“Stack using Array”.

1. Which of the following real world scenarios would you associate with a stack data structure?
a) piling up of chairs one above the other
b) people standing in a line to be serviced at a counter
c) offer services based on the priority of the customer
d) tatkal Ticket Booking in IRCTC
View Answer
Answer: a
Explanation: Stack follows Last In First Out (LIFO) policy. Piling up of chairs one above the other is based on LIFO, people standing in a line is a queue and if the service is based on priority, then it can be associated with a priority queue. Tatkal Ticket Booking Follows First in First Out Policy. People who click the book now first will enter the booking page first.

2. What does the following function check for? (all necessary headers to be included and function is called from main)

```
#define MAX 10

typedef struct stack
{
    int top;
    int item[MAX];
}stack;

int function(stack *s)
{
    if(s->top == -1)
        return 1;
    else return 0;
}
```

a) full stack
b) invalid index
c) empty stack
d) infinite stack
View Answer
Answer: c
Explanation: An empty stack is represented with the top-of-the-stack(‘top’ in this case) to be equal to -1.

. What does ‘stack underflow’ refer to?
a) accessing item from an undefined stack
b) adding items to a full stack
c) removing items from an empty stack
d) index out of bounds exception
View Answer
Answer: c
Explanation: Removing items from an empty stack is termed as stack underflow.

4. What is the output of the following program?

```
public class Stack
{
          protected static final int CAPACITY = 100;
          protected int size,top = -1;
          protected Object stk[];

          public Stack()
          {
                    stk = new Object[CAPACITY];
          }

          public void push(Object item)
          {
                    if(size_of_stack==size)
                    {
                              System.out.println("Stack overflow");
                                        return;
                    }
                    else
                    {
                              top++;
                              stk[top]=item;
                    }
          }
          public Object pop()
          {
                    if(top<0)
                    {
                              return -999;
                    }
                    else
```

```
                {
                        Object ele=stk[top];
                        top--;
                        size_of_stack--;
                        return ele;
                }
        }
}

public class StackDemo
{
        public static void main(String args[])
        {
                Stack myStack = new Stack();
                myStack.push(10);
                Object element1 = myStack.pop();
                Object element2 = myStack.pop();
                System.out.println(element2);
        }
}
```

a) stack is full
b) 20
c) 0
d) -999
View Answer
Answer: d
Explanation: The first call to pop() returns 10, whereas the second call to pop() would result in stack underflow and the program returns -999.

5. What is the time complexity of pop() operation when the stack is implemented using an array?
a) O(1)
b) O(n)
c) O(logn)
d) O(nlogn)
View Answer
Answer: a
Explanation: pop() accesses only one end of the structure, and hence constant time.

6. Which of the following array position will be occupied by a new element being pushed for a stack of size N elements(capacity of stack > N)?
a) S[N-1]

b) S[N]
c) S[1]
d) S[0]
View Answer
Answer: b
Explanation: Elements are pushed at the end, hence N.

7. What happens when you pop from an empty stack while implementing using the Stack ADT in Java?
a) Undefined error
b) Compiler displays a warning
c) EmptyStackException is thrown
d) NoStackException is thrown
View Answer
8. What is the functionality of the following piece of Java code?
Assume: 'a' is a non empty array of integers, the Stack class creates an array of specified size and provides a top pointer indicating TOS(top of stack), push and pop have normal meaning.

```
public void some_function(int[] a)
{
        Stack S=new Stack(a.length);
        int[] b=new int[a.length];
        for(int i=0;i<a.length;i++)
        {
                S.push(a[i]);
        }
        for(int i=0;i<a.length;i++)
        {
                b[i]=(int)(S.pop());
        }
        System.out.println("output :");
        for(int i=0;i<b.length;i++)
        {
                System.out.println(b[i]);
        }
}
```

a) print alternate elements of array
b) duplicate the given array
c) parentheses matching

d) reverse the array
View Answer
Answer: d
Explanation: Every element from the given array ‘a’ is pushed into the stack, and then the elements are popped out into the array ‘b’. Stack is a LIFO structure, this results in reversing the given array.

9. Array implementation of Stack is not dynamic, which of the following statements supports this argument?
a) space allocation for array is fixed and cannot be changed during run-time
b) user unable to give the input for stack operations
c) a runtime exception halts execution
d) improper program compilation
View Answer
Answer: a
Explanation: You cannot modify the size of an array once the memory has been allocated, adding fewer elements than the array size would cause wastage of space, and adding more elements than the array size at run time would cause Stack Overflow.

10. Which of the following array element will return the top-of-the-stack-element for a stack of size N elements(capacity of stack > N)?
a) S[N-1]
b) S[N]
c) S[N-2]
d) S[N+1]
View Answer
Answer: a
Explanation: Array indexing start from 0, hence N-1 is the last index.

