Lecture 7

เอกสารประกอบการบรรยาย
Data Structure

เรื่อง Linked List

Linked List

เนื้อหา

- โครงสร้างข้อมูลแบบลิงค์ลิสต์
- กระบวนงานและฟังก์ชั่นที่ใช้ดำเนินงานพื้นฐาน
- การสร้างลิงค์ลิสต์
- ลิงค์ลิสต์แบบซับซ้อน

จุดประสงค์การเรียนรู้

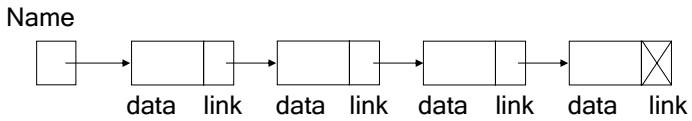
1. เพื่อให้นักศึกษาทราบโครงสร้างข้อมูลแบบลิงค์ลิสต์
2. เพื่อให้ทราบกระบวนการทำงานและฟังก์ชันที่ใช้ดำเนินงานพื้นฐานของลิงค์ลิสต์
3. เพื่อให้นักศึกษาทราบวิธีการสร้างลิงค์ลิสต์
4. เพื่อให้นักศึกษาทราบการทำงานของลิงค์ลิสต์แบบซับซ้อน

Linked List (Cont.)

ลิงค์ลิสต์ (Linked List) เป็นวิธีการเก็บข้อมูลอย่างต่อเนื่องของอิลิเมนต์ต่าง ๆ โดยมีพอยเตอร์เป็นตัวเชื่อมต่อ

แต่ละอิลิเมนท์ เรียกว่าโนด (Node) ซึ่งในแต่ละโนดจะประกอบไปด้วย 2 ส่วน คือ Data จะเก็บข้อมูลของอิลิเมนท์ และ ส่วนที่สอง คือ Link Field จะทำหน้าที่เก็บตำแหน่งของโนดต่อไปในลิสต์

Linked List (Cont.)

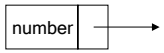


A linked list with a head pointer

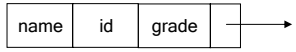
Name

An empty linked list

Linked List (Cont.)



A node with 1 data field



A node with 3 data fields

ในส่วนของ data อาจจะเป็นรายการเดี่ยว หรือเป็นเรคคอร์ดก็ได้

ในส่วนของ link จะเป็นส่วนที่เก็บตำแหน่งของโหนดถัดไปในโหนดสุดท้ายจะเก็บค่า Null ซึ่งไม่ได้ชี้ไปยังตำแหน่งใด ๆ เป็นตัวบอกการสิ้นสุดของลิสต์

Linked List (Cont.)

ในลิงค์ลิสต์จะมีตัวแปรสำหรับชี้ตำแหน่งลิสต์ (List pointer variable) ซึ่งเป็นที่เก็บตำแหน่งเริ่มต้นของลิสต์ ซึ่งก็คือ
โหนดแรกของลิสต์นั่นเอง ถ้าลิสต์ไม่มีข้อมูล ข้อมูลในโหนดแรกของลิสต์จะเป็น Null

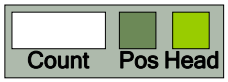
Linked List (Cont.)

โครงสร้างข้อมูลแบบลิงค์ลิสต์

โครงสร้างข้อมูลแบบลิงค์ลิสต์จะแบ่งเป็น 2 ส่วน คือ

1. Head Structure จะประกอบไปด้วย 3 ส่วน ได้แก่ จำนวนโหนดในลิสต์ (Count) พอยเตอร์ที่ชี้ไปยังโหนดที่เข้าถึง (Pos) และพอยเตอร์ที่ชี้ไปยังโหนดข้อมูลแรกของลิสต์ (Head)
2. Data Node Structure จะประกอบไปด้วยข้อมูล (Data) และพอยเตอร์ที่ชี้ไปยังข้อมูลตัวถัดไป

Linked List (Cont.)



Head Structure

```
list
   count    <integer>
   pos      <pointer>
   head     <pointer>
end list
```

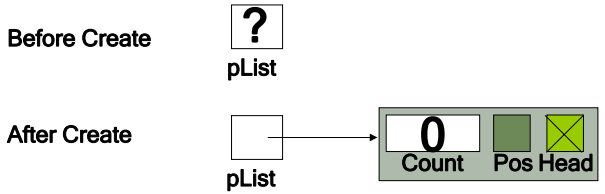
Data Node Structure

```
node
   data     <data type>
   link     <pointer>
end node
```


Linked List (Cont.)

