D Y PATIL
DEEMED TO BE
UNIVERSITY
RAMRAO ADIK
INSTITUTE OF TECHNOLOGY
NAVI MUMBAI

Data Strucure

Unit 1: Introduction to Data Structures

Faculty Name : Mrs. Kausar Fakir

Index -

Lecture 1 – Introduction to Data Structure, Concept of ADT	3
Lecture 2 – Types of Data Structure, Operation on Data Structure	20

Lecture 1

Introduction, Classification of Data Structures, Concept of ADT

Module wise weightage

Course Code	Course Name	Teaching Scheme (Contact Hours 45)			Credits Assigned	
		Theory	Practical	Tutorial	Theory	Total
2413FYC2T1	DATA STRUCTURE (DS)	03	--	--	03	03

Theory Evaluation Scheme										
Internal Assessment (50 Marks)								End Sem. Exam Marks	End Sem. Time Duration	Total
CA1	CA2	CA3	best of 2	Assignment / Tutorial	Quiz / Seminar	Open Book Test / Surprise Test / Capstone Project	Total			
10	10	10	20	10	10	10	50	100	03 Hrs	150

Data

Structure

- Defines how the object or thing is kept and organized
- Which structure is better

Structure

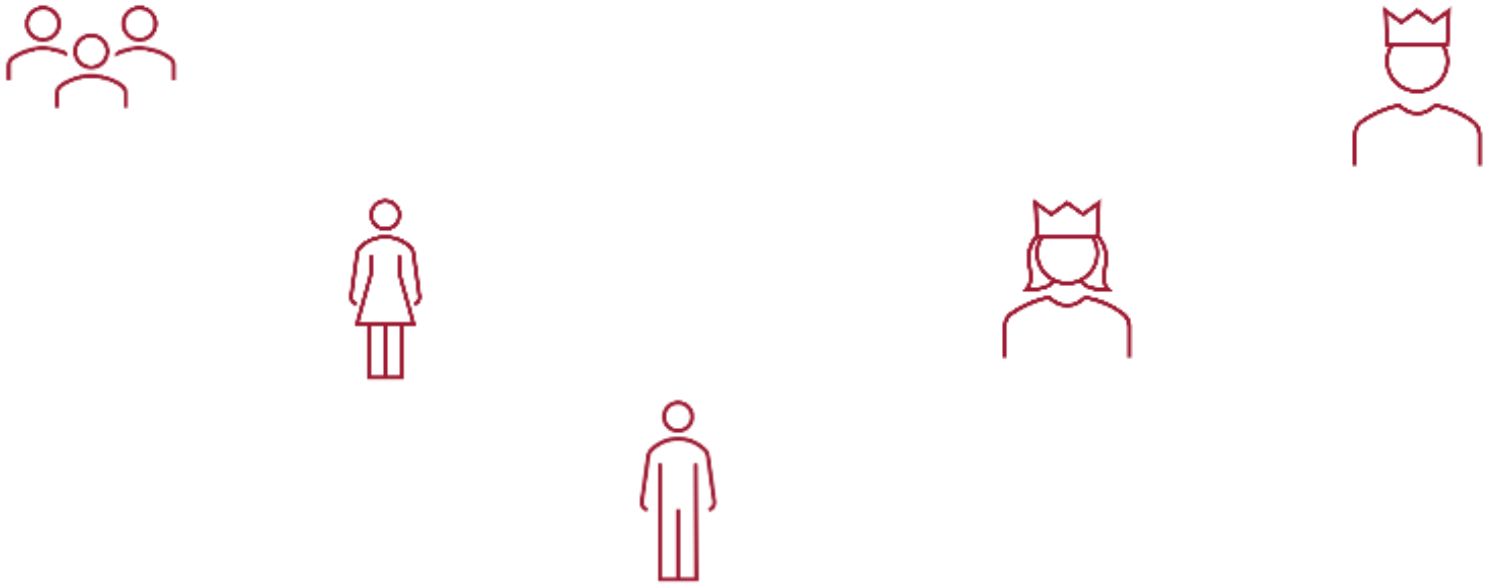
- Defines how the object or thing is kept and organized
- Which structure is better for keeping books?

Structure

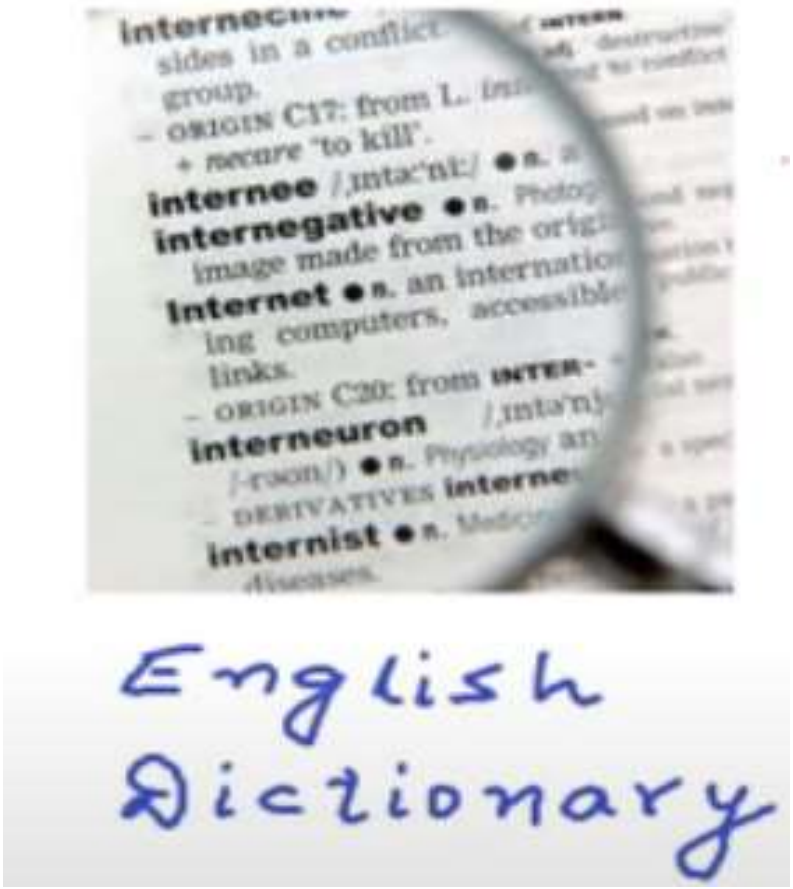
Which configuration of floor is better?

Choice of Structure

- As we have seen examples
- The choice of structure is user specific



Need of Data Structure



ABC Hardware
Cash Book - 03/01/2013 to 03/31/2013

S. no.	Date	Particulars	Debit	Credit
1	03/01/2013	Opening balance		50000
2	03/02/2013	Transport bill	2000	
3	03/07/2013	Goods sales		1500
4	03/08/2013	Bank Loan		5000
5	03/15/2013	Goods sales		1000
6	03/17/2013	Electiricty bill	1200	
7	03/21/2013	Good sales		1200
8	03/25/2013	Hardware purchase	500	
9	03/29/2013	Employee salary	20000	
10	03/31/2013	Closing Balance	35000	

Need of Data Structure

Introduction to Data Structure

- We have seen the structure, it defines the way object/things are:
  - Stored
  - Organised
- Data structures: Data structures is a way **to store and organize data**, so that it can be used efficiently
- Selecting proper structure, makes data to be used and processed efficiently
- Also the choice of data structure is application specific

Data Abstraction

ADT : Abstract Data Type

Let's break ADT:

- **Data type:**
  - A data type represents a set of possible values
    - such as {..., -2, -1, 0, 1, 2, ...}, or {true, false}
    - are categorization of **data** in every programming language
    - Examples: integer, float, character etc.
- **Abstract (Abstraction)** : The process of providing only essentials and hiding the details is known as abstraction.

Concept of ADT

- Abstract:

To *abstract* is to leave out information, keeping (hopefully) the more important parts. The end-user is not concerned about the details of how the methods carry out their tasks. They are only aware of the methods that are available to them and are only concerned about calling those methods and getting the results. They are not concerned about how they work.

- An Abstract Data Type (ADT) is:
  - a set of *values*
  - a set of *operations,* which can be applied uniformly to all these values

Concept of ADT

Example:

Seat Reservation System

Data

- Number of seats
- Seats reserved
- Seats available

Set of operations

- Book a seat
- Cancel a seat
- Find available seat

Concept of ADT

Example:

Library Management System

Data

- Number of books
- Book title, author, year....
- Book is available
- Issue date and return date.....

Set of operations

- Issue a book
- Return a book
- Pay fine
- Add a new book
- Remove any book.....

Concept of ADT

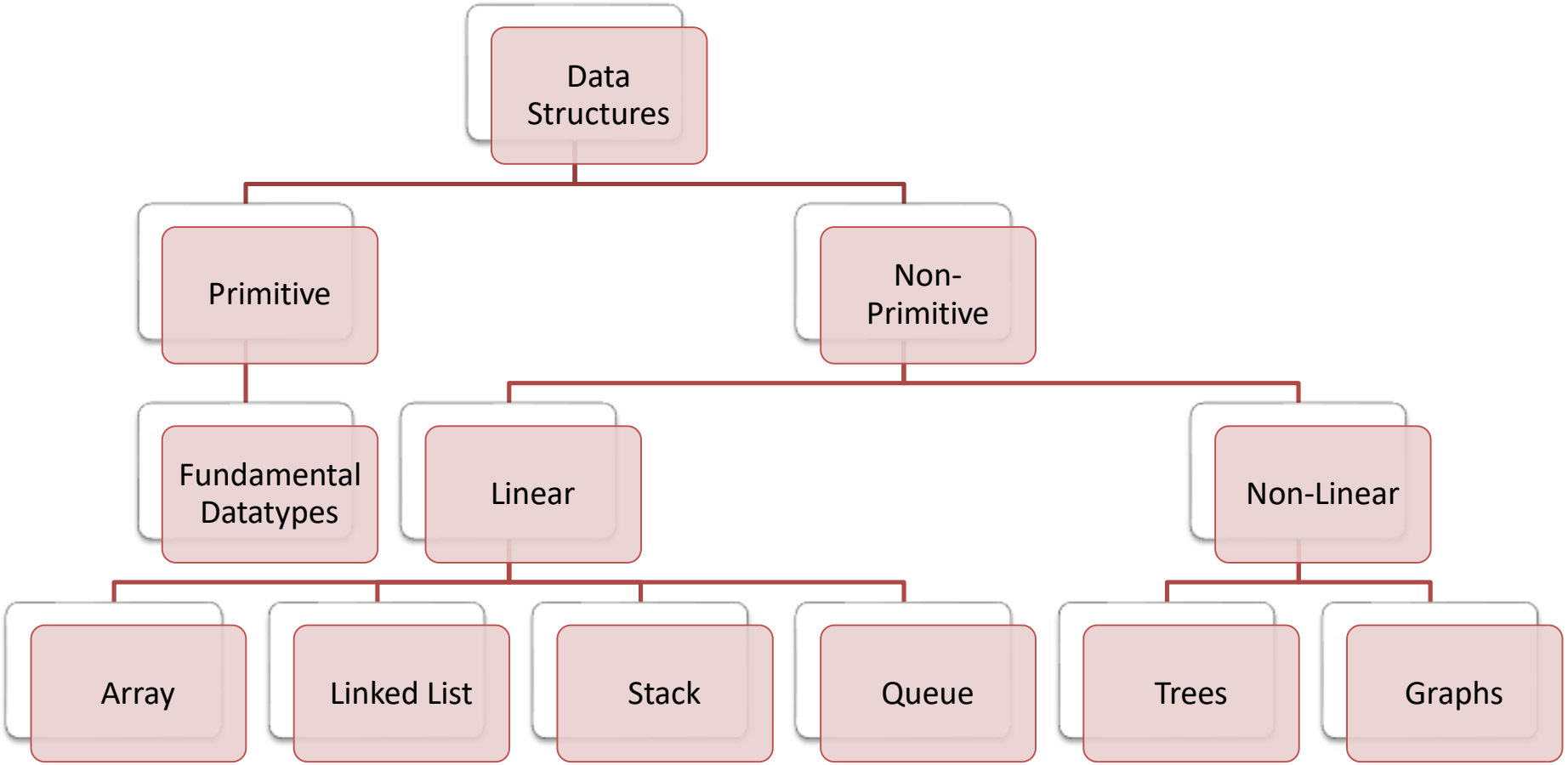
- ADT enables us to specify the general operations w.r.t to data structures, we use as per application requirements.
- In short enable us to emphasize what need to be done, rather than how it needs to be done.

