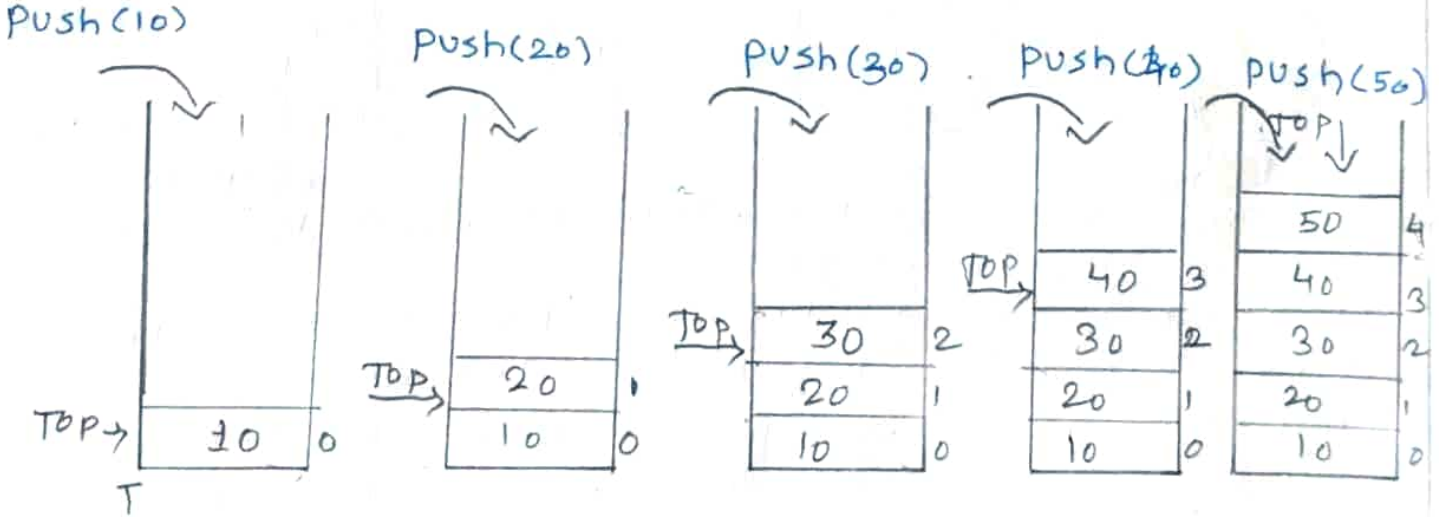
unit 2:
stack & queue

* stack:

- stack is a linear data structure that follow Last in first out principle (LIFO).
- stack is ordered list & collection of homogeneous elements
- stack can be defined as a container in which insertion and deletion can be done from the only one end known as the top of the stack.
- stack contain only one pointer i.e top pointer ~~Pointing~~ to pointing to topmost element of the stack
- whenever an element is added in the stack, it is added on the top of the stack and the element can be deleted only from the Top of the stack.
- It is called stack because it behave like a real world stack. eg stack of plates, Books.

* working with stack: * PUSH() operation.

