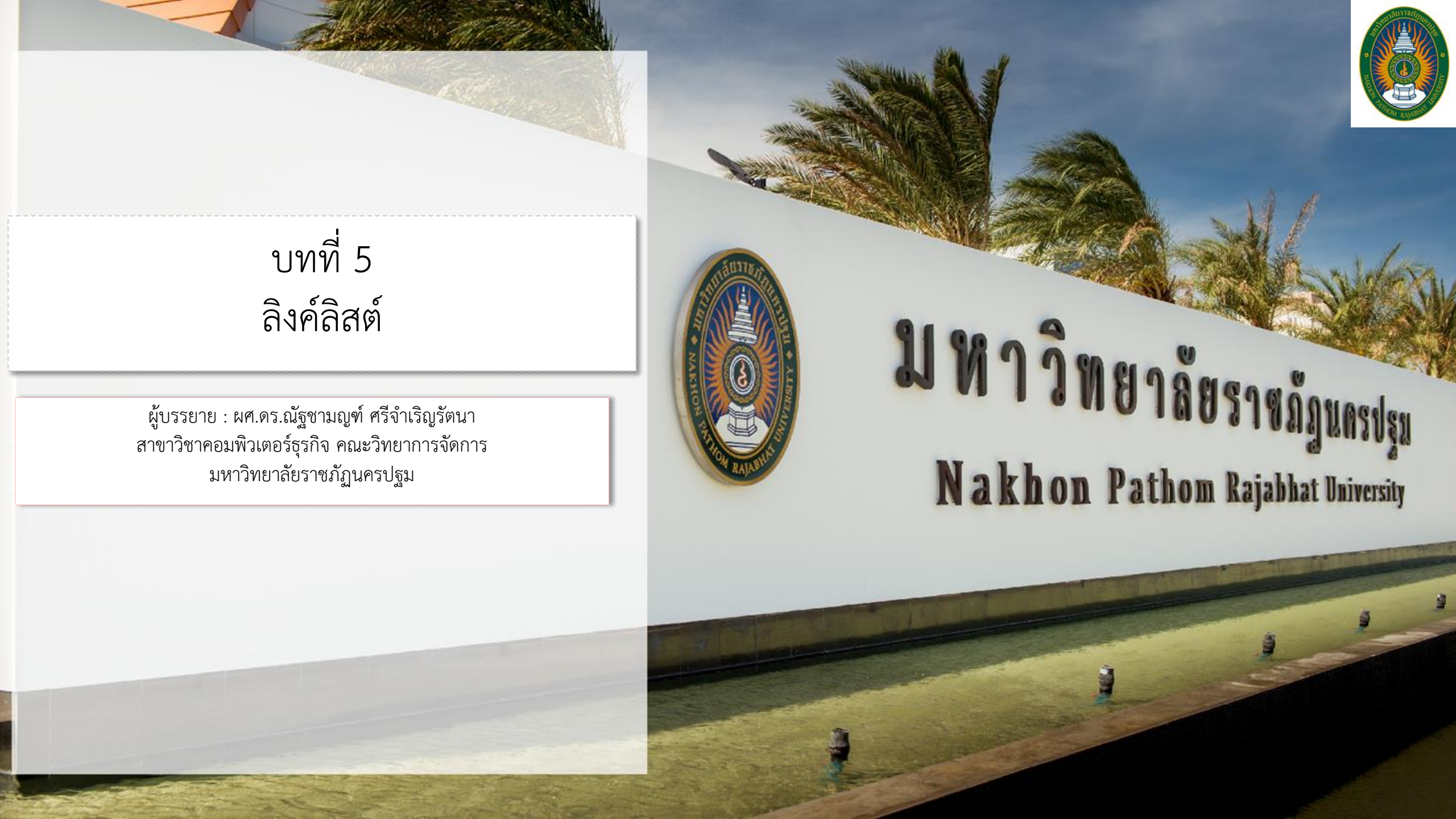
มหาวิทยาลัยราชภัฏนครปฐม



บทที่ 5
ลิงค์ลิสต์

ผู้บรรยาย : ผศ.ดร.ณัฐชามญฑ์ ศรีจำเริญรัตนา
สาขาวิชาคอมพิวเตอร์ธุรกิจ คณะวิทยาการจัดการ
มหาวิทยาลัยราชภัฏนครปฐม

ลิงค์ลิสต์

- พื้นฐานของลิงค์ลิสต์ เช่น วิธีการสร้างลิงค์ลิสต์ วิธีการเพิ่ม วิธีการลบ วิธีการค้นหาข้อมูล และวิธีอื่น ๆ ลิงค์ลิสต์เป็นทางเลือกหนึ่งสำหรับการเก็บข้อมูลที่มีความสัมพันธ์กัน
- ลิงค์ลิสต์สามารถใช้เก็บข้อมูลแทนอาร์เรย์ได้ เนื่องจากลิงค์ลิสต์มีลักษณะเป็นโครงสร้างข้อมูลแบบเชิงเส้นที่มีความคล้ายกับอาร์เรย์ที่เป็นรายการลำดับที่ต่อเนื่องกัน

ลิงค์ลิสต์

ข้อจำกัดในการใช้งานอาร์เรย์

- ข้อมูลอาร์เรย์ 100 อิลิเมนต์ ต้องจองพื้นที่ในหน่วยความจำเท่ากับจำนวนข้อมูลที่ต้องการใช้

  นั่นคือ 100 อิลิเมนต์ ถึงแม้จะลบข้อมูลบางส่วนในอาร์เรย์ไปแล้วอาร์เรย์ก็จะยังใช้พื้นที่ใน

  หน่วยความจำเท่ากับ 100 เช่นเดิม

- ลิงค์ลิสต์ ได้รับการพัฒนาเพื่อเป็นทางเลือกในการใช้งานที่เหมาะสม เพื่อให้หน่วยความจำสามารถนำมาใช้ได้เต็มประสิทธิภาพ เนื่องจากพื้นที่ข้อมูลที่ลิงค์ลิสต์ใช้งานอยู่นั้นจะเท่ากับข้อมูลที่กำลังใช้งานอยู่ เช่น ลิงค์ลิสต์มีข้อมูลที่ทำการใช้งานอยู่จำนวนหนึ่ง ต่อมาได้ทำการลบข้อมูลบางส่วนทิ้งไป พื้นที่ของลิงค์ลิสต์ที่กำลังใช้งานอยู่นี้ก็จะเท่ากับจำนวนลิงค์ลิสต์ที่เหลืออยู่ ส่วนพื้นที่ของข้อมูลที่ถูกลบออกไปก็จะสามารถนำไปใช้งานได้อย่างเต็มประสิทธิภาพต่อไป

