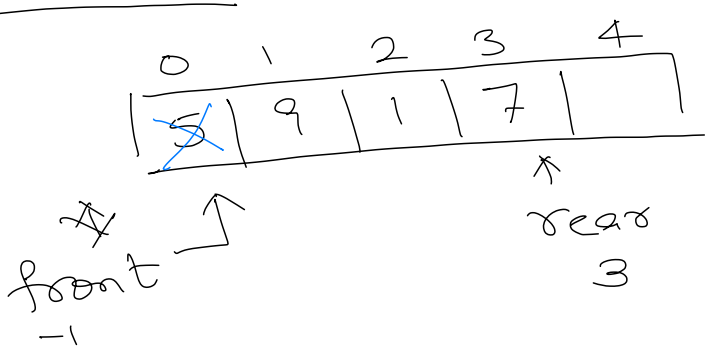
Linear Queue



Delete Q ( )

→ Move front to next element.

→ Remove element from front.

→ [Shift all elements of queue to left by 1 place.]

↓

This is done to avoid condition that queue is empty as well as full.

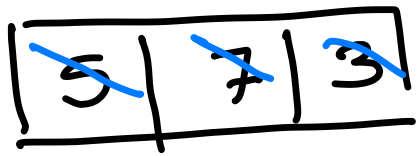
Inefficient way.

OR

→ if queue is empty AND queue is full then

→ Reset front and rear

front = -1, rear = -1.



Add Q(5)
Add Q(7)
Add Q(3)

front → -1 0 1 2
rear → -1 0 1 2

⇐ Queue is full.

Queue is empty.

Delete Q() ⇒ 5
Delete Q() ⇒ 7
Delete Q() ⇒ 3

---

class C1 {

data member
member functions F1();

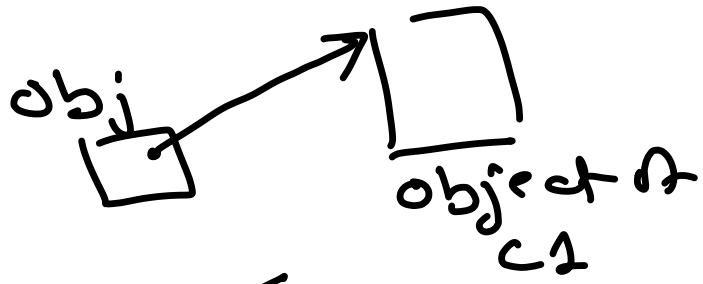
}

C1 obj;

↳ variable that stores reference to object of class.

```
obj.F1();
```
← Null Pointer Exception.

```
obj = new C1();
```




```
obj.F1();
```
✓

```
class Stack {
    public void Push(int)
    public int Pop() {...}
        :
}
```

Defining Stack ⇔ ADT

```
interface StackInt {
    public void Push(int);
    public int Pop();
        :
}
```

```
class StackUsingArray
      implements StackIntf
{
   :
}
```

```
Reverse(int [] elements,
        StackIntf stack)
   stack.Push(..);
     :
   stack.Pop();
}
```


```
Obj = new StackUsingArray();
Reverse(arr, Obj);
```

---

main()

