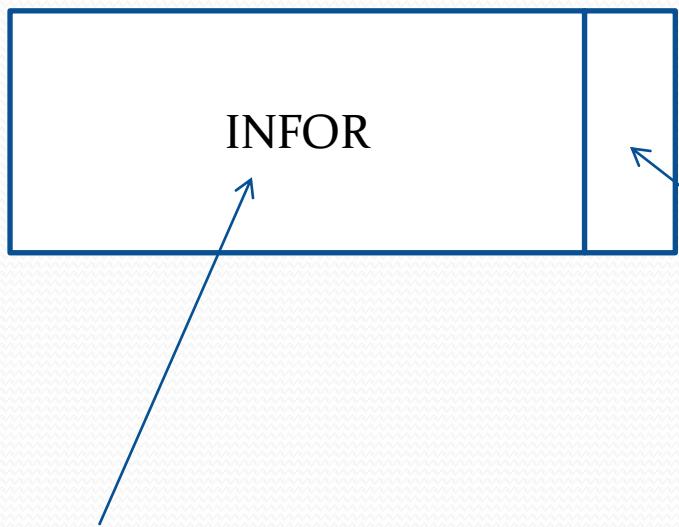


Lists

รายการโยง (Link Lists)

- เป็นข้อมูลแบบรายการต่อเนื่อง
- เชื่อมโยงจากรายการที่หนึ่ง ไปจนถึงรายการสุดท้าย
- ข้อมูลแต่ละรายการเรียกว่า โนด (Node)
- การเชื่อมโยงต่อกันใช้ตัวเชื่อมโยง ลิงค์ (Link)
- การเชื่อมโยงสามารถเชื่อมโยงได้ทั้งทางเดียว และสองทาง
- ข้อดี สามารถเพิ่ม ลบ ข้อมูลแต่ละรายการได้โดยง่าย

มุ่งมองลิงค์ลิสต์

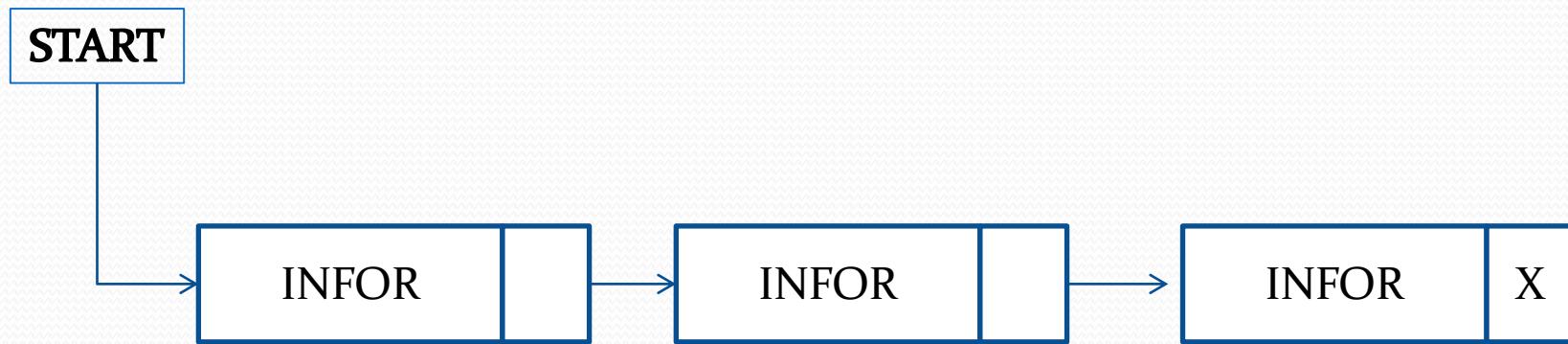


ข้อมูลในโนด
(Information)

โนด (Node)

ตัวเชื่อมไปยังโนดต่อไป
(Link Pointer)

ມູນມອງລົງຄໍລືສົດ

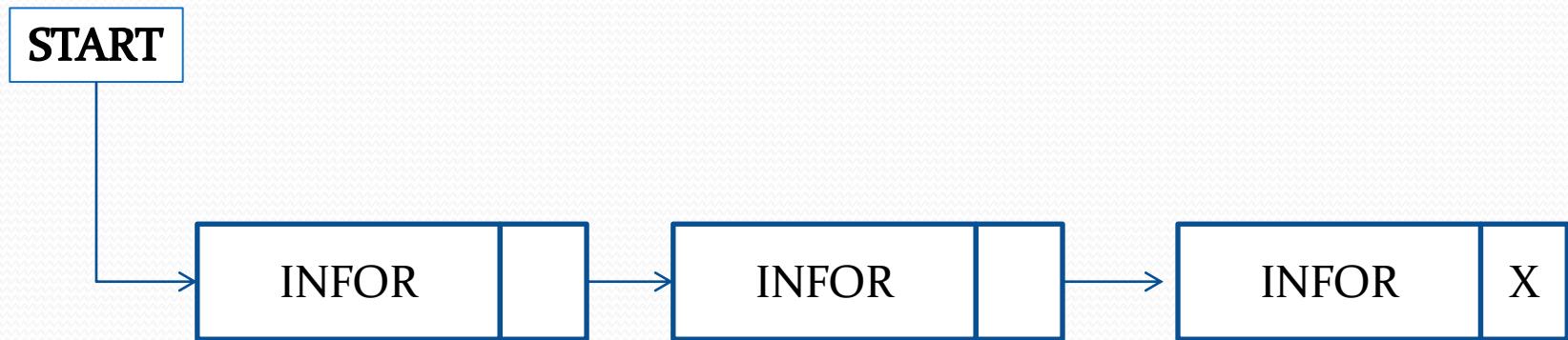


คำศัพท์ที่เกี่ยวข้องในเนื้อของบทนี้

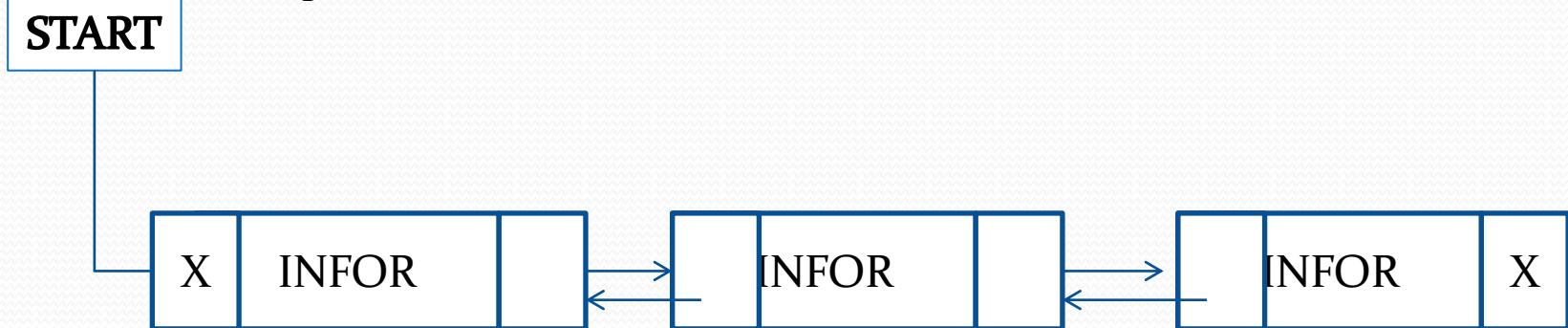
- Node
- Link
- INFOR
- PTR
- NULL
- One-way
- Two-ways
- Start
- Last
- Head
- AVAIL
- LOC
- LOCA
- LOCB
- SAVE

ประเภทของลิงค์ลิสต์

- One-way Link Lists

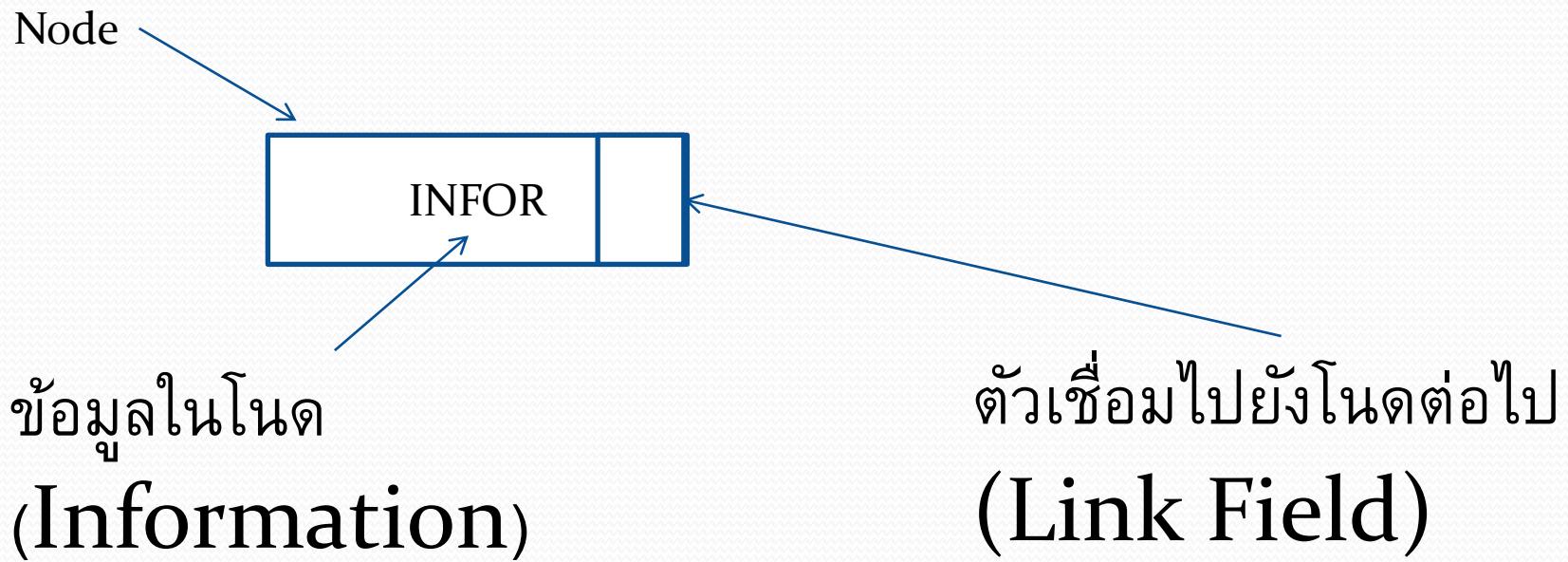


- Two-ways Link Lists

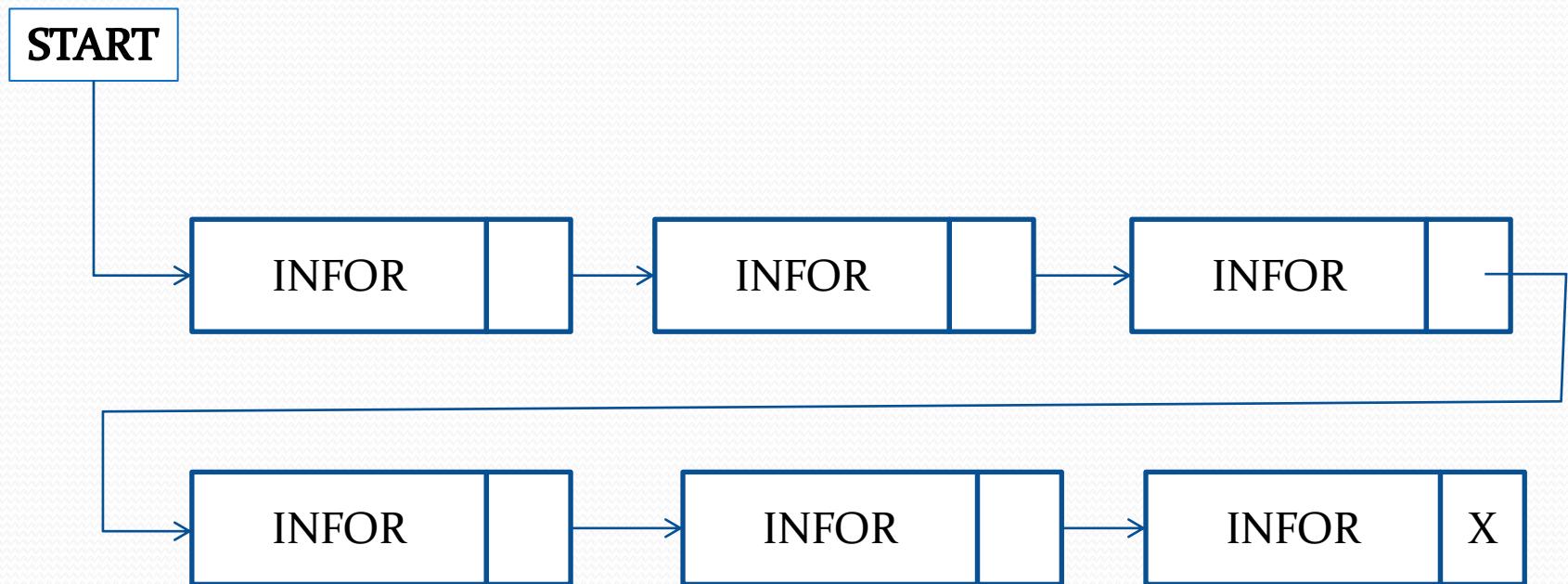


ลิงค์ลิสต์ทางเดียว(One-way Link Lists)

- เรียกแต่ละรายการข้อมูลว่า Node ประกอบด้วย 2 ส่วนคือ
 - ส่วนข้อมูล เรียกว่า **Information** ของแต่ละอิเลเมนต์
 - ส่วนลิงค์ฟิลด์ (**Link Field**)

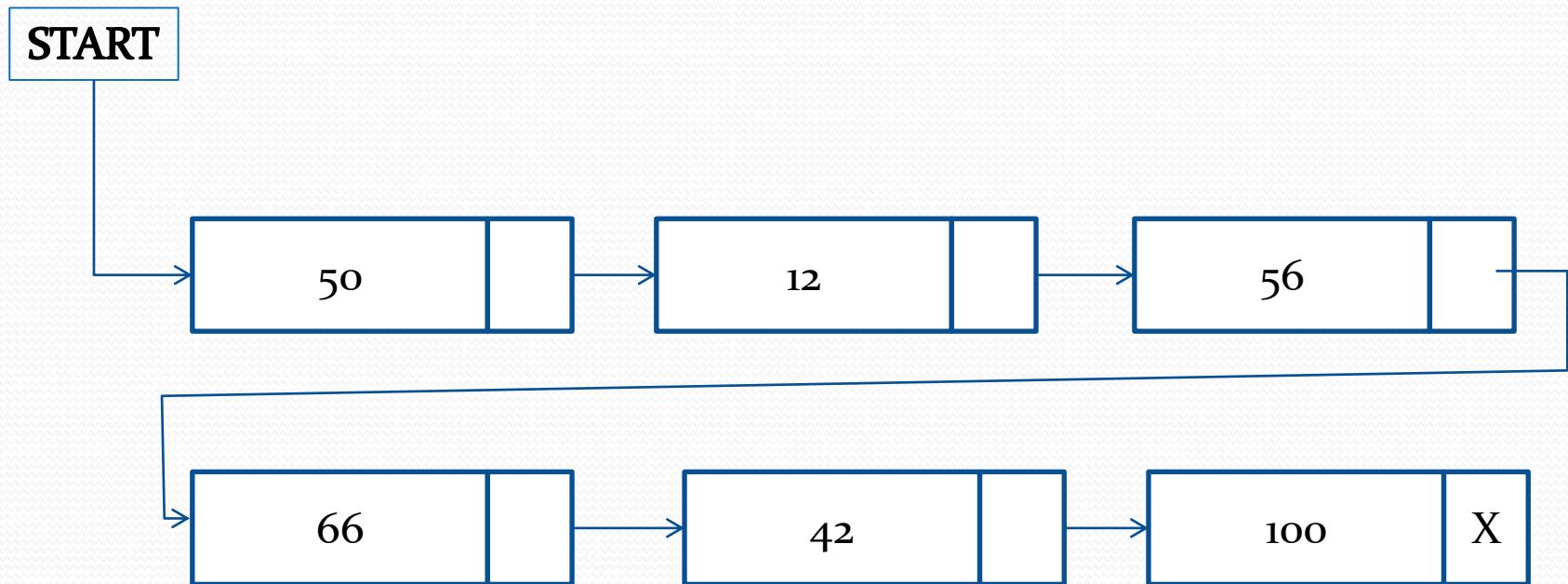


คุณลักษณะทั่วไป



X หมายถึง NULL

ตัวอย่างการเก็บข้อมูล



X หมายถึง NULL

ตัวอย่างการเก็บข้อมูล

START

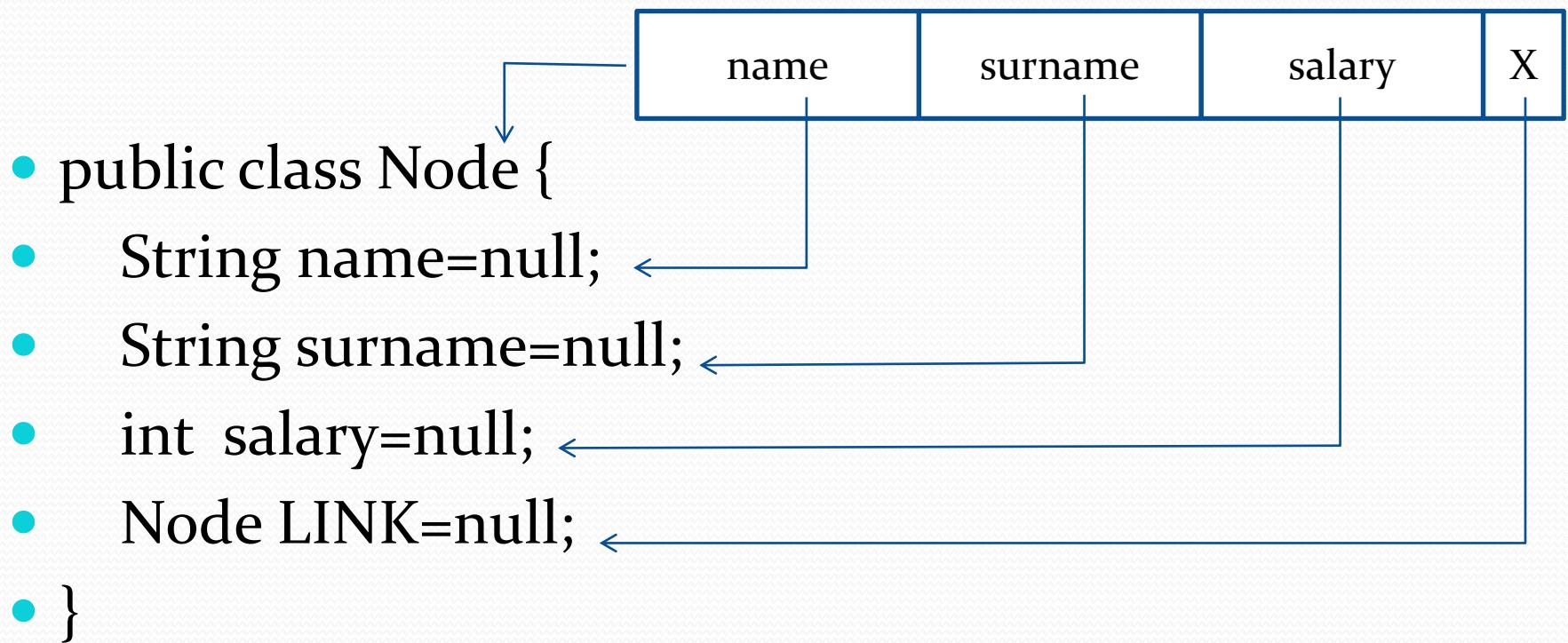


X หมายถึง NULL

แนวทางอิมเพลี่เมนต์

- public class Node {
- Object INFOR;
- Node LINK=null;
- }





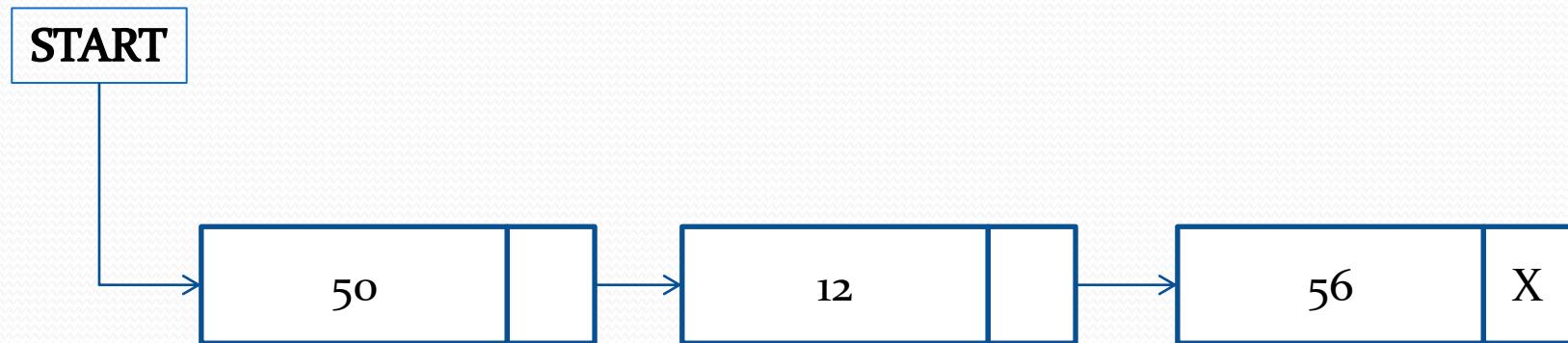
ตัวอย่างการสร้างโนดและบรรจุข้อมูล

- การสร้างโนดด้วยภาษาจาวา
 - Node NodeA = new Node();
- บรรจุข้อมูล
 - NodeA.INFOR = 50;

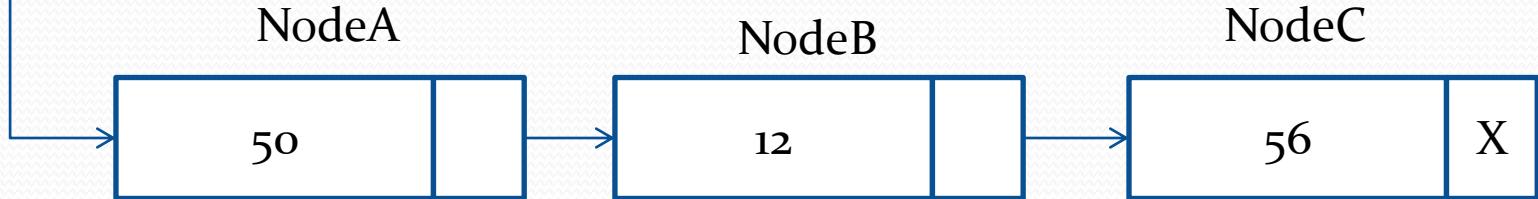
NodeA

50	X
----	---

การสร้างเป็นลิงค์ลิสต์



START



- Node START;
- Node NodeA = new Node();
- NodeA.INFO = 50;
- Node NodeB = new Node();
- NodeB.INFO = 12;
- Node NodeC = new Node();
- NodeC.INFO = 56;
- START.LINK = NodeA;
- NodeA.LINK = NodeB;
- NodeB.LINK=NodeC

กิจกรรมนักศึกษา

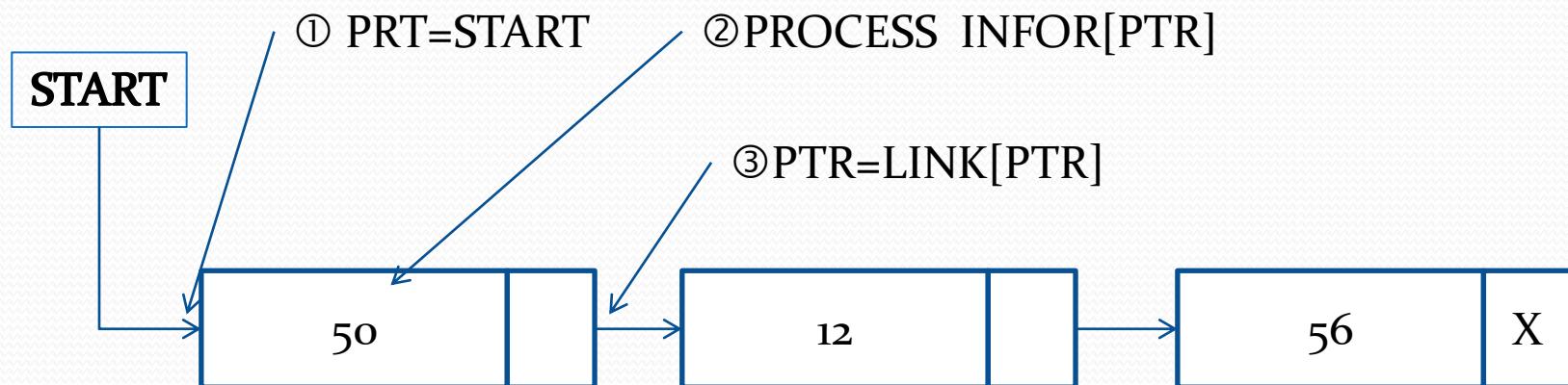
การดำเนินการกับลิงค์ลิสต์

- การท่อง
- การเพิ่ม
- การลบ

การท่องลิงค์ลิสต์(Traversing a Link Lists)

- เริ่มต้นด้วยการนำเอาพอยเตอร์ PTR ไปชี้ที่ START
- แล้วตามเข้าไปดำเนินการกับ INFOR ในแต่ละโนด
- หลังจากนั้นตามลิงค์ LINK ของแต่ละโนดไป จนกระทั่ง LINK เป็น NULL