กระบวนงานและฟังก์ชั่นที่ใช้ดำเนินงานพื้นฐาน

1. กระบวนงาน Create List
   - หน้าที่ สร้างลิสต์ว่าง
   - ผลลัพธ์ ลิสต์ว่าง



Linked List (Cont.)

Algorithm CreateList
- **Pre** Nothing
- **Post** Head node allocated or error returned
- **Return** Head node pointer or null if memory overflow

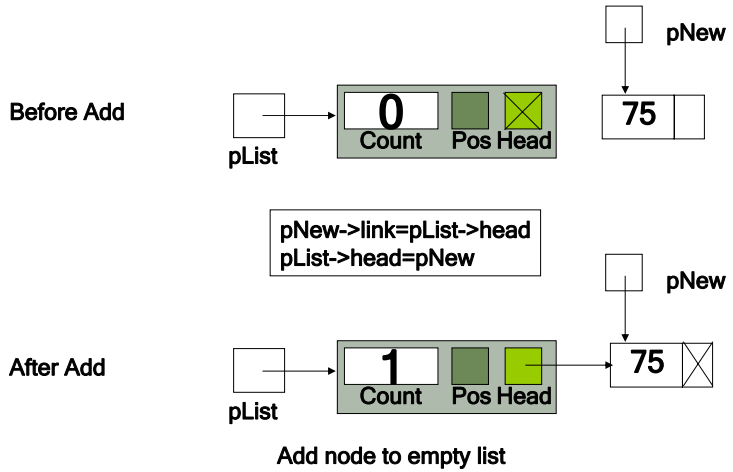
1. if (memory available)
   1 allocate (Pnew)
   2 pNew->head = null pointer
   3 pNew->count = 0
2. else
   1 pNew = null poiter
3. return pNew

End CreateList

Linked List (Cont.)

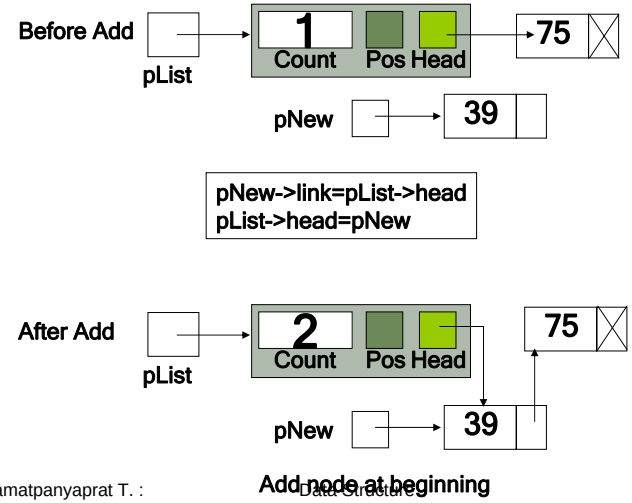
2. กระบวนงาน Insert Node
   - หน้าที่เพิ่มข้อมูลลงไปในลิสต์บริเวณตำแหน่งที่ต้องการ
   - ข้อมูลนำเข้า ลิสต์ ข้อมูล และตำแหน่ง
   - ผลลัพธ์ ลิสต์ที่มีการเปลี่ยนแปลง

Linked List (Cont.)



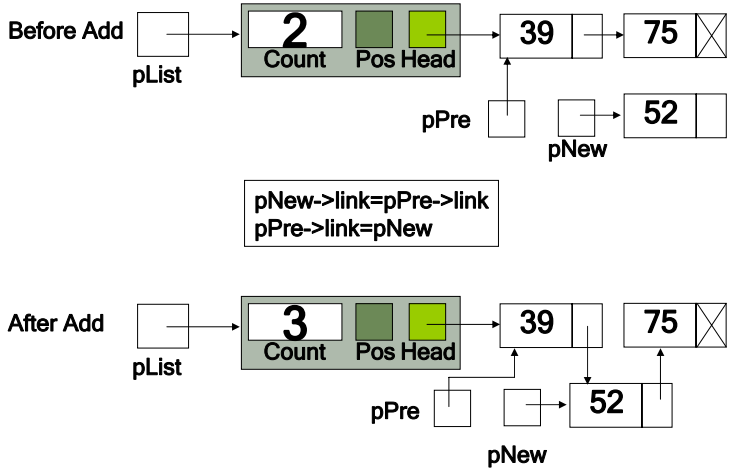
Add node to empty list

Linked List (Cont.)



