DATA STRUCTURES AND ALGORITHM

Stacks and Queues

1. PROGRAM TO SORT A STACK USING RECURSION

- **Sort**- ordering

Original Stack: [14, 9, 67, 91, 101, 25]
Sorted Stack: [9, 14, 25, 67, 91, 101]

Original Stack: [4, 9, 6, 8, 10, 5]
Sorted Stack is:[10, 9, 8, 6, 5, 4]

- **Stack-** LIFO,Top,Push,Pop
- **Stack and Array**

Why Sorting?

- memory management,
- maintaining the context of a process in the event of an interrupt,
- high-priority tasks
- Organised data is always preferred over random, as it can be more easily analysed and searched for keys more efficiently. It's always interesting to find out various other sorting techniques and which are the fastest and most preferable ones.

Sorting a Stack:

- Method 1:In this approach, a stack is sorted using another temporary stack.(iteration)
- Method 2: In this approach, a stack is sorted using recursion.

For Understanding more about Method 1 and 2:

https://www.enjoyalgorithms.com/blog/sort-stack-using-temporary-stack

Method 2:Does not use while loop as well as no another data structure is used.

The call stack is what a program uses to keep track of method calls. The call stack is made up of stack frames—one for each method call. For instance, say we called a method that rolled two dice and printed the sum.

Stack Frames:

A stack frame is a memory management technique used in some programming languages for generating and eliminating temporary variables. In other words, it can be considered the collection

of all information on the stack pertaining to a subprogram call. Stack frames are only existent during the runtime process.

Recursion:

- programming technique using function or algorithm that calls itself one or more times.

Difference Between Recursion and Iteration:

- In recursion, function **call itself** until the base or terminating condition is not true. On other hand, In Iteration **set of instructions repeatedly executes** until the condition fails. For example – when you use loop (for, while etc.) in your code.

- Iterative approach involves four steps, Initialization , condition, execution and updation.
- In recursive function, only base condition (terminate condition) is specified.

- Recursion takes more memory than iteration due to overhead of maintaining call stack .

- If recursion is not terminated (or base condition is not specified) than it creates stack overflow (where your system runs out of memory).

- Any recursive problem can be solved iteratively . But you can't solve all problems using recursion.

Program:

```
import java.util.Stack;

public class SortStack {

    public static void main(String[] args) {

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

         st.push(9);

         st.push(-1);

         st.push(120);

         st.push(2);
```

```
   System.out.println(st);
   sort(st);
System.out.println(st);
}
public static void sort(Stack<Integer> st) {
if(st.isEmpty())
{
  return;
}
int temp = st.pop();
sort(st);
insertAtCorrectPosition(st, temp);
}
public static void insertAtCorrectPosition(Stack<Integer> st, int
temp) {
if(st.isEmpty() || st.peek()<temp)
{
   st.push(temp);
    return;
}
int elem = st.pop();
insertAtCorrectPosition(st, temp);
```

st.push(elem);

}}

Explanation of the code :

import java.util.Stack;

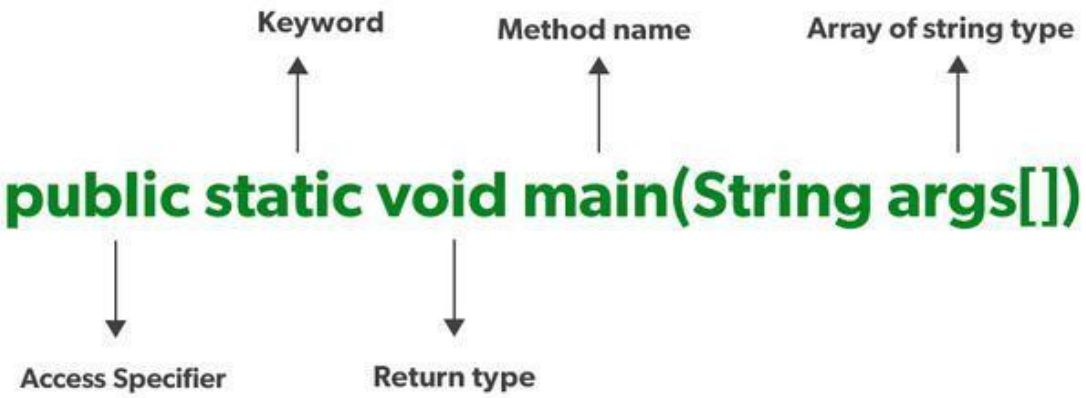
- **We are importing Stack class from util package in java**
- **Stack represents a LIFO stack of Objects.**
- **When a stack is first created, it contains no items.(empty stack).**

public class SortStack{}

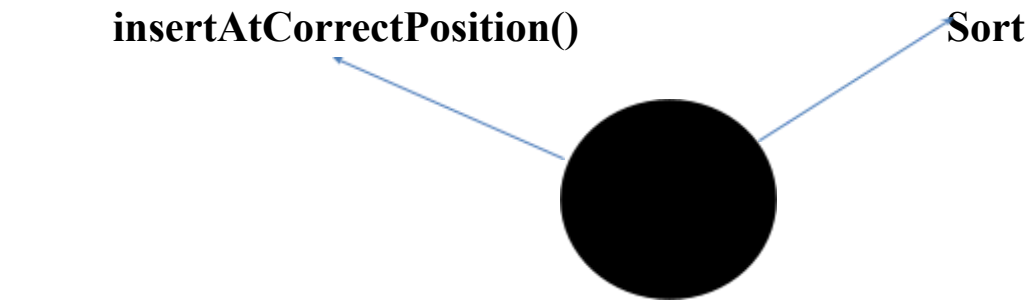
- Your java programs will always start with a class definition. Begin with the word "class" followed by the name of the program(Here,SortStack). Use curly braces to start and end the class definition.
- Public class means it will be available for all other classes in program.

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

- **This is the main Method.ie Main().**
- **The main method is public in Java because it has to be invoked by the JVM. So, if main() is not public in Java, the JVM won't call it. That's all about why the main method is declared public and static in Java.**



Do You Know?
You can write String…args instead of String[] args:
Hence (String… args) is an array of parameters of type String, whereas String[] is a single parameter. String[] can fulfill the same purpose but just (String… args)provides more readability and easiness to use. It also provides an option that we can pass multiple arrays of String rather than a single one using String[].