5.1 ทำความรู้จักลิงค์ลิสต์

- ลิงค์ลิสต์ (link list) มีโครงสร้างข้อมูลที่ขนาดสามารถเปลี่ยนแปลงได้
- ลิงค์ลิสต์ประกอบด้วยสมาชิกหลาย ๆ สมาชิกเชื่อมต่อกัน
- โดยทั่วไปโครงสร้างของลิงค์ลิสต์ประกอบด้วย 2 ส่วน คือ
    1) โครงสร้างโหนดส่วนเริ่มต้นลิสต์
    2) โครงสร้างโหนดส่วนเนื้อหา

5.1 ทำความรู้จักลิงค์ลิสต์(ต่อ)

1) โครงสร้างโหนดส่วนเริ่มต้นลิสต์

โครงสร้างโหนดส่วนเริ่มต้นลิสต์ บางครั้งเรียกว่า head node สร้างขึ้นหลังการสร้างลิงค์ลิสต์ ประกอบด้วย 2 ส่วนคือ

- ส่วนแรก count เป็นตัวบอกจำนวนสมาชิกของโครงสร้างลิงค์ลิสต์ที่ถูกสร้างขึ้น
- ส่วนที่สอง head เป็นส่วนใช้ระบุตำแหน่งลิงค์ลิสต์ตัวถัดไป

โครงสร้างลิงค์ลิสต์โหนดส่วนเริ่มต้น

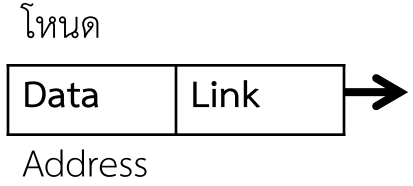
5.1 ทำความรู้จักลิงค์ลิสต์(ต่อ)

2) โครงสร้างโหนดส่วนเนื้อหา

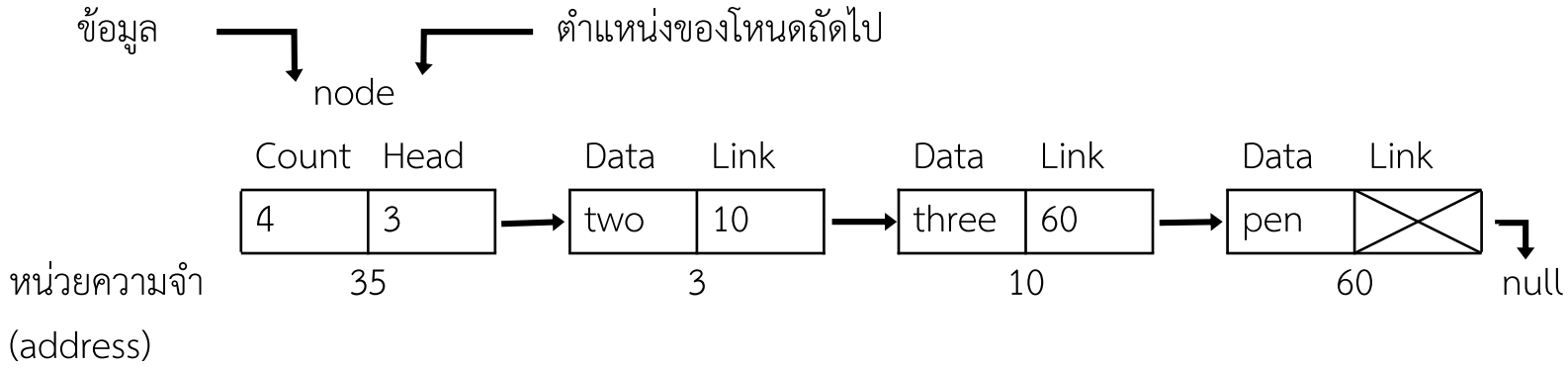
โหนดแต่ละตัวประกอบด้วย 2 ส่วน คือ

ส่วนแรกเป็นที่จัดเก็บข้อมูล (data)

ส่วนที่สองเป็นส่วนที่อ้างอิงตำแหน่งของโหนดถัดไป (link) ดังตัวอย่าง



โครงสร้างลิงค์ลิสต์โหนดส่วนเนื้อหา



5.2 ลิงค์ลิสต์ทิศทางเดียว

ลิงค์ลิสต์ทิศทางเดียว (singly link-list) ประกอบไปด้วยโครงสร้างโหนดที่มีพอยน์เตอร์ชี้ไปในทิศทางเดียวกันทั้งหมด คือ เริ่มจากโหนดส่วนเริ่มต้น และโหนดเริ่มต้นจะชี้ไปยังโหนดถัดไป เป็นลำดับต่อเนื่องไปเรื่อย ๆ โดยมีการดำเนินการของลิงค์ลิสต์ ดังต่อไปนี้

5.2 ลิงค์ลิสต์ทิศทางเดียว (ต่อ)

5.2.1 สร้างลิสต์

- โดยการประกาศโครงสร้างของลิงค์ลิสต์แบบสตรัคเจอร์
- ลิงค์ลิสต์ที่สร้างขึ้นประกอบด้วย “ข้อมูล” และตำแหน่งโหนดถัดไป นั่นคือ “พอยน์เตอร์”
- พอยน์เตอร์ทำหน้าที่ชี้ไปยังลิงค์ลิสต์ลำดับถัดไป
- จำนวนสมาชิกของลิงค์ลิสต์มีค่าเริ่มต้นเท่ากับศูนย์
- ลิงค์ลิสต์เริ่มต้นจะไม่มีการชี้ค่าตำแหน่งโหนดถัดไปดังนั้นค่าเริ่มต้นจึงเป็น Null

5.2 ลิงค์ลิสต์ทิศทางเดียว (ต่อ)

ตัวอย่างที่ 5.1 การประกาศโครงสร้างลิงค์ลิสต์แบบสตรัคเจอร์

```
1  struct node
2  {
3          int data;
4          node *next;
5  };
6  node *H_linklist;
```

5.2 ลิงค์ลิสต์ทิศทางเดียว (ต่อ)

5.2.2 การเพิ่มข้อมูลเข้าไปในลิงค์ลิสต์

การเพิ่มข้อมูลเข้าไปในลิสต์ต้องตรวจสอบให้แน่ชัดว่าโหนดที่จะทำการเพิ่มข้อมูลเข้าไปเป็นโหนดใดในลิสต์ การเพิ่มข้อมูลในลิงค์ลิสต์มี 3 รูปแบบ คือ