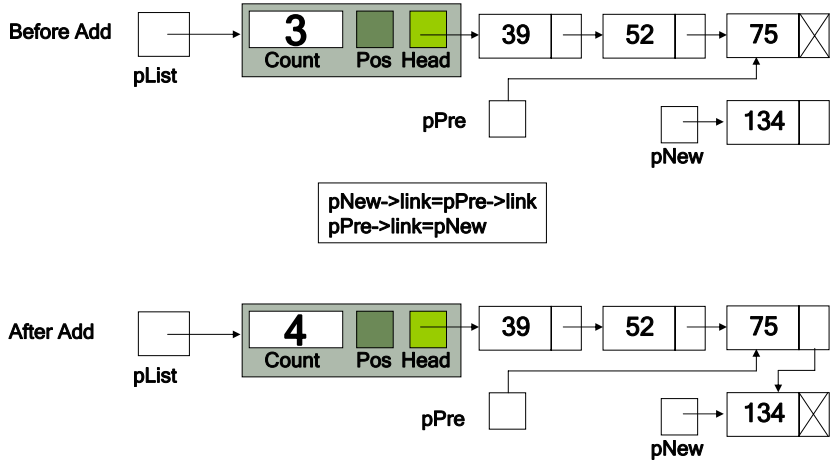
Add node at beginning

Linked List (Cont.)



Add node in middle

Linked List (Cont.)



Add node at end

Linked List (Cont.)

Algorithm insertNode (val pList <head pointer>,
val pPre <node pointer>,
val dataIn <dataType>)

Pre pList is a pointer to a valid list head structure
pPre is a pointer to data's logical predecessor
dataIn contains data to be inserted

Post data have been insert in sequence

Return true if successful, false if memory overflow

1. Allocate (pNew)
2. If (memory overflow)
   1 return false
3. pNew->data = dataIn

Linked List (Cont.)

3. If (pPre null)
   1 pNew->link = pList->head
   2 pList->head = pNew
5. else
   1 pNew->link = pPre->link
   2 pPre->link = pNew
6. pList->count = pList->count +1
7. Return true

End insertNode

Linked List (Cont.)

