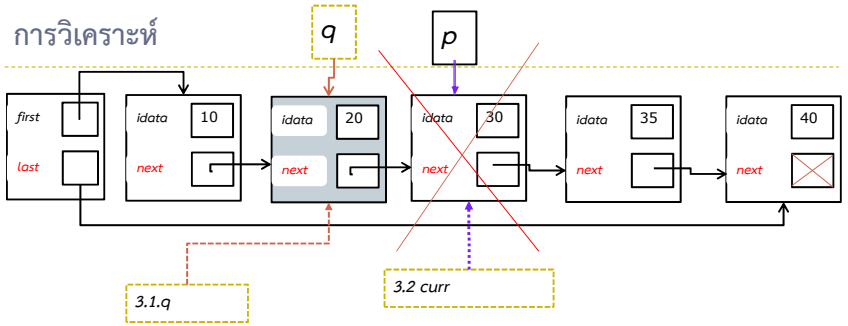
3.curr!=p
=> 3.1 q = curr
3.2 curr=curr.next

การวิเคราะห์



4.curr==p
=> exit

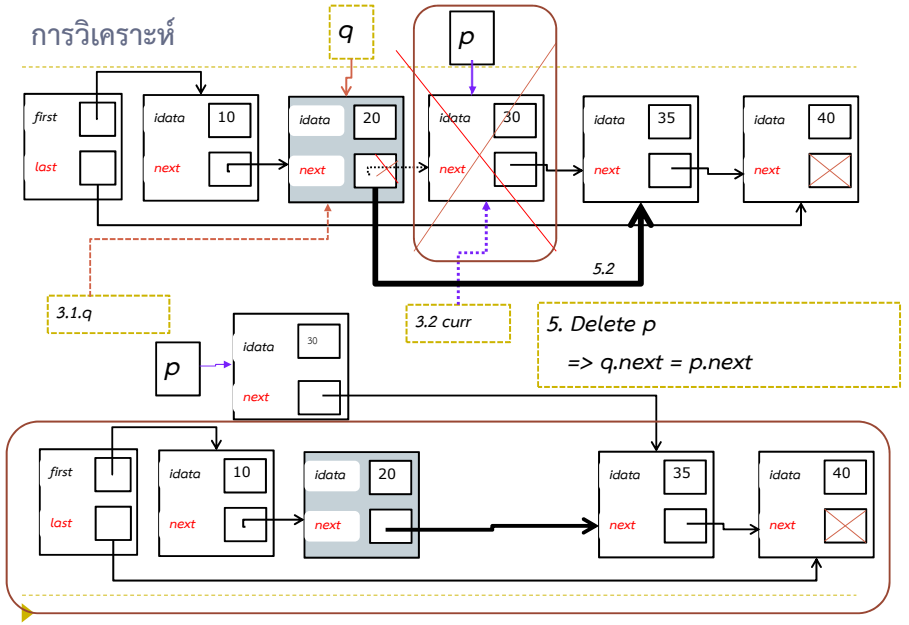
การวิเคราะห์



5. Delete p

=> q.next = p.next

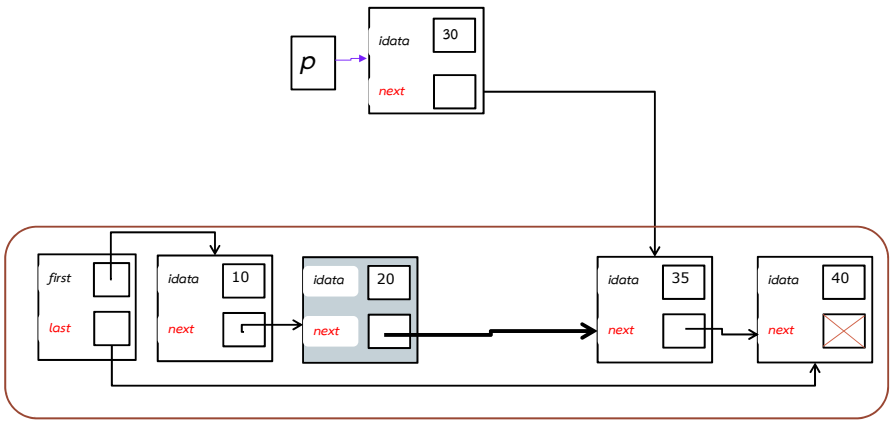
การวิเคราะห์



5. Delete p

=> q.next = p.next

การวิเคราะห์



การวิเคราะห์

หลักการ

1. ให้สร้างโหนด โหนดหนึ่งขึ้นมาเป็นตัววิ่ง ให้โหนดนั้นชื่อ curr
2. โดย curr จะเริ่มวิ่งจาก ต้น list (จาก first)
3. ให้สร้างโหนดอีกโหนดชื่อ q ไว้วิ่งตาม curr (curr จะอยู่ตำแหน่งถัดจาก q เสมอ)
4. ให้ เปรียบเทียบ curr กับ p

   ถ้าไม่เท่า

   ให้ q ชี้ที่เดียวกับ curr

   ขยับ curr ไปที่โหนดถัดไปในลิสต์

   Loop จนกว่าจะเจอ p หรือ curr เป็น null
5. เมื่อ curr เจอ p จะได้ node ก่อนหน้า p คือ q

   ให้ลบโหนด p ออกจาก list และ return

   ถ้าไม่เจอ return null

Pseudo Code

1. Create curr Node
2. *ให้ curr ชี้ไปที่เดียวกับ first;*
3. Create Node q
4. While (curr is not equal p) and (curr is not null)

   q = curr;

   curr = curr.next
5. if (curr==p) delete p

   change q.next to p.next

   return p
6. Else return null

ส่วนของโปรแกรมกรณีไม่ใช่ node แรกและ Node สุดท้าย ใช่ break

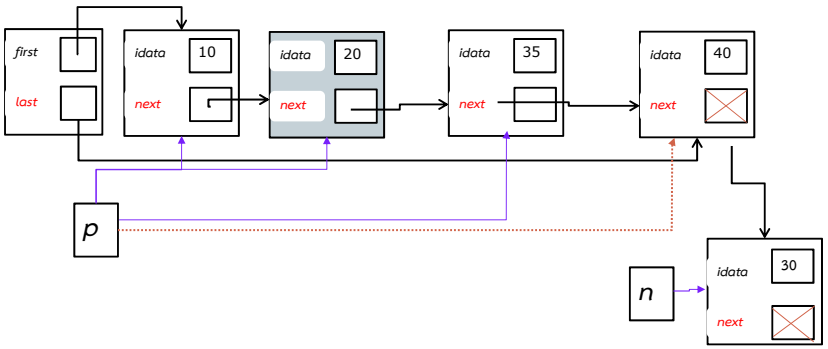
```
Node curr,q;
curr = first;
while (curr!=null)
{    if (curr!=p)
     { q = curr; curr=curr.next;
     }
     else break;
}
if (p==curr) {q.next = p.next; return(p);
else return null;
```


