

# Introduction to Data Structures

## Module 1

# Data

- Data can be numeric or character
- When we process this data, it becomes information.
- A string MH32V2126, is a sequence of character type data but when it represents the car registration number then it is processed data i.e. information.

# DATA STRUCTURES

Data may be organized in many different ways. The logical or mathematical model of a particular organization of data is called a Data Structure.

Or

In other words, a Data Structure is a way of organizing all the data items that considers not only the elements stored but also the relationship between them.

# Data Structure

- Data Structure is a way to organize the data in some way so we can perform operations on these data in effective ways.
- Eg-List, Stack, Queue, Tree

# Selection/choice of data structures

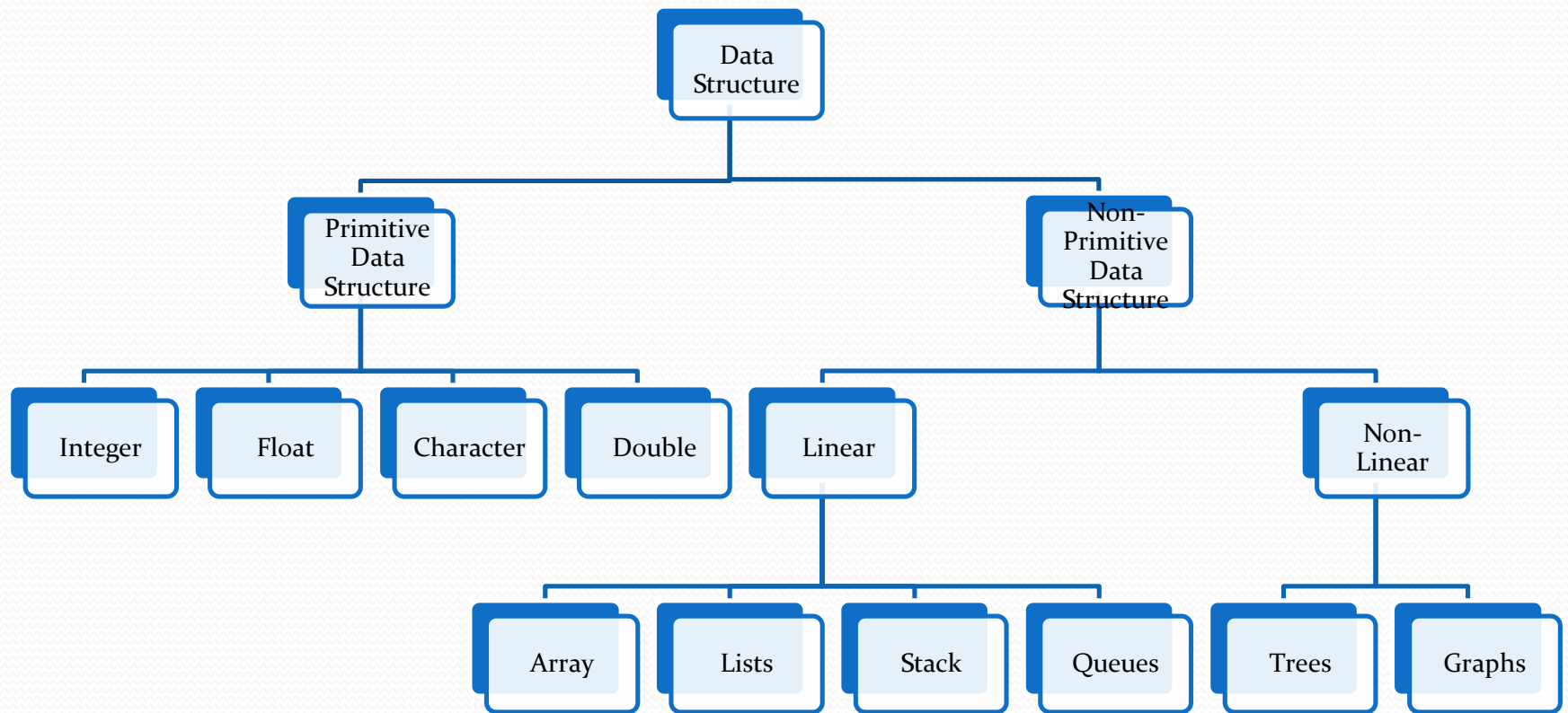
- The choice of Data Structure model depends on two considerations-
  - 1) It must be rich enough in structure to mirror the actual relationships of the data in the real world.
  - 2) The structure should be simple enough that one can effectively process the data when necessary, i.e. the type of operation to be performed.

- Data Structures affects the design of both the structural and functional aspects of a program.

ALGORITHM+DATA STRUCTURE=PROGRAM

- Algorithm is a step by step procedure to solve a particular problem.
- To develop a program we should select an appropriate data structure for that algorithm.
- Thus , algorithm and its associated data structure form a program.

