* Disadvantages of circular linked list.

i) Circular lists are complex as compared to singly linked lists.
ii) Reverse of circular list is a complex as compared to singly or doubly lists.
iii) If not handled carefully, then the code may go in an infinite loop.
iv) Harder to find the end of the list and loop control.
v) Inserting at start, we have to traverse the complete list to find last node.

* Stack

i) It is a collection / list of logically related similar type elements into which data elements can be added as well as deleted from only one end referred top end.

ii) In this collection / list, element which was inserted last only can be deleted first, so this list works in last in first out / first in last out manner, and hence it is also called as LIFO list / FILO List.

iii) Stack has one end whereas the Queue has two ends (front and rear).

iv) 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 stack.

v) Stack can be defined as a container in which insertion and deletion can be done from the one end known as the top of the stack.

* Some key points related to stack.

i) This is called as stack because it behaves like a real-word stack, piles of book etc

ii) A stack is an abstract data type with a pre-defined capacity, which means that it can store the elements of limited size.

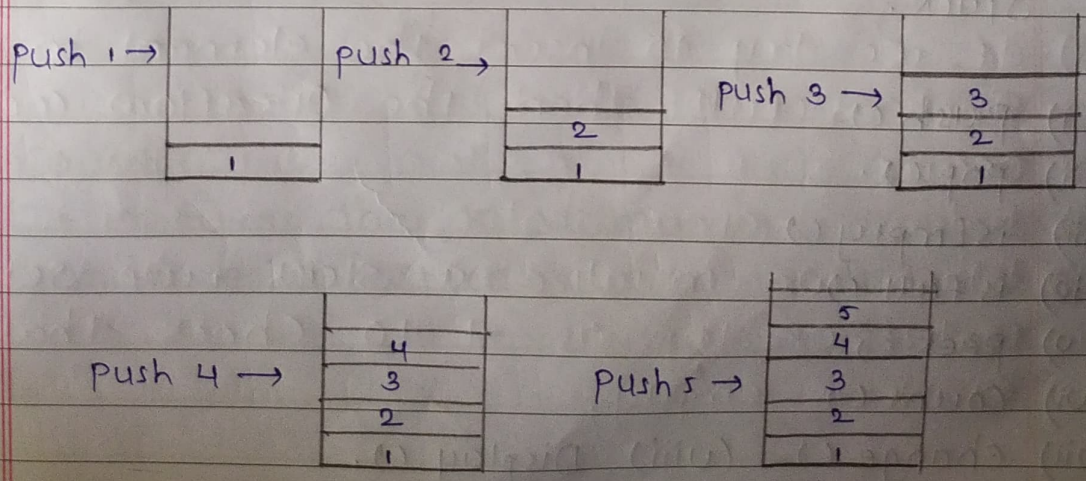
iii) It is a data element that follows some order to insert and delete the elements, and that order can be LIFO or FILO.

* Working of Stack.

i) Stack works on the LIFO pattern As we observe in the below figure are five memory block in the stack therefore the size of the stack is 5.

ii) Suppose we want to store the element in a stack and let's assume that stack is empty.

iii) We have taken the stack of size 5 as shown below in which we are pushing the element one by one until the stack becomes full.



iv) Since our stack is full as the size of the stack is 5. In the above cases, we can observe that it goes from top to bottom when we were entering the new element in the stack. The stack gets filled up from the bottom to the top.

v) When we perform the delete operation on the stack, there is only one way for entry and exit as the other end is closed.

vi) It follows the LIFO pattern, which means that the value entered first will be removed last.

vii) In example value 1 is entered first, so it will be removed only after the deletion of all other element.

* Some standard stack operation.

