05506006 Data Structure

Lecture 5: Linked Lists

ดร. รุ่งรัตน์ เวียงศรีพนาวัลย์

Outline

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

ปัญหาของโครงสร้างข้อมูลแบบอาร์เรย์

- ข้อดีของโครงสร้างข้อมูลแบบอาร์เรย์ คือ
  - 1. ใช้งานง่าย
  - 2. การเข้าถึงข้อมูล สามารถเข้าถึงได้ทันที ไม่ว่าจะเป็นอาร์เรย์ที่เก็บข้อมูลที่เรียงแล้วหรือยังไม่เรียง (เข้าผ่าน index) O(1)
- แต่ข้อเสียคือ
  - 1. ขนาดเปลี่ยนแปลงไม่ได้
    - ดังนั้นเมื่อต้องการเพิ่มขนาด จะต้องทำการคอมไพล์โค้ดใหม่ สำหรับคนที่สร้างอาร์เรย์ไว้โดยจองเนื้อที่เป็นจำนวนมากๆ (เผื่อเหลือ) เป็นการใช้เนื้อที่แบบไม่มีประสิทธิภาพ (จองไปไม่ได้ใช้)
  - 2. การแทรกข้อมูลลงไปที่ตำแหน่งที่มีข้อมูลอยู่แล้วของข้อมูลที่เรียงแล้วจะทำให้ต้องเลื่อนข้อมูลที่อยู่ก่อนหน้านั้น หรือ หลังจากตำแหน่งนั้นทั้งหมด
    - ดังนั้นในกรณีแย่ที่สุด เช่น แทรกข้อมูลไปที่ตำแหน่งแรก ถ้าข้อมูลมี 10 ตัว ต้องเลื่อนข้อมูลทุกตัวไปทางขวา 1 ตำแหน่ง (ดังนั้น ทำการเลื่อนทั้งหมด 10 ครั้ง) ถ้ามี n ตัว เลื่อน n ครั้ง O(n)
    - แต่การแทรกข้อมูลลงไปใน unordered array สามารถแทรกต่อท้ายได้เลย => O(1)
  - การลบข้อมูลไม่ว่าจะเป็นการลบค่าจากอาร์เรย์ที่ข้อมูลเรียงกัน หรือ ไม่เรียงกันจะต้องมีการเลื่อนค่าของตำแหน่งต่อๆ มา แทนตำแหน่งที่ลบ
    - ดังนั้นในกรณีแย่ที่สุด เช่น ลบข้อมูลตำแหน่งแรก ถ้าข้อมูลมี 10 ตัว ต้องเลื่อนข้อมูลที่เหลือ 9 ตัวไปทางซ้าย ตำแหน่ง (ดังนั้น ทำการเลื่อนทั้งหมด 9 ครั้ง) ถ้ามี n ตัว เลื่อน n-1 ครั้ง => O(n)

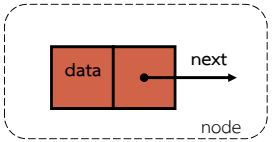
สรุป Big Oh ของข้อมูลประเภทอาร์เรย์

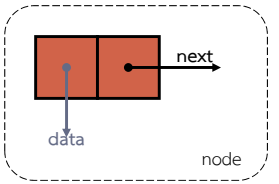
อัลกอริทึม	Running Time (Big Oh)
การค้นหาข้อมูลแบบเชิงเส้น (Linear Search)	O(n)
การค้นหาข้อมูลแบบไบนารี (Binary Search)	O(log n)
การแทรกข้อมูลในอาร์เรย์ที่ข้อมูลเรียงกัน (Insertion in ordered array)	O(n)
การแทรกข้อมูลในอาร์เรย์ที่ข้อมูลไม่เรียงกัน (Insertion in unordered array)	O(1)
การลบข้อมูลในอาร์เรย์ที่ข้อมูลเรียงกัน (Deletion in ordered array)	O(n)
การลบข้อมูลในอาร์เรย์ที่ข้อมูลไม่เรียงกัน (Insertion in unordered array)	O(n)

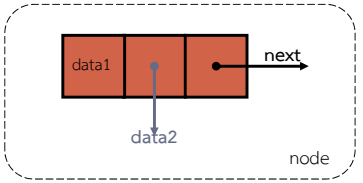
- จึงต้องมีโครงสร้างข้อมูลใหม่ที่
  - ขนาดของข้อมูลไม่คงที่ เป็น dynamic
  - การเพิ่ม หรือ การลบข้อมูล ทำได้อย่างรวดเร็ว