# CLASSIFICATION OF DATA STRUCTURES



# PRIMITIVE DATA STRUCTURES

- These are basic data structures and are directly operated upon by the machine level instructions.
- These in general have different representations on different computers.
- Integers, floating point numbers, character constants, double etc fall in this category.



# NON-PRIMITIVE DATA STRUCTURES

- These are more sophisticated data structures.
- These are derived from the basic data types.
- The non-primitive data structures emphasize on structuring of a
  - group of homogenous(same type) or
  - heterogenous (different type) data items.
- Eg- Arrays, lists, trees etc.

# NON-PRIMITIVE DATA STRUCTURES

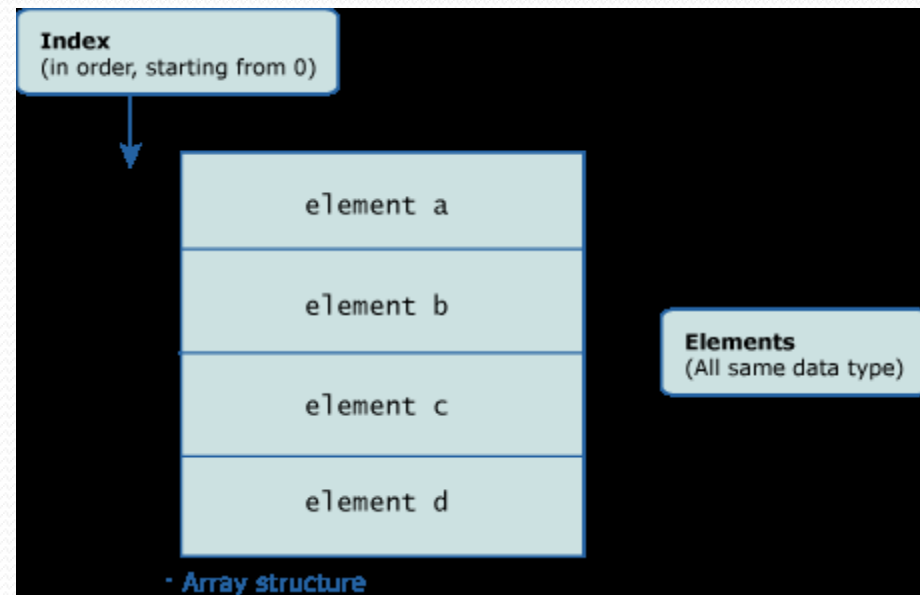
- Classified as
  - Linear data structures
  - Non- Linear data structures

# Linear Data Structures

- A data structure is said to be linear
  - if its elements form a sequence or a linear list.
- Here elements are arranged in one dimension or linear fashion.
- All one-one relation can be handled through linear data structures.
- Eg- Lists, stacks and queues

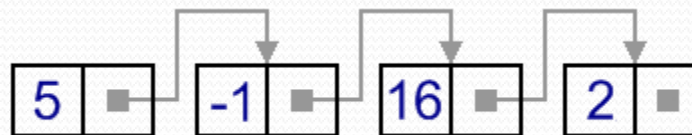
# Arrays

- An array can be defined as a collection of homogenous data type.
- An array can store either all integers, all floating point numbers, all characters or any other complex data type.
- Declaration of an integer array is as follows-  
`int a[10];`



# Lists

- Linked lists are special lists of some data elements linked to one another.
- The logical ordering is represented by
  - having each element pointing to the next element.
- Each element is called a node, which has two parts, INFO part which stores the information and pointer which points to the next element.



# Stacks



# Stacks

- ❑ It could be thought of just like a stack of plates placed on table in a party,
  - ❑ a guest always takes off a fresh plate from the top, and
  - ❑ the new plates are placed on the top of the stack .