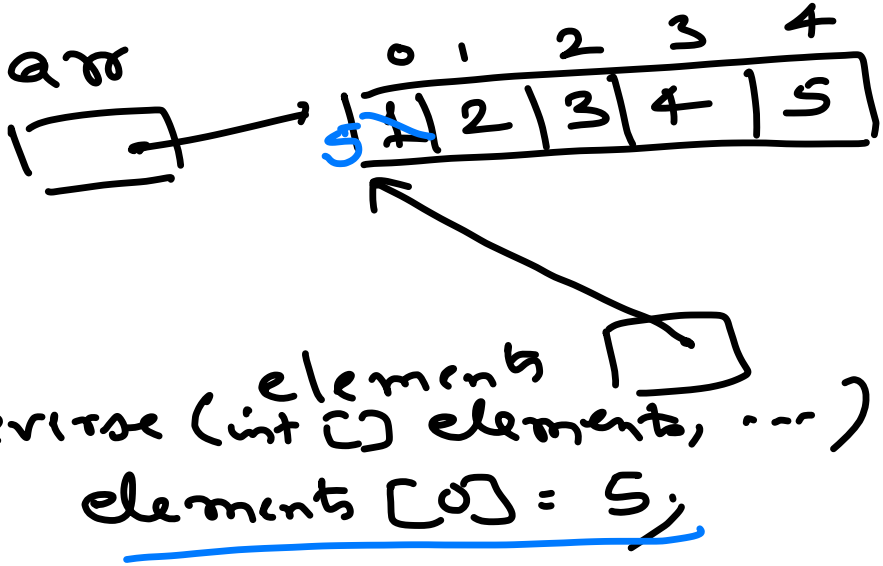
```
int [] arr = {1, 2, .. , 5};
```

Reverse (arr, ...);

⋮

}

OR

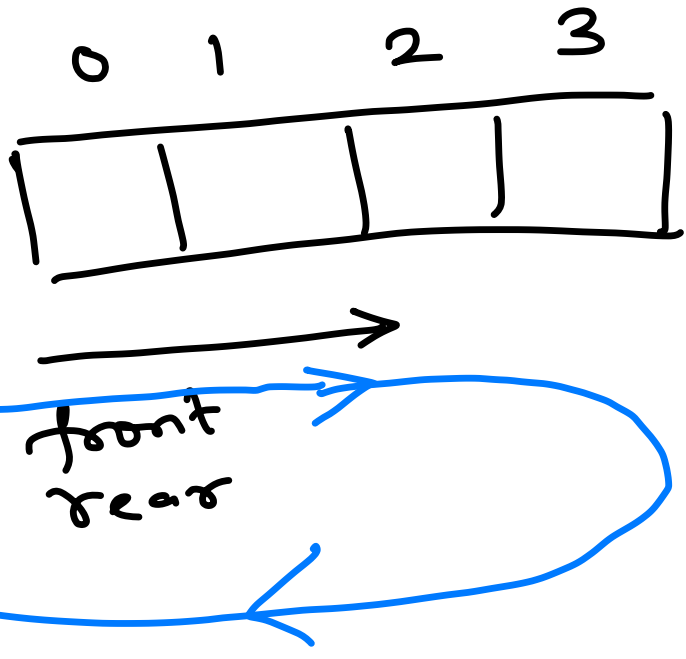


Reverse (int [] elements, ...) {

elements [0] = 5;

}

```
int [] arr;
arr = new int [5];
arr [0] = 1;   arr [2] = 3;
arr [1] = 2;   arr [3] = 4;
               arr [4] = 5;
```

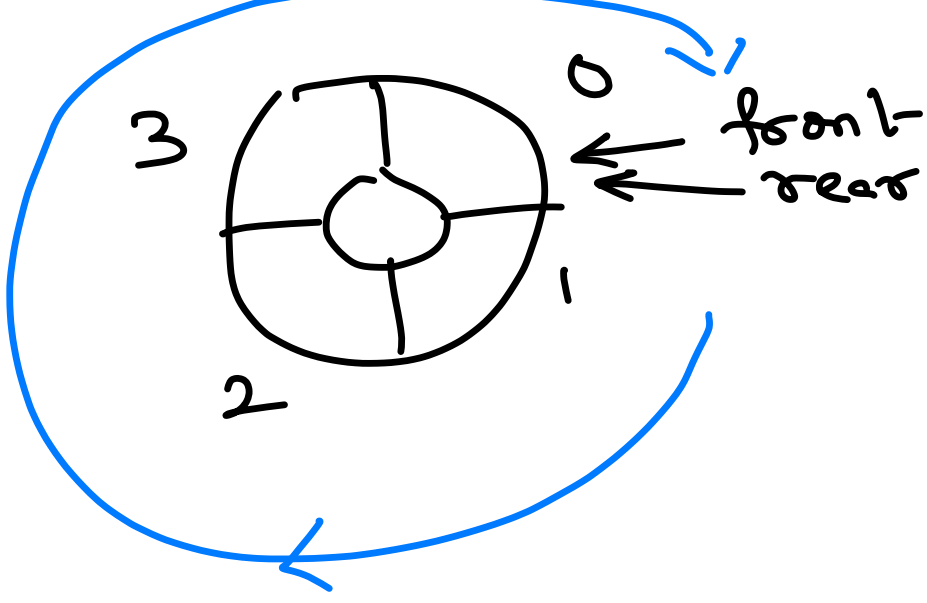



```
++rear;
if (rear == n)
    rear = 0;
```