- การเพิ่มที่ตำแหน่งปลายของโหนดลิสต์
- การเพิ่มที่ตำแหน่งเริ่มต้น
- การเพิ่มโหนดใหม่ระหว่างโหนดที่มีข้อมูลอยู่แล้ว

ดังแสดงในขั้นตอนการจัดการลิงค์ลิสต์และแสดงการทำงาน ดังนี้

5.2 ลิงค์ลิสต์ทิศทางเดียว (ต่อ)

แบบที่ 1 การเพิ่มที่ตำแหน่งปลายของโหนดลิสต์

ต้องรู้ตำแหน่งที่ต้องการแทรกโหนดก่อน เมื่อทำการจองพื้นที่และทำการแทรกข้อมูลจะกำหนดให้พอยน์เตอร์โหนดสุดท้ายชี้ไปยังโหนดใหม่และพอยน์เตอร์ next ของโหนดใหม่จะชี้ไปยังตำแหน่งโหนดถัดไป

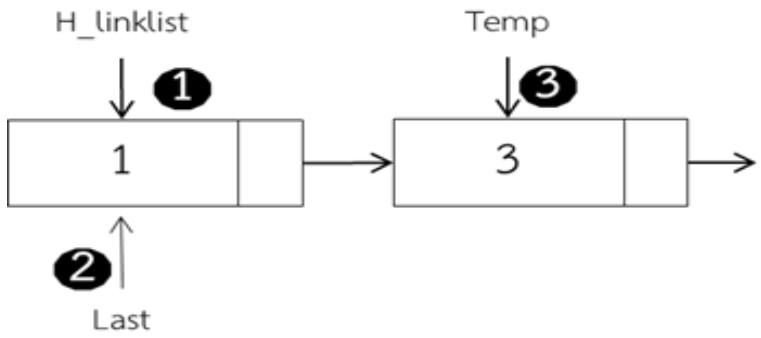
5.2 ลิงค์ลิสต์ทิศทางเดียว (ต่อ)

ตัวอย่างที่ 5.2 การเพิ่มข้อมูลเข้าไปในลิงค์ลิสต์ที่ตำแหน่งปลายของโหนดลิสต์

```
1   void addnode(int value)
2   {
3       node *temp =new node;
4       temp->data=value;
5       temp->next = NULL;
6       if(H_linklist==NULL)
7       {
8           H_linklist = temp;
9           return;
10          } else {
11          node* last = H_linklist;
12          while(last->next) last=last->next;
13          last->next = temp;
14      }
15  }
```

5.2 ลิงค์ลิสต์ทิศทางเดียว (ต่อ)

คำอธิบายการเพิ่มโหนดที่ตำแหน่งปลายของลิงค์ลิสต์



5.2 ลิงค์ลิสต์ทิศทางเดียว (ต่อ)

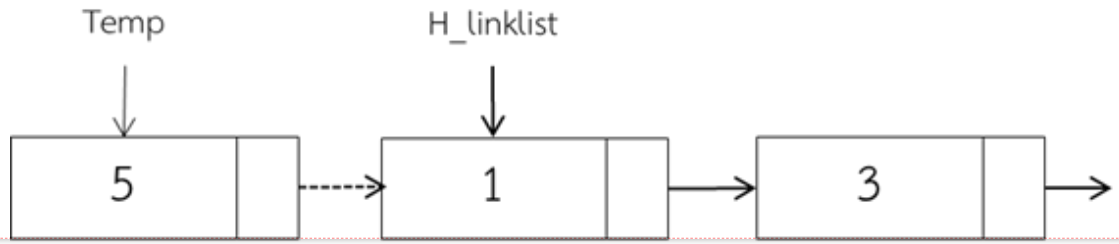
แบบที่ 2 การเพิ่มที่ตำแหน่งเริ่มต้น

กำหนดโหนดแรก first เป็นโหนดเริ่มต้น และพอยน์เตอร์ next ของโหนด first ชี้ไปยังโหนดถัดไป

ตัวอย่างที่ 5.3 การเพิ่มข้อมูลเข้าไปในลิงค์ลิสต์ที่ตำแหน่งเริ่มต้น

```
1  void add_first_node(int value)
2  {
3          node *temp = new node;
4          temp->data=value;
5          temp->next=H_linklist;
6          H_linklist=temp;
7  }
```

คำอธิบายการเพิ่มโหนดที่ตำแหน่งเริ่มต้นลิงค์ลิสต์



5.2 ลิงค์ลิสต์ทิศทางเดียว (ต่อ)

แบบที่ 3 การเพิ่มโหนดใหม่ระหว่างโหนดที่มีข้อมูลอยู่แล้ว

ค้นหาโหนดที่ต้องการแทรก กำหนดโหนดให้ค้นหาพบชี้ไปยังโหนดใหม่ และกำหนดโหนดใหม่ชี้ไปยังโหนดที่อยู่หลังโหนดที่ต้องการแทรก

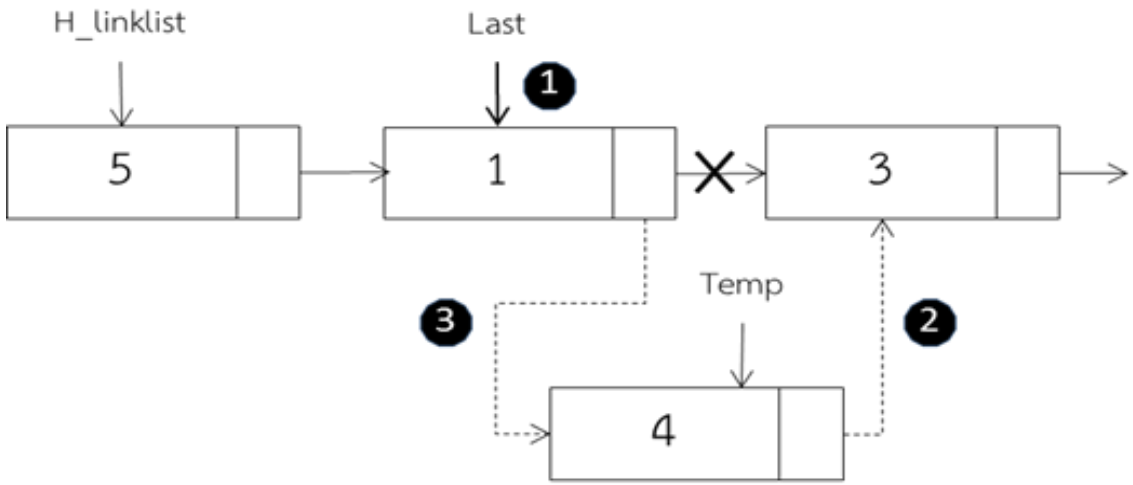
5.2 ลิงค์ลิสต์ทิศทางเดียว (ต่อ)