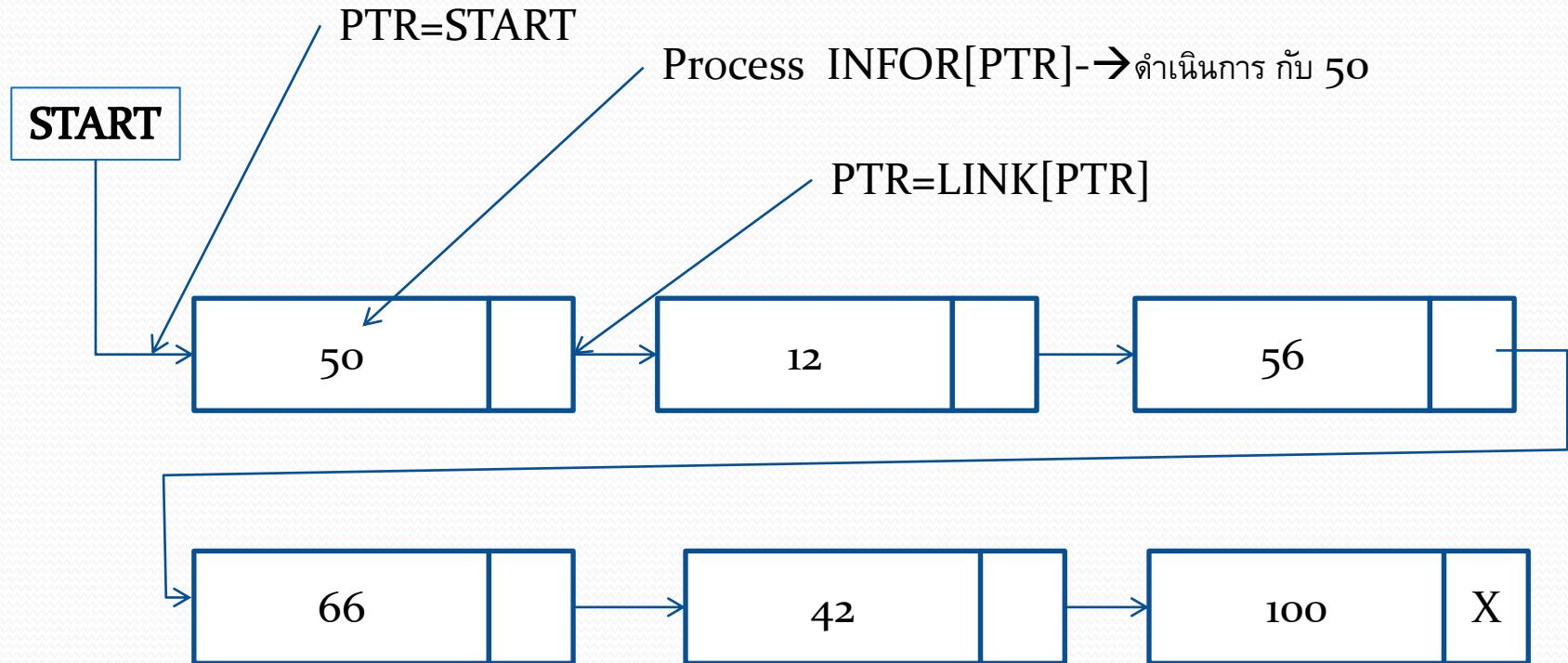
การท่องลิงค์ลิสต์



ดำเนินการ ② ③ จนกระทั่ง $\text{LINK}[\text{PRT}] = \text{NULL}$

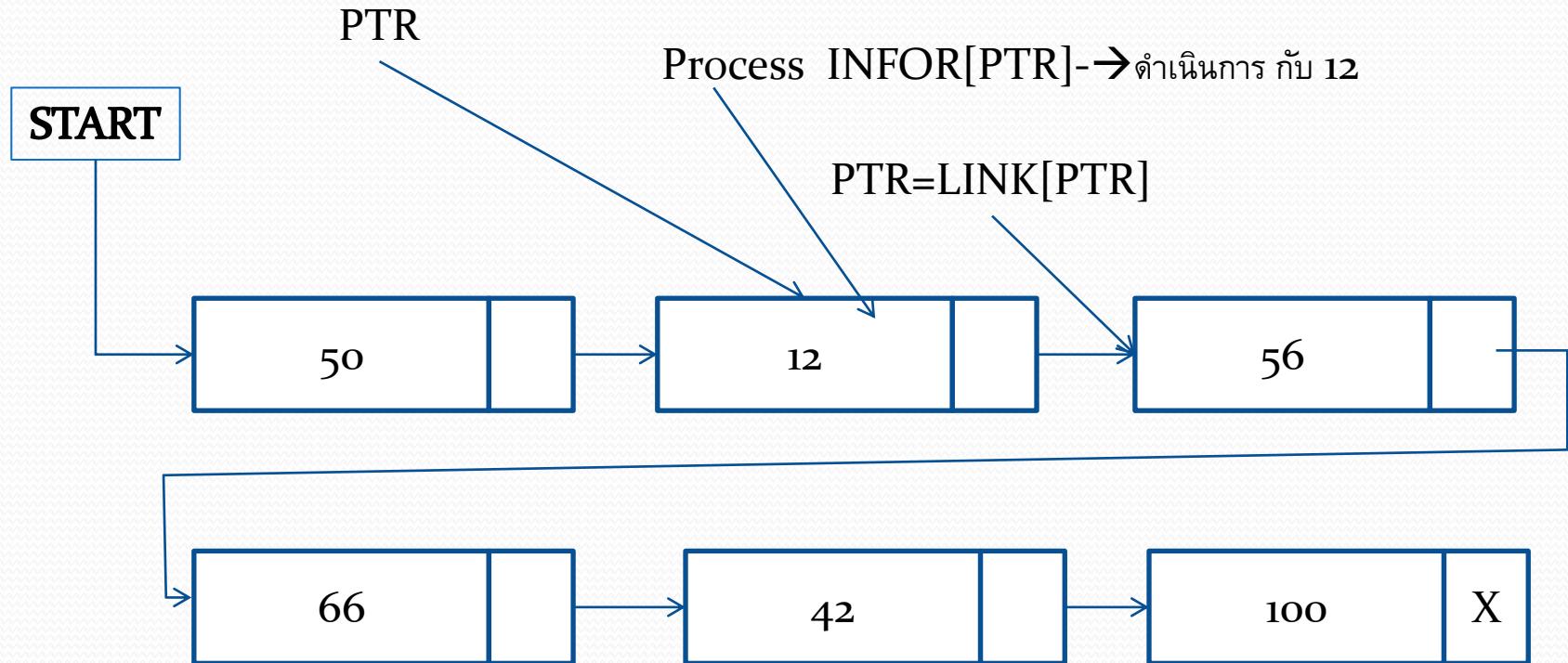
- Algorithm Traversing Link Lists
- Input : LIST, PTR, START
- Output : All nodes in LIST are traversed
- PTR=START
- While PTR!=NULL Do
 - Process INFOR[PTR]
 - PTR=LINK[PTR]
- End While
- Return

ตัวอย่างการท่องลิงค์ลิสต์ตามอัลกอริทึม



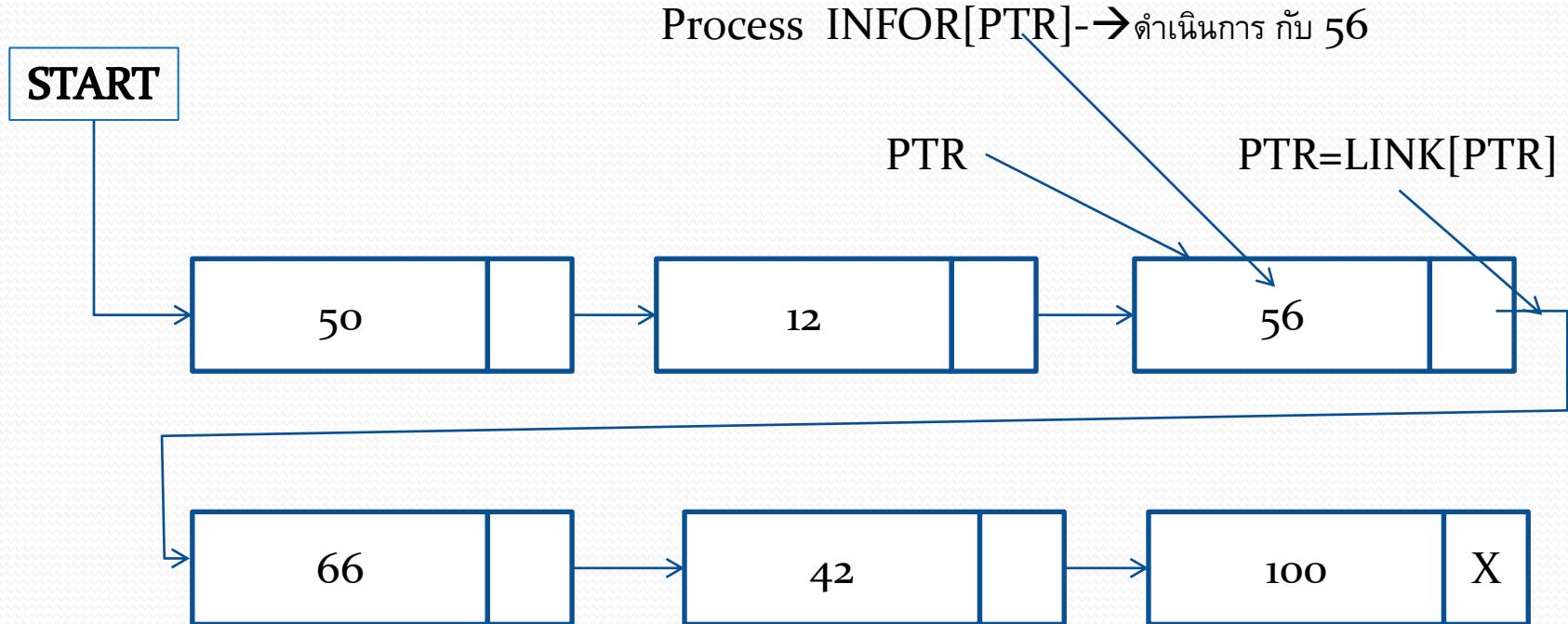
PTR=NULL? No, Loop While

ตัวอย่างการท่องลิงค์ลิสต์ตามอัลกอริทึม



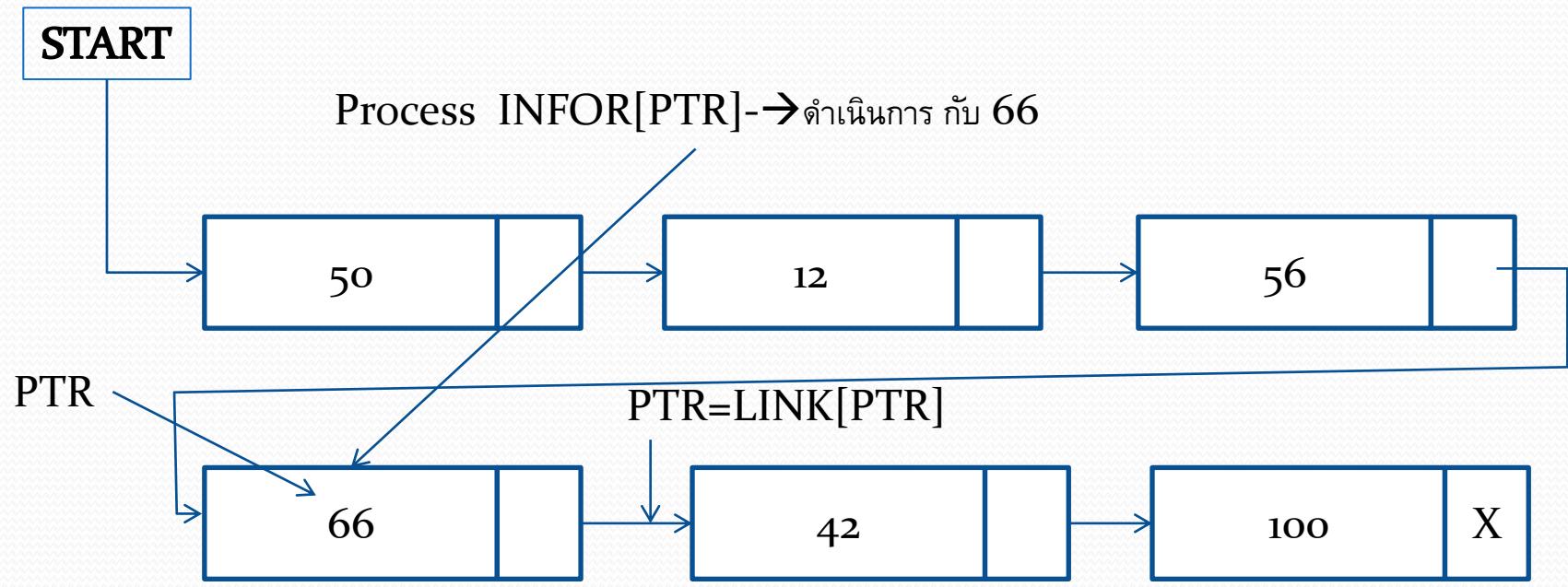
PTR=NULL? No, Loop While

ตัวอย่างการท่องลิงค์ลิสต์ตามอัลกอริทึม



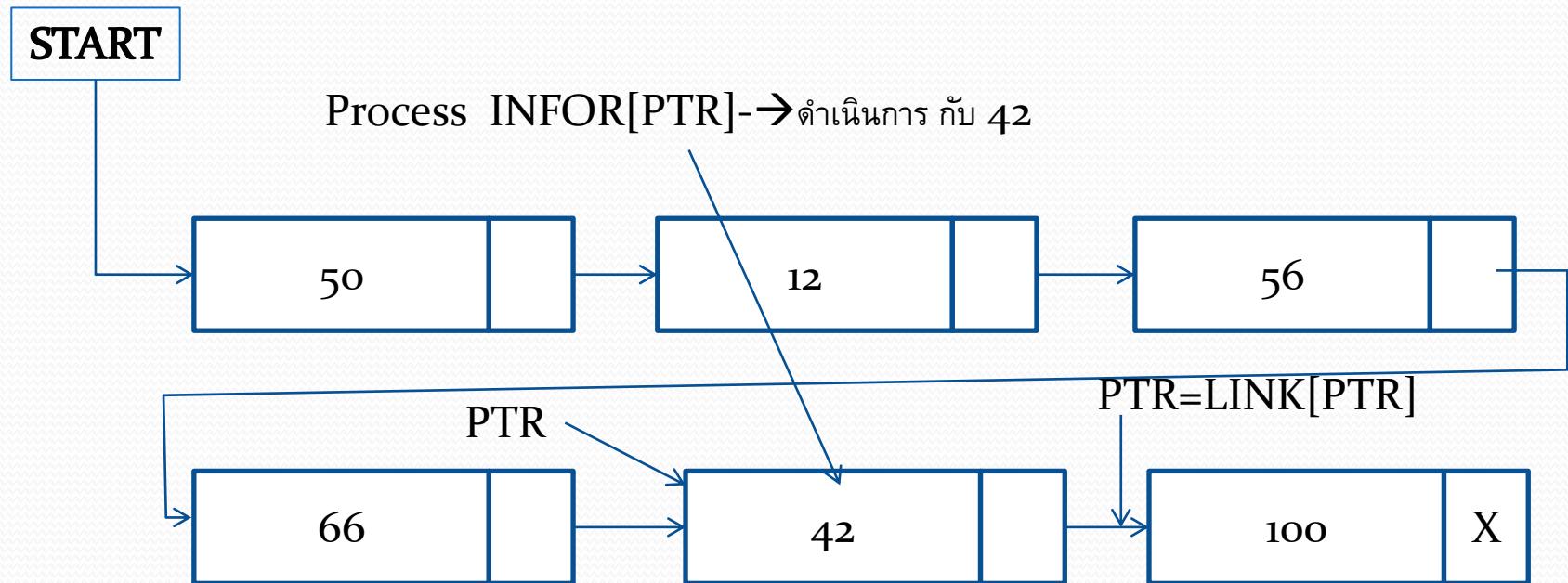
PTR=NULL? No, Loop While

ตัวอย่างการท่องลิงค์ลิสต์ตามอัลกอริทึม



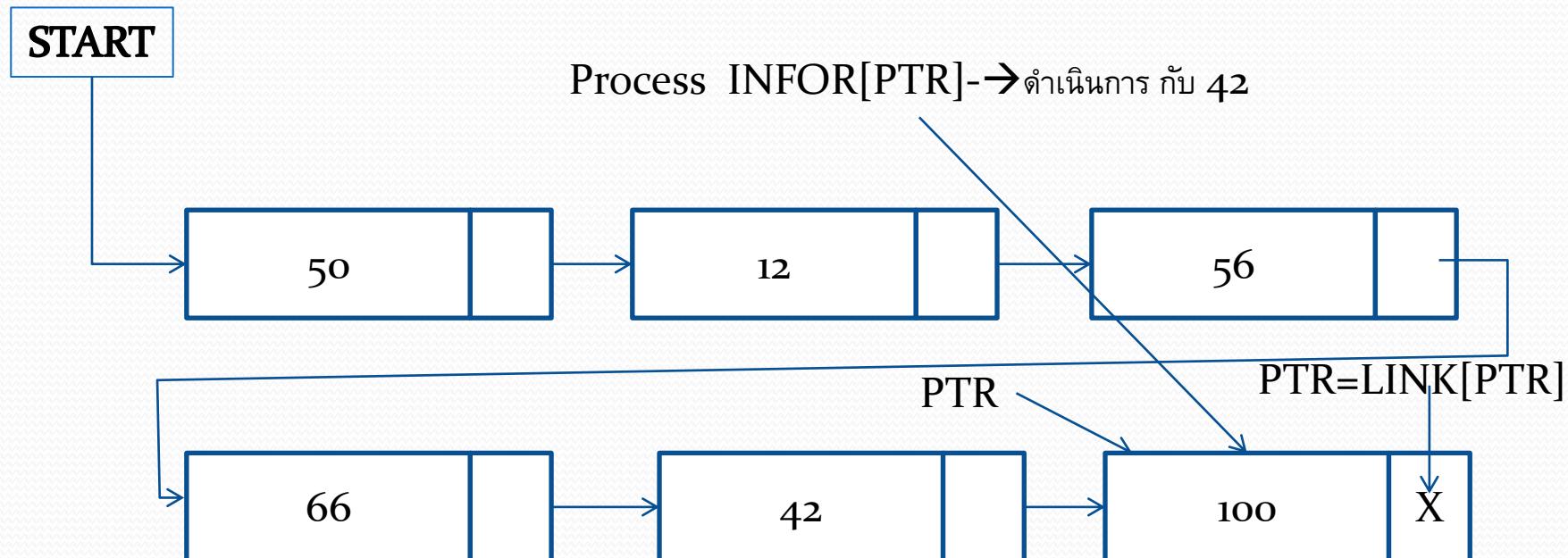
PTR=NULL? No, Loop While

ตัวอย่างการท่องลิงค์ลิสต์ตามอัลกอริทึม



PTR=NULL? No, Loop While

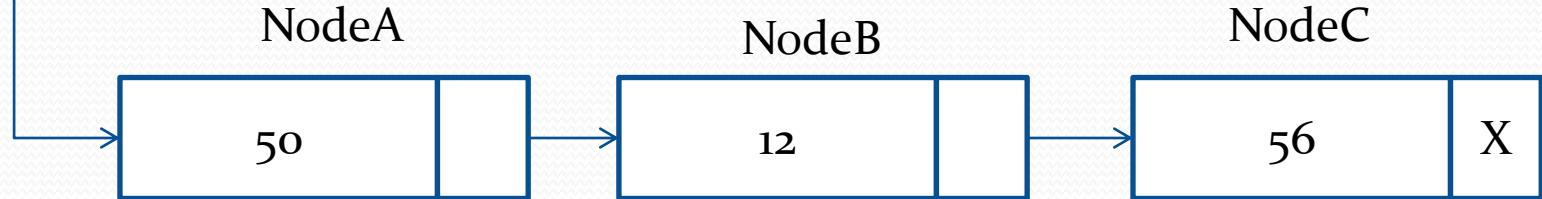
ตัวอย่างการท่องลิงค์ลิสต์ตามอัลกอริทึม



PTR=NULL? Yes, End While Loop

อิมพลีเมนต์

START



- Node START=null;
- Node NodeA = new Node();
- NodeA.INFO = 50;
- Node NodeB = new Node();
- NodeB.INFO = 12;
- Node NodeC = new Node();
- NodeC.INFO = 56;
- START.LINK = NodeA;
- NodeA.LINK = NodeB;
- NodeB.LINK=NodeC

```
• public class Algor51TraversingList {  
•     public static void main(String[] args) {  
•         Node START=null;  
•         Node NodeA = new Node();  
•         NodeA.INFOR = 50;  
•         Node NodeB = new Node();  
•         NodeB.INFOR = 12;  
•         Node NodeC = new Node();  
•         NodeC.INFOR = 56;  
•         START.LINK = NodeA;  
•         NodeA.LINK = NodeB;  
•         NodeB.LINK=NodeC  
•         Node PTR=null;  
•         PTR=START;  
•         while(PTR!=null){  
•             System.out.println(PTR.INFOR); //Apply PROCESS to INFOR[PTR]  
•             PTR=PTR.LINK;  
•         }  
•     }  
• }
```

ส่วนการกำหนดค่าให้กับโนด

อัลกอริทึมหลัก

กิจกรรมนักศึกษา

- รันโปรแกรมทดสอบ
- ดัดแปลงให้เข้ากับโจทย์ที่นักศึกษากำหนดเอง

การค้นหาในลิงค์ลิสต์ (Searching a Link Lists)

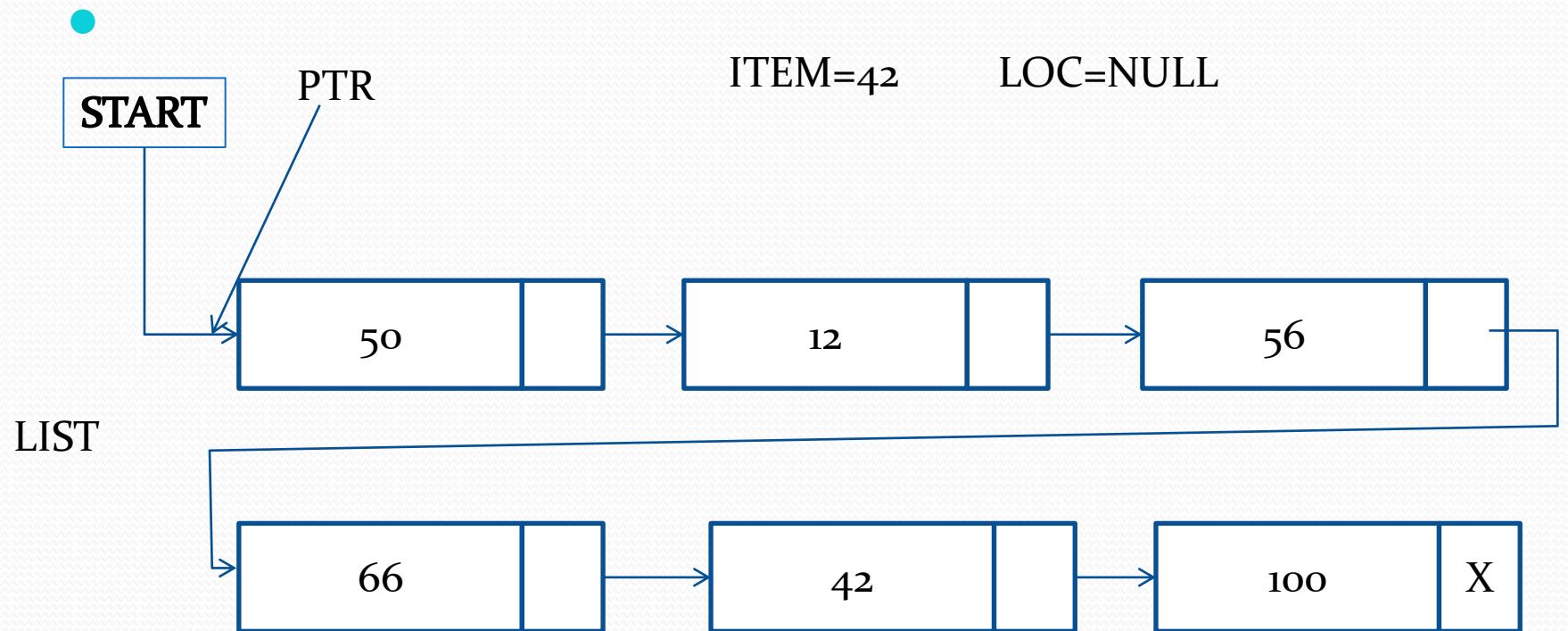
- หลักการค้นหา คือ
 - กำหนดค่าเริ่มต้น จาก โนดแรก
 - เทียบข้อมูลว่า เท่ากับ ข้อมูลในโนดหรือไม่
 - หากเท่ากัน ให้รายงานตำแหน่งโนดนั้น และออกจาก การค้นหา
 - หากไม่เท่ากัน ให้เลื่อนพอยเตอร์ไปยังโนดถัดไป
 - ดำเนินการจนกระทั่งพอยเตอร์ที่ชี้ต่อไปนั้น เป็นค่า null (ตัวสุดท้าย)

แนวทางของอัลกอริทึม

- กำหนด ITEM คือข้อมูลที่ต้องการค้นหา
- LOC คือตำแหน่งที่พบ กำหนดค่าเริ่มต้นเป็น NULL
- กำหนด PTR=START
- วนดำเนินการ ในขณะที่ PTR!=NULL
 - เทียบ ITEM = INFOR[PTR] หรือไม่
 - หากเท่ากัน LOC=PTR
 - หากไม่เท่ากัน PTR=LINK[PTR]
 - วนกลับไปดำเนินการ

- Algorithm SEARCH Item in Link Lists
- Input : LIST, PTR, START, LOC, ITEM
- Output : LOC
- PTR=START
- While PTR!=NULL Do
 - IF ITEM= INFOR[PTR] Then
 - LOC = PTR
 - Return LOC
 - Else
 - PTR=LINK[PTR]
 - End IF
 - LOC=NULL
- End While
- Return LOC