Following are some basic operation of stack.

i) push()
ii) pop()
iii) isEmpty()
iv) is full()
v) peek()
vi) Count()
vii) change() (viii) Display().

i) push ()
When we insert an element in a stack then the operation is known as a push. If a stack is full then the overflow condition occurs.

ii) pop ()
When we delete an element from the stack the operation is known as a pop. If the stack is empty means that no element exist in the stack this state is known as a underflow condition.

iii) isEmpty ()
It determines whether the stack is full or not.

iv) isFull ()
It determines whether the stack is full or not.

v) peek ()
It returns the element at the given position.

vi) count ()
It returns the total number of element avaliable in stack.

vii) change ()
It changes the element at the given positio

viii) display ()
It prints all the element available in the stack.

1] PUSH() Operation.

Algorithm.

Step 1: check stack is not full (if stack is not full then only we can push element into it).
Step 2: Increment the value of top by 1.
Step 3: Insert/push an element onto the stack at top end.

OR

i) Before inserting an element in a stack, we check whether the stack is full.

ii) If we try to insert the element in a stack is full, then the Overflow condition occurs.

iii) When we initialize a stack, we set the value of top as -1 to check that the stack is empty.

iv) When the new element is pushed in a stack, first the value of the top gets incremented i.e top = top + 1 and, the element will be placed at the new position of the top.

v) The element will be inserted until we reach the max size of the stack.

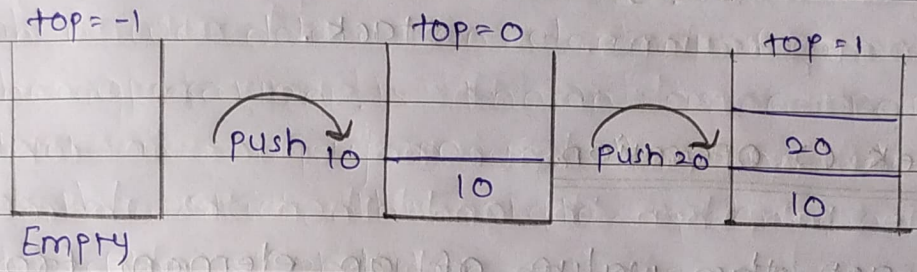



Fig: PUSH element in Stack.

2] pop() operation.

i) Before delering the element from the stack, we check whether stack is empty.

ii) If we try to delete the element from the empty stack, then underflow condition occurs.

iii) If the stack is not empty, we first access the element which is pointed by the top.

iv) Once pop operation is performed the top is decremented by 1. i.e top = top - 1

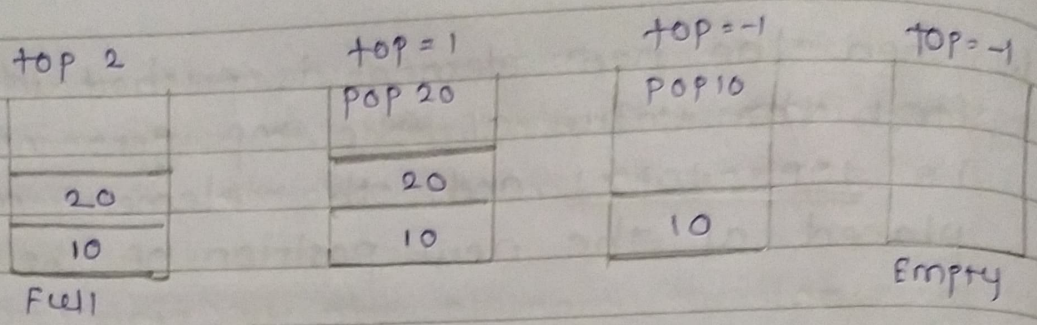



Fig: pop element in stack.

3) Peek () operation.

To get the value of an element from the stack which is at top end.

Step 1: check stack is not empty (if stack is not empty then only we can peek element from it).

Step 2: get / return the value of an element from the stack which is at top end.
(without push / pop an element from stack)

* Application of Stack.

i) Balancing of symbol.
ii) String Reversal.
iii) UNDO / REDO
iv) Recursion

v) DFS (Depth first search)
vi) Backtracking.
vii) Expression conversion.
viii) memory managment.

