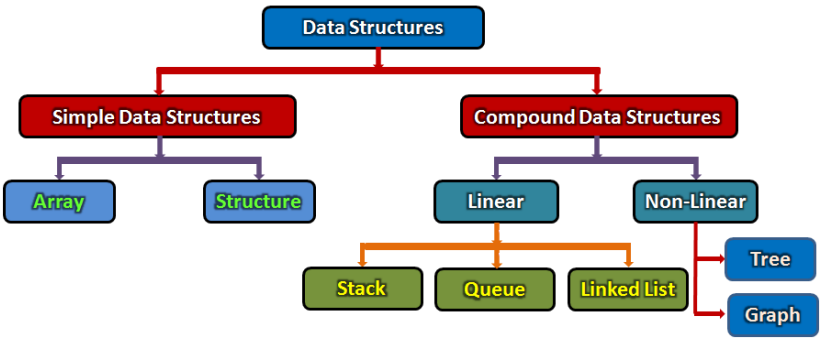
Chapter 3
Data Structures and Operations

Data structure is a particular way of organising logically related data items which can be processed as a single unit.

Classification of data structures



Depending upon memory allocation, data structures may be classified as ***static data structures*** and ***dynamic data structures***. Memory allocation is fixed for static data structures (eg: arrays) and the size cannot be changed during execution. Memory is allocated during execution for dynamic data structures (eg: linked list) and the size changes according to the addition or deletion of data items.

Operations on Data Structures

The operations performed on data structures are traversing, searching, inserting, deleting, sorting and merging.

Traversing is an operation in which each element of a data structure is visited.

Searching is the process of finding the location of a particular element in a data structure.

Insertion is the operation in which a new data is added at a particular place in a data structure.

Deletion is the operation in which a particular element is removed from the data structure.

Sorting is the technique of arranging the elements in a specified order (ascending or descending).

Merging is the process of combining elements of two sorted data structures to form a new one.

Stack and its Operations

Stack is a data structure that follows LIFO (Last In First Out) principle. It is an ordered list of items in which all insertions and deletions are made at one end, usually called **Top**.

Push Operation: It is the process of inserting a new data item into the stack at Top position. Once the stack is full and if we attempt to insert an item, an impossible situation arises, known as ***stack overflow***.

Pop Operation: It is the process of deleting an element from the top of a stack. If we try to delete an item from an empty stack, an unfavourable situation arises, known as ***stack underflow***.

Algorithm for Push:

Assume that STACK[N] is an array of stack with size N and TOS denotes the top position of the stack. Let VAL contains the data to be added into the stack.

```
Start
    1:  If  (TOS < N-1) Then          //Space availability checking (Overflow)
    2:      TOS = TOS + 1
    3:      STACK[TOS] = VAL
    4:  Else
    5:      Print "Stack Overflow "
    6:  End of If
Stop
```

Algorithm for Pop:

Assume that STACK[N] is an array of stack with size N and TOS denotes the top position of the stack. Let VAL be a variable to store the popped item.

```
Start
    1:  If  (TOS > -1) Then           //Empty status checking (Underflow)
    2:      VAL = STACK[TOS]
    3:      TOS = TOS - 1
    4:  Else
    5:      Print "Stack Underflow "
    3:  End of If
Stop
```

Queue and its Operations

Queue is a data structure that follows the FIFO (First In First Out) principle. A queue has two end points - **Front** and **Rear**. Insertion of a data item will be at the rear end and deletion will be at the front end.

Insertion is the process of adding a new item into a queue at the rear end. One the value of Rear equals the last position and if we attempt an insertion, queue overflow occurs.

Deletion is the process of removing the item at the front end of a queue. If we attempt a deletion from an empty queue, underflow occurs.