องค์ประกอบของข้อมูลแบบ 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



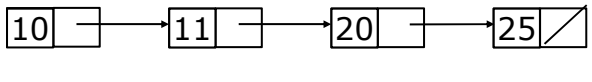


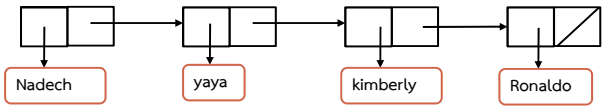


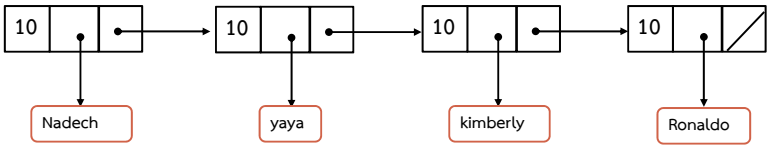
สำหรับโหนดสุดท้ายเนื่องจากไม่ชี้ไปไหนเลยให้เป็น null

ตัวอย่างของข้อมูลแบบ Link List

สำหรับโหนดสุดท้ายเนื่องจากไม่ชี้ไปไหนเลยให้เป็น null







การสร้าง Node ใน จาวา

```
class Node
{ public int iData;   // เก็บข้อมูลเป็นจำนวนเต็ม
  public Node next;    // ลิ้งค์ชี้ไปที่โหนดถัดไป
}
```

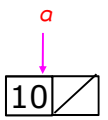
- การเรียกใช้ เช่น ต้องการเก็บข้อมูลลงใน Node a โดย Node a เป็น สมาชิกตัวเดียว
- 1. สร้างตัวแปรให้มีชนิดของข้อมูลเป็น Node

  Node a; a = new Node();

  Node a = new Node();
- 2. ทำการใส่ข้อมูลลงไปใน Node a

  a.iData = 10;

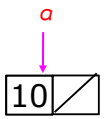
  a.next = null;



เพื่อให้สร้างง่ายขึ้น จะมีการใช้ Constructor method

```
class Node
{ public int iData;   // เก็บข้อมูลเป็นจำนวนเต็ม
  public Node next;    // ลิ้งค์ชี้ไปที่โหนดถัดไป
  public Node(int v, Node n)  // constructor method
  {     iData = v;
        next =  n;
}
```

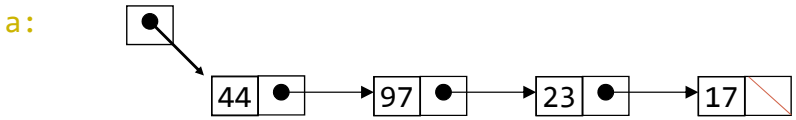
- Node a ; a = new Node(10,null); หรือ
- Node a = new Node(10,null);
- ให้นักศึกษาสร้างลิสต์ (10,20,30)



ตัวอย่าง

- จากคลาส Node ในหน้า 12 และ 13 ให้นักศึกษาเขียนลิสต์ซึ่งเป็นผลลัพธ์ที่ได้จาก Code ต่อไปนี้

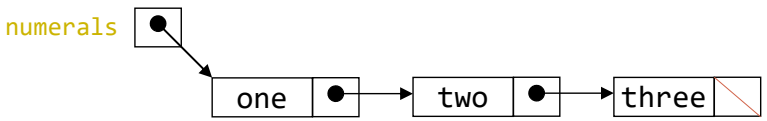
```
Node temp = new Node(17, null);
temp = new Node(23, temp);
temp = new Node(97, temp);
Node a = new Node(44, temp);
```



การสร้าง ลิสต์แบบง่าย

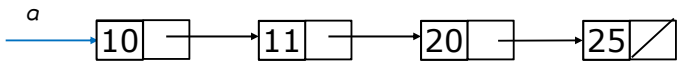
สร้าง list ("one", "two", "three"):

```
Node numerals = new Node();
numerals =
    new Node("one",
        new Node("two",
            new Node("three", null)));
```



