Question Answer

Q.1 Define stack?

Ans. A **stack** is a non-primitive linear data structure. It is an ordered list in which addition of new data item and deletion of already existing data item is done from only one end, known as Top of Stack (TOS). As all the deletion and insertion in a stack is done from top of the stack, the last added element will be the first to be removed from the stack. Due to this reason, the stack is also called **Last-In-First-Out (LIFO)** type of list. Consider some examples,

1. A common model of a stack is plates in a marriage party. Fresh plates are "pushed" onto the top and "popped" off the top.

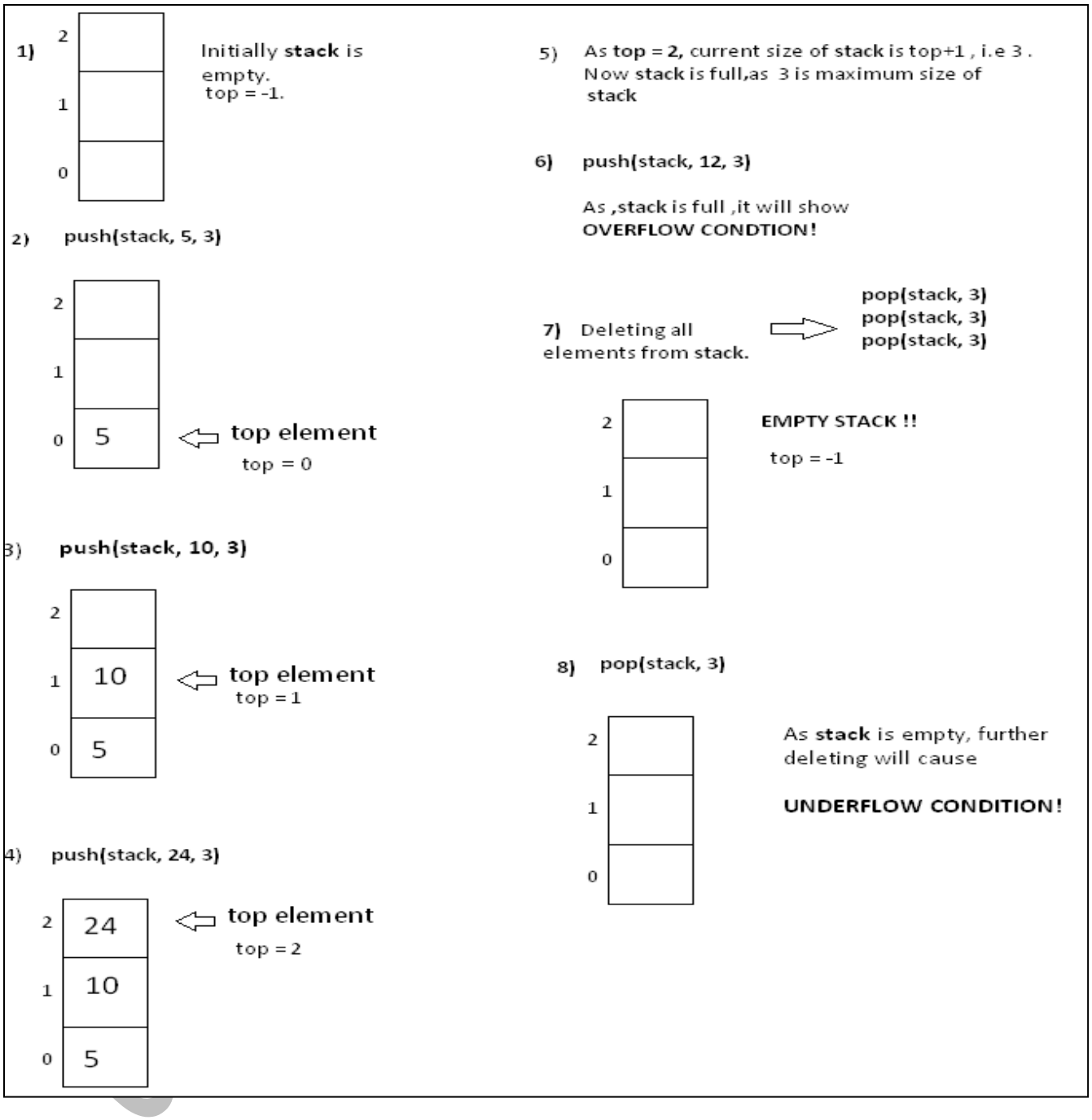
2. Some of you may eat biscuits. If you assume only one side of the cover is torn and biscuits are taken off one by one. This is called popping and similarly, if you want to preserve some biscuits for some time later, you will put them back into the pack through the same torn end called pushing.