“Stack using Linked List”.

1. What is the best case time complexity of deleting a node in a Singly Linked list?
a) O (n)
b) O (n^2)
c) O (nlogn)
d) O (1)
View Answer
Answer: d
Explanation: Deletion of the head node in the linked list is taken as the best case. The

successor of the head node is changed to head and deletes the predecessor of the newly assigned head node. This process completes in O(1) time.

2. Which of the following statements are not correct with respect to Singly Linked List(SLL) and Doubly Linked List(DLL)?
a) Complexity of Insertion and Deletion at known position is O(n) in SLL and O(1) in DLL
b) SLL uses lesser memory per node than DLL
c) DLL has more searching power than SLL
d) Number of node fields in SLL is more than DLL
View Answer
Answer: d
Explanation: To insert and delete at known positions requires complete traversal of the list in worst case in SLL, SLL consists of an item and a node field, while DLL has an item and two node fields, hence SLL occupies lesser memory, DLL can be traversed both ways(left and right), while SLL can traverse in only one direction, hence more searching power of DLL. Node fields in SLL is 2 (data and address of next node) whereas in DLL is 3(data, address to next node, address to previous node).

3. Given below is the Node class to perform basic list operations and a Stack class with a no arg constructor.
Select from the options the appropriate pop() operation that can be included in the Stack class. Also 'first' is the top-of-the-stack.

```
class Node
{
          protected Node next;
          protected Object ele;
          Node()
          {
                    this(null,null);
          }
          Node(Object e,Node n)
          {
                    ele=e;
                    next=n;
          }
          public void setNext(Node n)
          {
                    next=n;
          }
          public void setEle(Object e)
```

```
        {
                ele=e;
        }
        public Node getNext()
        {
                return next;
        }
        public Object getEle()
        {
                return ele;
        }
}

class Stack
{
        Node first;
        int size=0;
        Stack()
        {
                first=null;
        }
}
```

a)

Take Data Structure I Mock Tests - Chapterwise!
Start the Test Now: Chapter 1, 2, 3, 4, 5, 6, 7, 8, 9, 10

```
public Object pop()
{
        if(size == 0)
        System.out.println("underflow");
        else
        {
                Object o = first.getEle();
                first = first.getNext();
                size--;
                return o;
        }
}
```

b)

```
public Object pop()
```

```
{
        if(size == 0)
        System.out.println("underflow");
        else
        {
                Object o = first.getEle();
                first = first.getNext().getNext();
                size--;
                return o;
        }
}
```

c)

```
public Object pop()
{
        if(size == 0)
        System.out.println("underflow");
        else
        {
                first = first.getNext();
                Object o = first.getEle();
                size--;
                return o;
        }
}
```

d)

```
public Object pop()
{
        if(size == 0)
        System.out.println("underflow");
        else
        {
                first = first.getNext().getNext();
                Object o = first.getEle();
                size--;
                return o;
        }
}
```

View Answer

Answer: a
Explanation: pop() should return the Object pointed to by the node ‘first’. The sequence of operations is, first, get the element stored at node ‘first’ using getEle(), and second, make the node point to the next node using getNext().

4. What does the following function do?

```
public Object some_func()throws emptyStackException
{
        if(isEmpty())
                throw new emptyStackException("underflow");
        return first.getEle();
}
```

a) pop
b) delete the top-of-the-stack element
c) retrieve the top-of-the-stack element
d) push operation
View Answer
Answer: c
Explanation: This code is only retrieving the top element, note that it is not equivalent to pop operation as you are not setting the ‘next’ pointer point to the next node in sequence.

5. What is the functionality of the following piece of code?

```
public void display()
{
        if(size == 0)
                System.out.println("underflow");
        else
        {
                Node current = first;
                while(current != null)
                {
                        System.out.println(current.getEle());
                        current = current.getNext();
                }
        }
}
```

a) reverse the list
b) display the list
c) display the list excluding top-of-the-stack-element
d) reverse the list excluding top-of-the-stack-element
View Answer
Answer: b
Explanation: An alias of the node ‘first’ is created which traverses through the list and displays the elements.

6. What does ‘stack overflow’ refer to?
a) accessing item from an undefined stack
b) adding items to a full stack
c) removing items from an empty stack
d) index out of bounds exception
View Answer
Answer: b
Explanation: Adding items to a full stack is termed as stack underflow.