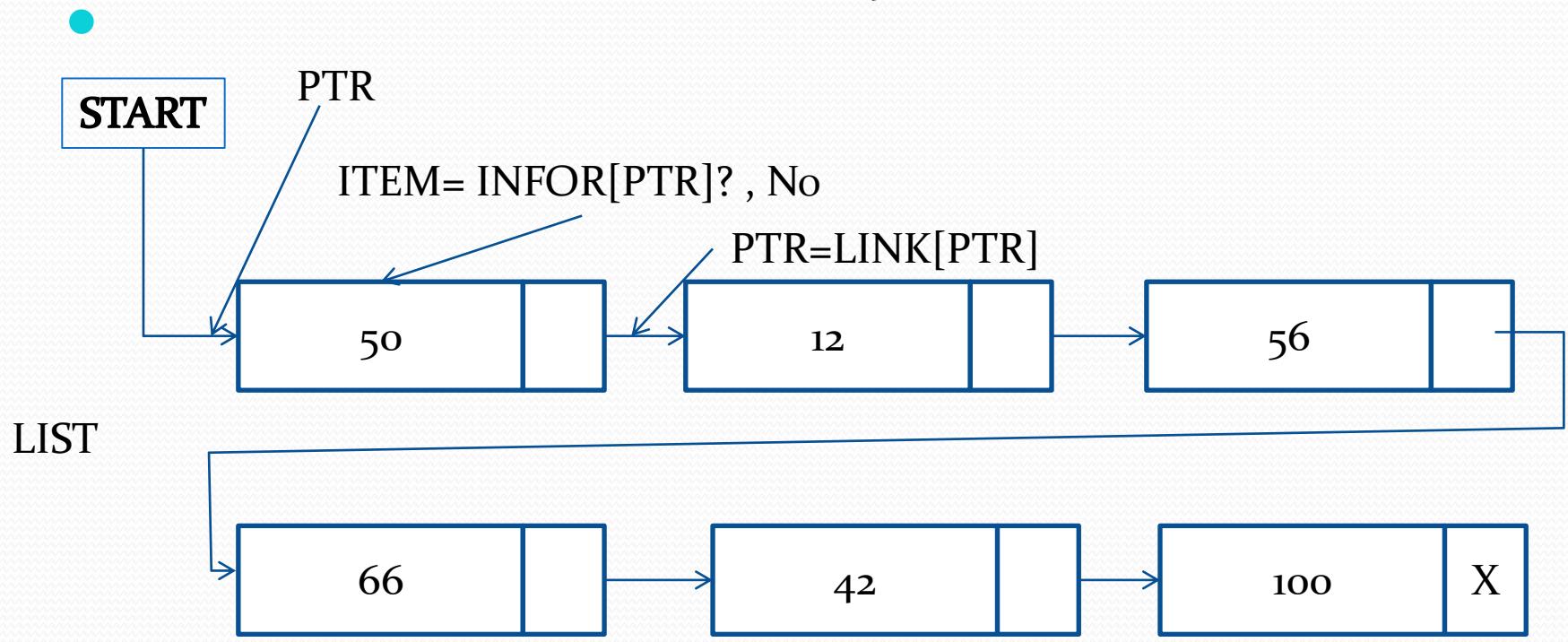
แนวทางการค้นตามอัลกอริทึม



แนวทางการค้นตามอัลกอริทึม

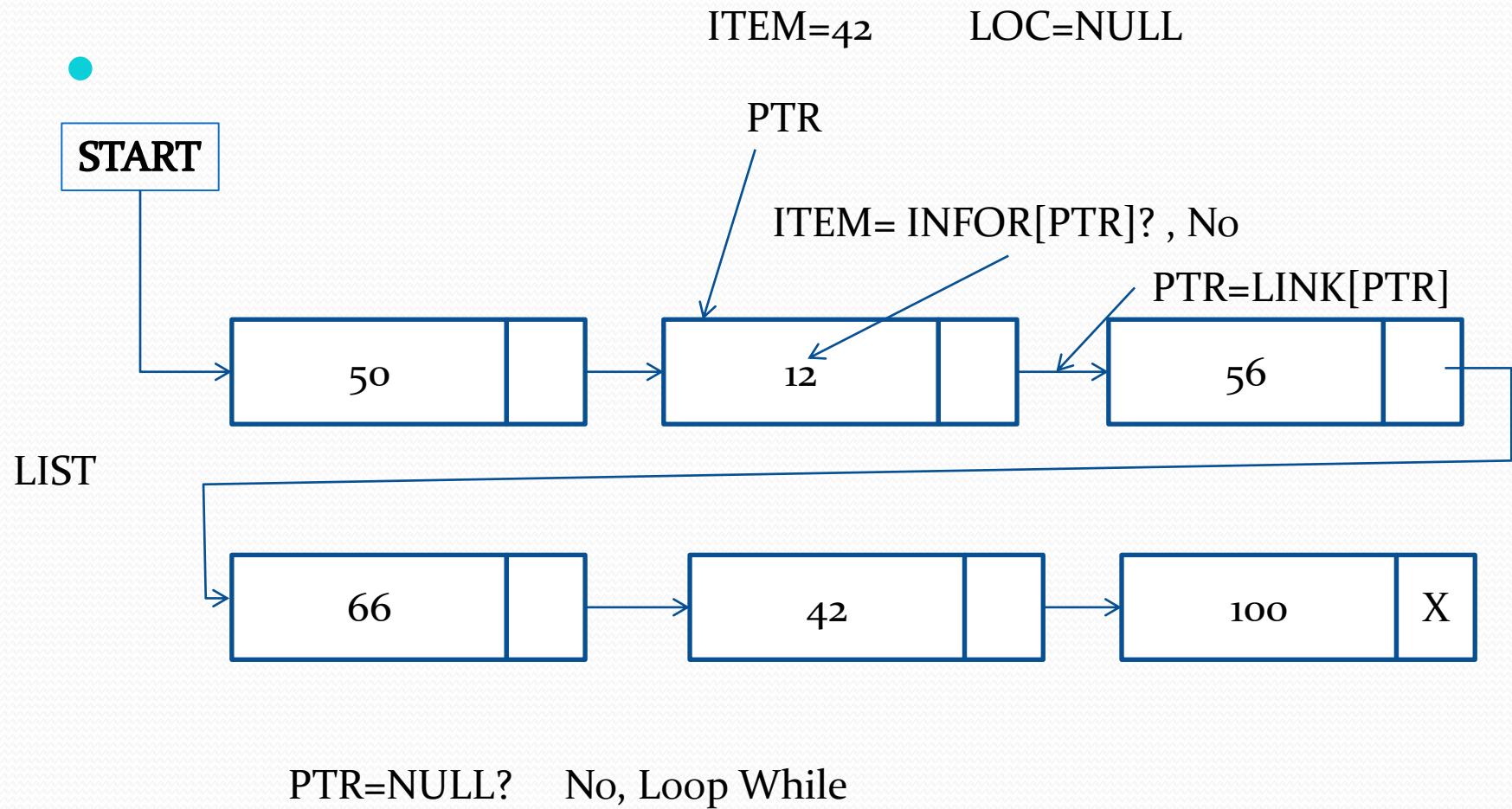
ITEM=42

LOC=NULL

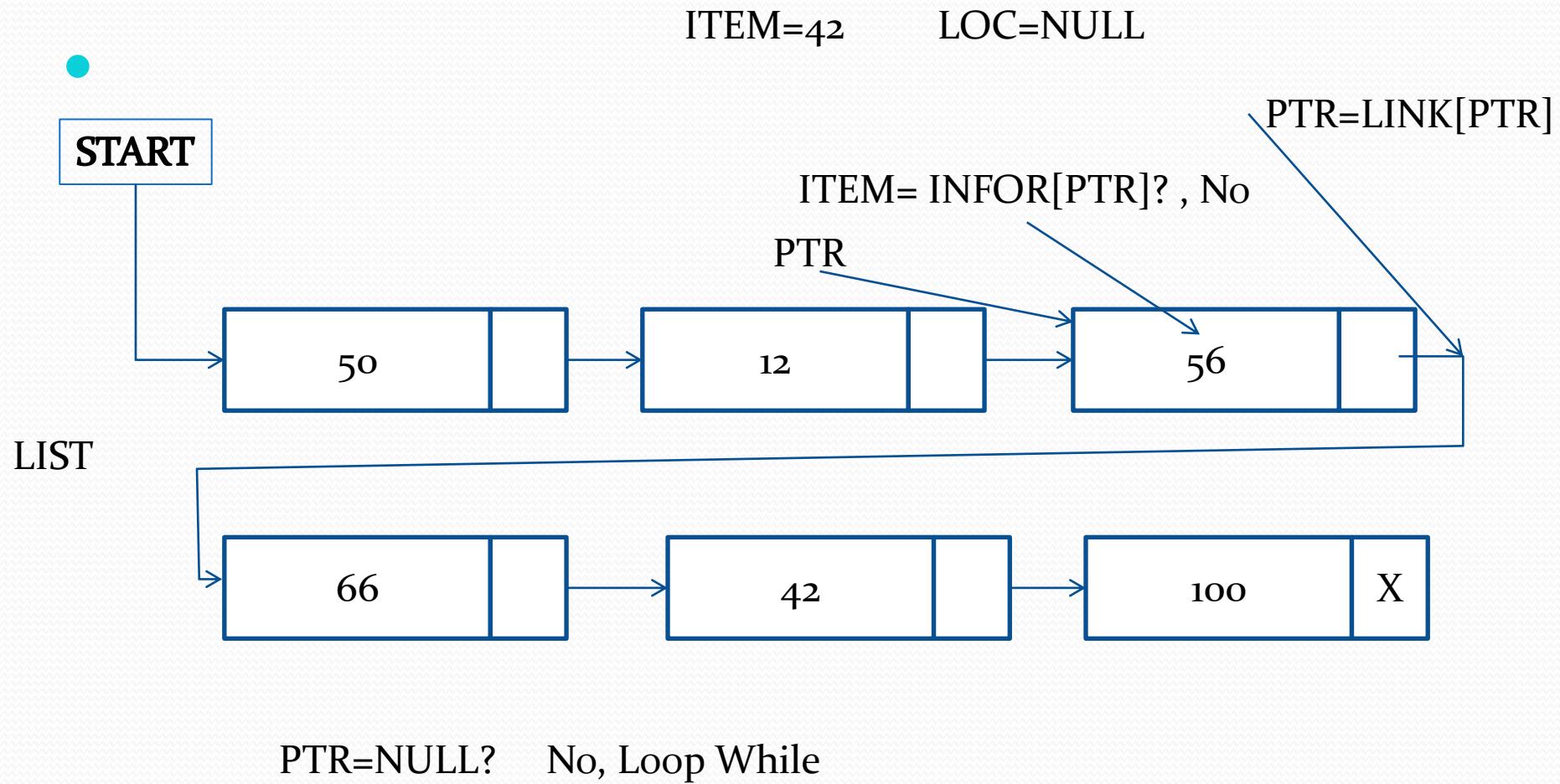


PTR=NULL? No, Loop While

แนวทางการค้นตามอัลกอริทึม



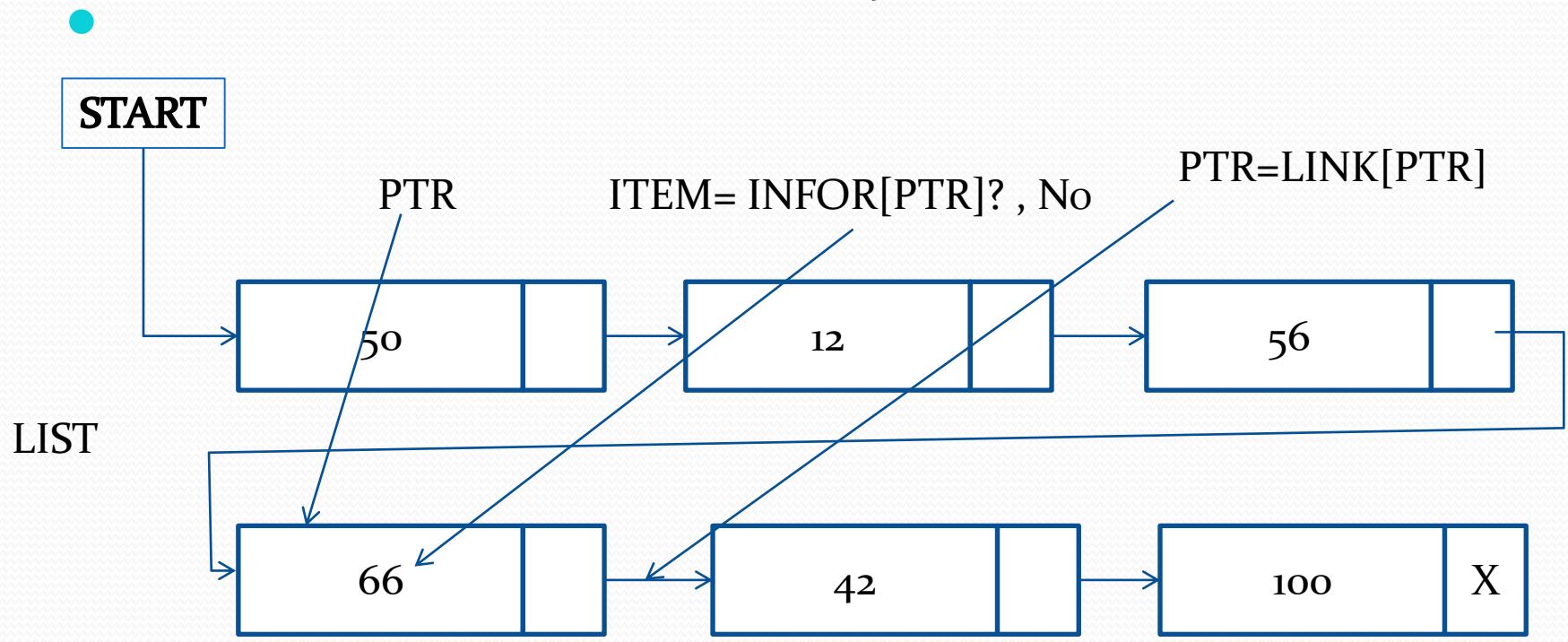
แนวทางการค้นตามอัลกอริทึม



แนวทางการค้นตามอัลกอริทึม

ITEM=42

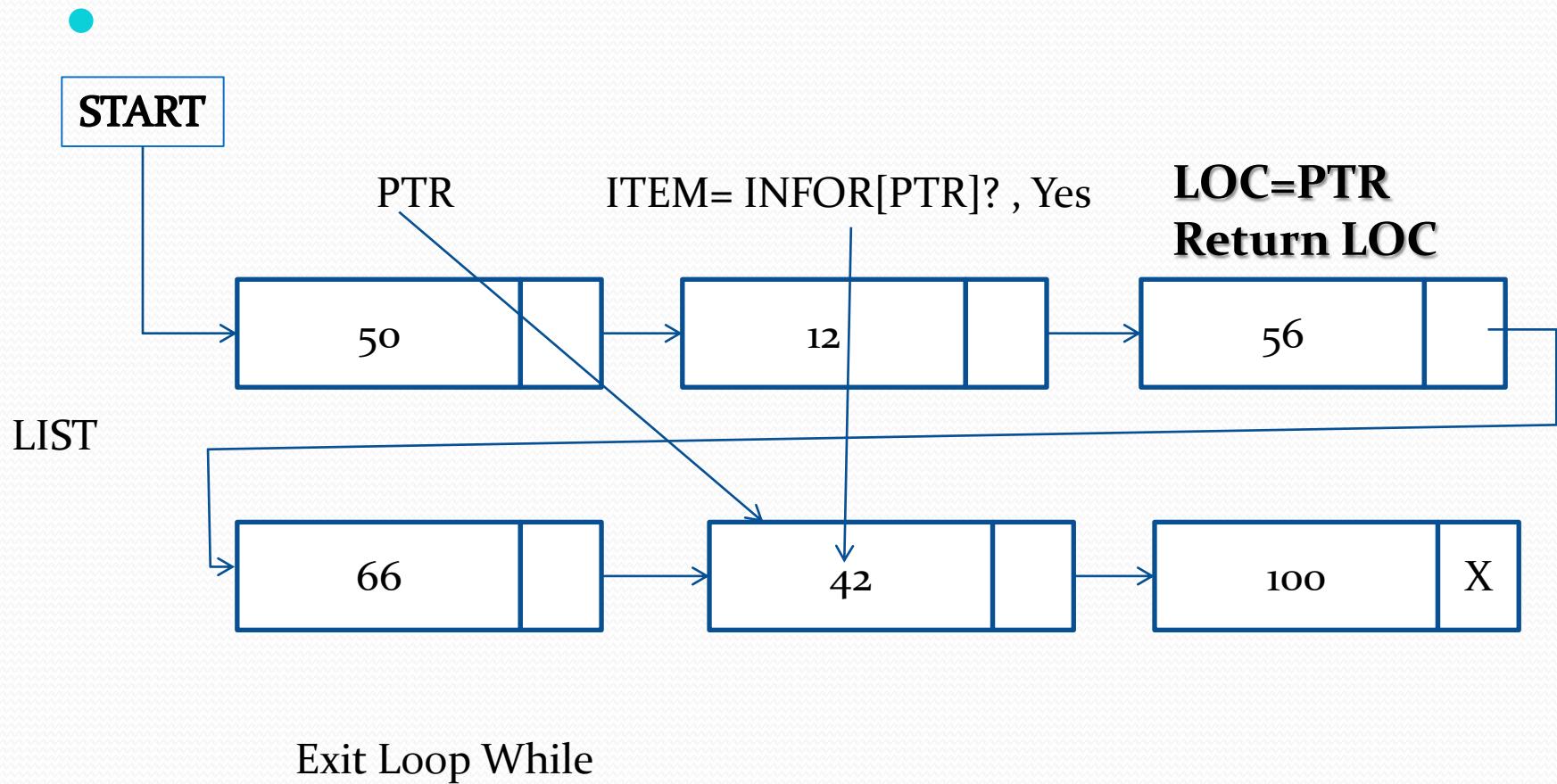
LOC=NULL



PTR=NULL? No, Loop While

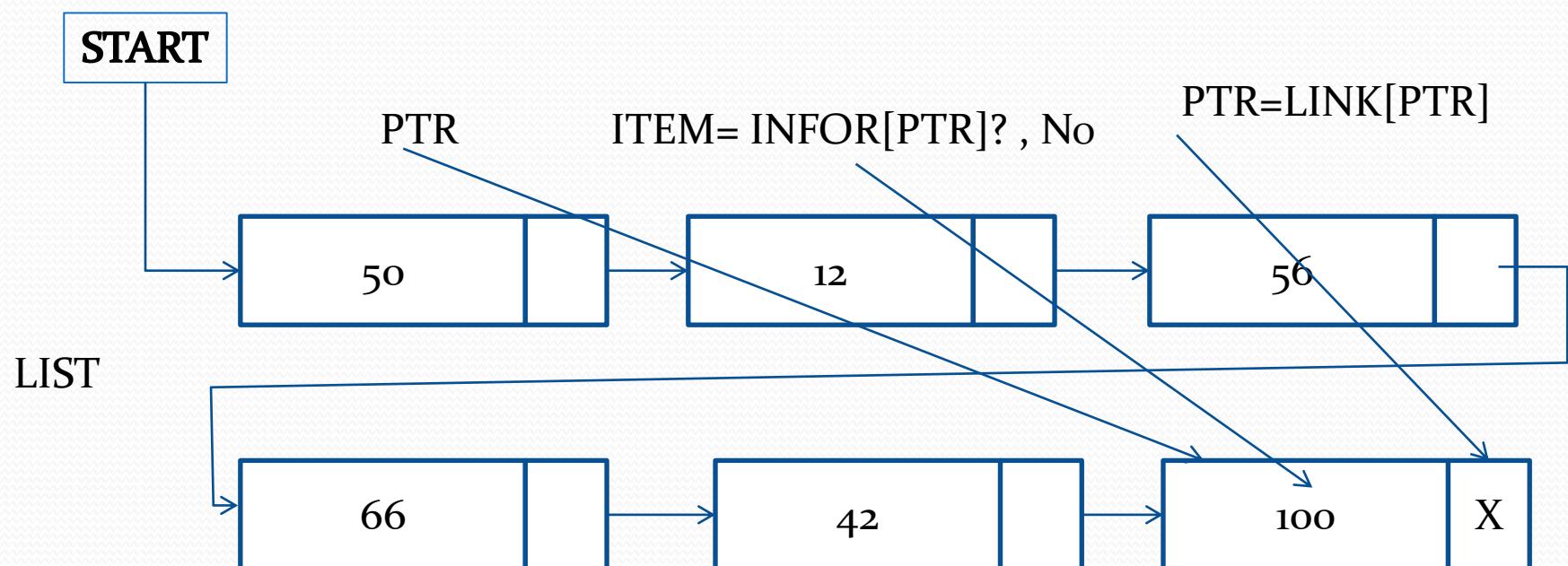
แนวทางการค้นตามอัลกอริทึม

ITEM=42



แนวทางการค้นตามอัลกอริทึม

- กรณีไม่พบ



PTR=NULL? Yes, Exit Loop While

**LOC=NULL
Return**

อิมพลีเมนต์

- package linklistimplementation;
- public class Algor52SEARCH {
- public static void main(String[] args) {
- Node START = new Node(); START.INFOR = 10;
- Node NodeA = new Node(); NodeA.INFOR = 20;
- Node NodeB = new Node(); NodeB.INFOR = 30;
- Node NodeC = new Node(); NodeC.INFOR = 40;
- Node NodeD = new Node(); NodeD.INFOR = 50;
- Node NodeE = new Node(); NodeE.INFOR = 60;
- Node NodeF = new Node(); NodeF.INFOR = 70;
- Node NodeG = new Node(); NodeG.INFOR = 80;
- Node NodeH = new Node(); NodeH.INFOR = 90;
- Node NodeI = new Node(); NodeI.INFOR = 100;
- START.LINK = NodeA; NodeA.LINK = NodeB; NodeB.LINK = NodeC;
- NodeC.LINK = NodeD; NodeD.LINK = NodeE; NodeE.LINK = NodeF;
- NodeF.LINK = NodeG; NodeG.LINK = NodeH; NodeH.LINK = NodeI;
- //Algorithm 5.2 SEARCH
- int ITEM = 140;
- Node LOC=null;
- Node PTR=null;
- PTR=START;
- while(PTR!=null){
- if(ITEM==(int)PTR.INFOR) {
- LOC = PTR;
- break;
- }else
- PTR=PTR.LINK;
- }
- //Show Result;
- if(LOC ==null)
- System.out.println("Not found.");
- else
- System.out.println("Found.");
- }
- }

```
• public class Algor52SEARCH {  
•     public static void main(String[] args) {  
•         Node START = new Node(); START.INFOR = 10;  
•         Node NodeA = new Node(); NodeA.INFOR = 20;  
•         ...  
•         Node NodeI = new Node(); NodeI.INFOR = 100;  
•         START.LINK = NodeA; NodeA.LINK = NodeB; NodeB.LINK = NodeC;  
•         NodeC.LINK = NodeD; NodeD.LINK = NodeE; NodeE.LINK = NodeF;  
•         NodeF.LINK = NodeG; NodeG.LINK = NodeH; NodeH.LINK = NodeI;  
•     }  
• }
```

ส่วนการกำหนดค่าให้กับโนด

```
• //Algorrithm 5.2 SEARCH  
•     int ITEM = 140;  
•     Node LOC=null;  
•     Node PTR=null;  
•     PTR=START;  
•     while(PTR!=null){  
•         if(ITEM==(int)PTR.INFOR) {  
•             LOC = PTR;  
•             break;  
•         }else  
•             PTR=PTR.LINK;  
•     }  
•     //Show Result;  
•     if(LOC ==null)  
•         System.out.println("Not found.");  
•     else  
•         System.out.println("Found.");  
•     }  
• }
```

อัลกอริทึมหลัก

กิจกรรมนักศึกษา

- รันโปรแกรมทดสอบ
- เขียนโปรแกรมเพิ่มเติมตามโจทย์ที่กำหนด

การค้นหาลิงค์ลิสต์ที่มีการจัดเรียง

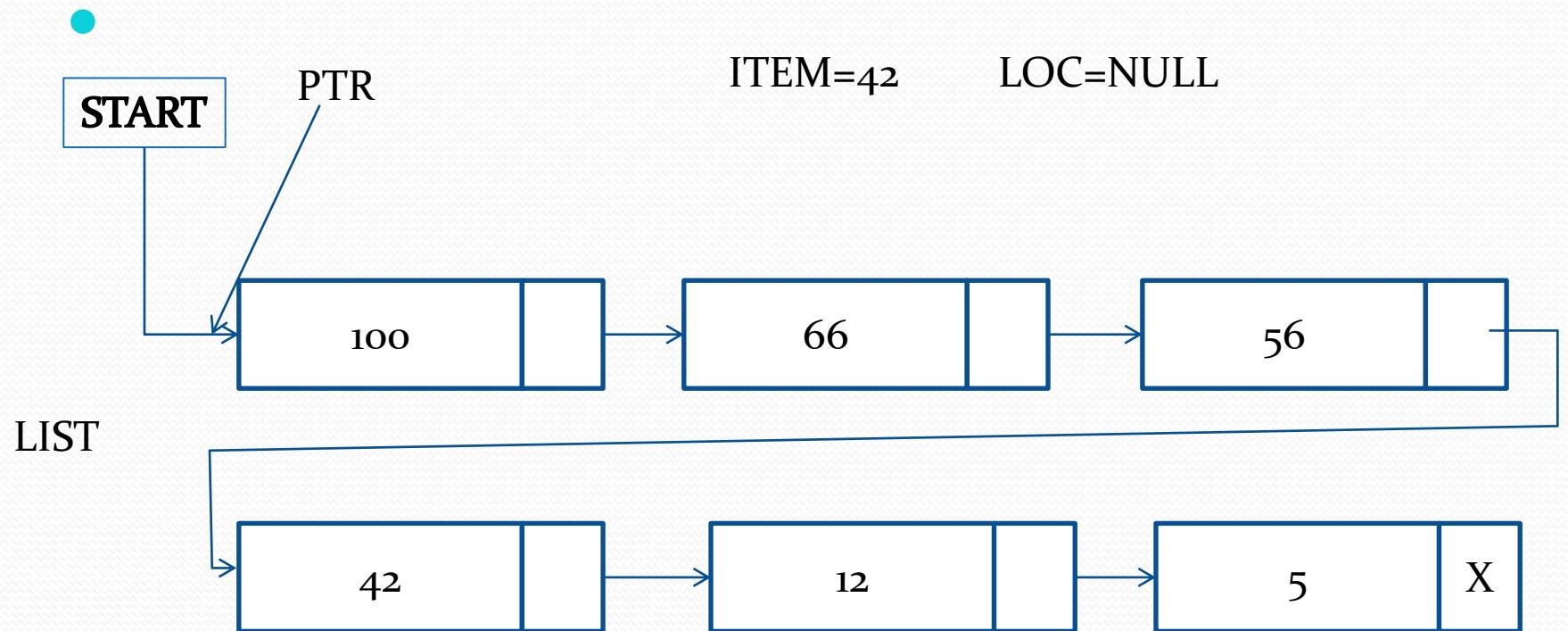
- การค้นหาใช้หลักการท่องไป และเปรียบเทียบเช่นเดียวกันกับแบบไม่มีการจัดเรียง
- เมื่อพบกิรายงานตำแหน่งที่พบร แล้วจบการค้นหา
- แต่เมื่อยังไม่พบจะต้องค้นไป จนกระทั่ง พบรัวที่มากกว่า หรือน้อยกว่า แล้วแต่ว่าการจัดเรียงนั้นเรียงข้อมูลแบบใด
- และหากยังไม่พบให้ค้นไปจนกระทั่งตัวสุดท้าย และรายงานว่าไม่พบข้อมูล