## i) Balancing of Symbol.

i) Stack is used for balancing a symbol. As we know each program has an opening and closing braces:

ii) When the opening braces come, we push the the braces in a stack, and when the closing braces appear we pop the opening braces from the stack.

iii) Therefore, the net value comes out to be zero. if any symbol is left in the stack, it means that some syntax occurs in a program.

## ii) String Reversal.

i) Stack is also used for reversing a string. for example, we want to reverse a "Bhakti" String. so we can achieve this with the help of stack.

ii) First, we push all the characters of the string in a stack until we reach the null character.

iii) After pushing all the characters, we start taking out of the character one by one until we reach bottom of stack.

iii) UNDO / REDO

i) It can also be used for performming UNDO/ REDO operations.

ii) For example, we have an editor in which we write 'a', then 'b' and then 'c' therefore the text written in an editor is abc.

iii) So, there are three states a, ab and abc which are stored in stack. There would be two stacks in which one stack shows UNDO state, and the other shows REDO state.

iv) If we want to perform UNDO operation and want to achieve 'ab' state, then we implement pop operation.

iv) Recursion

The recursion means that the function is calling itself again. To maintain the previous states, the compiler creates a system stack in which all the previous records of the function are maintained.

v) DFS (Depth First Search)

This search is implemented on a graph and graph uses the stack data structure.

vi) Backtracking.

Suppose, we have to create a path to solve a maze problem if we are moving in a particular path and we realize that we come on the wrong way. In order to come at the begi-nning of the path to create a new path we have to use stack data structure.

vii) Expression Conversion.

Stack can also be used for expression conve-rsion. This is one of the most important application of stack.

Following is the list.

i) Infix to prefix
ii) Infix to postfix
iii) prefix to infix
iv) prefix to postfix
v) postfix to infix

viii) memory managment.

i) The stack manages memory. The memory is assigned in the contiguous memory blocks.

ii) The memory is known as stack memory as all the variable are assigned in a function call stack memory.

iii) The memory size assigned to the program is known to compiler.

iv) When function is created all its variable are assigned in the stack memory when the function completed its execution, all the variables assigned in the stack are released.

Q. What is Expression.

An expression is a combination of an operands and operator.

There are 3 types of expression.

1) Infix expression :- a + b
(operator in two operands)

2) Prefix expression :- +ab
(operator before two operands)

3) Postfix expression :- ab+
(operator after two operands)

1) Infix to prefix.

i) First, reverse the infix expression given in the problem.

ii) Scan the expression from left to right.

iii) Whenever the operands arrive, print them.

iv) If the operands arrive and the stack is found to be empty, then simply push the operator into the stack.

v) If the incoming operator has higher procedu then the Top of the stack, push the incoming operator into the stack.

vi) If the incoming operator has the same precedence with a Top of the stack, push the incoming operator into the stack.

vii) If the incoming operator has lower precenden ce than the Top of the stack, pop and print the top of the stack. Test the incoming operator against the top of the stack again and pop the operator from the stack till it finds the operator of a lower precedence or same precedence.

viii) If the incoming operator has the same precedence with the top of the stack and

the incoming operator is ^, then pop the top of the stack till the condition is true. if the condition is not true push the ^ operator.

ix) When we reach the end of the expression pop and print all the operators from the top of the stack.

x) If the operator is ')', then push it into the stack.

xi) If the operator is '(', then pop all the operators from the stack till it finds, opening bracket in the stack.

xii) If the top of the stack is ')', push the operator on the stack.

xiii) At the end, reverse the output.

2] Infix to postfix.

i) scan the infix expression from left to right

ii) If the scanned character is an operand output it,

iii) Else,
1] If the precedance of the scanned operator is greater than the precedance

of the operator in the stack (or the stack is empty on the stack contains push it.