So now lets move to type of data structures, then we see how ADT helps us define operation with out bothering about type of data we store.

Classification of Data Structure

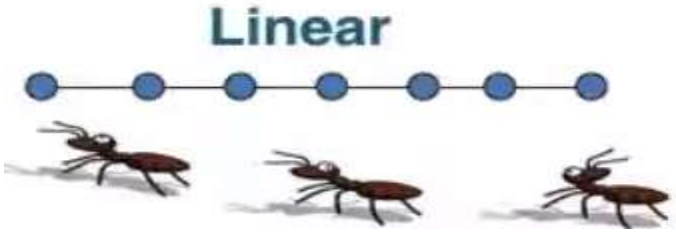
- Depending on the way, the elements in data structures mentioned previously can be classified into :
  - **Primitive Data Structures**: Fundamentals datatype supported by programming language e.g. int , float etc
  - **Non-primitive Data Structures**: Data structures created using combining primitive data structures. These can be further classified as:
    - Linear
    - Non-linear

Classification of Data Structure

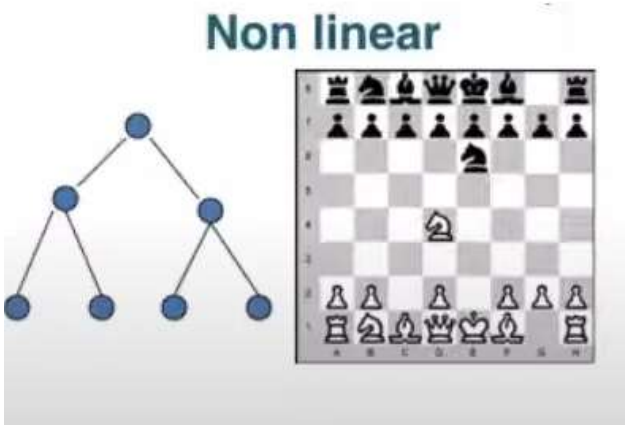


Classification of Data Structure

- **Linear:** Elements of the data structure forms linear relationship or sequential order.



- **Non-linear :** Elements of this data structure are stored/accessed in a non-linear order

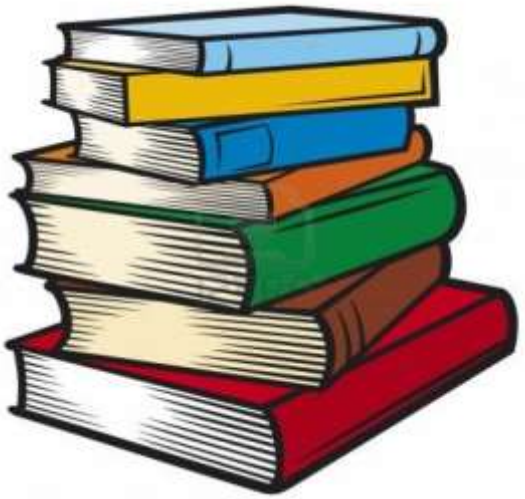


Classification of Data Structure

- Depending on memory allocated, data structures mentioned previously can be classified into :
  - Static Data Structures : memory is allocated at compile time
    - ✓ Fixed size
    - ✓ Ex: Array
  - Dynamic Data Structures: memory is allocated at execution time
    - ✓ Size can be changed
    - ✓ Ex: linked list

Example Application

- Stack: Last in First Out(LIFO)
- Ex: Tower of rings, Pile of dishes, Stack of Books, Undo operation, Web-browser Navigation

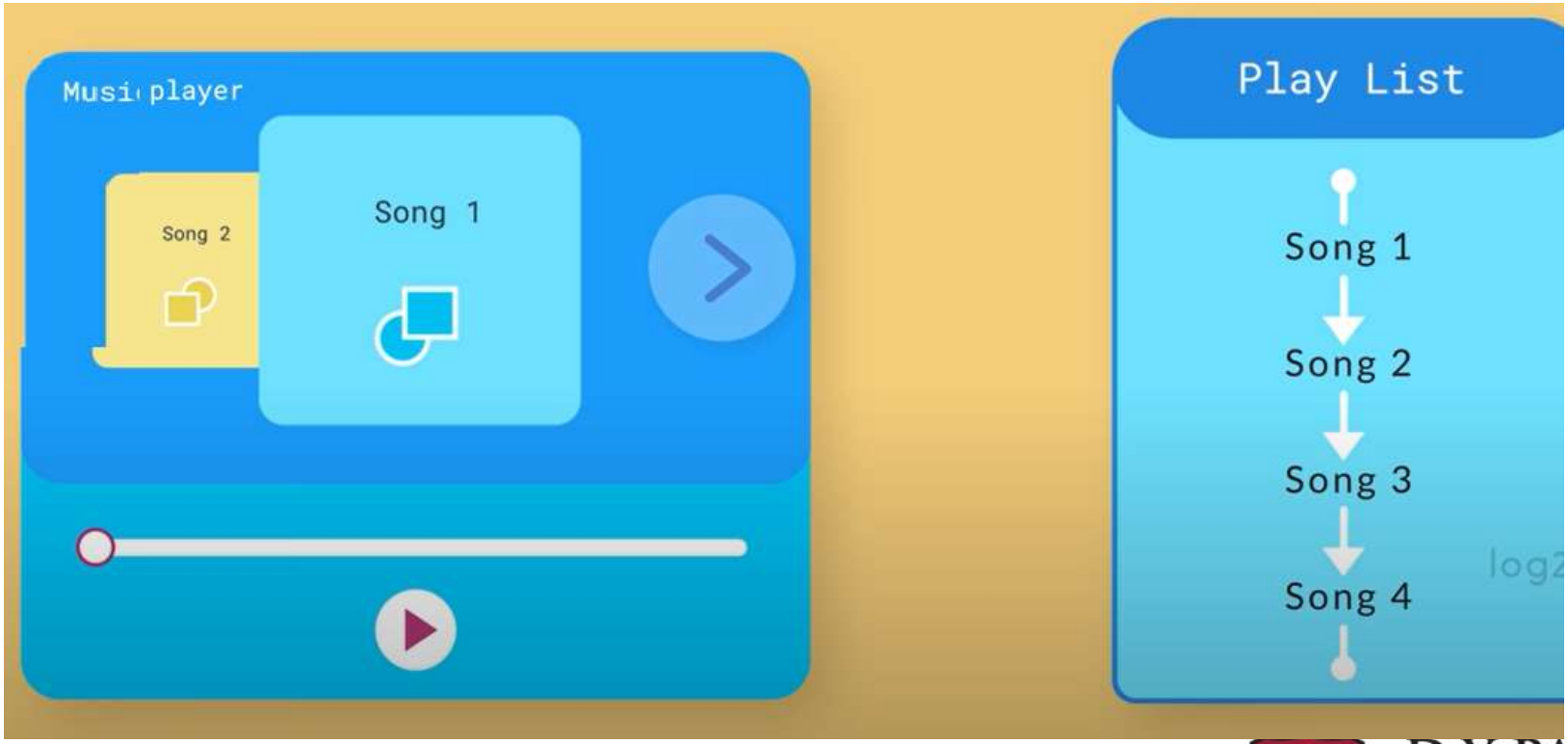


Example Application

- Queue: First In First Out(FIFO)

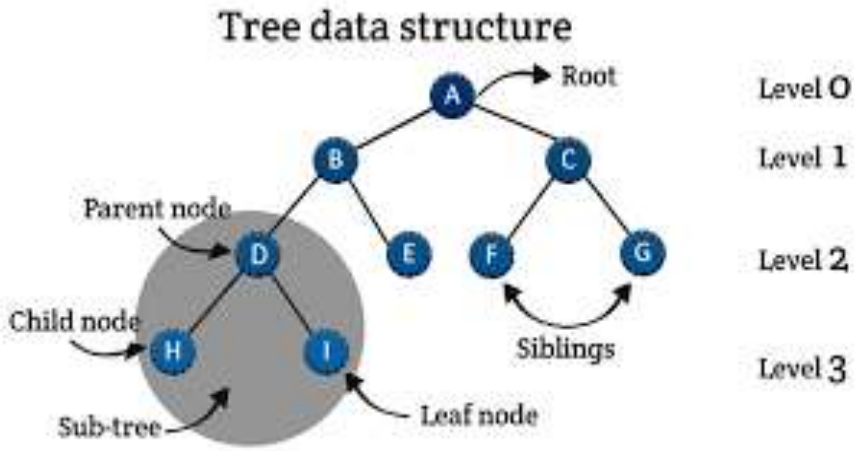
Linked List: Application

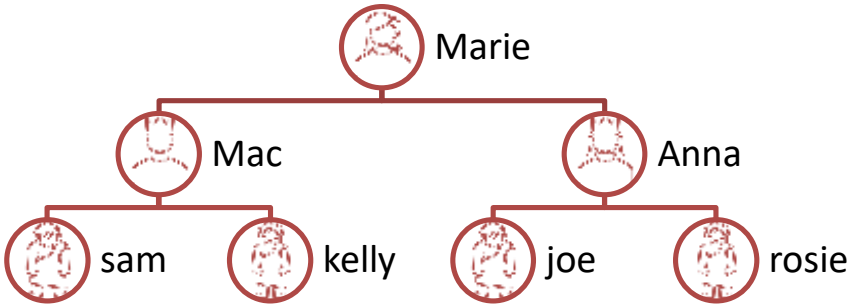
- **Playlist for Songs**

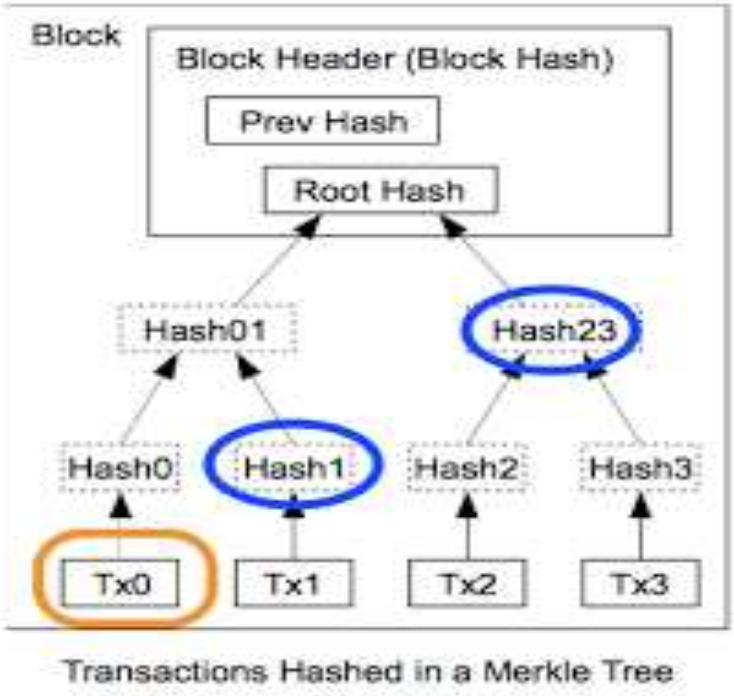


Example Application

- Trees:



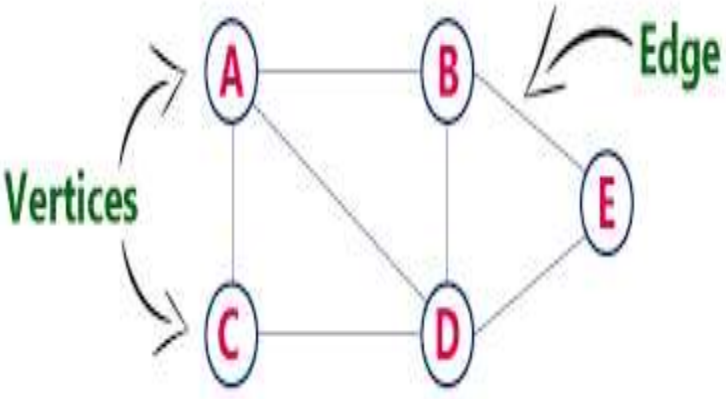


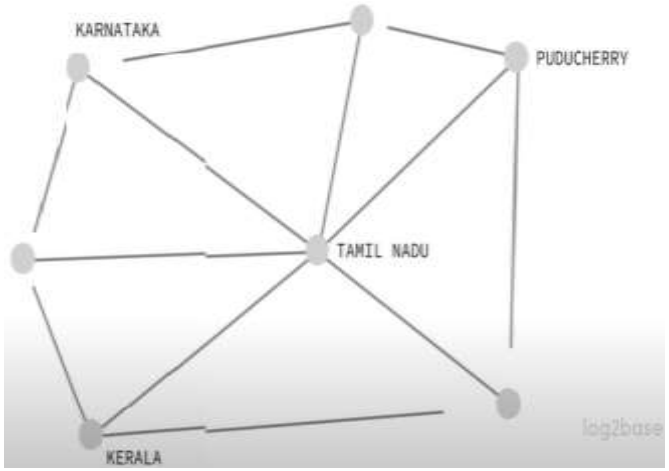


Transactions Hashed in a Merkle Tree

Example Application

- Graphs: Social Network Graph, Google Map





Operation on Data Structures

- Operation below can be performed on various data structures shown previously.
  - Traversing : It means to access each data item exactly once so that it can be processed
  - Insert : Add the data item.
  - Delete : Remove the data item
  - Search: locate a particular item, from given set of data
  - Sort : Sort the data on the given parameter
  - Merge : combine the data from different source/set.
- If you see, all these above operations are probably in most of applications we are using in day to day life for managing our information/data

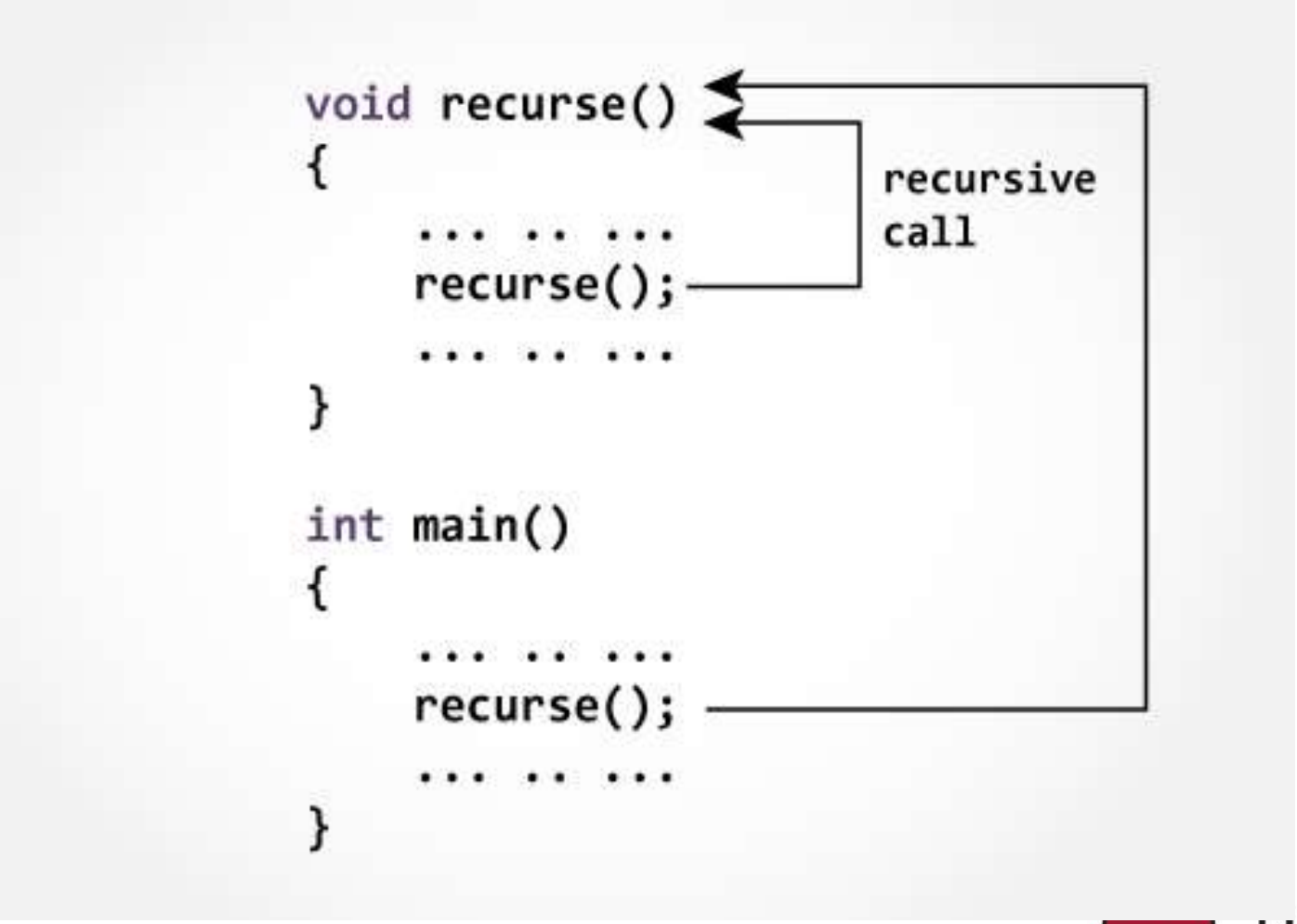
Unit 1- Lecture 2

Arrays and Recursion

Recursion

- A function that calls itself is known as a recursive function. And, this technique is known as recursion.
- For a problem to be written in recursive form, two conditions are to be satisfied:
  - It should be possible to express the problem in recursive form
    - Solution of the problem in terms of solution of the same problem on smaller sized data
  - The problem statement must include a stopping condition to break out of the recursion, otherwise recursion will occur infinite times.

How recursion works?



Need of Recursion

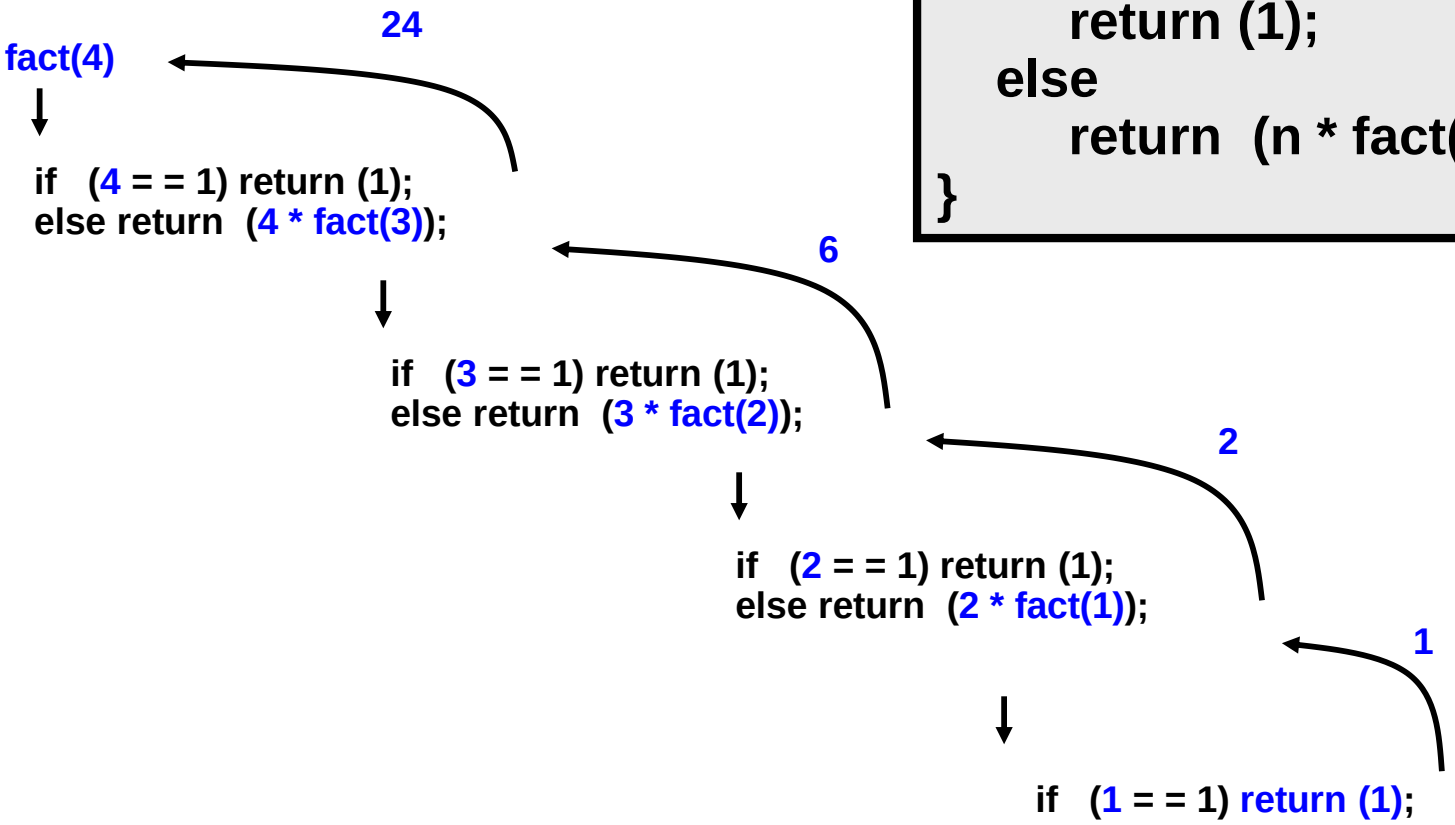
- Recursion is an amazing technique with the help of which we can reduce the length of our code and make it easier to read and write.
- A task that can be defined with its similar subtask, recursion is one of the best solutions for it.
- For example; The Factorial of a number.

