DATA STRUCTURES

CHAPTER 4

BY,

AMOGH

One mark questions:

1. What is data structure?

A: A data structure is a specialized format for organizing and storing data.

2. What is primitive data structure?

A: Data structures that are directly operated upon by machine-level instructions are known as primitive data structures.

3. Give an example for primitive data structure.

A: integer, real (float), logical data, character data, pointer and reference are primitive data structures.

4. What is non-primitive data structure?

A: Non-primitive data structures are more complex data structures. These data structures are derived from the primitive data structures. They stress on formation of groups of homogeneous and heterogeneous data elements.

5. Give an example for non-primitive data structure.

A: arrays, lists and files.

6. What is linear data structure?

A: Linear data structures are a kind of data structure that has homogenous elements. Each element is referred to by an **index**. The linear data structures are **Stack, Queues and Linked Lists.**

7. Give an example for linear data structure.

A: Stacks, Queues, Linked Lists**.**

8. Define traversing an array.

A: Accessing each element of the array exactly once to do some operation.

9. Define searching.

A: The process of finding the location of a data item in the given collection of data items is called as searching.

10. Define sorting.

A: The process of arrangement of data items in ascending or descending order is called sorting**.**

11. Define inserting.

A: The process of adding a new data item into the given collection of data items is called insertion**.**

12. Define deleting.

A: The process of removing an existing data item from the given collection of data items is called deletion.

13. What is an array?

A: An array is a collection of homogeneous elements with unique name and the elements are arranged one after another in adjacent memory location.

14. What is a stack?

A: A stack is an ordered collection of items where the addition of new items and the removal of existing items always take place at the **same end**. This end is commonly referred to as the “**top**”. The end opposite to top is known as the **base**.

15. What is push in stack?

A: push(item) adds a new item to the top of the stack. It needs the item **and returns nothing**.

16. What is pop in stack?

A: pop() removes the top item from the stack. It needs no parameters and **returns the item**. The stack is modified.

17. Which order stack follows?

A: LIFO(Last In First Out)

18. Give an example for static memory representation.

A: In the static memory allocation, the amount of memory to be allocated is predicted and pre known. This memory is allocated during the compilation itself.

All the variables declared normally, are allocated memory statically.

Example: int a; //Allocates 2 bytes of memory space during the //compilation time**.**

19. Give an example for dynamic memory representation.

A: In the dynamic memory allocation, the amount of memory to be allocated is not known. This memory is allocated during **run-time** as and when required.

The following codes demonstrate how to allocate memory for different variables.

To allocate memory of type integer, int *iptr = new int;

int *pNumber;

pNumber = new int;

The first line declares the pointer, pNumber. The second line then allocates memory for an integer and then makes pNumber point to this new memory.

20. What is a queue?

A: A queue is an ordered collection of items where an item is inserted at one end called the “**rear**,” and an existing item is removed at the other end, called the “**front**.” Queues maintain a **FIFO** ordering property.

21. Which order does the queue data structure follow?

A: FIFO(First In First Out)

22. What is enqueue?

A: enqueue(item) adds a new item to **the rear of the queue.** It needs the item and **returns nothing**. This operation is generally called as **push**.

23. What is dequeue?

A: dequeue() removes the **front item from the queue**. It needs no parameters and returns the item. The queue is modified. This operation is generally called as **pop**.

24. What is linked list?

A: A linked list is a linear collection of data elements called **nodes** and the linear order is given by means of **pointers.**

Each node contains two parts fields: **the data and a reference to the next node.** The first part contains the **information** and the second part contains the **address of the next node** in the list. This is also called the **link field.**

26. Name the type of memory allocation use by the linked list.

A: Dynamic Memory Allocation**.**

27. What is non-linear data structure?

A: A non-linear data structure is a data structure in which a data item is connected to several other data items. The data item has the possibility to reach one or more data items. **The data items are not arranged in a sequential structure**.

28. Give an example for non-linear data structure.

A: Trees and Graphs.

29. What is a node?

IN TERMS OF LINKED LIST

A: A linked list is a linear collection of data elements called nodes and the linear order is given by means of pointers.(in case of linked list)

DEFINITION IN TERMS OF TREE:

A node is a structure which may contain a value, a condition, or represent a separate data structure (which could be a tree of its own).

30. What is parent node?

A: A node that has a child is called the parent node (or ancestor node, or superior). A node has at most one parent.

31. What is a child node?

A: Each node in a tree has **zero or more** child nodes, which are below it in the tree.

32. What is the height of a tree?

A: The height of a node is the length of the longest downward path to a **leaf** from that node. The height of the root is the height of the tree.

33. What is the depth of a tree?

A: The depth of a node is the length of the path to its root.

34. What is a root node?

A: Node at the **"top" of a tree** - the one from which all operations on the tree commence. The root node may not exist (a NULL tree with no nodes in it) or have 0, 1 or 2 children in a binary tree.

35. What is internal node?

A: An internal node or inner node is any node of a tree that **has child nodes** and is thus not a **leaf node**.

36. What is binary tree?

A: A binary tree is a tree in which each node has **at most two descendants** - a node can have just one but it can't have more than two.

37. What is a complete tree?

A: Tree in which each leaf is at the same distance from the root. i.e. all the nodes have **maximum two subtrees**.

38. What is a graph?

A: A graph is a collection of nodes called **vertices**, and the connections between them, called **edges**.

39. Name the data structure that is called LIFO list.

A: Stack.

40. Name the data structure that is called FIFO list.

A: Queue

41. Which operator is used to allocate the memory dynamically?

A: new Operator.

Two marks questions:

1. What are the two types of data structures?

A:

Primitive data structures:

Data structures that are directly operated upon by machine-level instructions are known as primitive data structures.

The integer, real (float), logical data, character data, pointer and reference are primitive data structures**.**

Non primitive data structures:

Non-primitive data structures are more complex data structures. These data structures are derived from the primitive data structures. They stress on formation of groups of homogeneous and heterogeneous data elements. **Example – Arrays, List, Files**.

2. Mention the different operations performed on primitive data structure.

A:

The various operations that can be performed on primitive data structures

are:

- **Create:** Create operation is used to create a new data structure. This operation reserves memory space for the program elements. It can be carried out at compile time and run-time.

  For example, int x;

- **Destroy:** Destroy operation is used to **destroy or remove the data structures from the memory space**. When the program execution ends**, the data structure is automatically destroyed and the memory allocated is eventually de-allocated**. C++ allows the **destructor member function** destroy the object**.**

- **Select:** Select operation is used by programmers to access the data within data structure. This operation updates or alters data.

- **Update:** Update operation is used to change data of data structures. An assignment operation is a good example of update operation.

For example, int x = 2; Here, 2 is assigned to x.

Again, x = 4; 4 is reassigned to x. The value of x now is 4 because 2 is automatically replaced by 4, i.e. updated.

3. What is linear and non-linear data structure?

A: Linear data structures are a kind of data structure that has homogenous elements. Each element is referred to by an index. The linear data structures are **Stack, Queues and Linked Lists.**

A non-linear data structure is a data structure in which a **data item is connected to several other data items**. The data item has the possibility to reach one or more data items. Every data item is attached to several other data items in a way that is specific for reflecting relationships. **The data items are not arranged in a sequential structure.**

Example- Trees, Graphs.

4. What is an array? Mention the different types of arrays.

A: An array is a collection of homogeneous elements with unique name and the elements are arranged one after another in adjacent memory location.