# Stacks

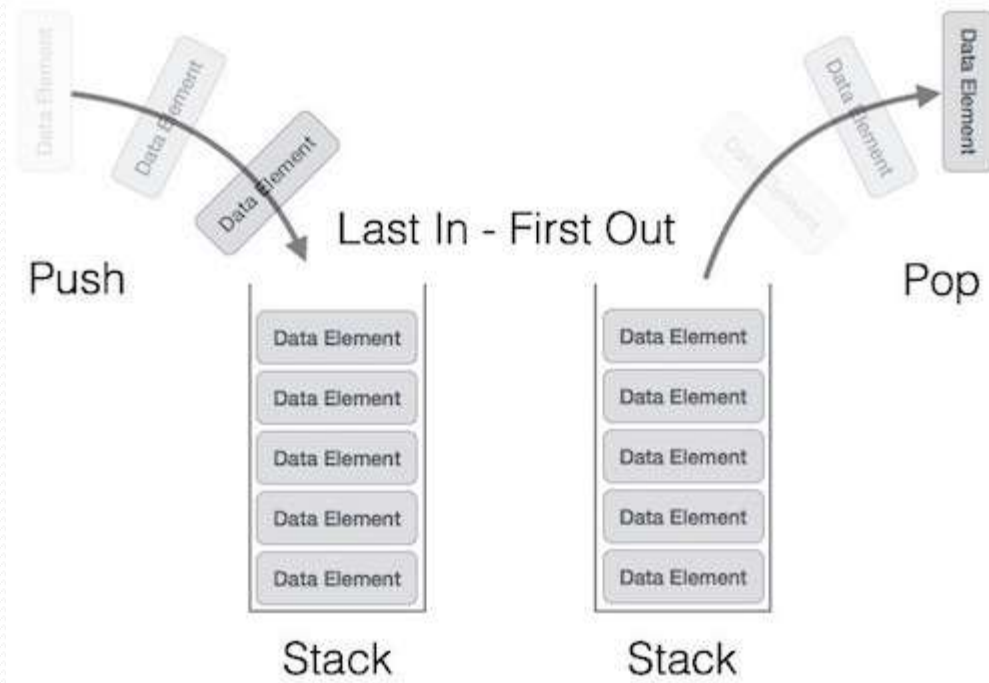
- ❑ The plate which is at the top is the first one to be removed,
- ❑ The plate which has been placed at the bottommost position remains in the stack for the longest period of time.
- ❑ So, it can be simply seen to follow **LIFO(Last In First Out)/FILO(First In Last Out) order**.





# Stacks

- Stack is a linear data structure which follows a particular order in which the operations are performed.
- The order may be LIFO (Last In First Out) or FILO (First In Last Out).



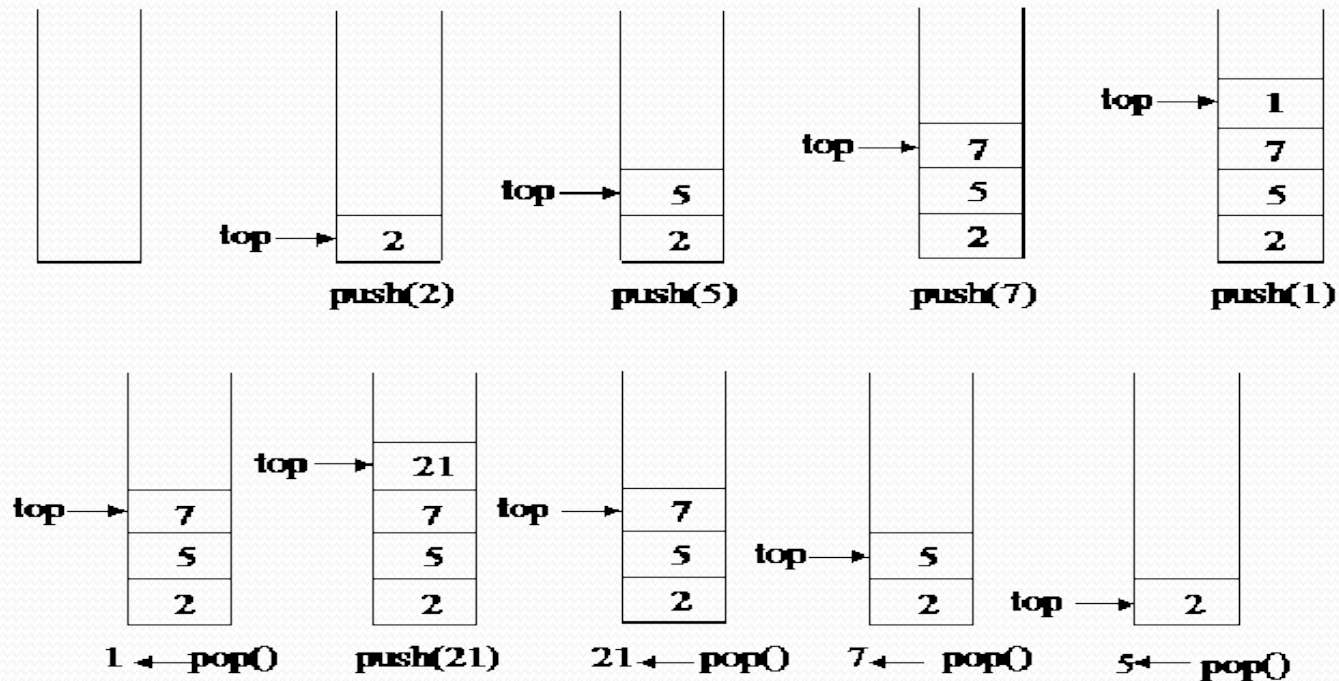
# Stacks

- A Stack is an ordered collection of elements , but it has a special feature that deletion and insertion of elements can be done only from one end, called the top of the stack(TOP).