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

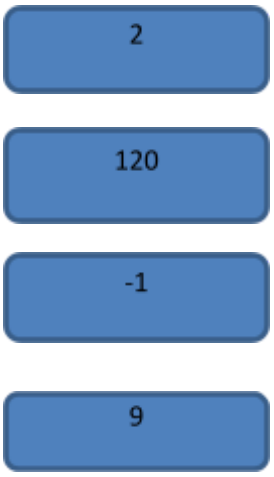
Object

Class

- **Decalaration of Obect (st) of Stack class**

st.push(9);

- **Inserting 9,-1,120,2 into st (object) Stack.**

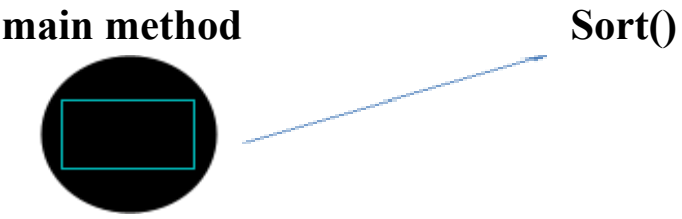


System.out.println(st);

- **Printing the Stack**
- **Output: [9, -1, 120, 2]**

sort(st);

- **Function call ie. The main method is calling sort method**
- **Now what is inside of the sort() is read by the computer.**



public static void sort(Stack<Integer> st) {}

- **Now we define the sort method as: public static void sort()**
- **We have to define st also as:Stack<Integer> st (the same definition it has on main method)**

sort():Ist Recursion

- **We have to pop out all the elements until the stack st gets empty.**

- **And store it in temp variable.**
- **As it is a stack containing a lot of elements it has to be recursed and the function is called until the stack st gets empty and gets stored in temp.**

```
if(st.isEmpty())

{

  return;

}
```

- **If the stack is empty means it has to return to the 2nd Recursion.ie the second method(insertAtCorrectPosition).**

```
int temp = st.pop();
```

- **The element is stored in temp.**
- **The numbers get stored accordingly as it is popped out.**


```
sort(st);
```

- **The same method is called again and again until the stack gets empty.**

```
insertAtCorrectPosition(st, temp);
```

- **The second method is called.**

public static void insertAtCorrectPosition(Stack<Integer> st, int temp) { }

- **Define the function insertAtCorrectPosition using public static void as well as st and temp which are in the parameters.**

```
if(st.isEmpty() || st.peek() < temp)

{

  st.push(temp);

   return;

}
```

- **Now we are checking whether the stack is empty and if so, push the value which is inside the temp variable into the stack.**
- **The second condition is whether the element inside the stack(st.peek()) is compared with the value inside the temp variable.**
- **If temp is greater than the value peeked then just push the value inside the stack.**
- **If not,**

```
int elem = st.pop();
```

- **Store the value popped in elem variable**
- **This will be stored as the call stack value.**

```
insertAtCorrectPosition(st, temp);
```

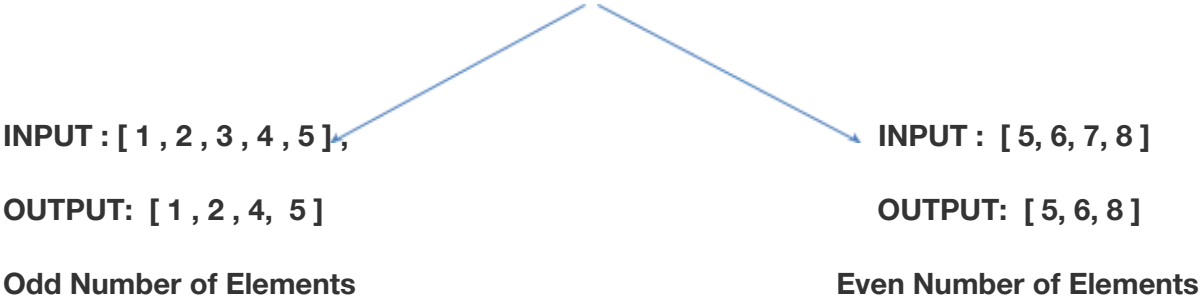
st.push(elem);

- **The function is called again and again till it is being sorted.**
- **The value is pushed from the call stack.**

Now from this program we can learn other two programs that are :

1. **Sorting the stack in ascending order.(if(st.isEmpty() || st.peek() < temp)**
2. **Sorting the stack in descending order.(if(st.isEmpty() || st.peek() > temp)**
3. **Reverse the stack.(if(st.isEmpty())**

2.PROGRAM TO DELETE MIDDLE ELEMENT OF A STACK



(N\2 +1)Element

5/2=2+1=3,so 3rd element deleted

4/2=2+1=3,so 3rd element deleted

Solution Approach

- Recursion
- keep removing the elements one by one from the top of the stack recursively.
- and then at the end push all of them except the middle one.

Steps:

- Declare and initialize a variable named current to 0.
- This “current” will keep record of the position we are at now.
- Pop the top element of the stack.Call the deleteMiddle function after incrementing current by one(which signifies that we are moving on to the next position).

- Keep repeating steps 2 and 3 until the stack is not empty or current is not equal to n.Once the stack is empty or current==n, means that we've popped every element of the stack.
- Now, keep pushing back the elements one by one except for the case where curr==n/2.
- Thus we have the stack now with all the elements except for the middle one.

Space Complexity:

- O(n), where n is the size of the stack.
- **Reason:** We haven't used any other data structure or any other stack. Therefore, the only space taken is the space to store the elements in the stack, i.e; the size of the stack.

Time Complexity:

- O(n), where n is the size of the stack.
- **Reason** : Since we're iterating over the stack recursively by making only one recursive call, which takes O(n) time and popping and pushing operations take only O(1) time, the overall time complexity will be O(n).

Program:

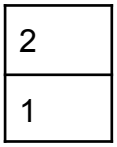
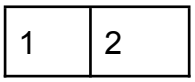
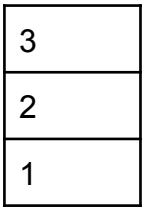
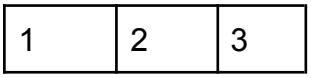
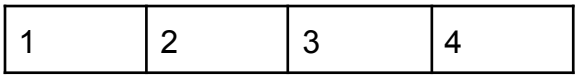
```
import java.io.*;

import java.util.Stack;

 class delMiddle{

   public static void main(String args[]){

  Stack<Character> st = new Stack<Character>();

     st.push('5');

     st.push('4');

     st.push('3');

     st.push('2');

     st.push('1');

     System.out.print("Before deleting:");

     System.out.println(st);
```

```
        System.out.print("After deleting:");
        deleteMiddle(st, st.size(), 0);
        System.out.print(st);
       }
static void deleteMiddle(Stack<Character> st, int n, int curr){
    if(st.empty() || curr == n){
        return;
       }

       char x = st.pop();
       deleteMiddle(st, n, curr+1);

       dele( st, n,curr,x);
   }
    static void dele(Stack<Character> st, int n, int curr, char x){
       if(curr != n/2){
          st.push(x);
       }
   } }
```