Properties of Recursion

- Performing the same operations multiple times with different inputs.
- In every step, we try smaller inputs to make the problem smaller.
- Base condition is needed to stop the recursion otherwise infinite loop will occur.

Execution of Recursion :

```
long  int  fact (int n)
{
   if   (n == 1)
        return (1);
   else
        return  (n * fact(n-1));
}
```



Example: Factorial of Number Using Recursion

```
#include <stdio.h>
#include<conio.h>
int fact(int n);
void main( )   {
    int number;
long int result;
    printf("Enter a positive integer: ");
    scanf("%d", &number);
    result = fact(number);
    printf("Factorial of %d = %d", number,
    result);
    getch();    }
```

```
int  fact (int n)
 {
    if(n == 1)
       return (1);
    else
      return  (n * fact(n-1));
}
```

Output :

Enter a positive integer:4

Factorial of 4 = 24

Program 2: Calculate n^m using Recursion.

```
#include <stdio.h>
int cal_power(int base, int power);
void main( )
{
    int base, power, result;
    printf("Enter a base value: ");
    scanf("%d", &base);
    printf("Enter a power value: ");
    scanf("%d", &power);
    result = cal_power(base, power);
    printf("%d raise to %d = %d",
    base, power,result);        }
```

```
int cal_power(int n, int m)
{
    if(m == 0)
        return (1);
    else if(m == 1)
        return (n);
    else
        return  (n *
     cal_power(n, m-1));
```

Output :

```
Enter a base value: 2
Enter a power value: 3
2 raise to 3 = 8
```

Program 3: Calculate Fibonacci series using Recursion.

```
#include <stdio.h>
int fibonacci(int);
int main(void)
{
    int n, terms;
    printf("Enter no. of terms: ");
    scanf("%d", &terms);
    for(n = 0; n <terms; n++)
      {
        printf("%d\t", fibonacci(n));      }   }
```

```
int fibonacci(int num)
{
    if(num == 0 || num == 1)
        return num;
    else
        return fibonacci(num-1) + fibonacci(num-2);    }
```

Output :

Enter no. of terms: 5

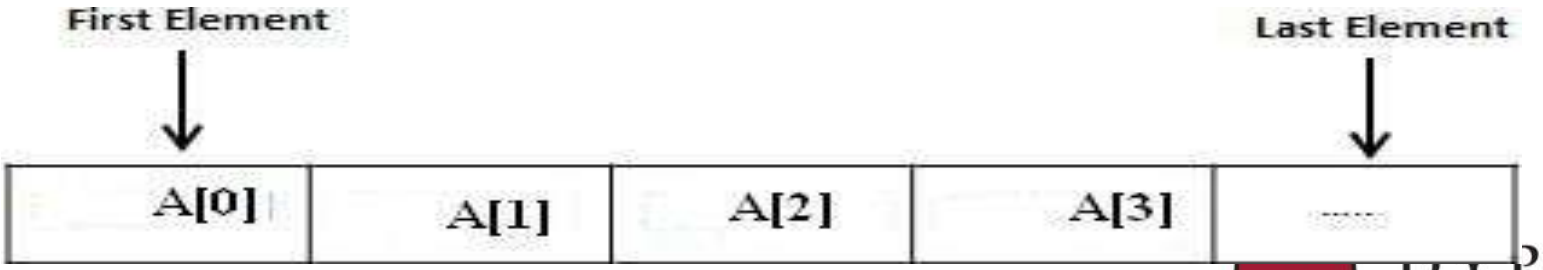
0 1 1 2 3

Recursion Demo

- https://visualgo.net/en/recursion?slide=2

Array Introduction

- An array is a linear data structure. Which is a finite collection of similar data items stored in successive or consecutive memory locations.
- For example an array may contains all integer or character elements, but not both.
- Each array can be accessed by using array index and it is must be positive integer value enclosed in square braces.
- Array index starts with 0 and ends with size-1.



Array Types

- Array can be categorized into different types. They are
  - One dimensional array
  - Two dimensional array
  - Multi dimensional array

Array Declaration

- One dimensional array is also called as linear array. It is also represents 1-D array.
- The one dimensional array stores the data elements in a single row or column.
- **Syntax:**

  datatype arrayName [arraySize];
- The **arraySize** must be an integer constant greater than zero.

- Eg:- double balance[10];
  int A[5];
  char Name[10];

Initializing Arrays

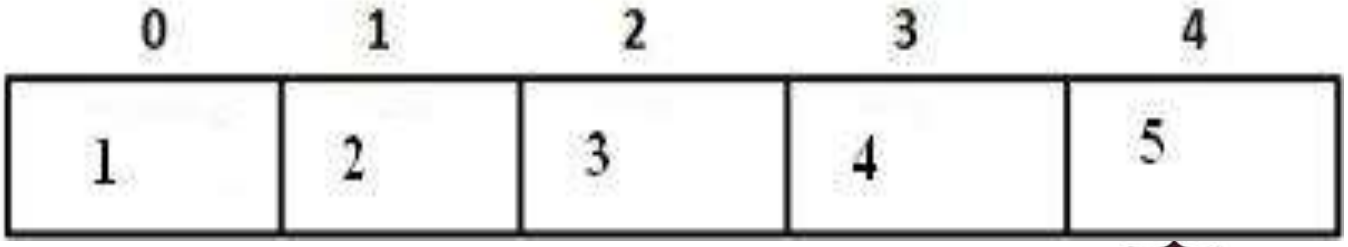
- You can initialize array either one by one or using a single statement as follows:

  double balance[5] = {1000.0, 2.0, 3.4, 17.0, 50.0};

  int A[5]={1,2,3,4,5};

- If you omit the size of the array, an array just big enough to hold the initialization is created. Therefore, if you write:

  int A[]={1,2,3,4,5};



2-D Array

- A two dimensional array is a collection of elements placed in rows and columns.
- **Declaration Syntax:**

  <data type> <array name> [row size] [column size];

- **Initialization Syntax:**

  <data type> <array name>[row size] [column size]={values};

2-Dimensional array

int A[3][3];

COL

ROW	0	1	2
0	A[0][0]	A[0][1]	A[0][2]
1	A[1][0]	A[1][1]	A[1][2]
2	A[2][0]	A[2][1]	A[2][2]

Memory Representation of 2-D array

A[0][0]	A[0][1]	A[0][2]	A[1][0]	A[1][1]	A[1][2]	A[2][0]	A[2][1]	A[2][2]
ROW 0			ROW 1			ROW 2		

2-D Array initialization

```
int A[][]={{1,2,3},
           {4,5,6},
           {7,8,9}};
```

Size of A is 3X3.

```
int B[][3]={{1,1,1},
            {2,2,2}};
```

Size of B is 2X3.

Note- size of row is optional

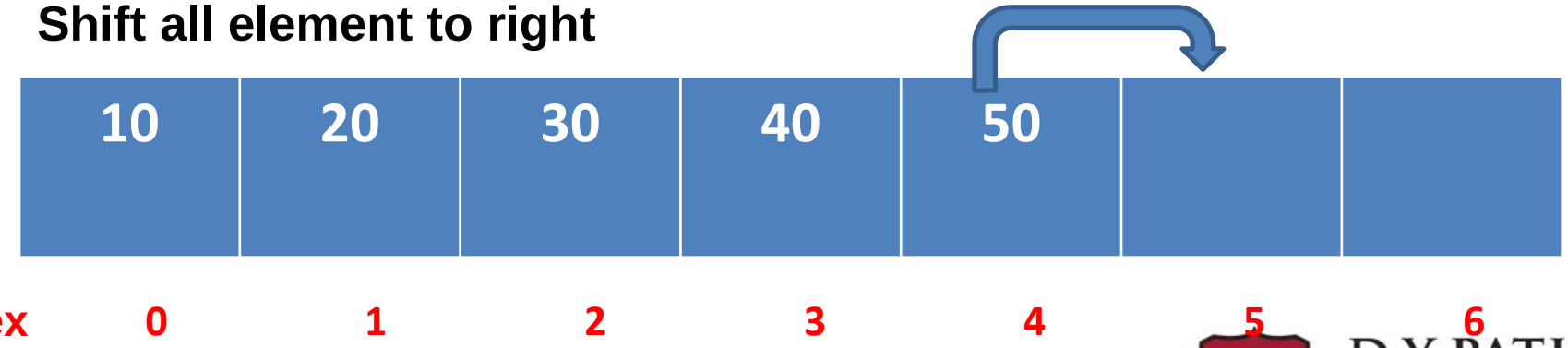
Array Operations

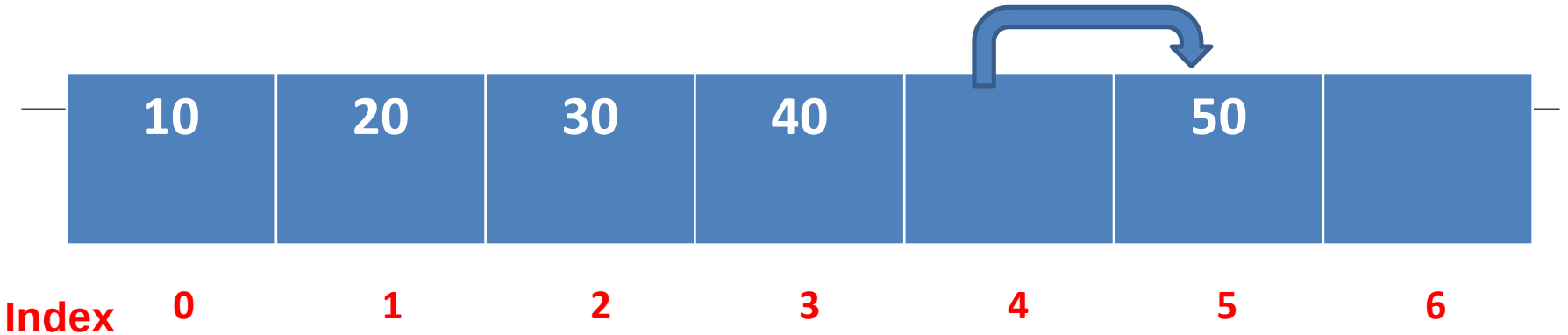
- There are several operations that can be performed on an array. They are
  - Insertion
  - Deletion
  - Traversal
  - Reversing
  - Sorting
  - Searching

