CSE221

Lecture 3:
Linked Lists

Hyungon Moon

Slide credits: The textbook authors, Won-ki Jeong,
Tsz-Chiu Au, and Myeongjae Jeon

UNIST
ULSAN NATIONAL INSTITUTE OF
SCIENCE AND TECHNOLOGY

Administrivia

- Lab0
  - Due tomorrow
  - Will pre-grade today (for your reference)
- Lab1
  - Will be out tonight

Administrivia

- TA office hour
  - Will be announced tomorrow through Gitlab.
- Uni server
  - I'll check the configuration today.
- VPN
  - 1st applicants: should have been set up by Changjoo.
  - Will give you another chance to apply.

Outline

- Singly linked list
- Doubly linked list
- Circular lists

Outline

- Singly linked list
- Doubly linked list
- Circular lists

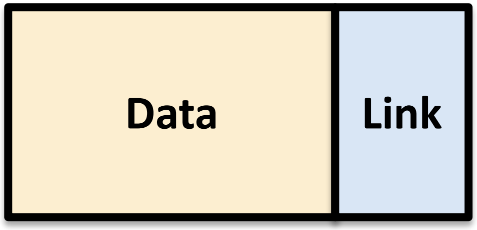
Sequential List Representation

- Elements of the list are stored in sequential order
  - – a[i][j] location is L_{ij}
  - – a[i][j+1] location is $L_{ij}+1$
- Insertion / deletion is expensive
  - – Need to move elements to keep correct sequential representation
- Resize needed

Linked List (Or List with Nodes)

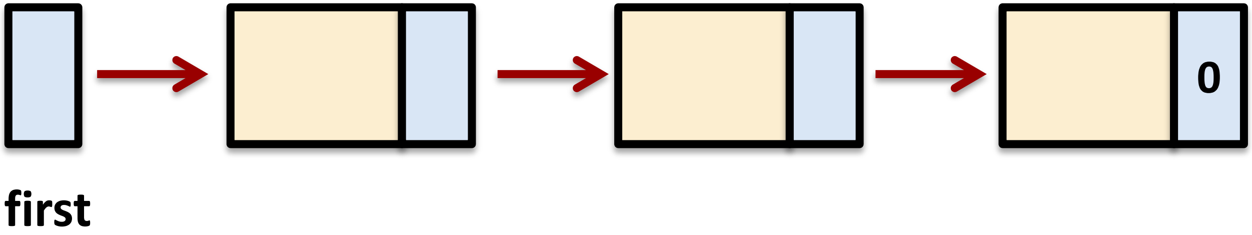
- Elements are stored in an arbitrary order in memory
- Order can be maintained by using explicit information (i.e., link)

- Node of linked list
  - Data field
  - Link (pointer) fields



Singly Linked List Representation

- Chain = singly linked list
- First pointer points to first node
- Null terminated at the end

