Data Structures

Topics Covered :

1. Overview & Why
2. Classification of Data structures
3. Array & Operations
4. Linked List & Opertions
5. Doubly Linked List & Operations
6. Circular Linked List
7. Stack & Implementation
8. Queue & Implementation

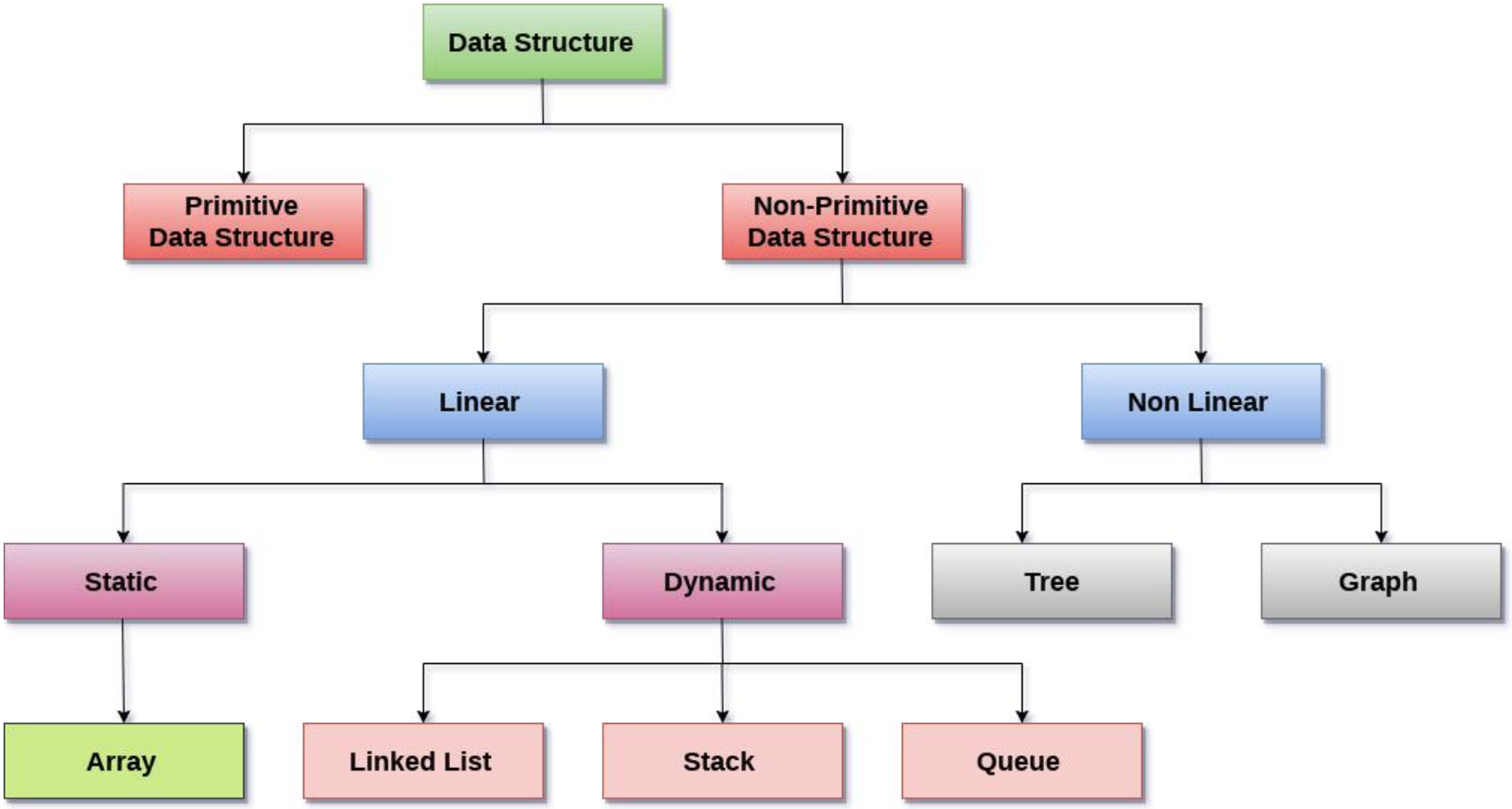
By,

Jishnu T U
C-DAC Trainer

Overview & Why ?

