Placement Assignment-1

**Ans1**- As we today data is most precious thing that all have data structures are used to store data as per user requirement they use different data structures, like if they want fast insertion and deletion of data they can use linkdelist etc.

**Ans2 -** Applications of data structure are :

a) Stack : Undo opereations , Browsing history

b) Queue : Task Scheduling , Data Packets in communication are arranged in queue

c) Graph : Finding shortest path , Facebbok Graphs's Api

d) Linkedlist : Web pages link, Music player

e) Tree : Database uses tree datastructure for indexing, DNS[Domain Name Server]

f) Hashtable : Data stored in database is key-value pair , Searching on google browser

**Ans3** : Insertion and Deletion is faster in linkedlist as compare to array.

**Ans4** : // Linkedlist structure in C

**Ans5** : In Doubly linkedList we can move back much faster than singly linkedlist as it contains address of previous and next node.

**Ans6** : **Stack** => is used to fetch and store data on the top it basically follows LIFO order and if we want any value we have to iterate over complete structure .

- Stack can be used to store different type of data.

**Array** => 1) If we know index value we can fetch data in O(1) time in this searching is index - -- based. -

2) Array stores homogenious data.

**Ans7** : 2 Queues

**Ans8 :** DFS[Depth First Search] - inorder , preorder , postorder

BFS [ Breadth First Search] - Level order

**Ans9** : Searching in BST is fast because in bst we arrange data in sorted manner left is always smaller than root and right is larger.

**Ans10** : Graph is used for finding shortest path , Connecting with different people Like Facebook used this for building connections etc.

**Ans11** : It is possible if we know the count of nodes but it is not efficient to find middle element every time in LinkedList because it is always search linearly.

**Ans12** : Memory leak in computers is basically a resource leakage[Inefficient usage of memory] that may occur when we have created any object which is no longer used in program and is not closed .

**Ans13** : If left of root is smaller and right of root is larger than root data.

**Ans14** : Stack due to its LIFO Order beacause recursion remember its caller so knows whom to return when the function has to return result.

**Ans15** : Undo Operations , Browsing History , Infix to postfix etc.

// Reverese a queue

**import** java.util.Deque;

**import** java.util.Iterator;

**import** java.util.LinkedList;

**import** java.util.Scanner;

**public** **class** Top {

**public** **static** **void** main(String[] args) {

Deque<Integer> d = **new** LinkedList<>();

Scanner in = **new** Scanner(System.***in***);

System.***out***.println("Enter Size : ");

**int** N = in.nextInt();

**while** (N-- > 0) {

d.add(in.nextInt());

}

Iterator<Integer> it = d.descendingIterator();

**while** (it.hasNext()) {

System.***out***.print(it.next() + " ");

}

}

}

// Rotate First K elements of Queue

import java.util.LinkedList;

import java.util.Queue;

import java.util.Scanner;

import java.util.Stack;

public class Top {

public static void main(String[] args) {

Queue<Integer> q = new LinkedList<>();

Scanner in = new Scanner(System.in);

// Assuming that queue has k elements

System.out.println("Enter Size :");

int N = in.nextInt();

System.out.println("Enter value of k");

int k = in.nextInt();

int count = 0;

Stack<Integer> stack = new Stack<>();

while (N-- > 0) {

if (count >= k) {

q.add(in.nextInt());

System.out.print(stack.peek() + " ");

stack.pop();

} else {

stack.push(in.nextInt());

}

count++;

}

while (!stack.isEmpty()) {

System.out.print(stack.peek() + " ");

stack.pop();

}

while (!q.isEmpty()) {

System.out.print(q.poll()+" ");

}

}

}

// Program to return the nth node from the end of the linkedlist

**import** java.util.Iterator;

**import** java.util.LinkedList;

**import** java.util.List;

**import** java.util.Scanner;

**public** **class** Top {

**public** **static** **void** main(String[] args) {

Scanner in = **new** Scanner(System.***in***);

System.***out***.println("Size of Array : ");

**int** n = in.nextInt();

System.***out***.println("Enter the value u want from last");

**int** last = in.nextInt();

List<Integer> list = **new** LinkedList<>();

**while** (n-- > 0) {

list.add(in.nextInt());

}

Iterator<Integer> it = ((LinkedList<Integer>) list).descendingIterator();

**while** (it.hasNext()) {

it.next();

**if** (last > 1)

it.remove();

**else**

**break**;

last--;

}

System.***out***.println(list.get(list.size() - 1));

/\*\*

\* We can do this by singly list also but for that we have to traverse the list

\*/

}

}

// Reverse a linkedlist

// I can reverse a list by the same way I have used for reversing a queue . [Doubly Linkedlist or List collection framework]

public static Node<Integer> reverse(Node<Integer> head) {

Node<Integer> prev = null;

Node<Integer> fast = head.next;

While(fast.next != null) {

prev = head;

head = fast

fast = fast.next;

head.next = prev;

}

return head;

}

// Replace the elements in array with their rank in array

**import** java.util.Arrays;

**import** java.util.HashMap;

**import** java.util.Map;

**import** java.util.Scanner;

**public** **class** Top {

**public** **static** **void** main(String[] args) {

Scanner in = **new** Scanner(System.***in***);

System.***out***.println("Size of Array : ");

**int** n = in.nextInt();

**int**[] array = **new** **int**[n];

**int**[] arrayClone = **new** **int**[n];

**for** (**int** i = 0; i < n; i++) {

**int** item = in.nextInt();

array[i] = item;

arrayClone[i] = item;

}

/\*\*

\* Made separate array with same elements because if we do int[] array2 = array;

\* this will give refrence to o array and changes we do in clone array are

\* reflected in both

\*/

Arrays.*sort*(arrayClone);

Map<Integer, Integer> hash = **new** HashMap<>();

**for** (**int** i = 0; i < n; i++) {

**if** (!hash.containsKey(arrayClone[i])) {

hash.put(arrayClone[i], i + 1);

}

}

System.***out***.println(hash);

**for** (**int** item : array) {

item = hash.get(item);

System.***out***.print(item + " ");

}

}

}