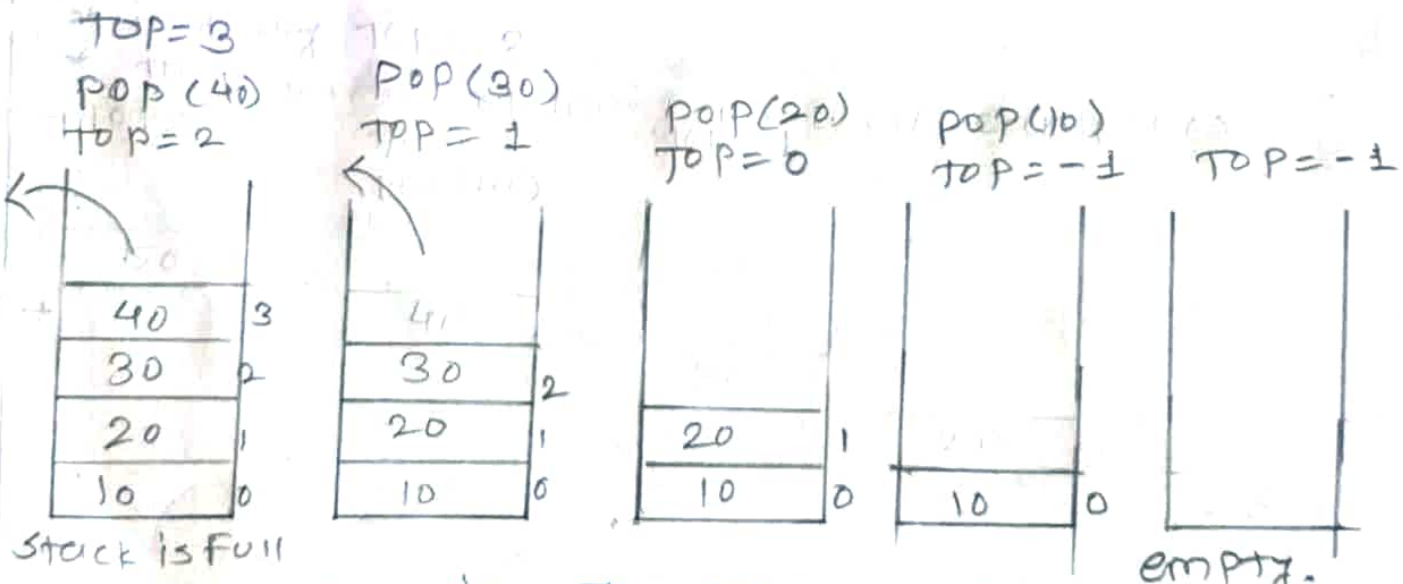
- stack works on LIFO principle
- lets consider the size of stack is 5.
- In stack we can Insert elements one by one untill stack become full.



In above figure we can observe that stack gets filled up from the bottom to top.

- suppose that, the stack is 'st' of size 'n' it means that st[n] is an Array, then the first element get stored at st[0] index, second element get stored at st[1] index and last element will stored at st[n-1] index.
- In above figure size of stack is 5. and we have inserted 5 elements in the stack. now there is no space available ~~is~~ in stack.
- If No space is available in stack and ~~The~~ we are trying to insert element in to the stack, then situation is known a stack overflow.
- If we trying to remove element from empty stack then this condition is known as underflow.

* pop() operation.



*- Before deleting the element from the stack, whether stack is empty.
- if stack is not empty, we first access the element which is pointed by the TOP
- once the pop operation is performed, the TOP is decremented by 1. that is top=top-1.

* Linked List implementation of stack.

- we can implement linked list by using array very easily. but array implementation of stack has limitation. That during execution of program, stack can not grow or shrink.
- This limitation is overcome by representing stack using linked list
- If we implement stack by using linked list system will allocate a memory to new node dynamically, so that stack can grow or shrink during execution of program, so that such type of stack never become Full.
- In this implementation, node of the linked list will represent element of stack.
- The insertion and deletion operation can be done by only one end called Top of the stack. so the link of bottom node is set to NULL.
- Figure below shows linked representation of stack.