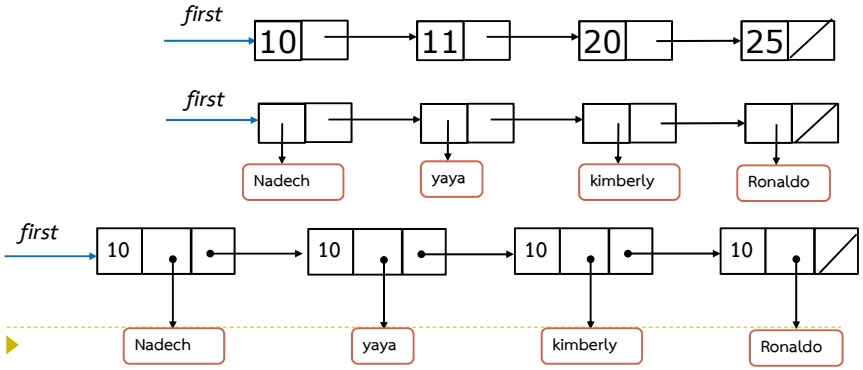
แบบฝึกหัด

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



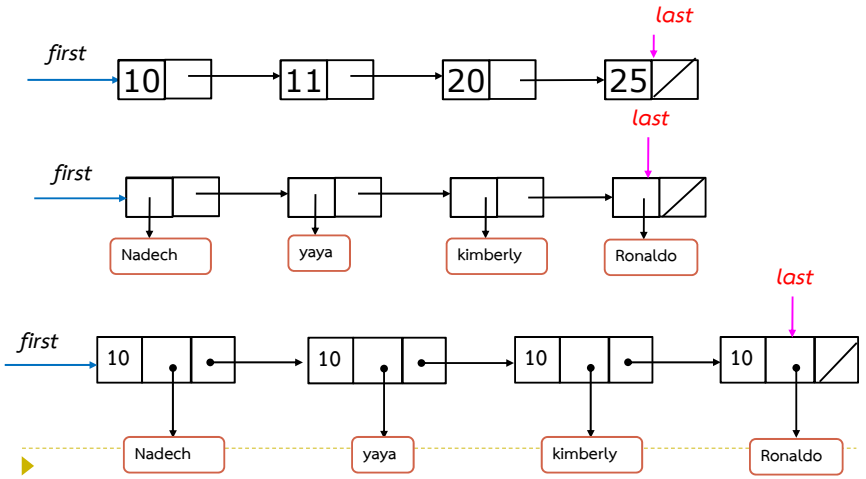
องค์ประกอบของข้อมูลแบบ Link List (ต่อ)

- การเข้าถึง node ต่างๆ ใน List จะต้องเริ่มเดินทางหรือ ท่อง(traverse) จาก Node แรก ไปตามลำดับทีละ Node จนถึง node สุดท้าย
- การจะท่องไปในลิสต์ได้ต้องเริ่มจากข้อมูลลำดับแรกในลิสต์จึงเรียกพอยน์เตอร์ที่ชี้ไปที่โหนดแรกของ ลิสต์ ว่า first (first มีชนิดข้อมูลเป็น Node เช่นกัน)



องค์ประกอบของข้อมูลแบบ Link List (ต่อ)

- นอกจากให้ first ชี้ไปที่ตำแหน่งแรกใน ลิสต์แล้ว บางลิสต์จะมีการเก็บพอยน์เตอร์ที่ชี้ไปที่โหนดสุดท้ายใน list ให้ชื่อว่า last (last มีชนิดข้อมูลเป็น Node เช่นกัน)



Operation พื้นฐานบน Link List

- การสร้างลิสต์
- การท่องไปในลิสต์ (Traversing the list)
- การเพิ่มสมาชิกลงไปในลิสต์ (Inserting an item in the list)
  - การเพิ่มที่ต้นลิสต์
  - การเพิ่มไปที่ท้ายลิสต์
  - *การเพิ่มที่ตำแหน่ง i ใน ลิสต์*
- การลบสมาชิกออกจากลิสต์ (Deleting an item from the list)
  - การลบสมาชิกตัวแรกในลิสต์
  - การลบสมาชิกตัวสุดท้ายในลิสต์
  - การลบที่ตำแหน่ง i ใน ลิสต์
- การหาค่าที่ตำแหน่ง i
- หาขนาดของลิสต์

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 linklist ใช้ในการจัดการเกี่ยวกับลิสต์
   - สร้างลิสต์
   - เพิ่มสมาชิกในลิสต์
   - ลบสมาชิกในลิสต์
   - ท่องไปในลิสต์
2. Class Node ใช้ในการสร้าง node
   1. ใส่ data ลงไปใน node
   2. กำหนดว่า ให้ next ชี้ไปที่ใด

การสร้าง ลิสต์ใหม่

เนื่องจากลิสต์ใหม่ยังไม่มีสมาชิก จึงเป็น ลิสต์ว่าง ดังนั้น

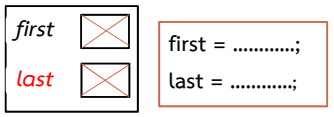
- **1. ให้ first ชี้ไปที่ null**
- **2. ให้ last ชี้ไปที่ null**

