UCS1302:
DATA STRUCTURES

Doubly Linked List

Session Meta Data

Author	**Dr. B. Bharathi**
Reviewer	
Version Number	1.2
Release Date	03 July 2019

Revision History

Revision Date	Details	Version no.
22 September 2017	1. New SSN template applied	1.2

Session Objectives

- To learn about doubly linked list ADT
- Implementation of doubly linked list

Session Outcomes

- At the end of this session, participants will be able to
  - Understand the concepts of doubly linked list
  - Implementation of doubly linked list ADT

Agenda

- Doubly linked list
- Implementation of doubly linked list operations

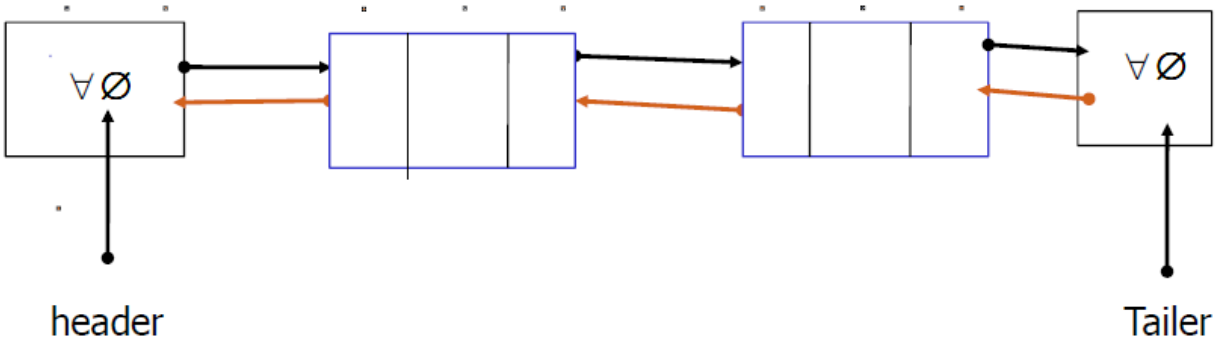
Doubly Linked List

Dr. B.Bharathi
SSNCE

July 3, 2019

Doubly Linked List

- Each node points to successor and the predecessor
- There are two NULL: at the first and last nodes in the list.
- Advantage: given a node, it is easy to visit its predecessor.
- Convenient to traverse lists backwards



Doubly Linked List

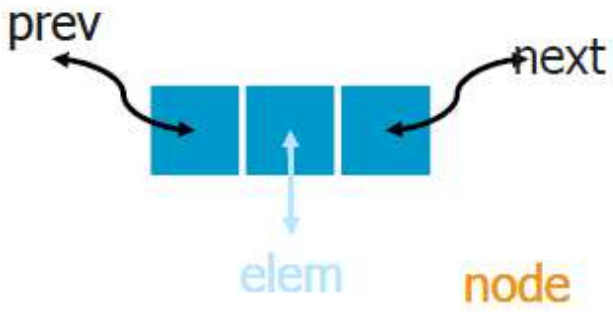
- Quick update operations:
  - insertions, deletions at *both ends (head and tail), and* also at the middle of the list.

- A node in a doubly-linked list store two references:
  - A ***next link; that points to the next node in the list, and***
  - A ***prev link; that points to the previous node in the list.***

Structure Node

```
typedef struct Node *PtrToNode;
typedef PtrToNode List;
typedef PtrToNode Position;

struct Node
{
    int Element;
    Position   Next;
    Position Prev;
};
```



Trailer and Header Nodes

- Two special nodes have been added at both ends of the doubly-linked list.
- Head and tail are dummy nodes, also called sentinels, do not store any data elements.
- Head: header sentinel has a null-prev reference (link).
- Tail: trailer sentinel has a null-next reference (link).

Trailer and Header Nodes

Special trailer and header nodes

```
PtrToNode create()
{
  PtrToNode h,t;
 h= malloc(( sizeof( struct Node ) ));
t=malloc(( sizeof( struct Node ) ));
h->next=t;
t->prev=h;
return(h);
}
```

header Tailer

Adding the linked list at the begining

```
void addbeg(List H, int X)
{
    Position Temp,P;
    Temp=malloc( sizeof( struct Node ) );
    Temp->Element = X;
    Temp->Next = H->Next;
    Temp->Prev = H;
    H->Next->Prev=Temp;
    H->Next=Temp;
}
```