Q2. Give the basic operation of stack.

Ans. The basic **operations** that can be performed on **stack** are as follows :

- **PUSH :** The process of adding a new element to the top of the stack is called PUSH operation. Pushing an element in the stack involve adding of element, as the new element will be inserted at the top, so after every push operation, the top is incremented by one. In case the array is full and no new element can be accommodated, it is called STACK-FULL condition. This condition is called STACK OVERFLOW.

- **POP :** The process of deleting an element from the top of the stack is called POP operation. After every pop operation, the stack is decremented by one. If there is no element on the stack and the pop is performed then this will result into STACK UNDERFLOW condition.



Q.3 Write down the algorithm for inserting an item into the stack (PUSH).

Ans. Let STACK[MAXSIZE] is an array for implementing the stack, MAXSIZE represents the max. size of array STACK. NUM is the element to be pushed in stack & TOP is the index number of the element at the top of stack.

Step 1 : **[Check for stack overflow ?]**
If TOP = MAXSIZE – 1, then :
Write : 'Stack Overflow' and return.
[End of If Structure]

Step 2 : Read NUM to be pushed in stack.

Step 3 : Set TOP = TOP + 1 **[Increases TOP by 1]**
Step 4 : Set STACK[TOP] = NUM [Inserts new number NUM in new TOP Position]
Step 5 : Exit

Q.4 Write down the C function for push operation in stack.

Ans. C function for push operation in stack :

```
void push()
{
if(top==MAXSIZE-1)
{
printf("\n\nStack is full(Stack overflow)");
return;
}
int num;
printf("\n\nEnter the element to be pushed in stack : ");
scanf("%d",&num);
top++;
stack[top]=num;
}
```

Q.5 Write down the algorithm for deleting an item from the stack (POP)

Ans. Let STACK[MAXSIZE] is an array for implementing the stack where MAXSIZE represents the max. size of array STACK. NUM is the element to be popped from stack & TOP is the index number of the element at the top of stack.

Step 1 **: [Check for stack underflow ?]**
If TOP = -1 : then
Write : 'Stack underflow' and return.

[End of If Structure]
Step 2 : Set NUM = STACK[TOP] **[Assign Top element to NUM]**
Step 3 : Write 'Element popped from stack is : ',NUM.
Step 4 : Set TOP = TOP - 1 **[Decreases TOP by 1]**

Step 5 : Exit

Q.6 Write the function of the Stack POP operation in C.

Ans. The function of the Stack POP operation in C is as follows :

```
void pop()
{
 if(top== -1)
 {
  printf("\n\nStack is empty(Stack underflow)");
  return;
 }
  int num;
  num=stack[top];
  printf("\n\nElement popped from stack : %d",num);
  top--;
}
```

Q.7 Write down the algorithms for push & pop for dynamic implementation using pointers.

1.Algorithm for inserting an item into the stack (PUSH)
2.Algorithm for deleting an item from the stack (POP)

Ans. Algorithm for inserting an item into the stack (PUSH)

Let **PTR** is the structure pointer which allocates memory for the new node & NUM is the element to be pushed into stack, TOP represents the address of node at the top of the stack, INFO represents the information part of the node and LINK represents the link or next pointer pointing to the address of next node.