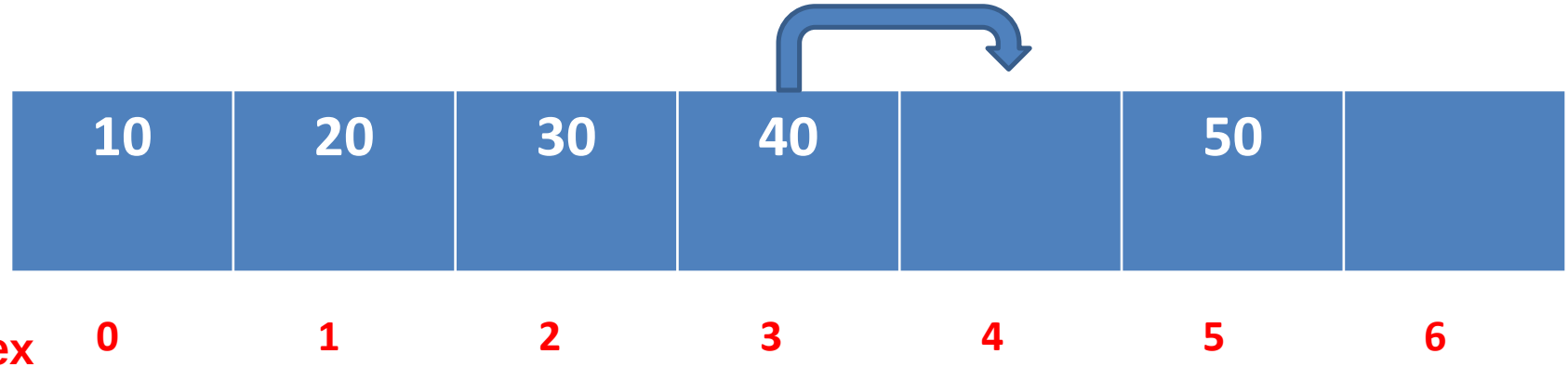
Insertion in array

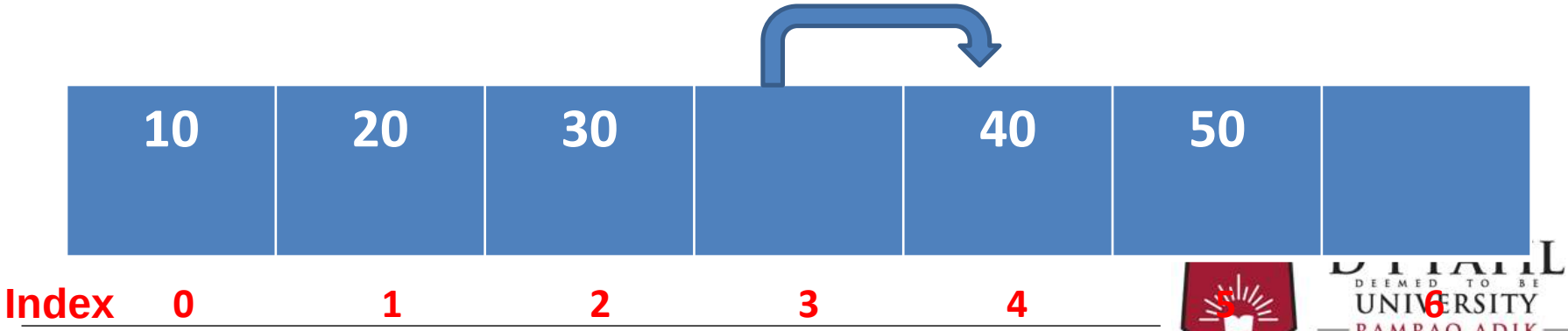
Consider Following Array

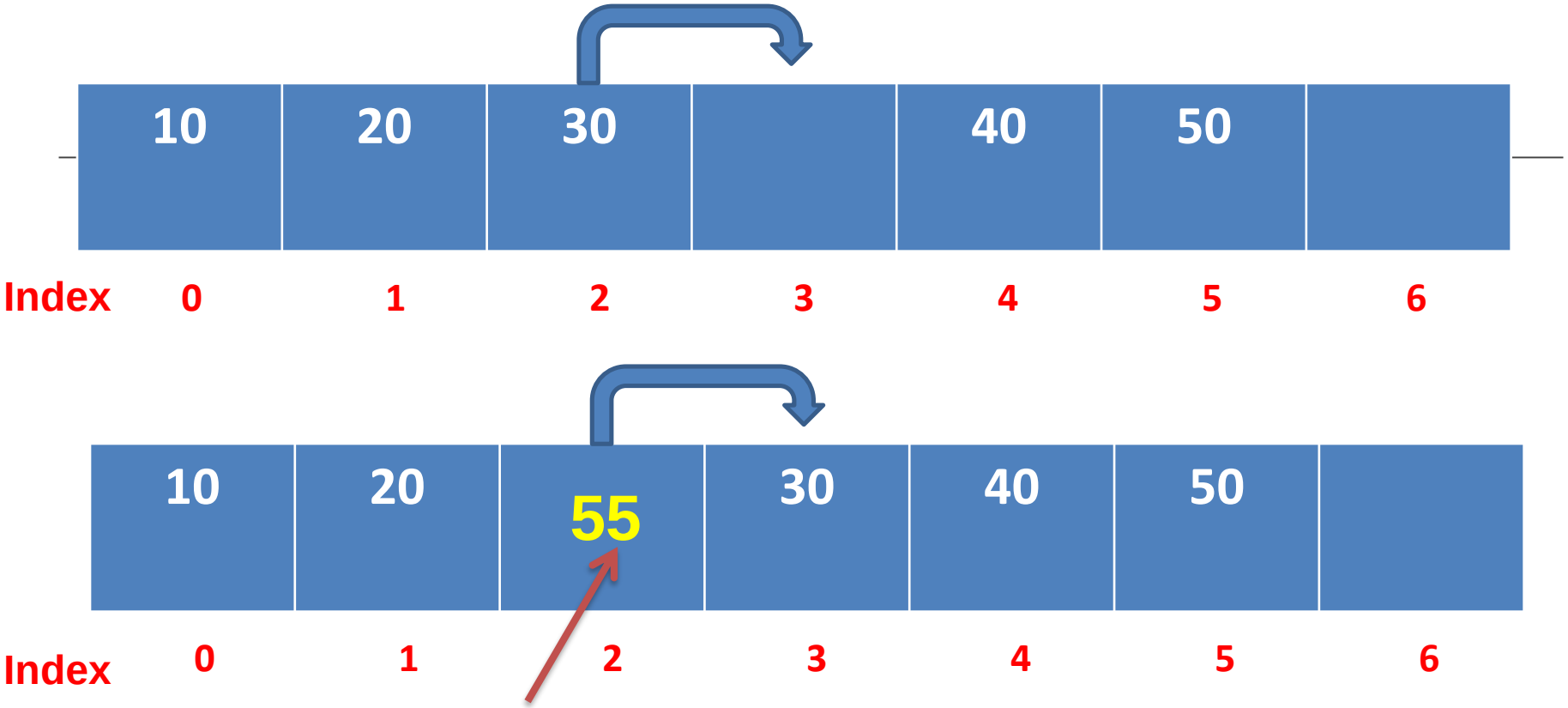
Insert New Element 55 at index =2
Shift all element to right