7. Given below is the Node class to perform basic list operations and a Stack class with a no arg constructor. Select from the options the appropriate push() operation that can be included in the Stack class. Also ‘first’ is the top-of-the-stack.

```
class Node
{
          protected Node next;
          protected Object ele;
          Node()
          {
                    this(null,null);
          }
          Node(Object e,Node n)
          {
                    ele=e;
                    next=n;
          }
          public void setNext(Node n)
          {
                    next=n;
          }
          public void setEle(Object e)
          {
                    ele=e;
```

```
        }
        public Node getNext()
        {
                return next;
        }
        public Object getEle()
        {
                return ele;
        }
}

class Stack
{
        Node first;
        int size=0;
        Stack()
        {
                first=null;
        }
}
```

a)

```
public void push(Object item)
{
        Node temp = new Node(item,first);
        first = temp;
        size++;
}
```

b)

```
public void push(Object item)
{
        Node temp = new Node(item,first);
        first = temp.getNext();
        size++;
}
```

c)

```
public void push(Object item)
{
```

```
        Node temp = new Node();
        first = temp.getNext();
        first.setItem(item);
        size++;
}
```

d)

```
public void push(Object item)
{
        Node temp = new Node();
        first = temp.getNext.getNext();
        first.setItem(item);
        size++;
}
```

View Answer

Answer: a

Explanation: To push an element into the stack, first create a new node with the next pointer point to the current top-of-the-stack node, then make this node as top-of-the-stack by assigning it to ‘first’.

8. Consider these functions:
push() : push an element into the stack
pop() : pop the top-of-the-stack element
top() : returns the item stored in top-of-the-stack-node
What will be the output after performing these sequence of operations

```
push(20);
push(4);
top();
pop();
pop();
pop();
push(5);
top();
```

a) 20
b) 4
c) stack underflow

d) 5
View Answer
Answer: d
Explanation: 20 and 4 which were pushed are popped by the two pop() statements, the recent push() is 5, hence top() returns 5.

9. Which of the following data structures can be used for parentheses matching?
a) n-ary tree
b) queue
c) priority queue
d) stack
View Answer
Answer: d
Explanation: For every opening brace, push it into the stack, and for every closing brace, pop it off the stack. Do not take action for any other character. In the end, if the stack is empty, then the input has balanced parentheses.

10. Minimum number of queues to implement stack is _____
a) 3
b) 4
c) 1
d) 2
View Answer
Answer: c
Explanation: Use one queue and one counter to count the number of elements in the queue.

“Stack using Queues”.

1. To implement a stack using queue(with only enqueue and dequeue operations), how many queues will you need?
a) 1
b) 2
c) 3
d) 4
View Answer
Answer: b
Explanation: Either the push or the pop has to be a costly operation, and the costlier operation requires two queues.

2. Making the push operation costly, select the code snippet which implements the same.(let q1 and q2 be two queues)
a)

```
public void push(int x)
{
        if(empty())
        {
            q1.offer(x);
        }
        else{
                if(q1.size()>0)
                {
                    q2.offer(x);
                    int size = q1.size();
                    while(size>0)
                    {
                        q2.offer(q1.poll());
                        size--;
                    }
                }
            else if(q2.size()>0)
            {
                q1.offer(x);
                int size = q2.size();
                while(size>0)
                {
                    q1.offer(q2.poll());
                    size--;
                }
            }
        }
    }
```

b)

```
public void push(int x)
{
        if(empty())
        {
            q1.offer(x);
        }
        else
```

```
        {
            if(q1.size()>0)
            {
                q1.offer(x);
                int size = q1.size();
                while(size>0)
                {
                    q2.offer(q1.poll());
                    size--;
                }
            }
            else if(q2.size()>0)
            {
                q2.offer(x);
                int size = q2.size();
                while(size>0)
                {
                    q1.offer(q2.poll());
                    size--;
                }
            }
        }
}
```

c)


```
public void push(int x)
{
        if(empty())
        {
            q1.offer(x);
        }
        else
        {
            if(q1.size()>0)
            {
                q2.offer(x);
                int size = q1.size();
                while(size>0)
```

```
                {
                    q1.offer(q2.poll());
                    size--;
                }
            }
            else if(q2.size()>0)
            {
                q1.offer(x);
                int size = q2.size();
                while(size>0)
                {
                    q2.offer(q1.poll());
                    size--;
                }
            }
        }
}
```

d)

Check this: Programming Books | Computer Science MCQs

```
public void push(int x)
{
        if(empty())
        {
            q1.offer(x);
        }
        else
        {
            if(q1.size()>0)
            {
                q2.offer(x);
                int size = q1.size();
                while(size>0)
                {
                    q2.offer(q2.poll());
                    size--;
                }
            }
            else if(q2.size()>0)
            {
                q1.offer(x);
```

```
                int size = q2.size();
                while(size>0)
                {
                    q2.offer(q1.poll());
                    size--;
                }
            }
        }
}
```

View Answer

Answer: a

Explanation: Stack follows LIFO principle, hence a new item added must be the first one to exit, but queue follows FIFO principle, so when a new item is entered into the queue, it will be at the rear end of the queue. If the queue is initially empty, then just add the new element, otherwise add the new element to the second queue and dequeue all the elements from the second queue and enqueue it to the first one, in this way, the new element added will be always in front of the queue. Since two queues are needed to realize this push operation, it is considered to be costlier.

