# 10 MCQ of Data Structure Paper-2(CS/IT) For NIC|SSC Scientific Assistant|IBPS IT Officer Exam 2017

**1)The time complexity for evaluating a postfix expression is**
a)O(n^2)
b)O(logn)
c)O(n)
d)O(nlogn)

**2)We use malloc and calloc for**
a)static memory allocation
b)dynamic memory allocation
c)both dynamic and static memory allocation
d)None of these

**3)What is the function of "Free(P)"**
a)Address that P is pointing to is unchanged but the data that reside at that address are now undefined
b)Delete P for further use
c)Insert element to 1
d)None of these

**4)A list can be initialized to the empty list by which operation**
a)list=1
b)list=NULL
c)list=0
d)None of these

**5)What can be said about the array representation of a circular queue when it contains only one element?**
a)front=rear=NULL
b)front=rear not equal to NULL

c)front=rear+1
d)front=rear-1

**6)r=malloc(sizeof(struct node))**
**In this expression what should be written before malloc for appropriate type casting**
a)(int*)
b)(char*)
c)(node*)
d)(struct node*)

**7)Stack is useful for implementing**
a)radix sort
b)breadth first search
c)recursion
d)none of these

**8)……..is called self referential structure**
a)stack
b)queue
c)linked list
d)graph

**9)The postfix equivalent of the prefix *+ab-cd is**
a)abcd+-*
b)ab+cd-*
c)ab+cd*-
d)ab+-cd*

**10)Among the following which is not C the primitive data types**
a)int
b)float
c)char
d)structure

**Answers:**
1)c

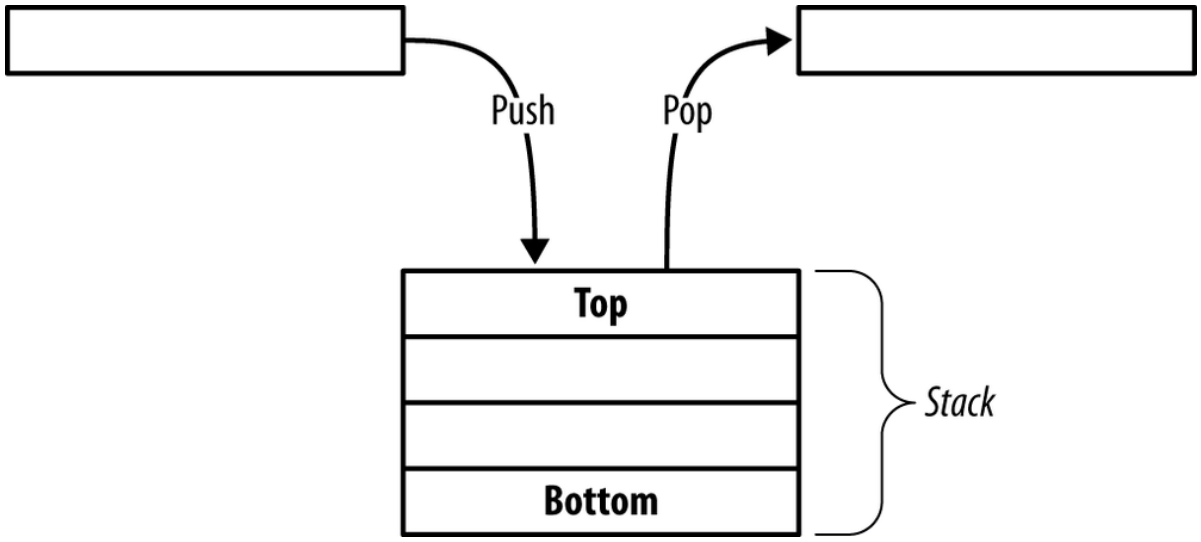
2)b
3)a
4)b
5)b
6)d
7)c
8)c
9)b
10)d

---

# Stack and Queue

**Stack:**

- Abstract Data Type
- A stack is a container of objects that are inserted and removed according to the last-in first-out (LIFO) principle. In the pushdown stacks only two operations are allowed: **push** the item into the stack, and **pop** the item out of the stack. A stack is a limited access data structure – elements can be added and removed from the stack only at the top. **push** adds an item to the top of the stack, **pop** removes the item from the top.