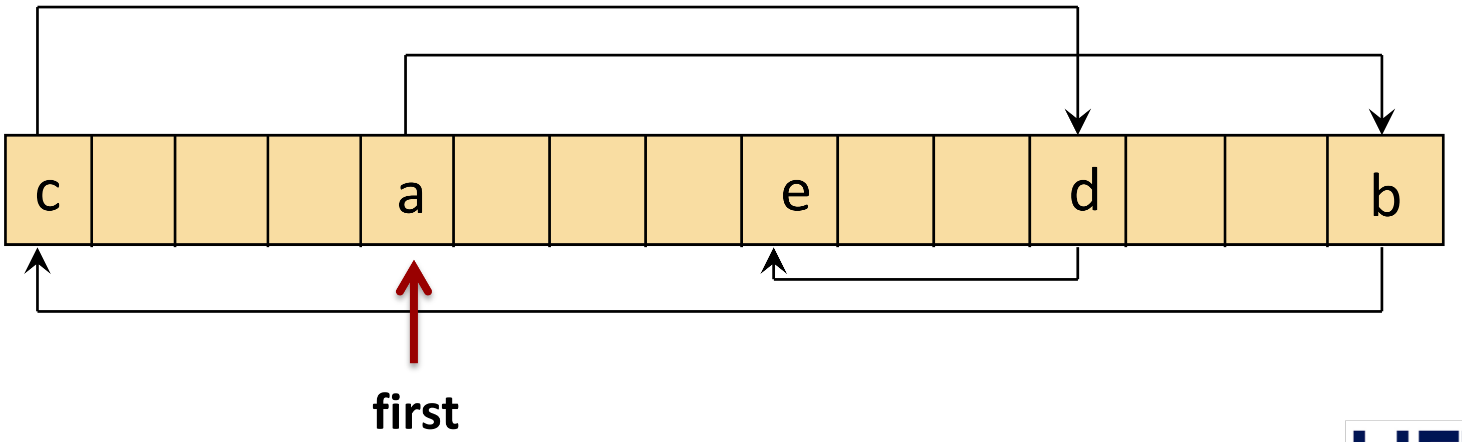


Memory Layout

- L=(a,b,c,d,e)
- Array representation

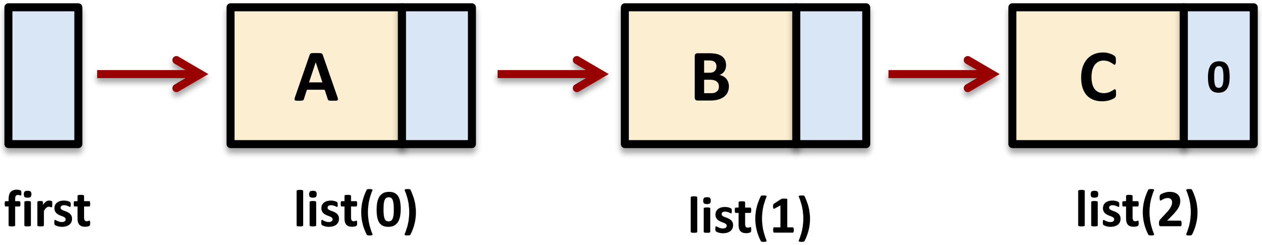
a	b	c	d	e										

- Linked list representation



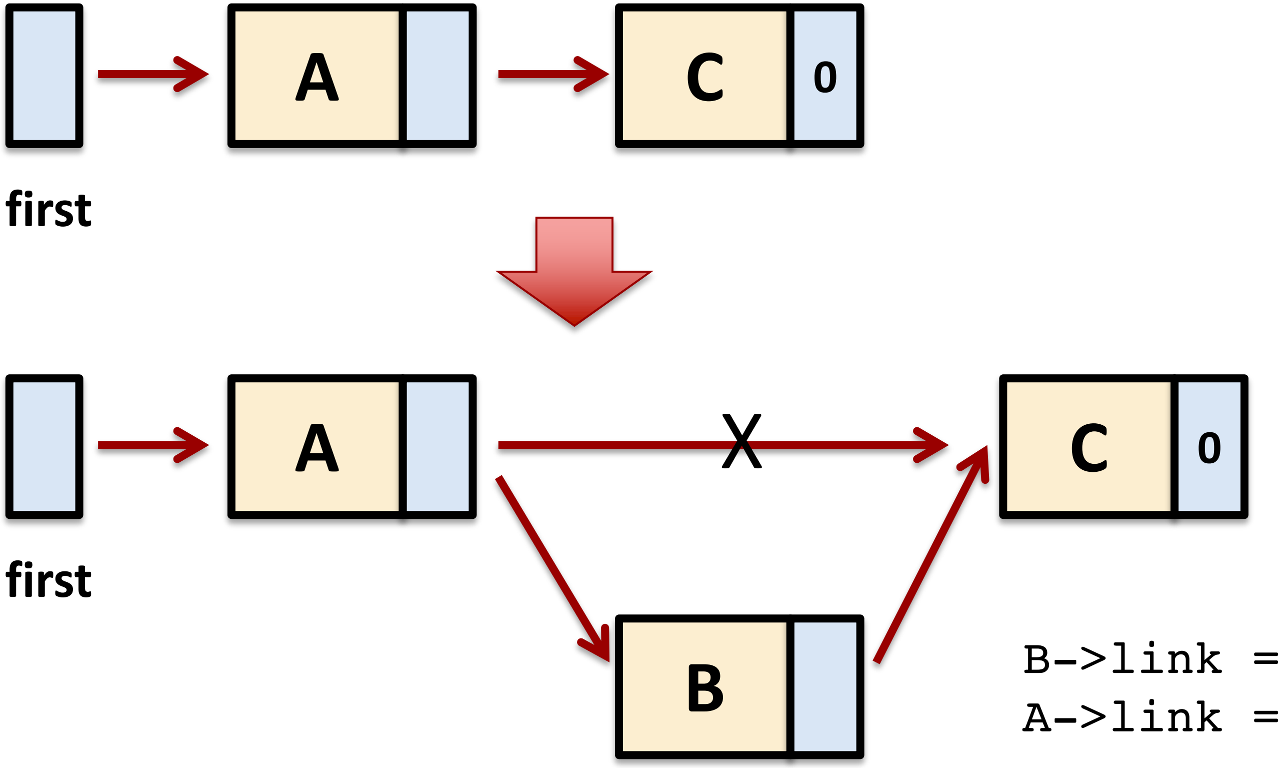
Basic Operations

- Access element
  - list(0) = first->data= A
  - list(1) = first->link->data= B
  - list(2) = first->link->link->data= C



Basic Operations

