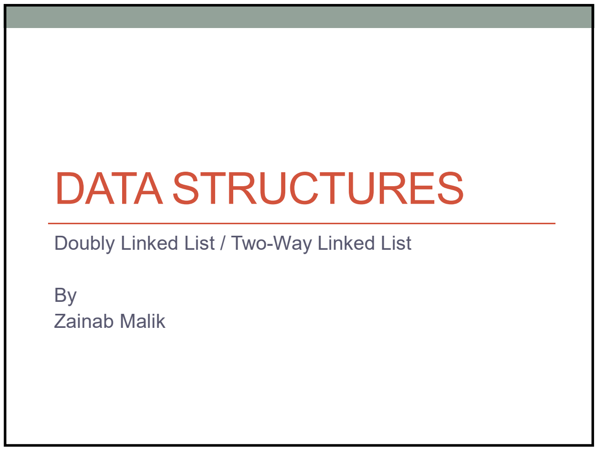
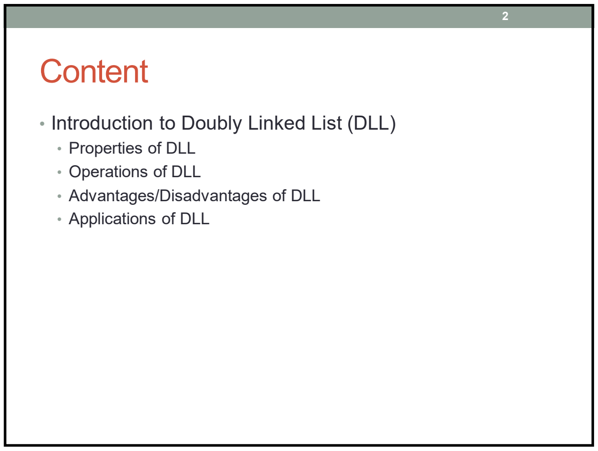


DATA STRUCTURES

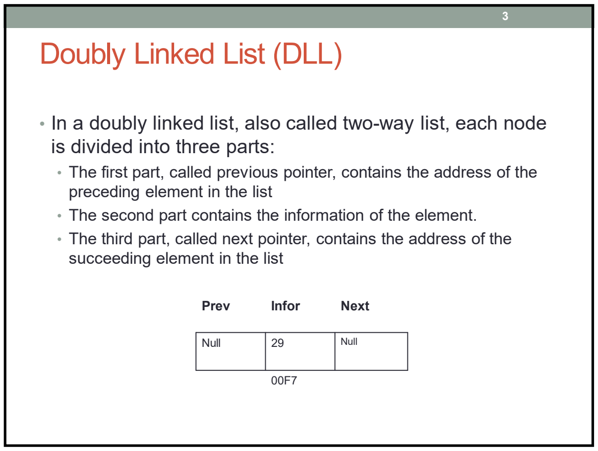
Doubly Linked List / Two-Way Linked List

By
Zainab Malik



Content

- Introduction to Doubly Linked List (DLL)
  - Properties of DLL
  - Operations of DLL
  - Advantages/Disadvantages of DLL
  - Applications of DLL

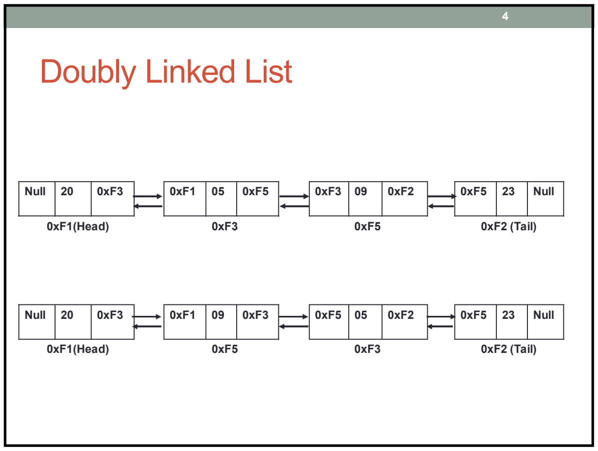


Doubly Linked List (DLL)

