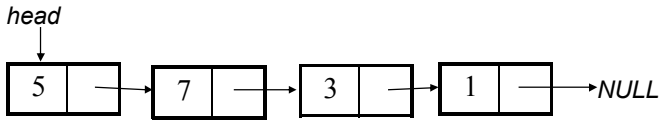
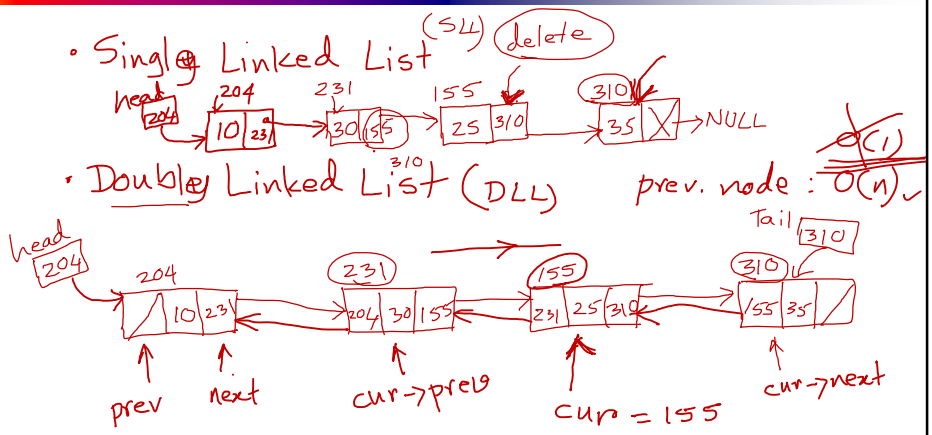
CSE 203: Data Structures and Algorithms-I

Linked Lists



Dr. Md. Abul Kashem Mia, Professor, CSE Dept and Pro-Vice Chancellor, UIU

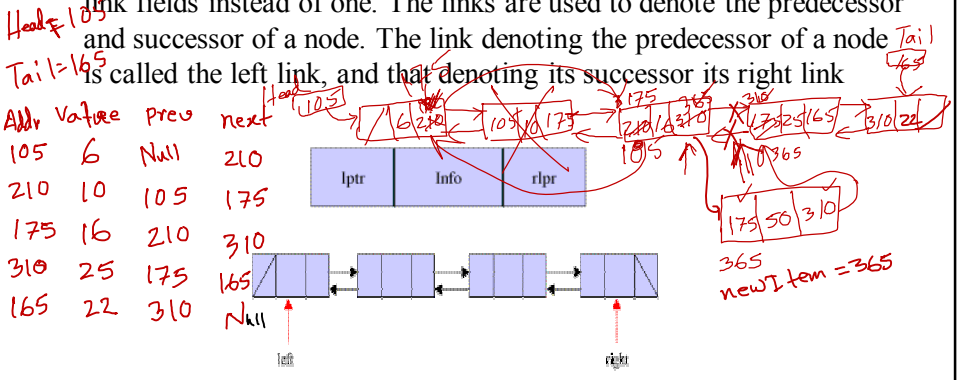
Introduction to Doubly Linked List



Dr. Md. Abul Kashem Mia, Professor, CSE Dept and Pro-Vice Chancellor, UIU

Introduction to Doubly Linked List

- We have discussed the details of linear linked list. In the linear linked list, we can only traverse the linked list in one direction.
- But sometimes, it is very desirable to traverse a linked list in either a forward or reverse manner.
- This property of a linked list implies that each node must contain two link fields instead of one. The links are used to denote the predecessor and successor of a node. The link denoting the predecessor of a node is called the left link, and that denoting its successor its right link



Basic Operation of Doubly Linked List

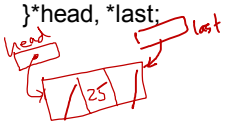
- Insert: Add a new node in the first, last or interior of the list.
- Delete: Delete a node from the first, last or interior of the list.
- Search: Search a node containing particular value in the linked list.