OR

```
rear = (rear + 1) % n;
```

0..(n-1)



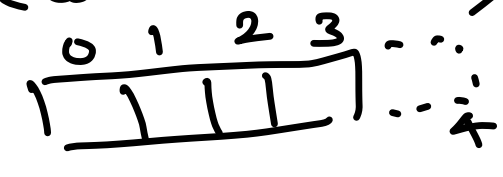
front equals rear ← Empty

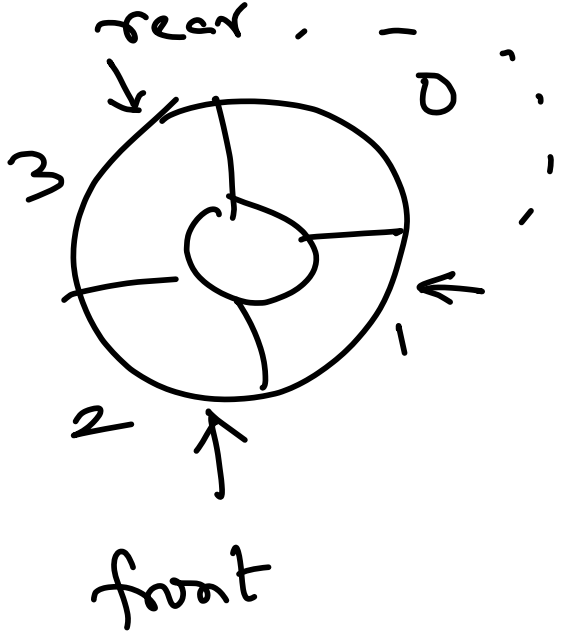
rear just before front ← Full

⇓

(rear+1) % n equals front

new int [4];





---

① Check for balanced parenthesis.

( ) ( ) ✓

([ )] ✗

) (✗

([ ]) ✓

Hint: Use stack.

boolean IsBalanced (String str)

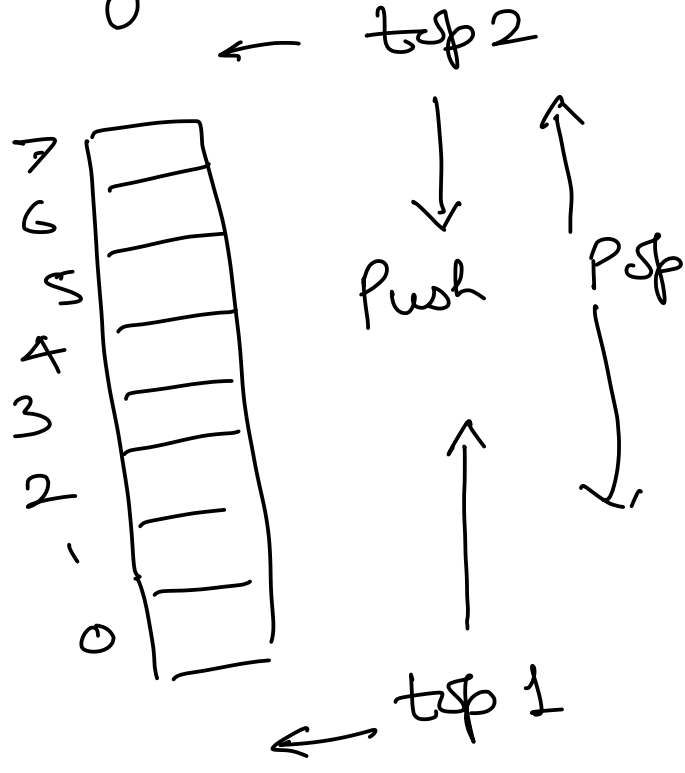
* ② Implement Stack using Queue.