Step 1 : Allocate memory for the new node using PTR.

Step 2 : Read NUM to be pushed into stack.

Step 3 : Set PTR->INFO = NUM

Step 4 : Set PTR->LINK=TOP

Step 5 : Set TOP = PTR

Step 6 : Exit

(B) Algorithm for deleting an item from the stack (POP)

Let **PTR** is the structure pointer which deallocates memory of the node at the top of stack & NUM is the element to be popped from stack, TOP represents the address of node at the top of the stack, INFO represents the information part of the node and LINK represents the link or next pointer pointing to the address of next node.

Step 1 : **[Check for Stack Underflow ?]**
If TOP = NULL : then
Write 'Stack Underflow' &
Return.
[End of If Structure]
Step 2 : Set PTR=TOP.
Step 3 : Set NUM=PTR->INFO

Step 4 : Write 'Element popped from stack is : ',NUM

Step 5 : Set TOP=TOP->NEXT

Step 6 : Deallocate memory of the node at the top using PTR.

Step 5 : Exit

Q. 8 Write down the function for push and pop operation in C for dynamic representation of stack.

Ans. Function for PUSH Operation using pointers:

```
void push()
{
struct stack *ptr;
int num;
ptr=(struct stack *)malloc(sizeof(struct stack));
printf("\nEnter the element to be pushed in stack : ");
scanf("%d",&num);
ptr->info=num;

ptr->link=top;
top=ptr;
}
```

Function for POP Operation using pointers

```
void pop()
{
if(top==NULL)
{
 printf("\nStack is empty(Stack underflow).");
 return;
}
struct stack *ptr;
int num;
ptr=top;
num=ptr->info;
printf("\nElement popped from stack : %d",num);
top=top->link
free(ptr);
}
```

Q.9 What is polish notion ?

Ans. A polish mathematician suggested a notation called Polish notation, which gives two alternatives to represent an arithmetic expression.

- The notations are prefix and postfix notations.
- The fundamental property of Polish notation is that the order in which the operations are to be performed is completely determined by the positions of the operators and operands in the expression.
- Hence parenthesis are not required while writing expressions in Polish notation. The Two types of polish notations are given below :

1. **Prefix notation**
2. **Postfix notation**

Q.10 Define Prefix notation?

Ans. The prefix notation is a notation in which the operator is written before the operands. For example, + AB

As the operator '+' is written before the operands A and B, this notation is called prefix notation (pre means before).

Q.11 Define Postfix notation?

Ans. The postfix notation is a notation in which the operator is written after the operands. For example,

AB +

As the operator '+' is written after the operands A and B, this notation is called postfix notation (post means after).

Q.12 Define Infix notation?

Ans. The **infix** notation is what we come across in our general mathematics, where the operator is written in-between the operands. For example : The expression to add two numbers A and B is written in infix notation as :

A + B

Note that the operator '+' is written in-between het operands A and B. The reason why this notation is called **infix**, is the place of operator in the expression.

Q.13 Write down the algorithm for converting infix expression into postfix expression.

Ans. Algorithm for converting infix expression into postfix expression

Let Q is an arithmetic expression written in infix notation. This algorithm finds the equivalent postfix expression P.

1. Push "(" onto STACK, and add")" to the end of Q.
2. Scan Q from left to right and repeat **Step**s 3 to 6 for each element of Q until the stack is empty.
3. If an operand is encountered, add it to P.
4. If a left parenthesis is encountered, push it onto STACK.
5. If an operator Ä is encountered, then :
   (a) Add Ä to STACK. [End of If structure].
   (b) Repeatedly pop from STACK and add P each operator (on the top of STACK) which has the same precedence as or higher precedence than Ä.
6. If a right parenthesis is encountered, then :
   (a) Repeatedly pop from STACK and add to P each operator (on the top of STACK until a left parenthesis is encountered.
   (b) Remove the left parenthesis. [Do not add the left parenthesis to P.]
   [End of if structure.]
   [End of **Step** 2 loop].
7. Exit.