ตัวอย่างที่ 5.4 การเพิ่มข้อมูลเข้าไปในลิงค์ลิสต์ระหว่างโหนดข้อมูลเดิม

```
1   void insert_node(int fnode,int value)
2   {
3    node *temp=new node;
4    temp->data = value;
5    temp->next = NULL;
6    node* last= H_linklist;
7    if(fnode>1)
8    {
9    for(int j=2;j<fnode;j++) last=last->next;
10     temp->next = last->next;
11     last->next = temp;
12   }  else {
13     temp->next = H_linklist;
14     H_linklist=temp;
15   }
16  }
```

5.2 ลิงค์ลิสต์ทิศทางเดียว (ต่อ)

คำอธิบายการแทรกข้อมูลเข้าสู่ลิงค์ลิสต์



5.2 ลิงค์ลิสต์ทิศทางเดียว (ต่อ)

5.2.3 การลบข้อมูลออกจากลิงค์ลิสต์

การลบโหนดในลิงค์ลิสต์ นอกจากทำให้โหนดได้รับการปรับเปลี่ยนให้ถูกต้องแล้วยังทำให้มีการคืนหน่วยความจำที่ไม่ได้ใช้งานคืนกับหน่วยความจำอีกทั้งยังสามารถนำหน่วยความจำที่คืนนั้นไปใช้ประโยชน์อื่น ๆ ได้อีกด้วย

การลบข้อมูลมีการทำงานได้หลายแบบ โดยทั่วไปจะแบ่งได้ 3 แบบ คือ

- การลบข้อมูลออกจากลิงค์ลิสต์ด้านเริ่มต้น
- การลบข้อมูลออกจากลิงค์ลิสต์ด้านปลาย
- การลบโหนดในตำแหน่งที่ระบุ

5.2 ลิงค์ลิสต์ทิศทางเดียว (ต่อ)

1) การลบข้อมูลออกจากลิงค์ลิสต์ด้านเริ่มต้น

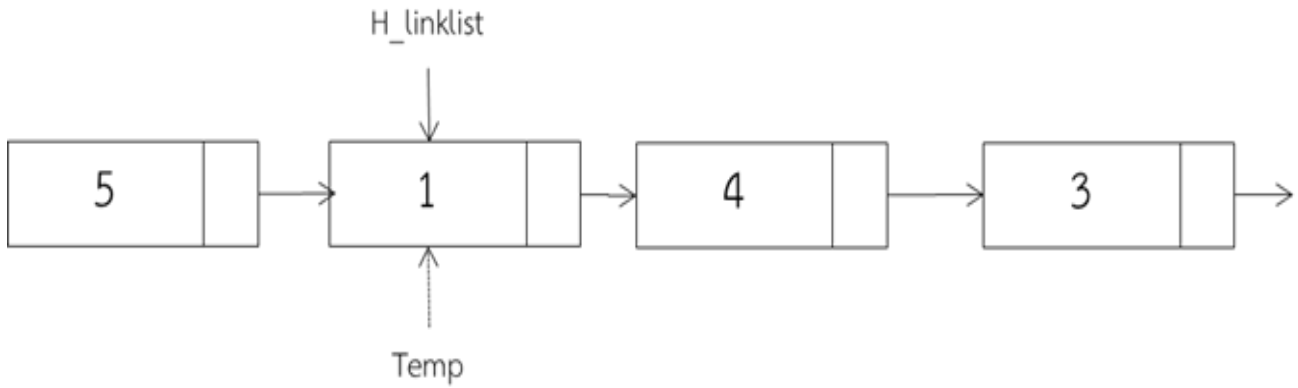
การลบโหนดในลิงค์ลิสต์จะทำให้เกิดการลดข้อมูลและยังเป็นการคืนค่าหน่วยความจำให้กับระบบ วิธีการ คือ กำหนดตำแหน่งโหนดที่ต้องการลบจากนั้นให้พอยน์เตอร์ที่ใช้สำหรับชี้ไปยังโหนดที่ต้องการลบ และให้ next ของโหนดก่อนหน้าชี้ไปยังโหนดที่อยู่ถัดต่อไปจากโหนดที่ต้องการลบ

5.2 ลิงค์ลิสต์ทิศทางเดียว (ต่อ)

ตัวอย่างที่ 5.5 การลบข้อมูลด้านเริ่มต้นออกจากลิงค์ลิสต์

```
1  void removefromfirst()
2  {
3          node *temp=new node;
4          temp->next = NULL;
5          temp = H_linklist->next;
6          H_linklist=temp;
7  }
```

คำอธิบายการลบข้อมูลส่วนเริ่มต้น



5.2 ลิงค์ลิสต์ทิศทางเดียว (ต่อ)

2) การลบข้อมูลออกจากลิงค์ลิสต์ด้านปลาย

การลบโหนดสุดท้ายสามารถทำได้โดยตรวจสอบโหนดอ้างอิงโหนดถัดไปที่มีค่า NULL เนื่องจากโหนดสุดท้ายเป็นโหนดที่ยังไม่มีข้อมูล

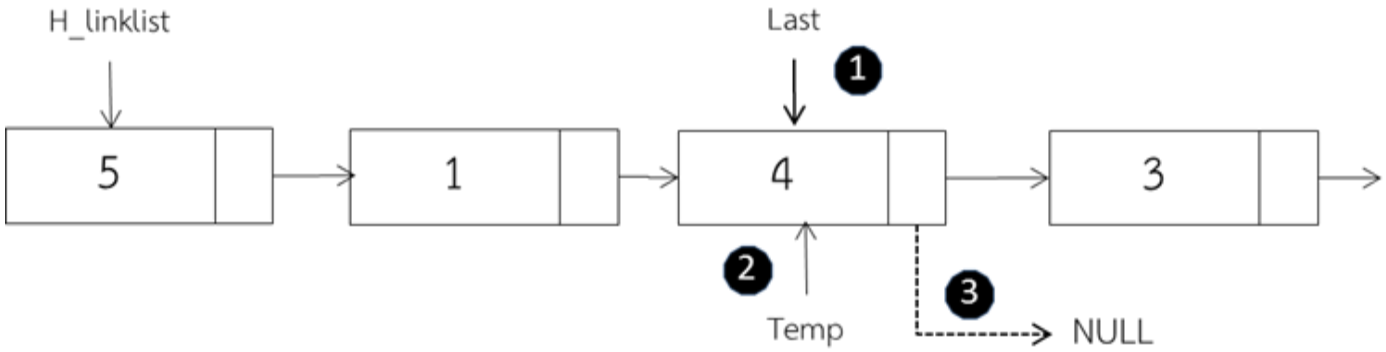
5.2 ลิงค์ลิสต์ทิศทางเดียว (ต่อ)