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

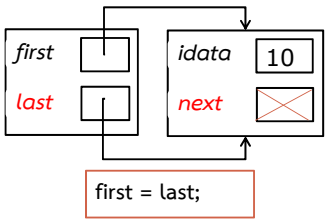
```
class LinkList
{      // เก็บข้อมูลเป็นจำนวนเต็ม
  Node first;    // ลิ้งค์ชี้ไปที่โหนดแรก
  Node last;     // ลิ้งค์ชี้ไปที่โหนดสุดท้าย
  public LinkList()  // constructor method
  {      first = null;
         last =  null;
  }
//..... เมธอดที่เป็น operation อื่นๆ  เช่น การเพิ่ม และ ลบสมาชิกใน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


ต่อไปนี้จะแทน ลิ้งค์ลิสต์ดังรูป

- **1. ลิสต์ว่าง**

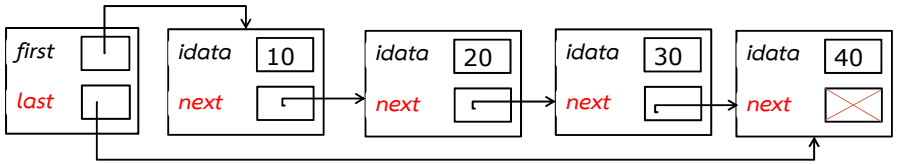


- **2. ลิสต์มีสมาชิก 1 ตัว**

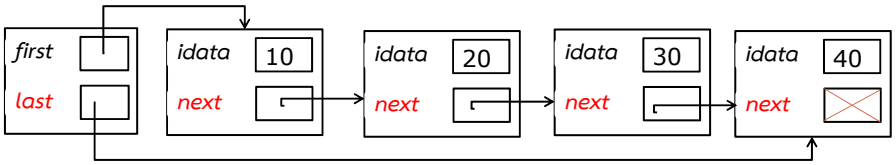


- **3.ลิสต์มีสมาชิกหลายตัว**

ในกรณี ลิสต์มีสมาชิก last.next = null; เสมอ



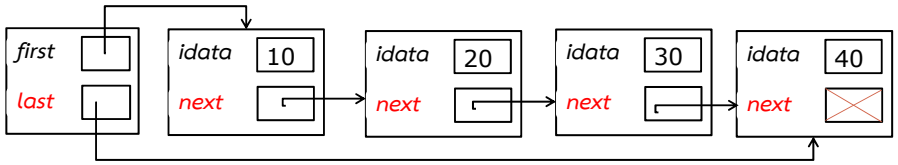
การอ้างถึงค่าใน ลิสต์



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

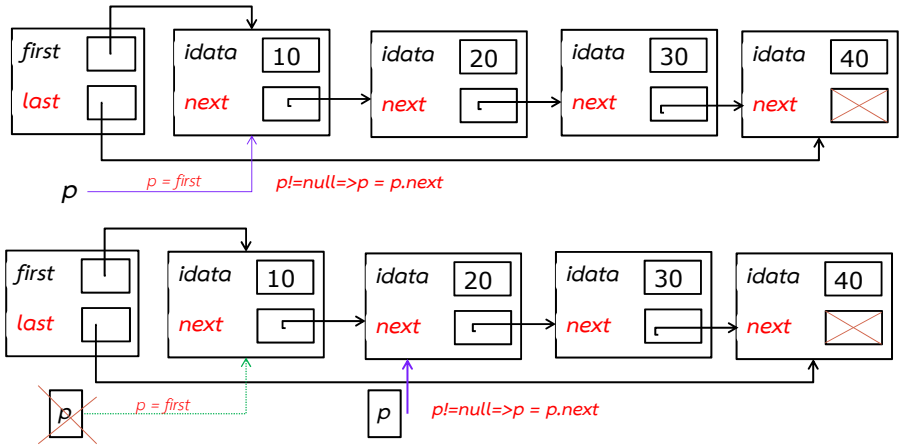
ถ้ามี 10 node จะต้อง next กี่รอบ

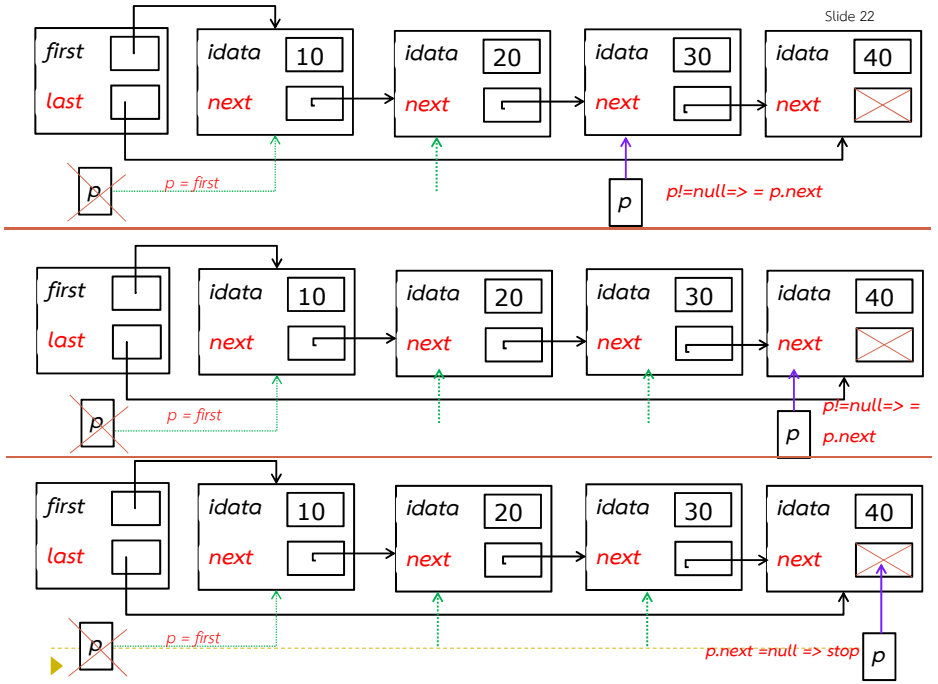
การแสดงค่าใน ลิสต์



1. สร้างตัววิ่ง (p)ซึ่งมีชนิดข้อมูลเป็นชนิดข้อมูลเดียวกับโหนด (Node)
2. ให้ตัววิ่งเริ่มวิ่งจากต้นลิสต์
3. เมื่อวิ่งไปถึงโหนดใดจะนำข้อมูลในโหนดนั้นมาปริ้นท์
4. ถ้ายังมีโหนดถัดไปจะเลื่อนไปที่โหนดถัดไป
5. วนลูปทำข้อ 3 จนกว่าตัววิ่งวิ่งไปครบทุกโหนดในลิสต์

การแสดงค่าใน ลิสต์





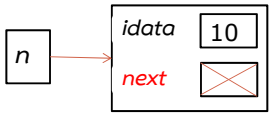
Pseudo Code และ Code ของการพิมพ์ข้อมูลในลิสต์ออกมาดู

1. Check
   if list is empty
      ปริ้นท์ 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 displayList()
{
  if (first==null) {
     System.out.println("List is Empty");
  }
  else
  {  Node p;    p = first;
     System.out.println("First=>Last: ");
     while (p!=null)
     {
          System.out.println(p.idata);
          p = p.next;
     }
   }
}
```


