

Chapter 2

Array and Linked List (General Linear Lists)

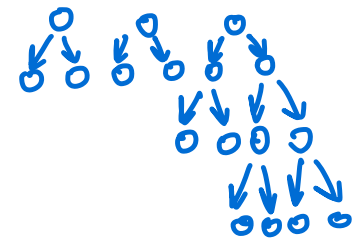
เก็บข้อมูล memory

↓
Data Structure

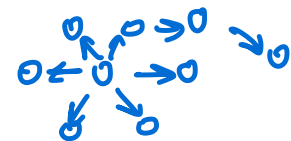
Linear List



ลำดับชั้น (ต้นไม้ : Tree)



Graph



จุดประสงค์

- เข้าใจแนวคิดและการทำงานเกี่ยวกับโครงสร้างข้อมูลแบบ Linear List $\circ \rightarrow \circ \rightarrow \circ$
- ประยุกต์แนวคิดโครงสร้างข้อมูลแบบ Linear List เข้ากับการเขียนโปรแกรมได้

ตัวอย่างการใช้ Linear List

- เกม

- High Scores

50	40	30	20	10
----	----	----	----	----

- Map

0	1	0
2	3	5
1	3	0

- ระบบปฏิบัติการ

- คิวการจัดการงาน

task 1 → task 2 → task 3

Linear List

- ในการสร้างโครงสร้างข้อมูลแบบลิสต์ มักสร้างโดย
 - อาร์เรย์ (*Array*)
 - ลิงค์ลิสต์ (*Linked List*) -> ลิงค์ลิสต์ คือ อะไร ?

Memory



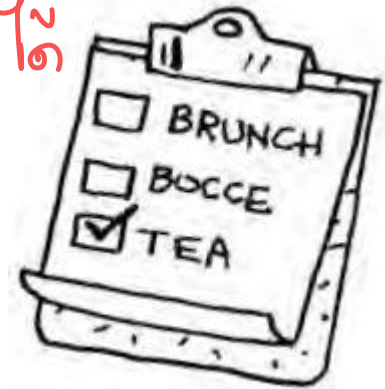
ADDRESS: fe0ffeeb

.
.
.
.

Array

pros: easy to find data

cons: - can't เพิ่มข้อมูลใหม่ที่ไม่ได้
- ลบข้อมูลใน ram ไม่ได้



YOUR
TO-DO
LIST

ex. ลบ 4 ที่

MEMORY IN USE
BY SOMEONE ELSE

BRU NCH	BO CCE	TEA	██████████
██████████	██████████		

FREE
MEMORY

→ ไม่ใส่ที่ว่าง ✗

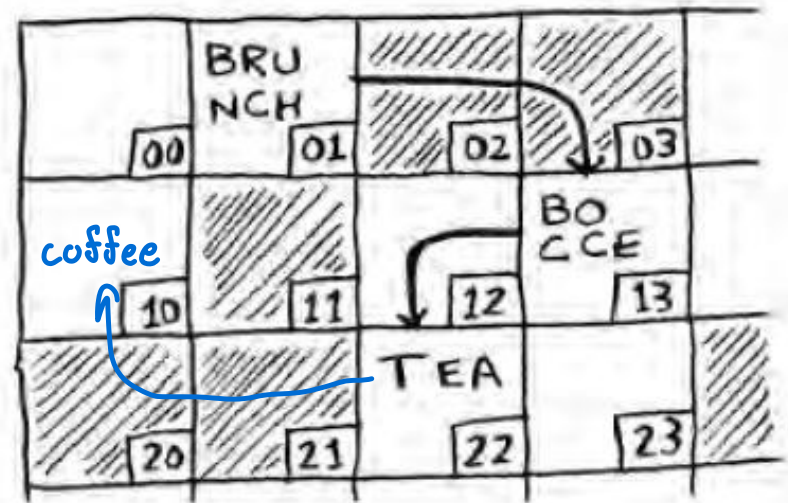
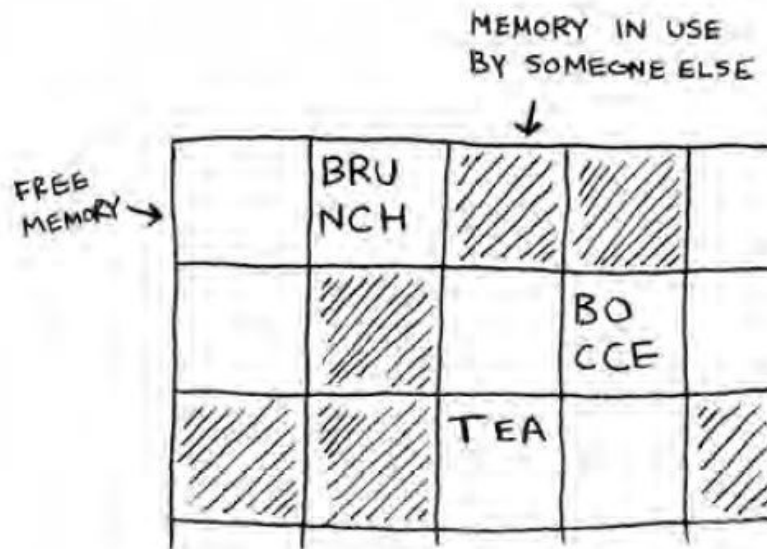
→ ใส่ทั้ง 2 ที่ ✗

→ ว่าง 4 ที่ ✓

Linked List

pros: เพิ่ม/ลบหาได้

cons: ใช้หน่วย data ว่างๆ array



Static & Dynamic Allocation

Array

- Static Memory Allocation
 - ในช่วงของการ Compile เครื่องได้จัดสรรหน่วยความจำ ให้กับตัวแปรแต่ละตัวล่วงหน้า ในขนาดที่คงที่
 - เมื่อมีการประมวลผล เนื้อที่เหล่านี้ไม่สามารถขยายหรือลดลงได้ (เช่น Array)

Static & Dynamic Allocation

- Dynamic Memory Allocation *linked list*
 - สามารถกำหนดตัวแปรไว้ก่อน แต่ยังไม่ต้องจัดสรรเนื้อที่หน่วยความจำจนกว่าจะถึงช่วง run time (ตอนประมวลผลโปรแกรม)
 - สามารถสร้างตัวแปรขึ้นได้ทุกครั้งที่ต้องการใช้ และสามารถทำลายลงเมื่อไม่ต้องการ
 - มีความยืดหยุ่น
 - ตัวแปรที่เป็นแบบ Dynamic นี้ได้แก่ Pointer และ Linked List

Array

- ตัวแปรที่เก็บชุดข้อมูลซึ่งมีชนิดเดียวกัน โดยทำการจองพื้นที่ในหน่วยความจำต่อเนื่องกัน
- ตัวอย่างการใช้ Array ในภาษา C
 - `int score[5] = {10, 20, 30, 40, 50}`
 - `score[0] -> 10`
 - `score[4] -> 50`

index					
	0	1	2	3	4
score	10	20	30	40	50

Linked List

- Singly Linked List

- ลิงค์ลิสต์ที่มีลิงค์ชี้ไปในทิศทางเดียว



- Doubly Linked List

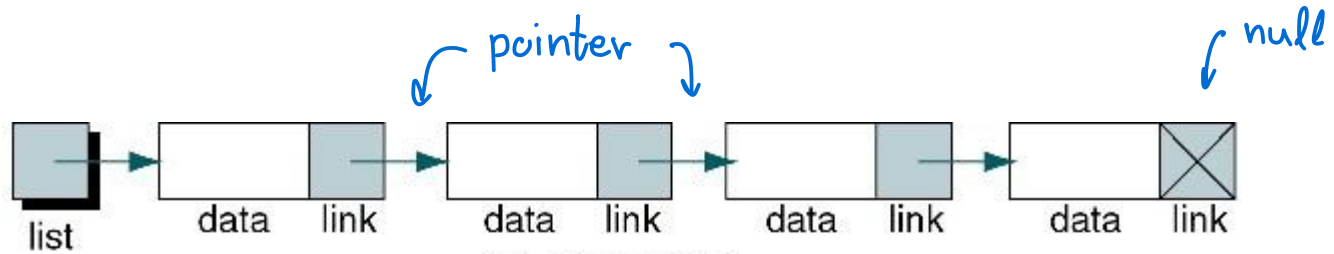
- ลิงค์ลิสต์ที่มีลิงค์ชี้ในสองทิศทาง (ไป-กลับ)