- In a doubly linked list, also called two-way list, each node is divided into three parts:
  - The first part, called previous pointer, contains the address of the preceding element in the list
  - The second part contains the information of the element.
  - The third part, called next pointer, contains the address of the succeeding element in the list

Prev	Infor	Next
Null	29	Null

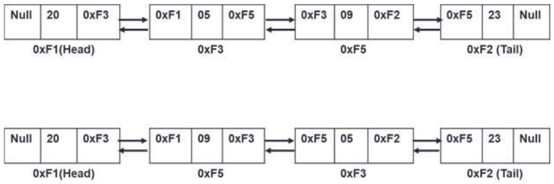
00F7



Doubly Linked List

Null	20	0xF3	⇄	0xF1	05	0xF5	⇄	0xF3	09	0xF2	⇄	0xF5	23	Null
0xF1(Head)				0xF3				0xF5				0xF2 (Tail)		

Null	20	0xF3	⇄	0xF1	09	0xF3	⇄	0xF5	05	0xF2	⇄	0xF5	23	Null
0xF1(Head)				0xF5				0xF3				0xF2 (Tail)		



Operations on Doubly Linked List (DLL)

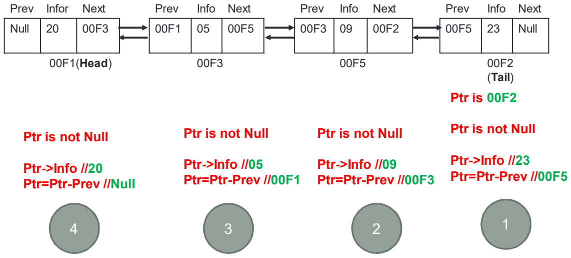
- Traversing
- Searching
- Insertion
  - AddToHead
  - AddToTail
  - AddAfterGivenElement
  - AddBeforeGivenElement
- Removal
  - RemoveFromHead
  - RemoveFromTail
  - RemoveGivenItem

Traversing & Searching in DLL

- Traversing and searching procedure is exactly same as in SLL
- However, we can traverse in reverse direction too i.e. from tail to head
  - ReverseTraversal()
    - Case1: list is empty
      - Display message “No element to traverse”
    - Case 2:

```
1. Set ptr at tail
2. Repeat steps 3-4 while (ptr!=0)
3.     Print Info of current ptr
4.     Move ptr to its prevPus node (ptr=ptr->prev)
5. Exit
```

reverseTraversing(list)