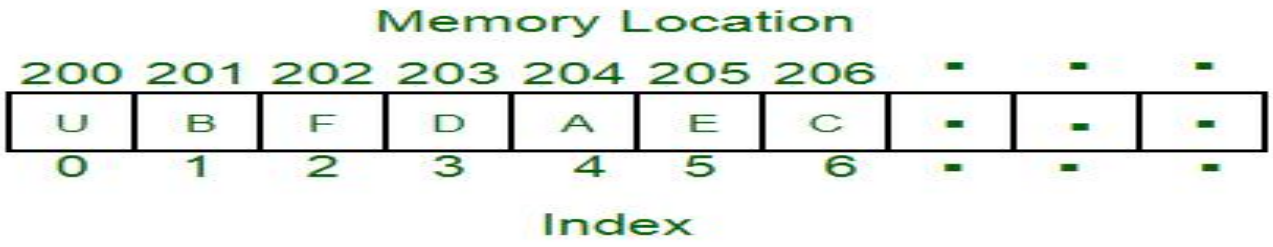
- Way of organizing data
- Reduce the space and time complexities of different tasks.
- High speed in processing
- Efficiency of a program
- Reusability and Abstraction

Data Structure Classification



Array

- Linear data structure
- Elements are stored at contiguous memory locations.
- Store multiple items of the same type together
- Advantages :
  - Random access is allowed
  - Ease of insertion and in Accesss
- Drawbacks:
  - Static memory allocation
  - Deletion requires Shifting of elements



Array - Accessing Element

- Accessed using an index number
- First element is numbered 0, so that the indices in an array of 10 elements run from 0 to 9.
- Complexity : O(1)

```
temp = myArray[3];
```

Array - Insertion of Element

- Unsorted
  - Keep track of number of elements in array - nElts
  - Insert using index and Increment the tracking object
  - Complexity : O(1)
- Sorted
  - Requires Searching for the postion
  - Shifting the elements to make the specfied postion to be vacant.
  - Complexity : $O(N^2)$

```
arr[0] = 77

arr[nElts] = 12
nElts = nElts +1
```

Deletion of Element

- Begins with a search for the specified item
- When we find it, we move all the items with higher index values down one element to fill in the “hole” left by the deleted element
- decrement nElems

```
for(j=0; j<nElems; j++) // look for it
            if(arr[j]== eltToBeDeleted){
            for(int k=j; k<nElems; k++) // higher ones down
                        arr[k] = arr[k+1];
            nElems--;               // decrement size
            break;
            }
 }
```

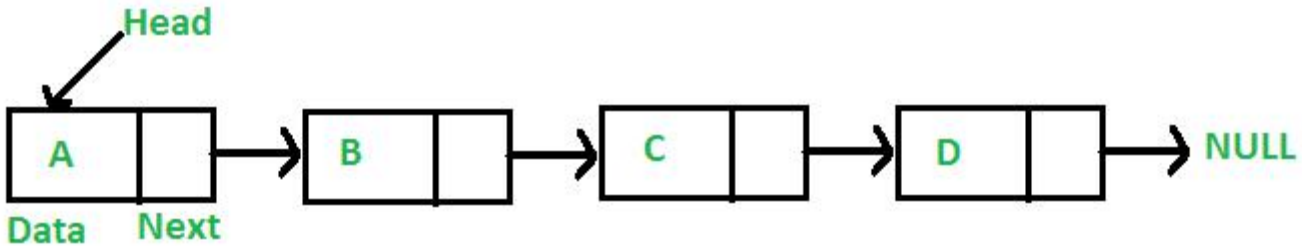
Linked list

- Linear data structure
- Elements are not stored at contiguous memory locations.
- Elements are linked using pointers
- Advantages :
  - Dynamic size (dynamic memory allocation)
  - Ease of insertion/deletion over unsorted approach
- Drawbacks:
  - Random access is not allowed
  - Extra memory space for a pointer

Singly Linked List

```
class Node
{
   public int data;
   public Node *next;
};
```

- Node
  - Each element in the List
  - consists of at least two parts:
    - Data
    - Pointer (Or Reference) to the next node
  - Tracking Element : Head/First
- Linked List