Q. 14 Write down the steps for the evaluation of postfix expression.

Ans. To evaluate a postfix expression using Stack data structure we can use the following steps...

1. Read all the symbols one by one from left to right in the given Postfix Expression
2. If the reading symbol is operand, then push it on to the Stack.
3. If the reading symbol is operator (+ , - , * , / etc.,), then perform TWO pop operations and store the two popped oparands in two different variables (operand1 and operand2). Then perform reading symbol operation using operand1 and operand2 and push result back on to the Stack.
4. Perform a pop operation and display the popped value as final result.

Q.15 Solve the postfix notion : 2 3 4 + * 6 –

Ans

Input token	Operation	Stack contents (top on the right)	Details
2	Push on the stack	2	
3	Push on the stack	2, 3	
4	Push on the stack	2, 3, 4	
+	Add	2, 7	Pop two values: 3 and 4 and push the result 7 on the stack
*	Multiply	14	Pop two values: 2 and 7 and push the result 14 on the stack
6	Push on the stack	14, 6	
–	Subtract	8	Pop two values: 14 and 6 and push the result 8 on the stack
(End of tokens)	(Return the result)	8	Pop the only value 8 and return it

Q.16 Explain the term recursion?

Ans. The process in which a function calls itself directly or indirectly is called recursion and the corresponding function is called as recursive function. Using recursive algorithm, certain problems can be solved quite easily.

Examples of such problems are Towers of Hanoi (TOH), Inorder/Preorder/Postorder Tree Traversals, DFS of Graph, etc.

Q.17 Explain Tower of Hanoi problem?

Ans. Tower of Hanoi, is a mathematical puzzle which consists of three towers (pegs) and more than one rings is as depicted −

These rings are of different sizes and stacked upon in an ascending order, i.e. the smaller one sits over the larger one. There are other variations of the puzzle where the number of disks increase, but the tower count remains the same.

A few rules to be followed for Tower of Hanoi are −

- Only one disk can be moved among the towers at any given time.
- Only the "top" disk can be removed.
- No large disk can sit over a small disk.

Tower of Hanoi puzzle with n disks can be solved in minimum $\mathbf{2^n-1}$ steps.

The steps to follow are −

Step 1 − Move n-1 disks from **source** to **aux**
Step 2 − Move n^{th} disk from **source** to **dest**
Step 3 − Move n-1 disks from **aux** to **dest**