# Operation on Stacks

- **Push operation**-The insertion of an element into stack
- **Pop operation** - Deletion of an element from the stack
- In stack we always keep track of the last element present in the list with a pointer called top.



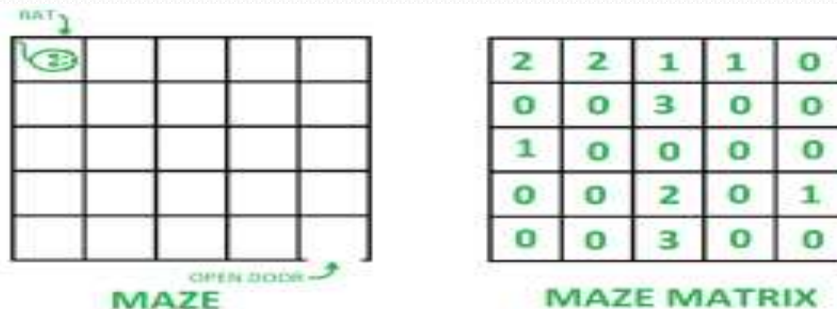
# Applications of stack:

- Expression evaluation and syntax parsing
- Balancing of symbols, paranthesis
- Infix to Postfix /Prefix conversion
- Reverse a String using Stack
- Redo-undo features at many places like editors, photoshop.
- Forward and backward feature in web browsers
- Used in many algorithms like Tower of Hanoi, tree traversals, stock span problem, histogram problem.

# Applications of stack:

- **Backtracking**

- Backtracking is an algorithmic-technique for
  - solving problems recursively by trying to build a solution incrementally, one piece at a time,
  - removing those solutions that fail to satisfy the constraints of the problem at any point of time
- Other applications can be Backtracking, Knight tour problem, rat in a maze, N queen problem and sudoku solver



- **Backtracking**-Finding the correct path in a maze.
  - There are a series of points, from the starting point to the destination.
  - We start from one point. To reach the final destination, there are several paths.
  - Suppose we choose a random path. After following a certain path, we realise that the path we have chosen is wrong. So we need to find a way by which we can return to the beginning of that path.

- **Backtracking**-Finding the correct path in a maze.
  - This can be done with the use of stacks. With the help of stacks, we remember the point where we have reached.
  - This is done by pushing that point into the stack.
  - In case we end up on the wrong path, we can pop the top point from the stack and thus return to the last point and continue our quest to find the right path.

# Applications of stack:

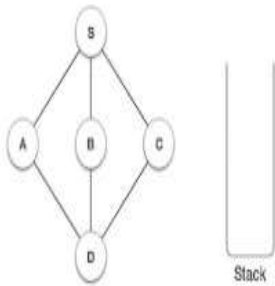
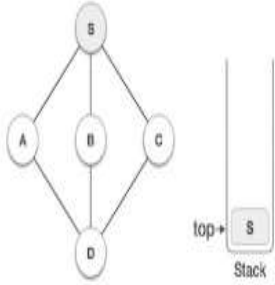
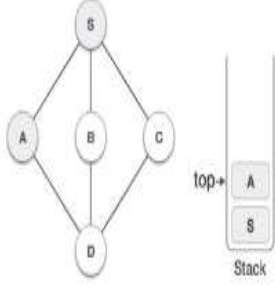
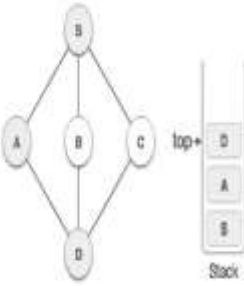
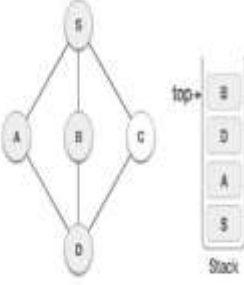
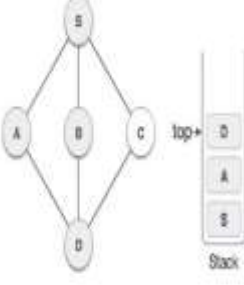
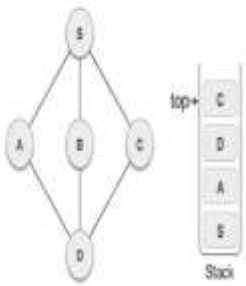
- **Depth-first search**
  - The prototypical example of a backtracking algorithm is depth-first search, which finds all vertices of a graph that can be reached from a specified starting vertex.
- In Graph Algorithms like Topological Sorting and Strongly Connected Components



# Depth-first search using Backtracking

- **Rule 1** – Visit the adjacent unvisited vertex. Mark it as visited. Display it. Push it in a stack.
- **Rule 2** – If no adjacent vertex is found, pop up a vertex from the stack. (It will pop up all the vertices from the stack, which do not have adjacent vertices.)
- **Rule 3** – Repeat Rule 1 and Rule 2 until the stack is empty.