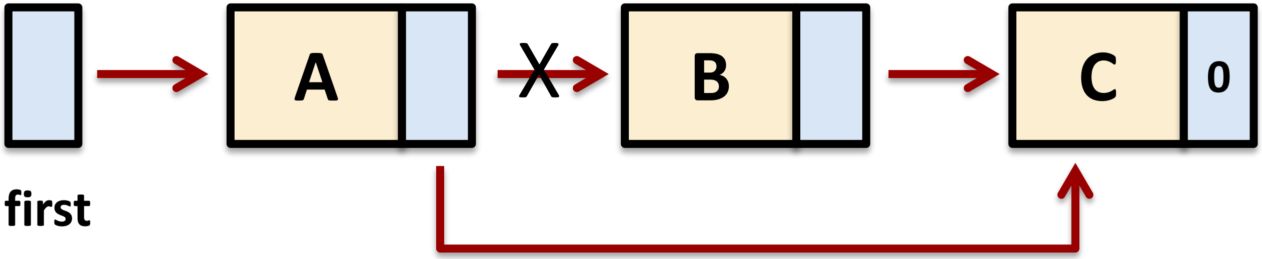
- Insert B after A



```
B->link = A->link
A->link = B
```

Basic Operations

- Delete B
  - Need A preceding B
    - A has to be either given or searched

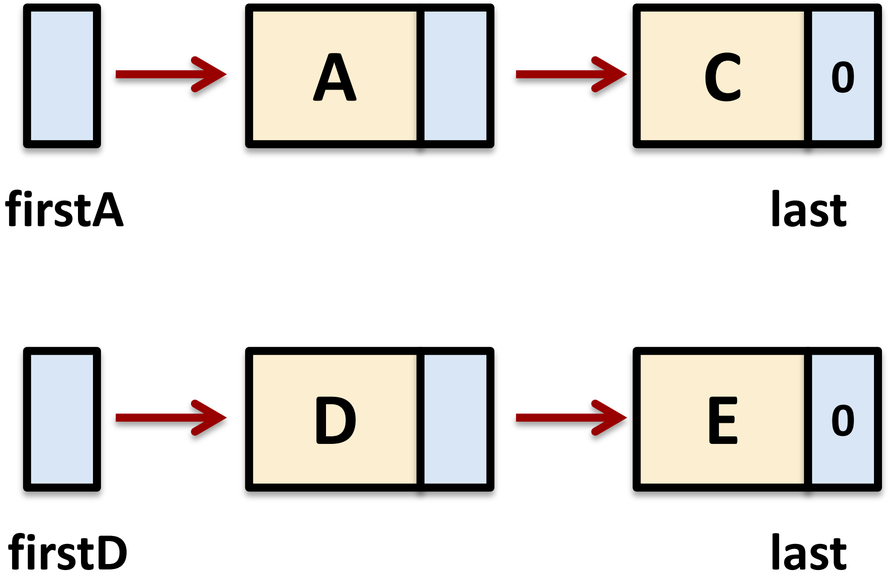


```
A->link = B->link
delete B
```

Note: Deleting last node requires traversing the entire list even though we have the pointer of the last node!