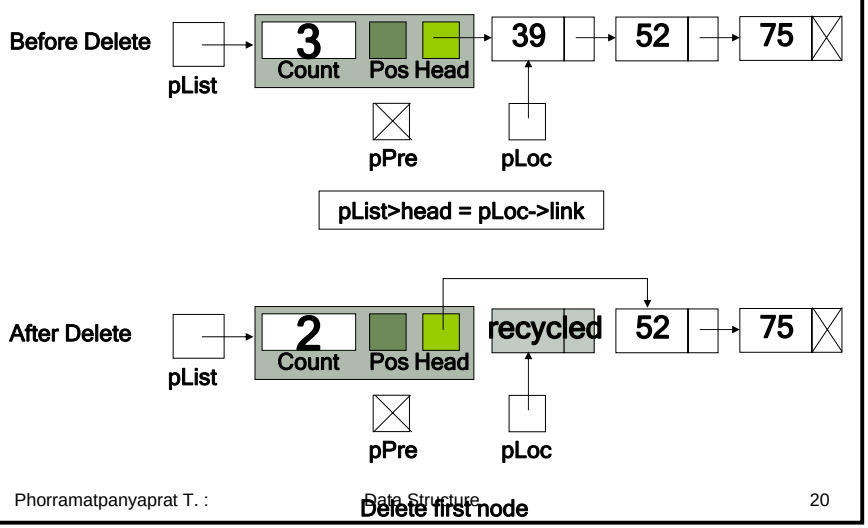
3. กระบวนงาน Delete Node

หน้าที่ ลบสมาชิกในลิสต์บริเวณตำแหน่งที่ต้องการ

ข้อมูลนำเข้า ข้อมูลและตำแหน่ง

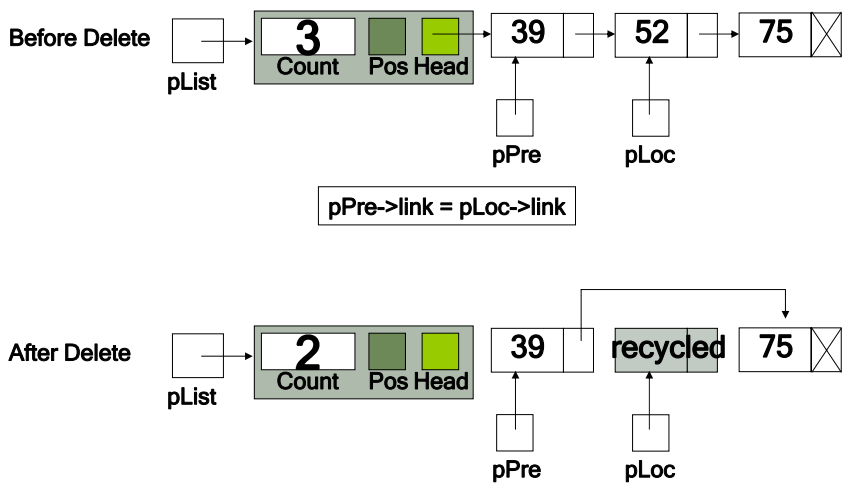
ผลลัพธ์ ลิสต์ที่มีการเปลี่ยนแปลง

Linked List (Cont.)



Delete first node

Linked List (Cont.)



Delete general case

Linked List (Cont.)

Algorithm deleteNode (val pList <head pointer>,
val pPre <node pointer>,
val pLoc <node pointer>,
ref dataOut <dataType>)

Pre pList is a pointer to a valid list head structure
pPre is a pointer to predecessor node
pLoc is a pointer to node to be deleted
dataOut is address to pass deleted data to calling module

Post data have been delete and return to caller

Linked List (Cont.)

1. dataOut = pLoc->data
2. if (pPre null)
   1 pList->head = pLoc->link
3. Else
   1 pPre->link = pLoc->link
4. pList->count = pList->count – 1
5. Release (pLoc)
6. Return

End deleteNode

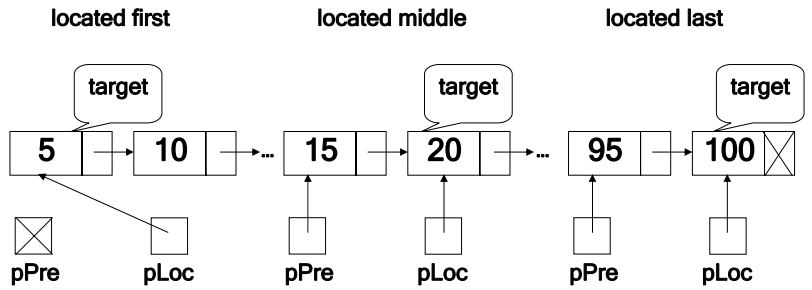
Linked List (Cont.)

4. กระบวนงาน Search list
   หน้าที่ ค้นหาข้อมูลในลิสต์ที่ต้องการ
   ข้อมูลนำเข้าลิสต์
   ผลลัพธ์ ค่าจริงถ้าพบข้อมูล ค่าเท็จถ้าไม่พบข้อมูล

Linked List (Cont.)

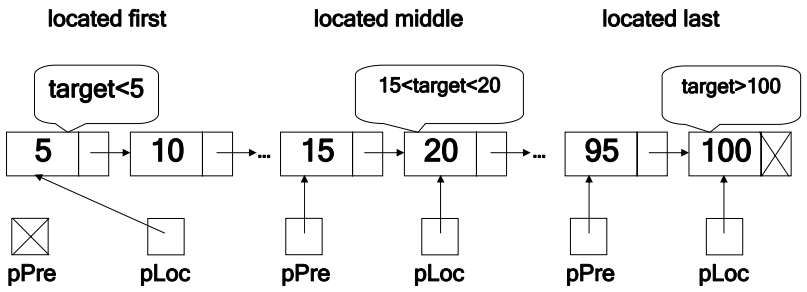
Condition	pPre	pLoc	return
target < first node	null	first node	false
target = frist node	null	first node	true
first<target<last	largest node<target	first node> target	false
target = middle node	node's predecessor	equal node	true
target = last node	last's predecessor	last node	true
target>last node	last node	Null	false

Linked List (Cont.)