ตัวอย่างที่ 5.6 การลบข้อมูลออกจากลิงค์ลิสต์ด้านปลาย

```
1   void removefromlast()
2   {
3       node *last;
4      last=H_linklist;
5      while(last->next->next!=NULL)
6      {
7        last=last->next;
8      }
9      node *temp;
10     temp=last->next;
11     last->next=NULL;
12  }
```

5.2 ลิงค์ลิสต์ทิศทางเดียว (ต่อ)

คำอธิบายการลบข้อมูลออกจากลิงค์ลิสต์ทางด้านปลาย



5.2 ลิงค์ลิสต์ทิศทางเดียว (ต่อ)

3) การลบโหนดในตำแหน่งที่ระบุ

ทำการค้นหาตำแหน่งโหนดที่ต้องการลบก่อน โดยใช้คีย์ในการค้นหาข้อมูลที่ต้องการลบ เมื่อพบตำแหน่งโหนดที่ต้องการ จึงทำการลบโหนดนั้น

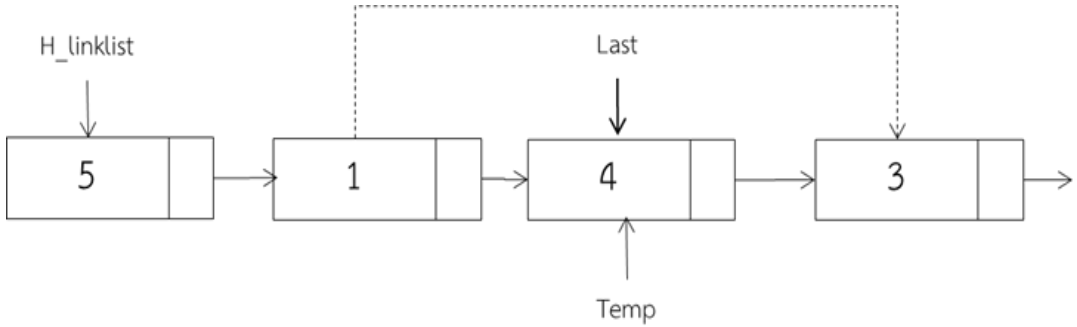
5.2 ลิงค์ลิสต์ทิศทางเดียว (ต่อ)

ตัวอย่างที่ 5.7 การลบข้อมูลออกจากลิงค์ลิสต์ในตำแหน่งที่ระบุ

```
1   void remove_node(int fnode)
2   {
3       node *temp=new node;
4       temp->next = NULL;
5       node*last= H_linklist;
6       if(fnode>1)
7       {
8       for(int j=2;j<fnode;j++) last=last->next;
9         temp->next = last->next;
10        last->next = temp->next->next;
11      }  else  {
12        temp = H_linklist->next;
13        H_linklist=temp;
14      }
15  }
```

5.2 ลิงค์ลิสต์ทิศทางเดียว (ต่อ)

คำอธิบายการลบข้อมูลออกจากลิงค์ลิสต์ในตำแหน่งที่ระบุ



5.2 ลิงค์ลิสต์ทิศทางเดียว (ต่อ)

5.2.4 การแสดงข้อมูลทั้งหมดภายในลิงค์ลิสต์

- การแสดงข้อมูลสามารถทำได้โดยใช้ฟังก์ชั่น while เพื่อแสดงข้อมูลสมาชิกหรือจำนวนอิลิเมนต์ทั้งหมดที่มีอยู่

- คำสั่ง while เป็นคำสั่งวนการทำงานแบบหนึ่ง โดย
  - กรณีเป็นจริงก็จะนำข้อมูลออกมาแสดงจนครบ
  - กรณีเงื่อนไขเป็นเท็จก็จะออกจากการทำงาน

- การตรวจสอบเงื่อนไขนี้จะใช้ในกรณีที่ยังมีข้อมูลในลิงค์ลิสต์อยู่
- ลักษณะเด่นของคำสั่ง while คือจะตรวจสอบเงื่อนไขก่อนทุกครั้งการทำงาน

5.2 ลิงค์ลิสต์ทิศทางเดียว (ต่อ)

ตัวอย่างที่ 5.8 การแสดงข้อมูลทั้งหมดภายในลิงค์ลิสต์

```
1   void show_list()
2   {
3           node *last;
4           last =H_linklist;
5           while(last!=NULL)
6           {
7                   cout<< last ->data<<endl;
8                   last = last ->next;
9           }
10  }
```

5.2 ลิงค์ลิสต์ทิศทางเดียว (ต่อ)

5.2.5 การนับจำนวนข้อมูลในลิงค์ลิสต์

การนับจำนวนสมาชิกในลิงค์ลิสต์สามารถใช้หลักการทำงานคล้ายกับการแสดงข้อมูล เนื่องจากสามารถทำได้โดยใช้ฟังก์ชั่น while หรือฟังก์ชั่น for เพื่อนับข้อมูลสมาชิกหรือจำนวนอิลิเมนต์ทั้งหมดที่มีอยู่ได้

5.2 ลิงค์ลิสต์ทิศทางเดียว (ต่อ)

ตัวอย่างที่ 5.9 การนับจำนวนข้อมูลในลิงค์ลิสต์

```
1   void countlist()
2   {
3           int count=0;
4           node * last;
5           for(last=H_linklist; last!=NULL; last = last ->next)
6           {
7                   count=count+1;
8           }
9           cout<<"\nTotal of elements ="<<count;
10  }
```

5.3 ลิงค์ลิสต์แบบสองทิศทาง

- โครงสร้างข้อมูลลิงค์ลิสต์แบบสองทิศทาง (doubly link-list)
- มีการพัฒนาโครงสร้างการทำงานมาจากลิงค์ลิสต์แบบทิศทางเดียว
- แต่ละโหนดจะมีลิงค์ชี้โหนดถัดไปที่โหนดปัจจุบันอ้างอิง
- คุณสมบัติที่เพิ่มขึ้น คือ แต่ละโหนดมีพอยน์เตอร์อ้างอิงโหนดที่ผ่านมา ทำให้แต่ละโหนดเดินหน้าและถอยหลังในลิสต์ได้
- การดำเนินการของลิงค์ลิสต์แบบสองทิศทางมีการดำเนินการคล้ายกับลิงค์ลิสต์แบบทิศทางเดียวกล่าวคือ สามารถดำเนินการเพิ่มข้อมูล ลบข้อมูล แทรกข้อมูล ได้
- สิ่งที่แตกต่าง คือ จะมีตัวแปรทำหน้าที่เชื่อมโยงลิงค์ลิสต์ก่อนหน้าที่โหนดเชื่อมโยงผ่านมา

