Data Structures - Lecture 11 PPT
KCS301
3rd Sem CSE

Vikas Kumar
Associate Professor
Department of Computer Science & Engineering
Moradabad Institute of Technology, Moradabad
(Affiliated to APJAKTU, Lucknow)

Key Learnings of Previous Lecture?

- We have discussed the process and implementation of following operations related to Singly Linked List:
  - Creation of Linked List
  - Traversal of a Linked List
  - Insertion of an element in Linked List
    - In the beginning
    - In the middle
    - At the last
  - Deletion of an element from linked List
    - Deletion of 1st node
    - Deletion from middle
    - Deletion of last node

Today's Agenda?

- Doubly Linked List
- Operations on Doubly Linked List
  - Traversal operation
  - Insertion operation
  - Deletion operation

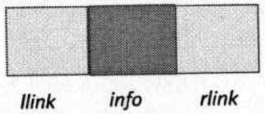
Singly Linked List - Observations

- In deletion operation we need the address of element previous to the element to be deleted.
- For finding the address of any node previous to a given node
  - We need to traverse upto that element O(n)

    or
  - We need move in backward direction O(1)
- In Singly linked list, we can traverse only in the direction of links i.e. forward direction but not in backward direction.
- To overcome this deficiency we will use a variation of linked list known as Doubly Linked List.

Doubly Linked List

- A Doubly Linked List is a variation of linked list in which we can traverse the list in both the directions.
- Thus, each node is divided into Three parts:
    - *Info* part contains the information of element, and
    - *llink* contains the address of previous element
    - *rlink* contains the address of next element



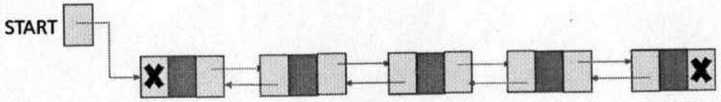
Structure of a node

In C, a node in doubly linked list can be implemented through a structure. If *info* field is of type integer then structure may be defined as follows:

```
struct dnode
{
  int info;
  struct dnode *llink , *rlink ;
}
```

Doubly Linked List

- Schematic diagram of a doubly linked list containing 5 nodes



- ***rlink* field of last node** and ***llink* field of first node** contain Null
- ***llink* field of all nodes except first** contains the address of previous node in list.
- ***rlink* field of all nodes except last** contains the address of next node in list.

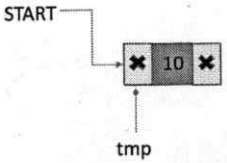
Creating a Doubly Linked List

```
struct dnode
 {
   int info;
   struct dnode *llink , *rlink;
 }

typedef struct dnode DNODE;
DNODE *start = NULL;
```

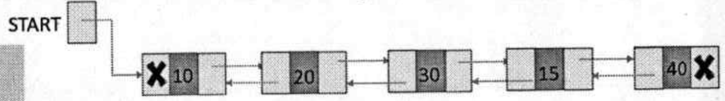
Create an Empty List

```
void createDoublylist ( int item )
{
  DNODE *tmp;

  tmp = (DNODE*) malloc (sizeof(DNODE));
  tmp→info = item;
  tmp→llink = NULL;
  tmp→rlink = NULL;
  start = tmp;
}
```

Adding 1st element to List



Traversal operation on Doubly Linked List

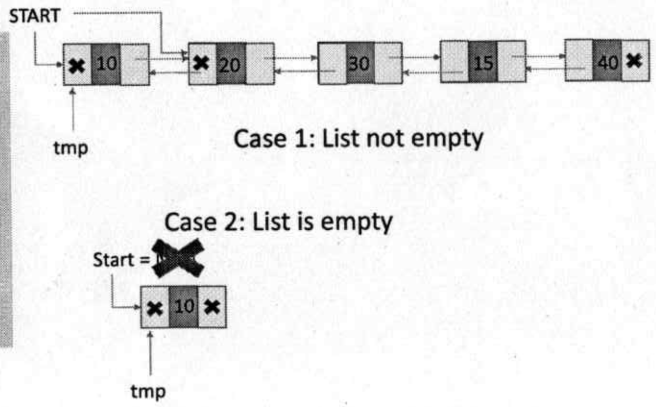


```
void traverseDoubly_forward()
{
  DNODE *q;

  if (start==NULL)
    {
      printf("List is empty");
      return;
    }
  q = start;
  printf("List is: ");
  while(q!=NULL)
    {
      printf("%d ",q→info);
      q = q→rlink;
    }
}
```