Successful searches (return true)

Linked List (Cont.)



Unsuccessful searches (return false)

Linked List (Cont.)

Algorithm searchList (val pList <head pointer>,
val pPre <node pointer>,
ref pLoc <node pointer>,
val target <key type>)

Pre pList is a pointer to a valid list head structure
pPre is a pointer to receive predecessor
pLoc is a pointer to receive current node
target is the key being sought

Post pLoc points to first node with equal or greater key or null if target > key of last node
pPre points to largest node smaller than key or null if target< key of first node

Return true if found, false if not found

Linked List (Cont.)

1. pPre = null
2. pLoc = pList->head
3. loop (pLoc not null AND target > pLoc->data.key)
   1 pPre = pLoc
   2 pLoc = pLoc->link
4. If (pLoc null)
   1 found = false
5. else
   1 if (target equal pLoc->data.key)
      1 found = true
   2 else
      1 found = false
6. Return found

Return searchList

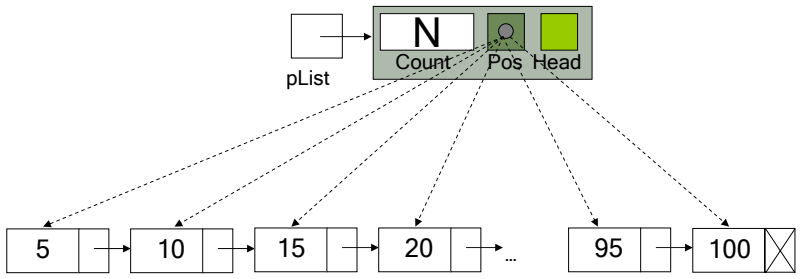
Linked List (Cont.)

5. กระบวนงาน Traverse

หน้าที่ ท่องไปในลิสต์เพื่อเข้าถึงและประมวลผลข้อมูลนำเข้าลิสต์

ผลลัพธ์ ขึ้นกับการประมวลผล เช่น เปลี่ยนแปลงค่าใน node , รวมฟิลด์ในลิสต์ , คำนวณค่าเฉลี่ยของฟิลด์ เป็นต้น

Linked List (Cont.)



Linked list traversal

Linked List (Cont.)

Algorithm traverse (val pList<head pointer>,val fromWhere <boolean>, ref dataPtr<pointer or dataType>)

Pre pList is a pointer to a valid list head structure
fromWhere is 0 to start at the first element
dataPtr is address of a pointer to data

Post address placed in dataPtr and return true or if end of list, return false

Return true if next element located, false if end of list

1. if (fromWhere is 0)
   1 if (pList->count is zero)
      1 success = false
   2 else
      1 pList->pos = pList->head
      2 dataPtr = address (pList->pos->data)
      3 success = true

Linked List (Cont.)

2. else
   1 if (pList->pos->link null)
      1 success = false
   2 else
      1 pList->pos = pList->pos->link
      2 dataPtr = address (pList->pos->data)
      3 success = true
3. return success

End traverse

Linked List (Cont.)

6. กระบวนงาน Retrieve Node

หน้าที่ หาตำแหน่งข้อมูลจากลิสต์

ข้อมูลนำเข้าลิสต์

ผลลัพธ์ ตำแหน่งข้อมูลที่อยู่ในลิสต์

Linked List (Cont.)

Algorithm retrieveNode (val pList <head pointer>,val key <key type>, ref dataOut <dataType>)
Pre pList is a pointer to a valid list head structure
key is a target of data to be retrieved
dataOut contains address for retrieved data
Post data placed at location specified in dataOut or error returned if not found
Return true if successful, false if data not found
1. found = searchList (pList, pPre, pLoc, key)
2. if (found)
   1 dataOut=pLoc->data
3. return found

End retrieveNode

Linked List (Cont.)

7. ฟังก์ชั่น EmptyList

หน้าที่ ทดสอบว่าลิสต์ว่างข้อมูลนำเข้า ลิสต์

ผลลัพธ์ เป็นจริง ถ้าลิสต์ว่าง

เป็นเท็จ ถ้าลิสต์ไม่ว่าง

Algorithm emptyList (val pList <head pointer>)
Pre pList is a pointer to a valid list head structure
Return true if list empty, false if list contains data
1. Return (pList->count equal to zero)

End emptyList

Linked List (Cont.)