การเพิ่มโหนดลงไปในลิสต์

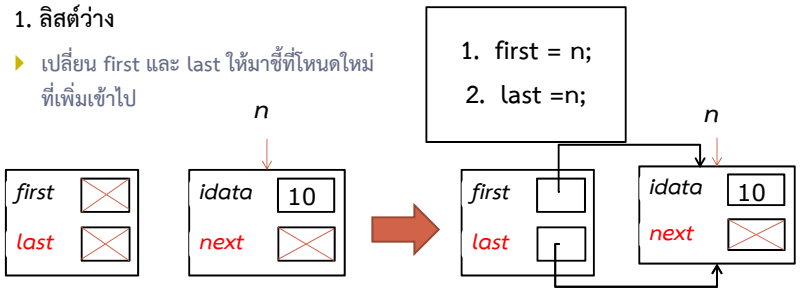
- ไม่ว่าจะเป็นการเพิ่มไปที่ตำแหน่งใด ขั้นตอนแรกต้องทำการสร้างโหนด และ ใส่ข้อมูลใหม่ลงไปในโหนดให้เรียบร้อยก่อน
  - เช่นในกรณี เก็บข้อมูลประเภท integer และต้องการเพิ่มโหนดที่มีข้อมูล 30
  - ภาษาจาวา *n = new Node (); n.idata = 30; n.next = null;*
  - หรือ *n = new Node(30,null);*



- แล้วค่อยทำการเพิ่มลงไปในลิสต์

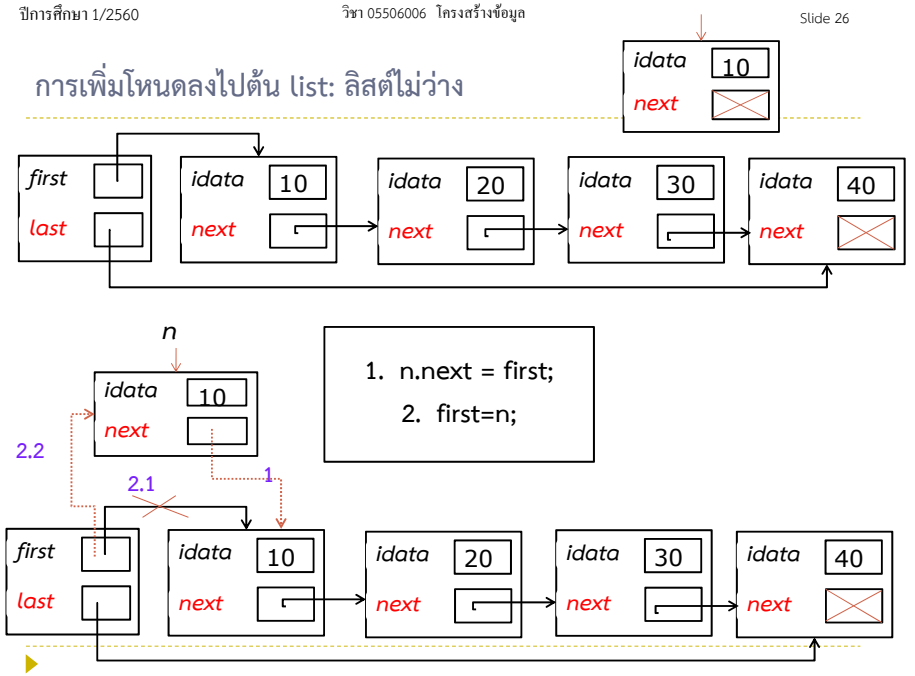
การเพิ่มโหนดลงไปต้น list

- มี 2 กรณี คือ
  - 1. ลิสต์ว่าง
    - เปลี่ยน first และ last ให้มาชี้ที่โหนดใหม่ที่เพิ่มเข้าไป