IMPLEMENTATION OF STACK USING QUEUE:

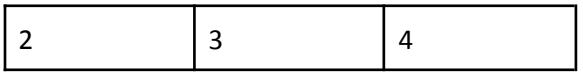
WHAT IS THE DIFFERENCE BETWEEN STACK AND QUEUE:

Stack	Queue
s.push(1); 1	push(1) 1

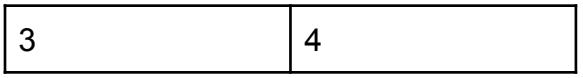
push(2);  	push(2); 
push(3);  	push(3); 
push(4);  	push(4); 

<table>
<tr>
<td>pop();
if Pop() The last element inserted (4) will be removed from the stack.(LIFO)

 
</td>
<td>pop();
if Pop() The first element inserted (1) will be removed from the stack.(FIFO)

 
</td>
</tr>
<tr>
<td>top();
3</td>
<td>top();
2</td>
</tr>
<tr>
<td>pop();
if Pop() again the last element(3) will be removed.

 
</td>
<td>pop();
if Pop() again the first element(2) will be removed.

 
</td>
</tr>
<tr>
<td>top();
2</td>
<td>top();
3</td>
</tr>
<tr>
<td>Remaining stack:

2
1</td>
<td>Remaining Queue:

3, 4</td>
</tr>
</table>

Now the implementation of stack using queue is that:

- the execution of all the functions in a queue will be same as that of stack
- the remaining queue elements will be the same as that of the stack.

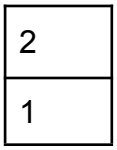
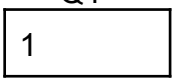
To do this, one queue is not enough, we need 2(Queue 1 and Queue 2)

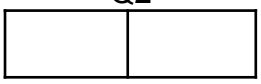
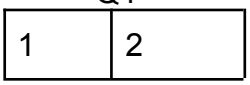
Implementation can be done in two ways.

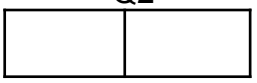
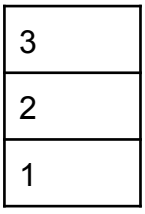
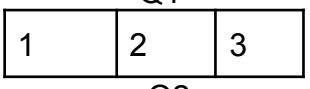
1. pop operation costly
2. push operation costly

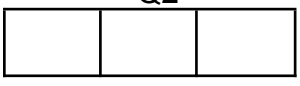
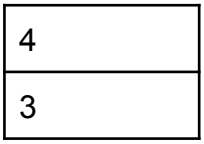
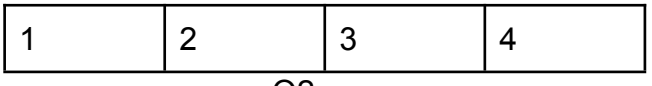
1.pop operation costly

<table>
<tr><th>Stack</th><th>Queue</th></tr>
<tr><td>s.push(1);
 

s.push(2);
 
</td><td>Q1.push(1)
Q1
 

Q2
 
Q1.push(2);
Q1
 

Q2
  </td></tr>
<tr><td>s.push(3);
 
</td><td>Q1.push(3);
Q1
 

Q2
  </td></tr>
<tr><td>s.push(4);
 
</td><td>Q1.push(4);
Q1
 

Q2</td></tr>
</table>

<table>
<tr>
<td>
<table>
<tr><td>2</td></tr>
<tr><td>1</td></tr>
</table>
</td>
<td>
<table>
<tr><td></td><td></td><td></td><td></td></tr>
</table>
</td>
</tr>
<tr>
<td>
s.pop();
if Pop() The last element inserted (4) ,will be removed from the stack.(LIFO)

<table>
<tr><td>3</td></tr>
<tr><td>2</td></tr>
<tr><td>1</td></tr>
</table>

We have 1,2,3 elements in the remaining stack.
</td>
<td>
Q1.pop();
The principle of queue is FIFO.So to get 4 removed from the queue as same as in stack we need to: Leave one element in q1 and push others in q2.

while (q1.size() != 1) {
q2.add(q1.peek());
q1.remove();
Till the size of Q1 becomes 1,the Q1 peeked element will be pushed into Q2, and then those elements are removed from Q1.

Q1
<table>
<tr><td>4</td><td></td><td></td><td></td></tr>
</table>

Q2
<table>
<tr><td>1</td><td>2</td><td>3</td><td></td></tr>
</table>

And now if I pop from Q1,then 4 is removed.And to regain Q1 with the same elements except 4, we just interchange the names of the queues.That is Q1=Q2.Now Q1 is back with elements.1,2,3 as in the stack.

Q1
<table>
<tr><td>1</td><td>2</td><td>3</td><td></td></tr>
</table>
Q2
<table>
<tr><td></td><td></td><td></td><td></td></tr>
</table>
</td>
</tr>
</table>

s.top();
3
	Q1.front();The principle of queue is FIFO.So to get 3 as the front element of the queue as same as in stack we need to: Leave one element(last one) in q1 and push others in q2.
	while (q1.size() != 1) {
	q2.add(q1.peek());
	q1.remove();
	Till the size of Q1 becomes 1,the Q1 peeked element will be pushed into Q2, and then those elements are removed from Q1.
	Q1
	3
	Q2
	1 2
	Now a temp variable is introduced and then that element is stored
	int temp = q1.peek();
	returned temp in top function so that the temp value will be the peek value of Q1.After that again it is removed from Q1 and pushed to Q2.
	q1.remove();
	q2.add(temp);
	And to regain Q1 with the same elements, we just interchange the names of the queues.That is Q1=Q2.Now Q1 is back with elements.1,2,3 as in the stack. 3 is not removed as the function is front() and not pop().
	Q1
	1 2 3
	Q2

<table>
<tr><td></td><td></td></tr>
<tr><td>s.pop();
if Pop() again the last element(3) will be removed.

<table><tr><td>2</td></tr><tr><td>1</td></tr></table></td><td>Q1.pop();
The principle of queue is FIFO.So to get 3 removed from the queue as same as in stack we need to: Leave one element in q1 and push others in q2.

while (q1.size() != 1) {
q2.add(q1.peek());
q1.remove();
Till the size of Q1 becomes 1,the Q1 peeked element will be pushed into Q2, and then those elements are removed from Q1.
Q1
<table><tr><td>3</td><td></td><td></td></tr></table>
Q2
<table><tr><td>1</td><td>2</td><td></td></tr></table>
And now if I pop from Q1,then 3 is removed.And to regain Q1 with the same elements except 3, we just interchange the names of the queues.That is Q1=Q2.Now Q1 is back with elements.1,2 as in the stack.