แนวทางอัลกอริทึม

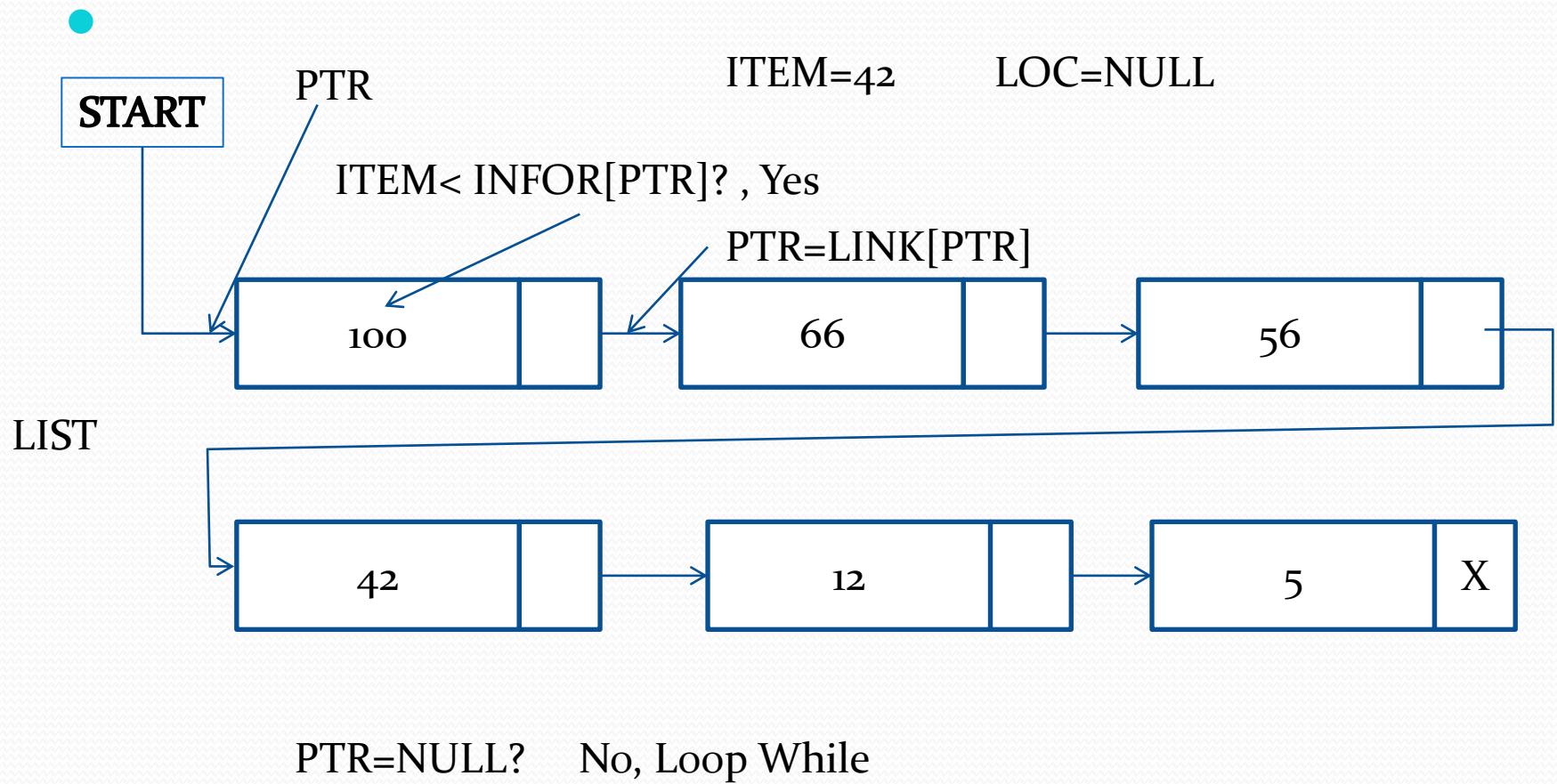
- กำหนด $\text{PTR}=\text{START}$
- วนทำงานในขณะที่ $\text{PTR} \neq \text{NULL}$
 - ถ้า $\text{ITEM} < \text{INFOR}[\text{PTR}]$ เลื่อนตัวชี้ต่อไป $\text{PTR} = \text{LINK}[\text{PTR}]$
 - ถ้า $\text{ITEM} = \text{INFOR}[\text{PTR}]$ กำหนด $\text{LOC} = \text{PTR}$ และจบการค้น
 - แต่ถ้ายังไม่ใช่ ให้กำหนด $\text{LOC} = \text{NULL}$ และจบการค้น เพราะว่า ตัวที่ค้นเกินค่าที่กำหนดไว้แล้ว (น้อยกว่าหรือมากกว่าค่าที่ต้องการค้นแล้ว)
- หากออกจากลูป และยังไม่เข้ากรณีใด ๆ ที่ผ่านมา ให้กำหนด $\text{LOC} = \text{NULL}$ แสดงว่าการค้นหาค่า ITEM นั้นยังน้อยกว่าหรือมากกว่าค่าใน LIST

- Algorithm SEARCH Item in Sorted Link Lists
- Input : LIST, PTR, START, LOC, ITEM
- Output : LOC
- PTR=START
- While PTR!=NULL Do
 - IF ITEM < INFOR[PTR] Then
 - PTR=LINK[PTR]
 - Else
 - IF ITEM=INFOR[PTR] Then
 - LOC = PTR
 - Return LOC
 - Else
 - LOC=NULL
 - Return LOC
 - End IF
 - LOC=NULL
- End While
- Return LOC

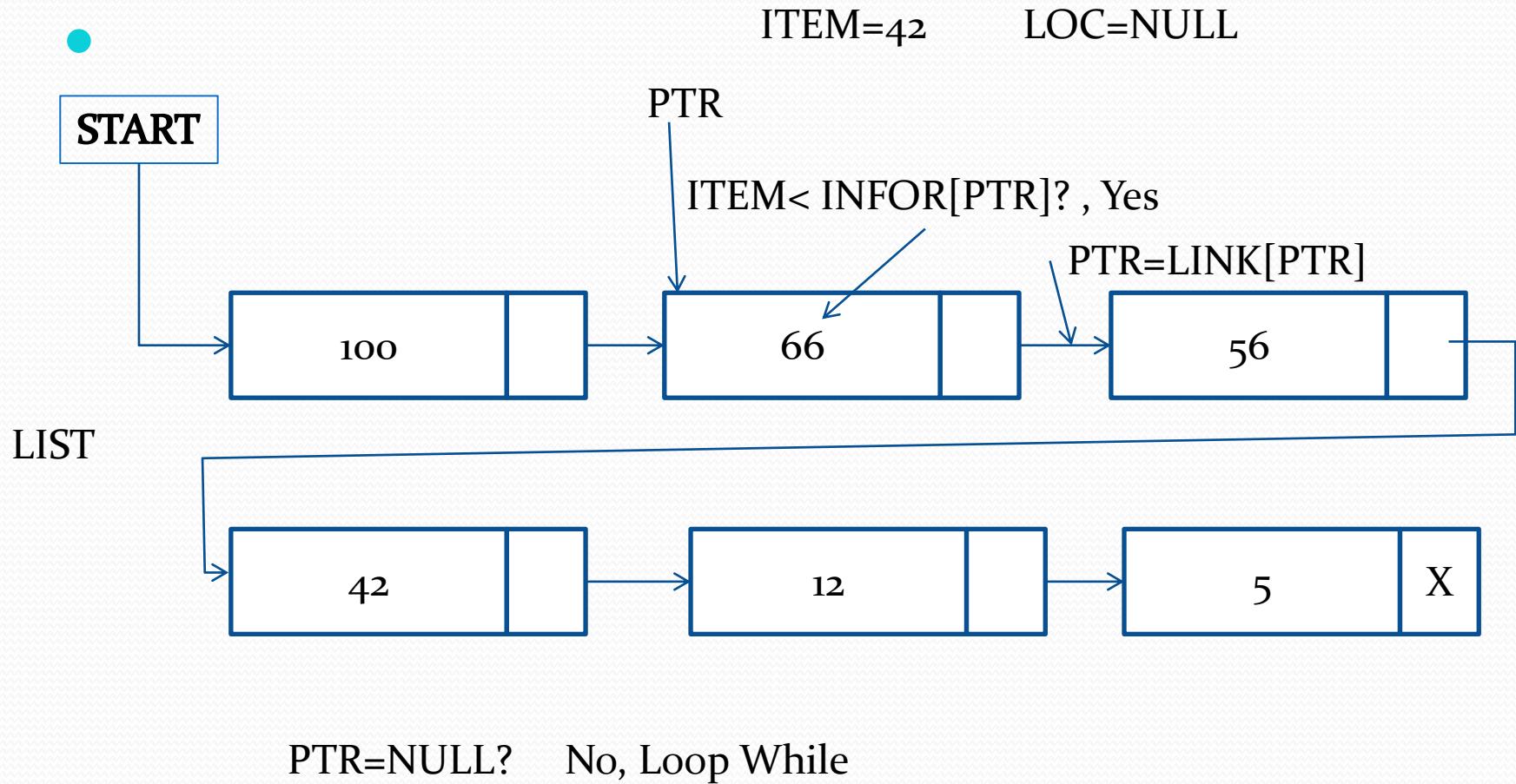
แนวทางการค้นตามอัลกอริทึม



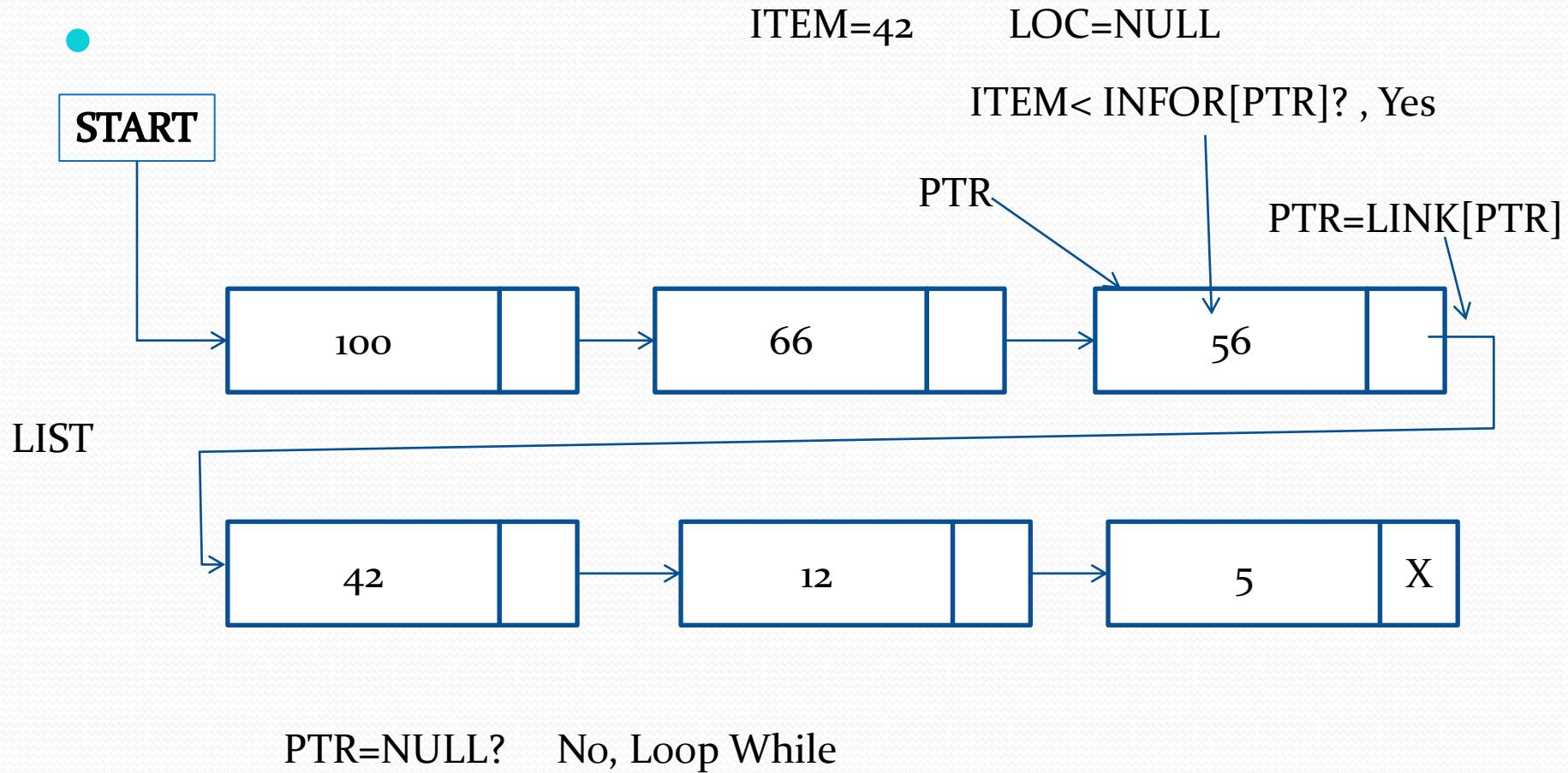
แนวทางการค้นตามอัลกอริทึม



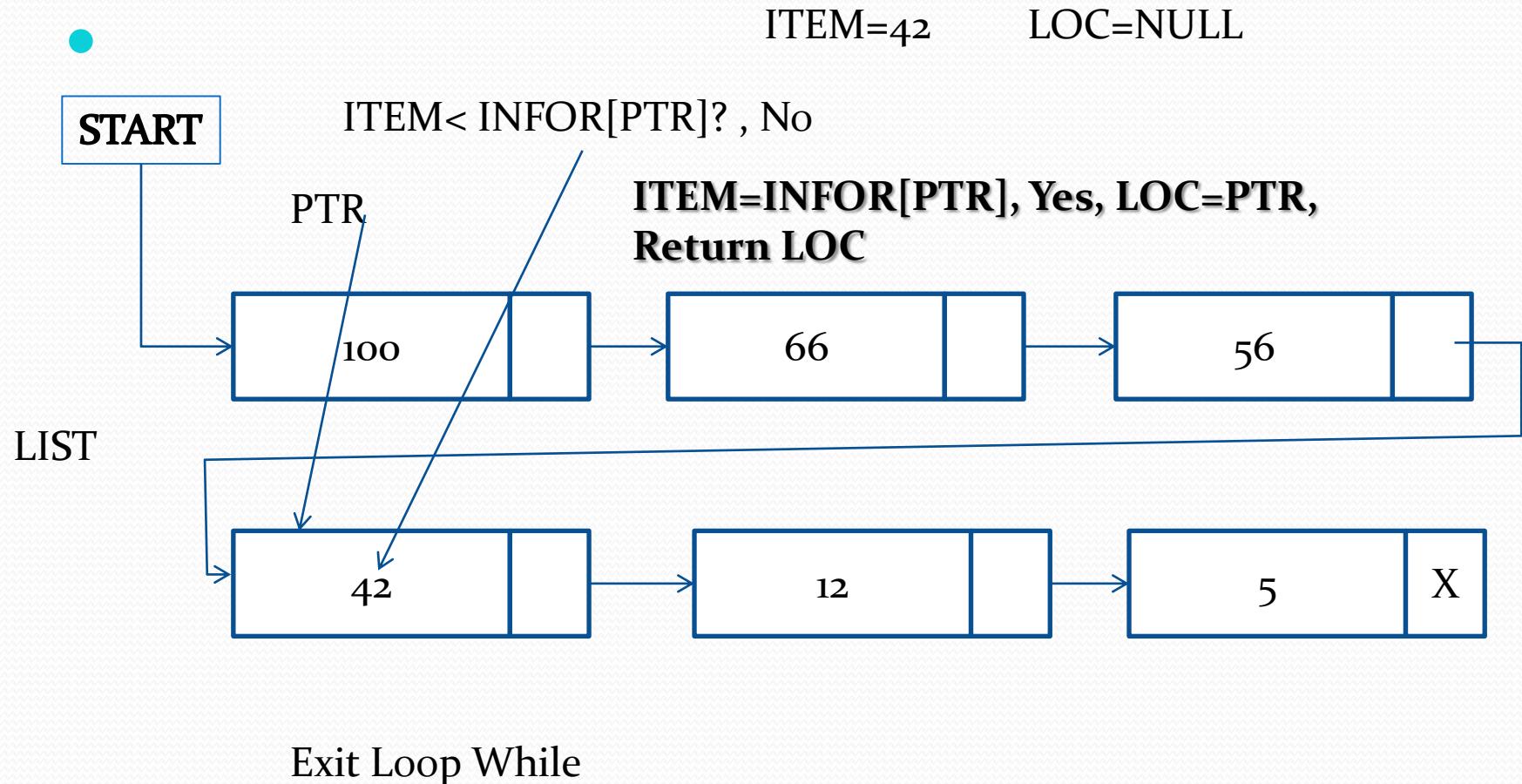
แนวทางการค้นตามอัลกอริทึม



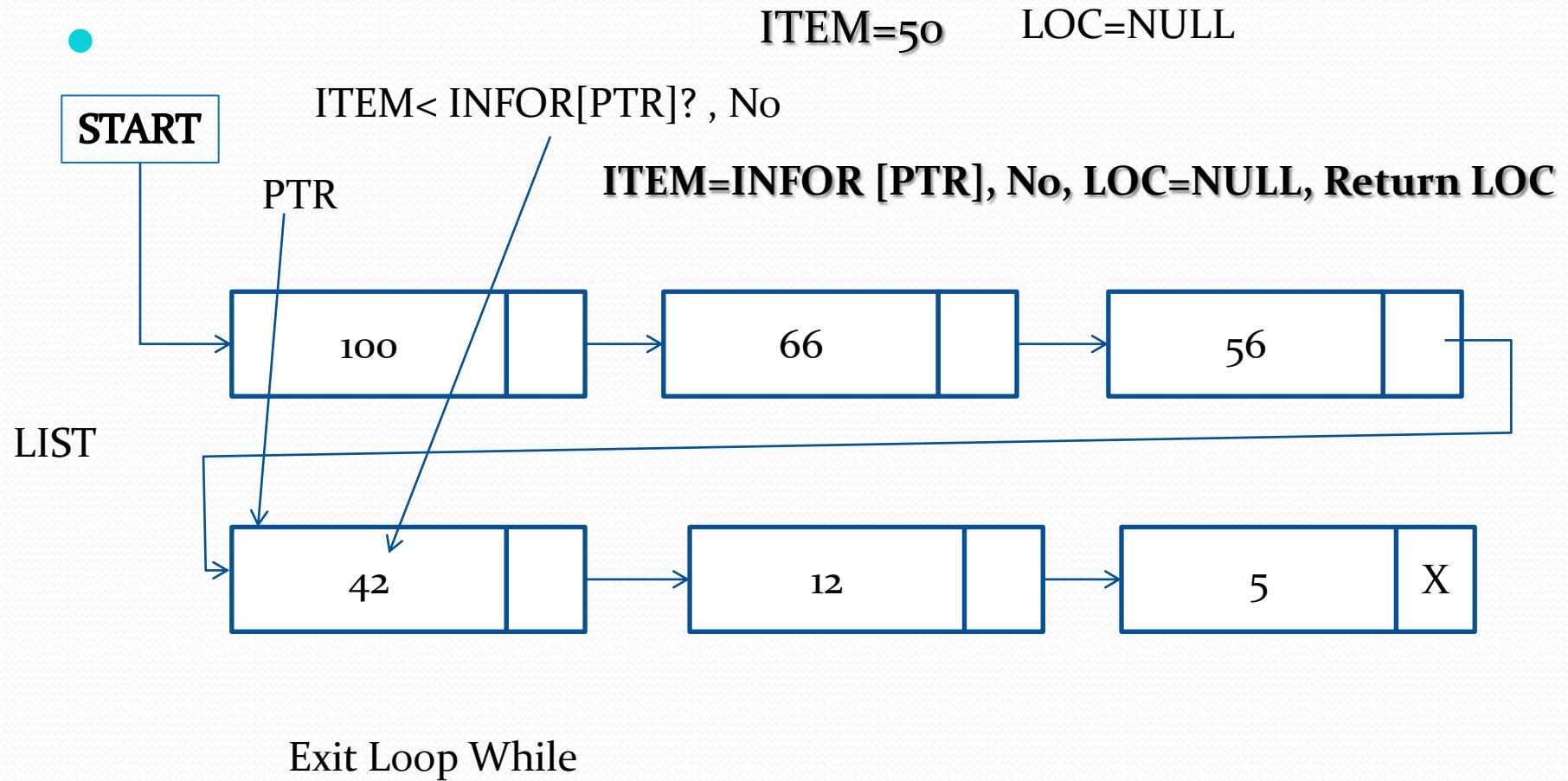
แนวทางการค้นตามอัลกอริทึม



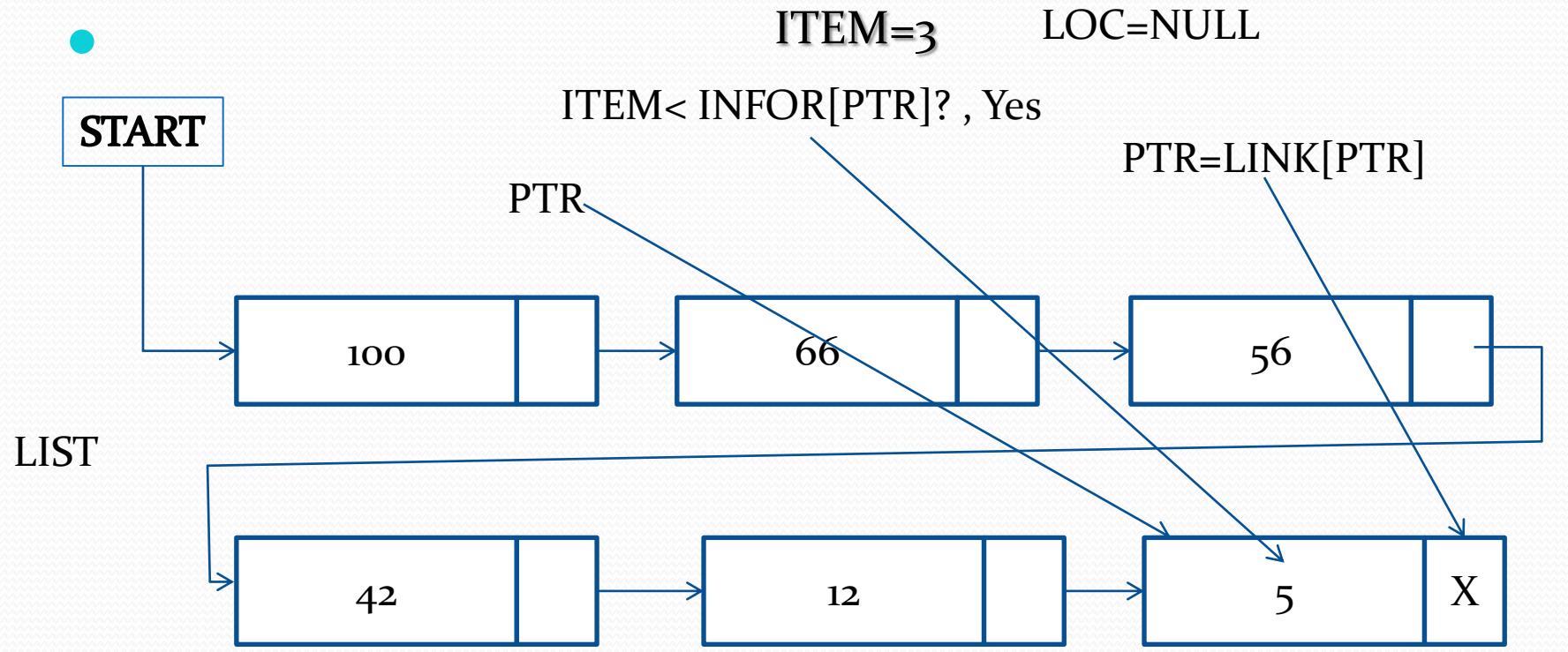
แนวทางการค้นตามอัลกอริทึม (พบ)



แนวทางการค้นตามอัลกอริทึม (ไม่พบ)



แนวทางการค้นตามอัลกอริทึม(ไม่พบ)



PTR=NULL? Yes, Exit Loop While, LOC=NULL, Return LOC

อิมพลีเมนต์

```
public class Algor53SRCHSL {  
    public static void main(String[] args) {  
        SEARCH(40);  
    }//end main  
  
    //Method of Algorithm 5.2  
    public static void SEARCH(int item) {  
        Node START = new Node(); START.INFOR = 100;  
        Node NodeA = new Node(); NodeA.INFOR = 90;  
        Node NodeB = new Node(); NodeB.INFOR = 80;  
        Node NodeC = new Node(); NodeC.INFOR = 70;  
        Node NodeD = new Node(); NodeD.INFOR = 60;  
        Node NodeE = new Node(); NodeE.INFOR = 50;  
        Node NodeF = new Node(); NodeF.INFOR = 40;  
        Node NodeG = new Node(); NodeG.INFOR = 30;  
        Node NodeH = new Node(); NodeH.INFOR = 20;  
        Node NodeI = new Node(); NodeI.INFOR = 10;  
  
        START.LINK = NodeA; NodeA.LINK = NodeB; NodeB.LINK = NodeC;  
        NodeC.LINK = NodeD; NodeD.LINK = NodeE; NodeE.LINK = NodeF;  
        NodeF.LINK = NodeG; NodeG.LINK = NodeH; NodeH.LINK = NodeI;  
  
        //Algorithm 5.2 SEARCH  
        int ITEM = item;  
        Node LOC=null;  
        Node PTR=null;  
        PTR=START;  
        while(PTR!=null){  
            if(ITEM<(int)PTR.INFOR) {  
                PTR=PTR.LINK;  
            }else  
                if(ITEM==(int)PTR.INFOR){  
                    LOC = PTR;  
                    break;  
                }else{  
                    LOC=null;  
                    break;  
                }  
        }//end while  
        //Show Result;  
        if(LOC ==null)  
    }
```

- public static void SEARCH(int item) {
 - Node START = new Node(); START.INFOR = 100;
 - Node NodeA = new Node(); NodeA.INFOR = 90;
 - Node NodeB = new Node(); NodeB.INFOR = 80;
 - Node NodeC = new Node(); NodeC.INFOR = 70;
 - Node NodeD = new Node(); NodeD.INFOR = 60;
 - Node NodeE = new Node(); NodeE.INFOR = 50;
 - Node NodeF = new Node(); NodeF.INFOR = 40;
 - Node NodeG = new Node(); NodeG.INFOR = 30;
 - Node NodeH = new Node(); NodeH.INFOR = 20;
 - Node NodeI = new Node(); NodeI.INFOR = 10;
- START.LINK = NodeA; NodeA.LINK = NodeB; NodeB.LINK = NodeC;
 - NodeC.LINK = NodeD; NodeD.LINK = NodeE; NodeE.LINK = NodeF;
 - NodeF.LINK = NodeG; NodeG.LINK = NodeH; NodeH.LINK = NodeI;

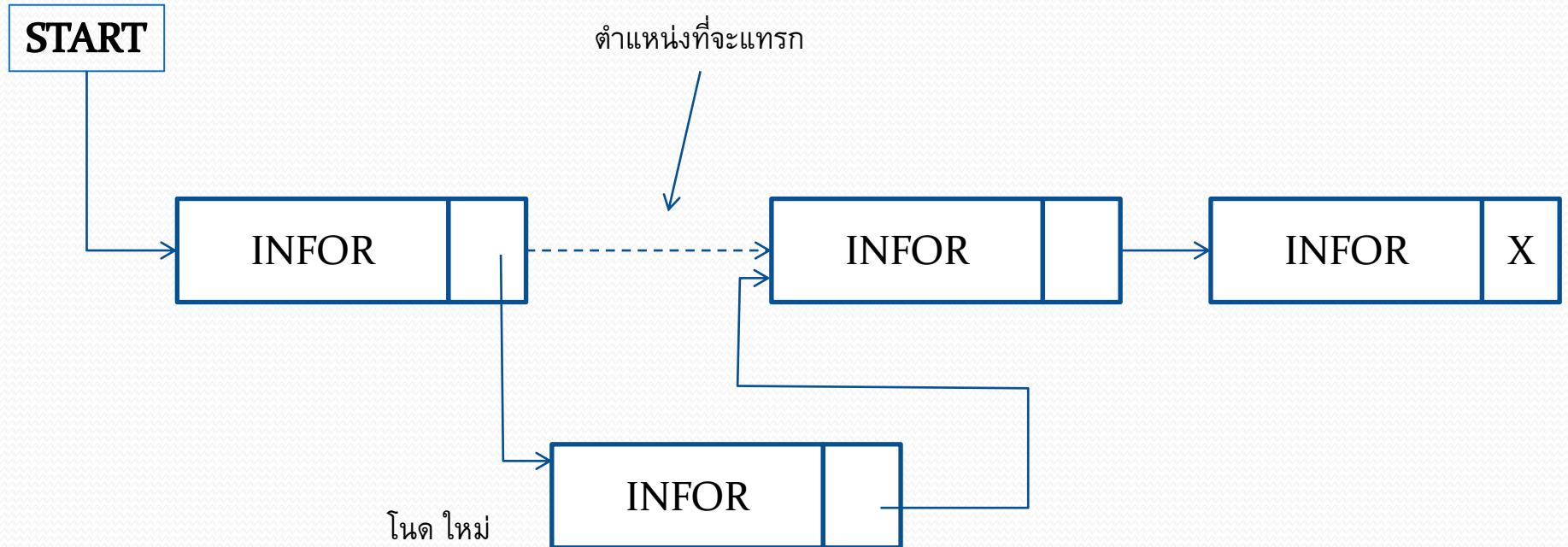
```
•     int ITEM = item;  
•     Node LOC=null;  
•     Node PTR=null;  
•     PTR=START;  
•     while(PTR!=null){  
•         if(ITEM<(int)PTR.INFOR) {  
•             PTR=PTR.LINK;  
•         }else  
•             if(ITEM==(int)PTR.INFOR){  
•                 LOC = PTR;  
•                 break;  
•             }else{  
•                 LOC=null;  
•                 break;  
•             }  
•         }//end while  
•         //Show Result;  
•         if(LOC ==null)  
•             System.out.println("Not found.");  
•         else  
•             System.out.println("Found.");  
•     }  
• }
```