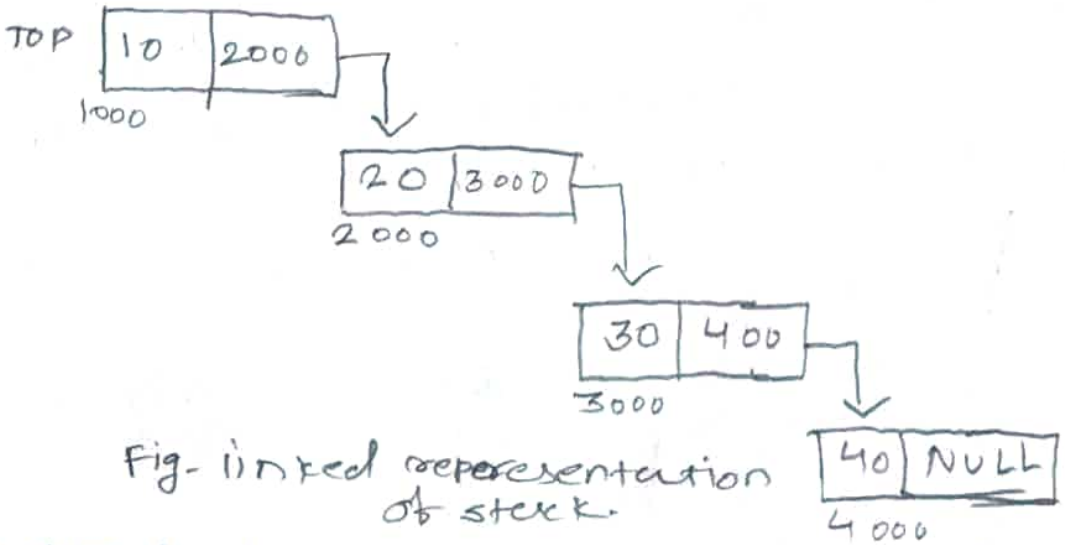


Fig- linked representation of stack.

- In this stack bottom element is 40, with next pointer is set to NULL
- Top is pointing top element That is 10.

- We can create structure to implement stack dynamically.

```
struct node
{
    int data;
    struct node *next;
}
```

* Algorithm to push() element into stack:-

- To insert element into stack we perform push() operation.
- It involved following steps.

step 1: start
step 2: if stack is full
        display message & exit
step 3: if stack is not full
        increment top by 1
        TOP ← TOP + 1
step 4: Add element to the stack, where TOP is pointing.
        stack[TOP] ← data
step 5: stop.

* Algorithm to pop() element from stack.

step 1: start
step 2: if stack is empty
        return NULL
step 3: if the stack is not empty, access the data element at which top is pointing
        data ← stack[top]
step 4: Decrease value of top by 1
        top ← top - 1
step 5: return Data [Display data]
step 6: stop.

*push() operation using linked list:-

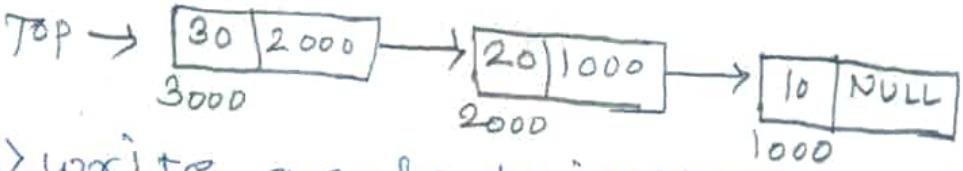
1. Initialization of stack.

```
struct node *s;
struct node *top = NULL;
```

2. push (s, 10)

3. Push (s, 20)

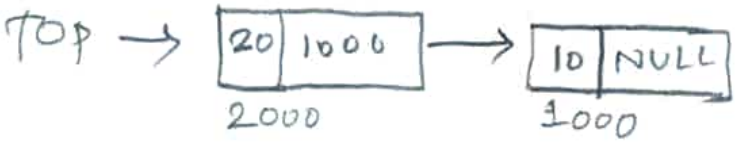
4) Push (s, 30)



Q) write a code to implement push() in c/c++/Java.

* POP() operation using linked list.

After calling POP():



After calling POP():

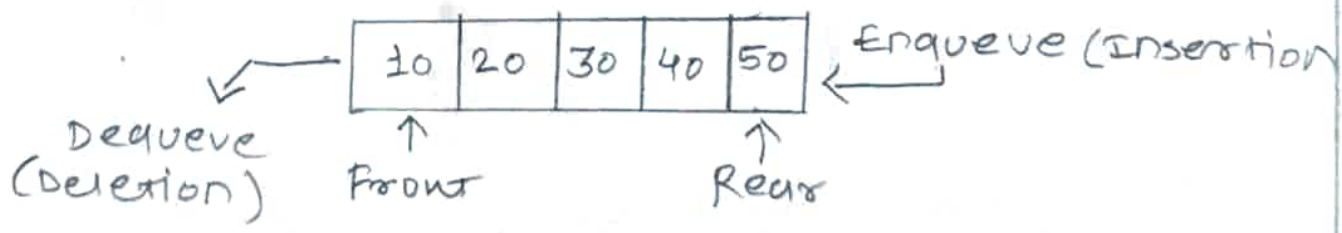
Here operations are same that we have Performed in linked list. Different is that we have to insert and delete element only from top of the stack.