# Depth-first search using Backtracking

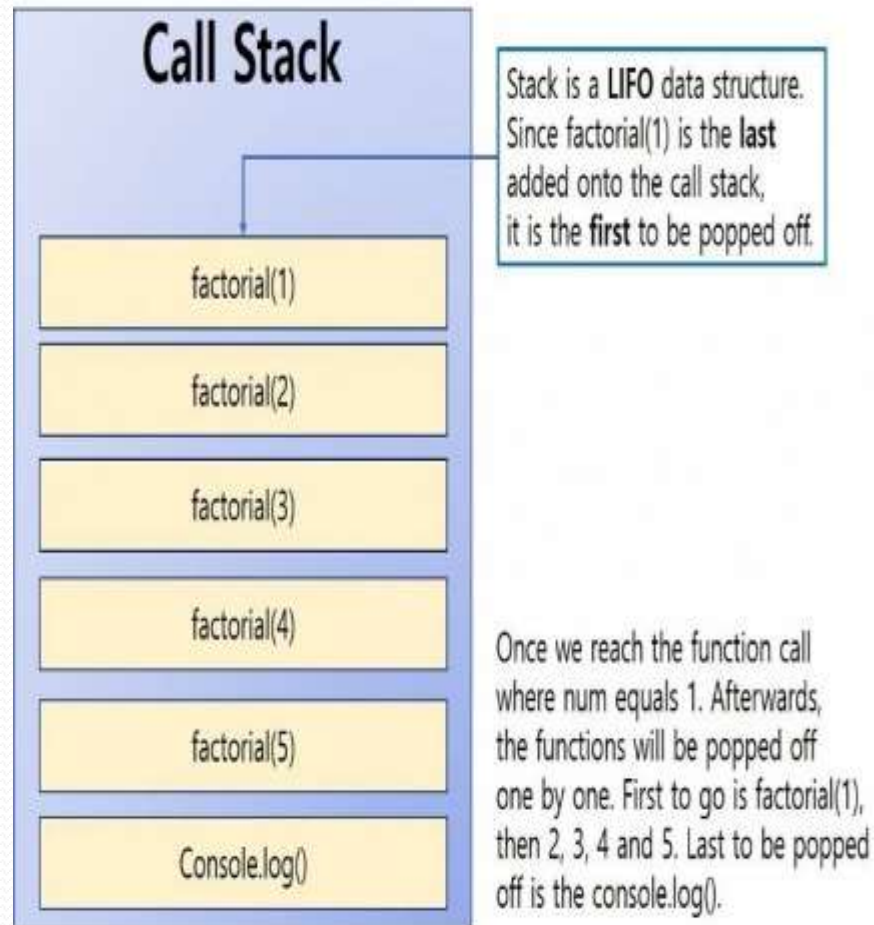
Step	Traversal	Description
1		Initialize the stack.
2		Mark S as visited and put it onto the stack. Explore any unvisited adjacent node from S. We have three nodes and we can pick any of them. For this example, we shall take the node in an alphabetical order.
3		Mark A as visited and put it onto the stack. Explore any unvisited adjacent node from A. Both S and D are adjacent to A but we are concerned for unvisited nodes only.
4		Visit D and mark it as visited and put onto the stack. Here, we have B and C nodes, which are adjacent to D and both are unvisited. However, we shall again choose in an alphabetical order.
5		We choose B, mark it as visited and put onto the stack. Here B does not have any unvisited adjacent node. So, we pop B from the stack.
6		We check the stack top for return to the previous node and check if it has any unvisited nodes. Here, we find D to be on the top of the stack.
7		Only unvisited adjacent node is from D is C now. So we visit C, mark it as visited and put it onto the stack.

As C does not have any unvisited adjacent node so we keep popping the stack until we find a node that has an unvisited adjacent node. In this case, there's none and we keep popping until the stack is empty.