method deleteNode(Node p)

```
public Node deleteNode(Node p)
{
     if (first==null) return null; // กรณี list empty
     else if (p==first)  return(deleteFirst());     // กรณี p อยู่ต้น list
          else if (p==last) return(deleteLast());  // กรณี p อยู่ท้าย list
               else   // กรณี p อยู่กลางลิสต์
               {     Node curr,q;
                     curr = first;
                     while (curr!=null)
                     {    if (curr!=p)
                            { q = curr; curr=curr.next;}
                          else break;
                     }
                    if (p==curr) {q.next = p.next; return p;}
                    else return null;
               }
```

การเพิ่มโหนดต่อจากโหนด p

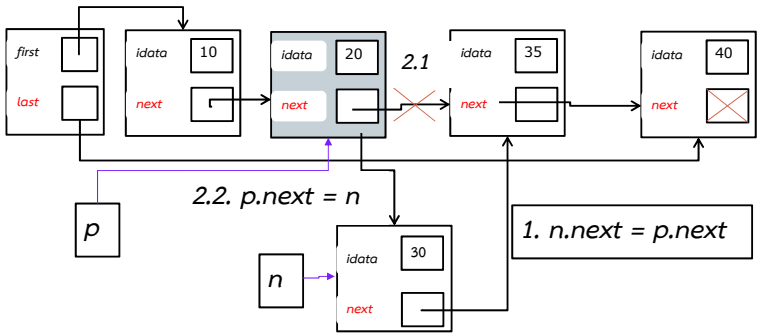
Slide 27



- การเพิ่มโหนดต่อจาก โหนด p มีโอกาสเกิดขึ้น 2 กรณี
  - 1. กรณี p เป็น โหนดสุดท้ายใน list (p=last) => insertLast
  - 2. กรณี p เป็น โหนดอื่นๆ

การเพิ่มโหนดต่อจากโหนด p (ต่อ)

Slide 28



- 2. กรณี p เป็น โหนดอื่นๆ ซึ่งทำได้โดย
  - 1. ให้ next ของ node ใหม่ ชี้ไปที่ node ที่เป็น next ของ p
  - 2. ให้ next ของ p เดิม ชี้ไปที่ที่ node ใหม่

Pseudo Code และ Code ของการเพิ่มโหนดลงไปท้าย Node p

1. สร้างและใส่ข้อมูลลง node n

2. ถ้า p เป็น null

   พิมพ์ *P is null.Can't Insert After*

3. Else

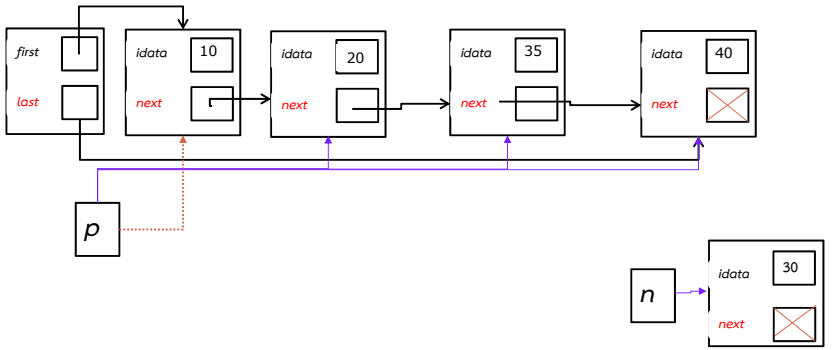
   -ให้ next ของ n ชี้ไปที่เดียวกับ next ของ p

   -เปลี่ยน next ของ p ให้ชี้ไปที่ n

4. ถ้า p เป็น node สุดท้าย เปลี่ยน last มาชี้ที่ n

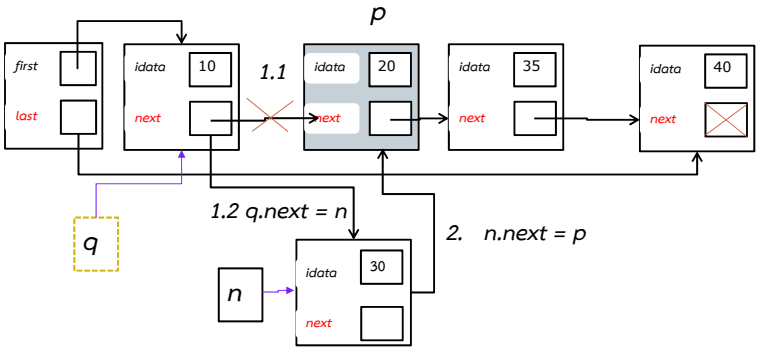
```
void insertAfterNode(int value,Node p)
{
  Node n = new Node(value,null);
  if (p==null)
  { System.out.println("P is null.Can't
    Insert After"); }
  else {
     n.next = p.next;
     p.next = n;
     if (p==last) last =n;
  }
}
```


การเพิ่มโหนดก่อนโหนด p



- การเพิ่มโหนดก่อนโหนด p มีโอกาสเกิดขึ้น 2 กรณี
  - 1. กรณี p เป็น โหนดแรกใน list (p=first) สามารถใช้ insertFirst ได้เลย
  - 2. กรณี p เป็น โหนดอื่นๆ ซึ่งจะค่อนข้างยุ่งยาก เพราะต้องรู้ว่า Node ก่อนโหนด p คืออะไร

การเพิ่มโหนดก่อนโหนด p (ต่อ)



- การหา Node ก่อน Node p ใช้วิธีเช่นเดียวกับ การหา Node ก่อน last หรือ Node ก่อน Node p ในการ DeleteNode (หา q)
- เมื่อหาได้
  - 1. ให้เปลี่ยนตัวชี้ของ q.next มาที่โหนดใหม่
  - 2. ให้ next ของโหนดใหม่ ชี้ไปที่ p