Queue:

- Queue is a Linear Data structure
- Queue is a linear collection of Homogenous data elements.
- Queue can be define as ordered list, which enables insertion to be performed at one end called REAR and delete operation to be performed at another end called ~~REAR~~ FRONT.
- Queue works on First In First Out principle (FIFO)

eg. Students waiting in the line in college for Admission process forms a queue.

- In the Queue we can insert element only from back side (Rear).
- In the Queue we can delete element only from Front side (front).
- Fig. below shows Queue.



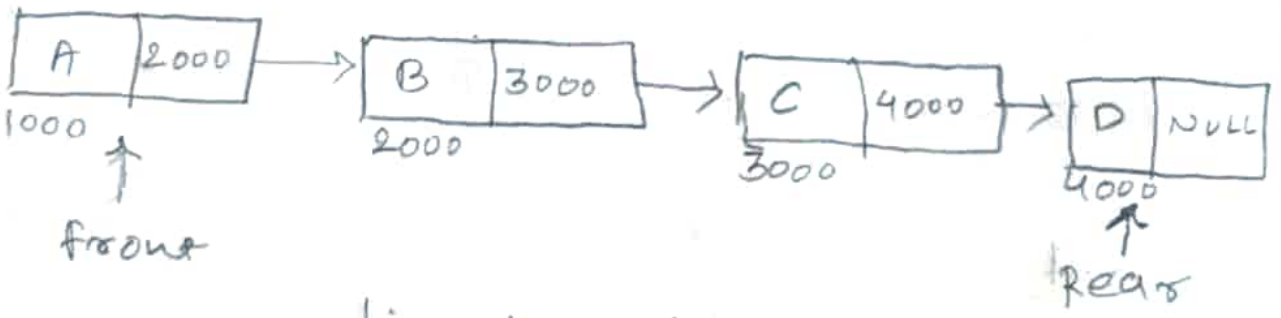
- In queue we perform Insertion operation is known as Enqueue operation.
- In queue we perform deletion operation is known as Dequeue operation.

*Linkedlist implementation of queue:

If we implement queue using Array, then we need to declare array in Advance due to this we cannot extend or shrink The size of array during program execution. so it may result in to wastage of memory or array size may ~~short~~ fall short.

So, to overcome this problem we can implement queue dynamically by using linked List.

- The elements in the linked list can allocated dynamically, hence it can grow or shrink runtime. and can grow as long as there is memory available.



big- linked representation of queue.

*structure to implement queue.

```
Struct queue
{
    int data;
    Struct queue *next;
int *front;
    int **rear;
};
```

* operation on queue:

There are two basic operation which can be implemented on linked queue.

1) Insertion of element.
2) Deletion of element.

1) Insert operation:

- In insertion operation we can insert element at the end of queue.

- To insert element in to queue, first allocate memory for new node.

- There can be two scenario, we insert element in to an empty queue. In this case the condition

front = NULL become true.

Now the new element will be added as the only element of the queue.

- second scenario, the queue contains more than one element, The condition front = NULL becomes false. In this case we need to update the end pointer rear so that the next pointer of rear will point to the new node ptr. Here we will make rear pointer point to the newly added node ptr. also make the next pointer of rear point to NULL.