Ptr is 00F2

Ptr is not Null

Ptr->Info //20
Ptr=Ptr-Prev //Null

Ptr is not Null

Ptr->Info //05
Ptr=Ptr-Prev //00F1

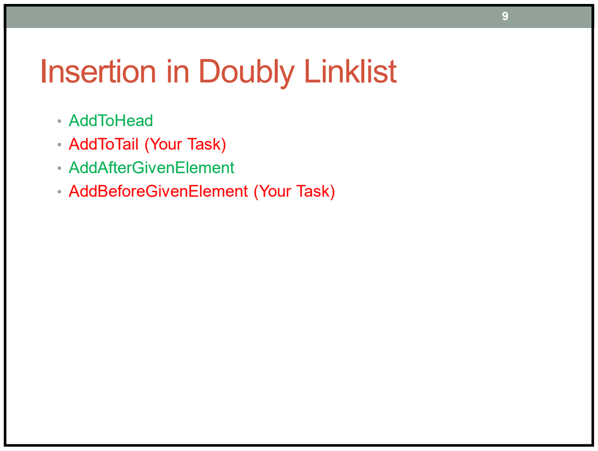
Ptr is not Null

Ptr->Info //09
Ptr=Ptr-Prev //00F3

Ptr is not Null

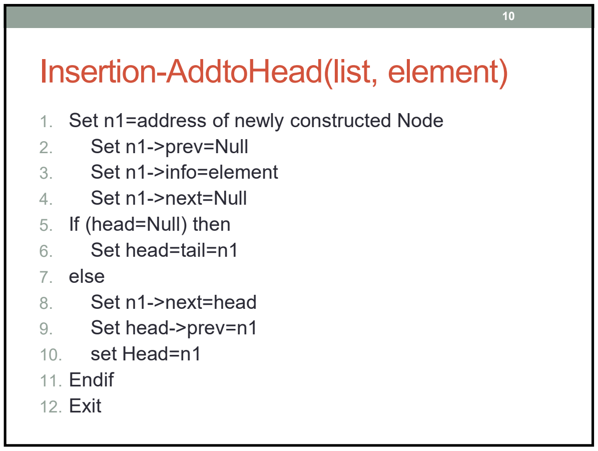
Ptr->Info //23
Ptr=Ptr-Prev //00F5

Ptr is Null (Exit)



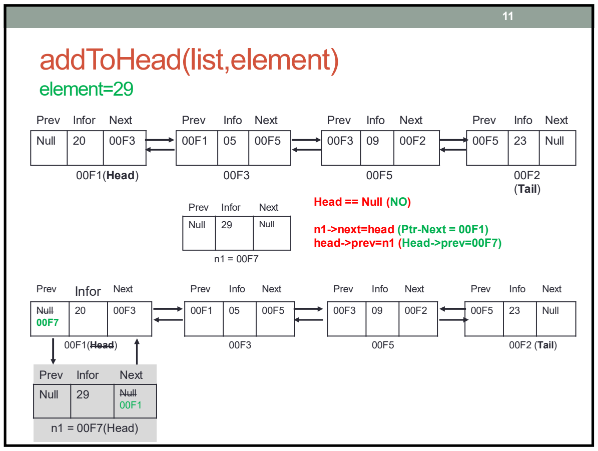
Insertion in Doubly Linklist

- AddToHead
- AddToTail (Your Task)
- AddAfterGivenElement
- AddBeforeGivenElement (Your Task)



Insertion-AddtoHead(list, element)

```
Set n1=address of newly constructed Node
    Set n1->prev=Null
    Set n1->info=element
    Set n1->next=Null
If (head=Null) then
    Set head=tail=n1
else
    Set n1->next=head
    Set head->prev=n1
    set Head=n1
Endif
Exit
```



addToHead(list,element)

element=29

Prev	Infor	Next
Null	20	00F3

00F1(**Head**)

Prev	Info	Next
00F1	05	00F5

00F3

Prev	Info	Next
00F3	09	00F2

00F5

Prev	Info	Next
00F5	23	Null

00F2 (**Tail**)

Prev	Infor	Next
Null	29	Null

n1 = 00F7

Head == Null (NO)

n1->next=head (Ptr-Next = 00F1)
head->prev=n1 (Head->prev=00F7)

Prev	Infor	Next
~~Null~~ **00F7**	20	00F3

00F1(~~**Head**~~)

Prev	Info	Next
00F1	05	00F5

00F3

Prev	Info	Next
00F3	09	00F2

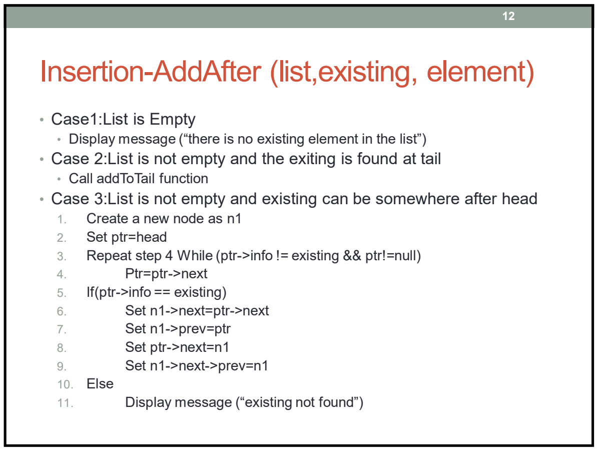
00F5

Prev	Info	Next
00F5	23	Null

00F2 (**Tail**)