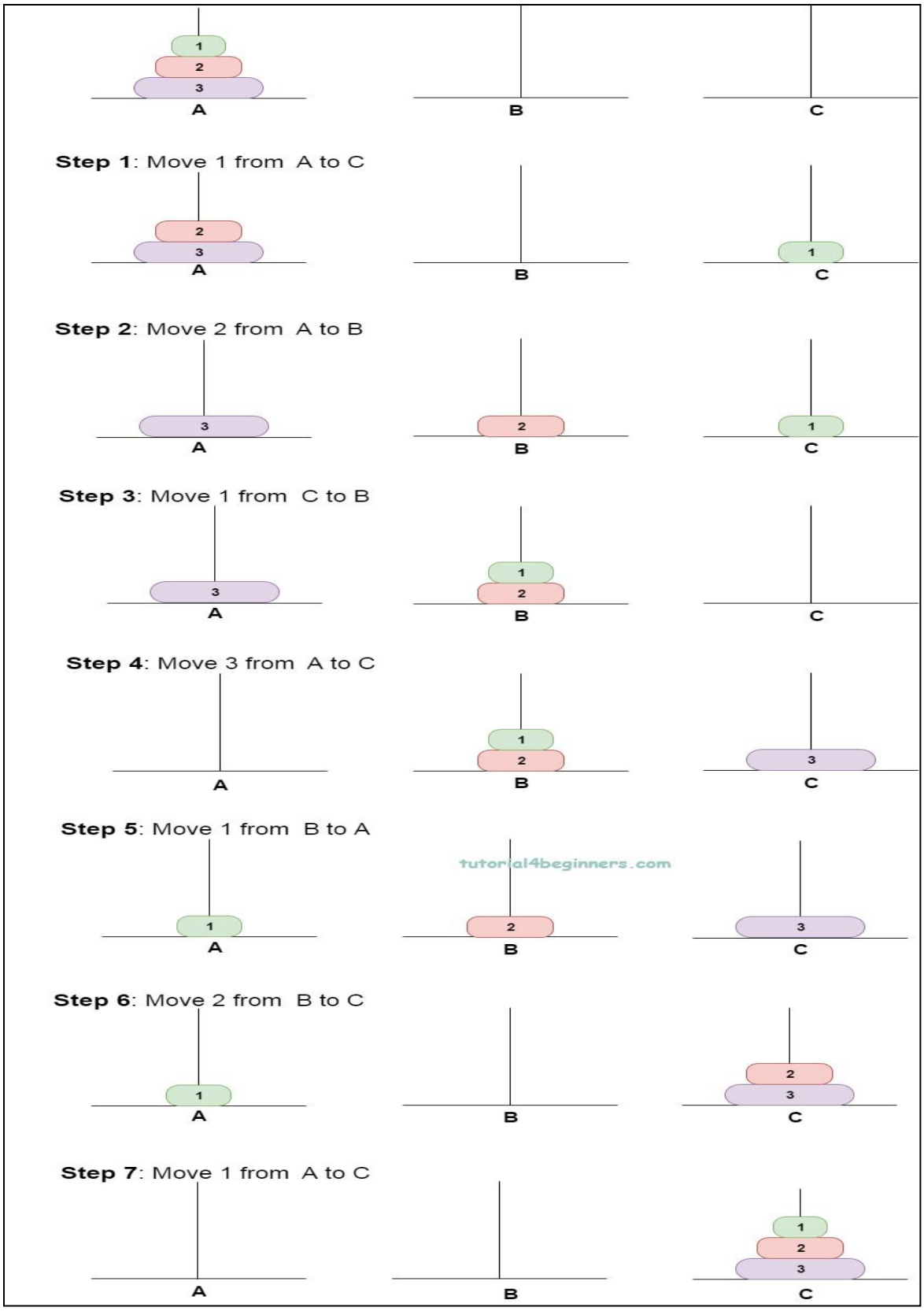
A recursive algorithm for Tower of Hanoi can be driven as follows −

```
START
Procedure Hanoi(disk, source, dest, aux)

  IF disk == 1, THEN
    move disk from source to dest
  ELSE
    Hanoi(disk - 1, source, aux, dest)    // Step 1
    move disk from source to dest         // Step 2
```

```
    Hanoi(disk - 1, aux, dest, source)     // Step 3
  END IF

END Procedure
STOP
```


1
2
3
A
B
C
Step 1: Move 1 from A to C
2
3
A
B
1
C
Step 2: Move 2 from A to B
3
A
2
B
1
C
Step 3: Move 1 from C to B
3
A
1
2
B
C
Step 4: Move 3 from A to C
A
1
2
B
3
C
Step 5: Move 1 from B to A
1
A
tutorial4beginners.com
2
B
3
C
Step 6: Move 2 from B to C
1
A
B
2
3
C
Step 7: Move 1 from A to C
A
B
1
2
3
C

Q.18 Give one example of recursion.

Ans. int factorial(int n)

```
{
   if (n == 0)
     return 1;
   return n*factorial(n-1);
}
```

Q. 19 Define Queue?

Ans. **Queue** is a non-primitive linear data structure that permits insertion of an element at one end and deletion of an element at the other end.