* ③ Implement Queue using Stack.

Hint: ② will need two queues.

③ will need two stacks.

④ Implement 2 Stacks in a single array.



Linked List

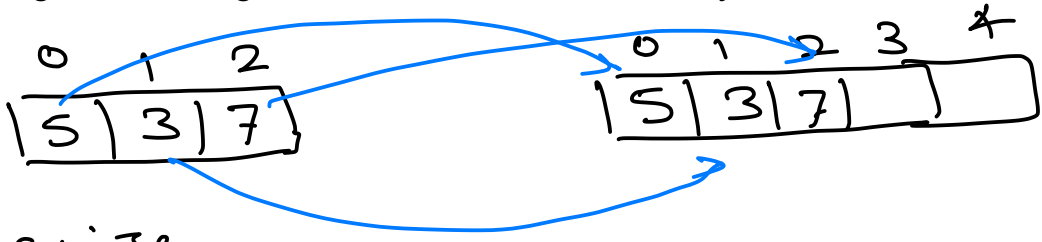
Array : Need?

When we need to store multiple elements.
And do same processing on those elements.

Properties of array:
- Data structure that stores multiple elements, all of the same type.
- All elements of an array are stored sequentially in memory, one after another.

Advantages of array:
- Efficient lookup OR Random access.
- Efficient in adding or removing elements at the end of array.

Disadvantages of array"
- Fixed size. Resizing of array is inefficient.
- Inserting and deleting of elements, in middle of array is inefficient.



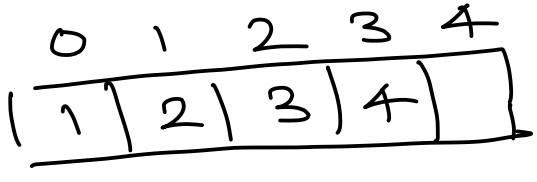
Resize
→ Create a larger array
→ Copy values from existing array to new one.
→ Release old memory.

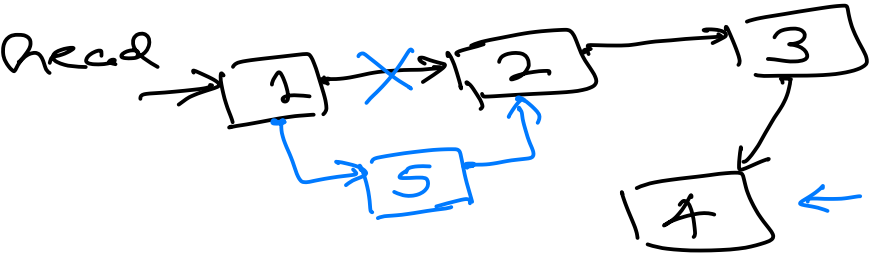
Initial array size = 1000 → Copy 1000 elem.

Resized array size = 2000 → Copy 2000 elements

Resized array size = 4000

Linked list → Do not store elements next to each other.





Properties of linked list
- Stores data as a chain of nodes. → where data is stored.
- Each node contains data and a pointer to next node in chain.
- We need to know where first node is of list - head.

Advantages of linked list
- Can easily grow / shrink in size.
- Efficient in insertion and deletion of elements.