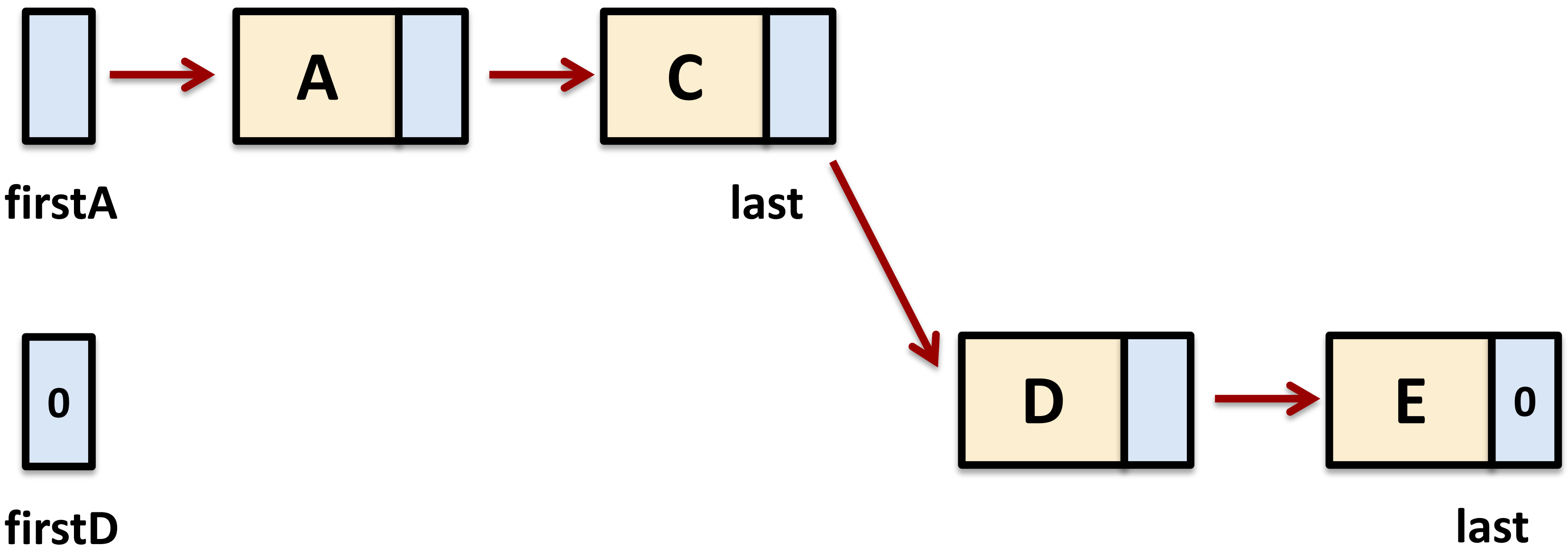
Basic Operations

- Concatenation



Basic Operations

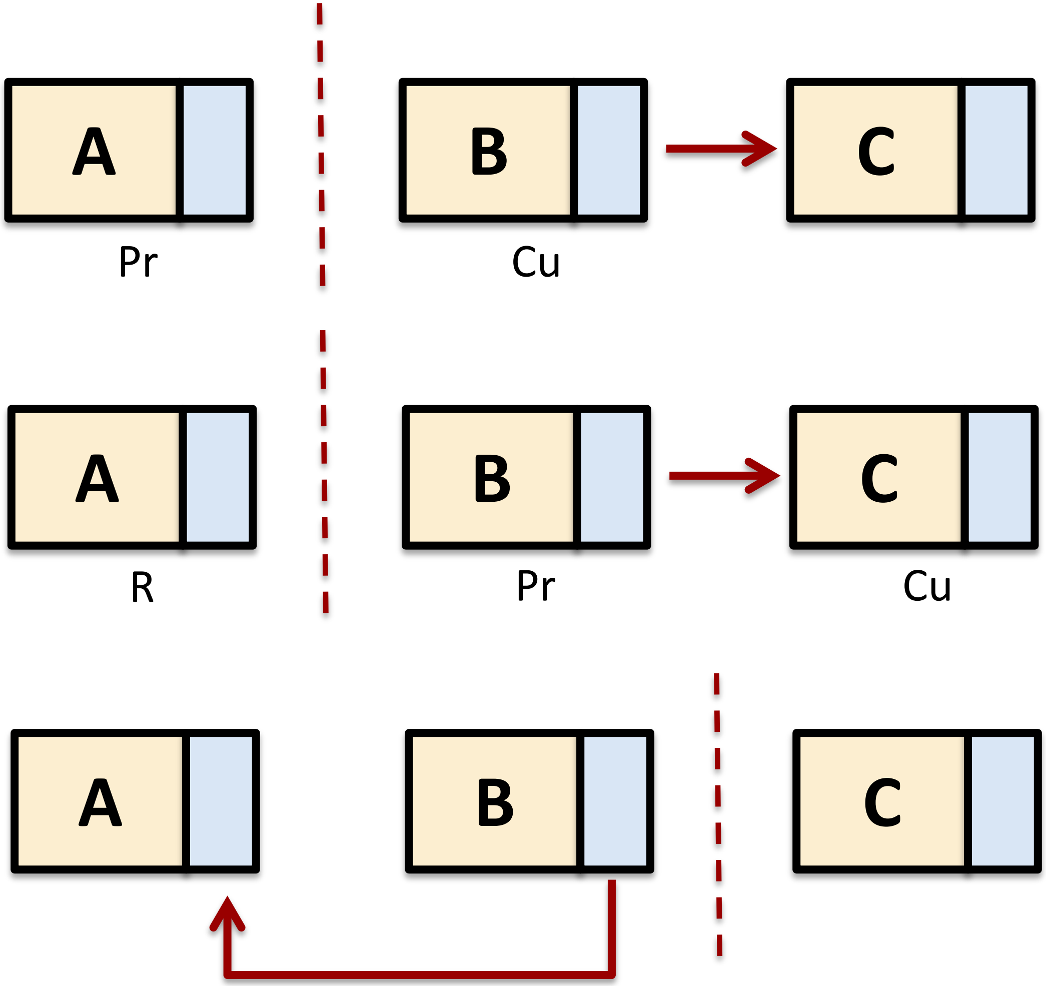
- Concatenation



```
C->link = firstD
firstD = NULL
```