8. ฟังก์ชั่น FullList
   หน้าที่ ทดสอบว่าลิสต์เต็มหรือไม่ข้อมูลนำเข้าลิสต์
   ผลลัพธ์ เป็นจริง ถ้าหน่วยความจำเต็ม
   เป็นเท็จ ถ้าสามารถมีโหนดอื่น

Linked List (Cont.)

Algorithm fullList (val pList <head pointer>)
Pre pList is a pointer to a valid list head structure
Return false if another, true if memory full
1. allocate (pNew)
2. if (allocation successful)
   1 release (pNew)
   2 return false
3. return true

End fullList

Linked List (Cont.)

9. ฟังก์ชั่น list count
   หน้าที่ นับจำนวนข้อมูลที่อยู่ในลิสต์
   ข้อมูลนำเข้าลิสต์
   ผลลัพธ์ จำนวนข้อมูลที่อยู่ในลิสต์

Algorithm listCount (val pList <head pointer>)
Pre pList is a pointer to a valid list head structure
Return count for number of node in list
1. Return (pList->count)

End listCount

Linked List (Cont.)

10. กระบวนงาน destroy list
    หน้าที่ ทำลายลิสต์
    ข้อมูลนำเข้า ลิสต์
    ผลลัพธ์ ไม่มีลิสต์

Linked List (Cont.)

Algorithm destroyList (ref pList <head pointer>)

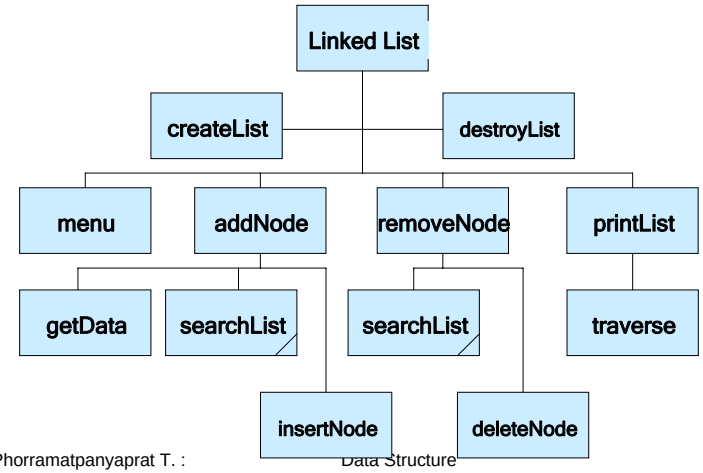
Pre pList is a pointer to a valid list head structure
Post All data and head structure deleted
Return null head pointer

1. Loop (pList->count not zero)
   1 dltPtr = pList->head
   2 pList->head = dltPtr->link
   3 pList->count = pList->count-1
   4 release (dltPrt)
2. release (pList)
3. Return null pointer

End destroyList

Linked List (Cont.)

การสร้าง Linked List



Linked List (Cont.)

program buildLinkList

1. print (Welcome to exploring linked list)
2. pList=createList
3. option = menu()
4. loop (option not quit)
   1 if (option add)
      1 getData (dataIn)
      2 addNode (pList, dataIn)
   2 elseif (option delete)
      1 print (Enter key of data to be delete)
      2 read (deleteKey)
      3 removeNode (pList,deleteKey)

Linked List (Cont.)

   3 else
      1 printList (pList)
   4 option=menu ()
5. destroyList (pList)
6. Print (Exploration complete)

End buildLinkedList

Linked List (Cont.)

Algorithm menu

Pre Noting
Post Valid choice

1. print (.....Menu.....)
2. print (A: Add new data)
3. print (D: Delete data)
4. print (P: Print List)
5. print (Q: Quit)
6. valid = false

Linked List (Cont.)

7. loop (valid false)
   1 print (Enter your choice:)
   2 read (choice)
   3 if (choice equal 'A' or 'D' or 'P' or 'Q')
      1 valid=true
   4 else
      1 print (Invalid choice. Choices are <A, D, P, Q>)
8. return choice

End menu

Linked List (Cont.)