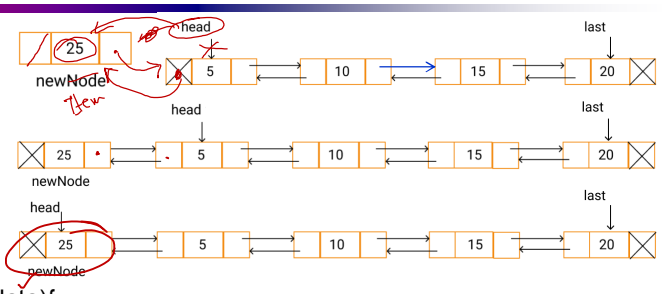
Insert First or Insert Last into a Doubly Linked List

- **Insertion** is to add a new node into a linked list. It can take place anywhere -- the first, last, or interior of the linked list.
- To add a new node to the head and tail of a double linked list is similar to the linear linked list.
- First, we need to construct a new node that is pointed by pointer *newItem*.
- Then the newItem is linked to the left-most node (or right-most node) in the list. Finally, the *Left* (or *Right*) is set to point to the new node.

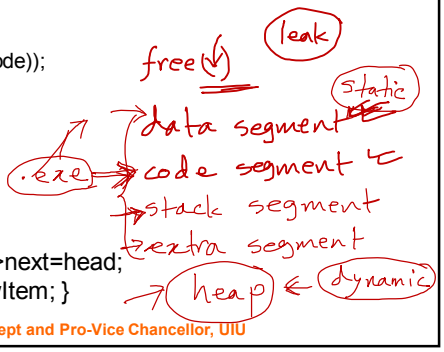
Insert (First) into a Doubly Linked List

```
struct dnode{
    struct dnode *prev;
    int value;
    struct dnode *next;
}*head, *last;
```



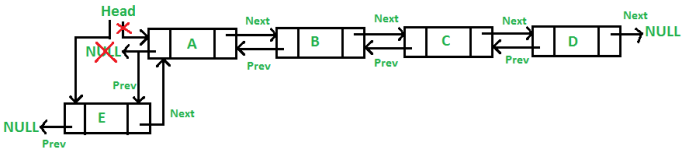



```
void insert_begning(int data){
    struct dnode *newItem;
    newItem=(struct dnode *)malloc(sizeof(struct dnode));
    newItem->value=data;
    if(head==NULL){
        head=newItem;        head->prev=NULL;
        head->next=NULL;     last=head;
    }
    else{
        newItem->prev=NULL;          newItem->next=head;
        head->prev=newItem;          head=newItem; }
}
```



Insert (First) into a Doubly Linked List

```
struct dnode{
    struct dnode *prev;
    int value;
    struct dnode *next;
}*head, *last;
```



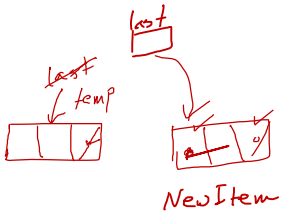

```
void insert_begning(int data){
    struct dnode *newItem;
    newItem=(struct dnode *)malloc(sizeof(struct dnode));
    newItem->value=data;
    if(head==NULL){
       head=newItem;        head->prev=NULL;
       head->next=NULL;     last=head;
    }
    else{
       newItem->prev=NULL;          newItem->next=head;
       head->prev=newItem;          head=newItem;
    }
}
```

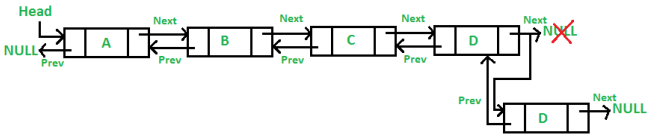

Insert (Last) into a Doubly Linked List

```
void insert_end(int data){
    struct dnode *newItem,*temp;
    newItem=(struct dnode *)malloc(sizeof(struct dnode));
    newItem->value=data;

    if(head==NULL){
        head=newItem;        head->prev=NULL;
        head->next=NULL;     last=head; }

    else{
       last=head;
       while(last != NULL){
          temp=last;
          last=last->next;
       }
     newItem->prev=temp;  newItem->next=NULL;
     temp->next=newItem;  last=newItem;
     }
}
```