3. Making the push operation costly, select the code snippet which implements the pop operation.

a)

```
public void pop()
{
        if(q1.size()>0)
        {
            q2.poll();
        }
        else if(q2.size()>0)
        {
            q1.poll();
        }
}
```

b)

```
public void pop()
{
        if(q1.size()>0)
        {
```

```
            q1.poll();
        }
        else if(q2.size()>0)
        {
            q2.poll();
        }
}
```

c)

```
public void pop()
{
        q1.poll();
         q2.poll();
}
```

d)

```
public void pop()
{
        if(q2.size()>0)
        {
            q1.poll();
        }
        else
        {
            q2.poll();
        }
}
```

View Answer

Answer: b

Explanation: As the push operation is costly, it is evident that the required item is in the front of the queue, so just dequeue the element from the queue.

4. Select the code snippet which returns the top of the stack.

a)

```
public int top()
{
       if(q1.size()>0)
       {
            return q1.poll();
```

```
        }
        else if(q2.size()>0)
        {
             return q2.poll();
        }
        return 0;
}
```

b)

```
public int top()
{
        if(q1.size()==0)
        {
             return q1.peek();
        }
        else if(q2.size()==0)
        {
             return q2.peek();
        }
         return 0;
    }
```

c)

```
public int top()
{
        if(q1.size()>0)
        {
             return q1.peek();
        }
        else if(q2.size()>0)
        {
             return q2.peek();
        }
        return 0;
}
```

d)

```
public int top()
{
        if(q1.size()>0)
```

```
        {
            return q2.peek();
        }
        else if(q2.size()>0)
        {
            return q1.peek();
        }
        return 0;
}
```

View Answer

Answer: c

Explanation: Assuming its a push costly implementation, the top of the stack will be in the front end of the queue, note that peek() just returns the front element, while poll() removes the front element from the queue.

5. Select the code snippet which return true if the stack is empty, false otherwise.

a)

```
public boolean empty()
{
    return q2.isEmpty();
}
```

b)

```
public boolean empty()
{
    return q1.isEmpty() || q2.isEmpty();
}
```

c)

```
public boolean empty()
{
    return q1.isEmpty();
}
```

d)

```
public boolean empty()
{
    return q1.isEmpty() & q2.isEmpty();
```

```
}
```

View Answer

Answer: b

Explanation: If both the queues are empty, then the stack also is empty.

6. Making the pop operation costly, select the code snippet which implements the same.

a)

```
public int pop()
{
          int res=-999,count=0;
          if(q1.size()>0)
         {
                    count = q1.size();
                    while(count>0)
                              q2.offer(q1.poll());
                    res = q1.poll();
          }
          if(q2.size()>0)
         {
                    count = q2.size();
                    while(count>0)
                              q1.offer(q2.poll());
                    res = q2.poll();
          }
          return res;
}
```

b)

```
public int pop()
{
          int res=-999,count=0;
          if(q1.size()>0)
         {
                    count = q1.size();
                    while(count>1)
                              q2.offer(q1.poll());
                    res = q2.poll();
          }
```

```
         if(q2.size()>0)
        {
                  count = q2.size();
                  while(count>1)
                           q1.offer(q2.poll());
                  res = q1.poll();
         }
         return res;
}
```

c)

```
public int pop()
{
         int res=-999,count=0;
         if(q1.size()>0)
        {
                  count = q1.size();
                  while(count>1)
                           q2.offer(q1.poll());
                  res = q1.poll();
         }
         if(q2.size()>0)
        {
                  count = q2.size();
                  while(count>1)
                           q1.offer(q2.poll());
                  res = q2.poll();
         }
         return res;
}
```

d)

```
public int pop()
{
         int res=-999,count=0;
         if(q1.size()>0)
        {
                  count = q2.size();
                  while(count>1)
                           q2.offer(q1.poll());
                  res = q1.poll();
```

```
        }
        if(q2.size()>0)
       {
                count = q1.size();
                while(count>1)
                        q1.offer(q2.poll());
                res = q2.poll();
        }
        return res;
}
```

View Answer

Answer: c

Explanation: Here the pop operation is costly, hence we need two queues, other than the first element, all the the elements from one queue are dequeued and enqueued to the second queue, hence only one element remains in the first queue which is the item we want, so dequeue it and return the result.

7. What is the functionality of the following piece of code?

```
public void fun(int x)
{
        q1.offer(x);
}
```

a) Perform push() with push as the costlier operation
b) Perform push() with pop as the costlier operation
c) Perform pop() with push as the costlier operation
d) Perform pop() with pop as the costlier operation

View Answer

Answer: b

Explanation: offer() suggests that it is a push operation, but we see that it is performed with only one queue, hence the pop operation is costlier.