2] Else, pop all the operarors from the stack which are greater than or equal to in precedence than that of the scanned operator.

iv) If the scanned character is an '(' push it to the stack.

v) If the scanned character is an ')' pop the stack and output it until a '(' is encountered, and discard both the parenthesis.

vi) Repeat step 2-6 until infix expression is scanned.

vii) print the output.

viii) pop and output from the stack until it is not empty.

3] Prefix to infix

i) Read the prefix expression in reverse order (from right to left).

ii) If the symbol is an operand, then push it onto the stack.

iii) If the symbol is an operator, then pop two operators from the stack create a string by concatenating the two operands and the operator between them
string = (operand1 + operator + operand 2)
and push the resultant string back to stack.

iv) Repeat the above steps until end to prefix expression.

iv) Prefix to postfix.

i) Read the prefix expression in reverse order (from right to left).

ii) If the symbol is an operand, then push it onto the stack.

iii) If the symbol is an operator, then pop two operands from stack create a string by concatenating the two operands and the operator after them.

string = operand 1 + operand 2 + operator.
and push the resultant string back to stack.

iv) Repeat the above steps until end of prefix expression.

## * Queue

i) Queue can be defined as an ordered list which enables insert operations to be performed at one end called Rear and delete operations to be performed at another end called Front.

ii) Queue is referred to be as first in first out list.

iii) Queue is a basic / linear similar data structure, which is a collection / list of logically related similar type of data element in which elements can be inserted from one end refferred as rear end.

iv) In data structure queue, elements which was inserted first can be deleted first, so this list works in first in first out / last in last out manner., and hence queue is also called as FIFO / LILO List

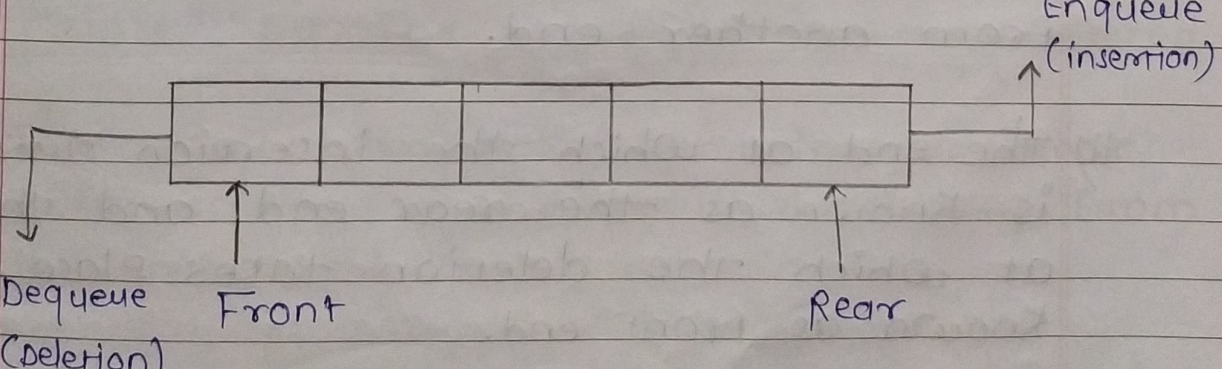



Fig: Queue.

We can perform basic 2 operation on queue in O(1) time.

1) enqueue.
2) dequeue.

1) enqueue
To insert an element into the queue from rear end.

2) dequeue.
To delete / remove an element from the queue which is at from end.

There are four types of queue.

i) Linear queue
ii) circular queue.
iii) priority queue.
iv) double ended queue (deque).