Basic Operations

- Reverse (at Current)



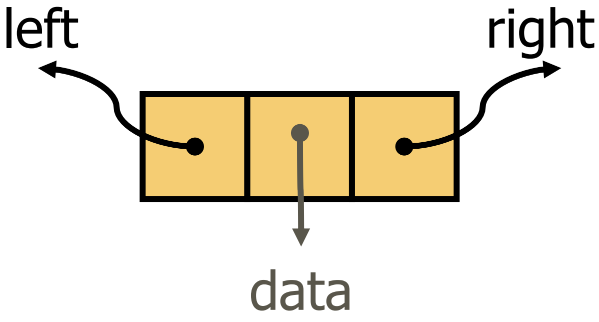
```
R = Previous
Previous = Current
Current = Current->link
Previous->link = R
```

Outline

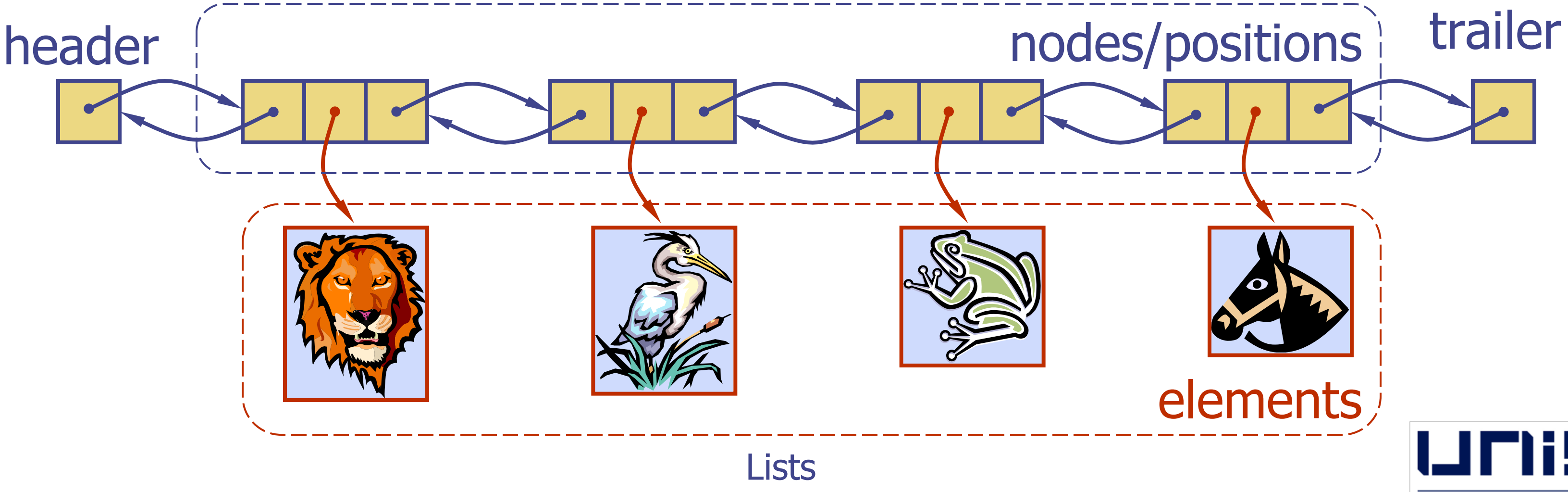
- Singly linked list
- Doubly linked list
- Circular lists