The data items in an array are called as **elements**. These elements are accessed by numbers called as **subscripts** or **indices**. ***Since the elements are accessed using subscripts, arrays are also called as subscripted variables***.

There are three types of array:

1) **One-dimensional Array** - An array with only one row or column is called one dimensional array

2)**Two-dimensional Array -** A two dimensional array is a collection of elements in which each element is identified by a pair of indices called **subscripts**.

3) **Multi-dimensional array**- **Multidimensional arrays** are an extension of 2-D matrices and use additional subscripts for indexing. A 3-D **array**, for example, uses three subscripts.

5. Mention two types of searching techniques.

A: The two types of searching techniques are Linear and Binary Search.

Linear Search - This is the simplest method in which the element to be searched is compared with each element of the array one by one from the beginning till end of the array. Since searching is one after the other it is also called as sequential search or linear search. It is slow compared to binary search.

Binary Search - When the elements of the array are in **sorted order**, the best method of searching is binary search. This method compares the element to be searched with the **middle element of the array**. If the comparison does not match the element is searched either at the right-half of the array or at the left-half of the array.

6. Write any two applications and arrays.

A:

1. Arrays are used to implement other data structures such as heaps, hash, tables, queues, stacks and strings etc.

2. Arrays are used to implement **mathematical vectors and matrices.**

3. Many databases include one-dimensional arrays whose elements are records.

7. What are PUSH and POP operations on stacks?

A:

push(item) adds a new item to the **top of the stack**. It needs the item and returns nothing.

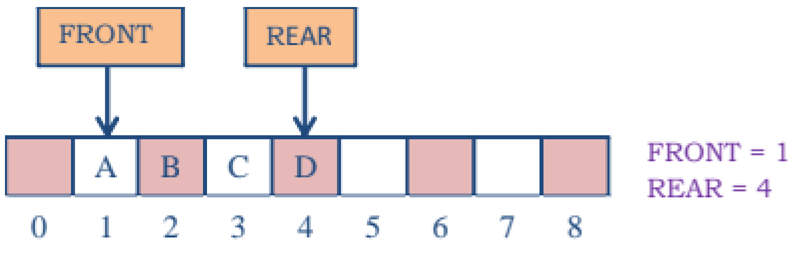
pop() removes the top item from the stack. It needs no parameters and **returns the item**. The stack is modified.

8. Write the memory representation of queues using arrays.

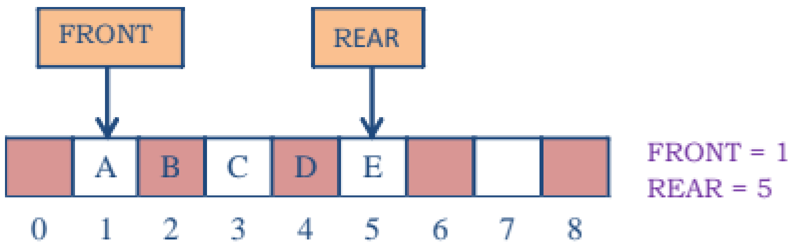
A:

Queue is represented in memory linear array. Let QUEUE be a linear queue. Two pointer variables called **FRONT and REAR are maintained.** The pointer variable FRONT contains **the location of the element to be removed** and the pointer variable REAR contains **location of the last element inserted**.

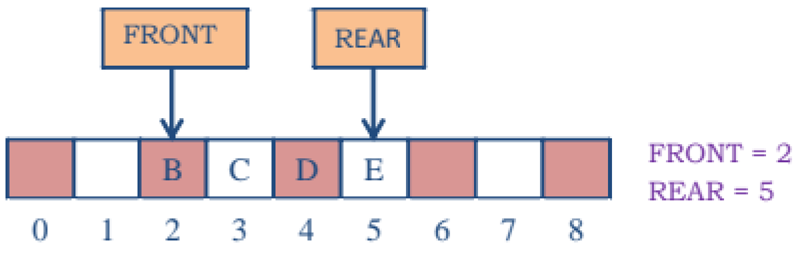
The condition **FRONT = NULL** indicates that the queue is empty and the condition REAR = N-1 indicates that the queue is full.



REAR = REAR +1,   QUEUE[REAR] = ' E'

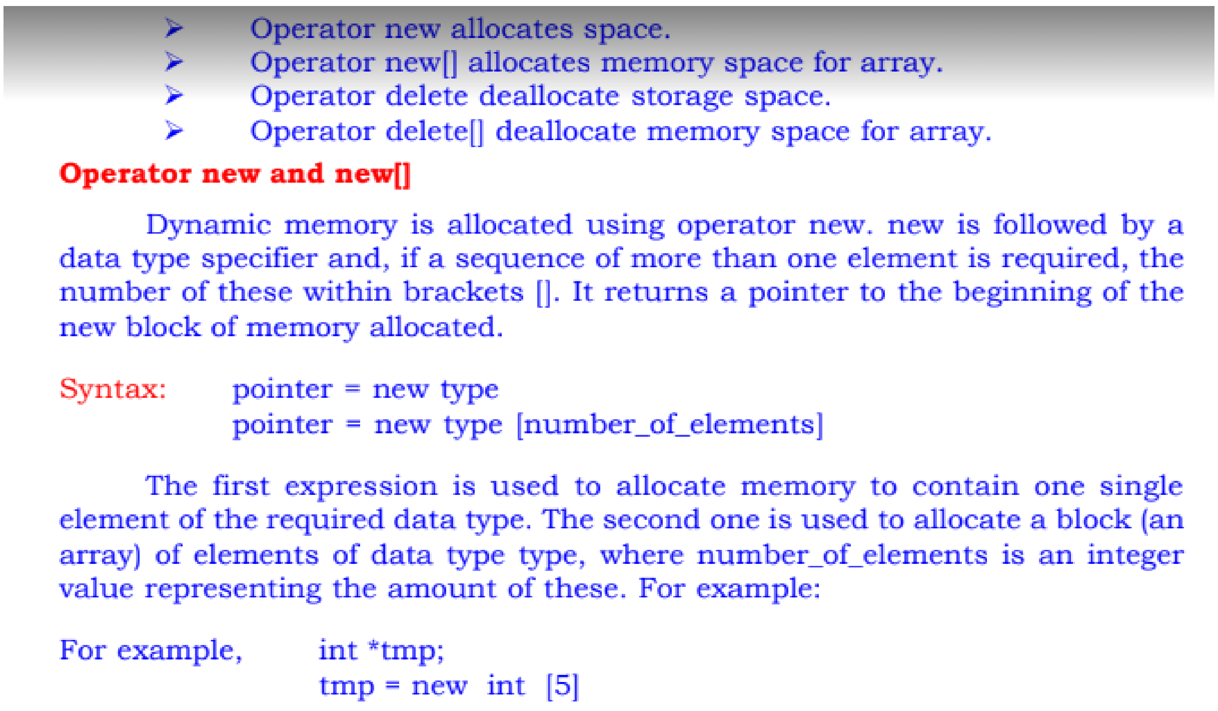


ITEM = QUEUE[FRONT], FRONT = FRONT + 1



9. What is the purpose of new and delete operators?

A:



- ➢ Operator new allocates space.
- ➢ Operator new[] allocates memory space for array.
- ➢ Operator delete deallocate storage space.
- ➢ Operator delete[] deallocate memory space for array.