1] Linear Queue.

i) In linear queue an insertion takes place from one end while the deletion occurs from another end.

ii) The end at which the insertion take place is known as the rear end, and the end at which the deletion takes place is known as front end.

iii) It strictly follows the FIFO rule. The linear Queue can be represented as,

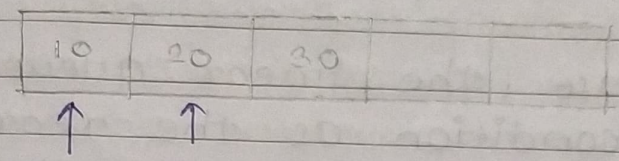



Fig: Linear Queue

iv) The above figure shows that the elements are inserted from the rear end and if we insert more elements in a Queue, then the rear value gets incremented on every insertion. If we want to show the deletion then it can be represented as;

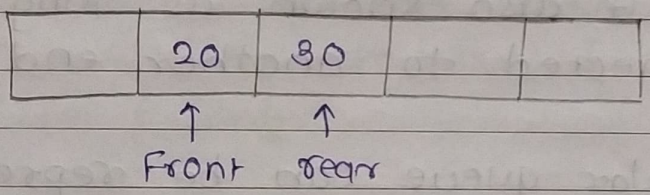



Fig: Deletion in Linear Queue.

v) In the above figure, we can observe that the front pointer points to the next element and the element which was previously pointed by the front pointer was delered.

vi) The major drawback of using a linear Queue is that insertion is done only from the rear end.

vii) If the first three elements are delered from the Queue, we cannot insert more

elements even though the space is available in a linear Queue.

viii) In this case, the linear queue shows the overflow condition as the rear is pointing to the last element of the Queue.

## 2] Circular Queue.

i) In circular Queue, all the nodes are represented as circular. It is similar to linear Queue except that the last element of the queue is connected to the first element. It also known as Ring Buffer as all connected to another end.

ii) The circular queue can be represented as;

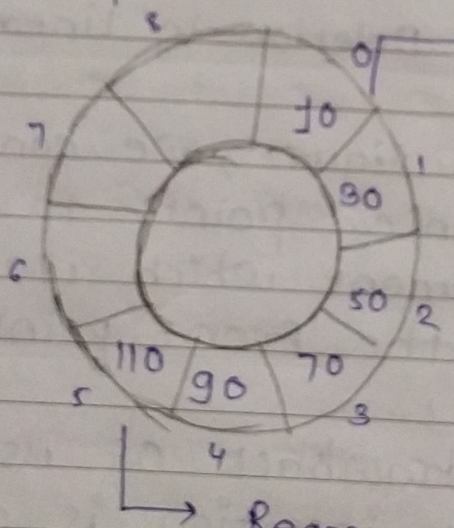



Fig: circular queue.

iii) The drawback that occurs in a linear queue is overcome by using the circular queue

iv) If the empty space is avaliable in a circular queue, the new element can be added in an empty space by simply incrementing the value of rear.

## 3] Priority Queue.

i) priority queue is another special type of queue data structure in which each element has some priority associated with it.

ii) Based on priority of the element the elements are arranged in a priority queue.

iii) If the elements occurs with the same priority then they are served according to the FIFO principal.

iv) In priority queue the insertion takes place based on the arrival while the deletion occurs based on the priority.

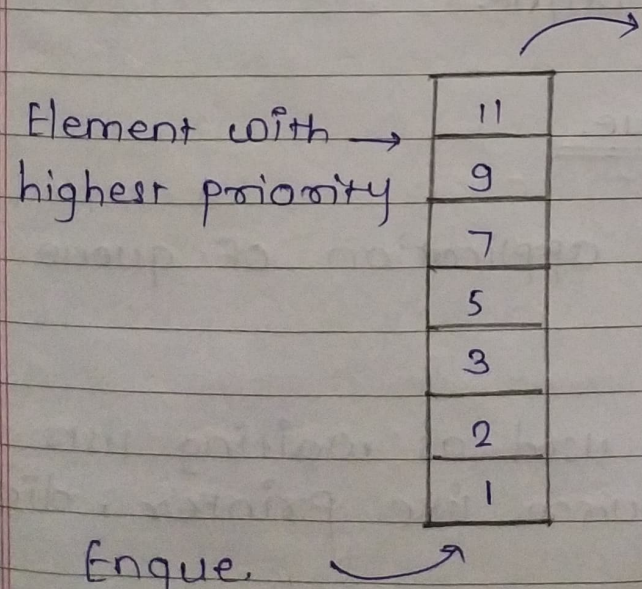



Fig: priority Queue.