Algorithm addNode (val pList <head pointer>,
val dataIn <dataType>)
Pre pList is a pointer to the head of the list
dataIn are data to be inserted into list
Post data have been inserted into list in key sequence

1. found = searchList (pList, pPre, pLoc, dataIn.key)
2. if (found)
   1 print (error : Data already in list. Not added)
3. else
   1 success = insertNode (pList, pPre, dataIn)

Linked List (Cont.)

4. if (success false)
   1 print (error : Out of memory)
   2 abort algorithm
5. Return

End addNode

Linked List (Cont.)

Algorithm removeNode (val pList <head pointer>,
val key <keyType>)
Pre pList is a pointer to the head of the list
key is the key to be located and deleted
Post the node has been delted or a warning message
printed if not found
1. found = searchList (pList, pPre, pLoc, key)
2. if (found)
1 deleteNode (pList, pPre, pLoc, deleteData)
3. else
1 print (error : Key not in list)
4. return
End removeNode

Linked List (Cont.)

Algorithm printList (val pList <head pointer>)
Pre pList is a valid linked list
Post All keys have been printed

1. if emptyList (pList))
1 print (No data in list)
2. else
1 print (***Begin Data Print ***)
2 count = 0
3 moreData = traverse (pList, count, dataPtr)

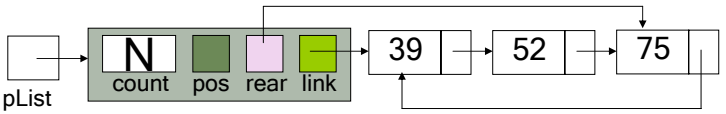
Linked List (Cont.)

4 Loop (moreData true)
1 count=count+1
2 print (count, dataPtr->key)
3 moreData = traverse (pList, count, dataPtr)
5 Print (*** End of Data Print ***)
3. return
End printList

Linked List (Cont.)

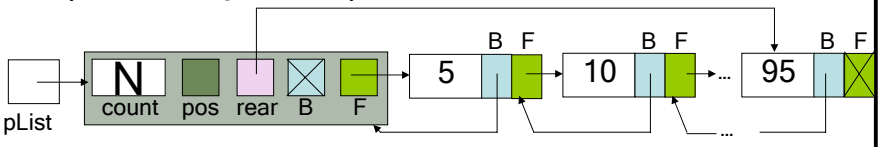
Linked List แบบซับซ้อน

1. Circular Linked List เป็นลิงค์ลิสต์ที่สมาชิกตัวสุดท้ายมีตัวชี้ (list) ชี้ไปที่สมาชิกตัวแรกของลิงค์ลิสต์ จะมีการทำงานไปในทิศทางเดียวเท่านั้น คือเป็นแบบวงกลม



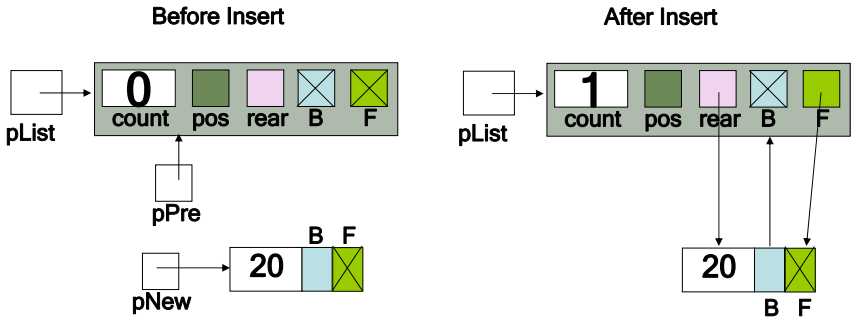
Linked List (Cont.)

2. Double Linked List เป็นลิงค์ลิสต์ที่มีทิศทางการทำงานแบบ 2 ทิศทาง ในลิงค์ลิสต์แบบ 2 ทิศทาง ส่วนข้อมูลจะมีตัวชี้ไปที่ข้อมูลก่อนหน้า (backward pointer) และตัวชี้ข้อมูลถัดไป (forward pointer)