Prev	Infor	Next
Null	29	~~Null~~ 00F1

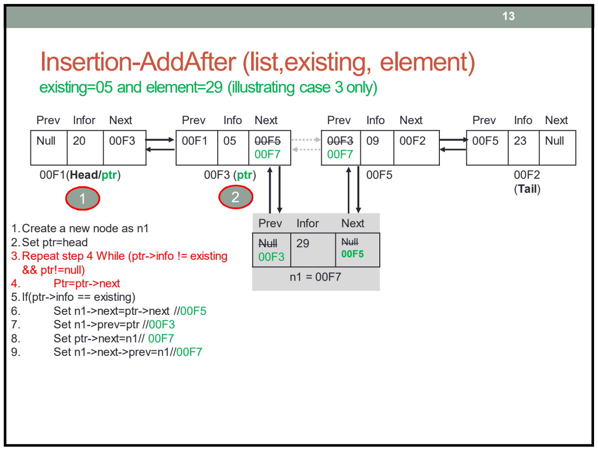
n1 = 00F7(Head)



Insertion-AddAfter (list,existing, element)

- Case1:List is Empty
  - Display message ("there is no existing element in the list")
- Case 2:List is not empty and the exiting is found at tail
  - Call addToTail function
- Case 3:List is not empty and existing can be somewhere after head

```
1.  Create a new node as n1
2.  Set ptr=head
3.  Repeat step 4 While (ptr->info != existing && ptr!=null)
4.          Ptr=ptr->next
5.  If(ptr->info == existing)
6.          Set n1->next=ptr->next
7.          Set n1->prev=ptr
8.          Set ptr->next=n1
9.          Set n1->next->prev=n1
10. Else
11.         Display message ("existing not found")
```



13

Insertion-AddAfter (list,existing, element)

existing=05 and element=29 (illustrating case 3 only)

Prev	Infor	Next
Null	20	00F3

00F1(**Head**/**ptr**)

Prev	Info	Next
00F1	05	~~00F5~~ 00F7

00F3 (**ptr**)

Prev	Info	Next
~~00F3~~ 00F7	09	00F2

00F5

Prev	Info	Next
00F5	23	Null

00F2 (**Tail**)