ลำดับในการเปลี่ยนไม่มีความสำคัญ เขียนขั้นตอนใดก่อนได้

Pseudo Code และ Code ของการเพิ่มโหนดลงไปก่อน Node p

1. สร้างและใส่ข้อมูลลง node n

2. ถ้า p เป็น null

   พิมพ์ *P is null.Can't Insert After*

3. Else

   -If p เป็น Node แรกหรือเปล่า

   - insertFirst(value)

   Else

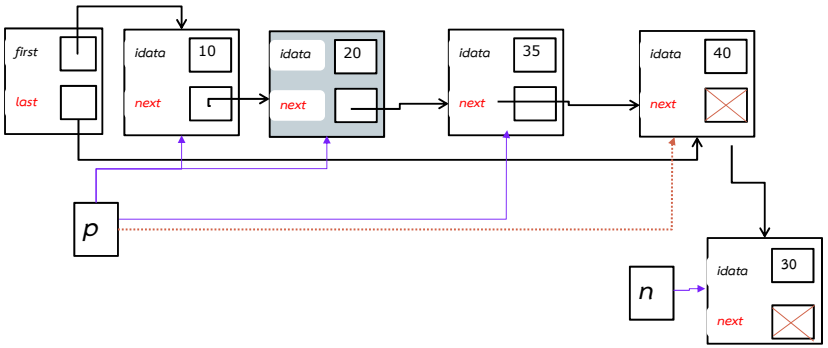
   - ให้ q คือ Node ก่อน Node p
   - หา Node q

   เปลี่ยน next ของ q ให้ชี้ไปที่ n

   เปลี่ยน next ของ n ให้ชี้ไปที่ q

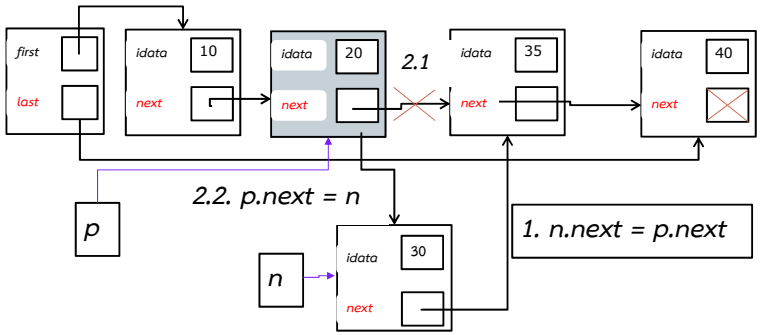
```
void insertBeforeNode(int value,Node p)
{ Node n = new Node(value,null);
  if (p==null)
  { System.out.println("P is null.Can't
    Insert After"); }
  else
  { if (p==first) insertFirst(value);
    else
    {  Node curr,q;
       while (curr!=null)
          if (curr!=p)
           {q=curr; curr=curr.next;
break;}
       if (p!=null)
       { q.next = n; n.next = p;}
       else System.out.println("Cannot
insert");
}}
```

การเพิ่มโหนดต่อจากโหนด p



- การเพิ่มโหนดต่อจาก โหนด p มีโอกาสเกิดขึ้น 2 กรณี
  - 1. กรณี p เป็น โหนดอื่นๆ
  - 2. กรณี p เป็น โหนดสุดท้าย
- ทั้งสองกรณีมีการทำงานเหมือนกัน แต่ในกรณีที่ 2 จะเพิ่มขั้นตอน การเปลี่ยนให้ node ใหม่เป็น last (n=last)

การเพิ่มโหนดต่อจากโหนด p (ต่อ)



1. ให้ next ของ node ใหม่ ชี้ไปที่ node ที่เป็น next ของ p
2. ให้ next ของ p เดิม ชี้ไปที่ที่ node ใหม่
3. ถ้า p เป็น node สุดท้าย ให้ เปลี่ยน last เป็น n

Pseudo Code และ Code ของการเพิ่มโหนดต่อจาก Node p

1. สร้างและใส่ข้อมูลลง node n

2. ถ้า p เป็น null

   พิมพ์ *P is null.Can't Insert After*

3. Else

   -ให้ next ของ n ชี้ไปที่เดียวกับ next ของ p

   -เปลี่ยน next ของ p ให้ชี้ไปที่ n

4. ถ้า p เป็น node สุดท้าย เปลี่ยน last มาชี้ที่ n

```
void insertAfterNode(int value,Node p)
{
  Node n = new Node(value,null);
  if (p==null)
  { System.out.println("P is null.Can't
    Insert After"); }
  else {
     n.next = p.next;
     p.next = n;
     if (p==last) last =n;
  }
}
```


ให้นักศึกษาสรุป method ทั้งหมดของ ลิ้งค์ลิสต์ที่เรียนมาทั้งหมด

ชื่อ	ใช้ในการทำอะไร	มีการ return ค่าหรือไม่ ถ้า return return เป็นชนิดอะไร
1. LinkList()	ใช้ในการสร้าง link list	ไม่มี
2. insertFirst()		

การวิเคราะห์ Big Oh

	Big Oh ดีที่สุด	Big Oh แย่ที่สุด
การเพิ่มข้อมูลลงในลิสต์		
- เพิ่มต้นลิสต์	O(1)	O(1)
- เพิ่มท้ายลิสต์	O(1)	O(1)
- เพิ่มก่อน Node p	O(1)	O(n)
- เพิ่มหลัง Node p	O(1)	O(1)
การลบข้อมูลลงในลิสต์		
- ลบต้นลิสต์	O(1)	O(1)
- ลบท้ายลิสต์	O(n)	O(n)
- ลบ Node p	O(1)	O(n)
การค้นหาข้อมูล	O(1)	O(n)

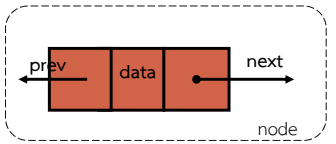
ปัญหาของ LinkList คืออะไร

Outline

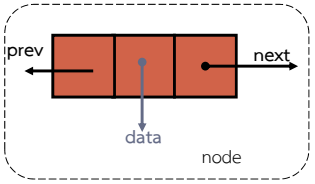
- โครงสร้างข้อมูลแบบ Double Linked List
- Operation ของโครงสร้างข้อมูลแบบ Linked List
  - การท่องไปใน ลิสต์
  - การเพิ่มข้อมูล
    - ต้นลิสต์
    - ท้ายลิสต์
  - การลบ
    - ต้นลิสต์
    - ท้ายลิสต์
  - การค้นหา