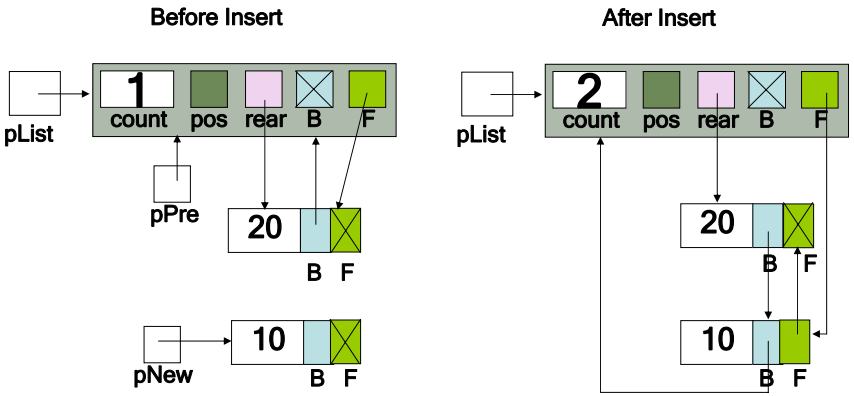
Linked List (Cont.)

กระบวนงาน Insert Node



Insert after last node (or null list)

Linked List (Cont.)



Insert between two node (or before first node)

Linked List (Cont.)

Algorithm insertDbl (val pList <node pointer>,
val dataIn <dataType>)

Pre pList is a pointer to a valid double linked list
dataIn contains the data to be inserted

Post The data have been inserted in sequence

Return 0 : failed-memory overflow
1 : successful
2 : failed-duplicate key presented

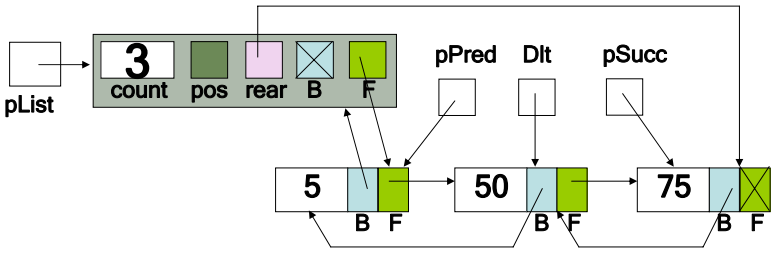
1. if (full list)
   1 return (0)
2. found = searchList (pList, pPre, pSucc, dataIn.key)

Linked List (Cont.)

```
if (found false)
    1  allocate (pNew)
    2  pNew->data = dataIn
    3  pNew->back=pPre
    4  pNew->fore=pPre->fore
    5  if (pPre->fore null)
       1  pList->rear=pNew
    6  else
       1  pSucc->back=pNew
    7  pPre->fore=pNew
    8  pList->count=pList->count+1
    9 return (1)
4.  Return (2)
End insertDbl
```

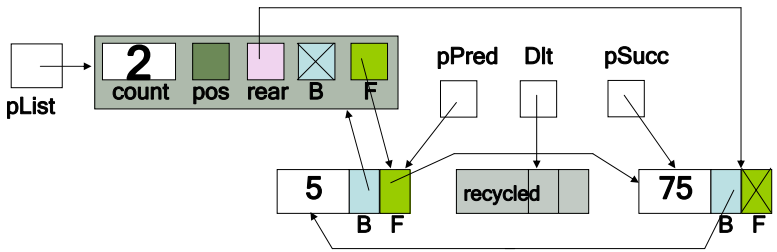

Linked List (Cont.)

กระบวนงาน Delete Node



before delete

Linked List (Cont.)



after delete

Linked List (Cont.)

```
Algorithm deleteDbl (val pList   <node pointer>,
                     val pDlt    <node pointer>)
    Pre      pList is a pointer to a valid double linked list
             pDlt is a pointer to the node to be deleted
    Post     node deleted
1.  if (pDlt null)
    1  abort (impossible condition in delete double)
2.  pList->count=pList->count-1
3.  pPred = pDlt->back
4.  pSucc = pDlt->fore
```

Linked List (Cont.)

3. pPred->fore=pSucc
4. If (pSucc not null)
   1 pSucc->back=pPred
7. Recycle (pDlt)
8. Return

End deleteDbl

แบบฝึกหัด

1. อธิบายโครงสร้างการทำงานของ ลิสก์เชื่อมโยง
2. ท่านสามารถลบ โหนดจากลิสก์ที่มีการเชื่อมโยงออกจากกันได้หรือไม่ กรณีที่ใช้ตัวชี้เป็นตัวดำเนินการเชื่อมโยง อธิบายหลักการดำเนินการ
3. ให้อธิบายหลักการทำงานของ Circular Linked List