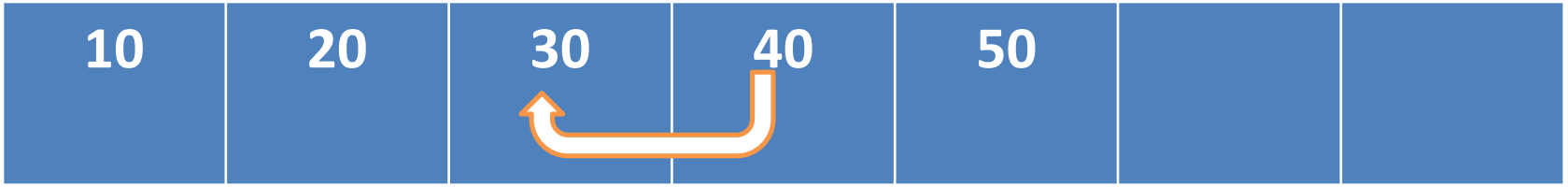


Deletion from array

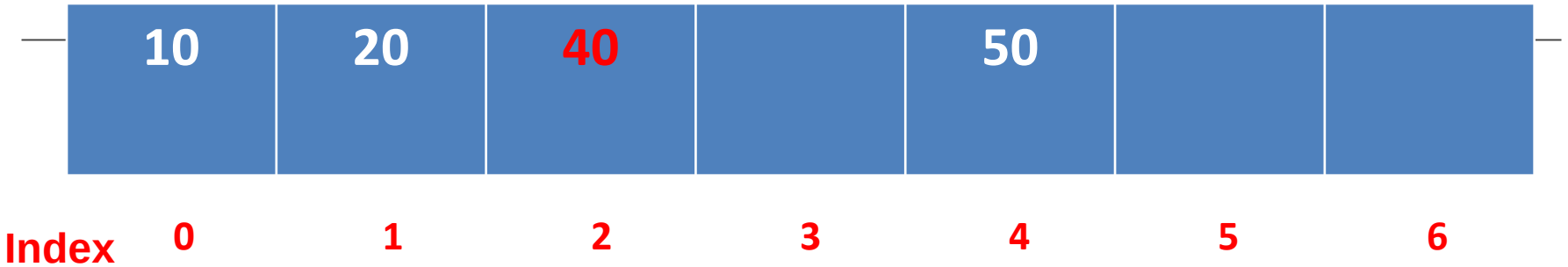
Consider Following Array

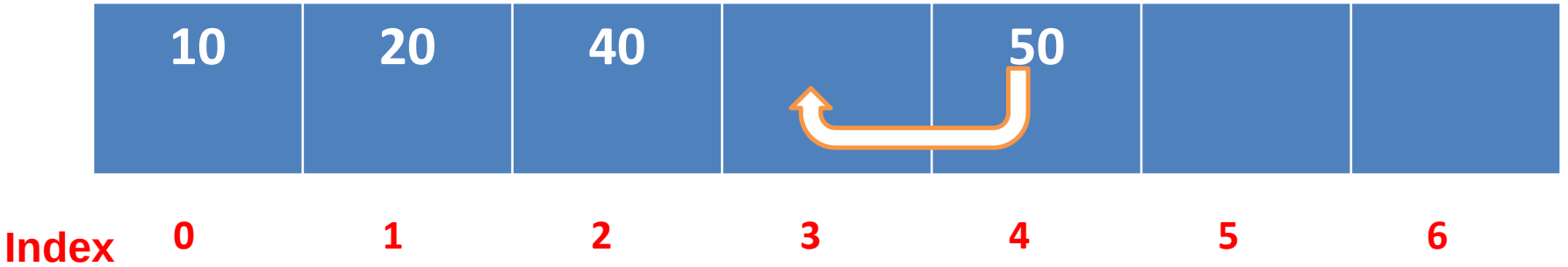
Index 0 1 2 3 4 5 6

Delete from index =2
Shift element to left

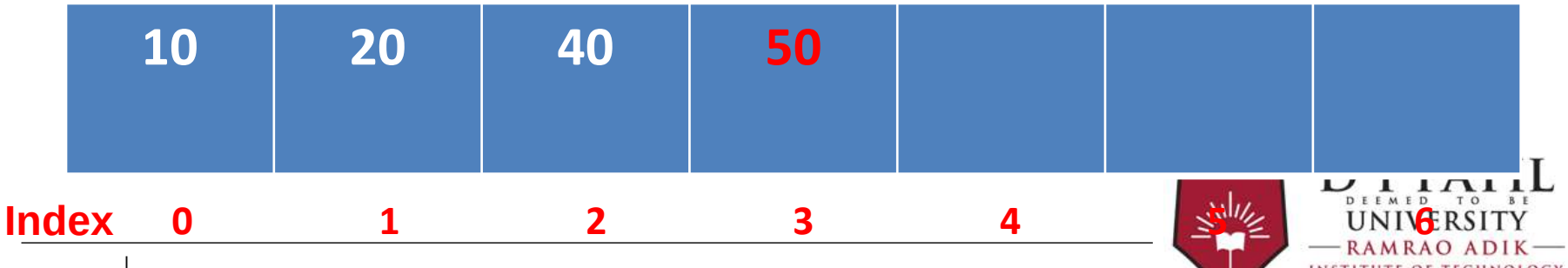


Index 0 1 2 3 4 5 6





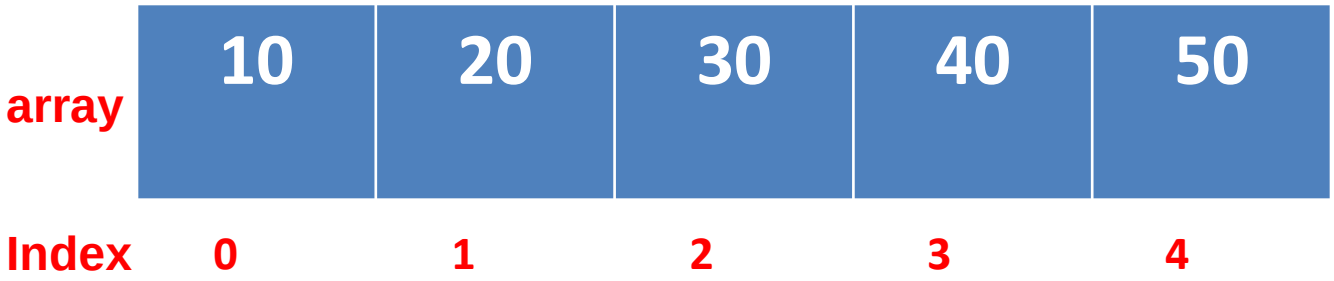
Element at index 2 is deleted

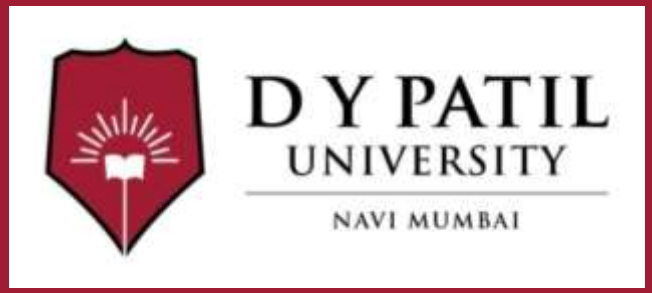


Traversal

- Traversal is nothing but display the elements in the array.
- Example:

```
for (i=0;i<5;i++)
{
printf ("%d\t", array[i]);
}
```





Subject Name: Data Structure

Unit No:2 Unit Name: Stack and Queues

Faculty Name : Mrs. Kausar Fakir

Index

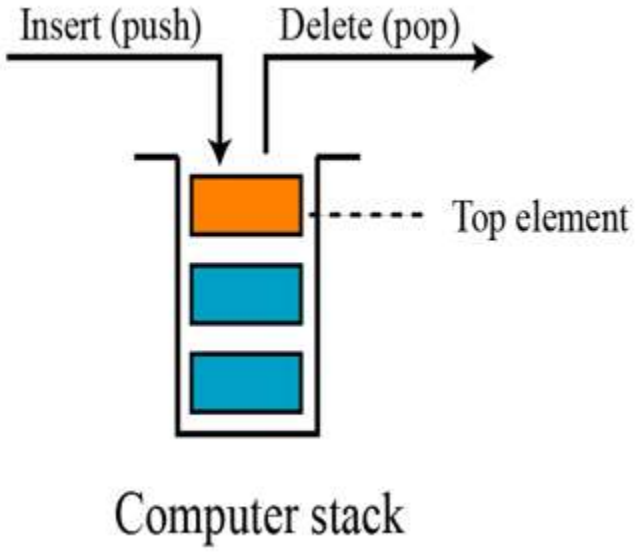
Lecture 3 – Introduction to Stack, Operations on Stack 24

Unit 2 - Lecture 3

Introduction to Stack, Operations on Stack

What is Stack ?

- Stack is an important data structure which stores its elements in an ordered manner.
- The last item placed on the stack will be the first item removed.
- **Only the top** of a stack is **accessible** and item must be inserted or removed from the top only
- A stack is also called **L**ast **I**n **F**irst **O**ut **(LIFO)** collection



Computer stack

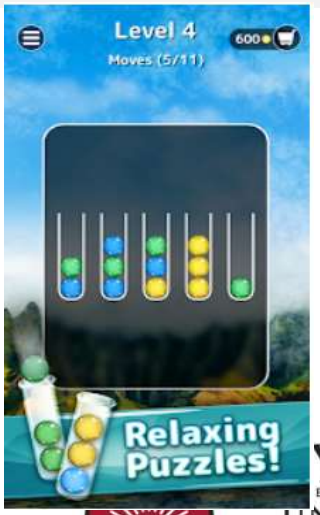
Introduction to Stack (Real World Example)

Stack of plates in cafeteria.

Pack of Tennis Ball

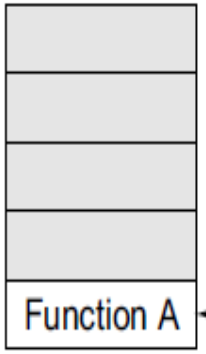
Mathematical puzzle called, Tower of Hanoi, where we have three rods or three pegs and multiple disks

Ballscapes: Ball Sort Puzzle & Color Sorting Games

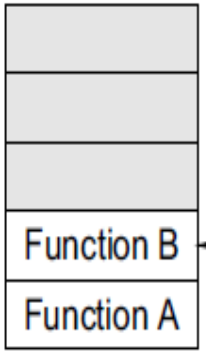


Need of Stack

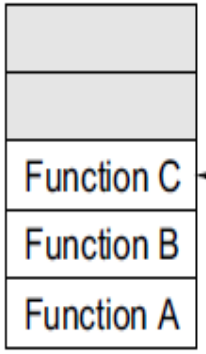
Consider an example, where we are executing function A. In the course of its execution, function A calls another function B. Function B in turn calls another function C, which calls function D.



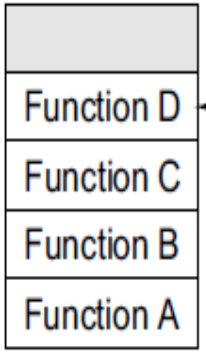
When A calls B, A is pushed on top of the system stack. When the execution of B is complete, the system control will remove A from the stack and continue with its execution.



When B calls C, B is pushed on top of the system stack. When the execution of C is complete, the system control will remove B from the stack and continue with its execution.



When C calls D, C is pushed on top of the system stack. When the execution of D is complete, the system control will remove C from the stack and continue with its execution.



When D calls E, D is pushed on top of the system stack. When the execution of E is complete, the system control will remove D from the stack and continue with its execution.

Introduction to Stack (Computer World Example)

Reversing a word : Think about how you would reverse a word using a stack

First all the letters are pushed into the stack and then popped out one by one to get the reversed word.

Undoing Changes in a Text Editor :

A stack is also commonly used in text editors. Changes that the user makes are pushed into a stack. While undoing, they are popped out.

ARRAY REPRESENTATION OF STACKS

- Stack is **last in, first out** data structure (LIFO).
- An element that is pushed last into a stack is the first to be popped out.
- Variable called **TOP** associated with it, which is used to store the address of the topmost element of the stack.
- It is this position where the element will be added to or deleted from.
- variable called **MAX**, which is used to store the maximum number of elements that the stack can hold.
- If TOP = NULL, then it indicates that the stack is empty and if TOP = MAX–1, then the stack is full.

Operations on Stack

- Stack is **last in, first out** data structure (LIFO).
- An element that is pushed last into a stack is the first to be popped out.
  - **Create Stack**
  - **Full**
  - **Empty**
  - **Push (Insert)**
  - **Pop (Delete)**
  - **Peek(Display)**

Create Stack

Create *stack* operation **creates an empty stack**. The following shows the format.

int stack[array size];

Empty Stack

Operation: Stack Empty

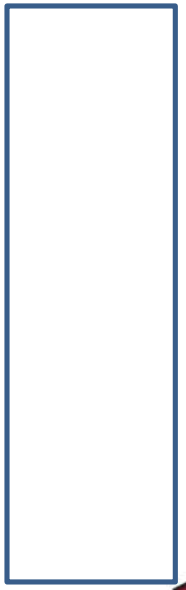
- The empty operation checks the status of the stack.
- This operation returns true if the stack is empty and false if the stack is not empty.

If TOP==-1, then

True

Else

False



Stack underflow condition

- The condition resulting from trying to pop an empty stack.

TOP =-1

Operation: Stack Full

- The full operation checks the status of the stack.
- This operation returns true if the stack is full and false if the stack is not full.

If TOP==Max-1, then

True

Else

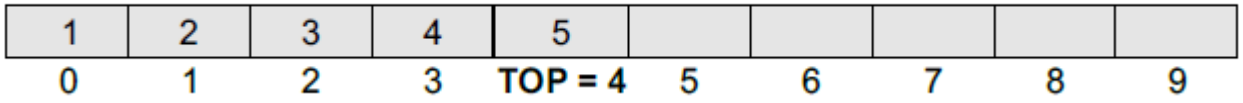
False

Stack overflow condition

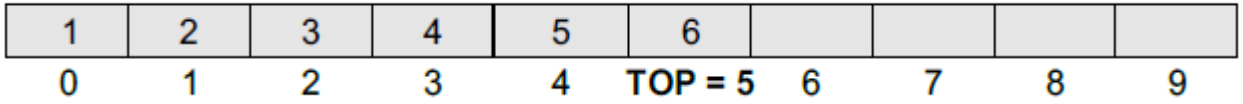
- The condition resulting from trying to push an element onto a full stack.

Operation: Pushing into a Stack

- The push operation is used to insert an element into the stack. The new element is added at the topmost position of the stack.
- However, before inserting the value, we must first check if TOP=MAX–1, because if that is the case, then the stack is full and no more insertions can be done.
- If an attempt is made to insert a value in a stack that is already full, an OVERFLOW message is printed.