❑ ลิงค์ลิสต์แบบสองทิศทาง

5.3 ลิงค์ลิสต์แบบสองทิศทาง(ต่อ)

5.3.1 สร้างลิสต์แบบสองทิศทาง

วิธีการสร้างลิงค์ลิสต์ นิยมใช้วิธีสร้างแบบไดนามิกส์ วิธีนี้เป็นวิธีที่ดีทำให้สามารถใช้พื้นที่ในการทำงานได้อย่างยืดหยุ่นเนื่องจากพื้นที่หน่วยความจำแบบไดนามิกส์จะมีพื้นที่ขนาดใหญ่สามารถรองรับการทำงานที่ต้องการใช้งานกับข้อมูลที่มีปริมาณมาก ๆ ได้ การสร้างลิสต์แบบสองทิศทางคล้ายกับลิสต์แบบทิศทางเดียว เพียงมีการเพิ่มคุณสมบัติส่วนอ้างอิงโหนดก่อนหน้าขึ้นมา

5.3 ลิงค์ลิสต์แบบสองทิศทาง(ต่อ)

ตัวอย่างที่ 5.10 การสร้างลิสต์แบบสองทิศทาง

```
1  struct node
2  {
3          int info;
4          struct node *next;
5          struct node *prev;
6  }*H_linklist;
```

5.3 ลิงค์ลิสต์แบบสองทิศทาง(ต่อ)

5.3.2 การเพิ่มข้อมูลเข้าไปในลิงค์ลิสต์แบบสองทิศทาง

การนำข้อมูลเข้าลิงค์ลิสต์แบบสองทิศทางบางครั้งเรียกว่าลิงค์ลิสต์สมมาตร เนื่องจากมีตัวชี้มาจากสองทิศทางทั้งด้านซ้ายและขวา ในการนำข้อมูลเพิ่มเข้าในลิงค์ลิสต์ลักษณะนี้จำเป็นต้องให้ความสำคัญกับลำดับในการชี้ตำแหน่งของ พอยต์เตอร์ ดังแสดงในตัวอย่างที่ 5.11 ดังต่อไปนี้

- การสร้างโหนดตำแหน่งเริ่มต้น
- การเพิ่มโหนดแรกในลิงค์ลิสต์แบบสองทิศทาง
- การเพิ่มโหนดใหม่ระหว่างโหนดที่มีข้อมูลอยู่แล้วในลิงค์ลิสต์แบบสองทิศทาง

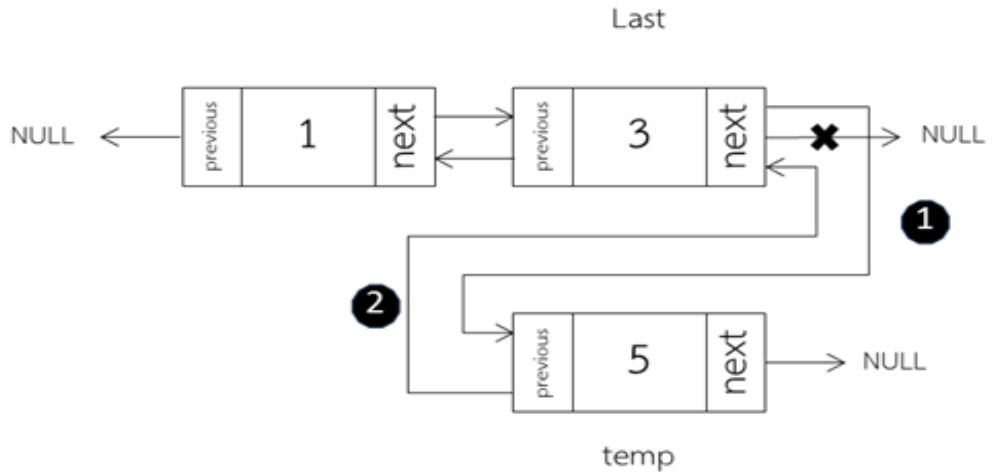
5.3 ลิงค์ลิสต์แบบสองทิศทาง(ต่อ)

ตัวอย่างที่ 5.11 การสร้างโหนดตำแหน่งเริ่มต้น

```
1   void double_Llist::create_node(int value)
2   {
3      struct node *last, *temp;
4      temp = new(struct node);
5      temp->info = value;
6      temp->next = NULL;
7      if (H_linklist == NULL)
8      {
9         temp->prev = NULL;
10        H_linklist = temp;
11     } else {
12        last = H_linklist;
13        while (last->next != NULL)   last = last->next;
14        last->next = temp;
15        temp->prev = last;
16     }
17  }
```

5.3 ลิงค์ลิสต์แบบสองทิศทาง(ต่อ)

การนำข้อมูลเข้าลิงค์ลิสต์แบบสองทิศทาง



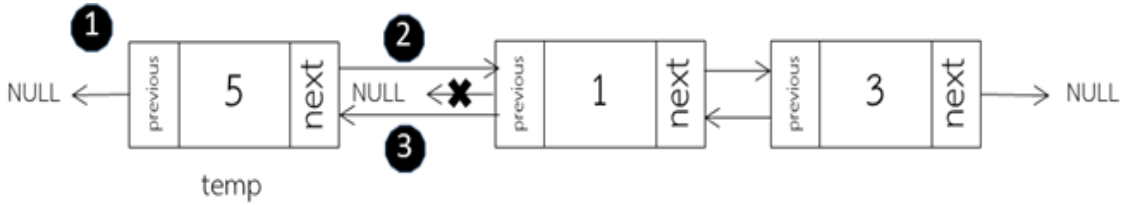
5.3 ลิงค์ลิสต์แบบสองทิศทาง(ต่อ)

ตัวอย่างที่ 5.12 การเพิ่มโหนดแรกในลิงค์ลิสต์แบบสองทิศทาง

```
1   void double_Llist::add_first_node(int value)
2   {
3      if (H_linklist == NULL)
4      {
5         cout<<"Create the list."<<endl;
6         return;
7      }
8      struct node *temp;
9      temp = new(struct node);
10     temp->prev = NULL;
11     temp->info = value;
12     temp->next = H_linklist;
13     H_linklist->prev = temp;
14     H_linklist = temp;
15     cout<<"Element Inserted"<<endl;
16  }
```