# Applications of stack:

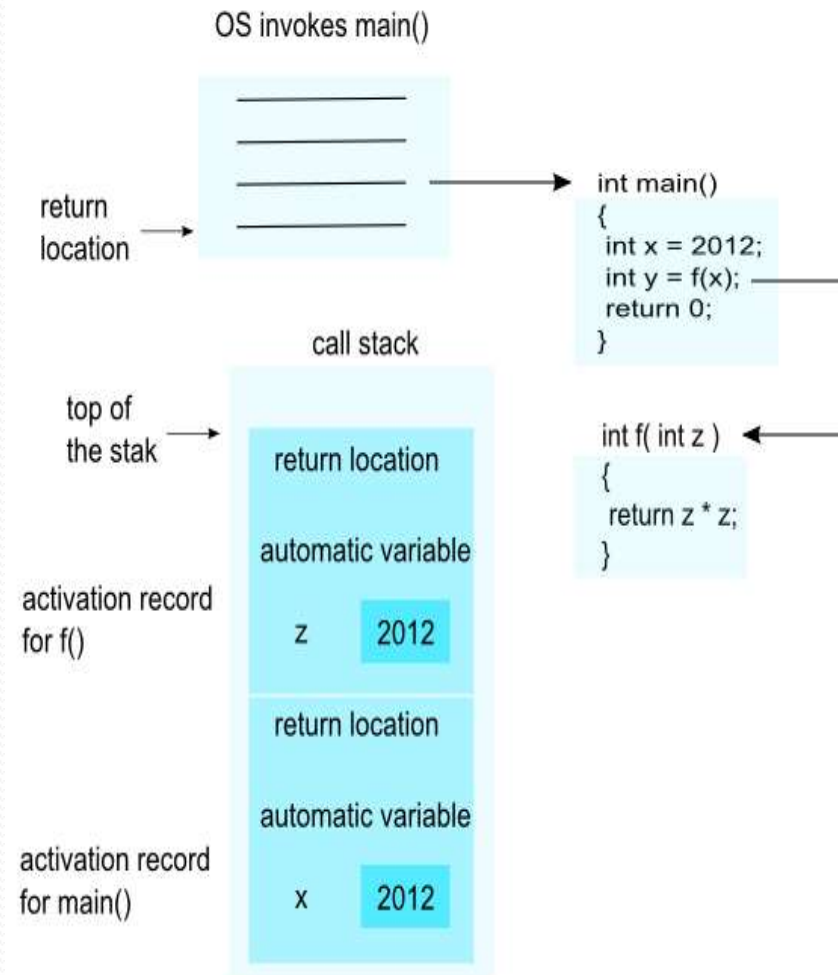
- **Compile time memory management**
- Almost all calling conventions

# Function calls using stack



- **Compile time memory management**
  - A number of programming languages are stack-oriented,
    - meaning they define most basic operations (adding two numbers, printing a character) as taking their **arguments from the stack**, and placing any **return values back on the stack**.

# Function calls using stack



- Almost all calling conventions—
  - the ways in which subroutines receive their parameters and return results—use a special stack (the "call stack") to hold information about procedure/function calling and nesting in order to switch to the context of the called function and restore to the caller function when the calling finishes.
- **Stacks are an important way of supporting nested or recursive function calls.**

# Applications of stack:

- **Efficient algorithms**

- The nearest-neighbor chain algorithm,
- Graham scan, an algorithm for the convex hull of a two-dimensional system of points. A convex hull of a subset of the input is maintained in a stack, which is used to find and remove concavities in the boundary when a new point is added to the hull

- **Security**

- Some computing environments use stacks in ways that may make them vulnerable to security breaches and attacks.
- Programmers working in such environments must take special care to avoid the pitfalls of these implementations.

## **Data Structures | Stack | Question 2**

Which one of the following is an application of Stack Data Structure?

- (A) Managing function calls
- (B) The stock span problem
- (C) Arithmetic expression evaluation
- (D) All of the above

## **Data Structures | Stack | Question 2**

Which one of the following is an application of Stack Data Structure?

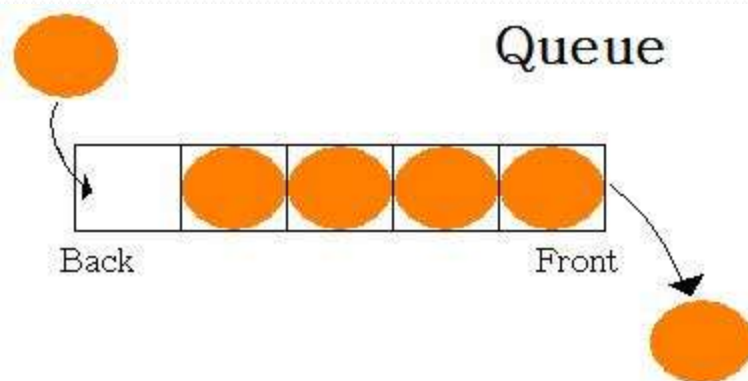
- (A) Managing function calls
- (B) The stock span problem
- (C) Arithmetic expression evaluation
- (D) All of the above

**Answer: (D)**



# Queues

- A Queue is a linear structure which follows a particular order in which the operations are performed.
- The order is FIFO, Queues are first in first out type of data structures( FIFO).
- In a queue new elements are added to the queue from one end called Rear end and the elements are always removed from the other end called the Front end.