Linked List: Accessing Element

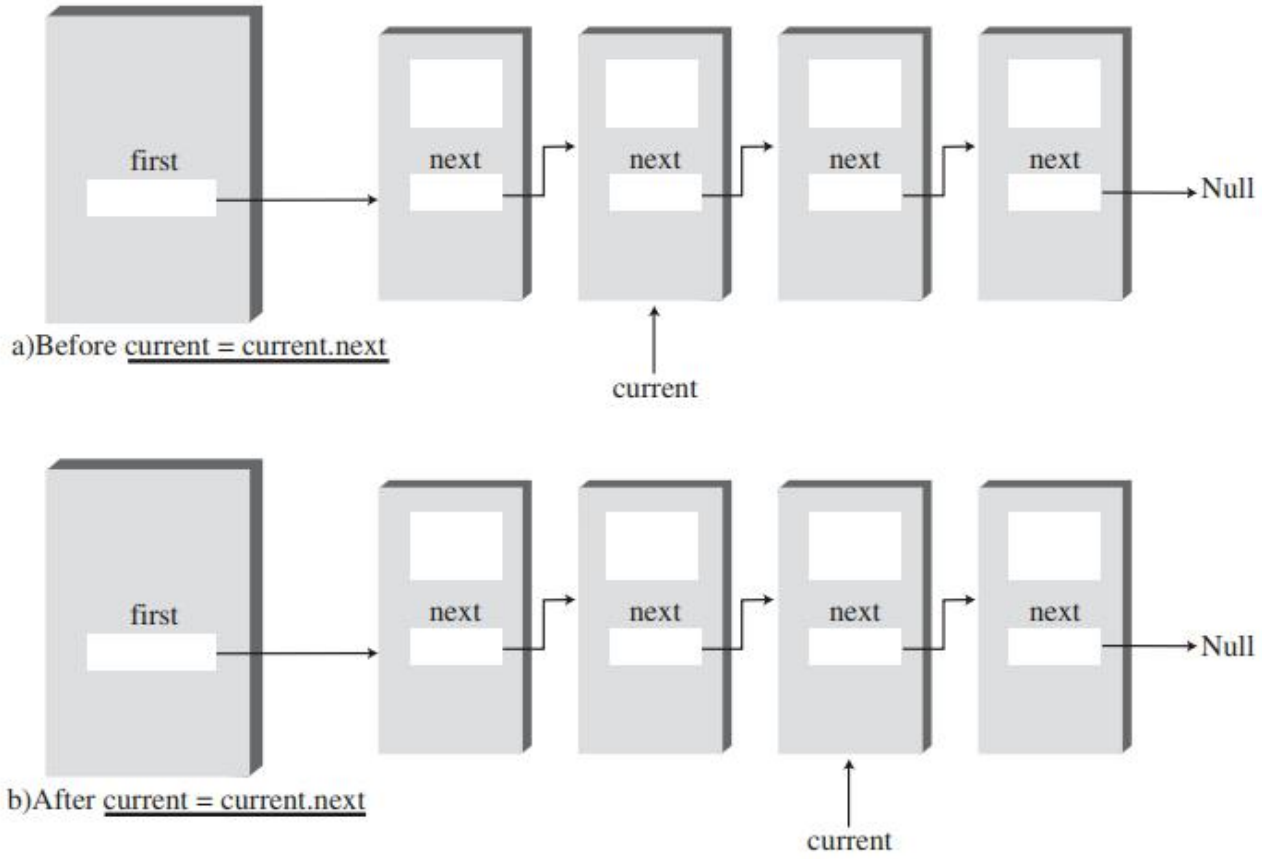


FIGURE 5.7 Stepping along the list.