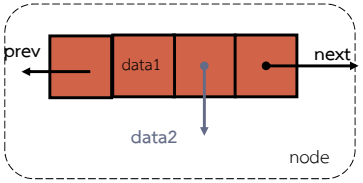
องค์ประกอบของข้อมูลแบบ Double Link List

- ลิสต์ของโหนดที่เก็บข้อมูลเชื่อมต่อกันเป็นสาย (ข้อมูลต้องเป็นข้อมูลประเภทเดียวกัน)
- แต่ละโหนดเก็บ



1. ข้อมูล (data)
   - ข้อมูลประเภทปกติ (primitive data type) เช่น
     - int (หมายเลข)
     - char (ตัวอักษร)
   - ข้อมูลประเภท object (จะเก็บลิ้งค์ชี้ไปที่ข้อมูล)
     - String เช่น ชื่อนักศึกษา
   - กลุ่มของข้อมูลหลายตัวผสมกัน
     - มีทั้ง หมายเลข ชื่อ
2. พอยน์เตอร์ (ลิ้งค์) ชี้ไปที่โหนดถัดไป
3. พอยน์เตอร์ (ลิ้งค์) ชี้ไปที่โหนดก่อนหน้าไป

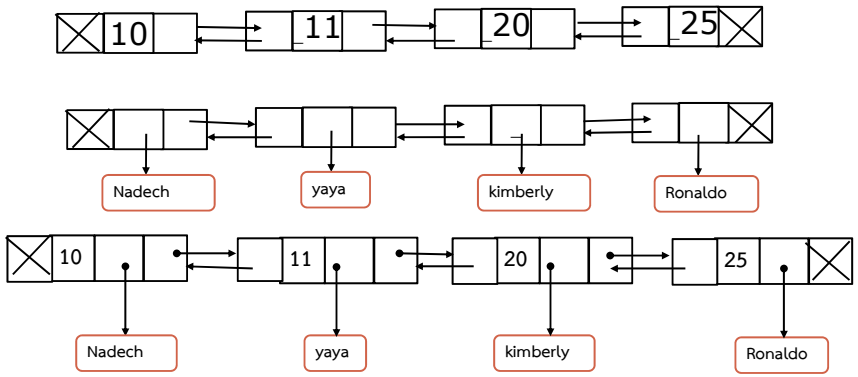




ตัวอย่างของข้อมูลแบบ Double Link List

next ของ โหนดสุดท้ายเนื่องจากไม่ชี้ไปไหนเลยให้เป็น null

prev ของ โหนดสุดท้ายเนื่องจากไม่ชี้ไปไหนเลยให้เป็น null เช่นกัน



การสร้าง Double LinkList Node ใน จาวา

```
class DNode
{ public int iData;   // เก็บข้อมูลเป็นจำนวนเต็ม
  public Dnode prev; // ลิ้งค์ชี้ไปที่โหนดก่อนหน้า
  public DNode next;    // ลิ้งค์ชี้ไปที่โหนดถัดไป
}
```

- การเรียกใช้ เช่น ต้องการเก็บข้อมูลลงใน Node a โดย Node a เป็น สมาชิกตัวเดียว
- 1. สร้างตัวแปรให้มีชนิดของข้อมูลเป็น Node

  DNode a; a = new DNode();

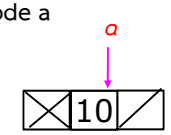
  DNode a = new DNode();

  2. ทำการใส่ข้อมูลลงไปใน Node a

  a.iData = 10;

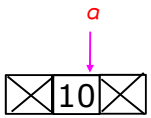
  a.prev = null;

  a.next = null;



เพื่อให้สร้างง่ายขึ้น จะมีการใช้ Constructor method

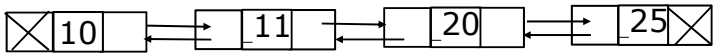
```
class DNode
{ public int iData;   // เก็บข้อมูลเป็นจำนวนเต็ม
  public Dnode prev; // ลิ้งค์ชี้ไปที่โหนดก่อนหน้า
  public DNode next;    // ลิ้งค์ชี้ไปที่โหนดถัดไป
  public DNode(int v)   // constructor method #1
  {     iData = v;
        prev  = null;
        next =  null;
  }
```



- DNode a ; a = new Node(10); หรือ
- DNode a = new Node(10);

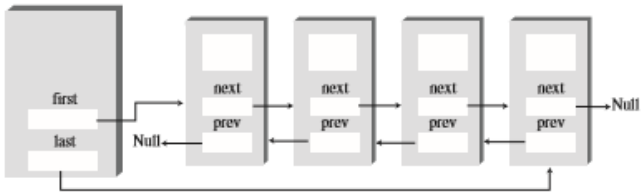
แบบฝึกหัด

- ให้นักศึกษาใช้ class DNode ใน หน้า 12 หรือ 13 สร้าง ลิสต์ต่อไปนี้ (เขียนเป็นคำสั่งทีละบรรทัด