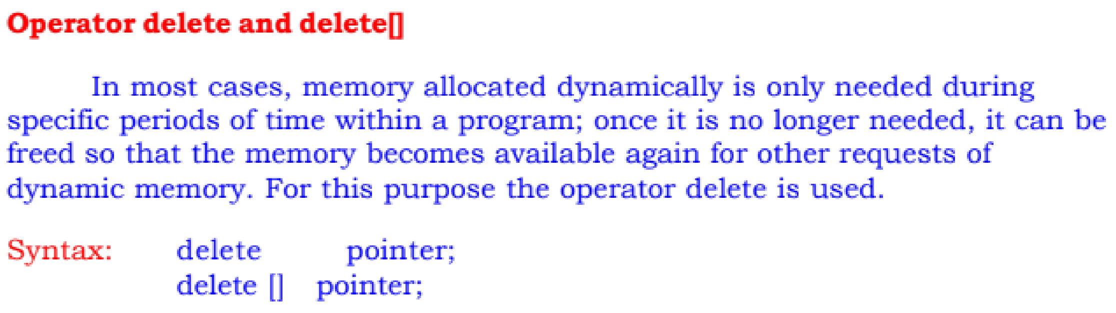
Operator new and new[]

Dynamic memory is allocated using operator new. new is followed by a data type specifier and, if a sequence of more than one element is required, the number of these within brackets []. It returns a pointer to the beginning of the new block of memory allocated.

```
Syntax:      pointer = new type
             pointer = new type [number_of_elements]
```

The first expression is used to allocate memory to contain one single element of the required data type. The second one is used to allocate a block (an array) of elements of data type type, where number_of_elements is an integer value representing the amount of these. For example:

```
For example,      int *tmp;
                  tmp = new  int  [5]
```



Operator delete and delete[]

In most cases, memory allocated dynamically is only needed during specific periods of time within a program; once it is no longer needed, it can be freed so that the memory becomes available again for other requests of dynamic memory. For this purpose the operator delete is used.

```
Syntax:      delete        pointer;
             delete []   pointer;
```

10. Mention the various operations performed on stacks.

A:

- **stack()** creates a new stack that is empty. It needs no parameters and returns an empty stack.

- **push(item)** adds a new item to the top of the stack. It needs the item and returns nothing.

- **pop()** removes the top item from the stack. It needs no parameters and returns the item. The stack is modified.

- **peek()** returns the **top item from the stack** but does not remove it. It needs no parameters. The stack is not modified.

- **isEmpty()** tests whether the stack is empty. It needs no parameters and returns a Boolean value.

- **size()** returns the number of items on the stack. It needs no parameters and returns an integer**.**

11. Mention the various operations performed on queues.

A:

- **Queue()** creates a new queue that is empty. It needs no parameters and returns an empty queue.

- **enqueue(item)** adds a new item to the rear of the queue. It needs the item and returns nothing. This operation is generally called as **push**.

- **dequeue()** removes the front item from the queue. It needs no parameters and returns the item. The queue is modified. This operation is generally called as **pop**.

- **isEmpty()** tests to see whether the queue is empty. It needs no parameters and returns a Boolean value.

- **size()** returns the number of items in the queue. It needs no parameters and returns an integer.

12. What are the different types of linked lists?

A:

There are three types of linked lists.

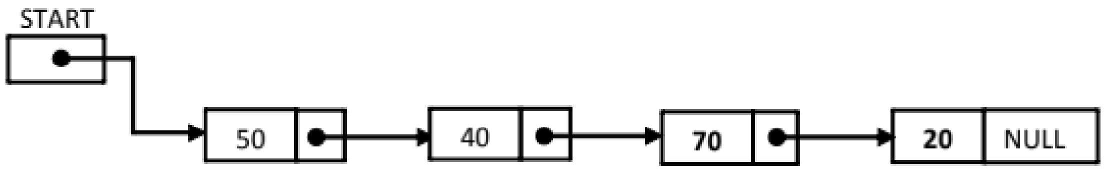
1. Singly linked list (SLL)

2. Doubly linked list (DLL)

3. Circular linked list (CLL)

Single linked list

A singly linked list contains two fields in each node – the **data field and link field**. The data field contains the **data** of that node while the link field contains **address of the next node.** Since there is **only one link field** in each node, the linked list is called as singly linked list.



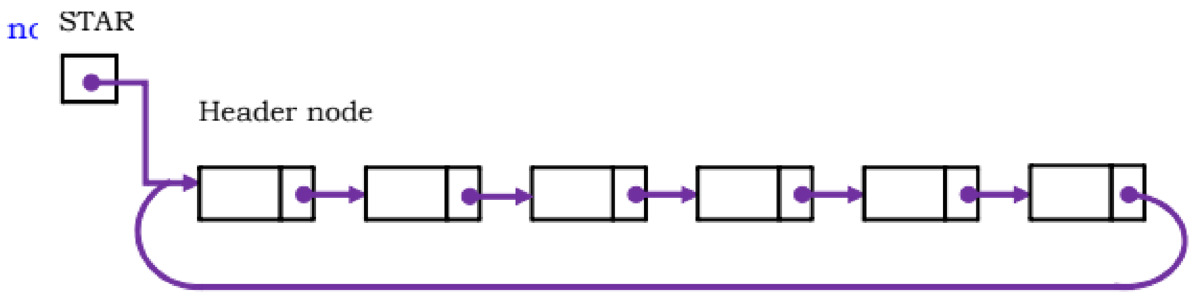
Circular linked lists:

In the singly linked lists, the link field of the

last node contains NULL.

In circular lists, if the link field of the last node contains the **address of the first node** , such a linked list is called as circular linked list.

In a circular linked list, it is possible to reach any node from any other



Doubly linked lists:

It is a linked list in which **each node is points both to the next node and also to the previous node**.

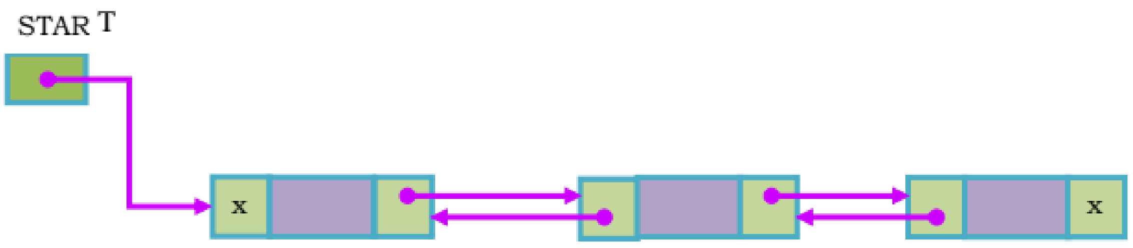
In doubly linked list each node contains three parts – FORW, BACK and INFO.

BACK: It is a pointer field containing the address of the previous node.

FORW: It is a pointer field that contains the address of the next node.

INFO: It contains the actual data.

In the first node, if **BACK contains NULL**, it indicates that it is the **first node** in the list. **The node in which FORW contains NULL indicates that the node is the last node in the linked list**.



Three marks questions:

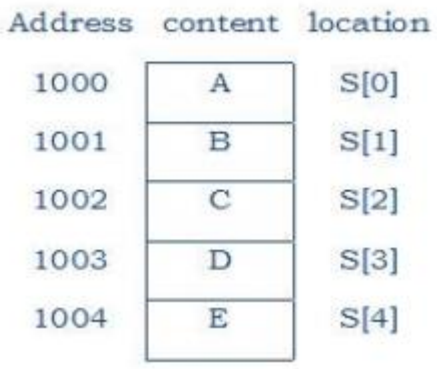
1. Give the memory representation of one-dimensional array.