- The end at which the deletion of an element take place is called **front**, and the end at which insertion of a new element can take place is called **rear**.
- The deletion or insertion of elements can take place only at the front and rear end of the list respectively.
- The first element that gets added into the queue is the first one to get removed from the list. Hence, Queue is also referred to as **First-In-First-Out (FIFO)** list.

Q.21 Explain queue with the help of example.

Ans.

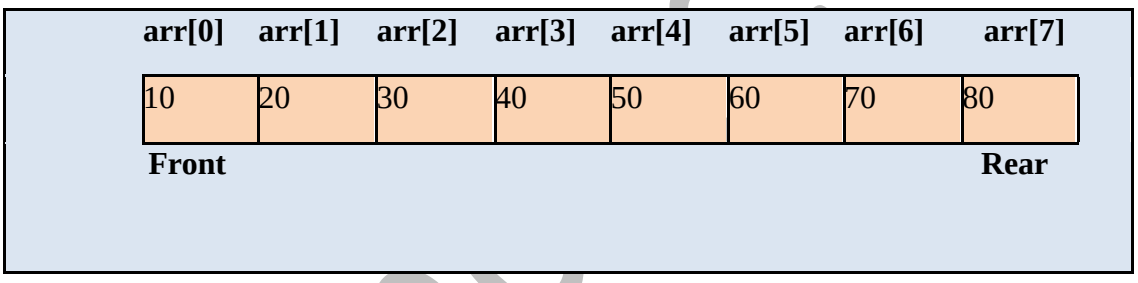
Q.22 State the implementation of the queue.

Ans. Queues can be implemented in two ways :

1. Static implementation (using arrays)
2. Dynamic implementation (using pointers)

Q.23 Explain the static representation of Queue?

Ans. Static implementation of Queue is represented by arrays. If Queue is implemented using arrays, we must be sure about the exact number of elements we want to store in the queue, because we have to declare the size of the array at design time or before the processing starts. In this case, the beginning of the array will become the front for the queue and the last location of the array will act as rear for the queue. Fig. (3) shows the representation of a queue as an array.



The following relation gives the total number of elements present in the queue, when implemented using arrays :

REAR – FRONT + 1

Also note that if front > rear, then there will be no element in the queue or queue is empty.

Q.24 What operations can be perform on queue?

Ans. OPERATIONS ON A QUEUE

The basic operations that can be performed on queue are :

1. To **Insert** an element in a Queue
2. To **Delete** an element from a Queue.
3. To **Traverse** all elements of a Queue.

Q.25 Write down the algorithm for Insertion in a linear queue.

Ans. Algorithm for Insertion in a Linear Queue

Let QUEUE[MAXSIZE] is an array for implementing the Linear Queue & NUM is the element to be inserted in linear queue, FRONT represents the index number of the element at the beginning of the queue and REAR represents the index number of the element at the end of the Queue.

Step 1 :If REAR = (MAXSIZE –1) : then
Write : "Queue Overflow" and return
[End of If structure]
Step 2 : Read NUM to be inserted in Linear Queue.
Step 3 : Set REAR := REAR + 1
Step 4 : Set QUEUE[REAR] := NUM
Step 5 : If FRONT = –1 : then
Set FRONT=0.
[End of If structure]
Step 6 : Exit

Q.26 Write down the algorithm for deletion in a linear queue.

Ans. Algorithm for Deletion from a Linear Queue

Let QUEUE[MAXSIZE] is an array for implementing the Linear Queue & NUM is the element to be deleted from linear queue, FRONT represents the index number of the element at the beginning of the queue and REAR represents the index number of the element at the end of the Queue.