กิจกรรมนักศึกษา

- รันโปรแกรมทดสอบ
- เขียนโปรแกรมเพิ่มเติมตามโจทย์กำหนด

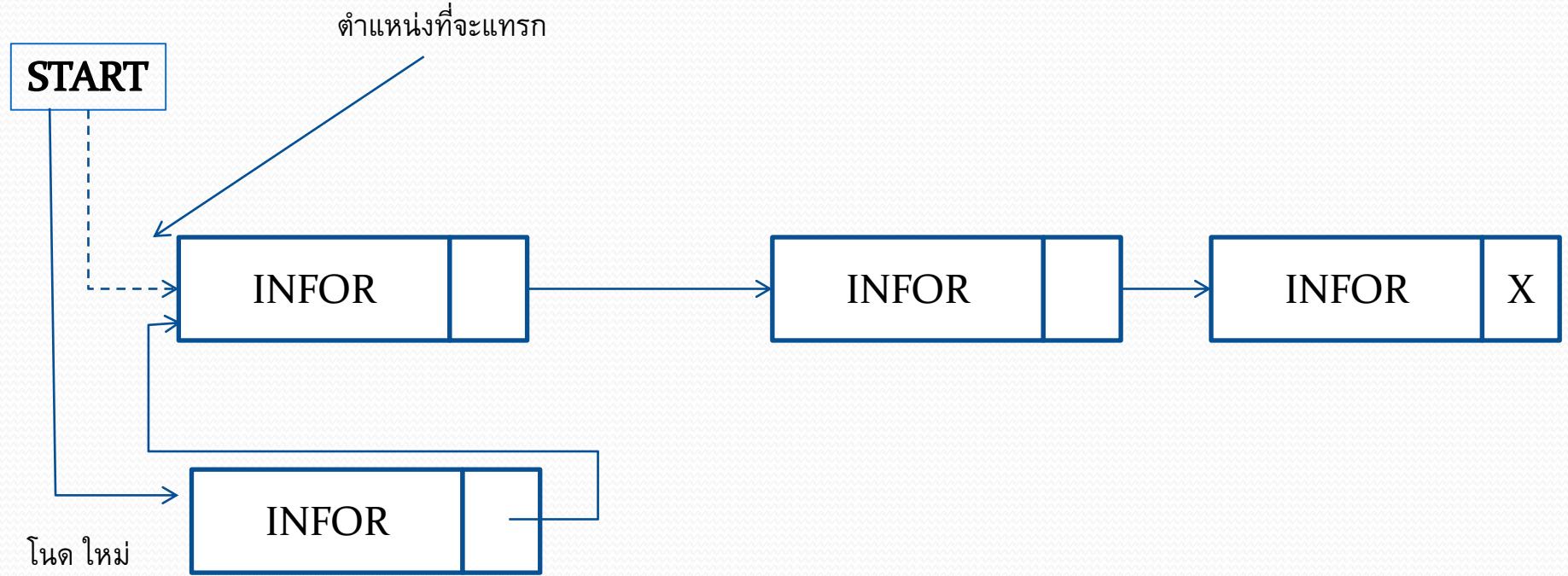
การแทรกข้อมูลลงในลิงค์ลิสต์ (Insertion into A Link Lists)

- การแทรกข้อมูลในลิสต์สามารถดำเนินการได้อย่างรวดเร็วโดย
 - เปลี่ยนพอยเตอร์ที่เคยซึ่งปะจงโนดถัดไป ให้มาซึ่งโนดใหม่ และเอาพอยเตอร์ของโนดใหม่ไปซึ่งโนดนั้นซึ่งอยู่

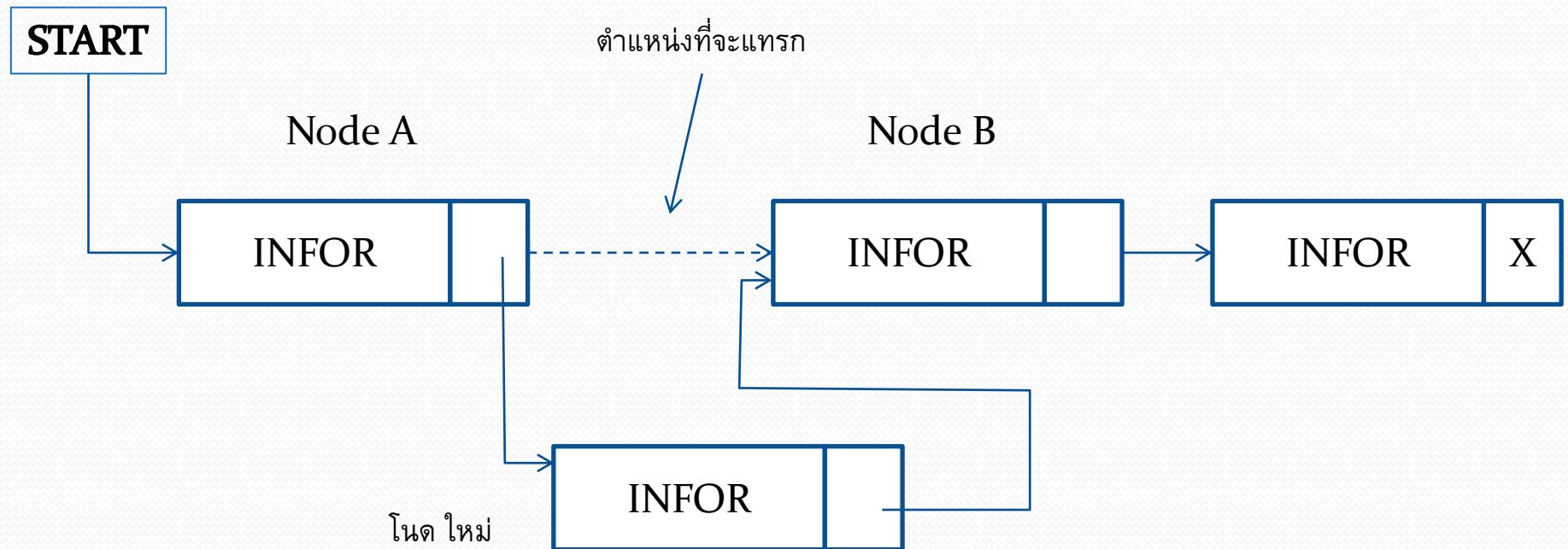


วิธีการแทรกข้อมูล

- แทรกที่ต้นลิสต์
- แทรกในตำแหน่งที่ต้องการ

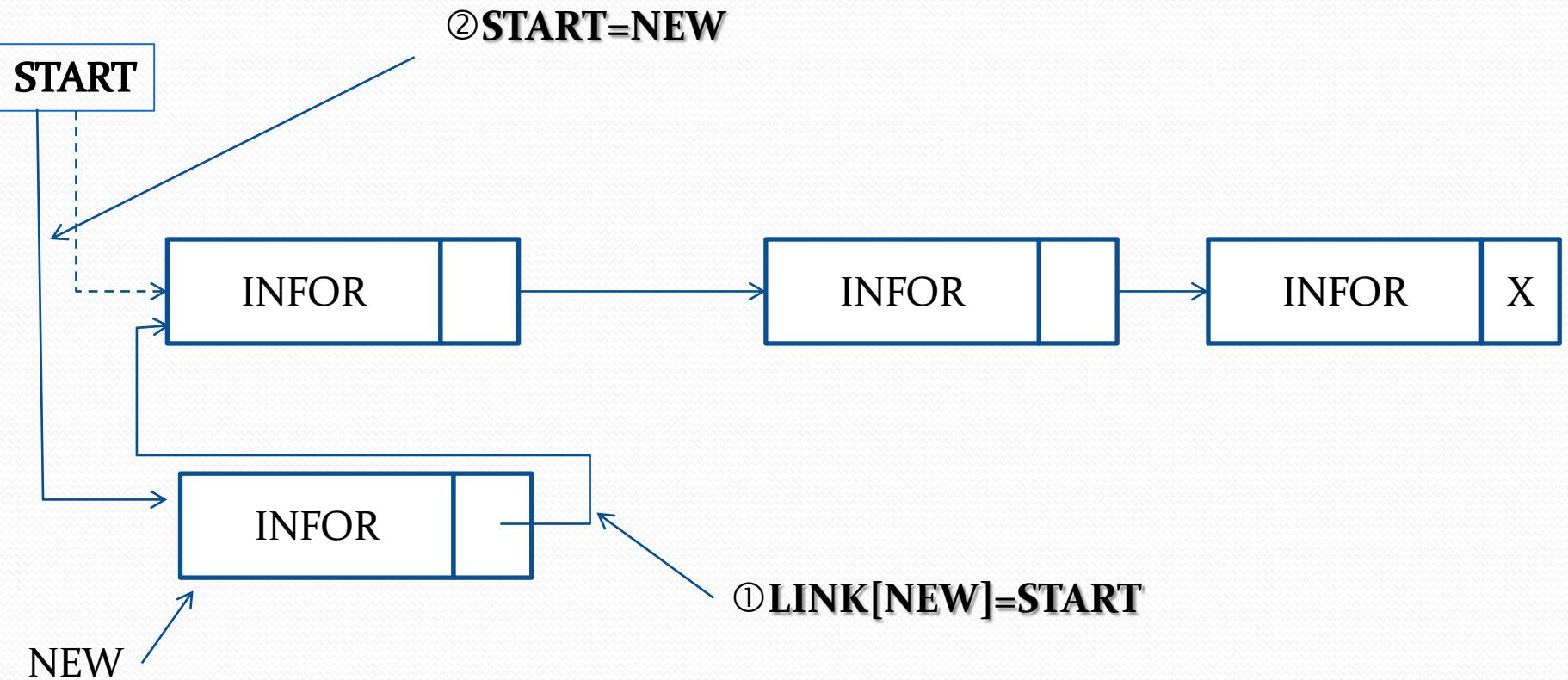


แทรกยังตำแหน่งที่ต้องการ



แนวทางอัลกอริทึมการแทรกต้นลิสต์

- NEW คือโหนดใหม่
- กำหนด $\text{INFOR}[\text{NEW}] = \text{ITEM}$
- กำหนด $\text{LINK}[\text{NEW}] = \text{START}$
- กำหนด $\text{START} = \text{NEW}$



- Algorithm Insert Item at first Node of Link Lists
- Input : LIST, START, NEW, ITEM
- Output : ITEM is inserted into LIST at the first node.
- INFOR[NEW]=ITEM
- LINK[NEW]=START
- START=NEW
- Exit

อิมพลีเมนต์

```
• public class Algor54INSFIRST {  
•     static Node START = new Node();  
  
•     public static void main(String[] args) {  
•         START.INFOR = 10;  
•         System.out.println("Data Before INSERT:");  
•         showDatainList();  
•         INSTFIRST(20);  
•         INSTFIRST(30);  
•         INSTFIRST(40);  
•         System.out.println("Data After INSERT:");  
•         showDatainList();  
•     }  
  
•     public static void INSTFIRST(int item){  
•         //Algorithm 5.4 SEARCH if AVAIL = NULL step. 1  
•         Node NEW=new Node(); //step. 2  
•         int ITEM=item;  
•         NEW.INFOR = item;  
•         NEW.LINK = START;  
•         START=NEW;  
•     }  
  
•     public static void showDatainList(){  
•         Node PTR=null;  
•         PTR=START;  
•         while(PTR!=null){  
•             System.out.println(PTR.INFOR);  
•             PTR=PTR.LINK;  
•         }  
•     }  
• }
```

- public class Algor54INSFIRST {
 - static Node START = new Node();
 - public static void main(String[] args) {
 - START.INFOR = 10;
 - System.out.println("Data Befor INSERT:");
 - showDatainList();
 - INSTFIRST(20);
 - INSTFIRST(30);
 - INSTFIRST(40);
 - System.out.println("Data After INSERT:");
 - showDatainList();
 - }

- public static void INSTFIRST(int item){
- //Algorithm 5.4 SEARCH if AVAIL = NULL step.
- 1
- Node NEW=new Node(); //step. 2
- int ITEM=item;
- NEW.INFOR = item;
- NEW.LINK = START;
- START=NEW;
- }
-

- public static void showDatainList(){
 - Node PTR=null;
 - PTR=START;
 - while(PTR!=null){
 - System.out.println(PTR.INFOR);
 - PTR=PTR.LINK;
 - }

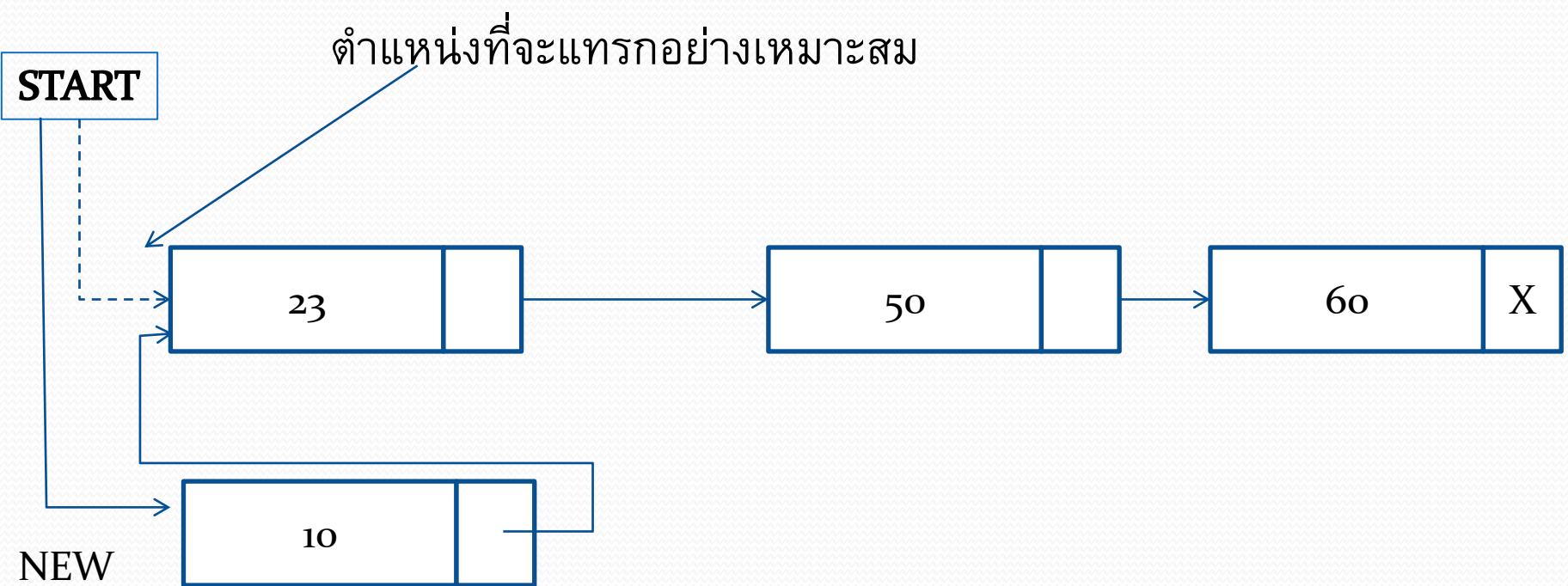
กิจกรรมนักศึกษา

- รันโปรแกรมทดสอบตามอัลกอริทึม
- สร้างลิสต์ของคน外องตามที่กำหนด

แทรกข้อมูลลงในลิสต์ตามตำแหน่งที่ต้องการ

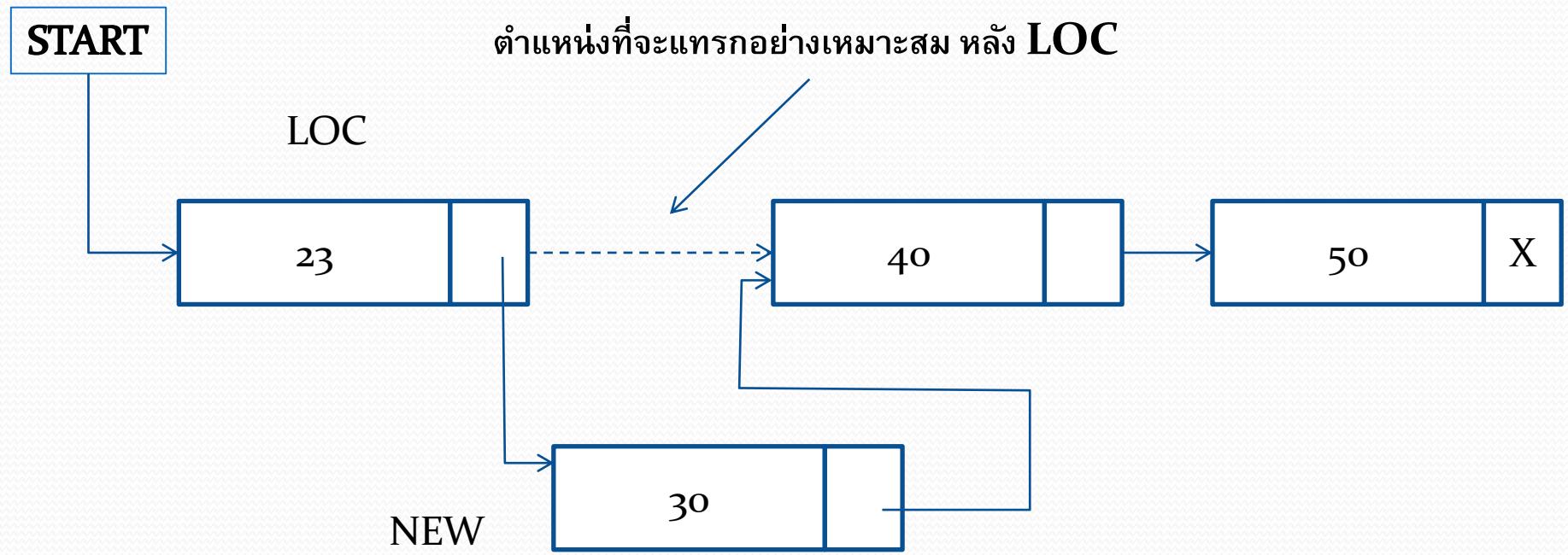
- แนวทางการแทรกคือ ต้องกำหนดโนดที่ต้องการแทรก ก่อน กำหนดให้เป็น LOC ซึ่งตำแหน่งของโนดที่จะแทรกข้อมูลเข้าไปต่อท้าย
- ตรวจสอบว่า $LOC=NULL$ หรือไม่ หากเป็น $NULL$
 - แสดงว่า ให้แทรกต้นลิสต์
 - กำหนด $LINK[NEW]=START$
 - $START=NEW$
 - หากไม่เป็น $NULL$
 - กำหนด $LINK[NEW]=LINK[LOC]$
 - $LINK[LOC]=NEW$

เมื่อตัวแหน่งเหมำะสมเป็นต้นลิสต์



LOC=NULL, LINK[NEW]=START, START=NEW

แทรกยังตำแหน่งที่ต้องการ



LINK[NEW]=LINK[LOC], LINK[LOC]=NEW

- Algorithm Insert Item at LOC in Link Lists
- Input : LIST, START, NEW, ITEM, LOC
- Output : ITEM is inserted into LIST after LOC node.
- INFOR[NEW]=ITEM
- IF LOC = NULL Then
- LINK[NEW]=START
- START=NEW
- Else
- LINK[NEW] = LINK[LOC]
- LINK[LOC] = NEW
- End IF
- Exit

อิมพลีเมนต์

```

public class Algor55INLOC {
    static Node START = new Node();

    public static void main(String[] args) {
        START.INFOR = 100;
        Node A = new Node(); A.INFOR = 90; START.LINK = A;
        Node B = new Node(); B.INFOR = 80; A.LINK = B;
        Node C = new Node(); C.INFOR = 70; B.LINK = C;
        Node D = new Node(); D.INFOR = 60; C.LINK = D;
        Node E = new Node(); E.INFOR = 50; D.LINK = E;
        System.out.println("Data Before INSERT:");
        showDataInList();
        INSTFIRST(70,65);
        System.out.println("Data After INSERT:");
        showDataInList();
    }

    public static void INSTLOC(int loc, int item){
        //Algorithm 5.5 SEARCH if AVAIL = NULL step. 1

        Node LOC=SEARCH(loc); //ค้นหาข้อมูลที่จะแทรก
        Node NEW=new Node(); //step. 2
        int ITEM=item;
        NEW.INFOR = item;
        if(LOC==null) {
            NEW.LINK = START; //insert first Node
            START=NEW;
        }else
        {
            NEW.LINK = LOC.LINK;
            LOC.LINK = NEW;
        }
    }

    public static Node SEARCH(int ITEM){
        Node LOC=null;
        Node PTR=null;
        PTR=START;
        while(PTR!=null){
            if(ITEM==(int)PTR.INFOR) {
                LOC = PTR;
                break;
            }else
                PTR=PTR.LINK;
        }
        //Show Result;
        if(LOC ==null)
            System.out.println("Not found.");
        else
            System.out.println("Found.");

        return LOC;
    }

    public static void showDataInList(){
        Node PTR=null;
        PTR=START;
        while(PTR!=null){
            System.out.println(PTR.INFOR);
            PTR=PTR.LINK;
        }
    }
}

```

- public class Algor55INLOC {
- static Node START = new Node();
-
- public static void main(String[] args) {
- START.INFOR = 100;
- Node A = new Node(); A.INFOR = 90; START.LINK = A;
- Node B = new Node(); B.INFOR = 80; A.LINK = B;
- Node C = new Node(); C.INFOR = 70; B.LINK = C;
- Node D = new Node(); D.INFOR = 60; C.LINK = D;
- Node E = new Node(); E.INFOR = 50; D.LINK = E;
- System.out.println("Data Before INSERT:");
- showDatainList();
- INSTFIRST(70,65);
- System.out.println("Data After INSERT:");
- showDatainList();
- }
-

- public static void INSTLOC(int loc, int item){
 - //Algorithm 5.5 SEARCH if AVAIL = NULL step. 1
 -
 - Node LOC=SEARCH(loc); //ค้นหาข้อมูลที่จะแทรก
 - Node NEW=new Node(); //step. 2
 - int ITEM=item;
 - NEW.INFOR = item;
 - if(LOC==null) {
 - NEW.LINK = START; //insert first Node
 - START=NEW;
 - }else
 - {
 - NEW.LINK = LOC.LINK;
 - LOC.LINK = NEW;
 - }
 - }

```
• public static Node SEARCH(int ITEM){  
•     Node LOC=null;  
•     Node PTR=null;  
•     PTR=START;  
•     while(PTR!=null){  
•         if(ITEM==(int)PTR.INFOR) {  
•             LOC = PTR;  
•             break;  
•         }else  
•             PTR=PTR.LINK;  
•     }  
•     //Show Result;  
•     if(LOC ==null)  
•         System.out.println("Not found.");  
•     else  
•         System.out.println("Found.");  
•  
•     return LOC;  
• }
```

- public static void showDatainList(){
- Node PTR=null;
- PTR=START;
- while(PTR!=null){
- System.out.println(PTR.INFOR);
- PTR=PTR.LINK;
- }
- }

กิจกรรมนักศึกษา

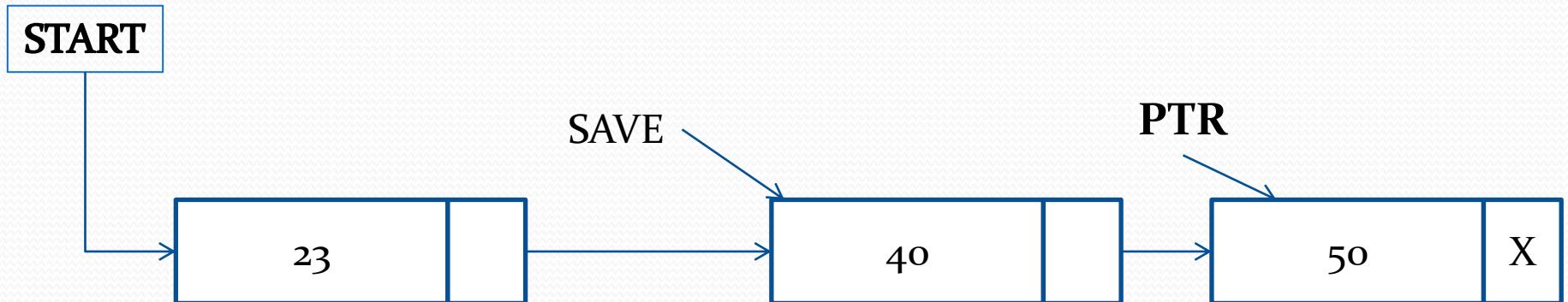
- รันโปรแกรมทดสอบตามอัลกอริทึมตัวอย่าง
- ทดลองเขียนโปรแกรมตามที่กำหนด

การแทรกโนดในลิสต์ที่มีการจัดเรียงไว้แล้ว

- ต้องค้นหาตำแหน่งที่จะแทรกให้พบก่อน LOC
 - โดยการใช้พอยเตอร์สองตัวค้นหาเข้าไปในลิสต์ PTR, SAVE
 - เมื่อค้นเข้าไปจนพบ ตำแหน่งที่ต้องการกำหนด $LOC=SAVE$ เพื่อคืนค่า LOC
- จากนั้นเรียกใช้งานอัลกอริทึมการแทรก ณ ตำแหน่งที่ต้องการให้ทำงาน

ค้นหาตำแหน่งที่ต้องการ

- ถ้า ITEM=45 ตำแหน่งที่เหมาะสมระหว่าง 40 กับ 50



ตำแหน่งที่จะแทรกต่อท้าย **LOC=SAVE**

- Algorithm Find LOC in a sorted Link Lists
- Input : LIST, START, SAVE, ITEM, LOC
- Output : LOC
- IF START=NULL Then
 - LOC=NULL
 - Return LOC
- End IF
- IF ITEM< INFOR[START] Then
 - LOC=NULL
 - Return LOC
- End IF
- SAVE=START, PTR=LINK[START]
- While PTR!=NULL Do
 - IF ITEM< INFOR[PTR] Then
 - LOC = SAVE;
 - Return LOC;
 - End IF
 - SAVE=PTR;
 - PTR=LINK[PTR]
- End While
- LOC=SAVE;
- Return LOC

- Algorithm Find LOC in a sorted Link Lists
- Input : LIST, START, SAVE, ITEM, LOC
- Output : LOC
- IF START=NULL Then
 - LOC=NULL
 - Return LOC
- End IF
- IF ITEM< INFOR[START] Then
 - LOC=NULL
 - Return LOC
- End IF
- SAVE=START, PTR=LINK[START]
- While PTR!=NULL Do
 - IF ITEM< INFOR[PTR] Then
 - LOC = SAVE;
 - Return LOC;
 - End IF
 - SAVE=PTR;
 - PTR=LINK[PTR]
- End While
- LOC=SAVE;
- Return LOC