A: Elements of linear array are stored in consecutive memory locations**.**

Let P be the location of the element. Address of first element of linear array A is given by **Base(A)** called the **Base address of A**. Using this we can calculate the address of any element of A by the formula

LOC(A[P]) = Base(A) + W(P – LB)

Here W is the number of words per memory cell**.**



Address	content	location
1000	A	S[0]
1001	B	S[1]
1002	C	S[2]
1003	D	S[3]
1004	E	S[4]

Example: Suppose if a string S is used to store a string ABCDE in it with starting address at 1000, one can find the address of fourth element as follows:

Now the **address of element S[3]** can be calculated as follows:

Address(S[3]) = Base(S) + W(P - LB)Here W = 1 for characters

= 1000 + 1(3 - 0)

= 1003.

2. Write an algorithm for traversing an array.

A:

Algorithm: Let A be a linear array with LB and UB as lower bound and upper bound. This algorithm traverses the array A by applying the operation PROCESS to each element of A.

```
Step 1 : for LOC = LB to UB
        PROCESS A[LOC]
        [End of for loop]
Step 2 : Exit
```

3. Write the memory representation arrays in row-major order.

A:

Suppose A is the array of order m x n. To store m*n number of elements, we need m*n memory locations. The elements should be in contiguous memory locations.

There are two methods:

- **Row-major order**
- **Column-major order**

Row-major order :

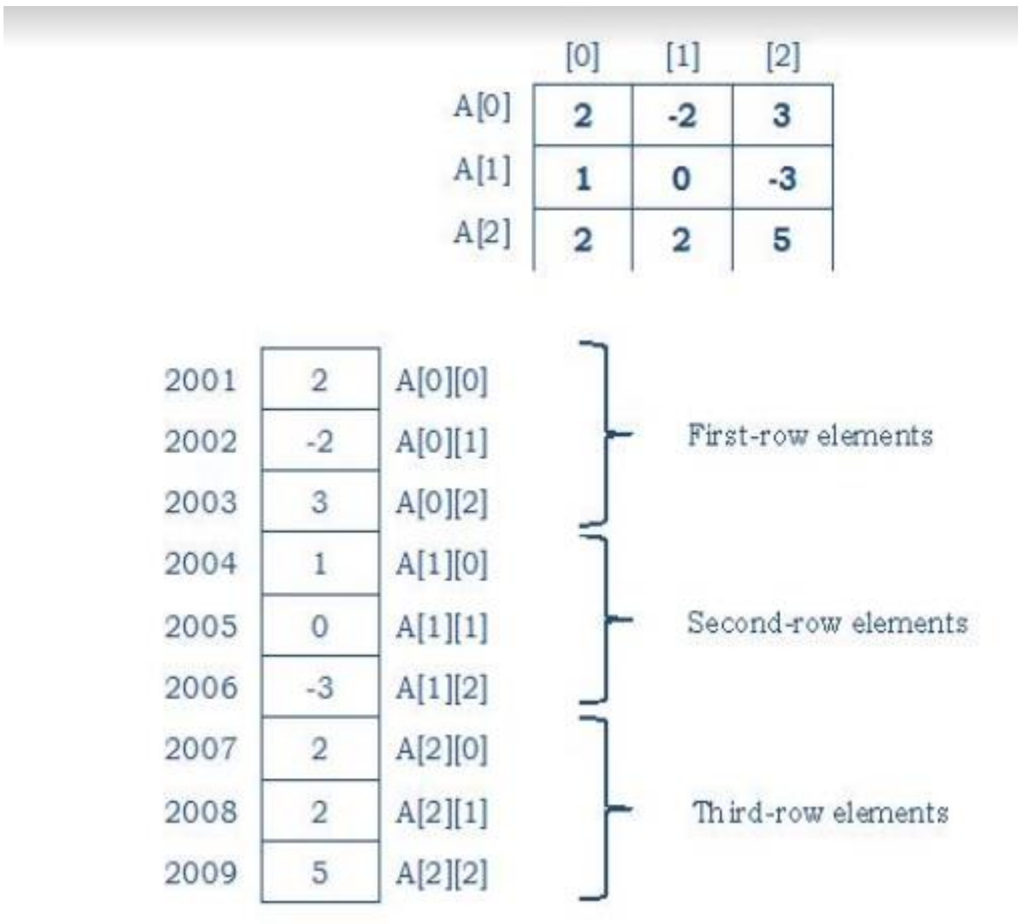
Let A be the array of order m x n. In row-major order, **all the first-row elements are stored in sequential memory locations and then all the second-row elements are stored and so on.**

Base(A) is the address of the first element. The **memory address** of any element A[I][J] can be obtained by the formula :-

LOC(A[I][J]) = Base(A) + W[n(I-LB) + (J-LB)]

where W is the number of words per memory location.

Example: Consider the array of order 3 x 3.



4. Write the memory representation arrays in column-major order.

A:

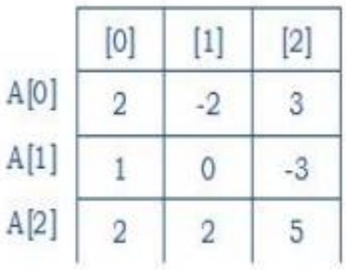
Let A be the array of order m x n. **In column-major order, all the first- column elements are stored in sequential memory locations and then all the second-column elements are stored and so on.**

Base(A) is the **address of the first element**. The memory address of any element A[I][J] can be obtained by the formula

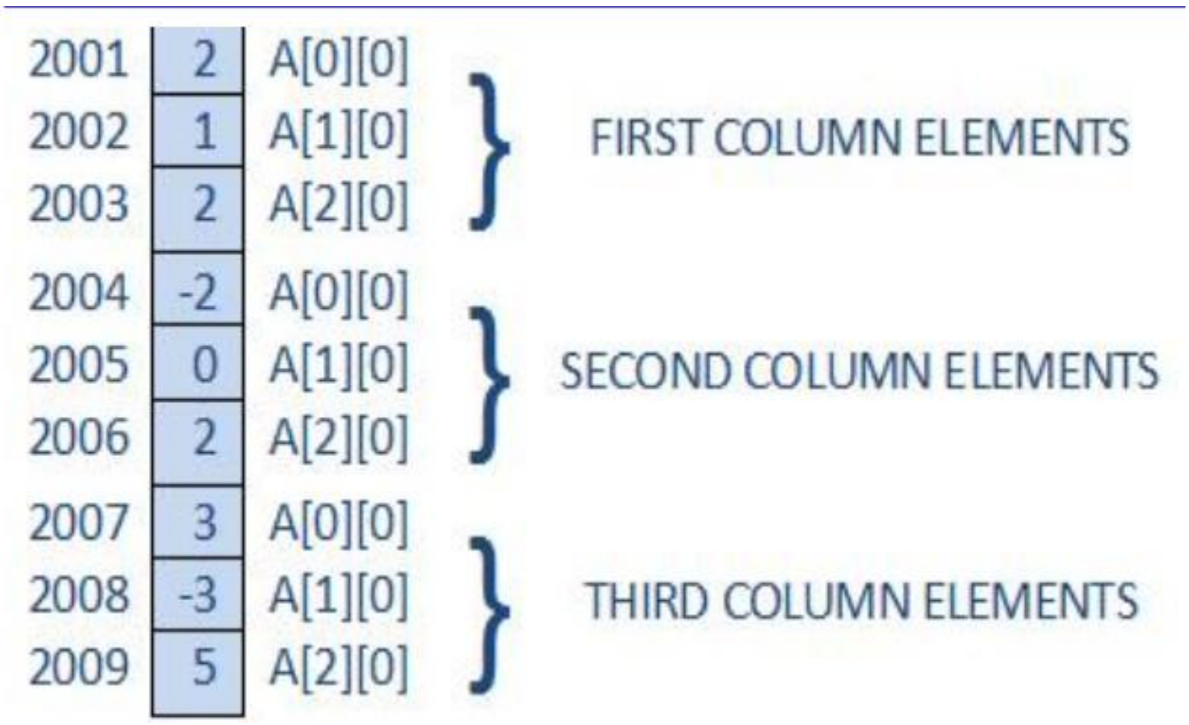
LOC(A[I][J]) = Base(A) + W[(I-LB) + m(J-LB)]

where W is the number of words per memory location.

