Link Lists Gone Wild: *Circular and Doubly Linked Lists*

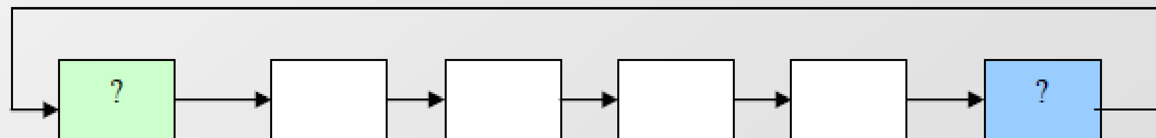
Computer Science Department
University of Central Florida

COP 3502 – Computer Science I

Circular Lists

- The linked list structures that we have just examined are all of the same type, called a *singly-linked list*. Each node in the list contains a single reference (pointer) to the node which logically follows it in the list. There are many different variations of linked lists that have been developed.

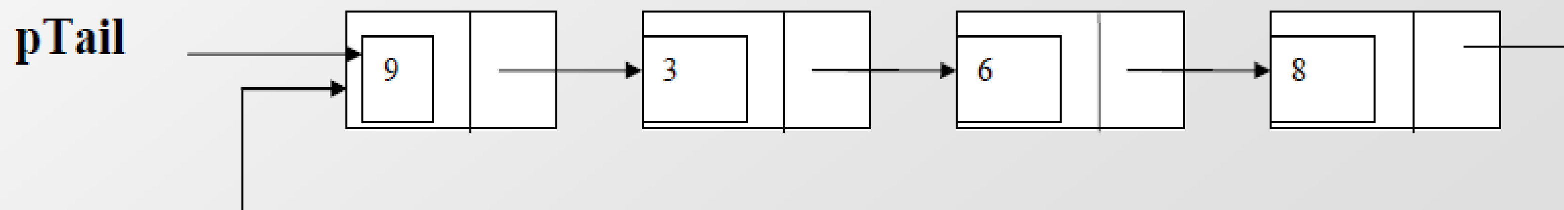
  – circular singly-linked lists – the last node in the list contains refers to the first node in the list.



Inserting a Node at Front of a Circular List

- Let us add a node containing data d, in front of a circular list pointed by pTail.

- The first node will be the node next to the tail node. The new node will have to be inserted just before the first node, i.e. in between the tail node and the first node.
  - get a new node from memory ,
  - put the data d in that node, and make it point to itself.
  - if pTail is empty, this node will be the only node in the circular list.
  - if it is not the first node, then its next part should contain the address which pTail was pointing to.
  - finally, its address must be stored in the next part of pTail.



Inserting a Node at Front of a Circular List

```
struct node* pNew = (struct node*) (malloc(sizeof(struct node)));
pNew ->data = d;
pNew->next = pNew;
if (pTail==NULL)
  pTail = pNew;
else{
  pNew->next = pTail->next;
  pTail->next = pNew;
}
```

- What is the complexity of this operation?
- Since the complete list is not being traversed, it is O(1).

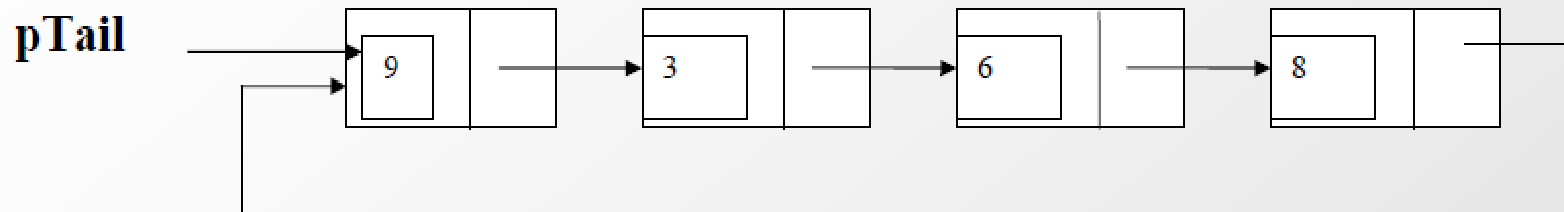
Inserting a Node at the End of a Circular List