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

การเพิ่มโหนดลงไปต้น list: ลิสต์ไม่ว่าง



Pseudo Code และ Code ของการเพิ่มโหนดต้นลิสต์ออกมาดู

1. สร้างและใส่ข้อมูลลง node n

2. If list ว่าง

   เปลี่ยน first และ last ให้ชี้ไปที่เดียวกับ n

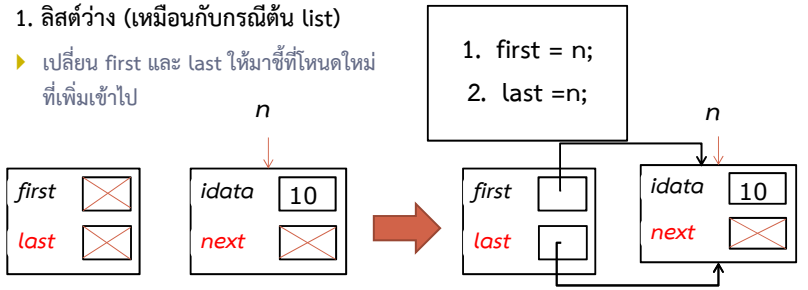
   Else

   - ให้ next ของโหนดใหม่ (n) ชี้ไปที่ first
   - เปลี่ยน first ให้ ชี้มาที่โหนดใหม่ n

```
public void insertFirst(int value)
{
  Node n = new Node(value,null);
  if (first==null) {
      first=n; last = n;
  }
  else
  {   n.next = first;
      first = n;
  }
}
```

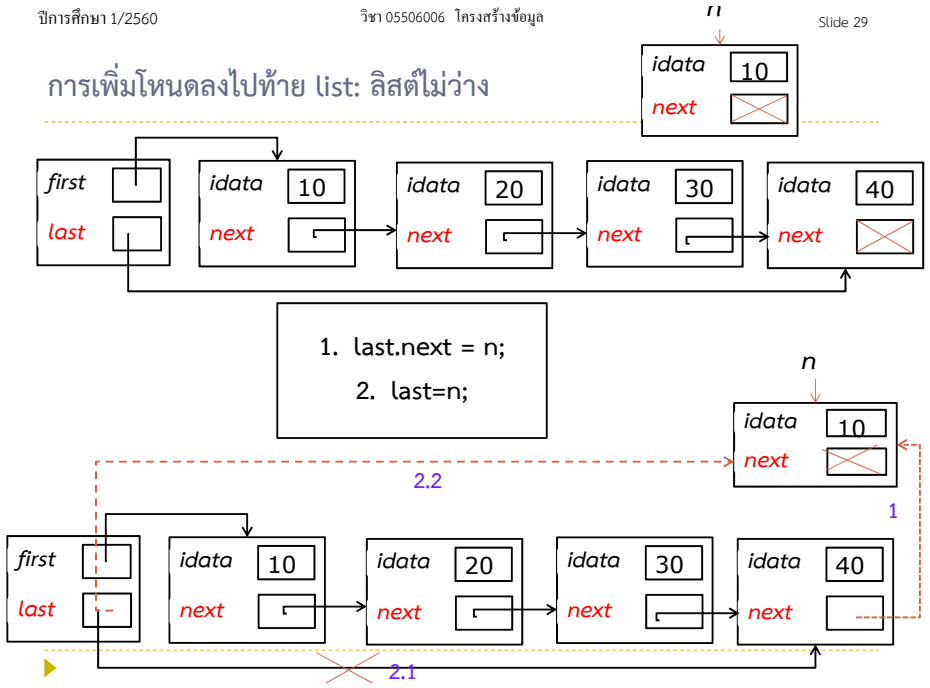

การเพิ่มโหนดลงไปท้าย list

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



  - 2. ลิสต์ไม่ว่าง
    - 1. ให้ n.next ชี้ไปที่ first เดิม;
    - 2. เปลี่ยน first ใหม่ให้ชี้ไปที่ n

การเพิ่มโหนดลงไปท้าย list: ลิสต์ไม่ว่าง



Pseudo Code และ Code ของการเพิ่มโหนดลงไปท้ายลิสต์

1. สร้างและใส่ข้อมูลลง node n

2. If list ว่าง

   เปลี่ยน first และ last ให้ชี้ไปที่เดียวกับ n

   Else

   - ให้ next ของ last ชี้ไปที่ n
   - เปลี่ยน last ให้ ชี้มาที่โหนดใหม่ n

```
public void insertLast(int value)
{
  Node n = new Node(value,null);
  if (first==null) {
       first=n; last = n;
  }
  else
  {    last.next = n;
       last = n;
  }
}
```