Example: Consider the array of order 3 x 3.



	[0]	[1]	[2]
A[0]	2	-2	3
A[1]	1	0	-3
A[2]	2	2	5



2001	2	A[0][0]	FIRST COLUMN ELEMENTS
2002	1	A[1][0]	
2003	2	A[2][0]	
2004	-2	A[0][0]	SECOND COLUMN ELEMENTS
2005	0	A[1][0]	
2006	2	A[2][0]	
2007	3	A[0][0]	THIRD COLUMN ELEMENTS
2008	-3	A[1][0]	
2009	5	A[2][0]	

5. Consider the array A of order 25 x 4 with base value 2000 and one word per memory location. Find the memory address of A[12][3] in row-major order.

A:

Given Base(A) = 2000, m = 25, n = 4 LB = 0

W = 1, I = 12, J = 3

Row-major order**: LOC(A[I][J]) = Base(A) + W[n(I-LB) + (J-LB)]**

LOC(A[12][3]) = 2000 + 1[4(12-0)+(3-0)]

= 2000 + 4(12) + 3

= 2000 + 48 + 3

= 2051

6. Consider the array A of order 25x4 with base value 2000 and one word per memory location. Find the address of A[12][3] in column-major order.

A:

Given Base(A) = 2000, m = 25, n = 4 LB = 0

W = 1, I = 12, J = 3

Column Major Order formula =

LOC(A[I][J]) = Base(A) + W[(I-LB) + m(J-LB)]

LOC(A[12][3]) = 2000 + 1[(12-0)+25(3-0)]

= 2000 + 1(12 + 75)

= 2000 + 87

= 2087

7. What are the advantages of arrays?

A:

- Arrays are used to implement mathematical vectors and matrices.
- It is used to **represent multiple data items** of **same type** by using only single name.
- It can be used to **implement** other data structures like **linked lists, stacks, queues, trees, graphs** etc.
- Two-dimensional arrays are used to represent **matrices**.

8. What are the disadvantages of arrays?

A:

1. We must know in advance that how many elements are to be stored in array(size)

2. Array is static structure. It means that array is of fixed size. The memory which is allocated to array cannot be increased or reduced.

3. Since array is of fixed size, if we allocate more memory than requirement then the memory space will be wasted. If we allocate less memory than requirement, then it will create problem.

4. The elements of array are stored in consecutive memory locations. So insertions and deletions are very difficult and time consuming.

9. Explain the memory representation stacks using array.

A:

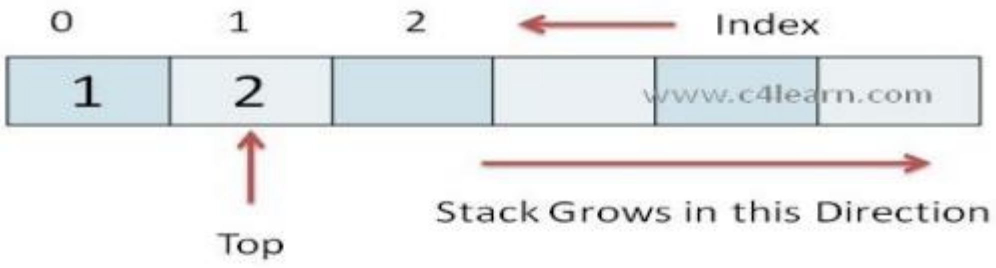
- Stack can be represented using a **one-dimensional array**. A block of memory is allocated which is required to accommodate the items to the full capacity of the stack. **The items into the stack are stored in a sequential order from the first location of the memory block.**

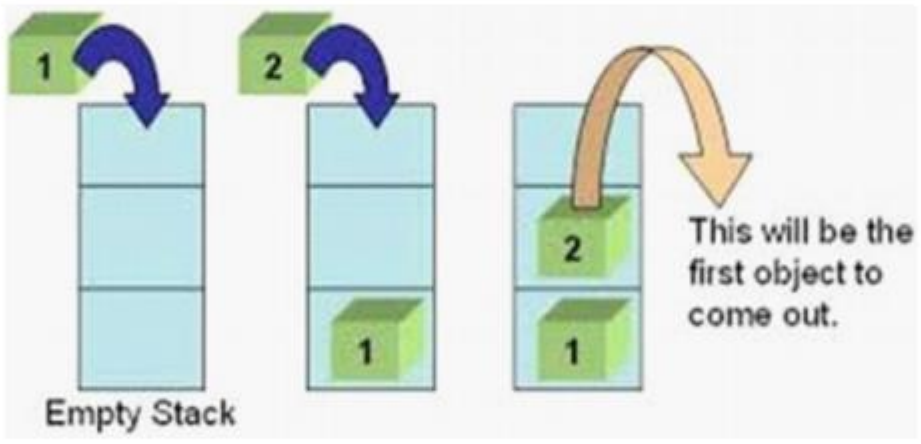
- **A pointer TOP contains the location of the top element of the stack. A variable MAXSTK contains the maximum number of elements that can be stored in the stack.**

- **The condition TOP = MAXSTK indicates that the stack is full and TOP = NULL indicates that the stack empty.**

- Representing a stack using arrays is easy and convenient. However, it is useful for **fixed sized stacks.** Sometimes in a program, the size of a stack may be required to increase during execution, i.e. **dynamic creation of a stack.**

- Dynamic creation of a stack is not possible using arrays. This requires **linked lists**.





10. Write an algorithm for push operation.

A:

PUSH(STACK , TOP, SIZE, ITEM)

STACK is the array that contains N elements and TOP is the pointer to the top element of the array. ITEM the element to be inserted. This procedure inserts ITEM into the STACK.

```
Step 1: If TOP = N -1then [check overflow]
            PRINT “Stack is full”
            Exit
            [End of If]
```

```
    Step 2: TOP = TOP + 1 [Increment the TOP]
    Step 3: STACK[TOP] = ITEM [Insert the ITEM]
    Step 4: Return
```

11. Write an algorithm for POP operation.

A:

POP(STACK , TOP, ITEM)

STACK is the array that store N items. TOP is the pointer to the top element of the array. This procedure deleted top element from STACK.

```
Step 1: If TOP = NULL then [check underflow]
    PRINT "Stack is empty"
    Exit
    End of If

Step 2: ITEM = STACK[TOP] [Copy the top element]
Step 3: TOP = TOP – 1 [Decrement the top]
Step 4: Return
```

12. Write any three applications of stacks.

A:

- The simplest application of a stack is to reverse a word. You push a given word to stack - letter by letter - and then pop letters from the stack
- Another application is an “undo” mechanism in text editors; this operation is accomplished by keeping all text changes in a stack.
- **Backtracking**: ***This is a process when you need to access the most recent data element in a series of elements.*** Once you reach a dead end, you must backtrack. But backtrack to where? to the previous choice point. Therefore, at each choice point you store on a stack all possible choices. Then backtracking simply means popping a next choice from the stack.
- Conversion of decimal number into binary
- To solve tower of Hanoi
- Expression evaluation and syntax parsing
- Conversion of infix expression into prefix and postfix.