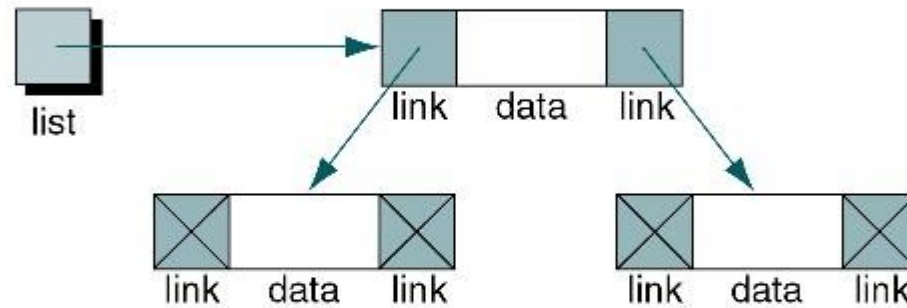
Linked List

- ลิงค์ลิสต์ : โครงสร้างข้อมูลที่ประกอบด้วยกลุ่มโหนดที่มีการเรียงลำดับ
- เรานำลิงค์ลิสต์มาใช้ในการสร้าง
 - *Linear structures*
 - *Non-linear structures*

Linked List (cont.)



(a) Linear list



(b) Non-linear list

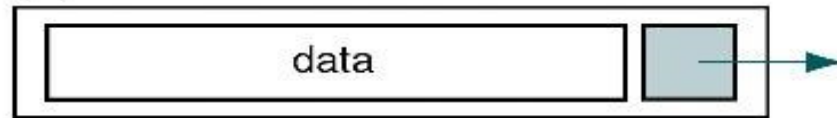


(c) Empty list

Nodes

- โหนด เป็นโครงสร้างที่ประกอบด้วย
 - ข้อมูล (*Data*)
 - ลิงค์ (*Link*)

(a) Node in a linear list

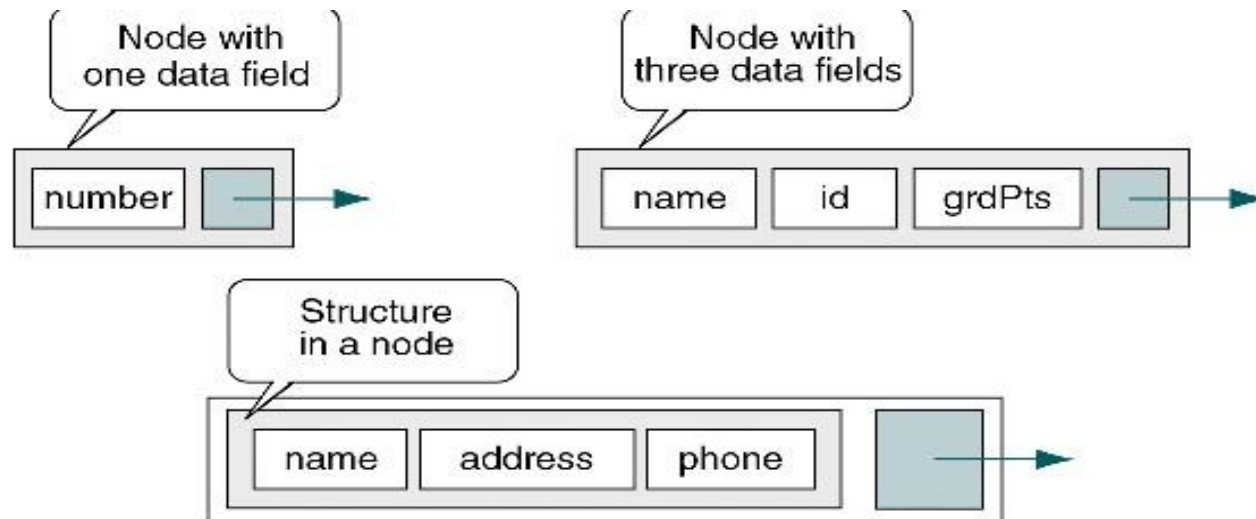


(b) Node in a non-linear list



Linked List Node Structures

- ข้อมูลของโหนด อาจจะประกอบด้วย
 - ข้อมูลชุดเดียว
 - ข้อมูลหลายชุด
 - ข้อมูลหนึ่งชุดที่ประกอบด้วยข้อมูลย่อยหลายชุด

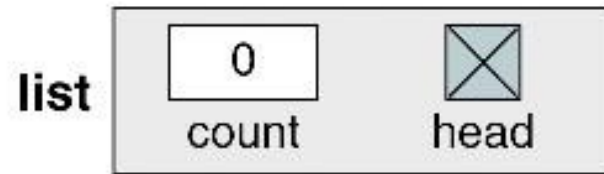


Links

- ลิงค์ของโหนด จะทำหน้าที่เชื่อมโหนดนั้นกับโหนดอื่นเข้าด้วยกัน
- เราสามารถนำพอยน์เตอร์ (pointer) มาช่วยในการสร้างลิงค์ได้
- พอยน์เตอร์ คือ ตัวชี้ไปยังข้อมูล
 - นั่นก็คือ ตัวเก็บตำแหน่ง (address) ของข้อมูลนั่นเอง

Create List

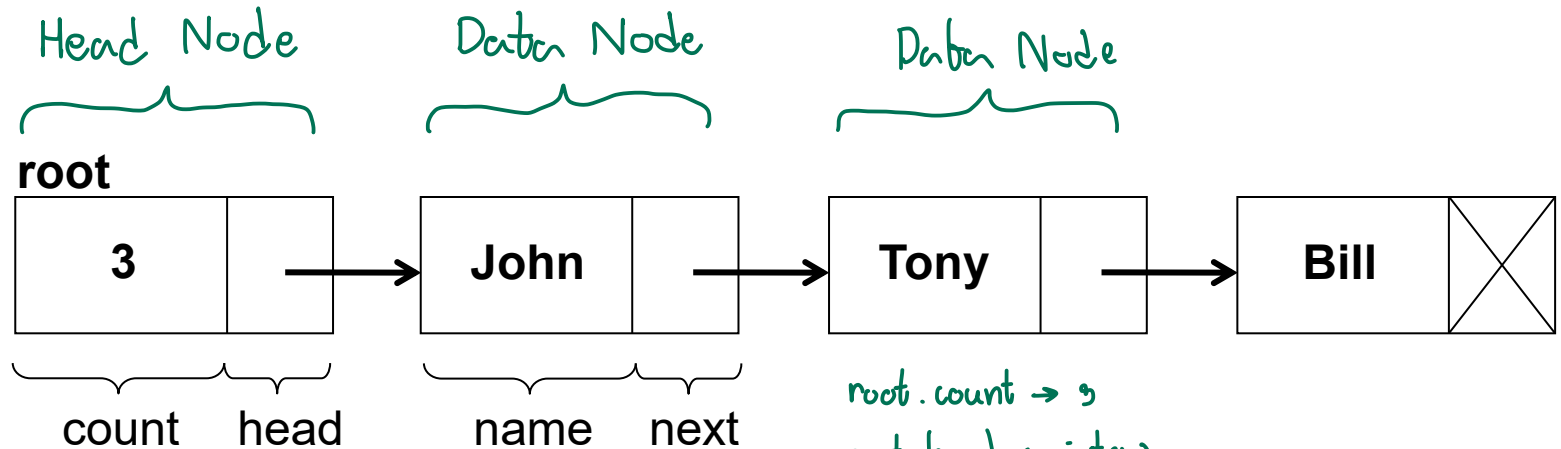
- Head Node : กำหนดข้อมูล
รายละเอียดเกี่ยวกับลิสต์



```
Head_node  
    int count  
    Data_node head  
end Head_node
```

```
Algorithm createList (list)  
Initializes metadata for list.  
    Pre    list is metadata structure passed by reference  
    Post   metadata initialized  
1 allocate (list)  
2 set list head to null  
3 set list count to 0  
end createList
```