กรณีไม่มีข้อมูลลิสต์

กรณีข้อมูลที่ค้นน้อยกว่าโอนดแรก

อัลกอริทึมแทรกตำแหน่งที่ต้องการ (ข้อมูลเรียงแล้ว)

- เรียก Find LOC in a sorted Link Lists ให้ทำงาน
- เรียก Insert Item at LOC in Link Lists ให้ทำงาน

อิมพลีเมนต์

```

• package linklistimplementation;
•
• /**
• *
• * @author ComSCIv3400
• */
• public class Algor56TO57FINDA {
•     static Node START = new Node();
•
•     public static void main(String[] args) {
•         START.INFOR = 50;
•         Node A = new Node(); A.INFOR = 60; START.LINK = A;
•         Node B = new Node(); B.INFOR = 70; A.LINK = B;
•         Node C = new Node(); C.INFOR = 80; B.LINK = C;
•         Node D = new Node(); D.INFOR = 90; C.LINK = D;
•         Node E = new Node(); E.INFOR = 100; D.LINK =E;
•         System.out.println("Data Before INSERT:");
•         showDatainList();
•         Node LOC=FINDA(70); //ค้นหาข้อมูลที่จะแทรก      //call FINDA(...)
•         INSLOC(LOC,75);    //Call INSLOC...
•         System.out.println("Data After INSERT:");
•         showDatainList();
•     }
•
•     public static void INSLOC(Node LOC, int item){
•         //Algorithm 5.5 SEARCH if AVAIL = NULL step. 1
•         Node NEW=new Node(); //step. 2
•         int ITEM=item;
•         NEW.INFOR = item;
•         if(LOC==null) {
•             NEW.LINK = START; //insert first Node
•             START=NEW;
•         }else
•         {
•             NEW.LINK = LOC.LINK;
•             LOC.LINK = NEW;
•         }
•     }
•     public static Node FINDA(int ITEM){
•         Node LOC=null;
•         Node PTR=null;
•         Node SAVE=null;
•         PTR=START;
•         if(START==null) { LOC=null; return LOC; }
•         if(ITEM<(int)PTR.INFOR) { LOC=null; return LOC; }
•         SAVE=START; PTR=START.LINK;
•         while(PTR!=null){
•             if(ITEM<(int)PTR.INFOR) {
•                 LOC = SAVE;
•                 return LOC;
•             }
•             SAVE=PTR;
•             PTR=PTR.LINK;
•         }
•         LOC=SAVE;
•         //Show Result;
•         if(LOC ==null)
•             System.out.println("Not found.");
•         else
•             System.out.println("Found.");
•     }

```

- public class Algor56TO57FINDA {
- static Node START = new Node();
-
- public static void main(String[] args) {
- START.INFOR = 50;
- Node A = new Node(); A.INFOR = 60; START.LINK = A;
- Node B = new Node(); B.INFOR = 70; A.LINK = B;
- Node C = new Node(); C.INFOR = 80; B.LINK = C;
- Node D = new Node(); D.INFOR = 90; C.LINK = D;
- Node E = new Node(); E.INFOR = 100; D.LINK = E;
- System.out.println("Data Before INSERT:");
- showDatainList();
- Node LOC=FINDA(70); //ค้นหาข้อมูลที่จะแทรก //call FINDA(...)
- INSLOC(LOC,75); //Call INSLOC(...)
- System.out.println("Data After INSERT:");
- showDatainList();
- }

- public static void INSLOC(Node LOC, int item){
 - //Algorithm 5.5 SEARCH if AVAIL = NULL step. 1
 - Node NEW=new Node(); //step. 2
 - int ITEM=item;
 - NEW.INFOR = item;
 - if(LOC==null) {
 - NEW.LINK = START; //insert first Node
 - START=NEW;
 - }else
 - {
 - NEW.LINK = LOC.LINK;
 - LOC.LINK = NEW;
 - }
 - }

```
• public static Node FINDA(int ITEM){  
•     Node LOC=null;  
•     Node PTR=null;  
•     Node SAVE=null;  
•     PTR=START;  
•     if(START==null) { LOC=null; return LOC; }  
•     if(ITEM<(int)PTR.INFOR) { LOC=null; return LOC; }  
•     SAVE=START; PTR=START.LINK;  
•     while(PTR!=null){  
•         if(ITEM<(int)PTR.INFOR) {  
•             LOC = SAVE;  
•             return LOC;  
•         }  
•         SAVE=PTR;  
•         PTR=PTR.LINK;  
•     }  
•     LOC=SAVE;  
•     //Show Result;  
•     if(LOC ==null)  
•         System.out.println("Not found.");  
•     else  
•         System.out.println("Found.");  
•  
•     return LOC;  
• }
```

กิจกรรมนักศึกษา

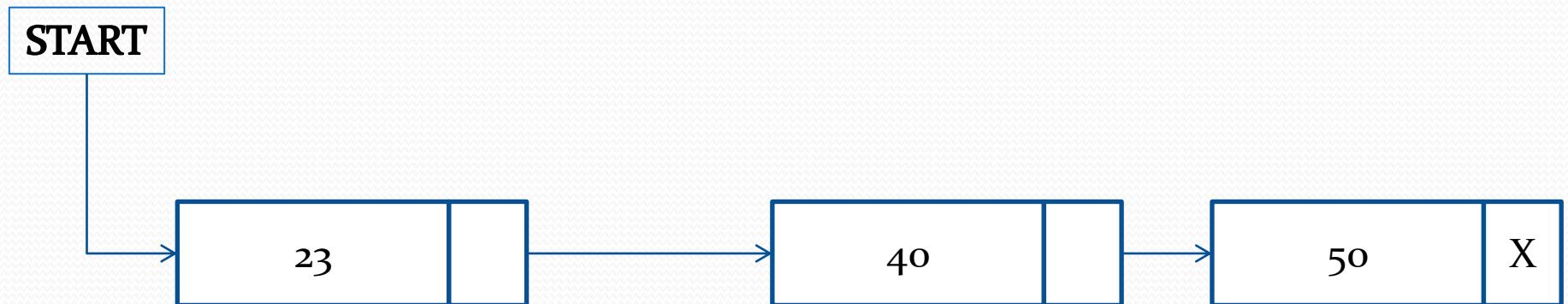
- รันโปรแกรมทดสอบตามอัลกอริทึม
- ทดลองเขียนโปรแกรมตามที่โจทย์กำหนด

การลบข้อมูลออกจากลิงค์ลิสต์ ()

- การลบข้อมูล nonduplicate ออกจากลิงค์ลิสต์ สามารถดำเนินการได้โดยເອົາພອຍເຕັມຂອງໂນດທີ່ອຢູ່ກ່ອນໜ້າທີ່ຈະລົບ ຂຶ້ขໍາມໄປຢັ້ງໂນດຕ່ອໄປທີ່ໂນດຈະລົບຂໍ້ອູ່ ເພີ່ງທ່ານີ້ ໂນດກີຈະຖຸກລົບໄປ
- ທັງຈາກນັ້ນຫາກຕ້ອງມີກາຣີຄືນຄ່າໜ່ວຍຄວາມຈຳກີສາມາດດໍາເນີນກາຣໄດ້ຕາມແຕ່ກາຣນຳໄປອິມພລືເມນຕ໌ ເຊັ່ນ ໃນກາໝາຊີ້ອາຈຕ້ອງຄືນໜ່ວຍຄວາມຈຳ ແຕ່ໃນກາໝາຈາວາອາຈໄມຕ້ອງຄືນເພຣະມີຮະບບຈັດກາຣໜ່ວຍຄວາມຈຳອູ່ແລ້ວ ເປັນຕັ້ນ

แสดงแนวคิดการลบ

- ก่อนลบ



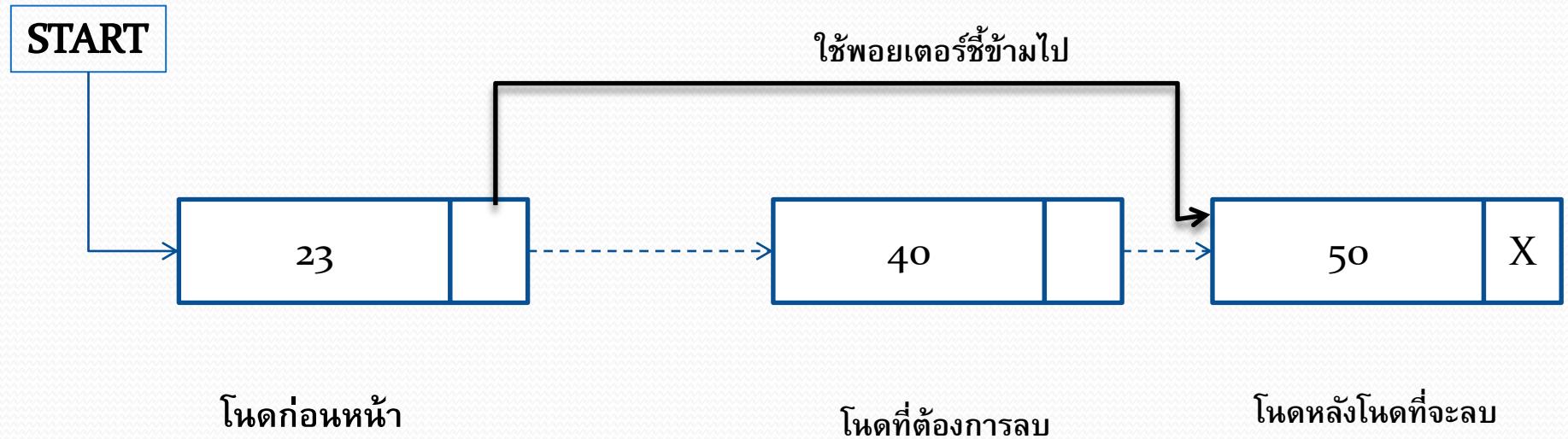
โนดก่อนหน้า

โนดที่ต้องการลบ

โนดหลังโนดที่จะลบ

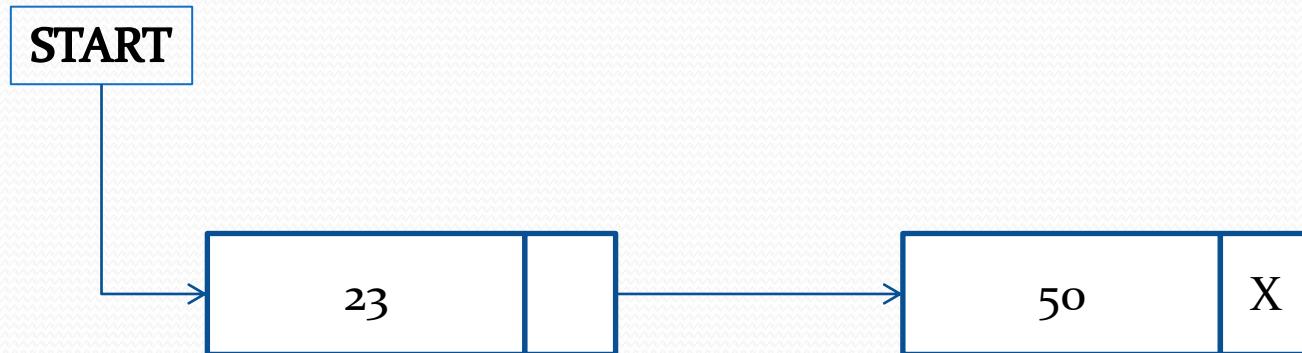
แสดงแนวคิดการลบ

- ขนะลบ

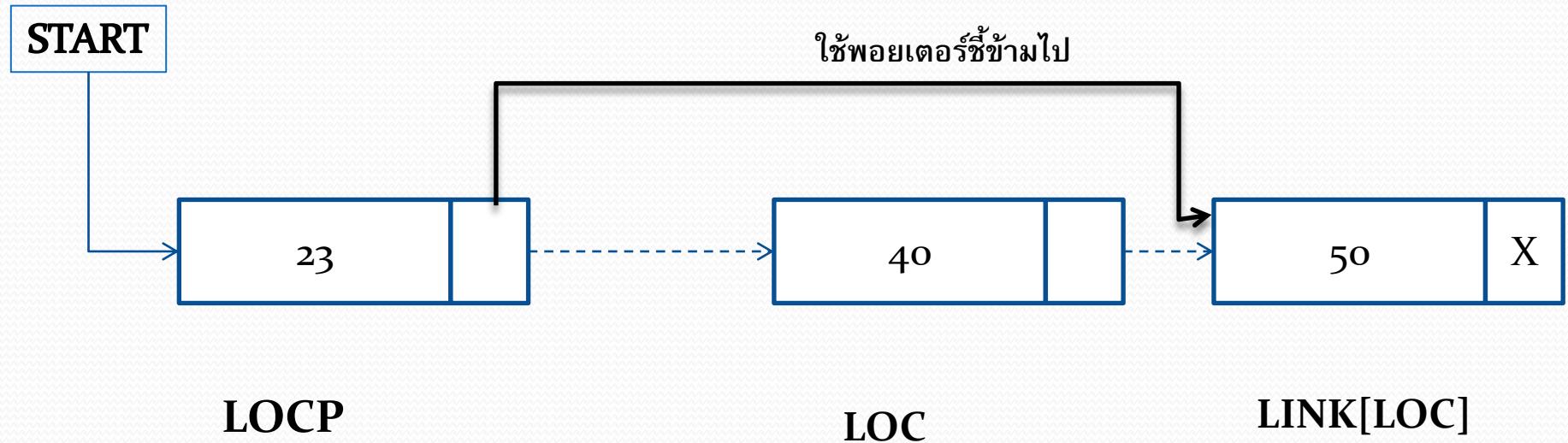


แสดงแนวคิดการลบ

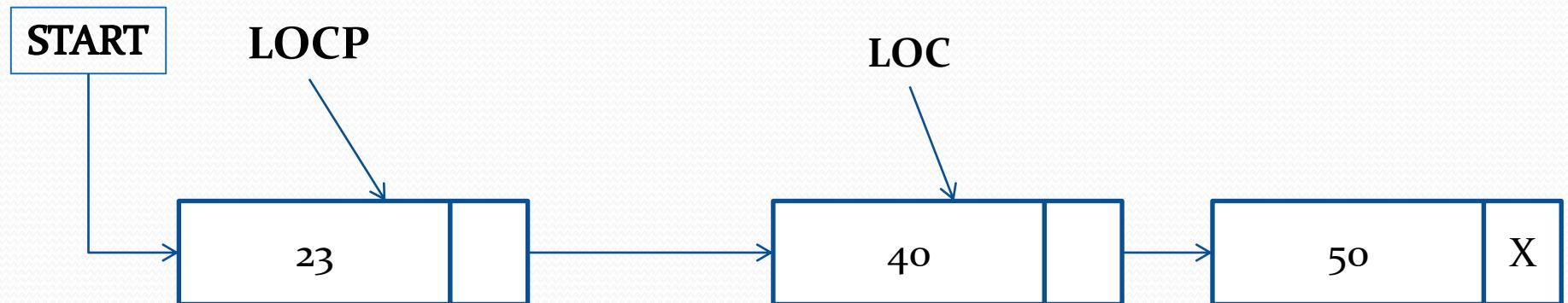
- หลังลบ



ตัวแปรที่เกี่ยวข้องกับการลบ



ก่อนการลบจะต้องระบุ LOCP, LOC

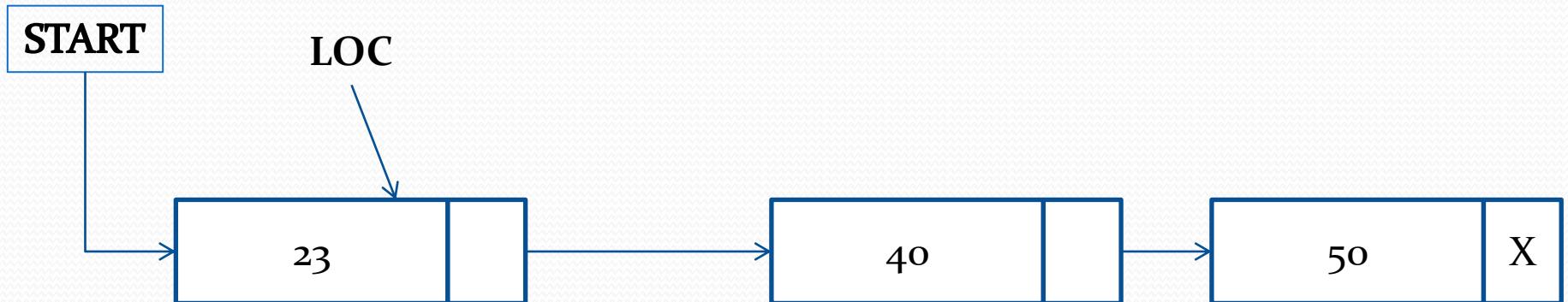


การลบเกิดขึ้น 2 กรณี

- ลบโนดแรก
 - $\text{LOC}_{\text{P}} = \text{NULL}$, $\text{LOC} = \text{START}$
- ลบโนดอื่นๆ
 - LOC_{P} , LOC ไม่เท่ากับ NULL

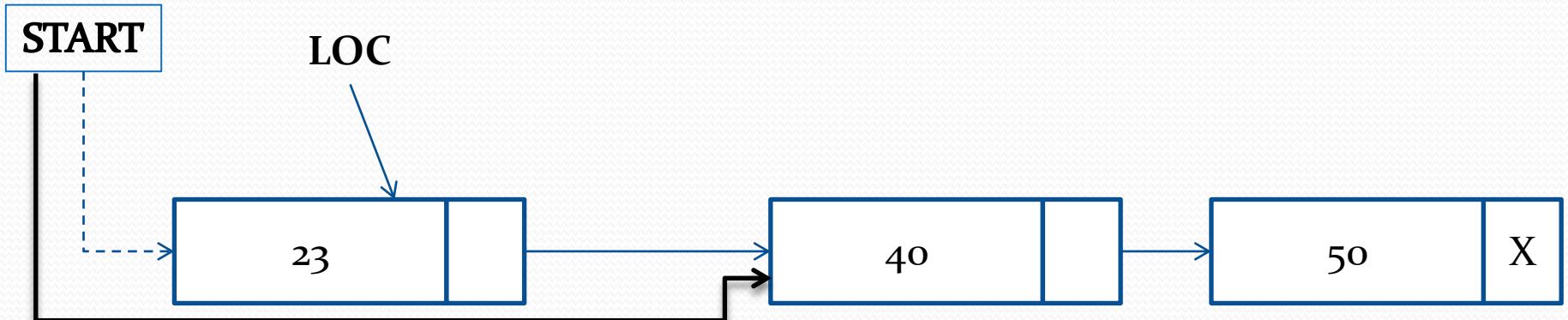
LOCP=NULL, LOC=START (ก่อนลบ)

LOCP=NULL



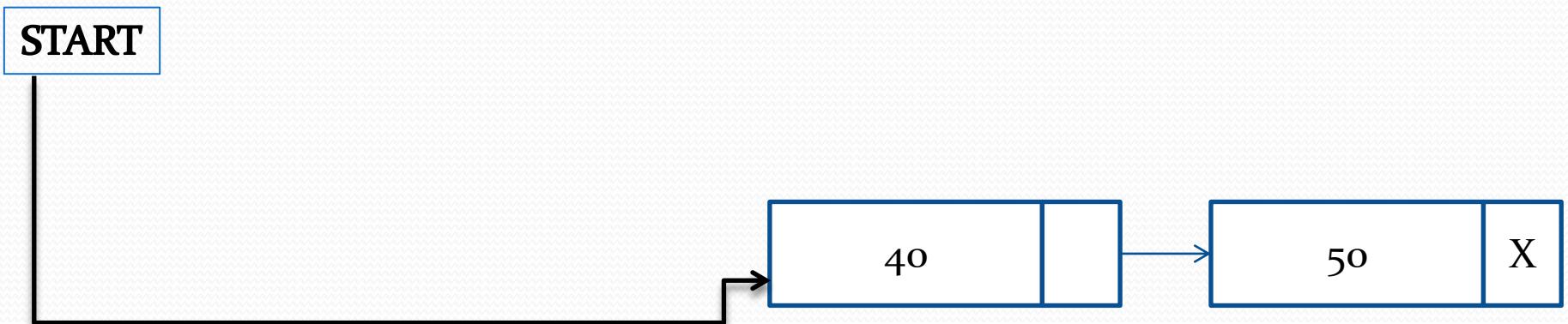
LOCP=NULL, LOC=START (ឧណະលប)

LOCP=NULL

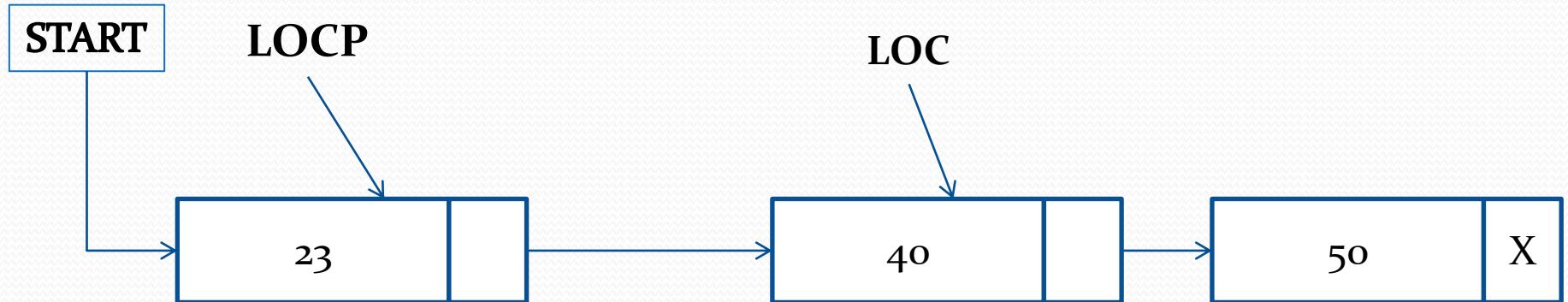


START=LINK[START]

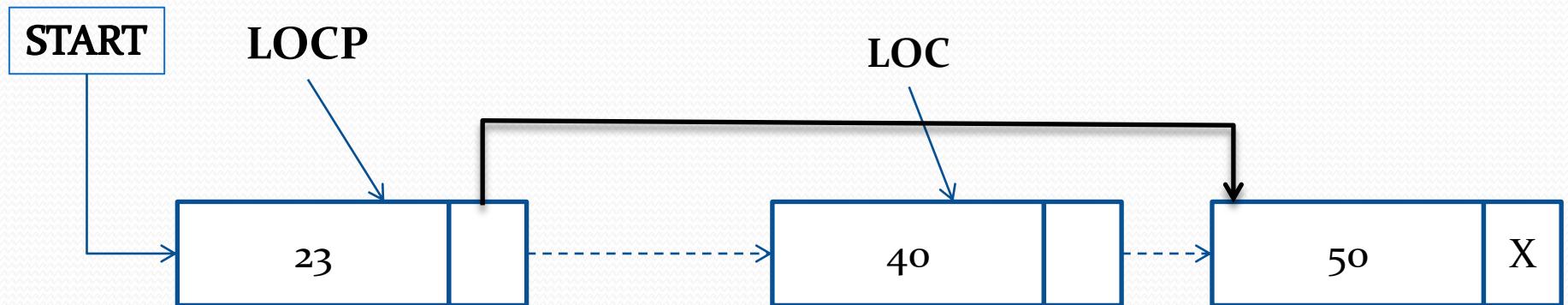
LOC=START (หลังลบ)



LOCP, LOC ไม่เป็น NULL(ก่อนลบ)

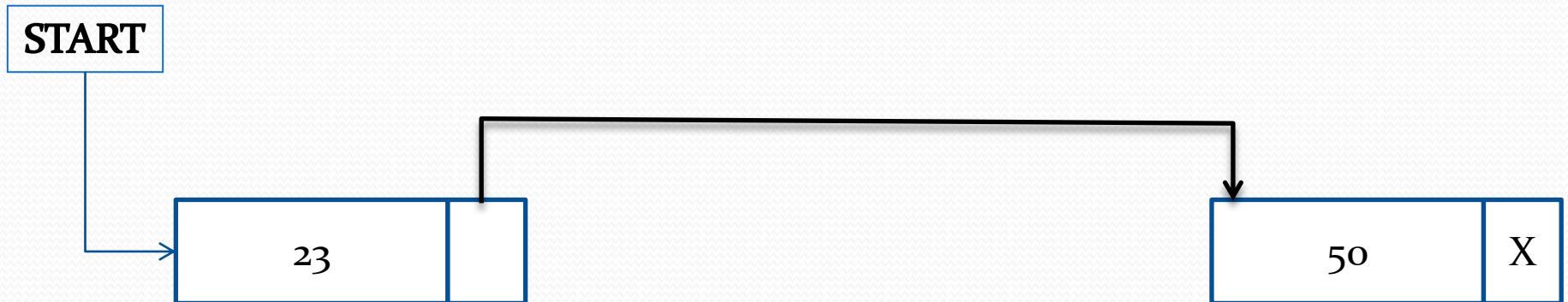


LOCP, LOC ไม่เป็น NULL(ขณะลับ)



$$\text{LINK[LOCP]} = \text{LINK[LOC]}$$

LOCP, LOC ไม่เป็น NULL(หลังลบ)



- Algorithm DEL Item in a Link Lists
- Input : LIST, INFOR, LINK, START, LOCP, LOC
- Output : Node LOC is deleted from LIST.
- IF LOC = NULL Then
- START= LINK[START]
- Else
- LINK[LOCP] = LINK[LOC]
- End IF
- Exit

อิมพลีเมนต์

```
• package linklistimplementation;

• /**
• *
• * @author ComSCIv3400
• */
• public class Algor58DEL {
•     static Node START = new Node();
•
•     public static void main(String[] args) {
•         START.INFOR = 50;
•         Node A = new Node(); A.INFOR = 60; START.LINK = A;
•         Node B = new Node(); B.INFOR = 70; A.LINK = B;
•         Node C = new Node(); C.INFOR = 80; B.LINK = C;
•         Node D = new Node(); D.INFOR = 90; C.LINK = D;
•         Node E = new Node(); E.INFOR = 100; D.LINK = E;
•         Node LOCP=B;
•         Node LOC=C;
•         System.out.println("Data Before INSERT:");
•         showDatainList();
•         DEL(LOCP,LOC,START);      //Call DEL()...
•         System.out.println("Data After INSERT:");
•         showDatainList();
•     }
•
•     public static void DEL(Node LOCP, Node LOC, Node START){
•         //Algorithm 5.8 Delete
•         if(LOCP==null){
•             START = START.LINK; //insert first Node
•         }else{
•             LOCP.LINK = LOC.LINK;
•         }
•     }
•
•     public static void showDatainList(){
•         Node PTR=null;
•         PTR=START;
•         while(PTR!=null){
•             System.out.println(PTR.INFOR);
•             PTR=PTR.LINK;
•         }
•     }
• }
```

```
• public class Algor58DEL {  
•     static Node START = new Node();  
•  
•     public static void main(String[] args) {  
•         START.INFOR = 50;  
•         Node A = new Node(); A.INFOR = 60; START.LINK = A;  
•         Node B = new Node(); B.INFOR = 70; A.LINK = B;  
•         Node C = new Node(); C.INFOR = 80; B.LINK = C;  
•         Node D = new Node(); D.INFOR = 90; C.LINK = D;  
•         Node E = new Node(); E.INFOR = 100; D.LINK = E;  
•         Node LOCP=B;  
•         Node LOC=C;  
•         System.out.println("Data Before INSERT:");  
•         showDatainList();  
•         DEL(LOCP,LOC, START);      //Call DEL(...)  
•         System.out.println("Data After INSERT:");  
•         showDatainList();  
•     }  
• }
```