13. Write any three applications of queues.

A:

➔Simulation

- ➔Various features of operating system.

[Operating systems often maintain a queue of processes that are ready to execute or that are waiting for a particular event to occur.]

- ➔Multi-programming platform systems

- ➔Different type of scheduling algorithm

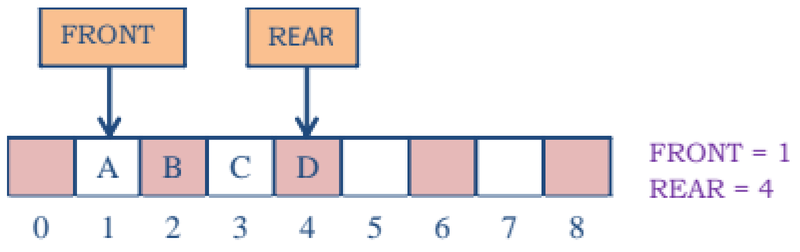
- ➔ Round robin technique or Algorithm

- ➔ Printer server routines

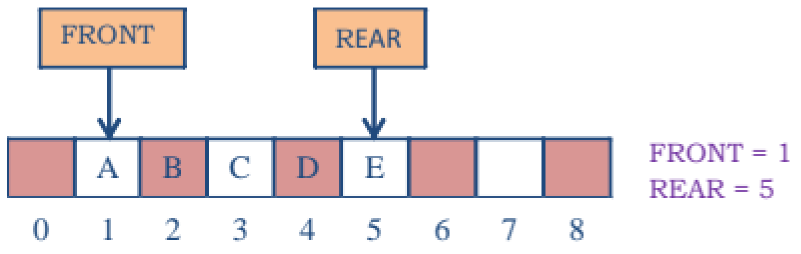
14. Explain the memory representation of queues using array.

A:

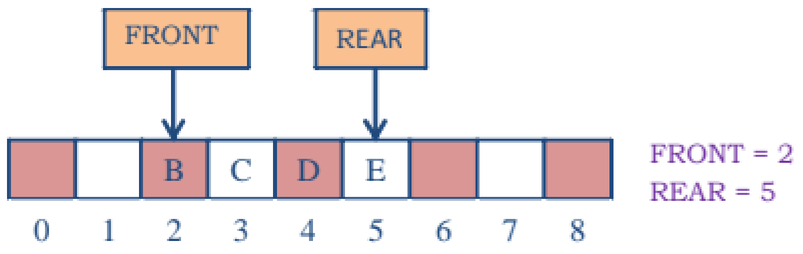
Queue is represented in memory linear array. Let QUEUE be a linear queue. Two pointer variables called **FRONT** and **REAR** are maintained. **The pointer variable FRONT contains the location of the element to be removed** and the **pointer variable REAR contains location of the last element inserted**. The condition **FRONT = NULL** indicates that the **queue is empty** and the **condition REAR = N** indicates that the **queue is full**.



REAR = REAR +1,   QUEUE[REAR] =' E'



ITEM = QUEUE[FRONT], FRONT = FRONT + 1



15. Explain types of linked list.

A:

There are three types of linked lists.

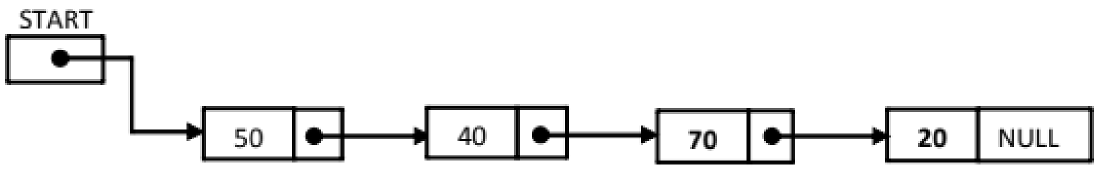
1. Singly linked list (SLL)

2. Doubly linked list (DLL)

3. Circular linked list (CLL)

Singly Linked List :-

A singly linked list contains two fields in each **node** – the **data field and link field**. ***The data field contains the data of that node while the link field contains address of the next node.*** Since there is only one link field in each node, the linked list is called as singly linked list.

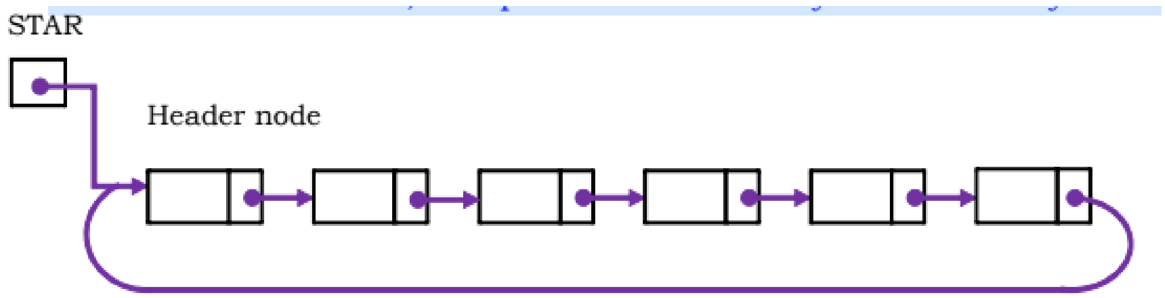


Circular linked lists:

In a **singly linked list**, a pointer is available to access all the **succeeding nodes,** but not **preceding nodes.** In the singly linked lists, the link field of the last node contains **NULL**.

In circular lists, the link field of the last node contains the address of the first node first node, such a linked list is called as circular linked list.

In a circular linked list, it is possible to reach any node from any other node



Doubly linked lists:

It is a linked list in which ***each node is points both to the next node and also to the previous node***.

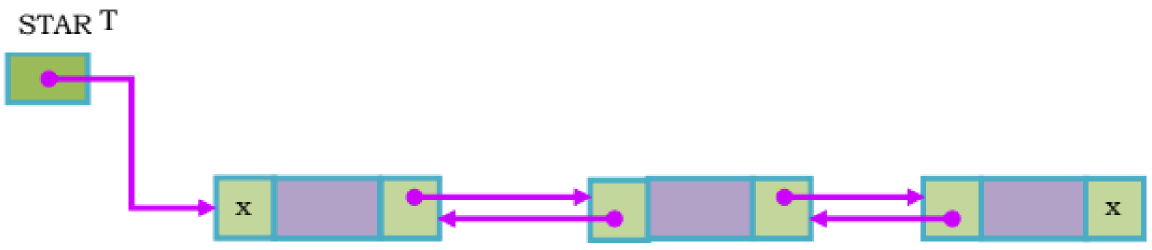
In doubly linked list each node contains three parts – **FORW, BACK and INFO.**

BACK: It is a pointer field containing the address of the previous node.

FORW: It is a pointer field that contains the address of the next node.

INFO: It contains the actual data.