Singly Linked List



□ เราจะสร้างลิงค์ลิสต์นี้ได้อย่างไร -> ต้องมีการกำหนดโครงสร้างของ
โหนดก่อน

□ ข้อมูล -> ชื่อ

□ ลิงค์ -> ชี้ไปยังโหนดถัดไป

root.count → 3
root.head (pointer)

root.head.name → John

root.head.next.name → Bill

root.head.next.next → Null

root.next.next.name → error
null

Node Structure

Person

string name

Person next

end Person

Head_node

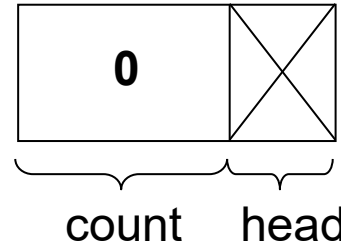
int count

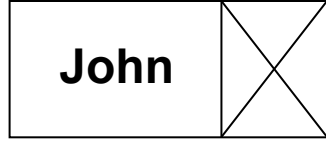
Person head

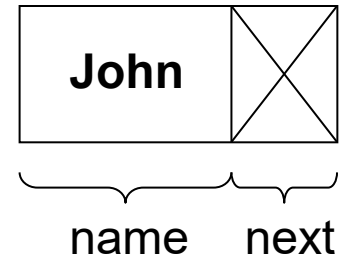
end Head_node

createList(root)

root



createDataNode('John', NULL)

pNew



createList(root)

createDataNode (d,p) :

pNew = allocate(Person)

name = d

next = p

return pNew

End createDataNode

Basic Operations

- Insertion
- Deletion
- Retrieval
- Traversal

Insertion

- เพิ่มโหนดที่เก็บชื่อ John เข้าไปในลิสต์

- Set root head to pNew

- Increase root.count

root.head = pNew

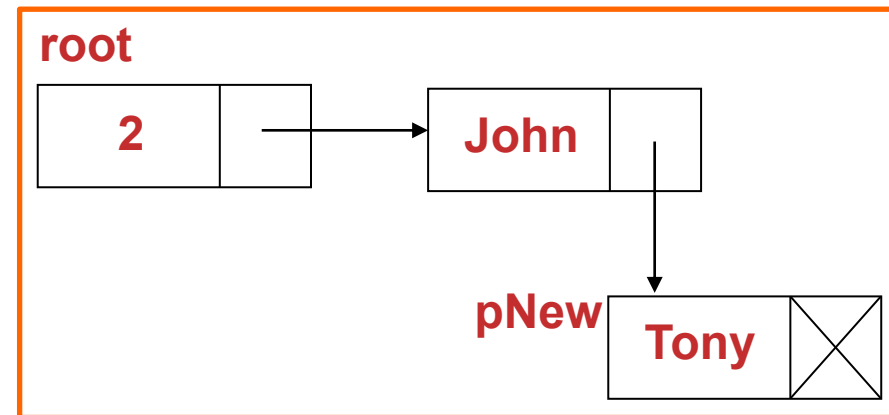
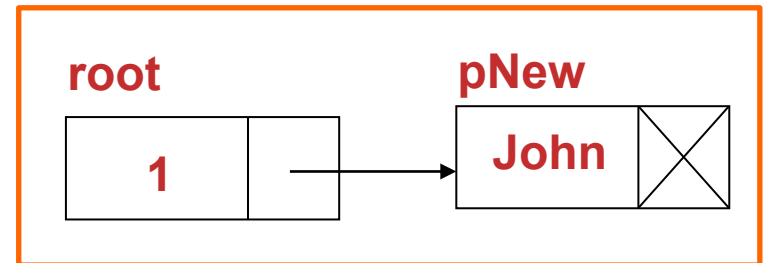
root.count++

- อยากจะเพิ่มโหนดที่เก็บชื่อ Tony ต่อท้ายเข้าไป ไม่ยากเลย

- createDataNode(Tony,null)

- Set next of last node to pNew

- Increase root count



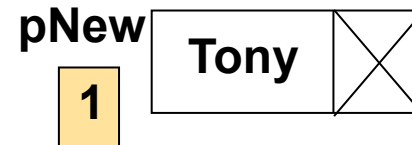
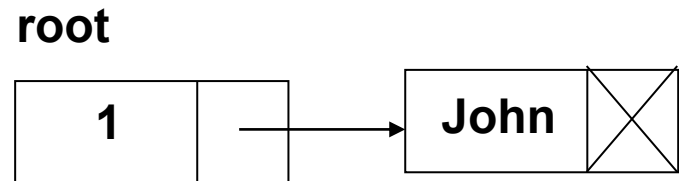
Insertion

- แทรกโหนดที่ส่วนหัวของลิสต์
- แทรกโหนดที่ส่วนท้ายของลิสต์
- แทรกโหนดตามการเรียงลำดับข้อมูล

Inserting Nodes at the Front of a List

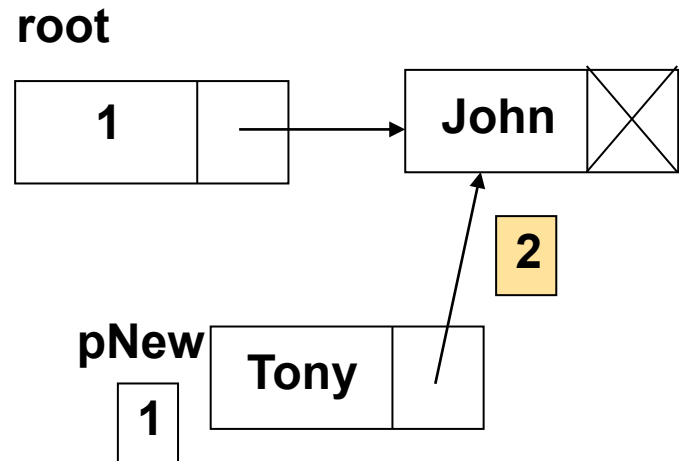
- ถ้าจะเพิ่มโหนดที่มีชื่อเป็น Tony ไว้ต้นลิสต์