Director
Moradabad Institute of Technology
Ram Ganga Vihar, Phase-2
Moradabad

Insert at the beginning of Doubly Linked List

```
void insertbegDoubly( int item)
{
 DNODE *tmp;

 tmp = (DNODE*) malloc (sizeof(DNODE));
 tmp→info = item;
 tmp→llink = NULL;
 tmp→rlinkt = start;
 if (start != NULL)
     start→llink = tmp;
 start = tmp;
}
```

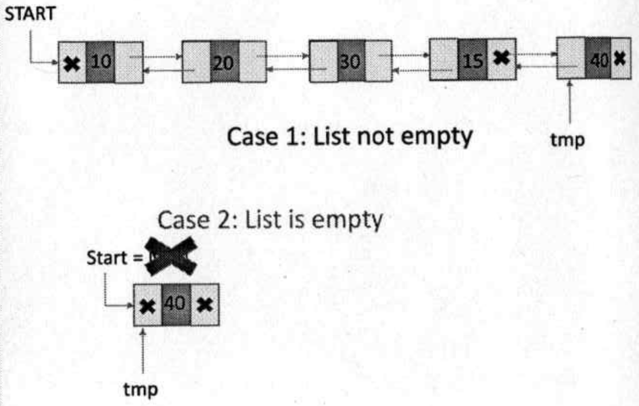


Case 1: List not empty

Case 2: List is empty

Insert at the end of Doubly Linked List

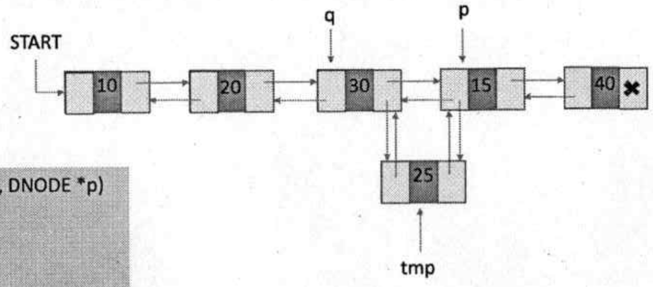
```
void insertlastDoubly( int item)
{
 DNODE *q , *tmp;

 tmp = (DNODE*) malloc (sizeof(DNODE));
 tmp→info = item;
 tmp→rlink = NULL;
 if (start == NULL)
   {
    tmp→llink = NULL;
    start = tmp;
   }
 else
   {
     q = start;
     while(q→rlink != NULL)
          q = q→rlink;
     tmp→llink = q;
     q→rlink = tmp;
   }
}
```



Case 1: List not empty

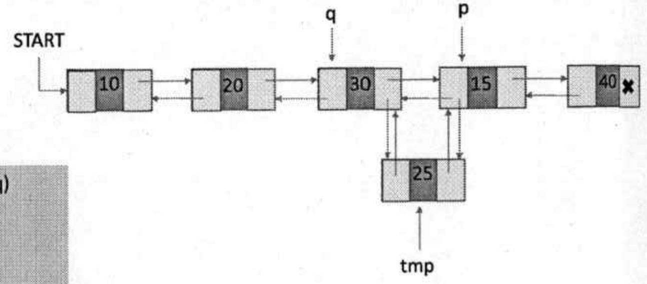
Case 2: List is empty

Insert between two consecutive nodes of Doubly Linked List



```
void insertmidDoubly( int item , DNODE *q , DNODE *p)
{
 DNODE *tmp;

 tmp = (DNODE*) malloc (sizeof(DNODE));
 tmp→info = item;

 tmp→llink = q;
 tmp→rlink = p;
 q→rlink = tmp;
 p→llink = tmp;
}
```