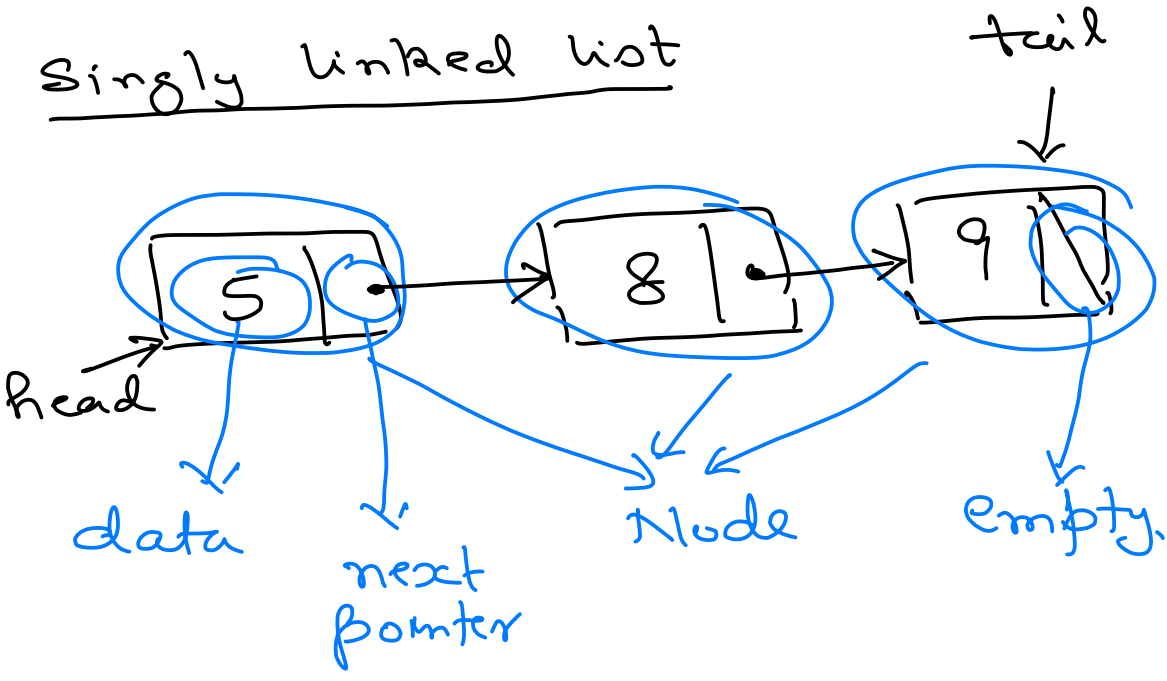
Disadvantages of linked list
- Random access is inefficient.

Types of linked list
- Singly linked list (Uni-directional)
- Doubly linked list (Bi-directional)
- Circular list.

One node keeps track of one neighbour only.

Each node keeps track of both of its neighbours

Singly linked list



Traversal

Starting from first element, access each element one at a time, till the last element.

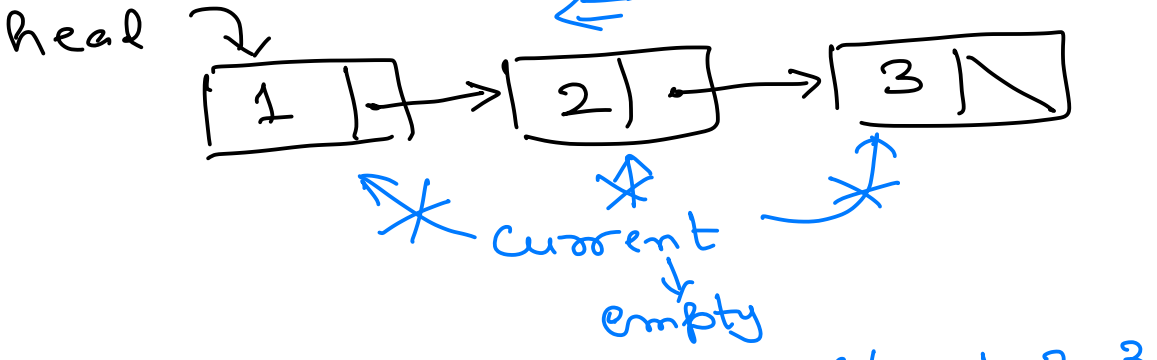
```
for ( i = 0; i < elements.length; ++i)
    stack.Push (elements[i]);
```

Traversal.

① What if list is empty?
② What if list is not empty?

head → empty ⇐ list is empty.

⇐ list is not empty.



o/p: 1 2 3

Traversal of singly list
- if list is empty them Stop.
- Set current to first node of list.
- while (current is not empty) do
  - Process current node.
  - Move current to current node's next.
- Stop.

Traversal of singly list (Optimised)
- Set current to first node of list.
- while (current is not empty) do
  - Process current node.
  - Move current to current node's next.
- Stop.