- Let us add a node containing data d, at end of a circular list pointed by pTail.
- The new node will be placed just after the tail node (which is the last node of the list), which means again it will have to be inserted in between the tail node and the first node.
- The code will be identical to the code given before for inserting a node in front of the list, except that pTail should now point to the new last node.
- What is the complexity? It is again O(1).
  - Compare the complexity with the complexity of adding a node at the end of a standard linked list which is O(n), because in that case you have to traverse the complete list to reach the last node (with NULL value in next field).

Inserting a Node at the End of a Circular List

```
struct node* pNew = (struct node*) (malloc(sizeof(struct node)));
    pNew ->data = d;
    pNew->next = pNew;
    if (pTail==NULL)
        pTail = pNew;
    else{
        pNew->next = pTail->next;
        pTail->next = pNew;
        pTail = pNew;
    }
```

Deleting the First Node in a Circular List



- The first node can be deleted by simply replacing the next field of tail node with the next field of the first node.

```
temp = pTail->next;
pTail->next = temp->next;
free(temp);
```

Deleting the Last Node in a Circular List

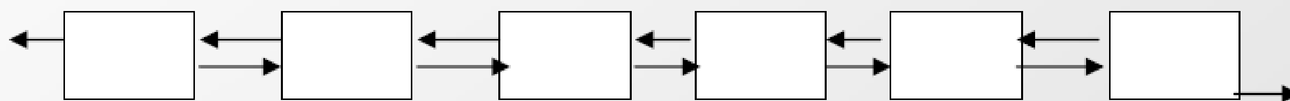
- Deletion of last node is a more complicated case.
  - The list has to be traversed to reach the last but one node.
  - This has to be named as the tail node, and its next field has to point to the first node.

- Consider the following list. To delete the last node 9, the list has to be traversed till you reach 8. The next field of 8 has to be changed to point to 3, and this node must be renamed pTail.

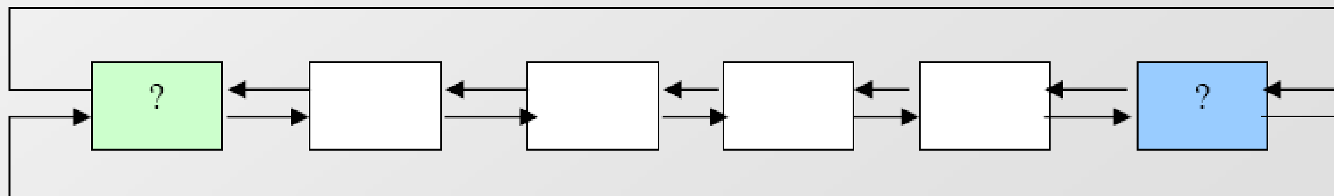
- This is left as an exercise for you. Write the code and work out its complexity.

Doubly Linked Lists

- *doubly-linked lists* – each node in the list contains a reference to both the node which immediately precedes it and to the node which follows it in the list.



- *circular doubly-linked lists* – same as a circular singly-linked list except that the nodes in the list are doubly-linked.