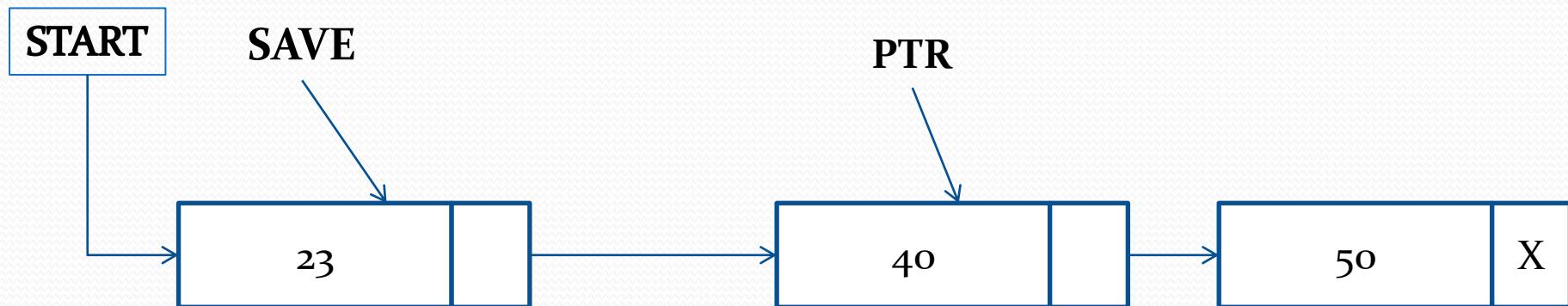
- public static void DEL(Node LOCP, Node LOC, Node START){
 - //Algorithm 5.8 Delete
 - if(LOCP==null) {
 - START = START.LINK; //insert first Node
 - }else
 - {
 - LOCP.LINK = LOC.LINK;
 - }
 - }
-
- public static void showDatainList(){
 - Node PTR=null;
 - PTR=START;
 - while(PTR!=null){
 - System.out.println(PTR.INFOR);
 - PTR=PTR.LINK;
 - }
- }

กิจกรรมนักศึกษา

- รันโปรแกรมทดสอบ
- ทดลองเขียนโปรแกรมลบโนดตามที่ต้องการ

การทำตำแหน่ง LOCP, LOC

- ใช้พอยเตอร์ SAVE เพื่อท่องและคืนค่าเป็น LOCP
- ใช้พอยเตอร์ PTR เพื่อท่องและคืนค่าเป็น LOC



- Algorithm FINB Find Item in a Link Lists
- Input : LIST, INFOR, LINK, START, LOCP, LOC, SAVE, PTR, ITEM
- Output : LOCP, LOC.
- IF START = NULL Then
- LOCP= LOC=NULL
- Return LOCP, LOC
- End IF
- IF INFOR[START]=ITEM Then
- LOCP=NULL,
- LOC = START
- Return LOCP, LOC
- End IF
- SAVE=START, PTR=LINK[START]
- While PTR!=NULL DO
- IF INFOR[PTR]=ITEM Then
- LOCP=SAVE, LOC=PTR
- Return LOCP, LOC
- End IF
- SAVE=PTR, PTR=LINK[PTR]
- End While
- End IF
- Exit

- Algorithm Full Deletion Item in a Link Lists
- Input : LIST, INFOR, LINK, START, LOCP, LOC, ITEM
- Output : Item at LOC is deleted from LIST.
- Call FINB(LIST, INFOR, LINK, START, LOCP, LOC, SAVE, PTR, ITEM)
- IF LOC=NULL Then
 - Write ITEM is not exist in LIST
- Return
- End IF
- IF LOCP=NULL Then
 - START=LINK[START]
- Else
 - LINK[LOCP]=LINK[LOC]
- End IF
- Exit

อิมพลีเมนต์

```
package linklistimplementation;

/*
 *
 * @author ComSCIv3400
 */
public class Algor59to10TFINDB {
    static Node START = new Node();
    static Node LOCP=null;
    static Node LOC=null;
    public static void main(String[] args) {
        START.INFOR = 50;
        Node A = new Node(); A.INFOR = 60; START.LINK = A;
        Node B = new Node(); B.INFOR = 70; A.LINK = B;
        Node C = new Node(); C.INFOR = 80; B.LINK = C;
        Node D = new Node(); D.INFOR = 90; C.LINK = D;
        Node E = new Node(); E.INFOR = 100; D.LINK = E;
        System.out.println("Data Before INSERT:");
        showDatainList();
        DEL(70);      //Call INSLOC...
        System.out.println("Data After INSERT:");
        showDatainList();
    }

    public static void DEL(int ITEM){
        //Algorithm 5.8 Delete
        FINDB(ITEM);
        if(LOCP==null) {
            START = START.LINK; //insert first Node
        }else
        {
            LOCP.LINK = LOC.LINK;
        }
    }

    public static boolean FINDB(int ITEM){

        Node PTR=null;
        Node SAVE=null;
        PTR=START;
        if(START==null) { LOC=null; LOCP=null; return true; }
        if((int)START.INFOR==ITEM) { LOC=START; LOCP=null; return true; }

        SAVE=START; PTR=START.LINK;
        while(PTR!=null){
            if((int)PTR.INFOR==ITEM) {
                LOC = PTR; LOCP=SAVE;
                return true;
            }
            SAVE=PTR;
            PTR=PTR.LINK;
        }
        LOC=null;
        //Show Result;
        if(LOC ==null){
            System.out.println("Not found.");
            return false;
        }
        else
        {
        }
    }
}
```

```
• public class Algor59to10TFINDB {  
•     static Node START = new Node();  
•     static Node LOCP=null;  
•     static Node LOC=null;  
•     public static void main(String[] args) {  
•         START.INFOR = 50;  
•         Node A = new Node(); A.INFOR = 60; START.LINK = A;  
•         Node B = new Node(); B.INFOR = 70; A.LINK = B;  
•         Node C = new Node(); C.INFOR = 80; B.LINK = C;  
•         Node D = new Node(); D.INFOR = 90; C.LINK = D;  
•         Node E = new Node(); E.INFOR = 100; D.LINK =E;  
•         System.out.println("Data Before INSERT:");  
•         showDatainList();  
•         DEL(70);      //Call INSLOC()...  
•         System.out.println("Data After INSERT:");  
•         showDatainList();  
•     }  
• }
```

- public static void DEL(int ITEM){
- //Algorithm 5.8 Delete
- FINDB(ITEM);
- if(LOC.P==null) {
- START = START.LINK; //insert first Node
- }else
- {
- LOC.P.LINK = LOC.LINK;
- }
- }

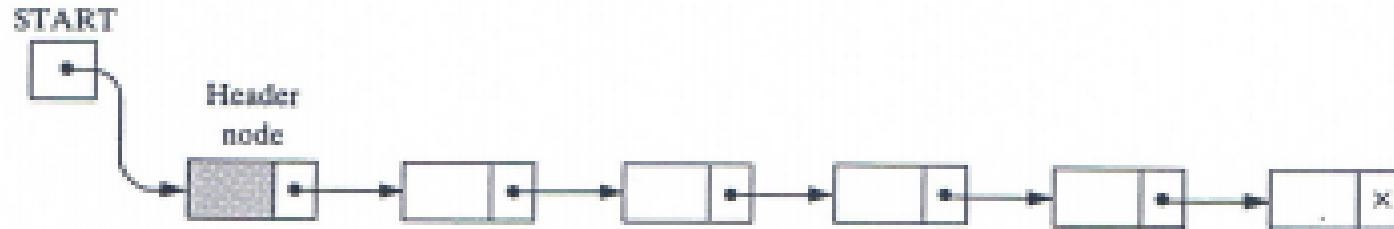
```
• public static boolean FINDB(int ITEM){  
•     Node PTR=null;  
•     Node SAVE=null;  
•     PTR=START;  
•     if(START==null) { LOC=null; LOCP=null; return true;}  
•     if((int)START.INFOR==ITEM) { LOC=START; LOCP=null; return true;}  
•  
•     SAVE=START; PTR=START.LINK;  
•     while(PTR!=null){  
•         if((int)PTR.INFOR==ITEM) {  
•             LOC = PTR; LOCP=SAVE;  
•             return true;  
•         }  
•         SAVE=PTR;  
•         PTR=PTR.LINK;  
•     }  
•     LOC=null;  
•     //Show Result;  
•     if(LOC ==null){  
•         System.out.println("Not found.");  
•         return false;  
•     }  
•     else  
•     {  
•         System.out.println("Found.");  
•         return true;  
•     }  
• }
```

กิจกรรมนักศึกษา

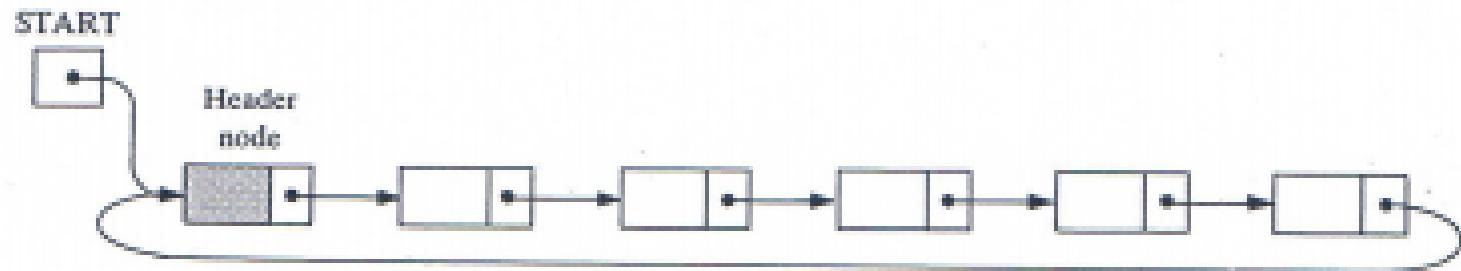
- รันโปรแกรมทดสอบการทำงานของอัลกอริทึม
- ทดลองเขียนโปรแกรมตามที่โจทย์กำหนด

ลิงค์ลิสต์แบบมีโนดนำ และวงกลม (Header Link Lists and Circular Link Lists)

- Grounded Link Lists



- Circular Link Lists



- Algorithm Traversing Circular Link Lists
- Input : LIST, PTR, START
- Output : All nodes in LIST are traversed
- PTR=START
- While PTR!=START Do
 - Process INFOR[PTR]
 - PTR=LINK[PTR]
- End While
- Return

อิมพลีเมนต์

```
• public class Algor511TraversingList {  
•     public static void main(String[] args) {  
•         Node START=new Node();  
•         Node NodeA = new Node(); NodeA.INFOR = 20;  
•         Node NodeB = new Node(); NodeB.INFOR = 30;  
•         Node NodeC = new Node(); NodeC.INFOR = 40;  
•         START.LINK = NodeA; NodeA.LINK = NodeB; NodeB.LINK = NodeC; NodeC.LINK  
• = START;  
•         Node PTR=null;  
•         PTR=START.LINK;  
•         while(PTR!=START){  
•             System.out.println(PTR.INFOR); //Apply PROCESS to INFOR[PTR]  
•             PTR=PTR.LINK;  
•         }  
•     }  
• }
```

กิจกรรมนักศึกษา

- รันโปรแกรมทดสอบ
- เขียนโปรแกรมสร้างโนดนำพิเศษ และสร้างการท่องตามโจทย์กำหนด

การค้นหาลิสต์ที่มีโนดนำ

- Algorithm SRCHL Searching in Circular Link Lists
- Input : LIST, PTR, START, INFOR, LINK, LOC, ITEM
- Output : LOC
- PTR=LINK[START]
- While INFOR[PTR] != ITEM and PTR!=START Do
 - PTR=LINK[PTR]
- End While
- IF INFOR[PTR]=ITEM Then
 - LOC=PTR
- Else
 - LOC=NULL
- End IF
- Return

ค้นหาตำแหน่งที่ต้องการลบ

- Algorithm FINDBHL Searching Node in Circular Link Lists
- Input : LIST, PTR, START, INFOR, LINK, LOC, LOCP, ITEM
- Output : LOC, LOCP
- SAVE=START;
- PTR=LINK[START]
- While INFOR[PTR]!=ITEM and PTR!=START Do
 - SAVE=PTR;
 - PTR=PTR.LINK;
- End While
- IF INFOR[PTR]=ITEM Then
 - LOC=PTR;
 - LOCP=SAVE;
- Else
 - LOC=NULL
 - LOCP=SAVE;
- End IF
- Return

การลบข้อมูล

- Algorithm FINDBHL Searching Node in Circular Link Lists
- Input : LIST, PTR, START, INFOR, LINK,LOC,ITEM
- Output : Node LOC is deleted form LIST.
- FINDBHL(ITEM);
- IF LOC=NULL Then
- Print ITEM is not in List
- Else
- LINK[LOCP] = LINK[LOC]
- End IF
- Return

อิมพลีเมนต์

```
• package linklistimplementation;

• /**
• *
• * @author ComSCIv3400
• */
• public class Algor51234 {
•     static Node START=new Node();
•     static Node LOC=null,LOCP=null, SAVE=null, PTR=null;
•
•     public static void main(String[] args) {
•
•         Node NodeA = new Node(); NodeA.INFOR = 20;
•         Node NodeB = new Node(); NodeB.INFOR = 30;
•         Node NodeC = new Node(); NodeC.INFOR = 40;
•         START.LINK = NodeA; NodeA.LINK = NodeB; NodeB.LINK = NodeC; NodeC.LINK = START;
•         showData(); //แสดงข้อมูล
•
•         if(SRCHL(40)!=null) { System.out.println("พบข้อมูลในลิสต์"); } //ตัวอย่างการค้นหา
•         DELLOCHL(30); //ลบข้อมูล
•         showData(); //แสดงข้อมูล
•     }
•     //end main
•     public static void showData(){ //แสดงข้อมูล
•         PTR=START.LINK;
•         while(PTR!=START){
•             System.out.println(PTR.INFOR); //Apply PROCESS to INFOR[PTR]
•             PTR=PTR.LINK;
•         }
•     }
•     public static Node SRCHL(int ITEM){ //Algorithm 5.12
•         PTR=START.LINK;
•         while((int)PTR.INFOR!=ITEM){
•             PTR=PTR.LINK;
•         }
•         if((int)PTR.INFOR==ITEM)
•             LOC=PTR;
•         return LOC;
•     }
•     public static void FINDBHL(int ITEM) { //Algorithm 5.13
•         SAVE=START;
•         PTR=START.LINK;
•         while(((int)PTR.INFOR!=ITEM)&& (PTR!=START)){
•             SAVE=PTR;
•             PTR=PTR.LINK;
•         }
•     }
• }
```

```
• public class Algor51234 {  
•     static Node START=new Node();  
•     static Node LOC=null,LOCP=null, SAVE=null, PTR=null;  
•  
•     public static void main(String[] args) {  
•  
•         Node NodeA = new Node(); NodeA.INFOR = 20;  
•         Node NodeB = new Node(); NodeB.INFOR = 30;  
•         Node NodeC = new Node(); NodeC.INFOR = 40;  
•         START.LINK = NodeA; NodeA.LINK = NodeB; NodeB.LINK = NodeC;  
•         NodeC.LINK = START;  
•         showData(); //แสดงข้อมูล  
•  
•         if(SRCHL(40)!=null) { System.out.println("พบข้อมูลในลิสต์"); } //ตัวอย่างการค้นหา  
•         DELLOCNL(30); //ลบข้อมูล  
•         showData(); //แสดงข้อมูล  
•     }  
•     //end main
```

- public static void showData(){ //แสดงข้อมูล
- PTR=START.LINK;
- while(PTR!=START){
 - System.out.println(PTR.INFOR); //Apply
PROCESS to INFOR[PTR]
 - PTR=PTR.LINK;
- }
- }

- public static Node SRCHL(int ITEM){ //Algorithm 5.12
 - PTR=START.LINK;
 - while((int)PTR.INFOR!=ITEM){
 - PTR=PTR.LINK;
 - }
 - if((int)PTR.INFOR==ITEM)
 - LOC=PTR;
 - return LOC;
 - }

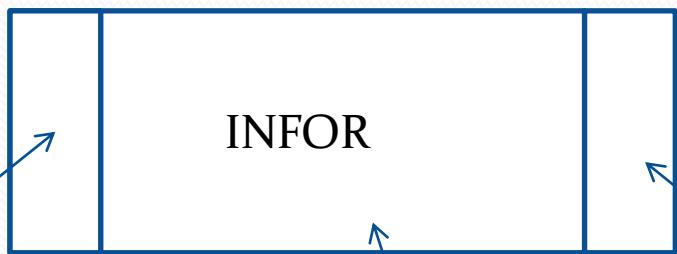
- public static void FINDBHL(int ITEM) { //Algorithm 5.13
- SAVE=START;
- PTR=START.LINK;
- while(((int)PTR.INFOR!=ITEM)&& (PTR!=START)){
- SAVE=PTR;
- PTR=PTR.LINK;
- }
- if((int)PTR.INFOR==ITEM){
- LOC=PTR;
- LOCP=SAVE;
- }else
- {
- LOC=null;
- LOCP=SAVE;
- }
- }

- public static void DELLOCHL(int ITEM){ //Algorithm
5.14
 - FINDBHL(ITEM);
 - if(LOC==null) {
System.out.println("ข้อมูล:"+ITEM+" ไม่อยู่ใน List");
 - }else
 - {
LOC.PLINK = LOC.LINK;
 - }
 - }

ลิงค์ลิสต์แบบสองทาง (TWO-Way Link Lists)

- แต่ละโนดประกอบด้วย 3 ส่วน
 - ข้อมูล (Information--INFOR)
 - พอยเตอร์ชี้ไปโนดต่อไป (Forward Pointer –FORW)
 - พอยเตอร์ชี้กลับไปโนดก่อนหน้า (Backward Pointer—BACK)

มุ่งมองลิงค์ลิสต์



โนด (Node)

ตัวเชื่อมไปยังโนดก่อนหน้า
(Backward Pointer)

ข้อมูลในโนด
(Information)

ตัวเชื่อมไปยังโนดต่อไป
(Forward Pointer)

อิมพลีเมนต์

- public class DNode {
- Object INFOR;
- DNode BACK=null;
- DNode FORW=null;
- }

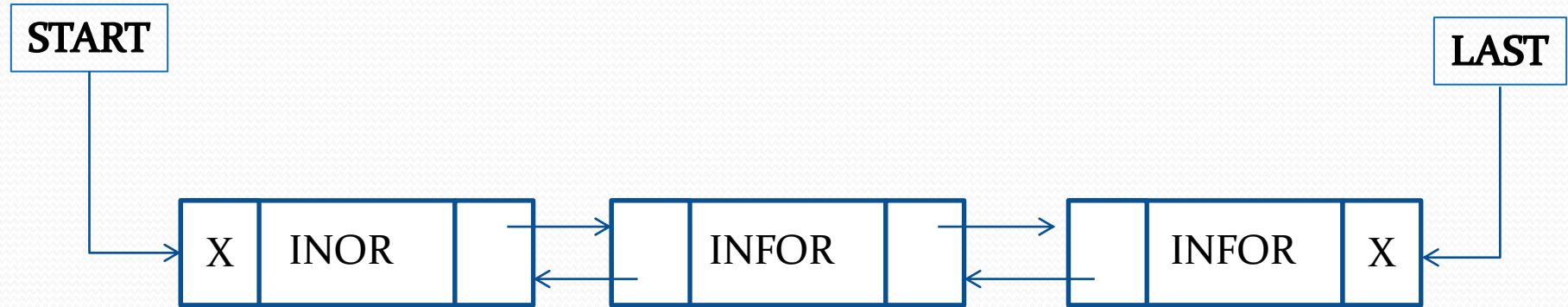
กิจกรรมนักศึกษา

- ทดลองเขียนโปรแกรมเพื่อสร้างໂນດ

คำศัพท์ที่เกี่ยวข้อง

- START
- LAST
- FORW
- BACK
- INFOR

มุ่งมองข้อมูลในลิงค์ลิสต์



การดำเนินการกับลิสต์แบบสองทาง

- การท่องลิงค์ลิสต์
- การค้นหาข้อมูล
- การแทรกข้อมูล
- การลบข้อมูล

การท่องลิงค์ลิสต์

- ใช้อัลกอริทึมการท่องตามปกติที่ใช้ในลิสต์แบบทางเดียว แต่สามารถ
 - ท่องจาก START จนถึง LAST
 - ท่องจาก LAST ย้อนมาจนถึง START

ตัวอย่างการอิมเพลี่เมนต์การท่องลิสต์

```

• package linklistimpdoublelist;

• public class Algor51TraversingList {
•     static DNode START;
•     static DNode LAST;
•     public static void main(String[] args) {
•         DNode NodeA = new DNode(); NodeA.INFOR = 10;
•         DNode NodeB = new DNode(); NodeB.INFOR = 20;
•         DNode NodeC = new DNode(); NodeC.INFOR = 30;
•         DNode NodeD = new DNode(); NodeD.INFOR = 40;
•         DNode NodeE = new DNode(); NodeE.INFOR = 50;
•         START = NodeA;
•         NodeA.FORW = NodeB; NodeB.BACK = NodeA;
•         NodeB.FORW = NodeC; NodeC.BACK = NodeB;
•         NodeC.FORW = NodeD; NodeD.BACK = NodeC;
•         NodeD.FORW = NodeE; NodeE.BACK = NodeD;
•         NodeE.FORW = null; LAST = NodeE;
•         System.out.println("Forward Traversing:");
•         ForwardTraversingList();
•         System.out.println("Backward Traversing:");
•         BackwardTraversingList();
•     }
•     //end main
•
•     public static void ForwardTraversingList(){
•         DNode PTR=null;
•         PTR=START;
•         while(PTR!=null){
•             System.out.println(PTR.INFOR); //Apply PROCESS to INFOR[PTR]
•             PTR=PTR.FORW;
•         }
•     }
•     public static void BackwardTraversingList(){
•         DNode PTR=null;
•         PTR=LAST;
•         while(PTR!=null){
•             System.out.println(PTR.INFOR); //Apply PROCESS to INFOR[PTR]
•             PTR=PTR.BACK;
•         }
•     }
}

```

- public class Algor51TraversingList {
- static DNode START;
- static DNode LAST;

- public static void main(String[] args) {
 - DNode NodeA = new DNode(); NodeA.INFOR = 10;
 - DNode NodeB = new DNode(); NodeB.INFOR = 20;
 - DNode NodeC = new DNode(); NodeC.INFOR = 30;
 - DNode NodeD = new DNode(); NodeD.INFOR = 40;
 - DNode NodeE = new DNode(); NodeE.INFOR = 50;
 - START = NodeA;
 - NodeA.FORW =NodeB; NodeB.BACK = NodeA;
 - NodeB.FORW =NodeC; NodeC.BACK = NodeB;
 - NodeC.FORW =NodeD; NodeD.BACK = NodeC;
 - NodeD.FORW =NodeE; NodeE.BACK = NodeD;
 - NodeE.FORW =null; LAST=NodeE;
 - System.out.println("Forward Traversing:");
 - ForwardTraversingList();
 - System.out.println("Backward Traversing:");
 - BackwardTraversingList();
 - }
- //end main

```
• public static void ForwardTraversingList(){
  •     DNode PTR=null;
  •     PTR=START;
  •     while(PTR!=null){
  •         System.out.println(PTR.INFOR); //Apply PROCESS to
INFOR[PTR]
  •         PTR=PTR.FORW;
  •     }
  • }
• public static void BackwardTraversingList(){
  •     DNode PTR=null;
  •     PTR=LAST;
  •     while(PTR!=null){
  •         System.out.println(PTR.INFOR); //Apply PROCESS to
INFOR[PTR]
  •         PTR=PTR.BACK;
  •     }
  • }
```

กิจกรรมนักศึกษา

- ทดลองรันโปรแกรมตามอัลกอริทึมตัวอย่าง
- เขียนโปรแกรมตามที่กำหนด

การค้นหา

- การค้นหาข้อมูลในลิสต์ ยังคงใช้อัลกอริทึมการค้นหาที่ผ่านมาได้
 - เริ่มค้นโดยเอาพอยเตอร์ PTR ชี้ที่ START และท่องเปรียบเทียบ
 - หรือหากต้องการค้นย้อนกลับจาก LAST ก็สามารถดำเนินการได้

ตัวอย่างการอิมพลีเมนต์

```

• package linklistimpdoublelist;
•
• public class Algor52SEARCH {
•     static DNode START, LAST;
•     public static void main(String[] args) {
•         DNode NodeA = new DNode(); NodeA.INFOR = 20;
•         DNode NodeB = new DNode(); NodeB.INFOR = 30;
•         DNode NodeC = new DNode(); NodeC.INFOR = 40;
•         DNode NodeD = new DNode(); NodeD.INFOR = 50;
•         DNode NodeE = new DNode(); NodeE.INFOR = 60;
•         DNode NodeF = new DNode(); NodeF.INFOR = 70;
•         DNode NodeG = new DNode(); NodeG.INFOR = 80;
•         DNode NodeH = new DNode(); NodeH.INFOR = 90;
•         DNode NodeI = new DNode(); NodeI.INFOR = 100;
•         START = NodeA;
•         NodeA.FORW =NodeB; NodeB.BACK = NodeA;
•         NodeB.FORW =NodeC; NodeC.BACK = NodeB;
•         NodeC.FORW =NodeD; NodeD.BACK = NodeC;
•         NodeD.FORW =NodeE; NodeE.BACK = NodeD;
•         NodeE.FORW =NodeF; NodeF.BACK = NodeE;
•         NodeF.FORW =NodeG; NodeG.BACK = NodeF;
•         NodeG.FORW =NodeH; NodeH.BACK = NodeG;
•         NodeH.FORW =NodeI; NodeI.BACK = NodeH;
•         NodeI.FORW =null; LAST = NodeH;
•         //Search Data
•         SEARCH(30);
•     }
•     public static void SEARCH(int ITEM){
•         //Algorrithm 5.2 SEARCH
•         DNode LOC=null;
•         DNode PTR=null;
•         PTR=START;
•         while(PTR!=null){
•             if(ITEM==(int)PTR.INFOR) {
•                 LOC = PTR;
•                 break;
•             }else
•                 PTR=PTR.FORW;
•         }
•         //Show Result;
•         if(LOC ==null)
•             System.out.println("Not found.");
•         else
•             System.out.println("Found.");
•     }
• }