Doubly Linked List

- Two link pointers
  - Left, right
- Can traverse both directions
- Node can be deleted using a single pointer

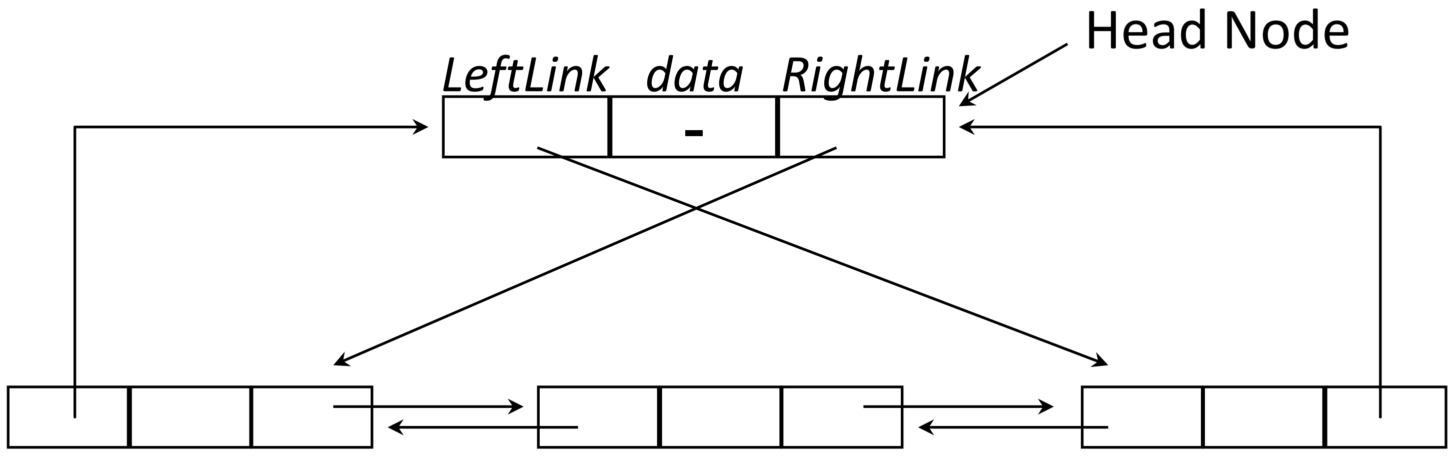


p->left->right = p->right; p->right->left = p->left; delete p;



Doubly Linked List

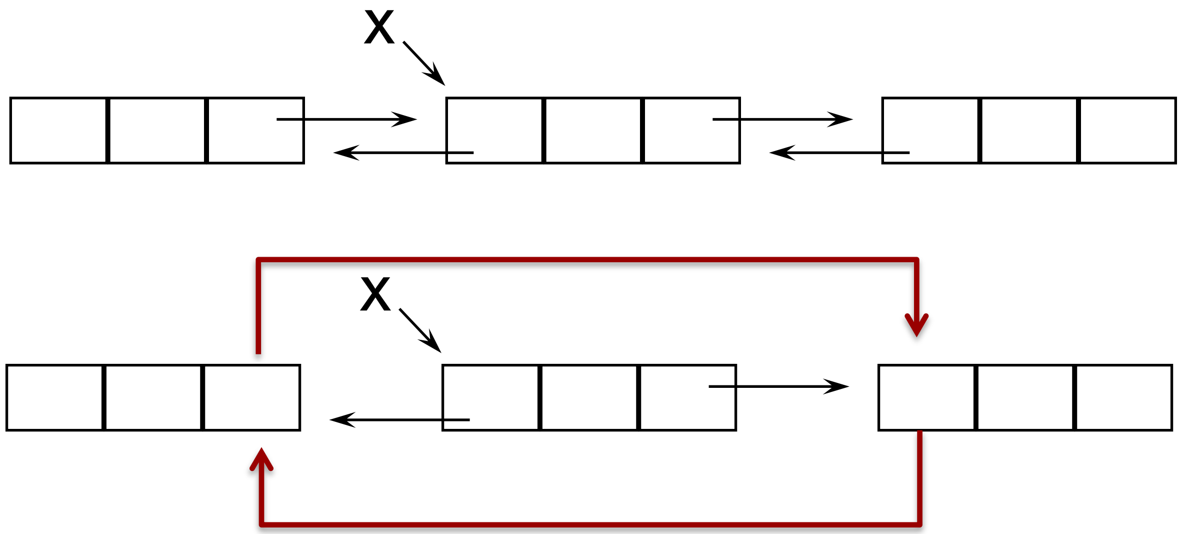
- Using a head node (version 2)
  - Can point to head and tail using a single node