- *createDataNode(Tony,null)*



Inserting Nodes at the Front of a List

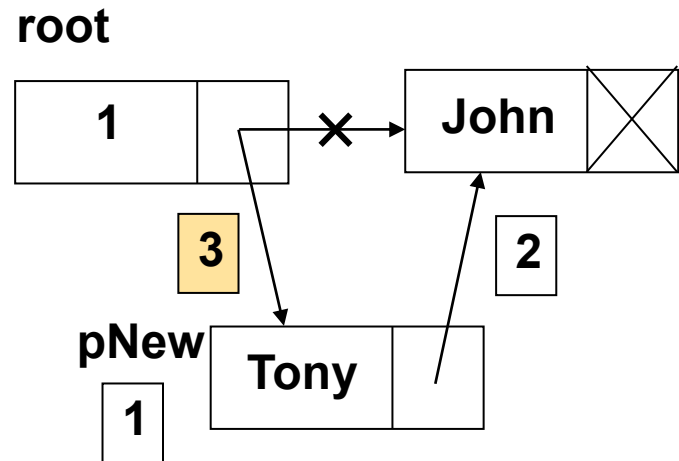
- ถ้าจะเพิ่มโหนดที่มีชื่อเป็น Tony ไว้ต้นลิสต์
 - createDataNode(Tony,null)
 - *pNew.next = root.head*



Inserting Nodes at the Front of a List

- ถ้าจะเพิ่มโหนดที่มีชื่อเป็น Tony ไว้ต้นลิสต์

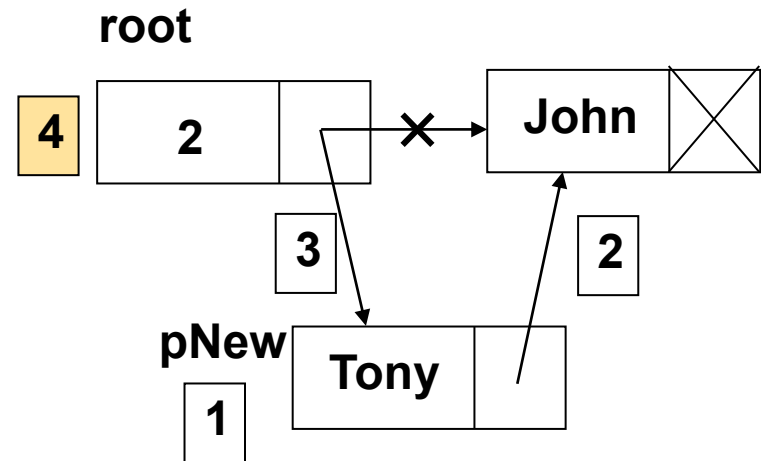
- createDataNode(Tony,null)
- pNew.next = root.head
- root.head = pNew



Inserting Nodes at the Front of a List

- ถ้าจะเพิ่มโหนดที่มีชื่อเป็น Tony ไว้ต้นลิสต์

- createDataNode(Tony,null)
- pNew.next = root.head
- root.head = pNew
- Increase root count



Traversing a Linked List

- เป็นการเข้าถึงโครงสร้างทีละโครงสร้างที่อยู่ในลิสต์นั้น
- ตัวอย่างการพิมพ์ค่าที่อยู่ใน Linked List

```
Algorithm printList(List root)
Data Node Person pos
    pos = root.head
    loop (pos != null)
        print pos.name
        pos = pos.next
    end loop
end printList
```

Inserting Nodes at the End of a List

- เหมือนกับตัวอย่างก่อนหน้านี้ ปัญหาคือ เรารู้ได้อย่างไรว่าโหนดไหนเป็นโหนดท้ายของลิสต์
 - ต้องท่อง (Traverse) เข้าไปในลิสต์จนถึงส่วนท้ายสุด

```
pNew = createDataNode(Tony,null)
```

```
If (root.head is null)
```

```
    root.head = pNew
```

```
else
```

```
    start = root.head
```

```
    loop (start.next != null)
```

```
        start = start.next
```

```
    end loop
```

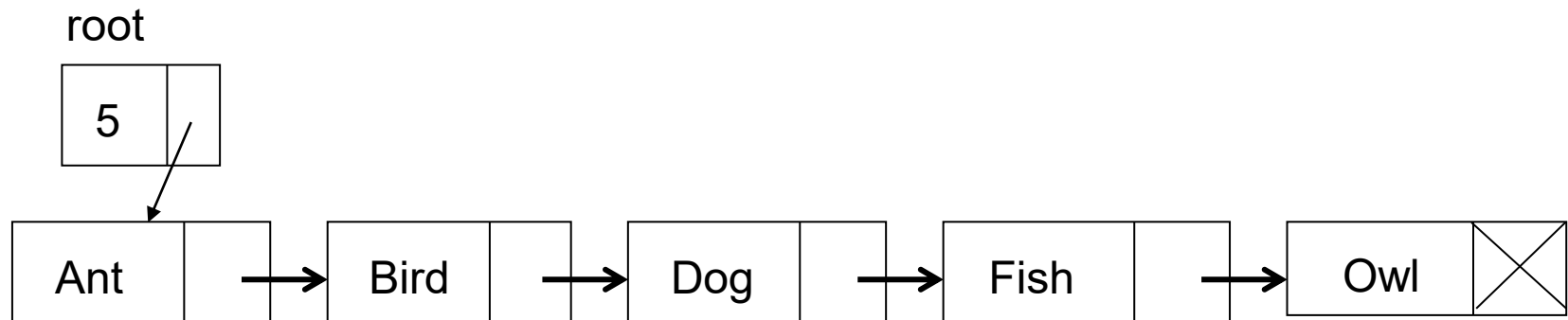
```
    start.next = pNew
```

```
end if
```

} Traverse
↓
ไปไล่ดูทุกโหนดของ linear list

Inserting Nodes into an Ordered List

- แทรกโหนดใหม่ โดยมีการเรียงลำดับข้อมูล
- ปัญหา คือ จะไปแทรกตรงไหน
 - ต้องท่องเข้าไปในลิสต์เรื่อยๆ ขณะเดียวกันก็ต้องเปรียบเทียบค่าข้อมูลไปด้วย