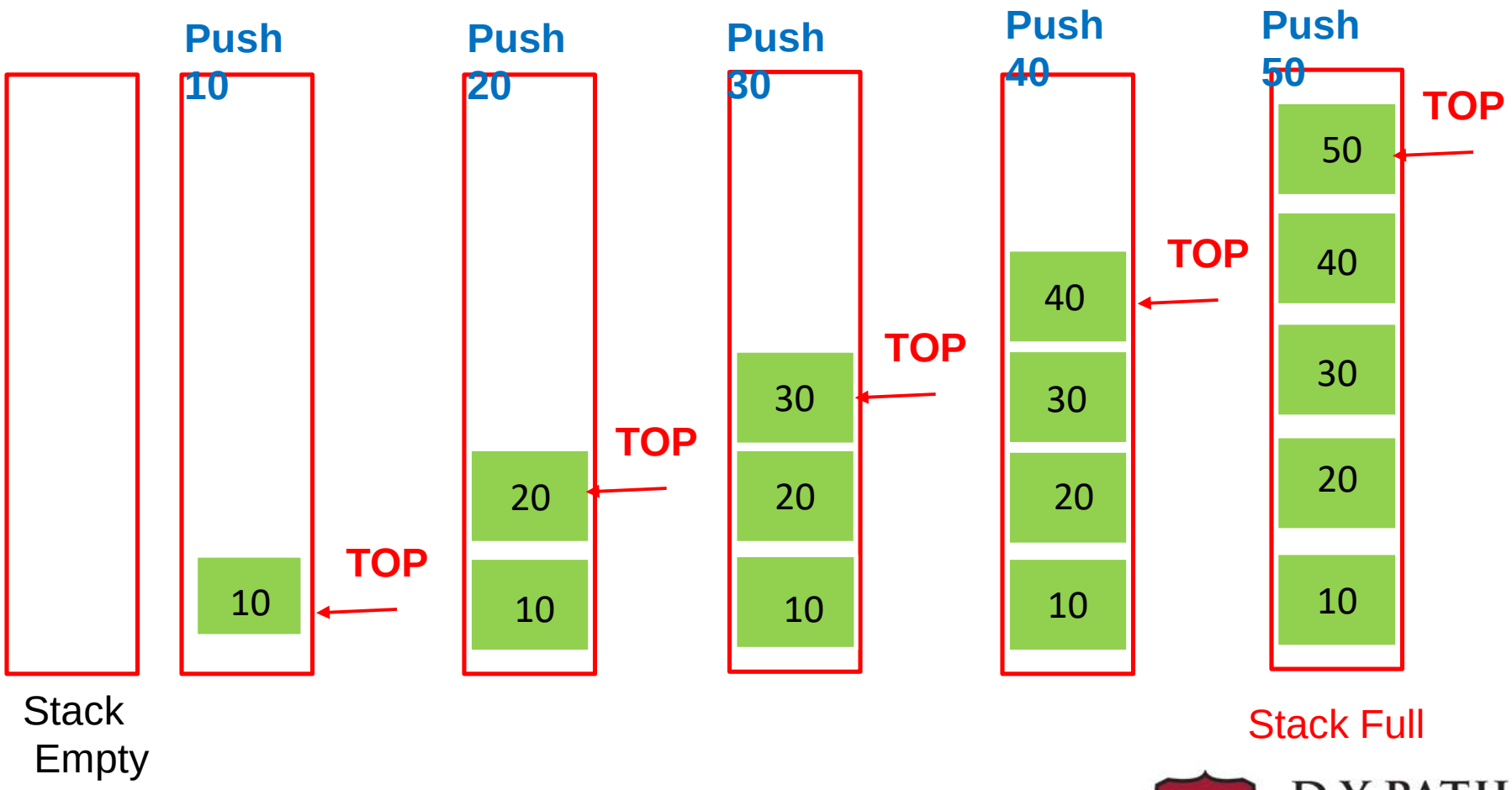


- To insert an element with value 6, we first check if TOP=MAX–1. If the condition is false, then we increment the value of TOP and store the new element at the position given by stack[TOP].

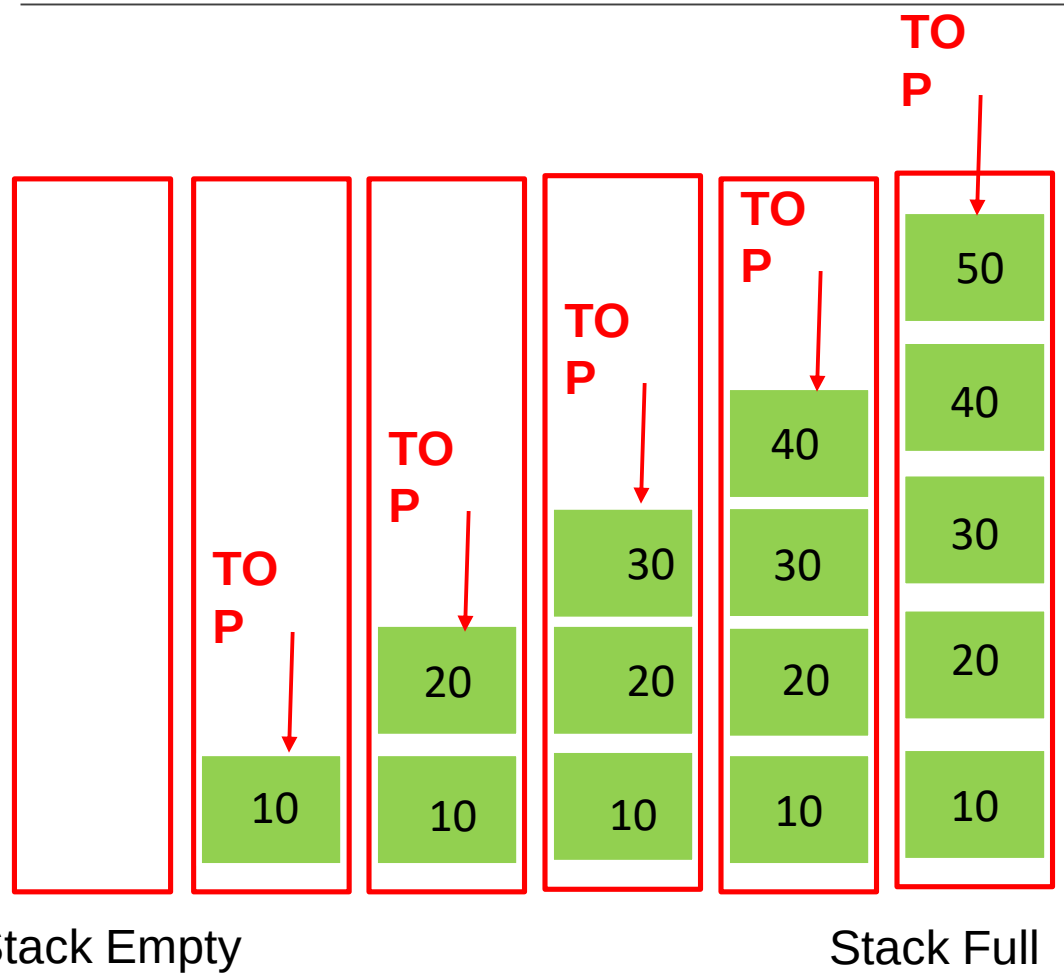


Operation: Pushing into a Stack

Push : 10, 20, 30, 40, 50



Operation: Stack Push



Step 1: If TOP=Max-1,then

print “overflow”

Goto Step 4

Step 2: Set TOP=TOP+1

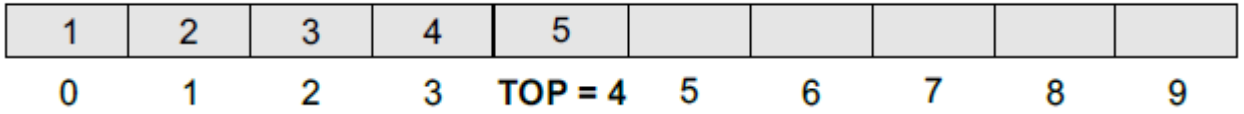
Step 3: Set

Stack[TOP]=value

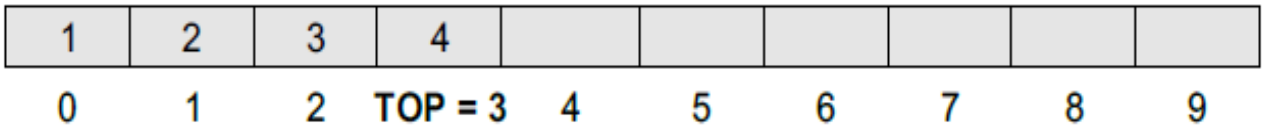
Step 4:End

Operation: Popping from Stack

- The pop operation is used to delete the topmost element from the stack. However, before deleting the value, we must first check if TOP=NULL because if that is the case, then it means the stack is empty and no more deletions can be done.
- If an attempt is made to delete a value from a stack that is already empty, an UNDERFLOW message is printed.

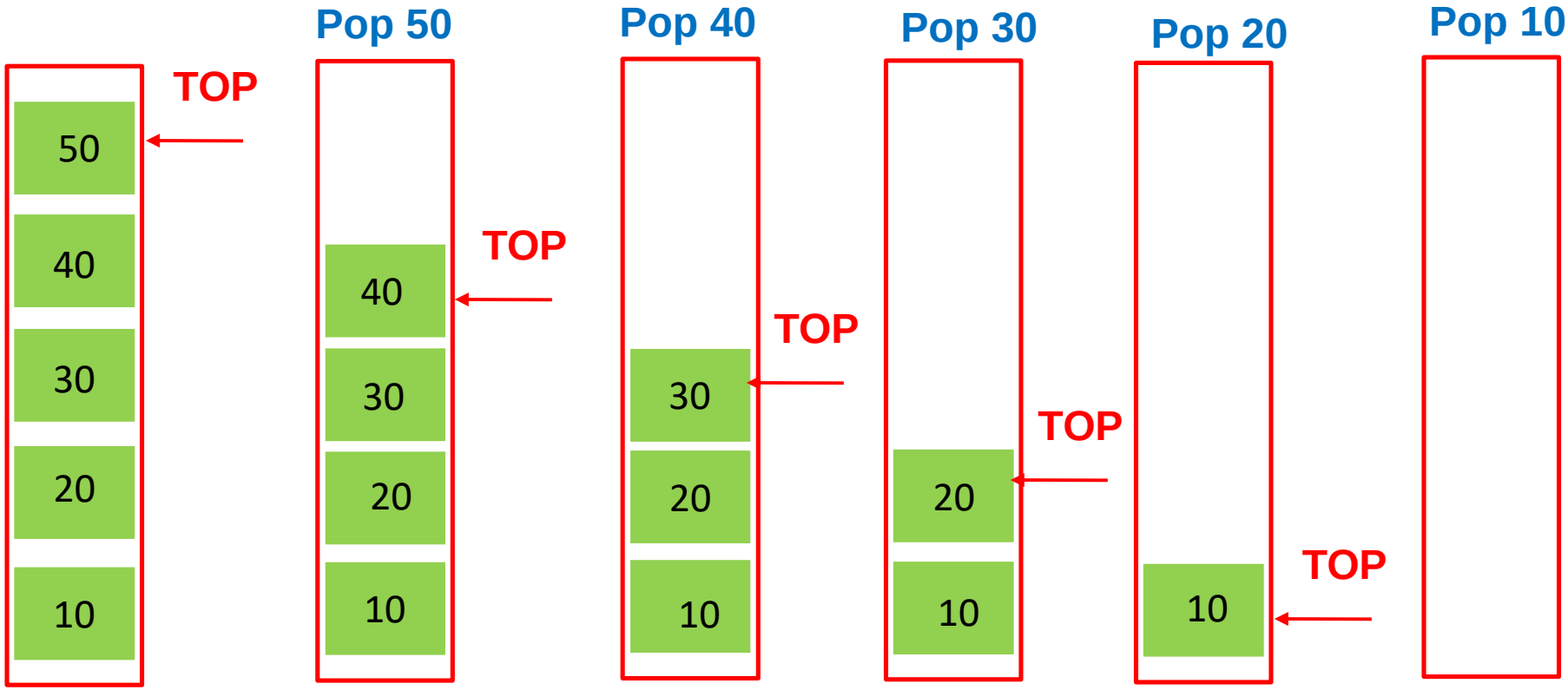


- To delete the topmost element, we first check if TOP=NULL. If the condition is false, then we decrement the value pointed by TOP. Thus, the updated stack becomes as shown in Fig