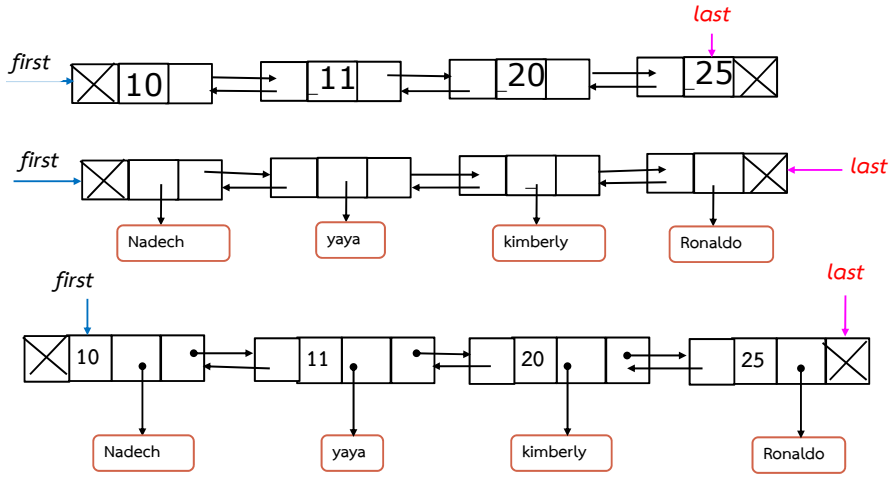


องค์ประกอบของ Double Link List

- การเข้าถึง node ต่างๆ ใน Double Linked List สามารถทำได้สองทิศทาง
  - ท่อง(traverse) จาก Node แรก ไปตามลำดับทีละ Node จนถึง node สุดท้าย
  - ท่อง(traverse) ย้อนกลับ จาก Node สุดท้าย ไปตามลำดับจนถึง node แรก
- ดังนั้นต้องมีตัวชี้ไปที่ต้น list => first
- และตัวชี้ไปที่ท้าย list คือ last
- Node แรกใน D List prev จะต้องเป็น Null (first.prev= null)
- Node สุดท้ายใน D List next จะต้องเป็น Null (last.next =null)



องค์ประกอบของข้อมูลแบบ Link List (ต่อ)



Operation พื้นฐานของ Double Linked List

1. การสร้าง Node ใหม่
2. การค้นหา
3. การเพิ่มสมาชิก
   1. การเพิ่มต้นลิสต์
   2. การเพิ่มท้ายลิสต์
   3. การเพิ่มที่ตำแหน่งอื่นๆ
4. การลบสมาชิกออกจากลิสต์ (Deleting an item from the list)
   - การลบสมาชิกตัวแรกในลิสต์
   - การลบสมาชิกตัวสุดท้ายในลิสต์
   - การลบที่ตำแหน่งอื่นๆ ในลิสต์

การสร้างลิสต์

- 1. สร้าง ลิสต์
  - เมื่อเริ่มสร้างให้ first เป็น null
- 2. เพิ่ม node ลงไปใน ลิสต์
  - ทำการสร้าง node ใหม่ ก่อน
  - ใส่ข้อมูลลงไปใน node
  - ทำการเพิ่มโหนดลงไปในลิสต์
    - โดยทำการเชื่อมโหนดนี้กับโหนดในลิสต์

ดังนั้นในจาวาจึงมักจะสร้างเป็น 2 class

1. class Dlinklist ใช้ในการจัดการเกี่ยวกับลิสต์
   - สร้างลิสต์
   - เพิ่มสมาชิกในลิสต์
   - ลบสมาชิกในลิสต์
   - ท่องไปในลิสต์
2. Class DNode ใช้ในการสร้าง node
   1. ใส่ data ลงไปใน node
   2. กำหนดว่า ให้ next ชี้ไปที่ใด
   3. กำหนดว่า ให้ prev ชี้ไปที่ใด

การสร้าง ลิสต์ใหม่

เนื่องจากลิสต์ใหม่ยังไม่มีสมาชิก จึงเป็น ลิสต์ว่าง ดังนั้น

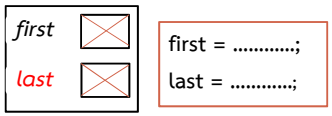
- 1. ให้ first ชี้ไปที่ null
- 2. ให้ last ชี้ไปที่ null

  ทั้ง first และ last จะมีชนิดของข้อมูลเป็น DNode
- (3. ในกรณีที่มีการนับจำนวนสมาชิก (nItem) ใน ลิสต์ ให้ค่า nItem=0)

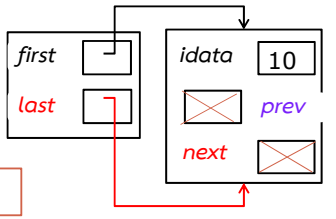
```
class DLinkList
{      // เก็บข้อมูลเป็นจำนวนเต็ม
  DNode first;    // ลิ้งค์ชี้ไปที่โหนดแรก
  DNode last;     // ลิ้งค์ชี้ไปที่โหนดสุดท้าย
  public DLinkList()  // constructor method
  {      first = null;
         last =  null;
  }
//..... เมธอดที่เป็น operation อื่นๆ เช่น การเพิ่ม และ ลบสมาชิกใน ลิสต์
}
```


ต่อไปนี้จะแทน ลิ้งค์ลิสต์ดังรูป

- 1. ลิสต์ว่าง

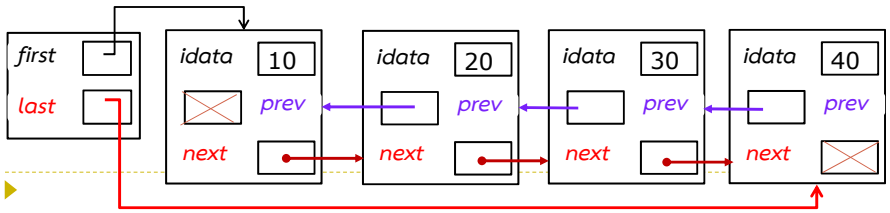


- 2. ลิสต์มีสมาชิก 1 ตัว

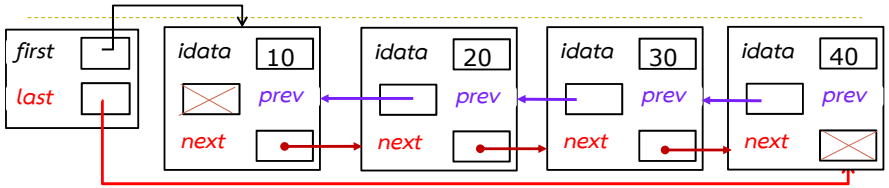


- 3.ลิสต์มีสมาชิกหลายตัว

ในกรณี ลิสต์มีสมาชิก last.next = null; เสมอ



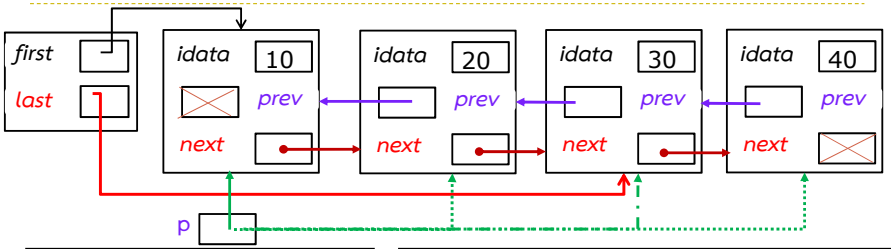
การอ้างถึงค่าในดับเบิลลิ้งค์ลิสต์



1. first.idata =
2. first.next.idata =
3. first.next.next.idata =
4. first.next.next.next.idata =
5. first.next.next.next =
6. first.next.next.next.next =
7. first.next.prev=