Inserting Nodes into an Ordered List

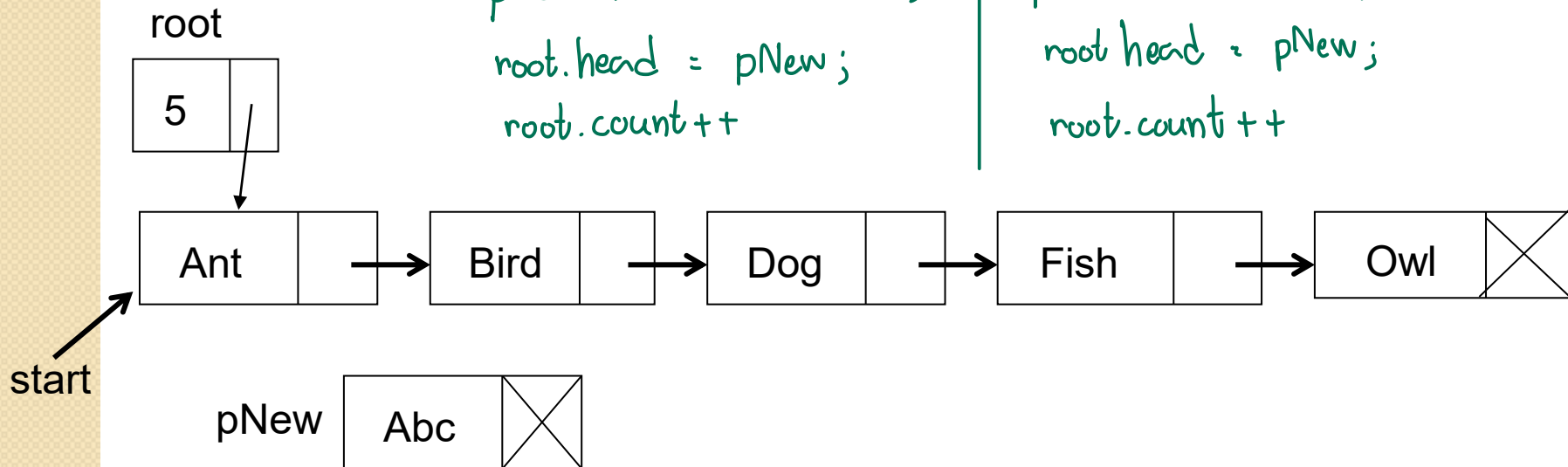
- กรณีแทรกโหนดด้านหน้าสุดของลิสต์

กรณี 1

```
pNew.next = root.head;  
root.head = pNew;  
root.count++
```

กรณี 2

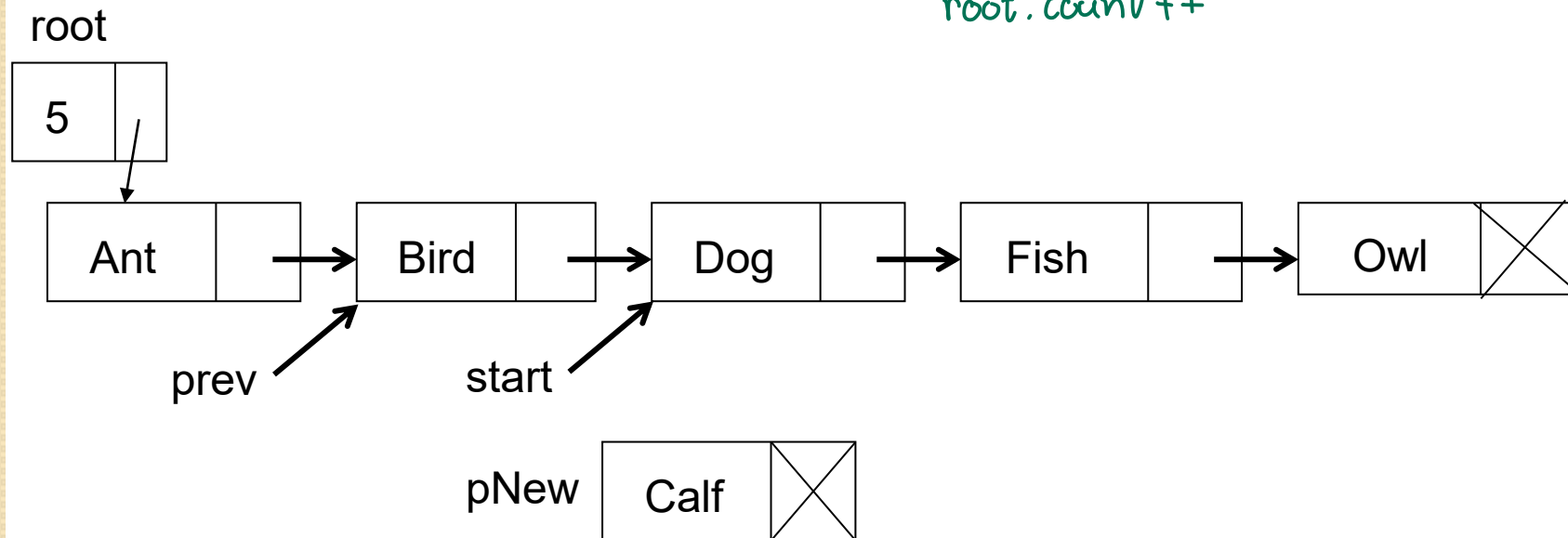
```
pNew.next = start;  
root.head = pNew;  
root.count++
```



Inserting Nodes into an Ordered List

- กรณีแทรกระหว่างโหนดในลิสต์

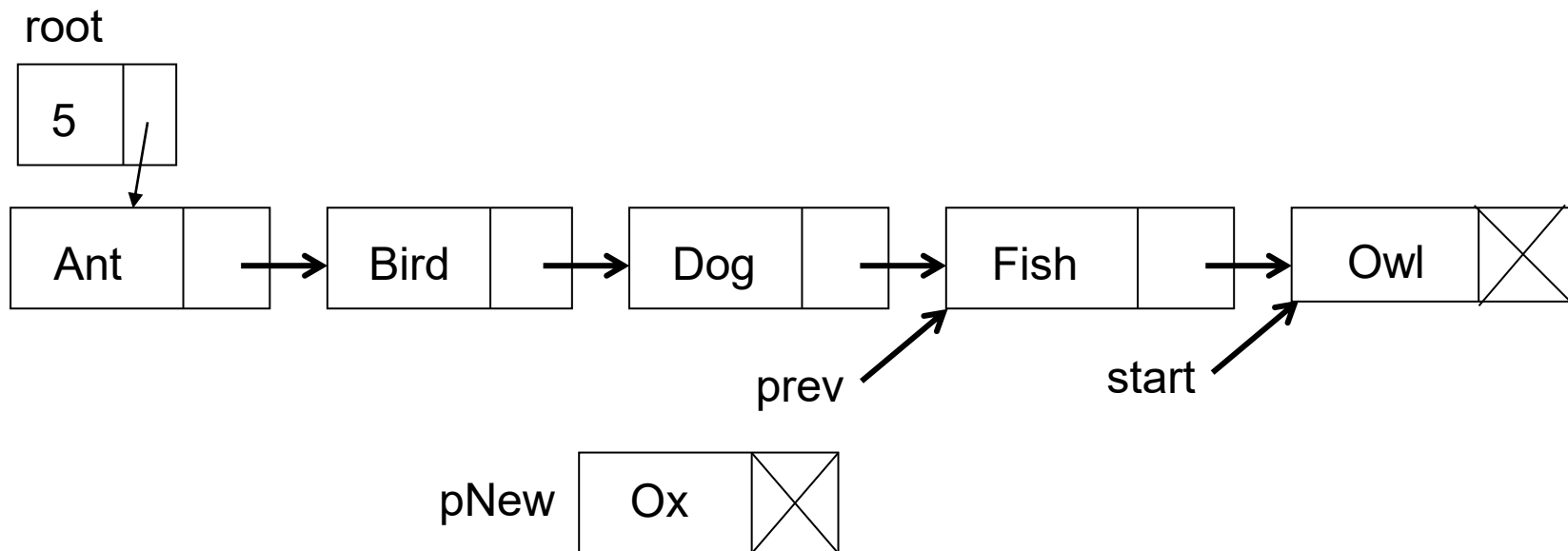
```
pNew.next = start;  
prev.next = pNew;  
root.count++
```



Inserting Nodes into an Ordered List

- กรณีที่แทรกปลายลิสต์

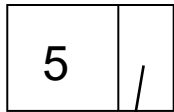
start.next = pNew
root.count++



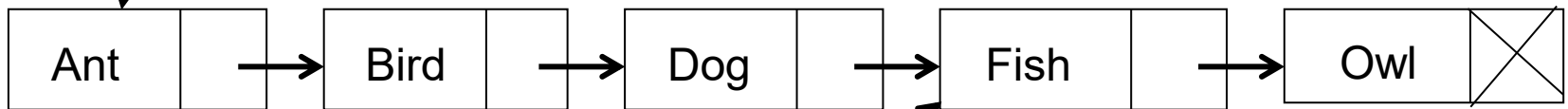
Delete

`prev.next = start.next;`
`delete start;` (deallocate: ~~memory~~ ^{free} memory)
`root.count--`

root



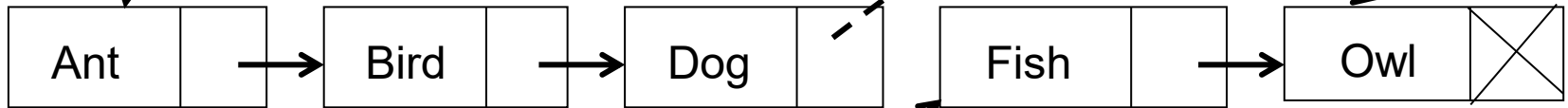
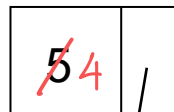
Before



prev

start

root



prev

start

After

Delete

- ในกรณีที่โหนดที่ต้องการลบเป็นโหนดแรกของลิสต์

- `prev.next = start.next` Error how to fix :