Q1
<table><tr><td>1</td><td>2</td><td></td><td></td></tr></table>
Q2
<table><tr><td></td><td></td><td></td><td></td></tr></table></td></tr>
<tr><td>top();
2</td><td>Q1.front();
The principle of queue is FIFO.So to get 2 as front element of the queue as same as in stack we need to: Leave one element(last one) in q1 and push</td></tr>
</table>

<table>
<tr>
<td></td>
<td>
others in q2.
<code>while (q1.size() != 1) {
q2.add(q1.peek());
q1.remove();</code>
Till the size of Q1 becomes 1,the Q1 peeked element will be pushed into Q2, and then those elements are removed from Q1.
Q1
<table><tr><td>2</td><td></td><td></td><td></td></tr></table>
Q2
<table><tr><td>1</td><td></td><td></td><td></td></tr></table>

Now a temp variable is introduced and then that element is stored
<code>int temp = q1.peek();</code>
returned temp in top function so that the temp value will be the peek value of Q1.After that again it is removed from Q1 and pushed to Q2.
<code>q1.remove();</code>
<code>q2.add(temp);</code>
And to regain Q1 with the same elements, we just interchange the names of the queues.That is Q1=Q2.Now Q1 is back with elements.1,2 as in the stack. 2 is not removed as the function is front() and not pop().
Q1
<table><tr><td>1</td><td>2</td><td></td><td></td></tr></table>
Q2
<table><tr><td></td><td></td><td></td><td></td></tr></table>
</td>
</tr>
<tr>
<td>Remaining stack:

2
1</td>
<td>Remaining Queue:

1, 2</td>
</tr>
</table>

Program:

```
import java.util.Queue;
import java.util.LinkedList;
public class BBQ2 {
    static class Stack{
     static Queue<Integer> q1 = new LinkedList<>();
     static Queue<Integer> q2 = new LinkedList<>();
    void pop()
    {
        if (q1.isEmpty())
            return;

        // Leave one element in q1 and
        // push others in q2.
        while (q1.size() != 1) {
            q2.add(q1.peek());
            q1.remove();
        }

        // Pop the only left element
        // from q1
        q1.remove();

        // swap the names of two queues
```

```
    Queue<Integer> q = q1;
    q1 = q2;
    q2 = q;
}
 int size()
{
    return q1.size();
}

void add(int x)
{
    q1.add(x);
}

int top()
{
    if (q1.isEmpty())
        return -1;

    while (q1.size() != 1) {
        q2.add(q1.peek());
        q1.remove();
    }
```

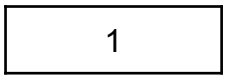
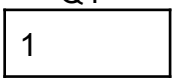
```
    // last pushed element
    int temp = q1.peek();

    // to empty the auxiliary queue after
    // last operation

    q1.remove();

    // push last element to q2
    q2.add(temp);

    // swap the two queues names
    Queue<Integer> q = q1;
    q1 = q2;
    q2 = q;
    return temp;
}
public static void main(String[] args)
{
    Stack s = new Stack();
    s.add(1);
    s.add(2);
    s.add(3);
    s.add(4);
```

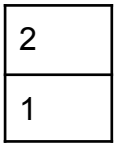
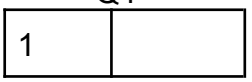
```
        s.pop();
        System.out.println(s.top());
        s.pop();
        System.out.println(s.top());
        s.pop();
        System.out.println(s.top());

    }
}
}
```

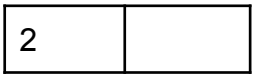
2.push operation costly

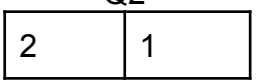
Note:If any element is inserted to Q2, then all the elements from the Q1 until it gets empty needed to be shifted to Q2.and the Name is interchanged(swap)

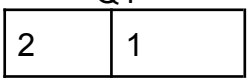
Stack	Queue

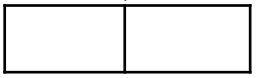
s.push(1);
 
s.push(2);
	Q1.push(1)
	we can directly push element to Q2 also.But the character of Q2 will be in such a way that, when any element is added to it, all the elements from Q1 has to be shifted to it.
	Q1
	 
	Q2

<table>
<tr>
<td> 
</td>
<td> 
Q2.push(2);
Q1
 

Q2
 

Now, after inserting the first element in Q2,All the elements from Q1 are shifted to Q2 until Q1 gets empty.
while (!q1.isEmpty()) {
q2.add(q1.peek());
q1.remove();
}
How the element is shifted from Q1 to Q2:
• checked whether the Q1 is empty.(In this case,Q1 is having 1.
• So q1.peek() will return 1.
• Now this peeked element is added to q2.
• And the q1.remove function is removed.
• again it will check whether it is empty or not.In this case,it is empty so the function inside the while loop will not work.
Q2
 

And to regain Q1 with the same elements, we just interchange the names of the queues.That is Q1=Q2.Now Q1 is back with elements.1,2 as in the stack.
Now the status of Q1 is :
Q1
 

Q2
  </td>
</tr>
<tr>
<td>s.push(3);</td>
<td>Q2.push(3);
Q1</td>
</tr>
</table>

3
2
1
	2 1
	Q2
	3
	Now, after inserting the element in Q2,All the elements are from Q1 is shifted to Q2 till Q1 gets empty.
	while (!q1.isEmpty()) {
	q2.add(q1.peek());
	q1.remove();
	How the element is shifted from Q1 to Q2:
	• checked whether the Q1 is empty.(In this case,Q1 is having 2,1.
	• So q1.peek() will return 2.
	• Now this peeked element is added to q2.
	• And the q1.remove function will remove 2 from q1.Now q1 is left with 1
	•
	• Q1
	1
	•
	• Again the function will check whether q1 is empty.in this case it is not empty(1 is present).
	• So q1.peek() will return 1.
	• Now this peeked element is added to q2.
	• And the q1.remove function will remove 1 from q1.Now q1 is empty.
	•
	Q1
	Q2
	3 2 1
	And to regain Q1 with the same elements, we just interchange the names of the queues.That is Q1=Q2.Now Q1 is back with elements.1,2,3 as in the stack.
	Q1
	3 2 1
	Q2

s.push(4);

4
3
2
1

Q2.push(4);

Q1

3	2	1	

Q2

4			

Now, after inserting the element in Q2,All the elements are from Q1 is shifted to Q2 till Q1 gets empty.

```
while (!q1.isEmpty()) {
        q2.add(q1.peek());
        q1.remove();
```