5.3 ลิงค์ลิสต์แบบสองทิศทาง(ต่อ)

คำอธิบายการเพิ่มโหนดแรกในลิงค์ลิสต์แบบสองทิศทาง



5.3 ลิงค์ลิสต์แบบสองทิศทาง(ต่อ)

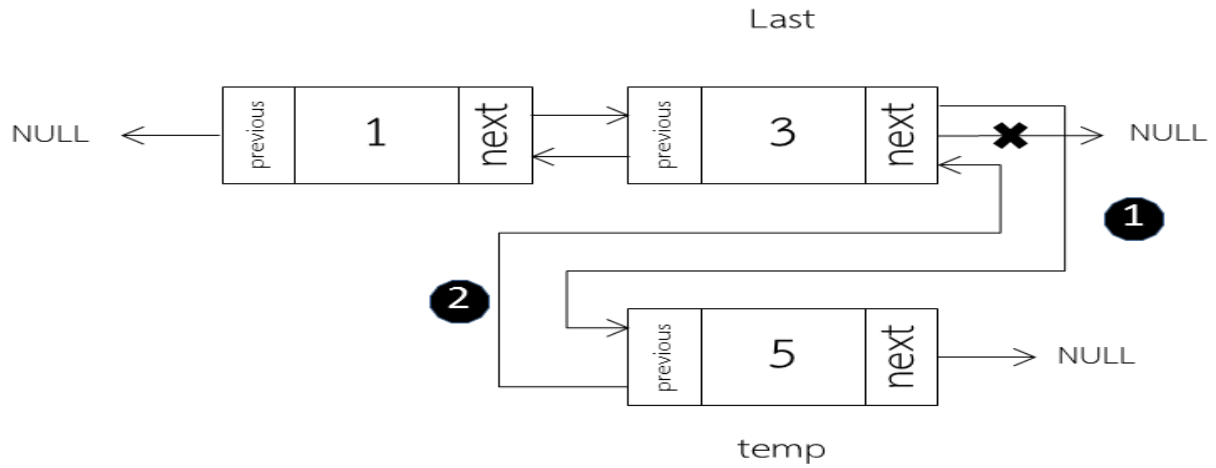
ตัวอย่างที่ 5.13 การเพิ่มโหนดใหม่ระหว่างโหนดที่มีข้อมูลอยู่แล้วในลิงค์ลิสต์แบบสองทิศทาง

```
1   void double_Llist::insert_node(int value, int pos)
2   {
3      if (H_linklist == NULL)
4      {
5         cout<<"Create the list."<<endl;
6         return;
7      }
8      struct node *tmp, *last;
9      int i;
10     last = H_linklist;
11     for (i = 0;i < pos - 1;i++)
12     {
13        last = last->next;
14        if (last == NULL)
15        {
16          cout<<"There are less than ";
17          cout<<pos<<" elements."<<endl;
18  return;
19        }
20     }
21     tmp = new(struct node);
22     tmp->info = value;
23     if (last->next == NULL)
24     {
25        last->next = tmp;
26        tmp->next = NULL;
27        tmp->prev = last;
28     } else {
29        tmp->next = last->next;
30        tmp->next->prev = tmp;
31        last->next = tmp;
32        tmp->prev = last;
33     }
34     cout<<"Element Inserted"<<endl;
35  }
```

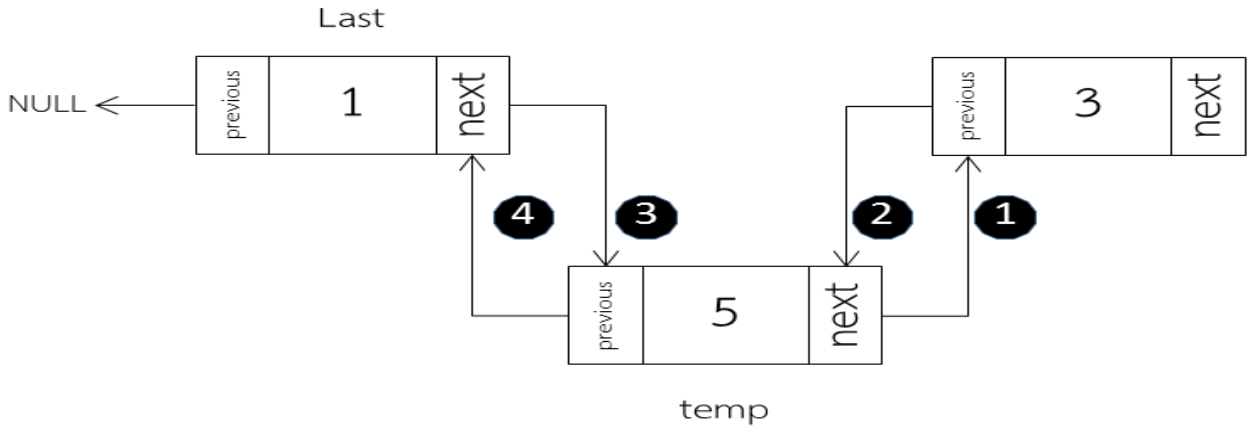
5.3 ลิงค์ลิสต์แบบสองทิศทาง(ต่อ)

คำอธิบายการเพิ่มโหนดใหม่ระหว่างโหนดที่มีข้อมูลอยู่แล้วในลิงค์ลิสต์แบบสองทิศทาง

1)



2)



5.3 ลิงค์ลิสต์แบบสองทิศทาง(ต่อ)

5.3.3 การลบข้อมูลที่ระบุในลิงค์ลิสต์แบบสองทิศทาง

ในการลบข้อมูลโหนดที่ไม่ต้องการออกจากหน่วยความจำหรือจากลิงลิสต์นั้น สามารถทำได้โดย

- ตรวจสอบข้อมูลในลิงค์ลิสต์ก่อน
  - ➢ ถ้าว่างจะไม่สามารถทำการลบข้อมูลได้
  - ➢ แต่ถ้ามีข้อมูลอยู่จึงสามารถทำการลบข้อมูลได้

5.3 ลิงค์ลิสต์แบบสองทิศทาง(ต่อ)