Linked List:Insertion of Element

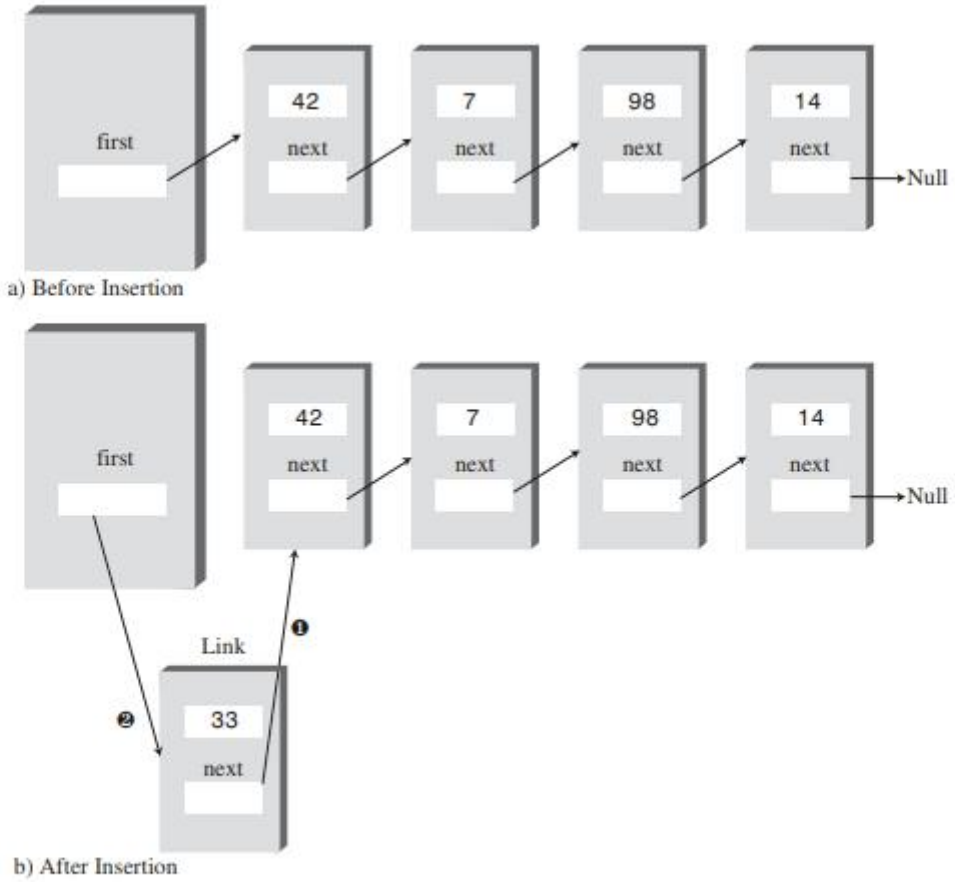


FIGURE 5.5 Inserting a new link.