How the element is shifted from Q1 to Q2:

- checked whether the Q1 is empty.(In this case,Q1 is having 3,2,1.
- So q1.peek() will return 3.
- Now this peeked element is added to q2.
- And the q1.remove function will remove 3 from q1.Now q1 is left with 2,1.
-
- Q1

2	1	

- Q2

4	3		

- Again the function will check whether q1 is empty.in this case it is not empty(2,1 is present).
- So q1.peek() will return 2.
- Now this peeked element is added to q2.
- And the q1.remove function will remove 2 from q1.Now q1 is left with 1.
-
- Q1

1		

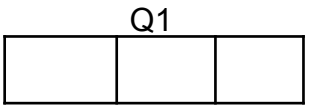
- Q2

4	3	2	

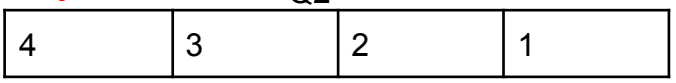
- Q1 is checked whether it is empty or not.Q1 is left with 1.

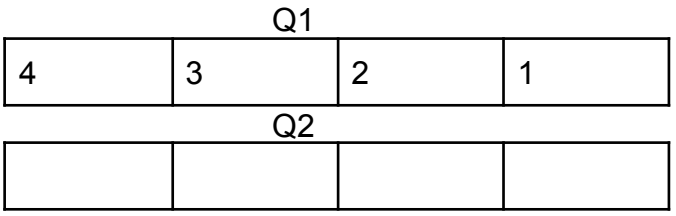
<table>
<tr>
<td></td>
<td>

So q1.peek() will return 1.
Now this peeked element is added to q2.
And the q1.remove function will remove 1 from q1.Now q1 is empty..

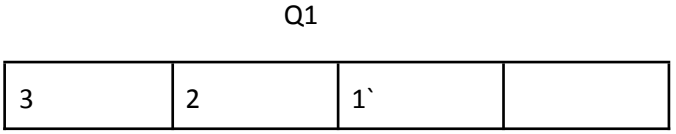


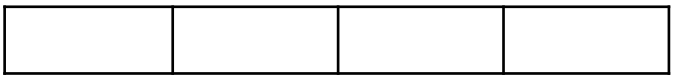
Q2

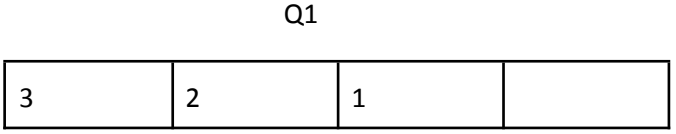


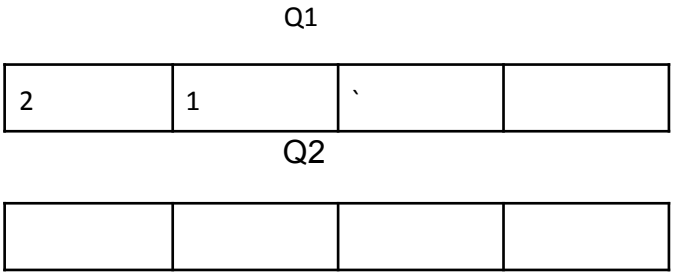
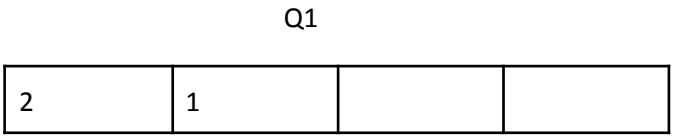
And to regain Q1 with the same elements, we just interchange the names of the queues.That is Q1=Q2.Now Q1 is back with elements.1,2,3 as in the stack.


</td>
</tr>
<tr>
<td>s.pop();
if Pop() The last element inserted (4) ,will be removed from the stack.(LIFO)
 

We have 1,2,3 elements in the remaining stack.</td>
<td>Q1.pop();
q1.pop() will remove 4 from Q1 as same as in stack.
 

We have 1,2,3 elements in the remaining queue.</td>
</tr>
<tr>
<td>s.top();
3</td>
<td>Q1.front();
q1.front() will return 3(because queue follows First in First Out (FIFO)), as same as in stack.
 
</td>
</tr>
</table>

s.pop(); if Pop() again the last element(3) will be removed.  We have 1,2, elements in the remaining queue.	Q1.pop(); q1.pop() will remove 3 from Q1 as same as in stack.  We have 1,2, elements in the remaining queue.
top(); 2	Q1.front(); q1.front() will return 2 (because queue follows First in First Out (FIFO)), as same as in stack. 
Remaining stack: 2 1	Remaining Queue: 1, 2

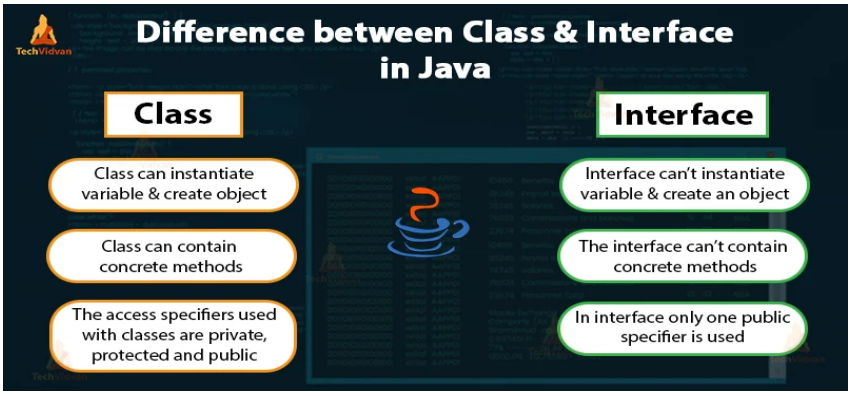
```
import java.util.Queue;
```

```
import java.util.LinkedList;
public class BBQ {

    static class Stack {
        // Two inbuilt queues
        static Queue<Integer> q1 = new LinkedList<Integer>();
        static Queue<Integer> q2 = new LinkedList<Integer>();

        // To maintain current number of
        // elements


        static void push(int x)
        {
            // Push x first in empty q2
            q2.add(x);

            // Push all the remaining
            // elements in q1 to q2.
            while (!q1.isEmpty()) {
                q2.add(q1.peek());
                q1.remove();
```

```
    }

    // swap the names of two queues
    Queue<Integer> q = q1;
    q1 = q2;
    q2 = q;
}

static void pop()
{

    // if no elements are there in q1
    if (q1.isEmpty())
        return;
    q1.remove();
}

static int top()
{
    if (q1.isEmpty())
        return -1;
    return q1.peek();
```

```
    }



}

// driver code
public static void main(String[] args)
{
    Stack s = new Stack();
    s.push(1);
    s.push(2);
    s.push(3);
    s.push(4);

    s.pop();
    System.out.println(s.top());
    s.pop();

    System.out.println(s.top());
    s.pop();
    System.out.println(s.top());
```

```
    }
}
```



The Queue interface is present in java. util package

A concrete class is a class that has an implementation for all of its methods. They cannot have any unimplemented methods. It can also extend an abstract class or implement an interface as long as it implements all their methods. It is a complete class and can be instantiated.