ตัวอย่างที่ 5.14 การลบข้อมูลที่ระบุในลิงค์ลิสต์แบบสองทิศทาง

```
1   void double_Llist::remove_node(int
    value)
2   {
3      struct node *tmp, *last;
4      if (H_linklist->info == value)
5      {
6         tmp = H_linklist;
7         H_linklist = H_linklist->next;
8         H_linklist->prev = NULL;
9         cout<<"Element Deleted"<<endl;
10        free(tmp);
11        return;
12     }
```

```
13     last = H_linklist;
14     while (last->next->next != NULL)
15     {
16        if (last->next->info == value)
17        {
18           tmp = last->next;
19           last->next = tmp->next;
20           tmp->next->prev = last;
21           cout<<"Element Deleted"<<endl;
22           free(tmp);
23           return;
24        }
25        last = last->next;
26     }
```

5.3 ลิงค์ลิสต์แบบสองทิศทาง(ต่อ)

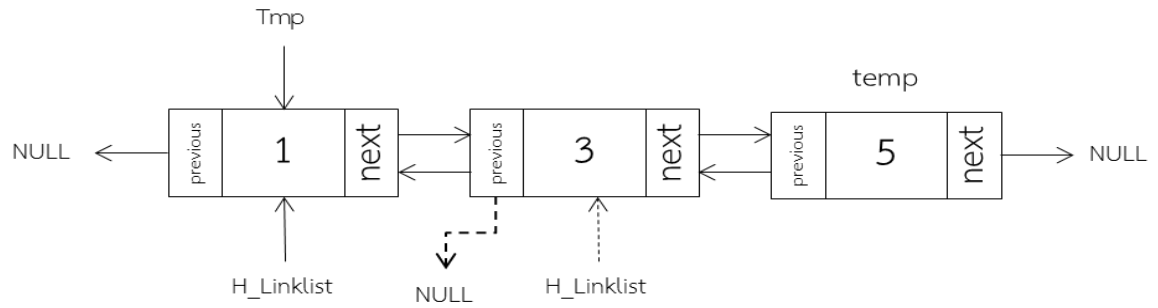
ตัวอย่างที่ 5.14 การลบข้อมูลที่ระบุในลิงค์ลิสต์แบบสองทิศทาง(ต่อ)

```
27      if (last->next->info == value)
28      {
29          tmp = last->next;
30          free(tmp);
31          last->next = NULL;
32          cout<<"Element Deleted"<<endl;
33          return;
34      }
35      cout<<"Element "<<value<<" not found"<<endl;
36  }
```

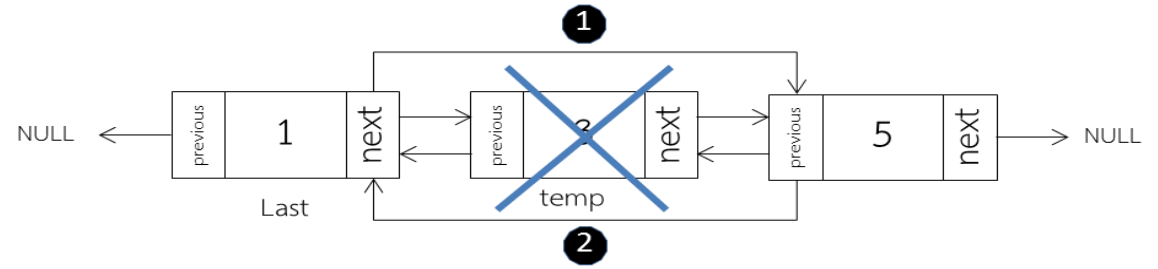
5.3 ลิงค์ลิสต์แบบสองทิศทาง(ต่อ)

คำอธิบายการลบข้อมูลที่ระบุในลิงค์ลิสต์แบบสองทิศทาง

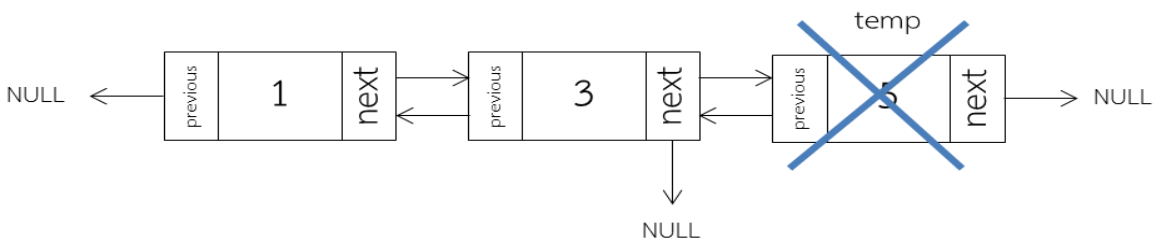
1)



2)



3)



5.3 ลิงค์ลิสต์แบบสองทิศทาง(ต่อ)

5.3.4 การแสดงข้อมูลทั้งหมดภายในลิงค์ลิสต์แบบสองทิศทาง

การแสดงข้อมูลภายในลิงค์ลิสต์แบบสองทิศทาง สามารถทำได้โดยใช้ฟังก์ชั่น while เพื่อแสดงข้อมูลสมาชิกหรือจำนวนอิลิเมนต์ทั้งหมดที่มีอยู่

❑ ตัวอย่างที่ 5.15 การแสดงข้อมูลทั้งหมดภายในลิงค์ลิสต์แบบสองทิศทาง

```
1   void double_Llist::show_list()
2   {
3      struct node *last;
4      if (H_linklist == NULL)
5      {
6         cout<<"List empty,nothing to
                  display"<<endl;
7         return;
8      }
9      last = H_linklist;
```

```
10  cout<<"The Doubly Link List is :"<<endl;
11     while (last != NULL)
12     {
13        cout<<last->info<<" <-> ";
14        last = last->next;
15     }
16     cout<<"NULL"<<endl;
17  }
```

5.3 ลิงค์ลิสต์แบบสองทิศทาง(ต่อ)

5.3.5 การนับจำนวนสมาชิกในลิงค์ลิสต์

การนับจำนวนสมาชิกสามารถทำได้โดยใช้ฟังก์ชั่น while เพื่อนับข้อมูลสมาชิกหรือจำนวนอิลิเมนต์ทั้งหมดที่มีอยู่ได้ โดยการนับจำนวนสมาชิกสามารถทำได้คล้ายกับลิงค์ลิสต์ที่ผ่านมา
ดังนี้

❑ ตัวอย่างที่ 5.16 การนับจำนวนสมาชิกในลิงค์ลิสต์

```
1   void double_Llist::countlist()
2   {
3   struct node *last = H_linklist;
4   int count = 0;
5   while (last != NULL)
6   {
7         last = last->next;
8         count++;
9   }
10  cout<<"Number of elements are: "<<count<<endl;
11  }
```

มหาวิทยาลัยราชภัฏนครปฐม