- One of the most interesting applications of stacks can be found in solving a puzzle called **Tower of Hanoi.** According to an old Brahmin story, the existence of the universe is calculated in terms of the time taken by a number of monks, who are working all the time, to move 64 disks from one pole to another. But there are some rules about how this should be done, which are:
    1. You can move only one disk at a time.
    2. For temporary storage, a third pole may be used.
    3. You cannot place a disk of larger diameter on a disk of smaller diameter.
- To use a stack efficiently, we need to check the status of stack as well. For the same purpose, the following functionality is added to stacks –
    - **peek()** – get the top data element of the stack, without removing it.
    - **isFull()** – check if stack is full.
    - **isEmpty()** – check if stack is empty.

  At all times, we maintain a pointer to the last PUSHed data on the stack. As this pointer always represents the top of the stack, hence named **top**. The **top** pointer provides top value of the stack without actually removing it.

## Run-time complexity of stack operations:

For all the standard stack operations (push, pop, isEmpty, size), the worst-case run-time complexity can be O(1). We say *can* and not *is* because it is always possible to implement stacks with an underlying representation that is inefficient. However, with the representations we have looked at (static array and a reasonable linked list) these operations take constant time. It's obvious that size and isEmpty constant-time operations. push and pop are also O(1) because they only work with one end of the data structure – the top of the

stack. The upshot of all this is that stacks can and should be implemented easily and efficiently.The copy constructor and assignment operator are O(n), where n is the number of items on the stack. This is clear because each item has to be copied (and copying one item takes constant time). The destructor takes linear time (O(n)) when linked lists are used – the underlying list has to be traversed and each item released (releasing the memory of each item is constant in terms of the number of items on the whole list).

**2)Queue:**

- Abstract data type
- First element is inserted from one end called **REAR**(also called tail), and the deletion of existing element takes place from the other end called as **FRONT**(also called head). This makes queue as FIFO data structure, which means that element inserted first will also be removed first.

**The following are operations performed by queue in data structures:**

- Enqueue (Add operation)
- Dequeue (Remove operation)
- Initialize

**Enqueue**

This operation is used to add an item to the queue at the rear end. So, the head of the queue will be now occupied with an item currently added in the queue. Head count will be incremented by one after addition of each item until the queue reaches the tail point. This operation will be performed at the rear end of the queue.

**Dequeue**

This operation is used to remove an item from the queue at the front end. Now the tail count will be decremented by one each time when an item is removed from the queue until the queue reaches the head point. This operation will be performed at the front end of the queue.

**Initialize**

This operation is used to initialize the queue by representing the head and tail positions in the memory allocation table (MAT).

**Few more functions are required to make the above-mentioned queue operation efficient. These are –**

- **peek()** – Gets the element at the front of the queue without removing it.
- **isfull()** – Checks if the queue is full.
- **isempty()** – Checks if the queue is empty.

In queue, we always dequeue (or access) data, pointed by **front** pointer and while enqueing (or storing) data in the queue we take help of **rear** pointer.

# Run time Complexity of queue Operations:

- **Insert: O(1)**
- **Remove: O(1)**
- **Size: O(1)**

**Circular Queue:** In a standard queue data structure re-buffering problem occurs for each dequeue operation. To solve this problem by joining the front and rear ends of a queue to make the queue as a circular queue.Circular queue is a linear data structure. It follows FIFO principle.

- In circular queue the last node is connected back to the first node to make a circle.
- Circular linked list follow the First In First Out principle
- Elements are added at the rear end and the elements are deleted at front end of the queue
- Both the front and the rear pointers points to the beginning of the array.
- It is also called as “Ring buffer”.
- Items can inserted and deleted from a queue in O(1) time.

**Circular Queue can be created in three ways they are**

1)Using single linked list
2)Using double linked list
3)Using arrays