Linear Search Algorithm

- sequential search algorithm
- starts at one end and goes through each element of a list until the desired element is found.
- Otherwise the search continues till the end of the data set.

Examples:

Input: arr[] = {10, 20, 80, 30, 60, 50,110, 100, 130, 170}, x(the target element) = 110;

Output: 6

Explanation: Element x is present at index 6

Input: arr[] = {10, 20, 80, 30, 60, 50,110, 100, 130, 170}, x = 175;

Output: -1

Explanation: Element x is not present in arr[].

It can be solved in two ways:1)Iteration,2)Recursion

Program:

```
class LinearSearch
 {
  public static int linearSearch(int array[], int x)
{
  int n = array.length;
       for (int i = 0; i < n; i++)
        {
                if (array[i] == x)
                      return i;
        }
              return -1;
  }

  public static void main(String args[]) {
  int array[] = { 2, 4, 0, 1, 9 };
```

```
  int x = 1;


  int result = linearSearch(array, x);


  if (result == -1)
    System.out.println("Element not found");
  else
    System.out.println("Element found at index: " + result);
  }
}
```

Explanation of the code:

Before moving on to the program,the concept of array can be learnt.

Array:

Arrays are used to store multiple values in a single variable, instead of declaring separate variables for each value.

To declare an array, define the variable type with square brackets:

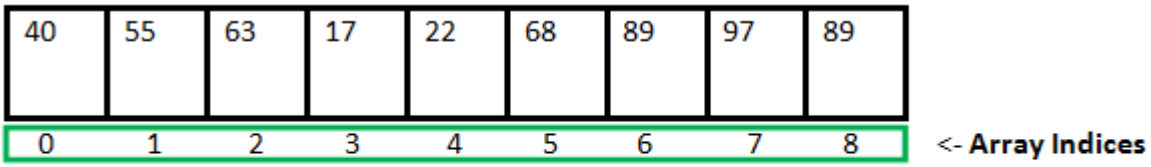
place the values in a comma-separated list, inside curly braces:

Examples:

```
  int array[] = { 2, 4, 0, 1, 9 }
```

```
String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};
```

In an array the index starts from 0.That is if we have 5 spaces and n=5 and we can store 5 elements in it where the element starts its location from 0.



Array Length = 9
First Index = 0
Last Index = 8

Starting with the main function,

- **Declaration of Array:** `int array[] = { 2, 4, 0, 1, 9 };`

  int is the type of array elements.

  array[] is the name of the array

  elements will be stored in {.....}that is enclosing the curly brackets.

- **Here, the array is declared with elements 2, 4, 0, 1, 9 .**

`int x = 1;`

- **Now you have to store the element which has to be searched in a variable.Here we are storing it in a variable ‘ x ’.**
- **Why are we storing it in a variable ?**

  First we cannot leave the element as such.It will be easy for us to check by giving it a name and storing it in that name.Same as your friends call you a nickname,that is instead of calling you your long name.. they will put a short name for you which is convenient for u as well as them.Same happens here also..we are naming the element as a name which is short and convenient for us.(usually we use x,y,a,b..)

`int result = linearSearch(array, x);`

- **This is known as function calling.**

- **linearSearch() is the function with parameters such as array and x.**
- **result is another variable wherein the product of linearSearch() is stored.That is something is returned from linearSearch().**

Let's move on to the linearSearch(),

```
public static int linearSearch(int array[], int x)
```

- **define the method as public static and int (as we are returning some values and that value is stored in the result variable which is declared in the main function).The parameters should also be defined using return type).**

```
int n = array.length;
```

- **n is another variable wherein the length of the array is stored.**

Now everything is set for the searching if the element.As this is linear search, we need to move to each and every position and check whether the element is equal to x(that is what is stored in x variable).For that we use loop.

```
    for (int i = 0; i < n; i++)
      {
            if (array[i] == x)
                  return i;
       }
          return -1;
 }
```

- **i is the index value and it starts from 0 and ends at n-1(so we can give i<n) .**
- **Now if condition is given wherein the value of array is checked with the value of x.If the statement is true, then the**

value of i(that is the position of that element) will be returned.

- or else -1 is returned.
- Now there can be two possibilities of returning the value:
    1. value of i
    2. -1
- The returned value will be stored in result variable as this is the product returned from linearSearch().

Now moving on to the main function:

- if condition is given and the value of result is checked ..That is whether it is -1 or not .
- If -1, we have to print The element is not found
- if not ,we have to print The element is found and the value is displayed using + .

Binary Search Algorithm

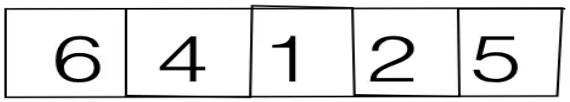
Bubble Sort

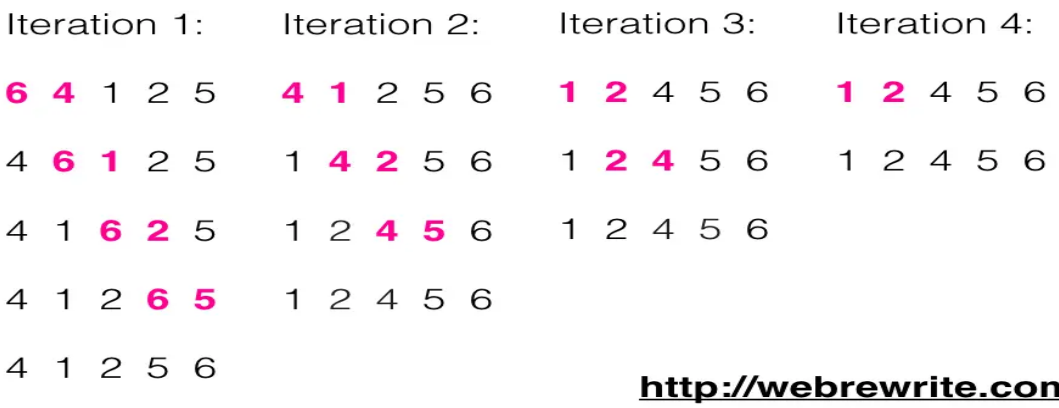
Bubble sort is a very basic and simple sorting algorithm which sort an unsorted array.