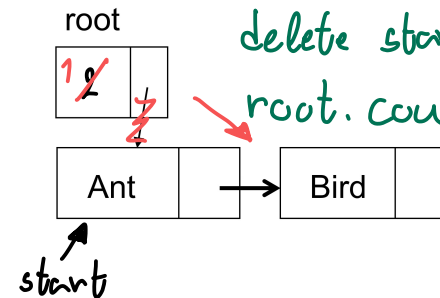
*root.head = start.next,
delete start;
root.count --*

if (`start == root.head`)

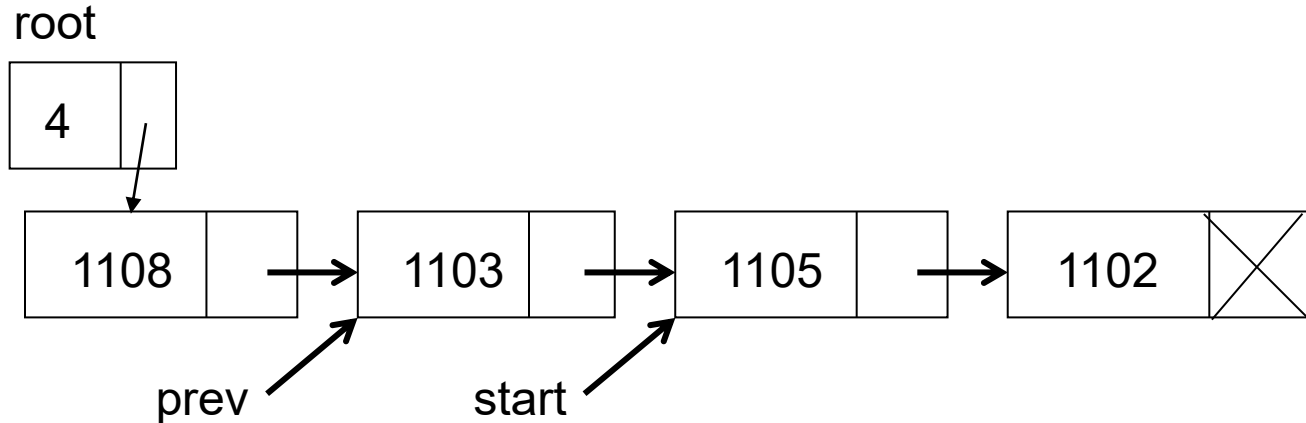
`root.head = start.next`

`delete start`

end if



Deleting Nodes from an Unordered List



- กรณีที่ต้องการลบโหนดที่อยู่หัวลิสต์

`root.head = root.head.next ; delete start;`

- กรณีที่ต้องการลบโหนดที่อยู่ระหว่างลิสต์

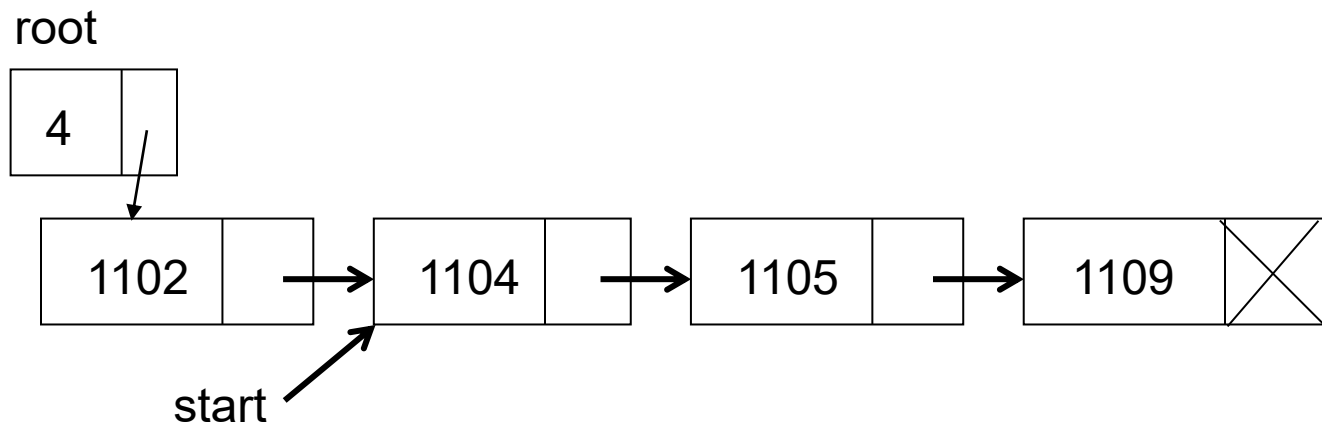
`prev.next = start.next; delete start;`

- กรณีที่ต้องการลบโหนดที่อยู่ท้ายลิสต์

`prev.next = NULL; delete start;`

Deleting Nodes from an Ordered List

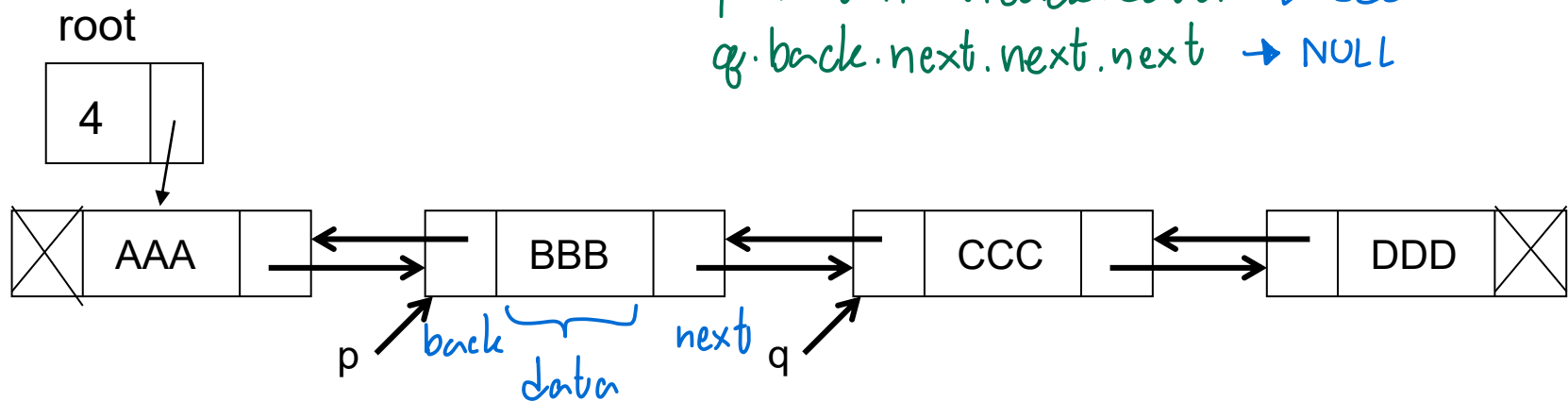
- ท่องลิสต์ไปเรื่อยๆ เมื่อพบโหนดที่ต้องการลบก็ลบไป
 - ปัญหา คือ ถ้าไม่มีโหนดที่ต้องการลบในลิสต์ จะต้องทำการท่องลิสต์จนหมด
 - สามารถทราบว่าโหนดนั้นไม่มีในลิสต์ แม้ว่าจะยังท่องลิสต์ไม่หมด
 - เช่น ถ้าต้องการลบโหนดที่เก็บเลข 1103 เมื่อท่องลิสต์จนไปถึงโหนดที่เก็บเลข 1104 แล้วยังไม่พบโหนดที่ต้องการ สรุปได้ว่า ไม่มีโหนดที่ต้องการลบ