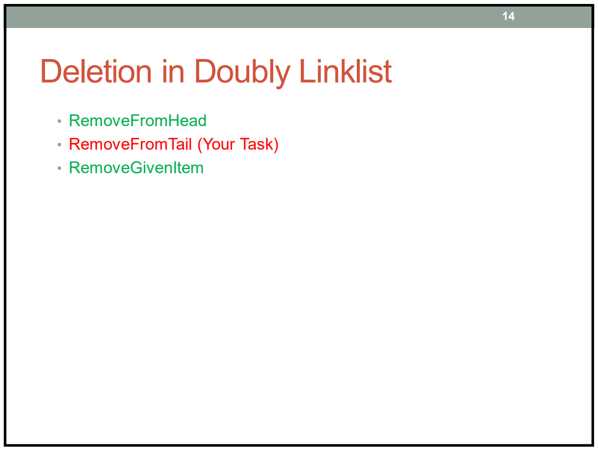
1

2

Prev	Infor	Next
~~Null~~ 00F3	29	~~Null~~ **00F5**

n1 = 00F7

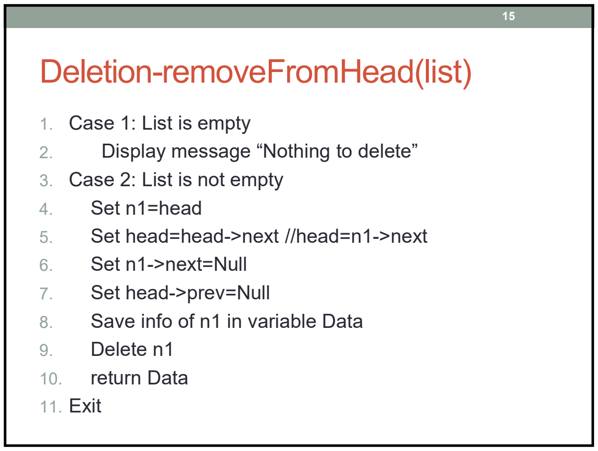
```
1. Create a new node as n1
2. Set ptr=head
3. Repeat step 4 While (ptr->info != existing
   && ptr!=null)
4.         Ptr=ptr->next
5. If(ptr->info == existing)
6.         Set n1->next=ptr->next //00F5
7.         Set n1->prev=ptr //00F3
8.         Set ptr->next=n1// 00F7
9.         Set n1->next->prev=n1//00F7
```



14

Deletion in Doubly Linklist

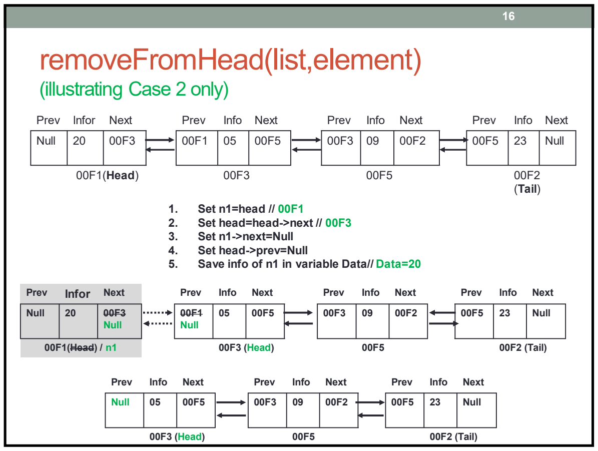
- RemoveFromHead
- RemoveFromTail (Your Task)
- RemoveGivenItem



15

Deletion-removeFromHead(list)

1. Case 1: List is empty
2. Display message “Nothing to delete”
3. Case 2: List is not empty
4. Set n1=head
5. Set head=head->next //head=n1->next
6. Set n1->next=Null
7. Set head->prev=Null
8. Save info of n1 in variable Data
9. Delete n1
10. return Data
11. Exit



16

removeFromHead(list,element)

(illustrating Case 2 only)

Prev	Infor	Next
Null	20	00F3

00F1(**Head**)

Prev	Info	Next
00F1	05	00F5

00F3

Prev	Info	Next
00F3	09	00F2

00F5

Prev	Info	Next
00F5	23	Null

00F2 (**Tail**)

```
1.   Set n1=head // 00F1
2.   Set head=head->next // 00F3
3.   Set n1->next=Null
4.   Set head->prev=Null
5.   Save info of n1 in variable Data// Data=20
```

Prev	Infor	Next
Null	20	~~00F3~~ Null

00F1(~~Head~~) / n1

Prev	Info	Next
~~00F1~~ Null	05	00F5

00F3 (Head)

Prev	Info	Next
00F3	09	00F2

00F5

Prev	Info	Next
00F5	23	Null

00F2 (Tail)

Prev	Info	Next
Null	05	00F5

00F3 (Head)

Prev	Info	Next
00F3	09	00F2

00F5

Prev	Info	Next
00F5	23	Null

00F2 (Tail)

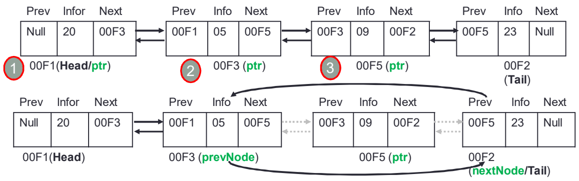
Deletion-RemoveGivenItem(list, item)