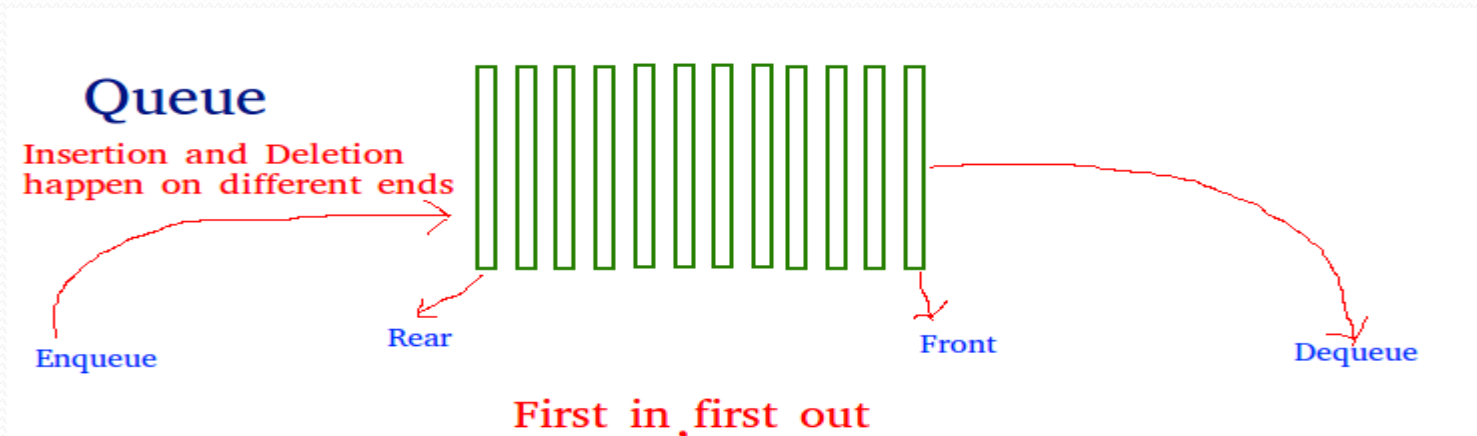


# Real life Examples of Queue

- The people standing in a railway reservation row are an example of queue. Each new person comes and stands at the end of the row (rear end of queue) and persons getting their reservations confirmed , get out of the row from the front end.
- A good example of a queue is any queue of consumers for a resource where the consumer that came first is served first.

# Operations on Queue:

- **Enqueue:**
  - Adds an item to the queue
  - Enter or Insert an item from Rear end
- **Dequeue:**
  - Removes an item from the queue.
  - Service or Delete From the **Front** end



# Applications of Queue:

- When things don't have to be processed immediately, but have to be processed in First In First Out order like **Breadth First Search**.
- When a resource is shared among multiple consumers. Examples include **CPU scheduling, Disk Scheduling**.
- When data is transferred asynchronously (data not necessarily received at same rate as sent) between two processes. Examples include **IO Buffers, pipes, file IO, etc.**

# Difference between Stacks and Queues

- The difference between stacks and queues is in removing.
  - In a stack, we remove the item which was most recently added;
  - In a queue, we remove the item which was least recently added.

# Difference between Stacks and Queues

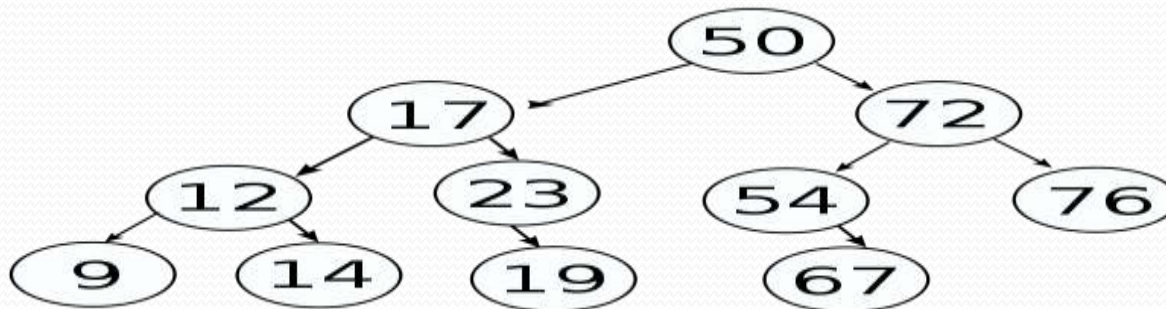
STACKS	QUEUES
Stacks are based on the LIFO principle, i.e., the element inserted at the last, is the first element to come out of the list.	Queues are based on the FIFO principle, i.e., the element inserted at the first, is the first element to come out of the list.
Insertion and deletion in stacks takes place only from one end of the list called the top.	Insertion and deletion in queues takes place from the opposite ends of the list. The insertion takes place at the rear of the list and the deletion takes place from the front of the list.
Insert operation is called push operation.	Insert operation is called enqueue operation.
Delete operation is called pop operation.	Delete operation is called dequeue operation.
In stacks we maintain only one pointer to access the list, called the top, which always points to the last element present in the list.	In queues we maintain two pointers to access the list. The front pointer always points to the first element inserted in the list and is still present, and the rear pointer always points to the last inserted element.
Stack is used in solving problems works on recursion.	Queue is used in solving problems having sequential processing.