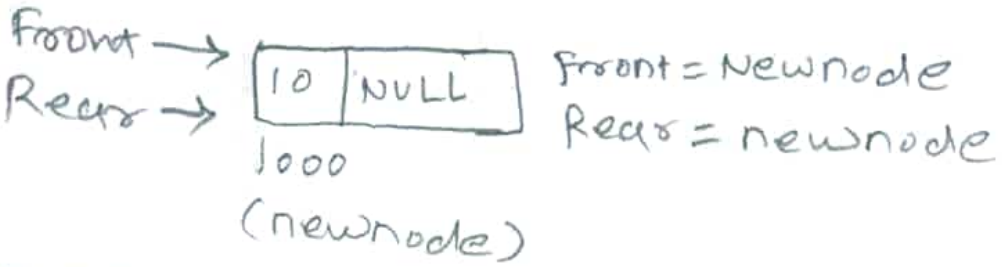
Steps to insert element:

1. Initialization of queue

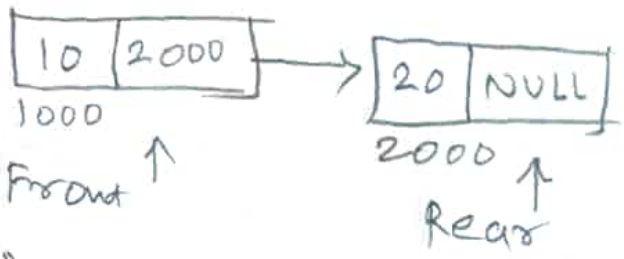
   front = rear = NULL
   (indicate queue is empty)

2. Insert elements

Insert 10:

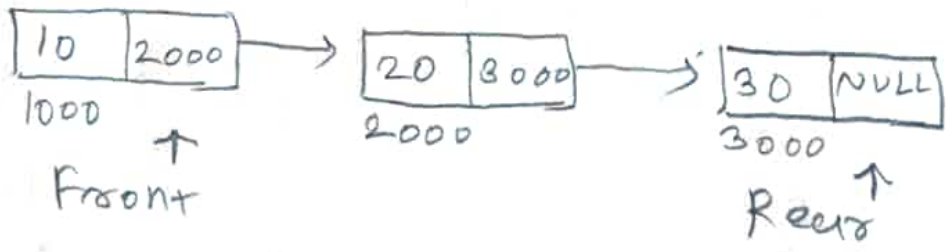


Insert (20):



Rear → Next = newnode

Insert (30):



Rear → Next = newnode.

Algorithm to insert element in queue

Step 1: start

Step 2: Allocate the space for new node ptr

Step 3: set
Ptr → data = val

Step 4: if front = NULL
set front = rear = NULL
set front → next = rear → next = NULL
else
set Rear → next = ptr
set Rear = ptr
set Rear → next = NULL
[end of IF] [end of IF]

Step 5: stop.

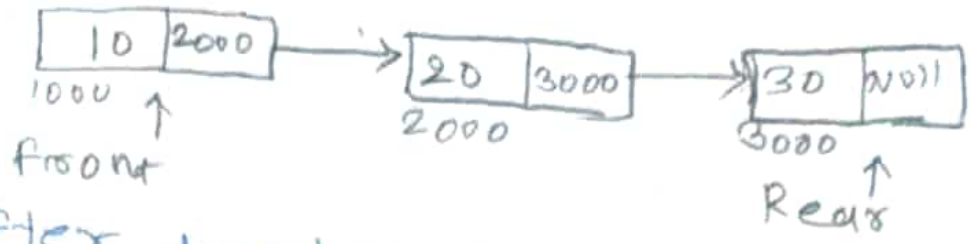
2) Deletion operation:

- Delete operation Remove element from queue which is first inserted among all the queue elements.

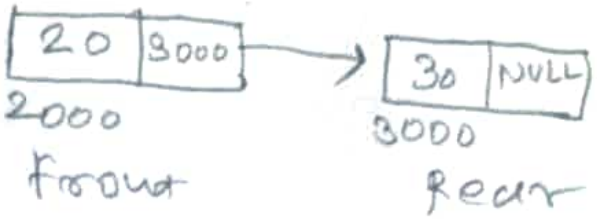
- Firsty we will check either the list is empty or not. The condition front = NULL become true if the queue is empty. In this case we simply write underflow on the consol and exit. Otherwise, we will delete the element which is pointed by front pointer. This is don by using following statement

ptr = front;
front = front → next;
free(ptr);

- consider following queue



After deletion of one element in queue will become like this.