การลบ:

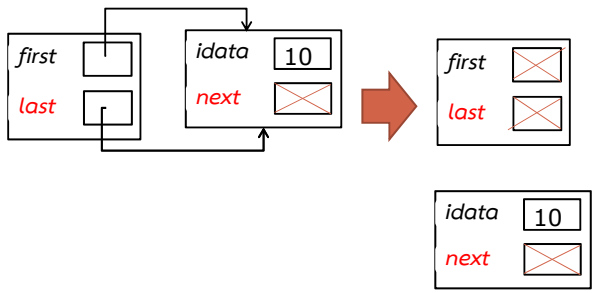
การลบต้น list และ การลบท้ายลิสต์

- 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:

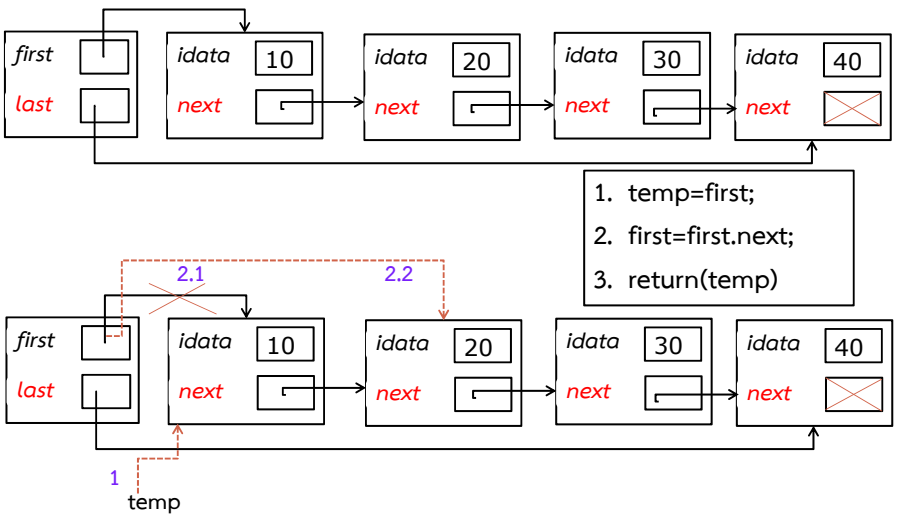
การลบกรณีมีสมาชิกตัวเดียว

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



การลบ:

การลบต้นลิสต์



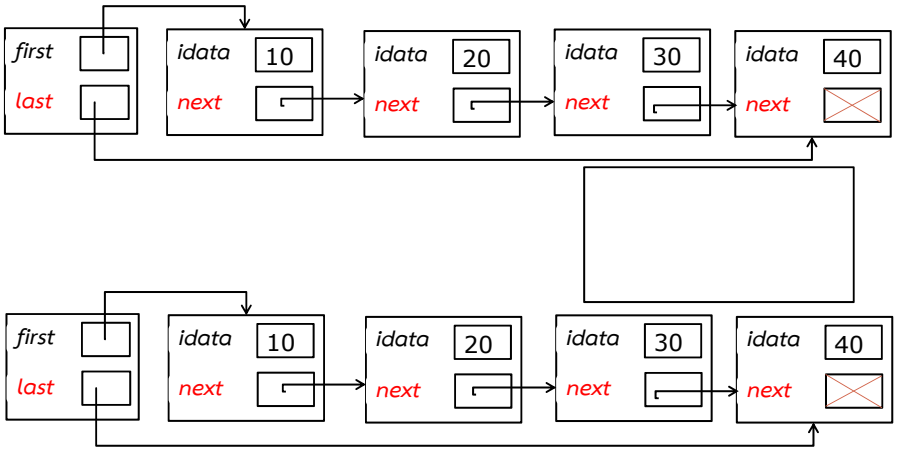
Pseudo Code และ Code ของการลบต้นลิสต์ออกมาดู

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
     return temp

```
public Node deleteFirst()
{
  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
      { first = first.next;}
      return(temp);
}
```

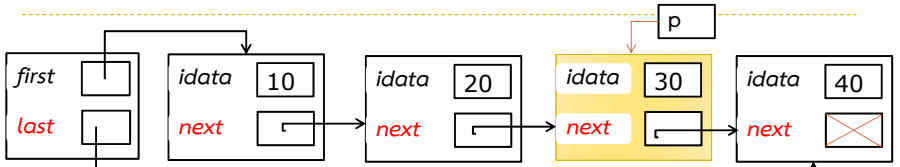

การลบ:

การลบท้ายลิสต์ ให้นักศึกษาลองวาดภาพการลบท้ายลิสต์



การลบ:

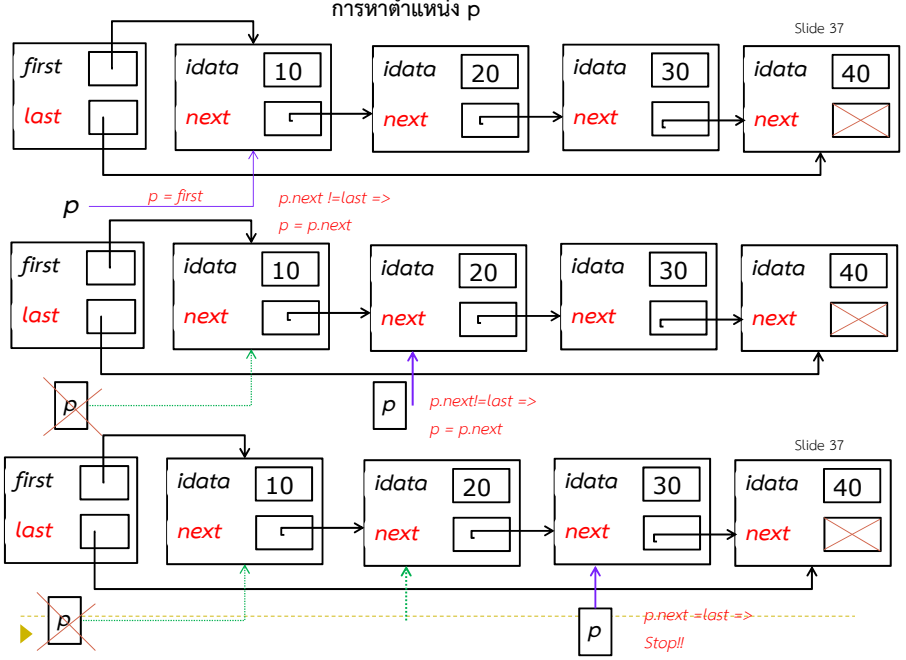
การลบท้ายลิสต์ จะต้องมีการหาว่าตำแหน่งก่อน last คือ ตำแหน่งอะไร



- เนื่องด้วย Single link list ไม่มีย้อนกลับ ดังนั้น เมื่อลบโหนดสุดท้ายแล้ว ถ้าไม่หาตำแหน่งโหนดก่อนสุดท้ายไว้ จะไม่สามารถเซตค่า last ใหม่ได้
- วิธีหา คือ
  - เนื่องจากตำแหน่งนี้อยู่ก่อน last ดังนั้น ถ้าตำแหน่งนี้คือ p ค่า p.next = last
  - ใช้วิธีมีตัววิ่งคล้ายกับ ตอนแสดงข้อมูล แต่คราวนี้ให้ p วิ่งไปถึงแค่ตำแหน่ง ก่อน last
    1. ให้ p เริ่มที่ first
    2. ถ้า p.next ไม่ใช่ last เลื่อน p ไปหนึ่งตำแหน่ง Loop กลับไปที่ 2

    เมื่อไหร่ p.next = last แสดงว่า p คือ ตำแหน่งก่อนสุดท้าย
- ทำการเซตให้ last ชี้ไปที่ p

การหาตำแหน่ง p



Pseudo Code และ Code ของการลบท้ายลิสต์ออกมาดู

1. If list ว่าง
   พิมพ์ list empty. Cannot
   Delete
   return null
   Else
   - *ให้ temp ชี้ไปที่เดียวกับ โหนด last*
   - If list มีสมาชิกตัวเดียว
     Set first และ last เป็น null
     Else
     ให้ p = first
     **ถ้า p.next != last ให้ เลื่อน p ไปโหนดถัดไป (p = p.next)
     Loop
     เปลี่ยน p.next เป็น null
     ให้ last = p
     return temp

```
public Node deleteFirst()
{
  Node temp;
  if (first==null) {
      System.out.println("List is empty.
Cannot delete");
      return(null);
  }
  else
  {   temp = last;  Node p = first;;
      if (first==last)
      { first = null; last = null;}
      else
      { while (p.next!=last) p=p.next;
        last = p; p.next = null;}
      return(temp);}
}
```


การบ้าน

1. จงเขียน ผังงานหรือโปรแกรมในการนับจำนวนสมาชิกในลิสต์ว่ามีทั้งหมดกี่โหนด
2. ให้ p คือ โหนดที่มีข้อมูลที่มีค่าเท่ากับ k

   2.1 จงเขียนผังงานหรือโปรแกรมหรือ PseudoCode หาตำแหน่งของ Node p ใน ลิสต์

   2.2 จงเขียนผังงานหรือโปรแกรมหรือ PseudoCode หาค่าของ Node p ใน ลิสต์

   method Node findkey(int k)

อ้างอิง

- สไลด์ประกอบการสอนวิชา SE 311 Algorithms Design and Analysis โดย ผศ.ดร. สมศรี บัณฑิตวิไล