Insert Interior of Doubly Linked List

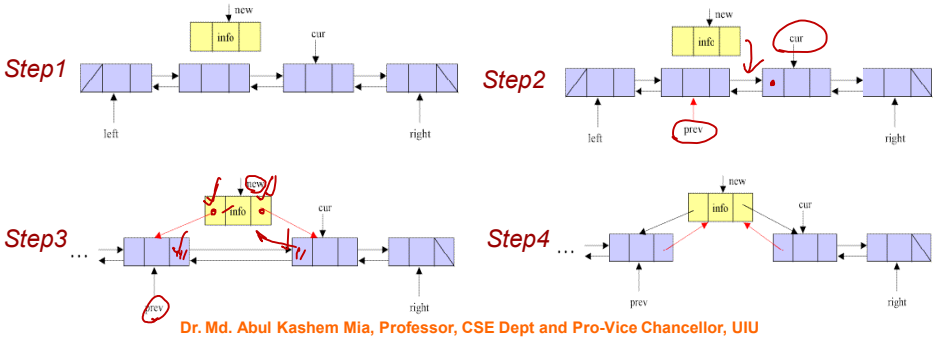
Insert a node before the node pointed by cur

Step1. Create a new node that is pointed by new

Step2. Set the pointer prev to point to the left node of the node pointed by cur.

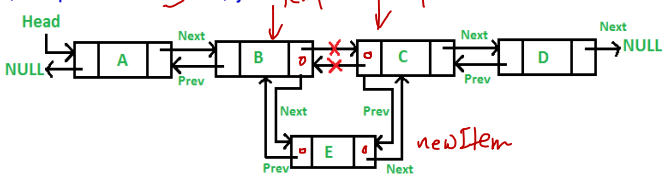
Step3. Set the left link of the new node to point to the node pointed by prev, and the right link of the new node to point to the node pointed by cur.

Step4. Set the right link of the node pointed by prev and the left link of the node pointed by cur to point to the new node.



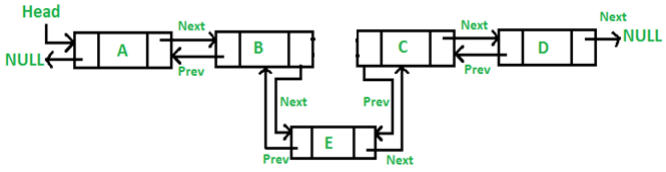
Insert Interior of Doubly Linked List

```
int insert_after(int data, int x){    \\ Insert after node x
    struct dnode *temp,*newItem,*temp1;
    newItem=(struct dnode *)malloc(sizeof(struct dnode));
    newItem->value=data;
      if(head==NULL){
        head=newItem;  head->prev=NULL;   head->next=NULL;  }
    else{
        temp=head;
        while(temp!=NULL && temp->value!=x)
            temp=temp->next;
        if (temp==NULL)
            printf("\n %d is not present in the list ",x);
        else{
        temp1=temp->next;  newItem->prev=temp; newItem->next=temp1;
        temp1->prev=newItem; temp->next=newItem; }
    }
}
```



Insert Interior of Doubly Linked List

```
int insert_after(int data, int x){    \\ Insert after node x
    struct dnode *temp,*newItem,*temp1;
    newItem=(struct dnode *)malloc(sizeof(struct dnode));
    newItem->value=data;
       if(head==NULL){
        head=newItem;  head->prev=NULL;   head->next=NULL;  }
    else{
         temp=head;
         while(temp!=NULL && temp->value!=x)
              temp=temp->next;
         if (temp==NULL)
             printf("\n %d is not present in the list ", x);
         else{
         temp1=temp->next;  newItem->prev=temp; newItem->next=temp1;
         temp1->prev=newItem; temp->next=newItem; }
    }
}
```

