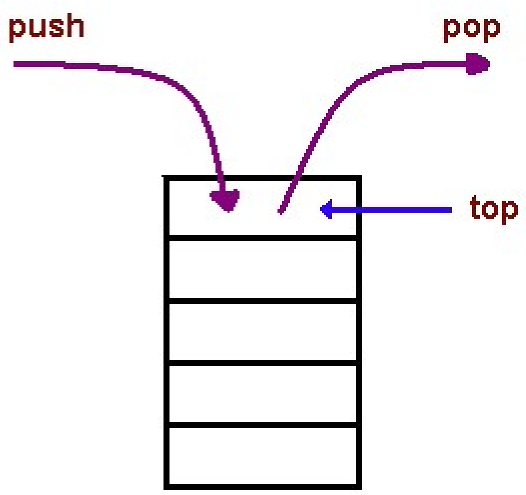
Chapter - 10 : Self Referential Structures and Linked Lists

Comparison of arrays and Linked List

	Arrays	**Linked List**
1. Data Structure	**Static** : Needs to define the size of arrays at compile time. It can not grow or shrink at runtime.	**Dynamic**- No need to define the size of Linked List. It can grow or shrink at runtime.
2. Insertion	**Difficult** : Need to shift the elements for insertion of new value.	**Easier**- No need to shift the elements for insertion of new value.
3. Deletion	**Difficult** : Need to shift the elements after deletion of value.	**Easier**- No need to shift the elements after deletion of value.
4. Memory Wastage	In arrays there is a lot of memory wastage, as if we have used only 10 spaces from the allocated 100 spaces.	Increase or decrease in size of the Linked List at runtime, helps to reduce the memory wastage.
5. Implementation of Stack and Queue	Difficult to implement the stack and queue data structures using arrays.	Easy to implement the stack and queue data structures using arrays.
6. Memory Usage	Memory required for storing the values only. No Extra overhead.	More memory required to store values and address of next node. More overhead to store values.
7. Traversal	Element access in arrays is easy. Any element can be accessed directly.	Element traversal is difficult. We can not randomly access any element.
8. Reverse Traversing	Reverse traversing is easy in singly linked list.	Reverse traversing is difficult in singly linked list. Address of only next node is stored, so reverse traversing is not possible.

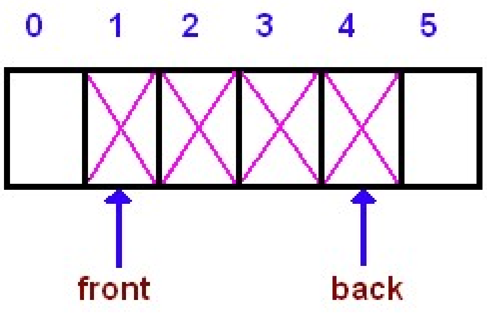
Stack

- A stack is a container of objects that are inserted and removed according to the last-in first-out (LIFO) principle.
- A stack is a limited access data structure - elements can be added and removed from the stack only at the top.
- A stack is a **recursive** data structure.
- Example - Stack of books; you can remove only the top book, also you can add a new book on the top.
- Two operations are allowed:
  1. **push** the item into the stack(to add the item), and
  2. **pop** the item out of the stack(to remove the item)



Queue

- A queue is a container of objects (a linear collection) that are inserted and removed according to the first-in first-out (FIFO) principle.
- An example of a queue is a line of students on the fees deposit counter.
- Two operations are allowed **enqueue** and **dequeue**.
  1. Enqueue means to insert an item into the back of the queue,
  2. Dequeue means removing the front item.



Comparison of Stack and Queue

STACKS	QUEUES
1. Stacks are based on the LIFO principle, i.e., the element inserted at the last, is the first element to come out of the list.	Queues are based on the FIFO principle, i.e., the element inserted at the first, is the first element to come out of the list.
2. Insertion and deletion in stacks takes place only from one end of the list called the top.	Insertion and deletion in queues takes place from the opposite ends of the list. The insertion takes place at the rear of the list and the deletion takes place from the front of the list.
3. Insert operation is called push operation.	Insert operation is called enqueue operation
4. Delete operation is called pop operation.	Delete operation is called dequeue operation.
5. In stacks we maintain only one pointer to access the list, called the top, which always points to the last element present in the list.	In queues we maintain two pointers to access the list. The front pointer always points to the first element inserted in the list and is still present, and the rear pointer always points to the last inserted element.

.