Linked List:Deletion of Element

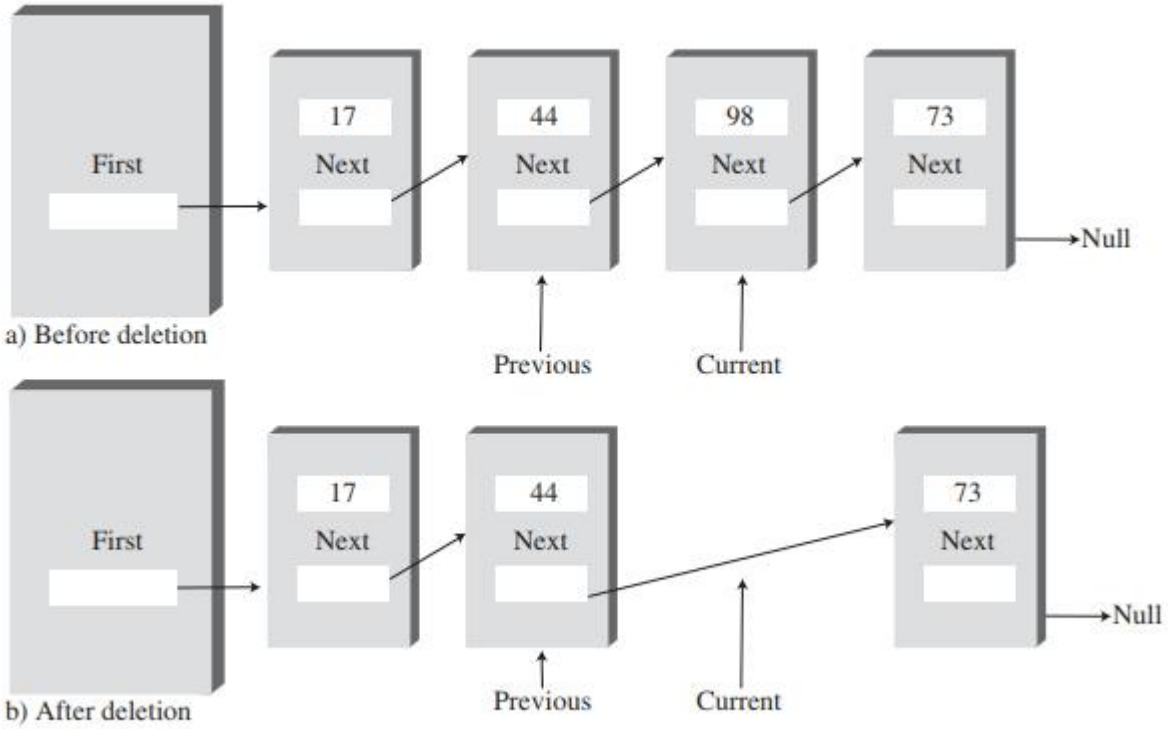
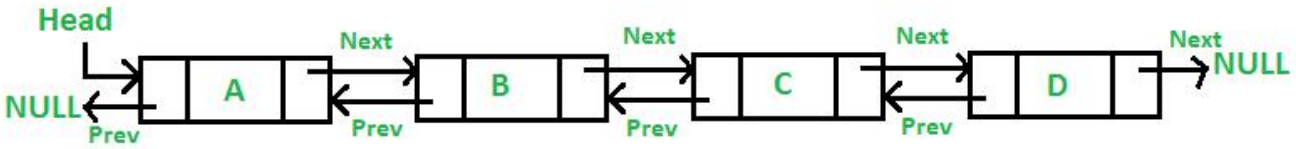


FIGURE 5.8 Deleting a specified link.

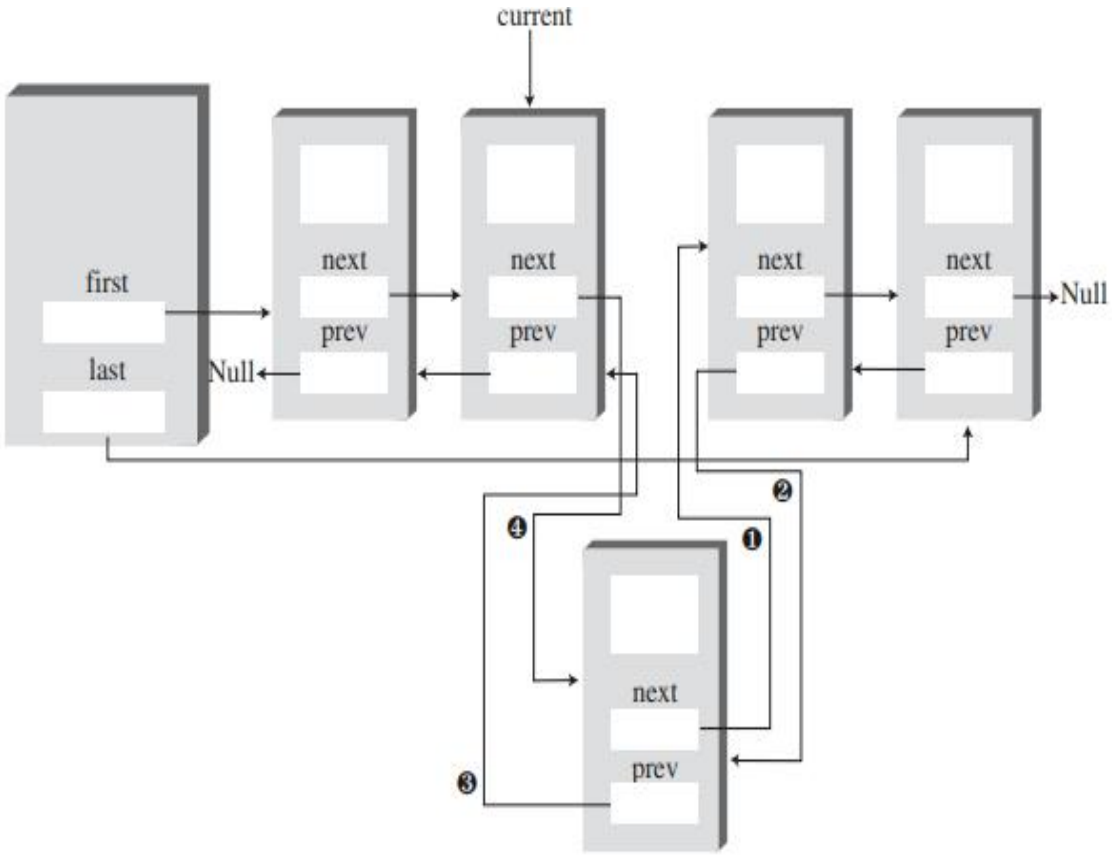
Doubly Linked List

```
class Node
{
   public int data;
   public Node *next;
   public Node *prev;
};
```