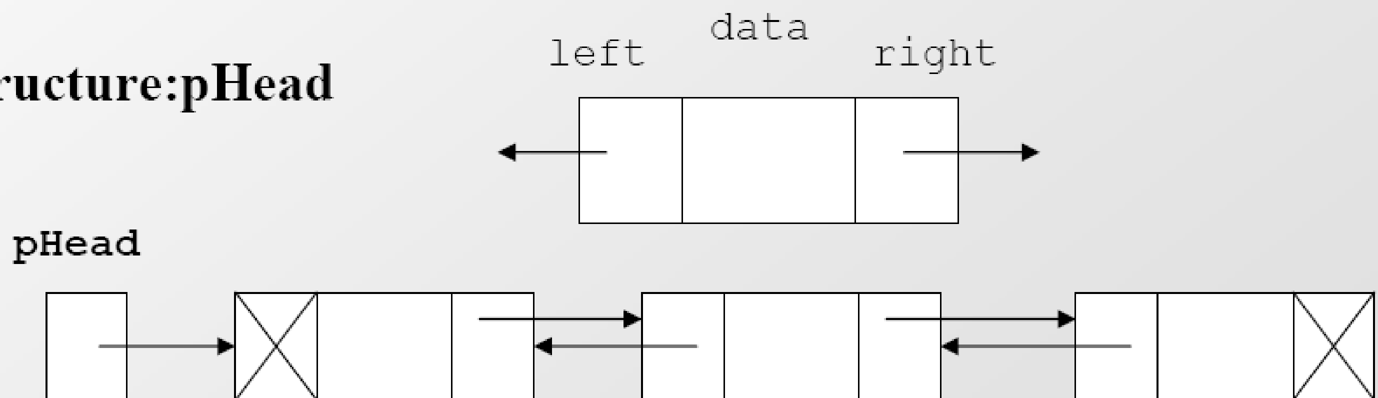
Doubly Linked List

- Simple linked lists only allow making search from the beginning to end.
- Doubly linked lists allow searches in both directions (while keeping a single pointer)

Each node contains two pointers, one to the next node, one to the preceding node.

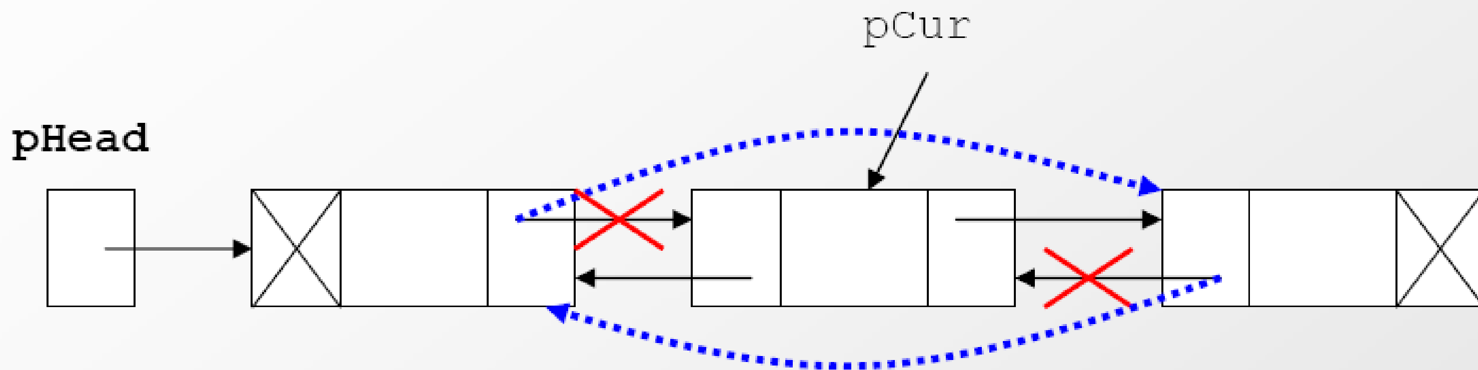
```
struct dllNode {
    int data;
    struct dllNode *left;
    struct dllNode *right;
}
```

- **Node Structure:pHead**



- **Advantage:**
  - insertion and deletion can be easily done with a single pointer.