- Case 1:List is Empty
  - Display message (“there is no existing element in the list to remove”)
- Case 2: Element found at head
  - Call removeFromHead
- Case 3: Element found at tail
  - Call removeFromTail
- Case 4: Element may be Somewhere in between

```
1.  Set ptr=head
2.  Repeat step 3 while (ptr->next !=Null && ptr->info != item)
3.      Ptr=ptr->next
4.  If (ptr->info=item)
5.      Set nextNode=ptr->next
6.      Set prevNode=ptr->prev
7.      prevNode->next=nextNode
8.      nextNode->prev=prevNode
9.      Set ptr->prev=Null
10.     Set ptr->next=Null
11.     Delete ptr
```

Deletion-RemoveGivenItem(list, item)

(item= 09 illustrating Case 3 only)



```
1.  Set ptr=head
2.  Repeat step 3 while (ptr->next !=Null && ptr->info != item)
3.      Ptr=ptr->next
4.  If (ptr->info=item)
5.      Set nextNode=ptr->next //00F2
6.      Set prevNode=ptr->prev // 00F3
7.      prevNode->next=nextNode // 00F2
8.      nextNode->prev=prevNode //00F3
9.      Set ptr->prev=Null
10.     Set ptr->next=Null
11. Delete ptr // delete 00F5
```

Advantages of Singular LinkedList

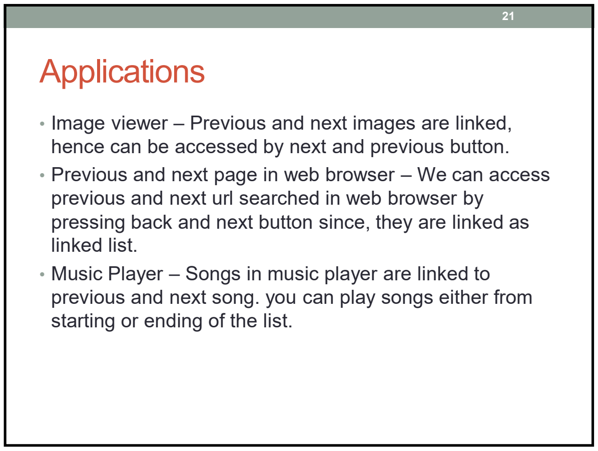
- **ADVANTAGE** :-

1. We can traverse in both direction i.e. from starting to end & as well as from end to starting.
2. It is easy to reverse the linked list.
3. If we are at a node, then we can go at any node. But in linked list, it is not possible to reach the previous node.

Disadvantages of Singular LinkedList

DISADVANTAGE :-

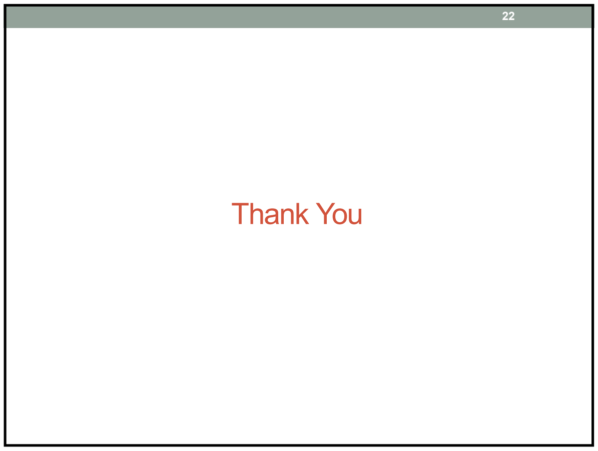
- It requires more space per space per node because extra field is required for pointer to previous node.
- Insertion and Deletion take more time than linear linked list because more pointer operations are required than linear linked list.



21

Applications

- Image viewer – Previous and next images are linked, hence can be accessed by next and previous button.
- Previous and next page in web browser – We can access previous and next url searched in web browser by pressing back and next button since, they are linked as linked list.
- Music Player – Songs in music player are linked to previous and next song. you can play songs either from starting or ending of the list.



22

Thank You