It works by comparing each pair of adjacent element and swap them if they are in a wrong order.

Bubble sort is not efficient for large data set.

Example:





Reverse queue:

```
public static void reverseQueue(Queue<Integer> q)

{
```

```
        // Base case
        if (q.isEmpty())
    {
        return;
    }

        // Dequeue current item (from front)
        int data = q.peek();
        q.remove();

        // Reverse remaining queue
        reverseQueue(q);

        // Enqueue current item (to rear)
        q.add(data);



}
// Driver code
public static void main(String...args) {
Queue<Integer> q = new LinkedList<Integer>();
    q.add(1);
        q.add(2);
        q.add(3);
        q.add(4);
        System.out.println(q);
        reverseQueue(q);
```

```
        System.out.println(q);
}
}
```

Check:

```
import java.util.Stack;
import java.util.LinkedList;
import java.util.Queue;

class checkSorted
{


    static boolean check(int n, Queue<Integer> q )
    {
  Stack<Integer> st = new Stack<Integer>();
        int i= 1;
        int fnt;

        // while given Queue
        // is not empty.
        while (q.size() != 0)
        {
            fnt = q.peek();
            q.poll();
```

```
// if front element is
// the expected element
if (fnt == i)
    i++;

else
{
    // if stack is empty,
    // push the element
    if (st.size() == 0)
    {
        st.push(fnt);
    }

    // if top element is less than
    // element which need to be
    // pushed, then return false.
    else if (st.size() != 0 &&
             st.peek() < fnt)
    {
        return false;
    }

    // else push into the stack.
    else
        st.push(fnt);
```

```
        }

        // while expected element are
        // coming from stack, pop them out.
        while (st.size() != 0 &&
               st.peek() == i)
        {
            st.pop();
            i++;
        }
    }

    // if the final expected element
    // value is equal to initial Queue
    // size and the stack is empty.
    if (i - 1 == n &&
            st.size() == 0)
        return true;

    return false;
  }

  // Driver Code
  public static void main(String args[])
  {
Queue<Integer> q = new LinkedList<Integer>();
```

```
        q.add(5);
        q.add(6);
        q.add(1);
        q.add(2);
        q.add(3);
        q.add(4);

        int n = q.size();

        if (check(n,q))
            System.out.print("Yes");
        else
            System.out.print("No");
    }
}
```

DSA – Practice Questions:

----------Learn Objective Type of Questions----------

1. **Linear Data Structures, and NonLinear data structures.(Subjective)**
2. **Asymptotic Notations ..Types(subjective)**
3. **Reverse a queue using recursion(subjective)**
4. **Reverse a stack using recursion(subjective)**

—----------

1. Queue is implemented using linked list.front and rear pointers are taken into notice.Which of the following pointers change during insertion into Empty Queue?//what is the change in the value as it moves

a.Both front and Queue
b)Only front
c)Only rear
d)None

Answer:A

2. Identify the postfix representation of the given infix expression
(X + Y) * Z – W * E / F

a. (X + Y) * Z – W * E / F
b. X Y + Z * W E * F - /
c. X Y * Z + W E * F / -
d. X Y + Z – W E * F / *

Answer:D

3.Which data structure is mainly used for implementing the recursive algorithm?//same answer for the question

Which data structure is mainly used for implementing the infix to postfix evaluation?//same answer for the question

Which data structure is mainly used for implementing the Parenthesis analysis?

a. Queue
b. Binary tree
c. Linked list
d. Stack

Answer:Stack

4. If the user tries to delete the element from the empty stack it is called ____

a. Garbage collection
b. Overflow
c. Underflow
d. None of the above

5. Consider the linked list implementation of a stack,which node is considered as Top.

Answer:First Node

6. Result of Top(Push(S,A)):

7. Stack can be implemented using ---------- and ----------:

a.Array and Binary tree

b.Linked List and Graph

c.Array and Linked List

d.Queue and Linked List

8. ---------- form of access is used to add and remove nodefrom a queue:

a.LIFO

b.FIFO

c.Both A and B

d.None

9. Application of Stack:

a.Fibonacci

b.Tower of Hanoi

c.Infix to Postfix

d.All of the above

10. Reversing a great deal of space for each stack in memeory will --------

a.Increase the number of times overflow may occur.

b.Decrease the number of times overflow may occur.

c.Increase the number of times underflow may occur.

d.Decrease the number of times overflow may occur.

11.Data Structure in which elements can be inserted or deleted from both the ends but not in the middle is?

a.Queue

b.Circular Queue

c.Dequeue

d.Priority Queue

12. Assume an array with a maximum capacity of 10 elements. If the user adds one element to an array. In example 2, the user creates an array with 10 elements. What is the space complexity in both cases?

a. O(n), O(1)
b. O(1), O(1)
c. O(n^2), O(1)
d. None.

13. What is the disadvantage of array data structure?

a. The amount of memory to be allocated should be known beforehand.
b. Elements of an array can be accessed in constant time.
c. Elements are stored in contiguous memory blocks.
d. Multiple data structures can be implemented using arrays.

14. Which one of the following is the size of int arr[3] assuming that int is of 4 bytes?

a. 9
b. 36
c. 12
d. None of the above

15. A queue data-structure can be used for

a.Expression parsing

b.recursion

c.resource allocation

d.all of the above

16. What data structure is used for breadth first traversal of a graph?queue

17. What will be the initial value with which top is initialized. Ans=-1

18. What data structure is used for depth first traversal of a graph?Stack

19. Identify an abstract data structure from the following

a) Graphs
b) Queue
c) Tree

d) Functions

20.A normal queue, if implemented using an array of size MAX_SIZE, gets full when?
a)Rear = MAX_SIZE – 1
b) Front = (rear + 1)mod MAX_SIZE
c) Front = rear + 1
d) Rear = front

21.A linear list of elements in which deletion can be done from one end (front) and insertion can take place only at the other end (rear) is known as _____
a)Queue
b)Stack
c)Tree
d) Linked list

22.In linked list implementation of a queue, front and rear pointers are tracked. Which of these pointers will change during an insertion into EMPTY queue?

a. Both front and rear pointer
b. Only front pointer
c. Only rear pointer
d. None

23.Which of the following is not the type of queue?

a. Ordinary queue
b. Circular queue
c. Priority queue
d. Single ended queue

24.Stack is used for

A. CPU Resource Allocation
B. Breadth First Traversal
C. Recursion
D. None of the above

25.In the stack, If user try to remove element from the empty stack then it called as

A. Empty Collection
B. Underflow of Stack
C. Garbage Collection
D. Overflow of Stack

26.Which of the following is an application of stack?