Doubly Linked List - Deletion

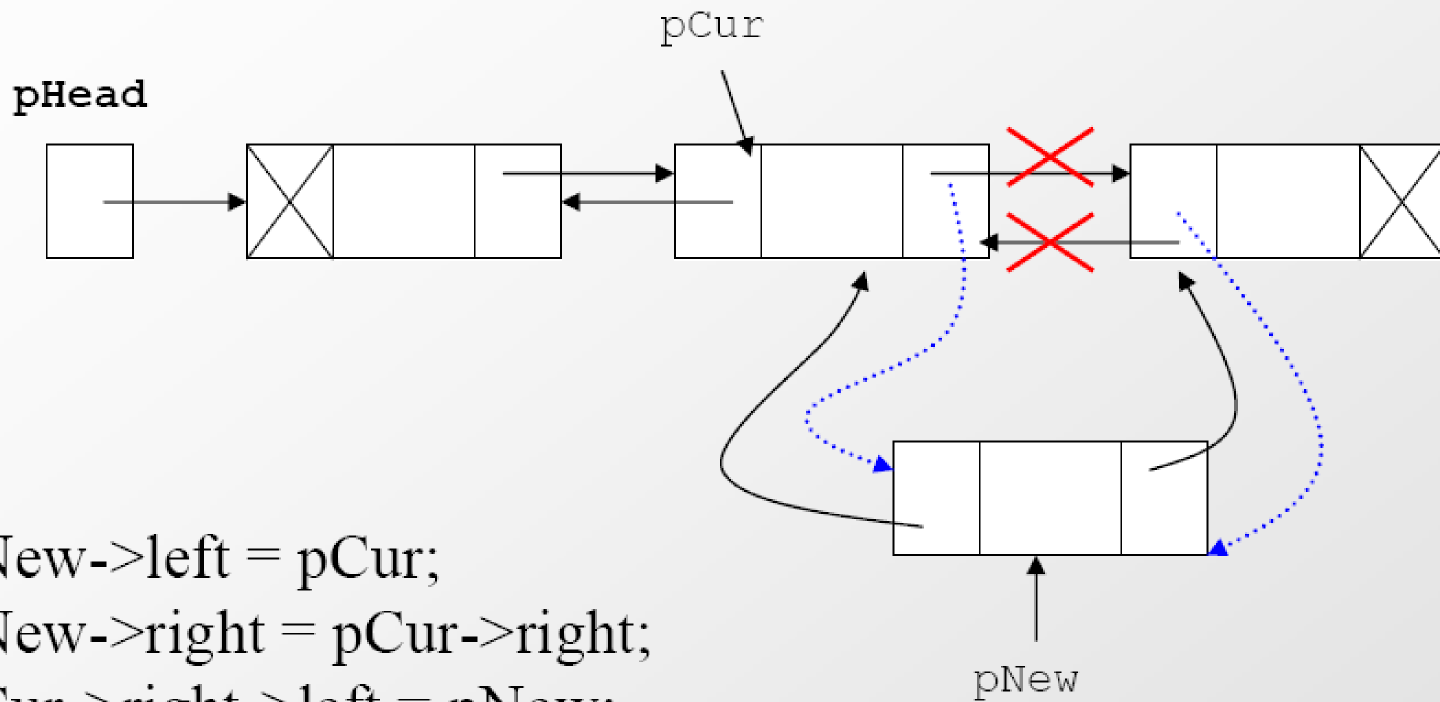


pCur->left->right = pCur->right;

pCur->right->left = pCur->left;

(Assuming pCur->left and pCur->right are not NULL)

Doubly Linked List - Insertion




```
pNew->left = pCur;
pNew->right = pCur->right;
pCur->right->left = pNew;
pCur->right = pNew;
```

- **Disadvantage of Doubly Linked Lists:**
  - extra space for extra link fields
  - maintaining extra link during insertion and deletion

Linked Lists Gone Wild

ISN'T

THAT

GREAT (*That we are finished with Linked Lists!*)!

Daily Demotivator

© DESPAIR.COM

WINNERS

Because Nothing Says "You're a Loser" Than Owning a Motivational Poster About Being a Winner.

Link Lists Gone Wild: *Circular and Doubly Linked Lists*

Computer Science Department
University of Central Florida

COP 3502 – Computer Science I