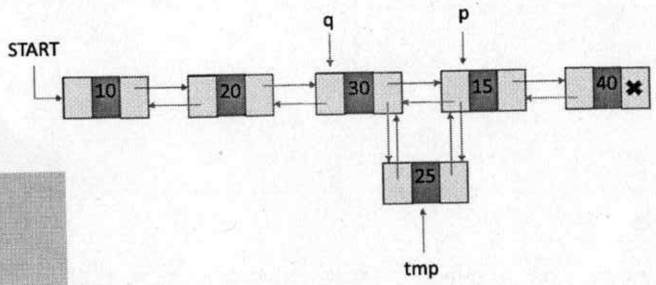
Insert after a given node of Doubly Linked List



```
void insertafterDoubly( int item , DNODE *q)
{
 DNODE *tmp , *p;

 p = q→rlink;
 tmp = (DNODE*) malloc (sizeof(DNODE));
 tmp→info = item;

 tmp→llink = q;
 tmp→rlink = p;
 q→rlink = tmp;
 p→llink = tmp;
}
```

Director
Moradabad Institute of Technology
Ram Ganga Vihar, Phase-2
Moradabad

Insert before a given node of Doubly Linked List



```
void insertafter( int item , DNODE *p)
{
 DNODE *tmp , *q;

 q = p→llink;
 tmp = (DNODE*) malloc (sizeof(DNODE));
 tmp→info = item;

 tmp→llink = q;
 tmp→rlink = p;
 q→rlink = tmp;
 p→llink = tmp;
}
```

General Insert function for Doubly Linked List covering all cases

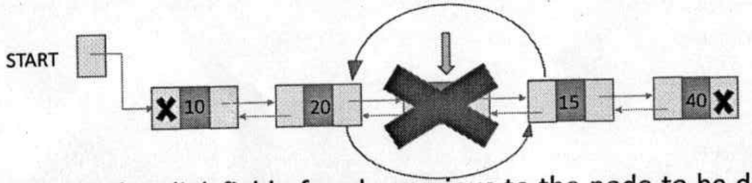
```
void insertDoubly( int item , DNODE *q , DNODE *p)
{
 DNODE *tmp;
  tmp = (DNODE*) malloc (sizeof(DNODE));
  tmp→info = item;
  tmp→llink = q;
  tmp→rlink = p;

  if (q == NULL) && (p == NULL) start = tmp; //Empty List
  if (q == NULL) && (p != NULL)  //Insert at beginning
   { p→llink = tmp;
    start = tmp;
   }
  if (q != NULL) && (p == NULL) q→rlink = tmp; //at end
  if (q != NULL) && (p != NULL) //between two nodes
   { q→rlink = tmp;
     p→llink = tmp;
   }
}
```

Assumptions:

1. Both p and q equals NULL shows that list is empty
2. Only q equals NULL shows that we want to insert at beginning of li
3. Only p equals NULL shows that we want to insert at the end of list
4. If none of p and q equals NULL we want to insert between two givı nodes of list

Deletion in Doubly Linked List

- Consider following linked list, we wish to delete 3rd node containing 30



- Simply set the *rlink* field of node previous to the node to be deleted, equal to the *rlink* field of the node that is to be deleted
- Similarly set the *llink* field of node after the node to be deleted, equal to the *llink* field of the node that is to be deleted
- Observe that node to be deleted is no more part of the list
- Now we may deallocate the memory allocated to this detached node and remove it

Director
Moradabad Institute of Technology
Ram Ganga Vihar, Phase-2
Moradabad

Delete element at the beginning of Linked List

```
void deletebeg( )
{
 DNODE *tmp;

 if (start == NULL)
  { printf("List Empty...Invalid Deletion");
     return;
  }
 else
  { tmp = start;
     if (start→rlink == NULL)  start = NULL;
     else
      { start = start→rlink;
         start→llink = NULL;
      }
    printf("Deleted Element is: %d", tmp→info);
    free(tmp);
  }
}
```

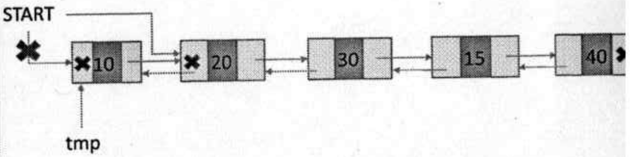
Case 1: List is Empty
Deletion is not possible