Algorithm to delete element

Step 1: start

Step 2: if front = NULL
write "Underflow"
Go to step 5
end of if

Step 3: set ptr = front

Step 4: set front = front → NEXT

Step 5: free(ptr)

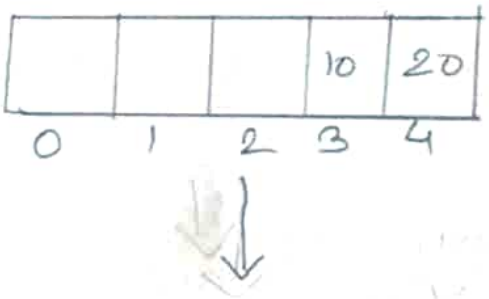
Step 6: stop.

*circular queue:

There is one limitation of Array implementation of a queue. If the rear reaches to the end position of the queue then there might be possibility that some vacant spaces are left in the beginning which cannot be utilized. So to overcome this problem circular queue was introduced.

- it means that, in circular queue insertion of new element is done at the very first location of the queue if the last location of the queue is full. but it is possible only when those location are empty.

- following fig. shows circular representation of queue.



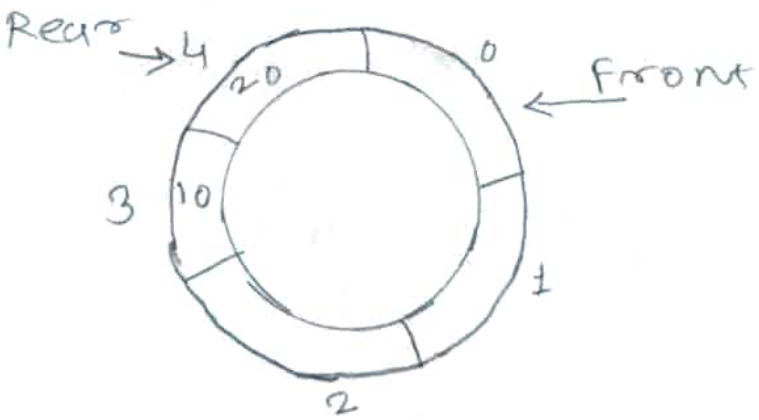


fig. circular queue Representation.

So, we can define circular queue as- circular queue is similar to linear queue as it also based on the FIFO principle except that the last position is connected to the first position in a circular queue which forms a circle. It is also known as Ring Buffer.

* operations on circular queue:

1. Enqueue():
   This function is used to insert the new value in the queue. The new element is always inserted from the rear.
2. dequeue():
   This function is used to delete the element from queue which is present at the front of queue.
3. front:
   It is used to get the front element from the queue.
4. Rear: it is used to get the rear element from queue.

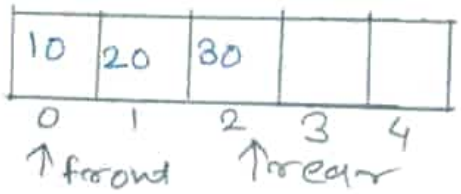
* Application of circular queue.

- memory management.
- CPU scheduling.
- Trafic system management.

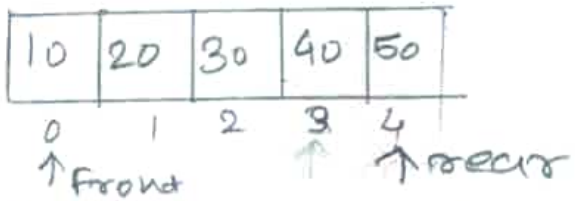
* Enqueue & Dequeue operation on circular queue.

- suppose a queue Q has element 10, 20, 30 and size of queue N=5

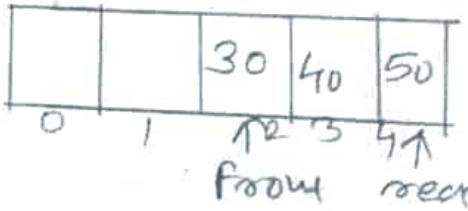
1.



2. Now insert 40, 50



3. After deletion of 10 & 20 queue looks like



4. Insert 60.

Here front is indicating to last location, so in the circular queue rear will indicate to 0th location because space is available there.