Doubly Linked List

- Delete

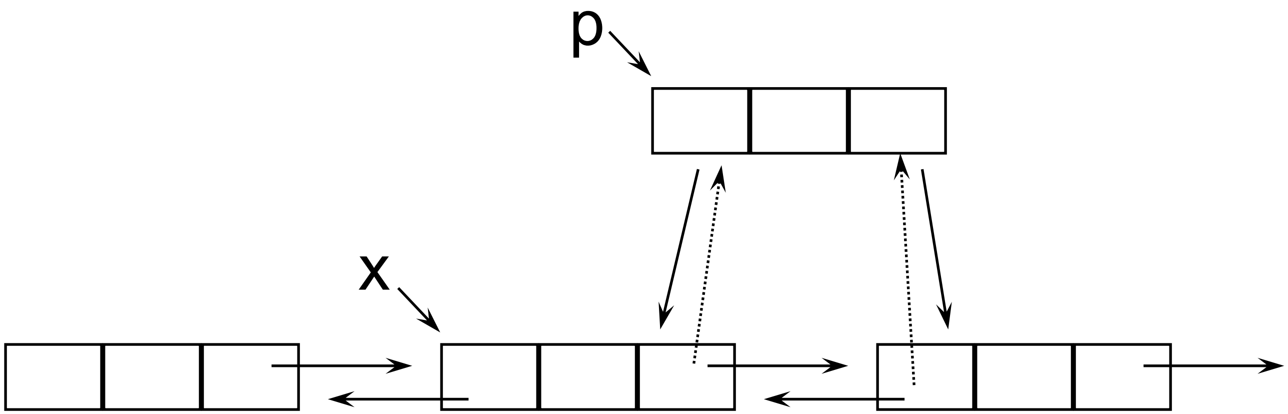
```
void DblList::Delete(DblListNode *x)
{
    x->left->right = x->right;
    x->right->left = x->left
    delete x;
}
```



Doubly Linked List

- Insert

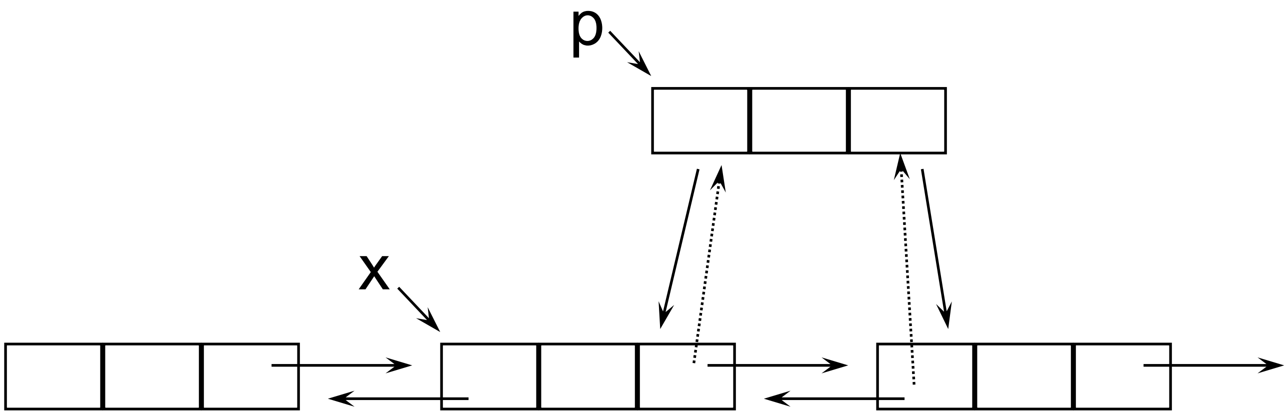
```
void DblList::Insert(DblListNode *p, DblListNode *x)
{
    p->llink = _______________;
    p->rlink = _______________;
    x->______________ = p;
    x->______________ = p;
}
```