Case 2 and Case 3

Case 2: List is not empty having only one element

START = NULL
tmp

Case 3: List is not empty having more than one element



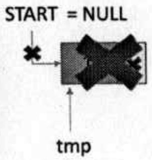
Delete last element of Linked List

```
void deletelast( )
{
 NODE *tmp , *locp;

 if (start == NULL)
   {
     printf("List Empty...Invalid Deletion");
     return;
   }
 tmp = start;
 while(tmp→next != NULL)
   {
     locp = tmp;
     tmp = tmp→next;
   }
 if (tmp == start)   start = NULL;
 else  locp→next = NULL;
 printf("Deleted Element is: %d", tmp→info);
 free(tmp);
}
```

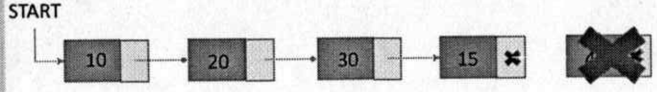
Case 1: List is Empty Deletion is not possible

Case 2 and Case 3

Case 2: List is not empty having only one element



Case 3: List is not empty having more than one element



Delete element specified by its value from Linked List

```
void delete_by_value( int data )
{
 NODE *tmp , *locp;

 if (start == NULL)
   {
    printf("List Empty...Invalid Deletion");
    return;
   }
-----------------------------------------------
 if (start→info == data)
  {
   tmp = start;
   start = start→next;
   free(tmp);
   return;
  }
```

Case 1: List is Empty Deletion not possible

Case 2: List not empty Data found in 1st node

```
locp = start;
while(locp→next→next != NULL)
  {
   if (locp→next→info == data)
    { tmp = locp→next;
      locp→next = tmp→next;
      free(tmp);
      return;
    }
   locp = locp→next;
  }
-----------------------------------------------
 if (locp→next→info == data)
  {
    tmp = locp→next;
    locp→next = NULL;
    free(tmp);
    return;
   }
 printf("Element %d not found in List." , data
}
```

Case 3: List not empt Data found after1st node but before last node

Case 4: List not empty contains more than one nodes and Data found in last node

General Delete function covering all cases

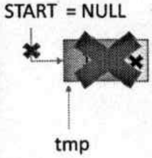
```
void delete( NODE *locp)
{
 NODE *tmp ;

 if (start == NULL)
   { printf("List Empty...Invalid Deletion");
     return;  }
-----------------------------------------------
 if (locp == NULL)
  { tmp = start;
    start = start→next;
    printf("Deleted Element is: %d", tmp→info);
    free(tmp);
    return; }
-----------------------------------------------
 tmp = locp→next;
 locp→next = tmp→next;
 printf("Deleted Element is: %d", tmp→info);
 free(tmp);
 return; }
```

Case 1: List is Empty Deletion is not possible

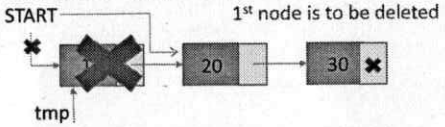
Case 2 Case 3

Case 4 & Case 5

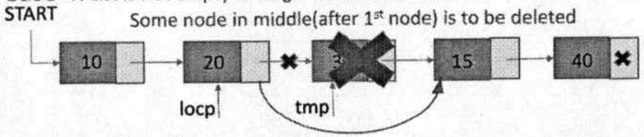
Case 2: List is not empty having only one element, 1st node be deleted



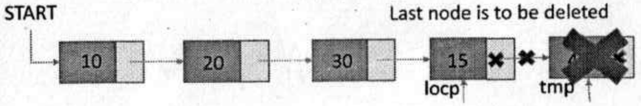
Case 3: List is not empty having more than one element 1st node is to be deleted



Case 4: List is not empty having more than one element Some node in middle(after 1st node) is to be deleted



Case 5: List is not empty having more than one element Last node is to be deleted



Concluding Remarks

- We have discussed the process and implementation of following operations related to Linked List:
  - Creation of Linked List
  - Traversal of a Linked List
  - Insertion of an element in Linked List
    - In the beginning
    - In the middle
    - At the last
  - Deletion of an element from linked List
    - Deletion of 1st node
    - Deletion from middle
    - Deletion of last node

Director
Moradabad Institute of Technology
Ram Ganga Vihar, Phase-2
Moradabad