Step 1 : If FRONT = -1 : then
Write : "Queue Underflow" and return
[End of If structure]
Step 2 : Set NUM := QUEUE[FRONT]
Step 3 : Write "Deleted item is : ", NUM
Step 4 : Set FRONT := FRONT + 1.
Step 5 : If FRONT>REAR : then
Set FRONT := REAR := -1.
[End of If structure]
Step 6 : Exit

Q.27 Write down the C function for insertion and deletion in a linear queue using array.

Ans. Function for insertion in a linear queue (using arrays) :

```
void lqinsert()
{
int num;
 if(rear==MAXSIZE-1)
 {
   printf("\nQueue is full (Queue overflow)");
   return;
 }
 printf("\nEnter the element to be inserted : ");
 scanf("%d",&num);
 rear++;
 queue[rear]=num;
 if(front==-1)
  front=0;
}
```

Function(Procedure) for Deletion from a Linear Queue :

```
void lqdelete()
{
 if(front == -1)
 {
  printf("\nQueue is empty (Queue underflow)");
  return;
 }
 int num;
 num=queue[front];
 printf("\nDeleted element is : %d",num);
 front++;
 if(front>rear)
  front=rear=-1;
}
```

Q.28 Write down the algorithms for insertion & deletion in a linear queue for dynamic implementation using linked list.

Ans. Algorithms for insertion & deletion in a linear queue for dynamic implementation using linked list :

1) Algorithm for inserting an element in a Linear Queue :

Let PTR is the structure pointer which allocates memory for the new node & NUM is the element to be inserted into linear queue, INFO represents the information part of the node and LINK represents the link or next pointer pointing to the address of next node. FRONT

represents the address of first node, REAR represents the address of the last node. Initially, Before inserting first element in the queue, FRONT=REAR=NULL.

Step 1 : Allocate memory for the new node using PTR.
Step 2 : Read NUM to be inserted into linear queue.
Step 3 : Set PTR->INFO = NUM
Step 4 : Set PTR->LINK= NULL
Step 5 : If FRONT = NULL : then
Set FRONT=REAR=PTR
Else
Set REAR->LINK=PTR;
Set REAR=PTR;
[End of If Else Structure]
Step 6 : Exit

2) Algorithm for Deleting a node from a Linear Queue :

Let PTR is the structure pointer which deallocates memory of the first node in the linear queue & NUM is the element to be deleted from queue, INFO represents the information part of the deleted node and LINK represents the link or next pointer of the deleted node pointing to the address of next node. FRONT represents the address of first node, REAR represents the address of the last node.

Step 1 : If FRONT = NULL : then

Write 'Queue is Empty(Queue Underflow)' and return.
[End of If structure]
Step 2 : Set PTR = FRONT
Step 3 : Set NUM = PTR->INFO
Step 4 : Write 'Deleted element from linear queue si : ',NUM.
Step 5 : Set FRONT = FRONT->LINK
Step 6 : If FRONT = NULL : then
Set REAR = NULL.
[End of If Structure].
Step 7 : Deallocate memory of the node at the beginning of queue using PTR.
Step 8 : Exit

Q.29 Define priority Queue?

Ans. A **priority queue** is a collection of elements where the elements are stored according to their priority levels. The order in which the elements get added or removed is decided by the priority of the element.

Following **rules** are applied to maintain a priority queue :

(1) The element with a higher priority is processed before any element of lower priority.

(2) If there are elements with the same priority, then the element added first in the queue would get processed.

Priority queues are used for implementing job scheduling by the operating system where jobs with higher priorities are to be processed first. Another application of Priority queues is simulation systems where priority corresponds to event times.

There are mainly two ways of maintaining a priority queue in memory. One uses a one-way list, and the other uses multiple queues. The ease or difficultly in adding elements to or deleting them from a priority queue clearly depends on the representation that one chooses

Q.30 Give two application of queue?

Ans. Applications of queue are:

1. Round Robin technique for processor scheduling is implemented using queues.
2. All types of customer service (like railway ticket reservation) center software's are designed using queues to store customers information.