- Node
  - Each element in the List
  - consists of at least three parts:
    - Data
    - Pointer (Or Reference) to the next node
    - Pointer Or Reference) to the previous node
  - Tracking Element : Head/First and Last/Tail
- Linked List



Doubly Linked List:Insertion of Element



Doubly Linked List:Deletion of Element

```
current.previous.next = current.next;
current.next.previous = current.previous;
```

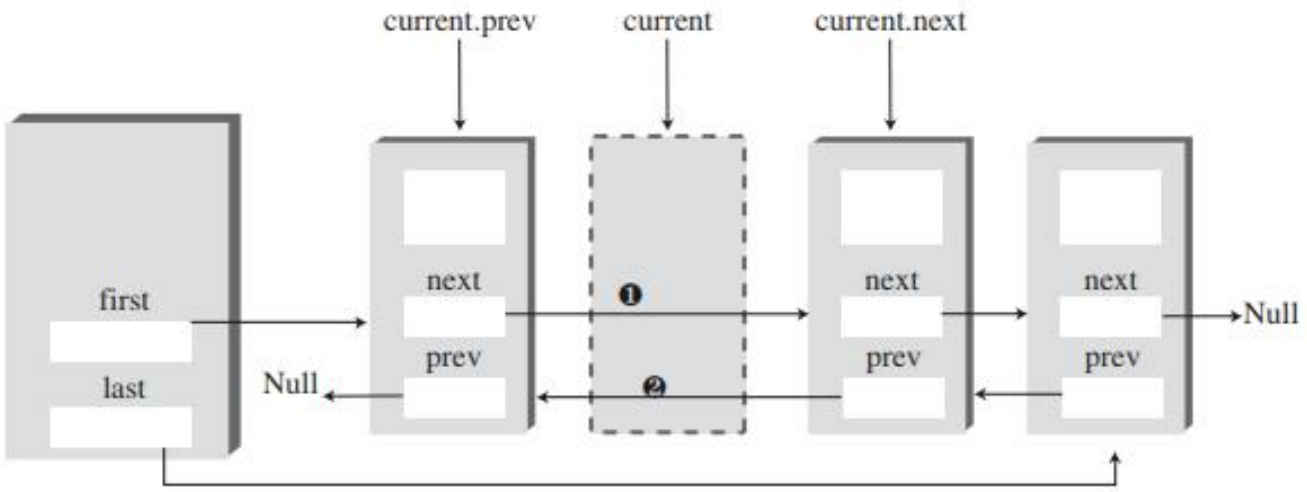
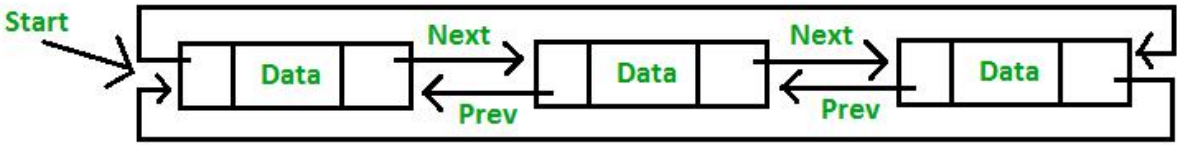


FIGURE 5.16 Deleting an arbitrary link.