1. last.idata=
2. last.prev.idata =
3. last.prev.prev.idata =
4. last.prev.prev.prev.idata =
5. last.prev.prev.prev =
6. last.prev.prev.prev.prev =
7. last.prev.next =

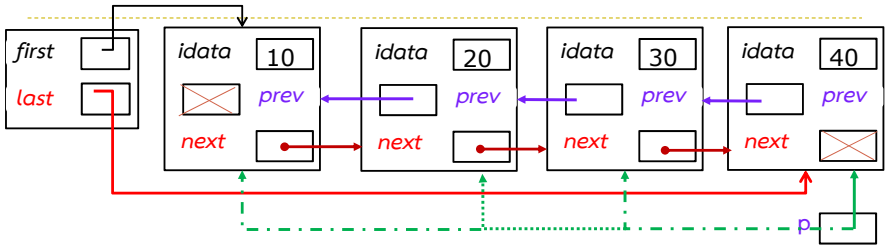
การท่องไปในลิสต์ จาก first -> last เหมือน Linked List



1. สร้างตัววิ่ง (p)ซึ่งมีชนิดข้อมูลเป็นชนิดข้อมูลเดียวกับโหนด (Node)
2. ให้ตัววิ่งเริ่มวิ่งจากต้นลิสต์
3. วิ่งไปเรื่อยๆ จนกว่า p = null

```
DNode p;
p=first;
while (p!=null)
{
      statement; statement; …
      p=p.next;
}
```

การท่องไปในลิสต์ จาก last-> first



1. สร้างตัววิ่ง (p)ซึ่งมีชนิดข้อมูลเป็นชนิดข้อมูลเดียวกับโหนด (Node)
2. ให้ตัววิ่งเริ่มวิ่งจากท้ายลิสต์
3. วิ่งย้อนไปเรื่อยๆ จนกว่า p = null

```
DNode p;
p=last;
while (p!=null)
{
      statement; statement; …
      p=p.prev;
}
```


Pseudo Code และ Code ของการพิมพ์ข้อมูลในลิสต์ออกมาดูจาก first -> Last

1. Check
   if list is empty
      ปรินท์ ลิสต์ว่าง ไม่มีข้อมูล
2. Else list ไม่ว่าง
   2.1 สร้างตัววิ่ง p มีชนิดเป็น Node
   2.2 ให้ p = first;
   2.3 while p!=null
      print p.idata
      p = p.next
   end while

```
public void displayDList()
{
  if (first==null) {
     System.out.println("List is Empty");
  }
  else
  {  DNode p;    p = first;
    System.out.println("First=>Last: ");
     while (p!=null)
     {
           System.out.println(p.idata);
           p = p.next;
     }
   }
}
```

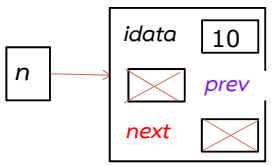

Pseudo Code และ Code ของการพิมพ์ข้อมูลในลิสต์ออกมาดูจาก Last -> first

1. Check
   if list is empty
      ปรินท์ ลิสต์ว่าง ไม่มีข้อมูล
2. Else list ไม่ว่าง
   2.1 สร้างตัววิ่ง p มีชนิดเป็น Node
   2.2 ให้ p = last;
   2.3 while p!=null
      print p.idata
      p = p.prev
   end while

```
public void displayListReverse()
{
  if (first==null) {
     System.out.println("List is
Empty");
  }
  else
  {  DNode p;    p = last;
     System.out.println("Last=>First:
");
     while (p!=null)
     {
           System.out.println(p.idata);
           p = p.prev;
     }
   }
}
```


การเพิ่มโหนดลงไปในลิสต์

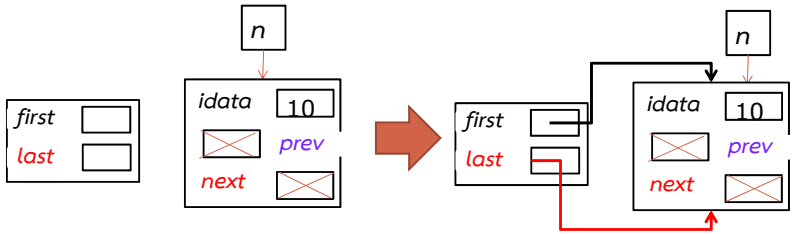
- เช่นเดียวกับ Single Link List ไม่ว่าจะเป็นการเพิ่มไปที่ตำแหน่งใด ขั้นตอนแรกต้องทำการสร้างโหนด และ ใส่ข้อมูลใหม่ลงไปในโหนดให้เรียบร้อยก่อน
  - เช่นในกรณี เก็บข้อมูลประเภท integer และต้องการเพิ่มโหนดที่มีข้อมูล 30
  - ภาษาจาวา *n = new DNode (); n.idata = 30; n.next = null; n.prev=null,*
  - หรือ *n = new Node(30);*



- แล้วค่อยทำการเพิ่มลงไปในลิสต์

การเพิ่มโหนดลงไปต้น list (ลิสต์ว่าง)

- มี 2 กรณี คือ ลิสต์ว่าง และ ลิสต์ไม่ว่าง
- กรณีลิสต์ว่าง
  - เปลี่ยน first และ last ให้มาชี้ที่โหนดใหม่ที่เพิ่มเข้าไป



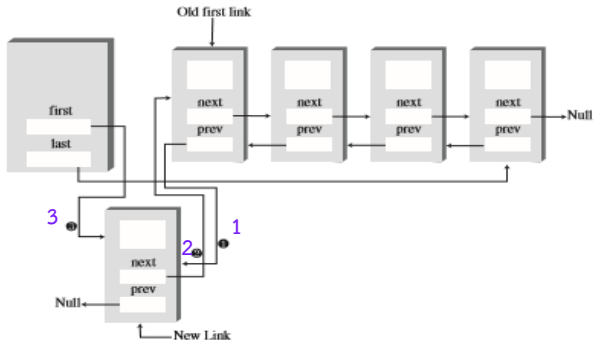
1. first = n;
2. last =n;