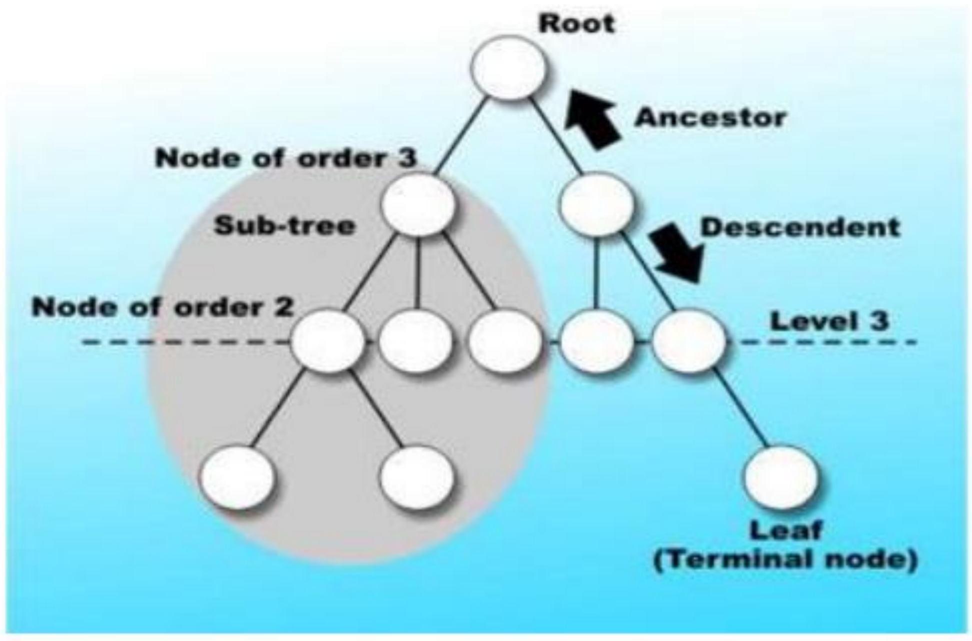
In the first node, **if BACK contains NULL**, it indicates that it is the **first node** in the list. The node in which **FORW contains NULL** indicates that the **node is the last node in the linked list.**



16. Define the following: a. Tree b. Graph c. Root node.

A:

a) A tree is a data structure consisting of nodes organized as a hierarchy as shown below.



b) A graph is a set of **vertices and edges** which connect them. A graph is a collection of nodes called vertices, and the connections between them, called edges**.**

c) Root Node: Node at the "top" of a tree - the one from which all operations on the tree commence. The root node may not exist (a NULL tree with no nodes in it) or have 0, 1 or 2 children in a binary tree**.**

Five marks questions:

1. What is primitive data structure? Explain the different operations performed on primitive data structure.

A: Data structures that are directly operated upon by machine-level instructions are known as primitive data structures.

The various operations that can be performed on primitive data structures are:

- **Create:** Create operation is used to create a new data structure. This operation reserves memory space for the program elements. It can be carried out at compile time and run-time.

For example, int x;

- **Destroy:** Destroy operation is used to **destroy or remove the data structures from the memory space**. When the program execution ends**, the data structure is automatically destroyed and the memory allocated is eventually de-allocated**. C++ allows the **destructor member function** destroy the object**.**

- **Select:** Select operation is used by programmers to access the data within data structure. This operation updates or alters data.

- **Update:** Update operation is used to change data of data structures. An assignment operation is a good example of update operation.

For example, int x = 2; Here, 2 is assigned to x.

Again, x = 4; 4 is reassigned to x. The value of x now is 4 because 2 is automatically replaced by 4, i.e. updated.

2. Explain the different operations performed on linear data structure.

A: The basic operations on non-linear data structures are as follows:

- ➔ **Traversal:** The process of accessing each data item exactly once to perform some operation is called traversing.

- ➔ **Insertion:** The process of adding a new data item into the given collection of data items is called insertion.

- ➔ **Deletion:** The process of removing an existing data item from the given collection of data items is called deletion.

- ➔ **Searching:** The process of finding the location of a data item in the given collection of data items is called as searching.

- ➔ **Sorting:** The process of arrangement of data items in ascending or descending order is called sorting**.**

➔ **Merging:** The process of combining the data items of two structures to form a single structure is called merging.

3. Write an algorithm for searching an element using linear search method.

A:

Algorithm: A is the name of the array with N elements. ELE is the element to be searched. This algorithm finds the location loc where the search ELE element is stored**.**

```
Step 1: LOC = -1
Step 2: for P = 0 to N-1
            if( A[P] = ELE)
            LOC = P
            GOTO Step 3
            [End of if]
            [End of for]

Step 3: if(LOC >= 0)
            PRINT LOC
```

```
        else
        PRINT “Search is unsuccessful”
```

Step 4: Exit

4. Write an algorithm for searching an element using binary search method.

A:

Algorithm: A is the **sorted array** with **LB as lower bound** and **UB as the upper bound** respectively. Let B, E, M denote beginning, end and middle locations of the segments of A.

```
Step 1: set B = 0, E = n-1 LOC=-1
Step 2: while (B <= E)
        M= int(B+E)/2
        if(ELE = A[M])
            loc = M
            GOTO Step 4
        else
        if(ELE <A[M])
            E = M-1
        else
            B = M+1
        End of while
**Step 3:** if(LOC >= 0)
        PRINT LOC
      else
    PRINT "Search is unsuccessful"
**Step 4:** Exit
```

5. Write an algorithm for inserting an element into the array.

A:

Algorithm: A is the array with N elements. ITEM is the element to be inserted in the position P.

```
**Step 1:** for I = N-1 down to P
        A[I+1] = A[I]
        [End of for]
**Step 2:**  A[P] = ITEM
```

Step 3: N = N+1

Step 4: Exit

6. Write an algorithm for deleting an element from the array.

A:

Algorithm: A is the array with N elements. ITEM is the element to be deleted in the position P and it is stored into the variable Item.

```
Step 1: Item = A[P]
Step 2: for I = P to N-1
            A[I] = A[I+1]
            End of for
Step 2: N = N-1
Step 4: Exit
```

7. Explain the different operations performed on stacks.

A:

➔ **stack()** creates a new stack that is empty. It needs no parameters and returns an empty stack.

➔ **push(item)** adds a new item to the top of the stack. It needs the item and returns nothing**.**

➔ **pop()** removes the top item from the stack. It needs no parameters and returns the item. The stack is modified.

➔ **peek()** returns the top item from the stack but does not remove it. It needs no parameters. The stack is not modified.

➔ **isEmpty()** tests whether the stack is empty. It needs no parameters and returns a Boolean value**.**

➔ **size()** returns the number of items on the stack. It needs no parameters and returns an integer.

8. Write the applications of stacks.

A:

- The simplest application of a stack is to reverse a word. You push a given word to stack - letter by letter - and then pop letters from the stack

- Another application is an “undo” mechanism in text editors; this operation is accomplished by keeping all text changes in a stack.
- **Backtracking**: ***This is a process when you need to access the most recent data element in a series of elements.*** Once you reach a dead end, you must backtrack. But backtrack to where? to the previous choice point. Therefore, at each choice point you store on a stack all possible choices. Then backtracking simply means popping a next choice from the stack.
- Conversion of decimal number into binary
- To solve tower of Hanoi
- Expression evaluation and syntax parsing
- Conversion of infix expression into prefix and postfix.

9. What is a queue? Explain different types of queues.

A:

A queue is an ordered collection of items where an **item is inserted at one end called the “rear,”** and an **existing item is removed at the other end, called the “front.”** Queues maintain a FIFO ordering property.