```

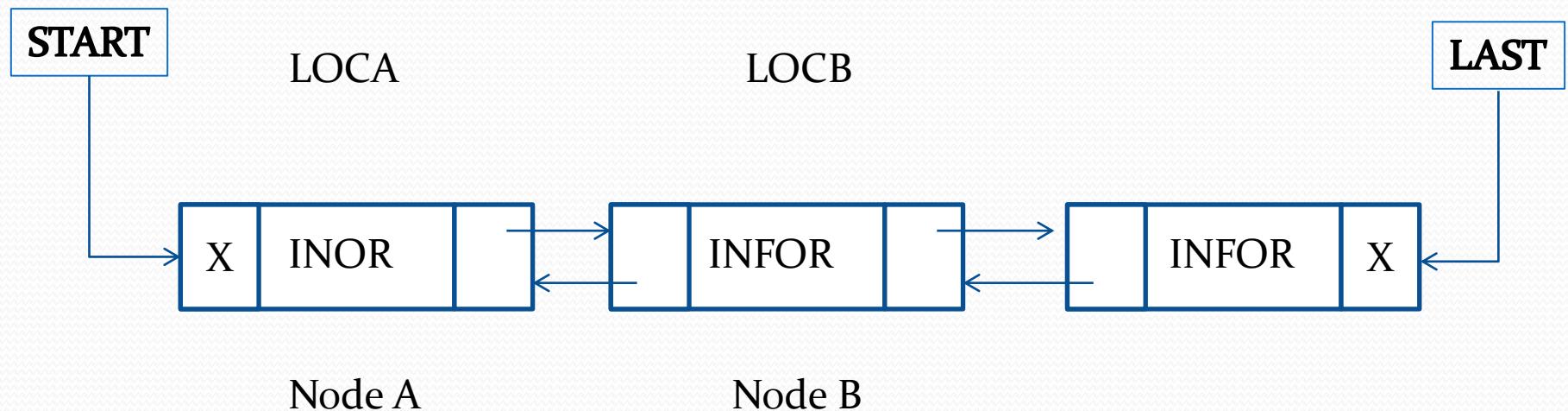
```
• public class Algor52SEARCH {  
•     static DNode START, LAST;  
•     public static void main(String[] args) {  
•         DNode NodeA = new DNode(); NodeA.INFOR = 20;  
•         DNode NodeB = new DNode(); NodeB.INFOR = 30;  
•         DNode NodeC = new DNode(); NodeC.INFOR = 40;  
•         DNode NodeD = new DNode(); NodeD.INFOR = 50;  
•         DNode NodeE = new DNode(); NodeE.INFOR = 60;  
•         DNode NodeF = new DNode(); NodeF.INFOR = 70;  
•         DNode NodeG = new DNode(); NodeG.INFOR = 80;  
•         DNode NodeH = new DNode(); NodeH.INFOR = 90;  
•         DNode NodeI = new DNode(); NodeI.INFOR = 100;  
•         START = NodeA;  
•         NodeA.FORW =NodeB; NodeB.BACK = NodeA;  
•         NodeB.FORW =NodeC; NodeC.BACK = NodeB;  
•         NodeC.FORW =NodeD; NodeD.BACK = NodeC;  
•         NodeD.FORW =NodeE; NodeE.BACK = NodeD;  
•         NodeE.FORW =NodeF; NodeF.BACK = NodeE;  
•         NodeF.FORW =NodeG; NodeG.BACK = NodeF;  
•         NodeG.FORW =NodeH; NodeH.BACK = NodeG;  
•         NodeH.FORW =NodeI; NodeI.BACK = NodeH;  
•         NodeI.FORW =null; LAST = NodeH;  
•         //Search Data  
•         SEARCH(30);  
•     }
```

- public static void SEARCH(int ITEM){
• //Algorithm 5.2 SEARCH
• DNode LOC=null;
• DNode PTR=null;
• PTR=START;
• while(PTR!=null){
• if(ITEM==(int)PTR.INFOR) {
• LOC = PTR;
• break;
• }else
• PTR=PTR.FORW;
• }
• //Show Result;
• if(LOC ==null)
• System.out.println("Not found.");
• else
• System.out.println("Found.");
• }
• }

การแทรกโนดใหม่ในลิสต์แบบสองทาง

- การแทรกจะต้องคำนึงพ้อยเตอร์ที่ซึ่ไปข้างหน้าและย้อนกลับ
- โดยกำหนดให้โนดก่อนจะตำแหน่งแทรก เป็นโนด A และ หลังการแทรกรคือโนด B
 - กำหนดตำแหน่งเป็น LOCA และ LOCB ตามลำดับ

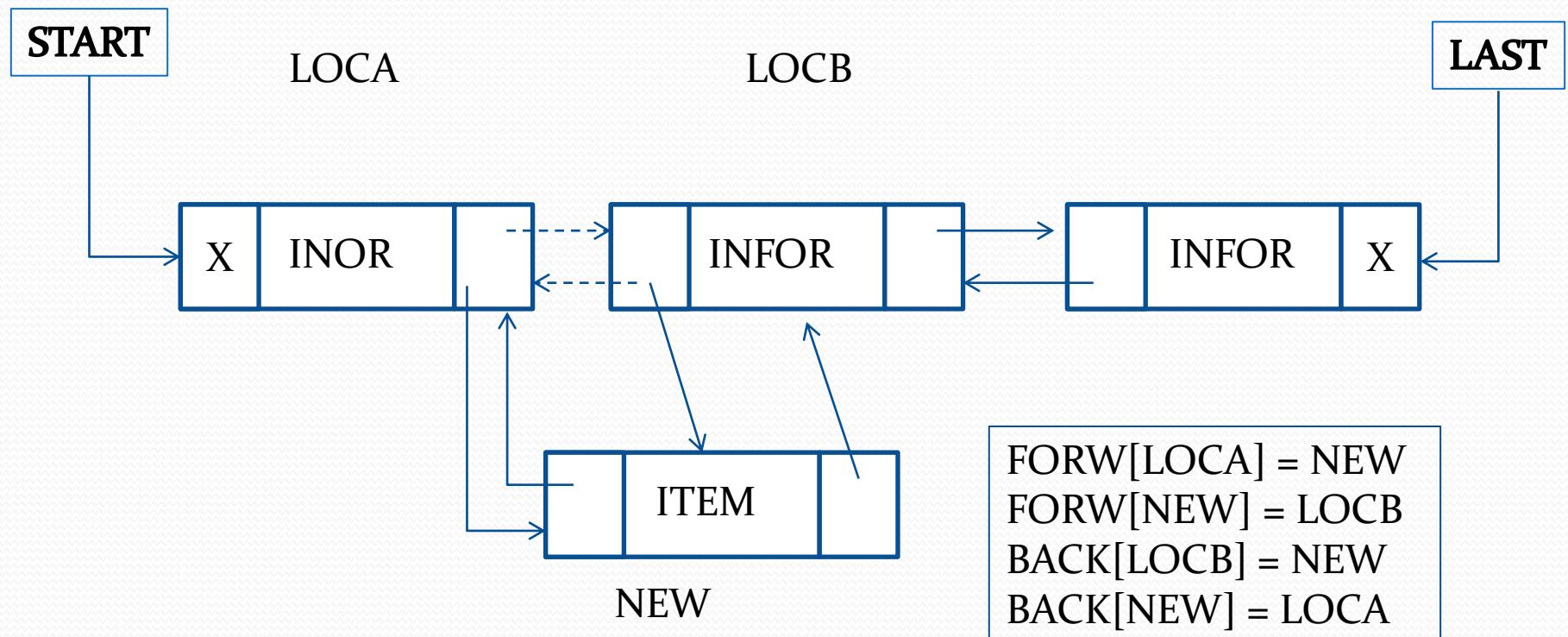
แสดงแนวคิดกำหนดตัวประกอบแทรก



ขณะดำเนินการแทรก

- FORW[LOCA] = NEW
- FORW[NEW] = LOCB
- BACK[LOCB] = NEW
- BACK[NEW] = LOCA

ແສດງແນວຄົດຂະໜະເທຣກ

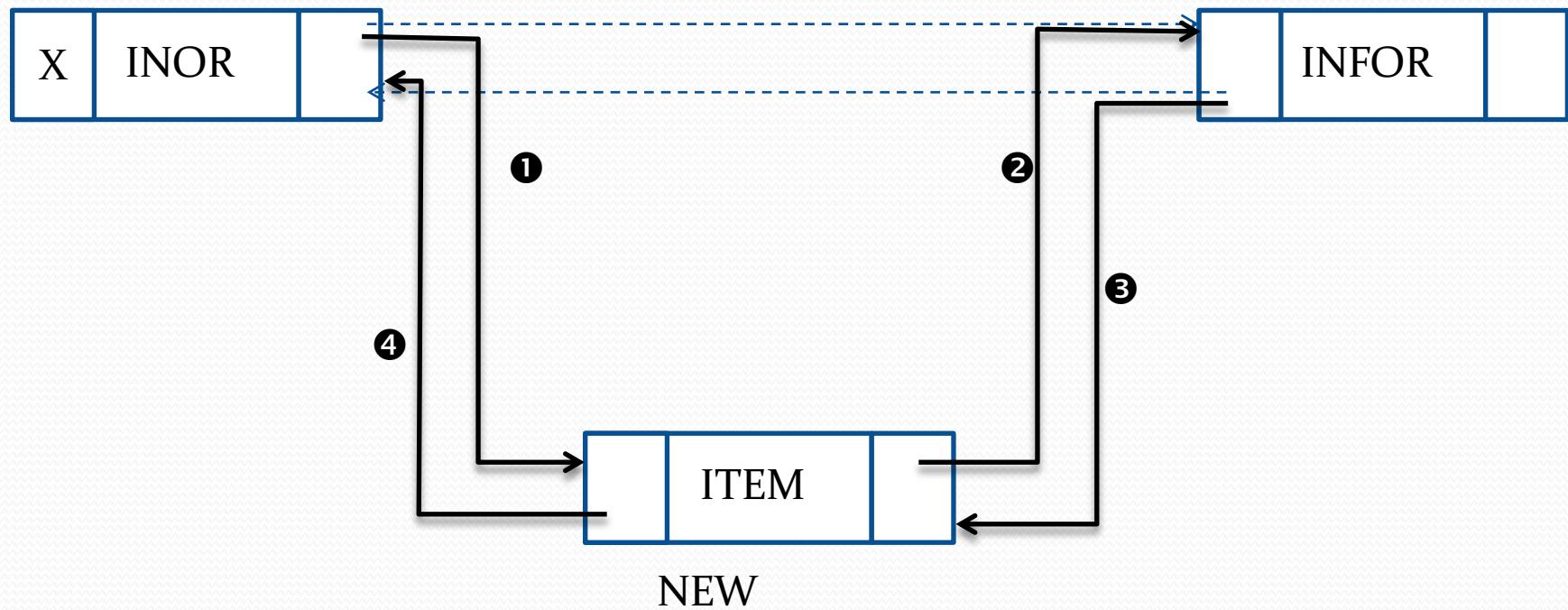


แสดงแนวคิดขนะโทรก

FORW[LOCA] = NEW
FORW[NEW] = LOCB
BACK[LOCB] = NEW
BACK[NEW] = LOCA

LOCA

LOCB



- Algorithm INSTWL Insert Nod in Two-ways Link Lists
- Input : LIST, START, INFOR, FORW, BACK, LOC, NEW
- Output : Node LOC is inserted into LIST.
 $FORW[LOCA] = NEW$
 $FORW[NEW] = LOCB$
 $BACK[LOCB] = NEW$
 $BACK[NEW] = LOCA$
- Return

อิมพลีเมนต์

```

• package linklistimpdoublelist;
•
• public class Algor516INSTWL {
•     static DNode START;
•     static DNode LAST;
•
•     public static void main(String[] args) {
•         DNode NodeA = new DNode(); NodeA.INFOR = 10;
•         DNode NodeB = new DNode(); NodeB.INFOR = 20;
•         DNode NodeC = new DNode(); NodeC.INFOR = 30;
•         DNode NodeD = new DNode(); NodeD.INFOR = 40;
•         DNode NodeE = new DNode(); NodeE.INFOR = 50;
•         DNode NodeF = new DNode(); NodeF.INFOR = 35; //New Node to be inserted.
•         START = NodeA;
•         NodeA.FORW =NodeB; NodeB.BACK = NodeA;
•         NodeB.FORW =NodeC; NodeC.BACK = NodeB;
•         NodeC.FORW =NodeD; NodeD.BACK = NodeC;
•         NodeD.FORW =NodeE; NodeE.BACK = NodeD;
•         NodeE.FORW =null; LAST=NodeE;
•         System.out.println("Before Inserting:");
•         ForwardTraversingList();
•         INSTWL(NodeC,NodeD, NodeF);
•         System.out.println("After Inserting:");
•         ForwardTraversingList();
•     }
• //end main
•
•     public static void INSTWL(DNode LOCA,DNode LOCB, DNode NEW){
•         LOCA.FORW = NEW; NEW.FORW = LOCB;
•         LOCB.BACK = NEW; NEW.BACK = LOCA;
•     }
•
•     public static void ForwardTraversingList(){
•         DNode PTR=null;
•         PTR=START;
•         while(PTR!=null){
•             System.out.println(PTR.INFOR); //Apply PROCESS to INFOR[PTR]
•             PTR=PTR.FORW;
•         }
•     }
• }

```

- public class Algor516INSTWL {
- static DNode START;
- static DNode LAST;

- public static void main(String[] args) {
 DNode NodeA = new DNode(); NodeA.INFOR = 10;
 DNode NodeB = new DNode(); NodeB.INFOR = 20;
 DNode NodeC = new DNode(); NodeC.INFOR = 30;
 DNode NodeD = new DNode(); NodeD.INFOR = 40;
 DNode NodeE = new DNode(); NodeE.INFOR = 50;
 DNode NodeF = new DNode(); NodeF.INFOR = 35; //New Node to be inserted.
- START = NodeA;
 NodeA.FORW =NodeB; NodeB.BACK = NodeA;
 NodeB.FORW =NodeC; NodeC.BACK = NodeB;
 NodeC.FORW =NodeD; NodeD.BACK = NodeC;
 NodeD.FORW =NodeE; NodeE.BACK = NodeD;
 NodeE.FORW =null; LAST=NodeE;
 System.out.println("Before Inserting:");
 ForwardTraversingList();
 INSTWL(NodeC,NodeD, NodeF);
 System.out.println("After Inserting:");
 ForwardTraversingList();
 }- //end main

- public static void INSTWL(DNode LOCA,DNode LOCB,
DNode NEW){
 - LOCA.FORW = NEW; NEW.FORW = LOCB;
 - LOCB.BACK = NEW; NEW.BACK = LOCA;
 - }
-
- public static void ForwardTraversingList(){
 - DNode PTR=null;
 - PTR=START;
 - while(PTR!=null){
 - System.out.println(PTR.INFOR); //Apply PROCESS
to INFOR[PTR]
 - PTR=PTR.FORW;
 - }
 - }

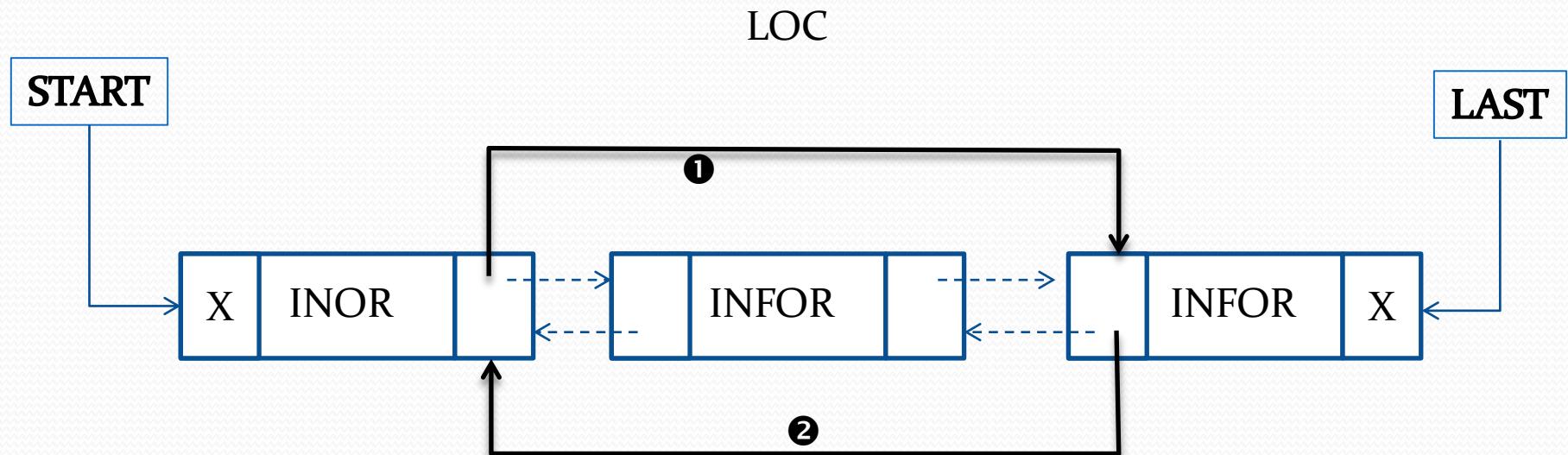
กิจกรรมนักศึกษา

- รันโปรแกรมทดสอบตามอัลกอริทึม
- เขียนโปรแกรมตามที่กำหนด

การลบในดของลิสต์แบบสองทาง

- นำເອົາພອຍເຕັມໂນດທີ່ອຸ່ງຫັ້ງລັບໂນດທີ່ຈະລັບຊື່ຂໍາມກັບໄປຢັງໂນດທີ່ອຸ່ງກ່ອນໜ້າໂນດທີ່ຈະລັບ
- นำເອົາພອຍເຕັມໂນດກ່ອນທີ່ຈະລັບຊື່ຂໍາມໄປຢັງໂນດຫັ້ງໂນດທີ່ຈະລັບ
 - $\text{FORW}[\text{BACK}[\text{LOC}]] = \text{FORW}[\text{LOC}]$
 - $\text{BACK}[\text{FORW}[\text{LOC}]] = \text{BACK}[\text{LOC}]$

แสดงแนวคิดขนะลับโนด



$$\textcircled{1} \text{ FORW[BACK[LOC]] = FORW[LOC]}$$

$$\textcircled{2} \text{ BACK[FORW[LOC]] = BACK[LOC]}$$

- Algorithm DELTWL Deletion Nod in Two-ways Link Lists
- Input : LIST, START, INFOR, FORW, BACK, LOC
- Output : Node LOC is deleted form LIST.
- $\text{FORW}[\text{BACK}[\text{LOC}]] = \text{FORW}[\text{LOC}]$
- $\text{BACK}[\text{FORW}[\text{LOC}]] = \text{BACK}[\text{LOC}]$
- Return

อิมพลีเมนต์

```

• package linklistimpdoublelist;

• public class Algor515DELTWL {
    static DNode START;
    static DNode LAST;

    • public static void main(String[] args) {
        DNode NodeA = new DNode(); NodeA.INFOR = 10;
        DNode NodeB = new DNode(); NodeB.INFOR = 20;
        DNode NodeC = new DNode(); NodeC.INFOR = 30;
        DNode NodeD = new DNode(); NodeD.INFOR = 40;
        DNode NodeE = new DNode(); NodeE.INFOR = 50;
        START = NodeA;
        NodeA.FORW = NodeB; NodeB.BACK = NodeA;
        NodeB.FORW = NodeC; NodeC.BACK = NodeB;
        NodeC.FORW = NodeD; NodeD.BACK = NodeC;
        NodeD.FORW = NodeE; NodeE.BACK = NodeD;
        NodeE.FORW = null; LAST = NodeE;
        System.out.println("Before Deleting:");
        ForwardTraversingList();
        DELTWL(NodeC);
        System.out.println("After Deleting:");
        ForwardTraversingList();
    }
    //end main

    • public static void DELTWL(DNode LOC){
        LOC.BACK.FORW = LOC.FORW;
        LOC.FORW.BACK = LOC.BACK;
    }

    • public static void ForwardTraversingList(){
        DNode PTR=null;
        PTR=START;
        while(PTR!=null){
            System.out.println(PTR.INFOR); //Apply PROCESS to INFOR[PTR]
            PTR=PTR.FORW;
        }
    }
}

```

- public class Algor515DELTWL {
- static DNode START;
- static DNode LAST;

- public static void main(String[] args) {
 - DNode NodeA = new DNode(); NodeA.INFOR = 10;
 - DNode NodeB = new DNode(); NodeB.INFOR = 20;
 - DNode NodeC = new DNode(); NodeC.INFOR = 30;
 - DNode NodeD = new DNode(); NodeD.INFOR = 40;
 - DNode NodeE = new DNode(); NodeE.INFOR = 50;
 - START = NodeA;
 - NodeA.FORW =NodeB; NodeB.BACK = NodeA;
 - NodeB.FORW =NodeC; NodeC.BACK = NodeB;
 - NodeC.FORW =NodeD; NodeD.BACK = NodeC;
 - NodeD.FORW =NodeE; NodeE.BACK = NodeD;
 - NodeE.FORW =null; LAST=NodeE;
 - System.out.println("Before Deleting:");
 - ForwardTraversingList();
 - DELTWL(NodeC);
 - System.out.println("After Deleting:");
 - ForwardTraversingList();
 - }
- //end main

- public static void DELTWL(DNode LOC){
 LOC.BACK.FORW = LOC.FORW;
 LOC.FORW.BACK = LOC.BACK;
}
- public static void ForwardTraversingList(){
 DNode PTR=null;
 PTR=START;
 while(PTR!=null){
 System.out.println(PTR.INFOR); //Apply PROCESS
 to INFOR[PTR]
 PTR=PTR.FORW;
 }
}

กิจกรรมนักศึกษา

- รันโปรแกรมทดสอบตามอัลกอริทึม
- เขียนโปรแกรมตามที่กำหนด

Java LinkedList ADT

`java.util.LinkedList`

● add(Object e)	boolean
● add(int i, Object e)	void
● addAll(Collection clctn)	boolean
● addAll(int i, Collection clctn)	boolean
● addFirst(Object e)	void
● addLast(Object e)	void
● clear()	void
● clone()	Object
● contains(Object o)	boolean
● containsAll(Collection clctn)	boolean
● descendingIterator()	Iterator
● element()	Object
● equals(Object o)	boolean
● get(int i)	Object
● getClass()	Class<?>
● getFirst()	Object
● getLast()	Object
● hashCode()	int
● indexOf(Object o)	int
● isEmpty()	boolean
● iterator()	Iterator
● lastIndexOf(Object o)	int
● listIterator()	ListIterator
● listIterator(int i)	ListIterator
● notify()	void
● notifyAll()	void
● offer(Object e)	boolean
● offerFirst(Object e)	boolean
● offerLast(Object e)	boolean
● peek()	Object
● peekFirst()	Object
● peekLast()	Object
● poll()	Object
● pollFirst()	Object

● peekFirst()	Object
● peekLast()	Object
● poll()	Object
● pollFirst()	Object
● pollLast()	Object
● pop()	Object
● push(Object e)	void
● remove()	Object
● remove(Object o)	boolean
● remove(int i)	Object
● removeAll(Collection clctn)	boolean
● removeFirst()	Object
● removeFirstOccurrence(Object o)	boolean
● removeLast()	Object
● removeLastOccurrence(Object o)	boolean
● retainAll(Collection clctn)	boolean
● set(int i, Object e)	Object
● remove(int i)	Object
● removeAll(Collection clctn)	boolean
● removeFirst()	Object
● removeFirstOccurrence(Object o)	boolean
● removeLast()	Object
● removeLastOccurrence(Object o)	boolean
● retainAll(Collection clctn)	boolean
● set(int i, Object e)	Object
● size()	int
● subList(int i, int il)	List
● toArray()	Object[]
● toArray(Object[] ts)	Object[]
● toString()	String
● wait()	void
● wait(long l)	void
● wait(long l, int i)	void
● new	

```
• package JavaADT;  
• import java.util.LinkedList;  
• public class implLinkLists1 {  
•     public static void main(String args[]) {  
•         LinkedList<String> myList= new LinkedList();  
•         myList.add("test");  
•         myList.add("Nok");  
•         myList.add("kai");  
•         myList.add("Jat");  
•         System.out.println(myList);  
•     }  
• }
```

```
• package JavaADT;  
• import java.util.LinkedList;  
  
• public class SimpleJavaLinkedListExample {  
•     •     •     public static void main(String[] args) {  
•         •         •         LinkedList myList = new LinkedList();  
•         •         •         //add elements to LinkedList  
•         myList.add(1);  
•         myList.add(2);  
•         myList.add(3);  
•         myList.add(4);  
•         myList.add(5);  
•         System.out.println("LinkedList contains : " + myList);  
•         System.out.println(myList.getFirst());  
•         System.out.println(myList.peek());  
•     }  
• }
```

- package JavaADT;
- import java.util.LinkedList;
-
- public class SimpleJavaLinkedListExample1 {
-
- public static void main(String[] args) {
-
- LinkedList myList = new LinkedList();
- myList.add(1);
- myList.add(2);
- myList.add(3);
- myList.add(4);
- myList.add(5);
- System.out.println(myList.indexOf(3));
- System.out.println(myList.get(myList.indexOf(3)));
- }
- }

- package JavaADT;
- import java.util.LinkedList;
- import java.util.Collections;
-
- public class SimpleJavaLinkedListExample2 {
-
- public static void main(String[] args) {
-
- LinkedList myList = new LinkedList();
- myList.add(4);
- myList.add(2);
- myList.add(9);
- myList.add(1);
- myList.add(5);
- Collections.sort(myList);
- System.out.println(myList);
-
- }
- }

- package JavaADT;
- import java.util.Iterator;
- import java.util.List;
- import java.util.ArrayList;
- public class ListExample {
- public static void main(String[] args) {
- // List Example implement with ArrayList
- List<String> ls=new ArrayList<String>();
- ls.add("one");
- ls.add("Three");
- ls.add("two");
- ls.add("four");
- Iterator it=ls.iterator();
- while(it.hasNext())
- {
- String value=(String)it.next();
- System.out.println("Value :" +value);
- }
- }

- package JavaADT;
- import java.util.LinkedList;
- import java.util.Collections;

- public class implLinkListsSort {
- public static void main(String args[]) {
- LinkedList<String> myList= new LinkedList();
- myList.add("test");
- myList.add("Nok");
- myList.add("kai");
- myList.add("Jat");
- System.out.println("Before Sorting:"+myList);
- Collections.sort(myList);
- System.out.println("After Sorting:"+myList);
- }
- }

- package JavaADT;
- import java.util.LinkedList;
- import java.util.Collections;
- public class implLinkListsIns {
- public static void main(String args[]) {
- LinkedList<String> myList= new LinkedList();
- myList.add("test");
- myList.add("Nok");
- myList.add("kai");
- myList.add("Jat");
- Collections.sort(myList);
- System.out.println("Before Inserting:"+myList);
- myList.add("Boy");
- myList.add("Zombies");
- Collections.sort(myList);
- System.out.println("Before Inserting:"+myList);
- }
- }

- package JavaADT;
- import java.util.LinkedList;
- public class implLinkListsRmv {
- public static void main(String args[]) {
- LinkedList<String> myList= new LinkedList();
- myList.add("test");
- myList.add("Nok");
- myList.add("kai");
- myList.add("Jat");
- System.out.println("Before Removing:"+myList);
- myList.remove(myList.indexOf("Nok"));
- System.out.println("After Removing:"+myList);
- }
- }

References

- Seymour Lipschutz:แปลโดย อุดม จินประดับ และ ดร.สมคิด เว่องชนา
สกุลไทย. ทฤษฎีและตัวอย่างโจทย์โครงสร้างข้อมูล---**Theory and problem of Data Structures.** กรุงเทพฯ : แมคกรอ-อิล อินเตอร์เนชันแนล เอ็นเตอร์ไพรส์, อิงค์, (2540).
- เชาวลิต ขันคำ. (2554). การเขียนโปรแกรมภาษาคอมพิวเตอร์. ฉะเชิงเทรา : มหาวิทยาลัยราชภัฏราชบูรณะครินทร์.