การเพิ่มโหนดลงไปต้น list (ลิสต์ไม่ว่าง)

2. ลิสต์ไม่ว่าง

- 1. ให้ n.next ชี้ไปที่ตำแหน่งเดียวกับ first;
- 2. ให้ first.prev ให้ชี้ไปที่ตำแหน่งเดียวกับ n;
- 2. เปลี่ยน first ใหม่ให้ชี้ไปที่ n

2 บรรทัดนี้ สลับที่ได้



Pseudo Code และ Code ของการเพิ่มโหนดต้นลิสต์ออกมาดู

1. สร้างและใส่ข้อมูลลง node n

2. If list ว่าง

   เปลี่ยน first และ last ให้ชี้ไปที่เดียวกับ n

   Else

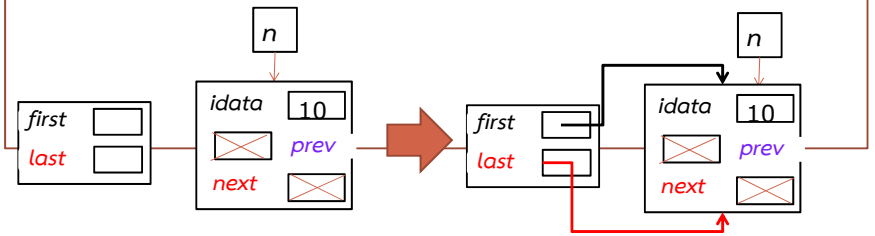
   - ให้ next ของโหนดใหม่ (n) ชี้ไปที่ first
   - เปลี่ยน first.prev ให้ ชี้มาที่ โหนด ใหม่ n
   - เปลี่ยน first ให้ไปชี้ที่ n

```
public void insertDFirst(int value)
{
  DNode n = new DNode(value);
  if (first==null) {
       first=n; last = n;
  }
  else
  {    n.next = first;
       first.prev = n;
       first = n;
  }
}
```

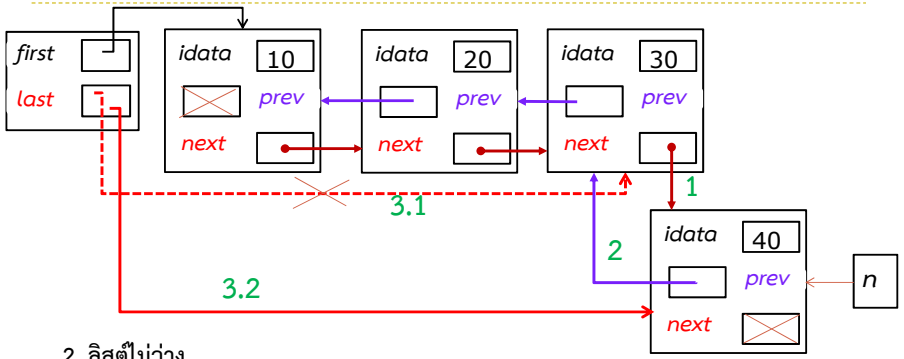

การเพิ่มโหนดลงไปท้าย list กรณีลิสต์ว่าง

- มี 2 กรณี คือ
- 1. ลิสต์ว่าง (เหมือนกับกรณีต้น list)
  - เปลี่ยน first และ last ให้มาชี้ที่โหนดใหม่ที่เพิ่มเข้าไป

1. first = n;
2. last =n;



การเพิ่มโหนดลงไปท้าย list กรณีลิสต์ไม่ว่าง



2. ลิสต์ไม่ว่าง

- 1. ให้ last.next ชี้ไปที่ n
- 2. ให้ n.prev ชี้ไปที่ last เดิม;
- 3. เปลี่ยน last ใหม่ให้ชี้ไปที่ n

```
1. last.next = n;
2. n.prev  = last;
3. last = n;
```

ขั้นตอนที่ 1 และ 2 สลับที่ได้

Pseudo Code และ Code ของการเพิ่มโหนดลงไปท้ายลิสต์กรณีลิสต์ไม่ว่าง

1. สร้างและใส่ข้อมูลลง node n

2. If list ว่าง

   เปลี่ยน first และ last ให้ชี้ไปที่เดียวกับ n

   Else

   - ให้ last.next ชี้ไปที่ n
   - ให้ n.prev ชี้ไปที่ last เดิม;
   - เปลี่ยน last ใหม่ให้ชี้ไปที่ n

```
public void insertDLast(int value)
{
  DNode n = new DNode(value);
  if (first==null) {
       first=n; last = n;
  }
  else
  {    last.next = n;
       n.prev = last;
       last = n;
  }
}
```


การลบ:

การลบต้น list และ การลบท้ายลิสต์

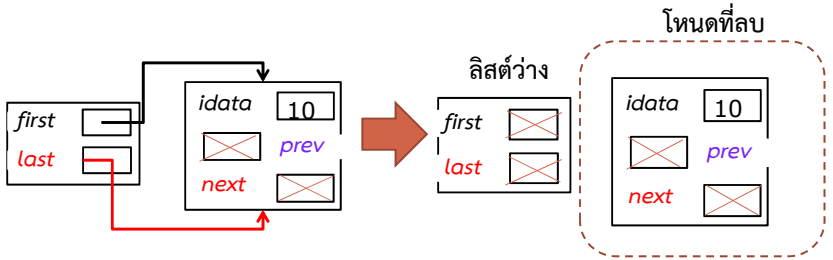
เหมือนกับ Single Linked List คือ

- ไม่ว่าลบแบบใดผลลัพธ์ เป็น
  - node ที่ถูกลบ
  - ลิสต์ที่จำนวนสมาชิกลดลง
- ลบต้น ลิสต์ มี 3 กรณี
  - 1. ไม่มีโหนดใน list
    - ลบไม่ได้ต้องแสดงข้อความว่าลบไม่ได้
  - 2. สมาชิกมีโหนดเดียวใน list
    - ผลลัพธ์ของลิสต์ที่ได้คือว่าง
  - 3. มีจำนวนสมาชิกหลายโหนดใน list
    - ลิสต์จะมีสมาชิกลดลง 1 ตัว

การลบ:

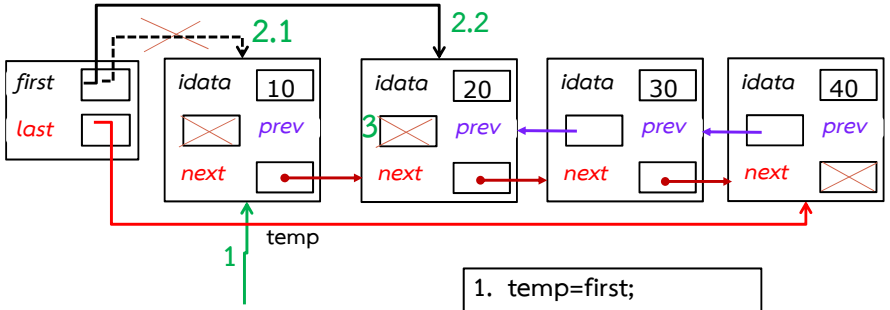
การลบกรณีมีสมาชิกตัวเดียว

- มีสมาชิกตัวเดียว (first = last)
  - เหมือนกันทั้งลบต้นและลบท้ายลิสต์
- return โหนดที่ลบ และ ได้ลิสต์ว่าง



การลบ:

การลบต้นลิสต์ มีสมาชิกมากกว่าหนึ่งโหนด



1. temp=first;
2. first=first.next;
3. first.prev=null;
4. return(temp)

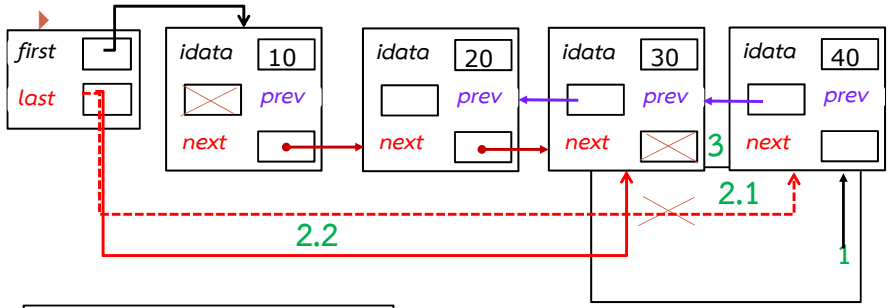
Pseudo Code และ Code ของการลบต้นลิสต์ออกมาดู

1. If list ว่าง
   พิมพ์ list empty. Cannot Delete
   return null
   Else
   - ให้ temp ชี้ไปที่เดียวกับโหนด first
   - If list มีสมาชิกตัวเดียว
     Set first และ last เป็น null
     Else
     - เปลี่ยน first ให้ ชี้ไปที่โหนด first.next
     - เปลี่ยน first.prev ให้เป็น null
     - return temp

```
public Node deleteFirst()
{

DNode temp;
  if (first==null) {
       System.out.println("List is
empty. Cannot delete");
       return(null);
  }
  else
  {    temp = first;
       if (first==last)
       { first = null; last = null;}
       else
       { first = first.next;
         first.prev = null;
         return(temp);
}
```


การลบ:

การลบท้ายลิสต์ ให้นักศึกษาลองวาดภาพการลบท้ายลิสต์



1. temp=last;
2. last=last.prev;
3. last.next=null;
4. return(temp)

การลบท้ายลิสต์สามารถทำได้เลยไม่ต้องหาโหนดก่อนหน้าเหมือนใน Singly Link List

Pseudo Code และ Code ของการลบท้ายลิสต์ออกมาดู

1. If list ว่าง
   พิมพ์ list empty. Cannot Delete
   return null
   Else
   - ให้ temp ชี้ไปที่เดียวกับโหนด last
   - If list มีสมาชิกตัวเดียว
     Set first และ last เป็น null
     Else
     - เปลี่ยน last ให้ ชี้ไปที่โหนด last.prev
     - เปลี่ยน last.next ให้เป็น null
     - return temp

```
pudblic Node deleteDFirst()
{
  Node temp;
  if (first==null) {
       System.out.println("List is
empty. Cannot delete");
       return(null);
  }
  else
  {    temp = first;
       if (first==last)
       { first = null; last = null;}
       else
       { last = last.prev;
         last.next = null;
         return(temp);
}
```

การค้นหา Node ใน Double Link List ที่มีข้อมูลเท่ากับค่าที่ต้องการค้นหา

- การค้นหา Node ใน Double Link List ในกรณีที่หาจาก first-> last จะเหมือนกับใน Single Linked List แทบทุกประการ ให้ขีดเส้นใต้ โค้ดที่ต้องเปลี่ยน

```
public Node findNode (int key) //
{    if (first==null)
      {  System.out.println("List
            return null;  }
      else
      {      Node p=new Node();

            p  = first;
            while (p!=null)
            {    if (p.idata==key)
                     break;
                 else p = p.next);
            }
           if (p==null)
           {
               return null; return -1; }
           else
           {    return p;  return p.idata;}
      }
                                  }
```


ให้นักศึกษาเขียน method การลบโหนดที่ตำแหน่ง p ใน Double Link List

- DLNode deleteDLNode(DLNode p)

การวิเคราะห์ Big Oh ของ Double Linked List

	Big Oh ดีที่สุด	Big Oh แย่ที่สุด
การเพิ่มข้อมูลลงในลิสต์		
- เพิ่มต้นลิสต์		
- เพิ่มท้ายลิสต์		
- เพิ่มก่อน Node p		
- เพิ่มหลัง Node p		
การลบข้อมูลลงในลิสต์		
- ลบต้นลิสต์		
- ลบท้ายลิสต์		
- ลบ Node p		
การค้นหาข้อมูล		

การบ้าน

1. จงเขียน ผังงานหรือโปรแกรมในการนับจำนวนสมาชิกใน Double Linked ลิสต์ว่ามีทั้งหมดกี่โหนด โดยให้เริ่มนับจาก last
2. จงเขียน method ชื่อ CopyReverseList
   โดย method นี้จะสร้าง Double Link List ใหม่ซึ่งมีสมาชิกอยู่ในลำดับที่ตรงกันข้าม กับ Double Link List ที่เป็น input
3. ให้นักศึกษาเขียน method printReverseList() เพื่อทำการพิมพ์ค่าใน Double Link List แบบ reverse order