Queue can be of four types:

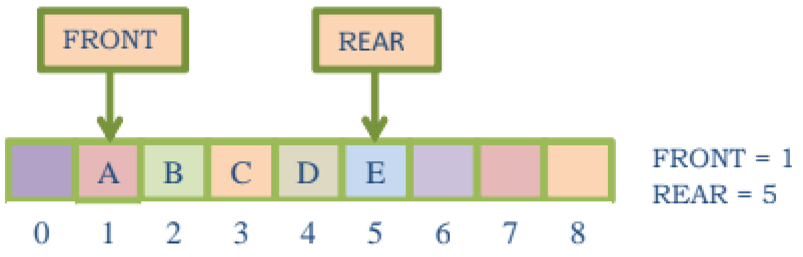
1. Simple Queue

2. Circular Queue

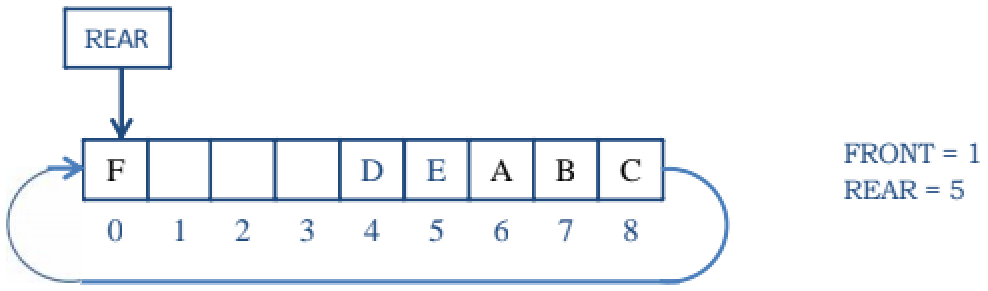
3. Priority Queue

4. Dequeue (Double Ended queue)

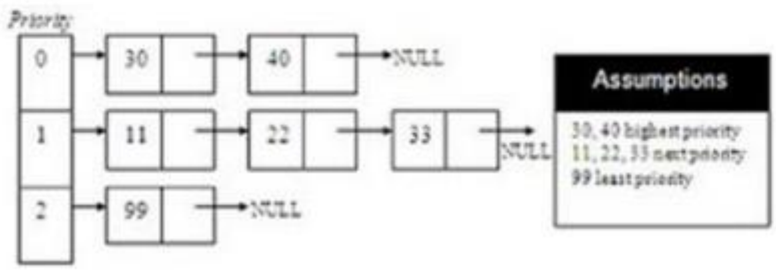
Simple Queue: In Simple queue insertion occurs at the rear end of the list, and deletion occurs at the front end of the list.



Circular Queue: A circular queue is a queue in which all nodes are treated as circular such that the last node follows the first node.

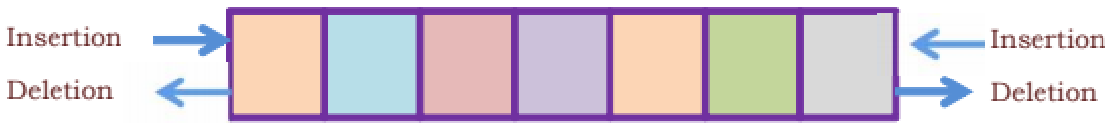


Priority Queue: A priority queue is a queue that contains items that have some preset priority. An element can be inserted or removed from any position depending on some priority.



Dequeue (Double Ended queue):

It is a queue in which insertion and deletion takes place at both the ends.



10. Explain the different operations performed on queues.

A:

➔**Queue()** creates a new queue that is empty. It needs no parameters and returns an empty queue.

➔ **enqueue(item)** adds a new item to the rear of the queue. It needs the item and returns nothing. This operation is generally called as push.

➔ **dequeue()** removes the front item from the queue. It needs no parameters and returns the item. The queue is modified. This operation is generally called as pop.

➔ **isEmpty()** tests to see whether the queue is empty. It needs no parameters and returns a Boolean value.

➔ **size()** returns the number of items in the queue. It needs no parameters and returns an integer.

11. Write an algorithm to insert an item into the queue.

A:

Algorithm: Let QUEUE be the linear array consisting of N elements.

FRONT is the pointer that contains the location of the element to be deleted and REAR contains the location of the inserted element. ITEM is the element to be inserted.

```
Step 1: If REAR = N-1 Then [Check for overflow]
    PRINT "Overflow"
    Exit
Step 2: If FRONT = NULL Then [Check whether QUEUE is empty]
        FRONT = 0
        REAR = 0
      Else
        REAR = REAR + 1 [Increment REAR Pointer]
Step 3: QUEUE[REAR] = ITEM [Copy ITEM to REAR position]
Step 4: Return
```

12. Write an algorithm to delete an item from the queue.

A:

Algorithm: Let QUEUE is the linear array consisting of N elements.

FRONT is the pointer that contains the location of the element to be deleted and REAR contains the location of the inserted element. This algorithm deletes the element at FRONT position.

```
Step 1: If FRONT = NULL Then [Check whether QUEUE is empty]
        PRINT "Underflow"
        Exit

Step 2 : ITEM = QUEUE[FRONT]
Step 3: If FRONT = REAR Then [If QUEUE has only one element]
            FRONT = NULL
            REAR = NULL
        Else
            FRONT = FRONT + 1 [Increment FRONT pointer]
Step 4: Return
```

13. Write the applications of queues.

A:

- ➔Simulation
- ➔Various features of operating system.

[Operating systems often maintain a queue of processes that are ready to execute or that are waiting for a particular event to occur.]

➔ Multi-programming platform systems

➔ Different type of scheduling algorithm

➔ Round robin technique or Algorithm

➔ Printer server routines

14. What are the operations performed on the linked list?

A:

The operations that are performed on linked lists are

1. Creating a linked list
2. Traversing a linked list
3. Inserting an item into a linked list
4. Deleting an item from the linked list
5. Searching an item in the linked list
6. Merging two or more linked lists

Creating a linked list

- ➔ ***Linked list is linear data structure which contains a group of nodes and the nodes are sequentially arranged***.

- ➔ Nodes are **composed of data and address of the next node** or reference of the next node. These nodes are sequentially or linearly arrayed that is why the Linked list is a linear data structure.

- ➔ In linked list we start with a node and create nodes and link to the starting node in order and sequentially. The pointer START contains the location of the first node. Also the next pointer field of the last node must be assigned to NULL.

The nodes of a linked list can be created by the following structure declaration.

```
struct Node

{

int data;

Node* link;

}

Node *node1, *node2;

OR
```

```
struct Node
{
int data;
Node* link;
} *node1, node2;
```

15. Define the following : a. Root Node b. Leaf Node c. Height d. Depth e. Internal node.

A:

a)Root Node - Node at the “top” of a tree - the one from which all operations on the tree commence. The root node may not exist (a NULL tree with no nodes in it) or have 0, 1 or 2 children in a binary tree.

b) **Leaf Node -** Node at the “bottom” of a tree - farthest from the root. Leaf nodes have no children.

c) **Height of Tree**- Number of nodes which must be traversed from the root to reach a leaf of a tree.

d) **Depth -** The depth of a node is the length of the path to its root

e) An **internal node** or inner node is any node of a tree that has child nodes and is thus not a leaf node.

Note : Please go through Linked List Operations

1) Traversal of Linked List

2) Insertion of node at front of Linked List, Insertion of node at the end of Linked List, Insertion at any position

3) Deletion of node at front of Linked List, Deletion of node at the end of Linked List, Insertion at any position