# Non-Linear Data Structures

- A data structure is said to be non-linear if traversal of nodes is nonlinear in nature.
- All one-many, many-one or many-many relations are handled through non-linear data structures.
- Every data element can have a number of predecessors as well as successors.
- Trees, graphs and tables are its examples.

# Trees

- A tree is a non- linear data structure used to represent hierarchical relationship existing among several data items.
- It is a finite set of one or more data items such that-
  - There is a special data item called the root of the tree.
  - Its remaining data items are partitioned into number of mutually exclusive subsets, each of which is itself a tree, and they are called subtrees.





# Applications of Tree Data Structure

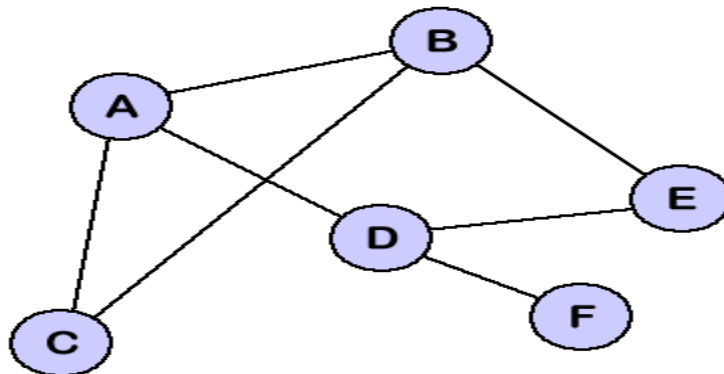
- Store hierarchical data, like folder structure, organization structure, XML/HTML data.
- Binary Search Tree is a tree that allows fast search, insert, delete on a sorted data. It also allows finding closest item
- Heap is a tree data structure which is implemented using arrays and used to implement priority queues.

# Applications of Tree Data Structure

- B-Tree and B+ Tree : They are used to implement indexing in databases.
- Syntax Tree: Used in Compilers.
- Trie : Used to implement dictionaries with prefix lookup.
- Spanning Trees and shortest path trees are used in routers and bridges respectively in computer networks

# Graphs

- Graph is a non-linear data structure capable of representing many kinds of physical data structures.
- It has found its application in divers fields like Geography, Chemistry and Engineering Sciences.
- A graph  $G(V,E)$  is a set of vertices  $V$  and a set of edges  $E$ .
  - An edge connects a pair of vertices and many have weight such as length, cost etc.
  - Vertices are shown as point or circles and edges are drawn as arcs or line segment.
  - Thus an edge can be represented as  $E=(V,W)$  where  $V$  and  $W$  are pair of vertices.



# Applications of Graph Data Structure

- Google maps
- Facebook
- World Wide Web
- Operating System
  
- ??

# Applications of Graph Data Structure

- In Computer science graphs are used to represent the flow of computation.
- Google maps –
  - uses graphs for building transportation systems,
  - where intersection of two(or more) roads are considered to be a vertex and
  - the road connecting two vertices is considered to be an edge,
  - thus their navigation system is based on the algorithm to calculate the shortest path between two vertices.

# Applications of Graph Data Structure

- In Facebook,
  - users are considered to be the vertices and
  - if they are friends then there is an edge running between them.
  - Facebook's Friend suggestion algorithm uses graph theory.
  - Facebook is an example of undirected graph.

# Applications of Graph Data Structure

- In World Wide Web,
  - web pages are considered to be the vertices.
  - There is an edge from a page  $u$  to other page  $v$  if there is a link of page  $v$  on page  $u$ .
  - This is an example of Directed graph.
  - It was the basic idea behind Google Page Ranking Algorithm.

# Applications of Graph Data Structure

- In Operating System,
  - we come across the Resource Allocation Graph where
    - each process and resources are considered to be vertices.
  - Edges are drawn from
    - resources to the allocated process, or
    - from requesting process to the requested resource.
  - If this leads to any formation of a cycle then a deadlock will occur.