iv) Double ended Queue (Deque).

i) both the linear queue and deque are different as the linear queue follows the FIFO principal whereas deque does not follow the FIFO principal.

ii) The deque stands for double ended queue In the queue insertion take place from one end while the deletion takes place from another end.

iii) The end at which insertion occurs is known as the rear end whereas the end at which deletion occurs is known as front end.

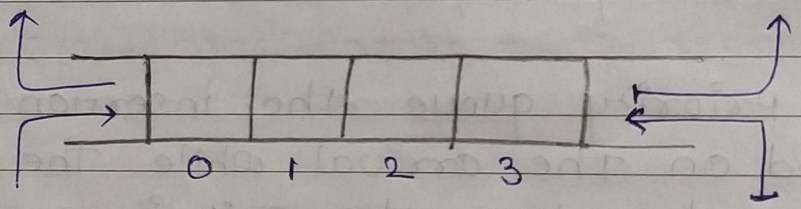



Fig: Deque.

* Application of Queue.

There are various application of queue as below.

1) Queues are widely used as waiting lists for a single shared resource like printer, disk, cpu.

ii) Queues are used in asynchronomus transfer of data (where data is not being transferred at the same rate between two processes) for eg. pipes, files IO sockets.

iii) Queues are used as buffers in most of the application like MP3 media player, CD player etc.

iv) Queus are used to maintain the play list in media players in order to add and remove the songs from list.

v) Queus are used in operating systems for handling interrupts.

* Complexity.

| Data Struct | Time Complexity | | | | | | | | Space complexity |
|---|---|---|---|---|---|---|---|---|---|
| | Average | | | | Worst | | | | Worst. |
| | access | search | insertion | deletion | Access | search | Insertion | Deletion | |
| Queue | $\theta(n)$ | $\theta(n)$ | $\theta(1)$ | $\theta(1)$ | $O(n)$ | $O(n)$ | $O(1)$ | $O(1)$ | $O(n)$ |

## * Tree data Structure

i) We read the linear data structures like an array, linked list, stack and queue in which all the elements are arranged in sequential manner.

ii) A tree is also one of the data structure that represent hierarchical data suppose we want to show the employees and their position in the hierarchical form then it can be represented as shown below:

## * Some key points about tree.

i) A tree data structure is defined as a collection of objects or enities known as nodes that are linked together to represent or simulate hieachy.

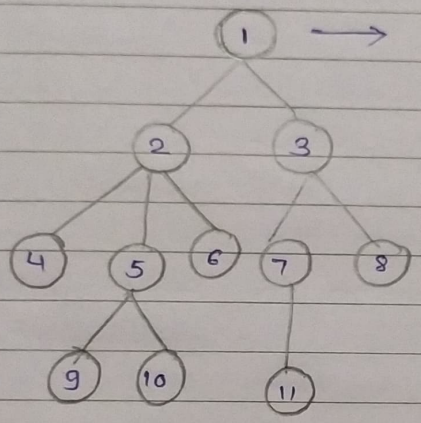
ii) A tree data structure is a non-linear data structure because it does not store in a sequential manner.

iii) It is hierarchical structure as element in a tree are arranged in multiple level.

iv) In tree data structure, the topmost node is known as a root node. Each node contains some data, and data can be of any type.

v) Each node contains some data structure and the link or refrence of other nodes that can be called children.

* Some basic terms used in Tree.




In above data structur, each node labelled with some number. Each arrow show in the above figure is known as a link between two nodes.

i) Root
The root node is the topmost node in the tree hi