Adding the linked list at the End

```
void addend(List T, int X,List H)
{
    Position Temp,P;
    Temp=malloc( sizeof( struct Node ) );
    Temp->Element = X;
    Temp->Next = T;
    Temp->Prev = T->Prev;
    T->Prev->Next=Temp;
    T->Prev=Temp;
}
```

Find the position of the list

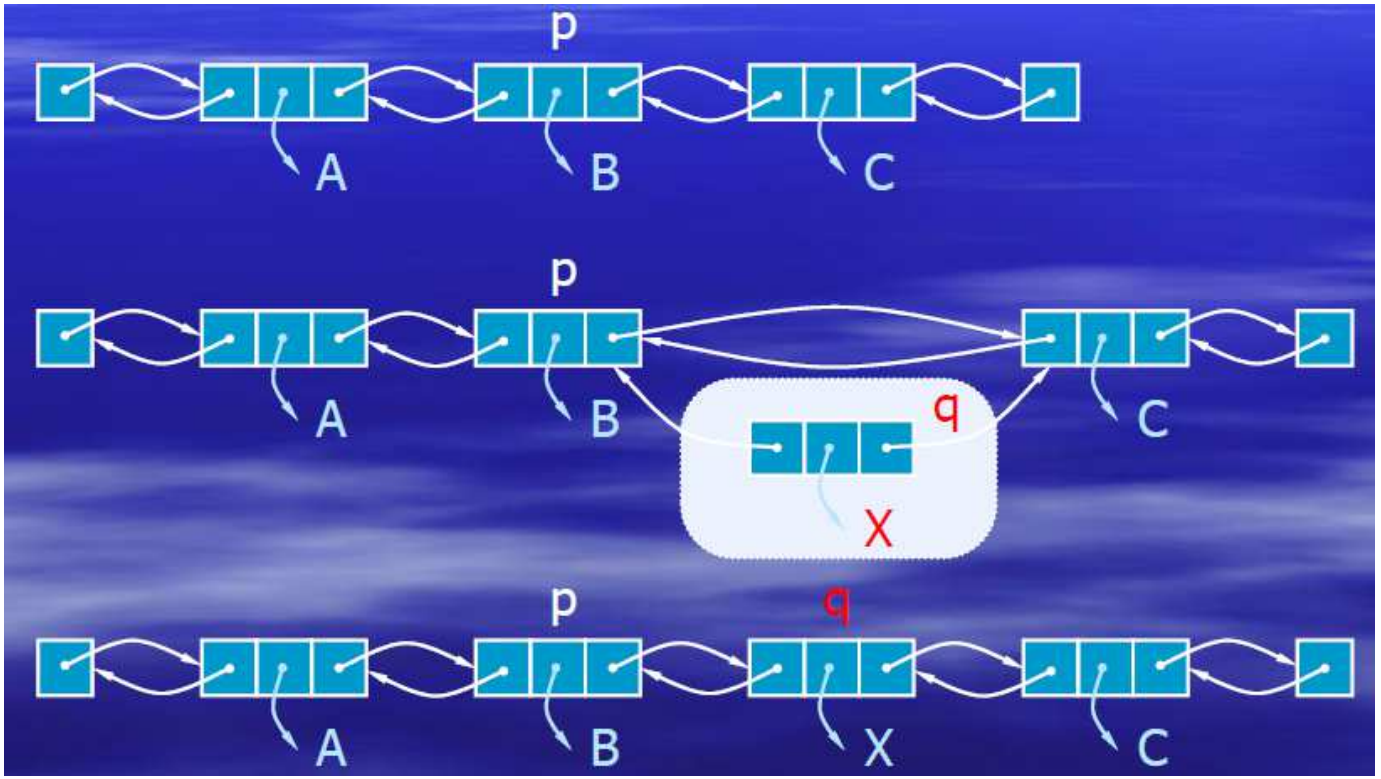
```
Position Find( int X, List L )
{
     int i=1;
     Position P;
    P = L->Next;
    while( P != NULL && P->Element != X )
    {
        P = P->Next;
         i++;
    }
    printf("number in position %d",i);
    return P;
}
```

Insertion of the new node after position p

```
Insert (int X,position P,list h)    //X parameter
{      P=P->Prev;                   //not needed
     Temp = malloc( sizeof( struct Node ) );
     Temp->Element = X;
     Temp->Next = P->Next;
     Temp->Prev = P;
     Temp->Next->Prev=Temp; //can also be P->Next->Prev =
Temp
     P->Next=Temp;
}
```