Operation: Popping from Stack

Pop : 50, 40, 30, 20, 10



Stack Full

Stack Empty

Operation: Stack Pop

Step 1: If TOP= -1, then

print “Underflow”

Goto Step 4

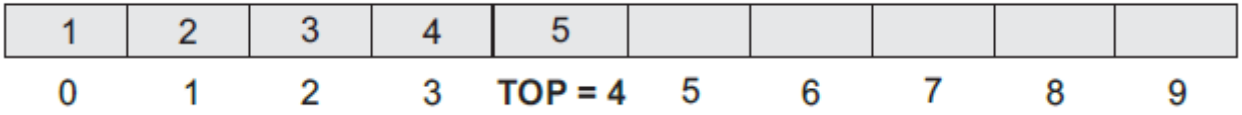
Step 2: Set value=Stack[TOP]

Step 3: Set TOP=TOP-1

Step 4:End

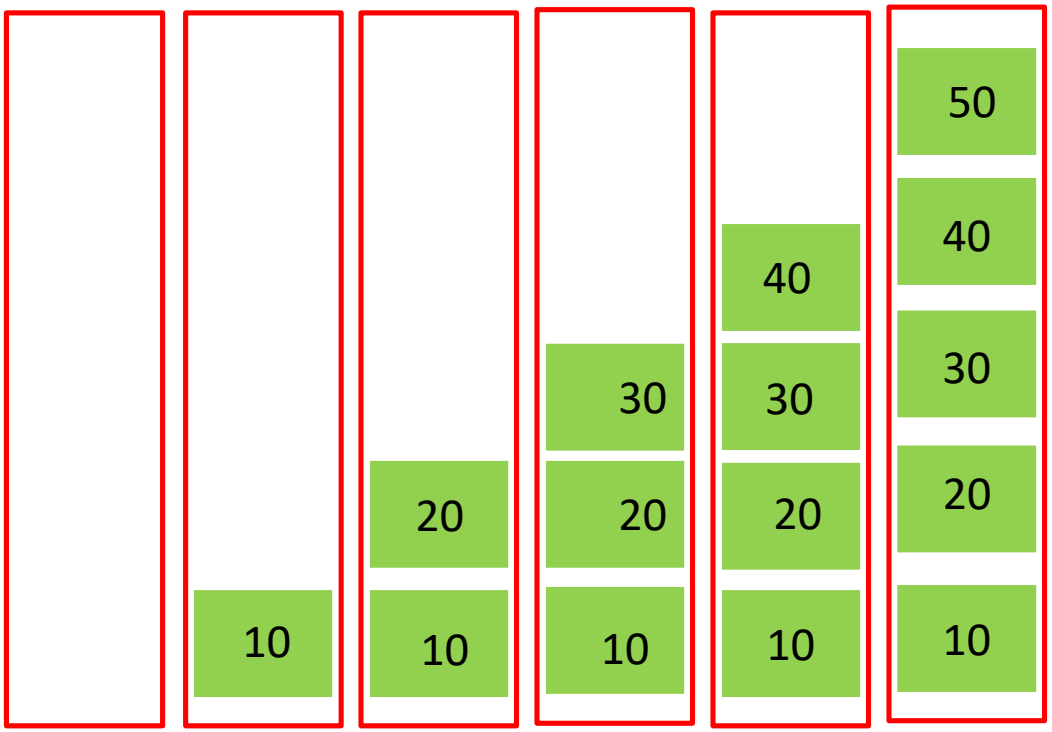
Operation: Stack Peep

- Peek is an operation that returns the value of the topmost element of the stack without deleting it from the stack.
- However, the Peek operation first checks if the stack is empty, i.e., if TOP = NULL, then an appropriate message is printed, else the value is returned.



- Here, the Peek operation will return 5, as it is the value of the topmost element of the stack.

Operation: Stack Peep



Step 1: If **TOP= -1,** then

print “Underflow”

Goto Step 4

Step 2: **Return value=Stack[TOP]**

Step 3:End

Problem Solving

Given the following sequence of letters and asterisks:
EAS*Y*QUE***ST***IO*N***

Consider the stack data structure, supporting two operations push and pop, as discussed in class. Suppose that for the above sequence, each letter (such as E) corresponds to a push of that letter onto the stack and each asterisk (*) corresponds a pop operation on the stack. Show the sequence of values returned by the pop operations.

http://ds1-iiith.vlabs.ac.in/data-structures-1/exp/stacks-queues/exp.html#Stacks%20:%20Arrays

SOLUTION

INPUT	STACK	OUTPUT
E	E	
A	E, A	
S	E, A, S	
*		S
Y	E, A, Y	
*		S, Y
Q	E, A, Q	
U	E, A, Q, U	
E	E, A, Q, U, E	
*		S, Y, E
*		S, Y, E, U
*		S, Y, E, U, Q

SOLUTION

INPUT	STACK	OUTPUT
S	E, A, S	
T	E, A, S, T	
*		S, Y, E, U, Q, T
*		S, Y, E, U, Q, T, S
*		S, Y, E, U, Q, T, S, A
I	E, I	
O	E, I, O	
*		S, Y, E, U, Q, T, S, A, O
N	E, I, N	
*		S, Y, E, U, Q, T, S, A, O, N
*		S, Y, E, U, Q, T, S, A, O, N, I
*		S, Y, E, U, Q, T, S, A, O, N, I, E

Program

```
#include <stdio.h>
int stack[100];
int top=-1;
int n;

int main()
{
   int choice;
   printf("Enter the height of the stack:
");
   scanf("%d", &n);

   do{
      printf("\nEnter Your Choice");
      printf("\n1.Push");
      printf("\n2.Pop");
      printf("\n3.Peak");
      printf("\n4.END");
      scanf("\n%d", &choice);

switch(choice)
    {
       case 1:
         push();
          break;
       case 2:
         pop();
          break;
        case 3:
         peek();
          break;
        case 4:
        printf("The Program is
Terminated");
          break;
        default:
         printf("Enter a valid choice");
         break;
```

Program

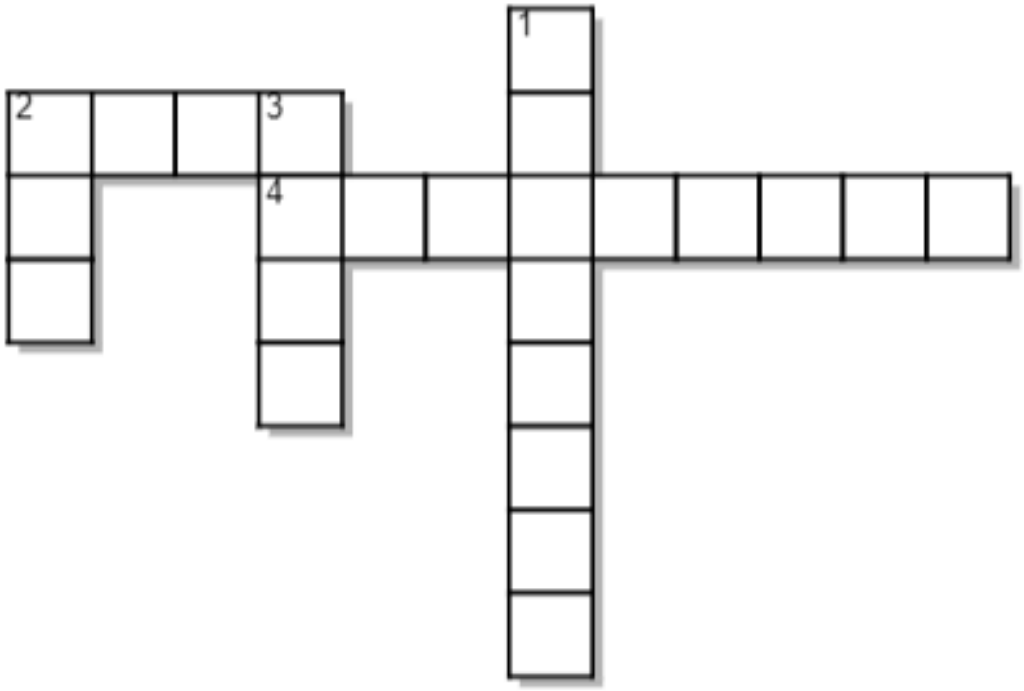
```
 }
      }
   }while(choice!=4);
   return 0;
}

void push()
{
   int num;
   if(top==n-1)
   {
      printf("\nStack Overflowed");
   }
   else
   {
      printf("\nEnter an element: ");

      scanf("%d", &num);
      top++;
      stack[top]=num;
   }
}
```

```
void pop()
{
   int num;
   if(top==-1)
   {
      printf("\nStack Underflowed");
   }
   else
   {
      num=stack[top];
      top--;
      printf("\n%d has been popped", num);
   }
}
```

Program

```
void peek()
{
    if(top==-1)
    {
        printf("\nStack is Empty");
    }
    else
    {
        printf("%d is the top element",
stack[top]);
    }
}
```

Cross Word Puzzle

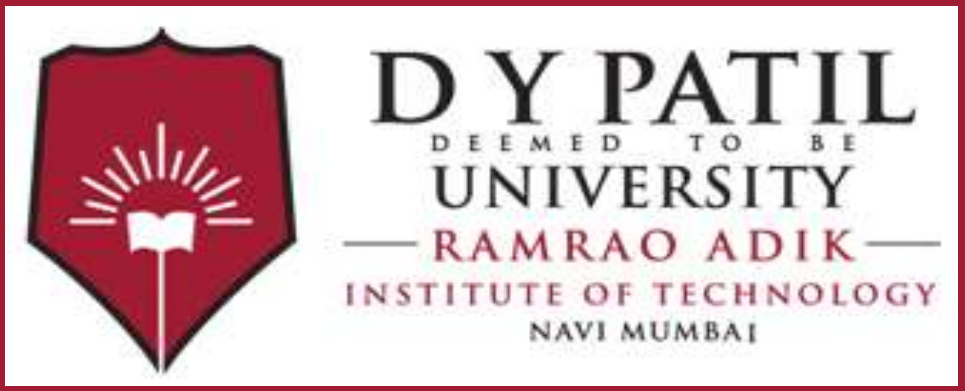


ACROSS

2 get top element
4 pop from empty stack

DOWN

1 push into full stack
2 delete element
3 insert element


D Y PATIL
DEEMED TO BE
UNIVERSITY
RAMRAO ADIK
INSTITUTE OF TECHNOLOGY
NAVI MUMBAI

Thank You