Algorithm for Insertion:

Assume that Q[N] is an array of queue with size N and FRONT and REAR denote the front and rear positions of the queue. Let VAL contains the data to be added into the queue.

```
Start
    1:  If  (REAR == -1) Then             //Empty status checking
    2:      FRONT = REAR = 0
    3:      Q[REAR] = VAL
    4:  Else If (REAR < N-1) Then         //Space availability checking
    5:      REAR = REAR + 1
    6:      Q[REAR] = VAL
```

```
    7:    Else
    8:        Print "Queue Overflow "
    9:    End of If
Stop
```

Algorithm for Deletion:

Assume that Q[N] is an array of queue with size N and FRONT and REAR denote the front and rear positions of the queue. Let VAL be a variable to store the deleted data.

```
Start
    1:    If  (FRONT > -1) Then          // Empty status checking
    2:        VAL = Q[FRONT]
    3:        FRONT = FRONT + 1
    4:    Else
    5:        Print "Queue Underflow "
    6:    End of If
    7:    If (FRONT > REAR) Then       // Checking the deletion of last element
    8:        FRONT = REAR = -1
    9:    End of If
Stop
```

Circular queue

It is a queue in which the two end points meet. Its advantage over linear queue is that space utilization is the maximum. Overflow occurs only if all the locations are filled with data items.

Linked List and Operations

Linked list is a collection of nodes, where each node consists of two parts – a ***data*** and a ***link***. Link is a pointer to the next node in the list. The address of the first node is stored in a special pointer called START. Linked list is a dynamic data structure. Memory is allocated during run time. So there is no problem of overflow. It grows as and when new data items are added, and shrinks whenever any data is removed. Linked list is created with the help of self referential structures.

Creation of a Linked List:

Step 1: Create a node and obtain its address.
Step 2: Store data and NULL in the node.
Step 3: If it is the first node, store its address in START.
Step 4: If it is not the first node, store its address in the link part of the previous node.
Step 5: Repeat the steps 1 to 4 as long as the user wants.

Traversing a linked list

Step 1: Get the address of the first node from START and store it in Temp.
Step 2: Using the address in Temp, get the data of the first node and store in Val.
Step 3: Also get the content of the link part of this node (i.e., the address of the next node) and store it in Temp.
Step 4: If the content of Temp is not NULL, go to step 2; otherwise stop.

Insertion in a linked list

Step 1: Create a node and store its address in Temp.
Step 2: Store the data and link part of this node using Temp.
Step 3: Obtain the addresses of the nodes at positions (POS-1) and POS in the pointers PreNode and PostNode respectively, with the help of a traversal operation.
Step 4: Copy the content of Temp (address of the new node) into the link part of node at position (POS-1), which can be accessed using PreNode.
Step 5: Copy the content of PostNode (address of the node at position POS) into the link part of the new node that is pointed to by Temp.

Deletion in a linked list

Step 1: Obtain the addresses of the nodes at positions (POS-1) and (POS+1) in the pointers PreNode and PostNode respectively, with the help of a traversal operation.
Step 2: Copy the content of PostNode (address of the node at position POS+1) into the link part of the node at position (POS-1), which can be accessed using PreNode.
Step 3: Free the node at position POS.

Questions from Previous Years' Question Papers

1. Attempting to insert in an already full stack leads to _____. (1) (March 2016)
2. Explain how push operation is done in a stack. (2) (March 2016)
3. Linked list usually do not have the problem of overflow. Discuss. (2) (March 2016)
4. Write two advantages of linked list over arrays. (2) (SAY 2016)
5. Consider the following cases:
   (i) Paper cups are arranged on a dining table one above the other.
   (ii) Many people are waiting in a row to tickets for a cinema.
   Identify and compare the data structures that you know in connection with the above mentioned contexts. (3) (SAY 2016)
6. Queue follows the _____ principle. (1) (March 2017)
7. How does stack overflow and underflow occur? (2) (March 2017)
8. Write a procedure to implement traversal operation in a linked list. (2) (March 2017)
9. Name the data structure that follows LIFO principle.
   (a) stack (b) queue (c) array (d) linked list (1) (SAY 2017)
10. Write an algorithm to perform insertion operation in a Queue. (2) (SAY 2017)
11. Match the following: (2) (SAY 2017)

A	B	C
1. Stack	i. Front	a. Inserting a new item.
2. Queue	ii. Push	b. Elements are accessed by specifying its position.
3. Array	iii. Start	c. Contains the address of the first node.
4. Linked List	iv. Subscript	d. Removing an item.