Doubly Linked List

- เป็นลิงค์ลิสต์ที่มีลิงค์ 2 ทิศทาง คือ จากหน้าไปหลัง และ หลังไปหน้า

$p.next.next.back.data \rightarrow 'ccc'$
 $q.back.next.next.next \rightarrow NULL$



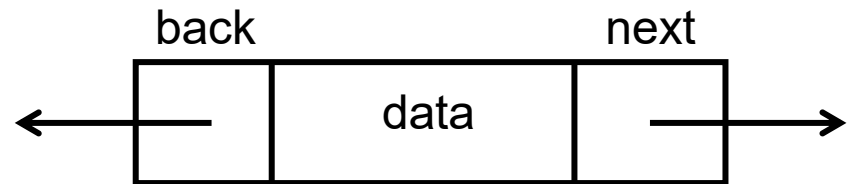
node

type data

node back

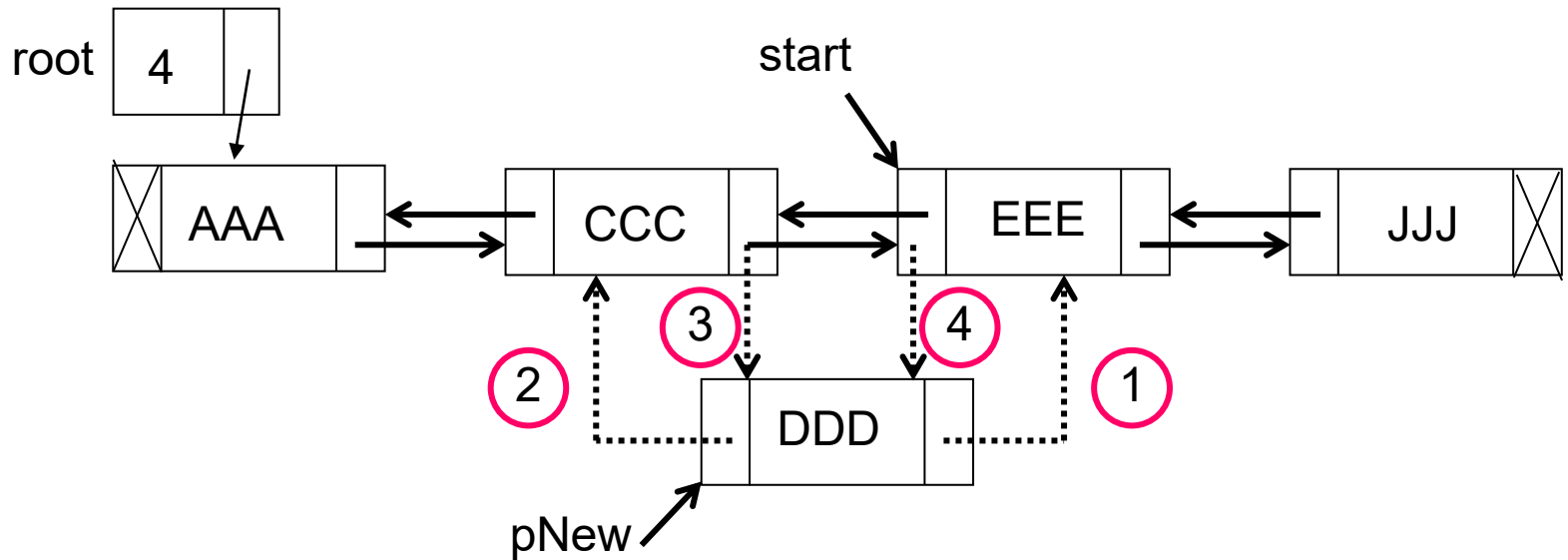
node next

end node



Inserting Nodes into a Doubly Linked List

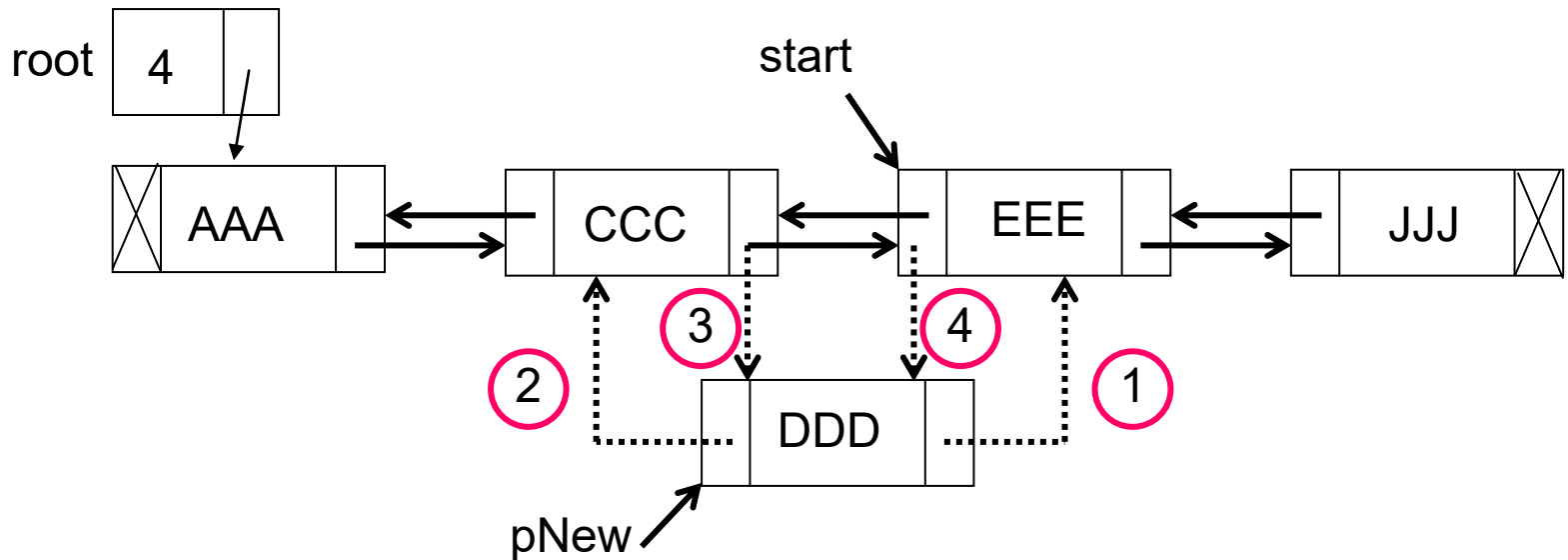
```
pNew.next = start;  
pNew.back = start.back;  
start.back.next = pNew;  
start.back = pNew;  
root.count++
```