Circular Linked List

```
class Node
{
   public int data;
   public Node *next;
   public Node *prev;
};
```

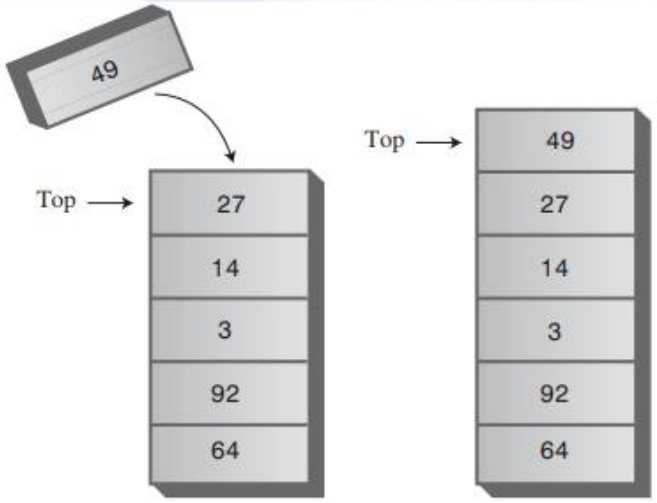
- Node
  - consists of at least three parts:
    - Data
    - Pointer (Or Reference) to the next node
    - Pointer Or Reference) to the previous node
  - Two consecutive elements are linked or connected by previous and next pointer
  - Last node points to first node by next pointer
  - First node points to last node by previous pointer
- Linked List



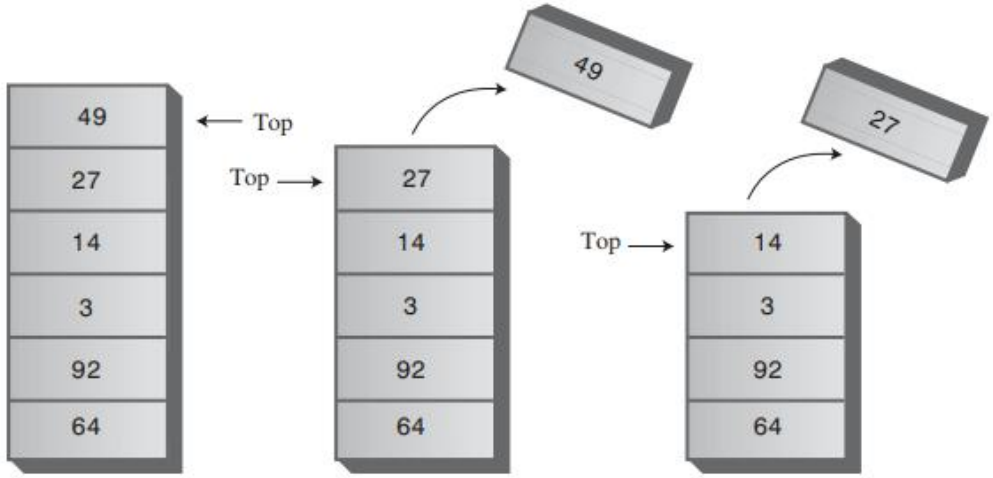
Stack

- Linear data structure
- New element is added at one end and (top) an element is removed from that end only.
- LIFO(Last In First Out) or FILO(First In Last Out).
- Example : plates stacked over one another
- Can be Implemented using Array or Linked List
- Tracking Element : Top
- Application :
  - Processing of subroutine calls and returns
  - reversing a string
  - backtracking - Eg :- Maze Game

Stack Example



New item pushed on stack



Two items popped from stack

Implementation

```
class StackX
{
        private int maxSize; // size of stack array
        private long[] stackArray;
        private int top; // top of stack
--------------------------------------------------------
public void StackX(int s) // constructor
{
        maxSize = s; // set array size
        stackArray = new long[maxSize]; // create array
        top = -1; // no items yet
}
--------------------------------------------------------

public void push(long j) // put item on top of stack
{
stackArray[++top] = j; // increment top, insert item
}
```