Doubly Linked List

- Insert

```
void DblList::Insert(DblListNode *p, DblListNode *x)
{
    p->llink = x;
    p->rlink = x->rlink;
    x->rlink->llink = p;
    x->rlink = p;
}
```

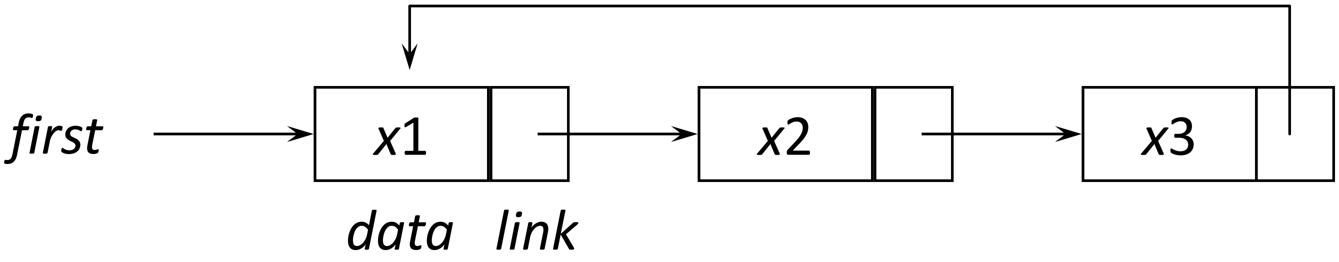


Outline

- Singly linked list
- Doubly linked list
- Circular lists

Circular Linked List

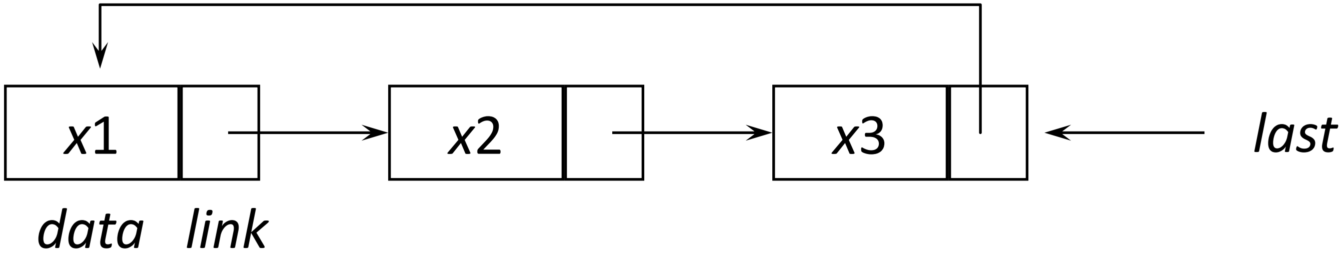
- Link of the last node points to the first node
  - – last->link = first
  - – No null pointer
- Efficient for circular accessing problems
  - – Round-robin scheduling



Circular Linked List

- Insert at the end is inefficient
  - Need to search *last* from *first*

- Keep *last* instead of *first*
  - Insert at the end and front can be O(1)
  - first = last-> link



Available Space Lists

- New (malloc) and delete (free) are expensive
- We need O(n) time to delete every node in the list of size n
- We can manage a pool (list) of free nodes
  - When adding a new node, request a free node from the pool
  - When a node is deleted, return it to the pool
  - Can be done in O(1)

Available Space Lists

- *avail*: first pointer of available space list

- GetNode()

```
template <class Type>
ListNode<Type>* CircList::GetNode()
// Getting a node from the pool
{
   ListNode<Type>* x;
   if(!avail) x = new ListNode<Type>;
   else { x = avail; avail = avail->link; }
   return x;
}
```

Available Space Lists

- RetNode()

```
template <class Type>
void CircList<Type>::RetNode(ListNode<Type>* x)
// Return x to the free node pool
{
   x->link = avail;
   avail = x;
   x = 0;
}
```

Available Space Lists

- Delete entire circular list in O(1)

```
template <class KeyType>
void CircList<Type>::~CircList()
// Delete the circular linked list
{
   if (last) {
      ListNode<Type>* first = last->link; // assume we store last
         last->link = avail; // last node linked to avail
         avail = first;  // first node of list becomes front of
avail
                          list
         last = 0;
   }
}
```

Questions?