- ต้องการแทรกโหนด pNew ไว้หน้า start

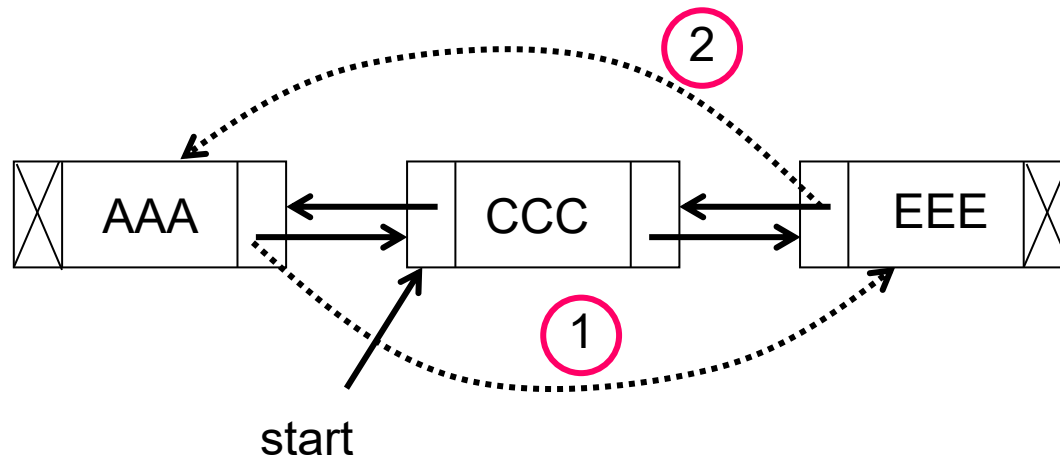
Inserting Nodes

into a Doubly Linked List



- ต้องการแทรกโหนด pNew ไว้หน้า start
pNew.next = start
pNew.back = start.back
start.back.next = pNew
start.back = pNew

Deleting Nodes from a Doubly Linked List



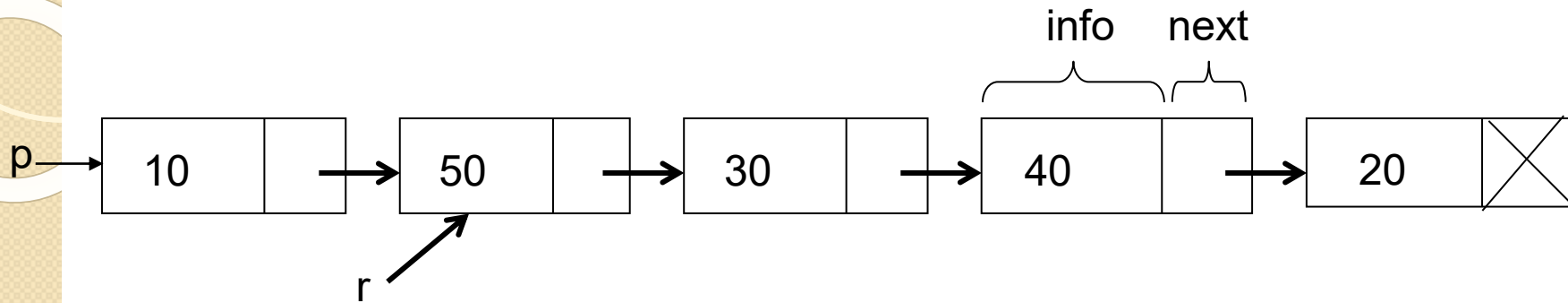
- ต้องการลบโหนด start

```
start.back.next = start.next;  
start.next back = start back;  
delete start;
```


List in Python

- ตัวอย่างการใช้ List ใน Python
 - `my_list = [10, True, 2.5, 50, False]`
 - `my_list[0] -> 10`
 - `my_list.insert(0, 15.5)`
 - `del my_list[1]`
- ลิสต์ในไพธอน ไม่ใช่ลิงค์ลิสต์ แต่เป็นอาร์เรย์ของพอยน์เตอร์ (แอดเดรส) ที่ชี้ไปที่วัตถุต่างๆ

Quiz 1

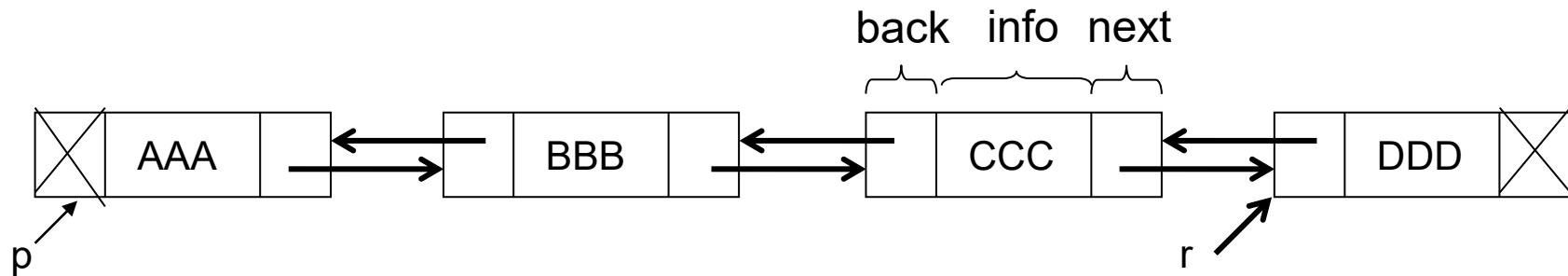


- จงประมวลผลคำสั่งต่อไปนี้ พร้อมวาดภาพผลลัพธ์ของแต่ละคำสั่ง
 - $r.next.next.info = 23$
 - $p.next.next = r.next.next$
 - $p.next.info = 68$
 - $r.next.next.next = p.next$
 - $p.next = null$

Quiz 2

- จากผลลัพธ์ในข้อก่อนหน้านี้ จงหาผลลัพธ์ของ
loop (r.next != null)
 print r.info
 r = r.next
end loop

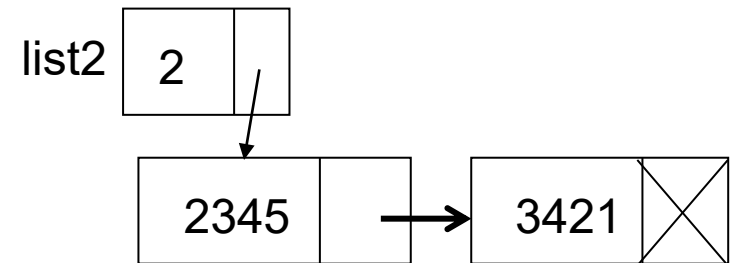
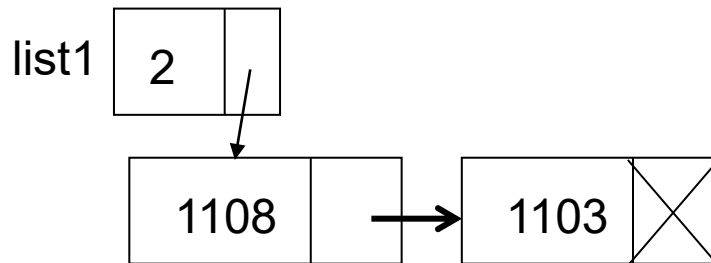
Quiz 3



- จงประมวลผลคำสั่งต่อไปนี้ พร้อมวาดภาพผลลัพธ์ของแต่ละคำสั่ง
 - $r.back.back = p$
 - $r = r.back$
 - $r.back.next.info = 'XXX'$
 - $r.back = p.back$

Quiz 4

- จงเขียนอัลกอริทึม append ที่ใช้สำหรับต่อลิสต์ 2 ลิสต์เข้าด้วยกัน
- เช่น `append(list1, list2) -> list1` และ `list2` มีขนาดและข้อมูลเป็นเท่าไรก็ได้



- ผลลัพธ์