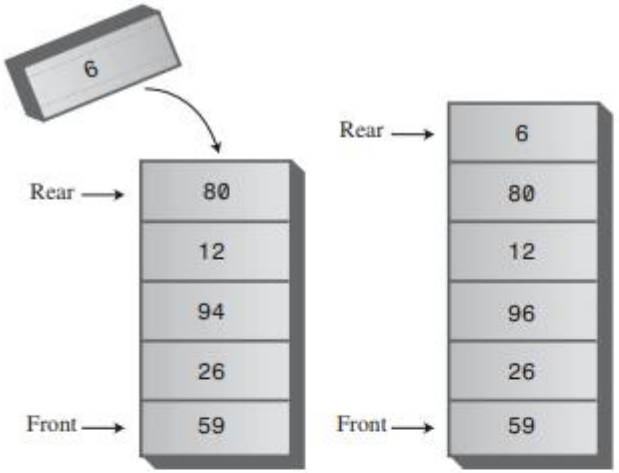
Cont.

```
public long pop() // take item from top of stack
{
return stackArray[top--]; // access item, decrement top
}
-------------------------------------------------
public long peek() // peek at top of stack
{
return stackArray[top];
}
-------------------------------------------------------
public boolean isEmpty() // true if stack is empty
{
return (top == -1);
}
-------------------------------------------------------
public boolean isFull() // true if stack is full
{
return (top == maxSize-1);
}
-------------------------------------------------------
} // end class StackX
```

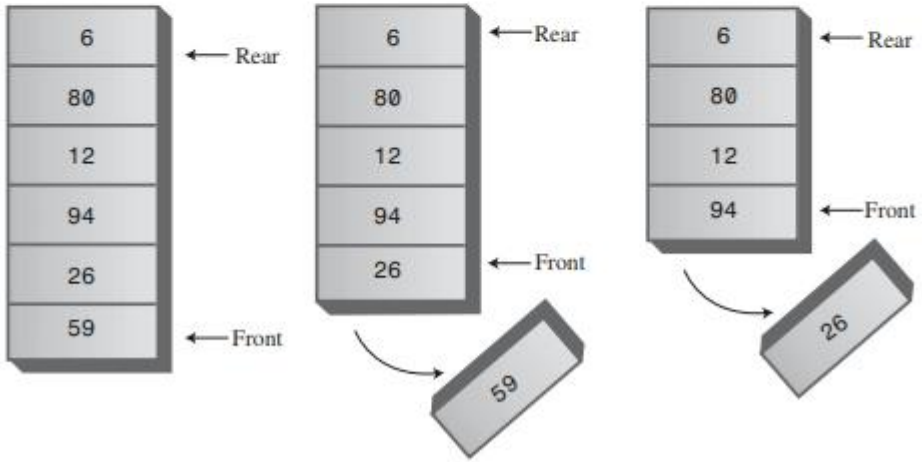
Queue

- Linear data structure
- New element is added at one end and (rear) an element is removed other end (front).
- First In First Out (FIFO)..
- Example : Queue for Customer Service
- Can be Implemented using Array or Linked List
- Tracking Element : Rear and Front
- Applications :
  - Time sharing system
  - Load Balancing system- Eg:- AWS-SQS, Hold for Tech Support
  - Print spooling,

Queue Example



New item inserted at rear of queue



Two items removed from front of queue

Implementation

```
class Queue
{
        private int maxSize;
        private long[] queArray;
        private int front;
        private int rear;
--------------------------------------------------------
public Queue(int s) // constructor
{
        maxSize = s+1; // array is 1 cell larger
        queArray = new long[maxSize]; // than requested
        front = 0;
        rear = -1;
}
--------------------------------------------------------
public void insert(long j) // put item at rear of queue
{
if(rear == maxSize-1)
rear = -1;
queArray[++rear] = j;
}
```

```
public long remove() // take item from front of queue
{
        long temp = queArray[front++];
        if(front == maxSize)
        front = 0;
        return temp;
}
-------------------------------------------------------
public long peek() // peek at front of queue
{
                return queArray[front];
}
-------------------------------------------------------
public boolean isEmpty() // true if queue is empty
{
                return(rear+1==front || (front+maxSize-1==rear) );
}
-------------------------------------------------------
public boolean isFull() // true if queue is full
{
                return (rear+2==front ||(front+maxSize-2==rear) );
}
} // end class Queue
```