A. finding factorial
B. tower of Hanoi
C. infix to postfix
D. all of the above

27.Which is the pointer associated with the stack?

A. FIRST
B. FRONT
C. TOP
D. REAR

28.. New nodes are added to the ……. of the queue.

A. Front
B. Back
C. Middle
D. Both A and B

29.. Deletion operation is done using ……. in a queue.

A. front
B. rear
C. top
D. list

30. In linked representation of stack ……. holds the elements of the stack.

A. INFO fields
B. TOP fields
C. LINK fields
D. NULL fields

31. User push 1 element in the stack having already five elements and having stack size as 5 then stack becomes

A. Overflow
B. Underflow
C. User Flow
D. Crash

32. Act of adding values into a stack is called

A. Popping
B. Polling
C. Pushing
D. None

33. push() and pop() functions are found in

A. queues
B. lists
C. stacks
D. trees

34. The elements are removal from a stack in ……. order.

A. Reverse
B. Hierarchical
C. Alternative
D. Sequential

35.The stack size is 5.Push(a),pop(),push(b),push(c),pop(),push(d),pop(),pop(),push(e).Select the correct statement:

A.Underflow occurs

B.Stack operations Performed smoothly

C.Overflow

D.None

36.A null pointer in the last node signals

a)Beginning of the stack

b)Bottom

c)middle

d)In between some value

37. Which one of the following is an application of Queue Data Structure?

A. When a resource is shared among multiple consumers.

B. When data is transferred asynchronously (data not necessarily received at same rate as sent) between two processes

c. Load Balancing

d.All of the above

38.How many stacks are needed to implement a queue. Consider the situation where no other data structure like arrays, linked list is available to you.

A.1

B.2

C.3

D.4

39.How many queues are needed to implement a stack. Consider the situation where no other data structure like arrays, linked list is available to you.

A.1

B.2

C.3

D.4

40. Which of the following is true about linked list implementation of queue?

A. In push operation, if new nodes are inserted at the beginning of linked list, then in pop operation, nodes must be removed from end.

B. In push operation, if new nodes are inserted at the end, then in pop operation, nodes must be removed from the beginning

C. Both of the above

D.None

41. Suppose a circular queue of capacity (n – 1) elements is implemented with an array of n elements. Assume that the insertion and deletion operation are carried out using REAR and FRONT as array index variables, respectively. Initially, REAR = FRONT = 0. The conditions to detect queue full and queue empty are:

A. Full: (REAR+1) mod n == FRONT, empty: REAR == FRONT

B. Full: (REAR+1) mod n == FRONT, empty: (FRONT+1) mod n == REAR

C. Full: REAR == FRONT, empty: (REAR+1) mod n == FRONT

D. Full: (FRONT+1) mod n == REAR, empty: REAR == FRONT

42. Which one of the following is an application of Stack Data Structure?

A. Managing function calls

B. The stock span problem

C. Arithmetic expression evaluation

D. All of the above

43. Which one of the following is an application of Stack Data Structure?

A. Managing function calls

B. The stock span problem

C. Arithmetic expression evaluation

D. All of these

44. Which one of the following is an application of Queue Data Structure?

A. When data is transferred asynchronously (data not necessarily received at same rate as sent) between two processes

B. When a resource is shared among multiple consumers.

C. Load Balancing

D. All of the above

45.The five items K, L, M, N and O are pushed in a stack, one after the other starting from O. The stack is popped four times and each element is stored in a queue. Then three elements are deleted from the queue. Remaining element in queue is:

a. K
b. N and O
c. M
d. N

46. A data structure nin which deletion can be done from one end (front) and insertion can take place only at the other end (rear) is known as _____

a)Queue

b)Stack

c)Tree

d) Linked list

Ans:a

47.Advantage and disadvantage of arrays

47.Abstract Data type:Stack and Queue

48.queue

add(1)....add(2).....add(3) -------FIFO(First IN FIRST OUT)

The one who comes first goes out first.So here 1 goes out when remove() is called.

1	2	3

STACK:

push(1)....push(2).....push(3) -------FILO(First IN Last OUT)

The one who comes first goes out last.So here 3 goes out when pop() is called.

3
2
1

49. Which of these is an incorrect Statement?

a) It is necessary to use new operator to initialize an array

b) Array can be initialized using comma separated expressions surrounded by curly braces

c) Array can be initialized when they are declared

d) None of the mentioned

Ans:a

50.Which of these is an incorrect array declaration?

a) int arr[] = new int[5]

b) int [] arr = new int[5]

c) int arr[] = new int[5]

d) int arr[] = int [5] new

Ans:D

51.Which of the following can be used to implement stack?

i) Array

ii) Linked List

iii) Tree

iv) Graph

Stack can be implemented using array and linked list.

52.Which statement is correct with respect to stack?

1) It is a non-linear data structure.

2) Stack is a LIFO data structure

The correct answer is 2.

53. The term Push and Pop is related to

a) Queue

b) Stack

c) Both

d) None

54. Choose correct output for the following sequence of operations.

push(5)

push(8)

pop

push(2)

push(5)

pop

pop

pop

push(1)

pop

a) 8 5 2 5 1

b) 8 5 5 2 1

c) 8 2 5 5 1

d) 8 1 2 5 5

55. In which data structure, element is inserted at one end called Rear and deleted at other end called Front.

a) Stack

b) Queue

c) Both

d) Binary Tree

56. Stack can be implemented using _____ and _____ ?

a) Array and Binary Tree

b) Linked List and Graph

c) Array and Linked List

d) Queue and Linked List

57. Postfix form of following expression.

D + (E * F)

a) EF * D+

b) DEF * +

c) DEF +*

d) EFD *+

58. When the function calls another function then the details of the previous function are stored in Stack?

a) True

b) False

59. Insertion and Deletion operation in Queue is known as?

a) Push and Pop

b) Enqueue and Dequeue

c) Insert and Delete

d) None

60. A stack data structure cannot be used for

a) Implementation of Recursive Function

b) Allocation Resources and Scheduling

c) Reversing string

d) Evaluation of string in postfix form

61. Which of the following principle does queue use?

a) LIFO

b) FIFO

c) Both

d) None of the above

62. The process of inserting an element in the stack is called?

a) Enqueue

b) Insert

c) Push

d) Pop