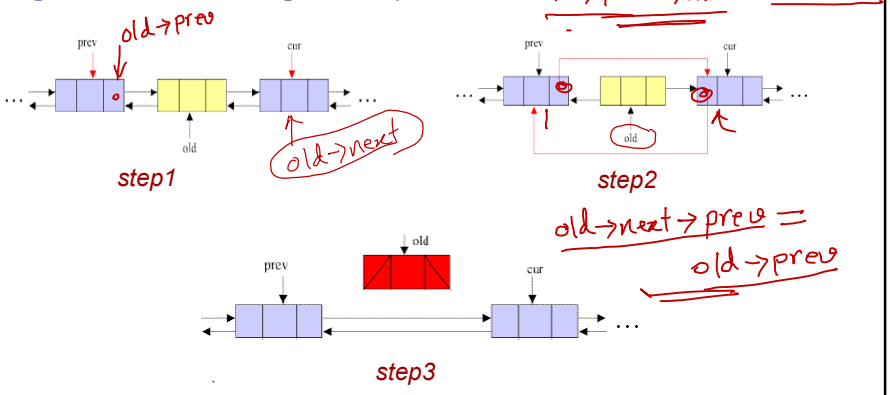


Deletion of Doubly Linked List

- **Deletion** is to remove a node from a list. It can also take place anywhere -- the first, last, or interior of a linked list.
- To delete a node from a double linked list is easier than to delete a node from a linear linked list.
- For deletion of a node in a single linked list, we have to search and find the predecessor of the discarded node. But in the double linked list, no such search is required.
- Given the address of the node that is to be deleted, the predecessor and successor nodes are immediately known.

Deletion of Doubly Linked List (Cont.)

- Step1. Set pointer *prev* to point to the left node of *old* and pointer *cur* to point to the node on the right of *old*.
- Step2. Set the right link of *prev* to *cur*, and the left link of *cur* to *prev*.
- Step3. Discard the node pointed by *old*.



Deletion of Doubly Linked List (Cont.)

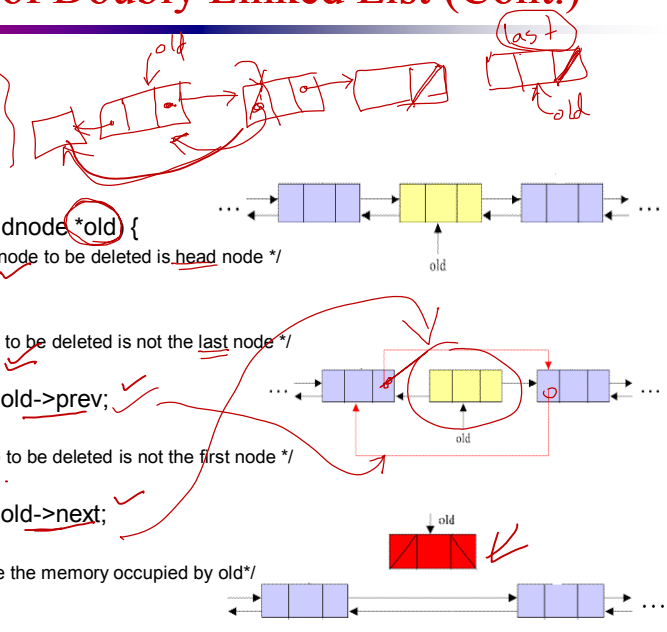
```
struct dnode {
   struct dnode *prev;
   int value;
   struct dnode *next;
   };

void deleteNode (struct dnode *old) {
   if(head == old)     /* If node to be deleted is head node */
      head = old->next;

   /* Change next only if node to be deleted is not the last node */
   if(old->next != NULL)
      old->next->prev = old->prev;

   /* Change prev only if node to be deleted is not the first node */
   if(old->prev != NULL)
      old->prev->next = old->next;

   free(old);   /* Finally, free the memory occupied by old*/
   return;
}
```



Advantages/Disadvantages of Doubly Linked List

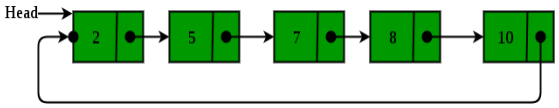
- **Advantages over singly linked list:**
  - A DLL can be traversed in both forward and backward direction.
  - We can quickly insert a new node before a given node.
  - The delete operation in DLL is more efficient if pointer to the node to be deleted is given.
    - In singly linked list, to delete a node, pointer to the previous node is needed. To get this previous node, sometimes the list is traversed. In DLL, we can get the previous node using 'prev' pointer.

- **Disadvantages over singly linked list:**
  - Every node of DLL require extra space for an previous pointer.
  - All operations require an extra pointer 'prev' to be maintained.

Circular Linked List

- In a circular linked list every element has a link to its next element and the last element has a link to the first element.
- That means circular linked list is similar to the single linked list except that the last node points to the first node in the list. There is no NULL at the end.
- A circular linked list can be a singly circular linked list or doubly circular linked list.

Singly circular linked list



Doubly circular linked list