Insertion

- We visualize operation AddAfter(p, X), which returns position q

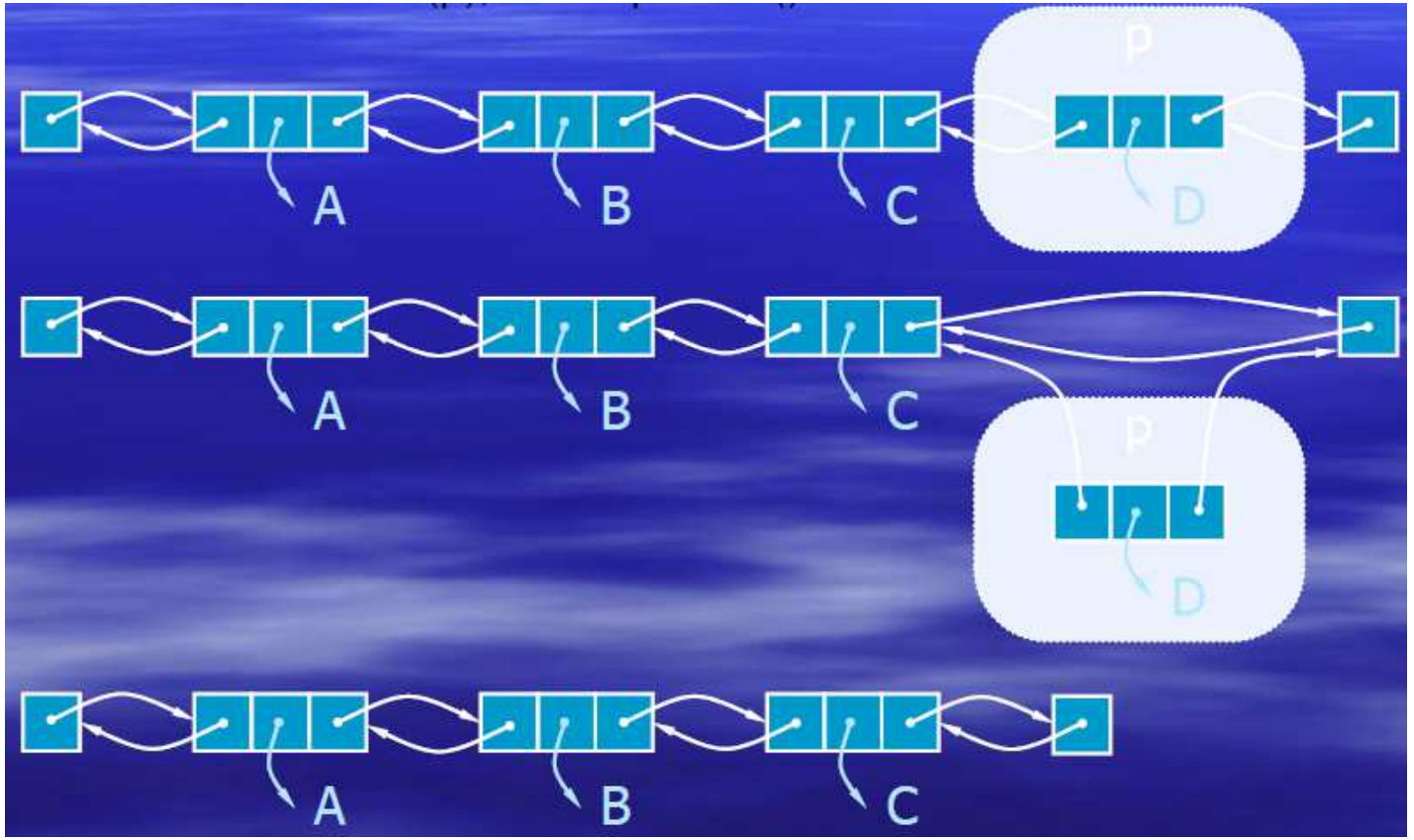


Deletion of the node at any position p

```
void delete(position p, int data) //deleting p
   if(p==NULL) //changed as p
         {
      printf("Element %d is not present in the
   list\n",data);
          return;
         }
   temp=p;
 temp->next->prev = temp->prev;
 temp->prev->next = temp->next
   free(temp);
         return;
}
```

Deletion

- We visualize remove(p), where p = last()



To find it is empty

```
int IsEmpty( List L )
{
    return L->Next == NULL;
}
```

Summary

- Doubly linked list
- Doubly linked list operations