# Data Structures

## Lab Manual

**Subject Code: KCS-351**
**Class: II Year, III Semester (IT)**

**Prepared by**
**Mr. Ajay Raj Parashar**
**Assistant Professor**


**Department of Information Technology**
**Hindustan College of Science and Technology**
**Farah, Mathura – 281122 (U.P.)**

**Content for Laboratory manual**

1. Title page ( College logo , Department name, Lab Course title with code )
2. Course outcomes (COs) of the Lab course
3. Course outcome (CO)-Program outcome (PO) mapping
4. Course outcome (CO)-Program specific outcome (PSO) mapping
5. Evaluation scheme
6. List of experiments
7. Details of each experiment-
    - Title of experiment
    - Objective of experiment with CO
    - Background theory
    - List of components or resources required for experiment ( if any)
    - Block diagram or flow chart (if any )
    - Explanation of working principle / algorithm /Pseudocode
    - Key parameters and their values used in the experiment
    - Experiment sample outputs
    - conclusion

Note: Laboratory manual should be in spiral binding

RAJEEV KUMAR UPADHYAY

Digitally signed by RAJEEV KUMAR UPADHYAY
DN: C=IN, O=Personal, PostalCode=282001, S=Uttar Pradesh, SERIALNUMBER=AA3E8C12CFAA909878 5ACF2B07E25E09D7F5B87A4DCA301247 D08CBAEE03B9A3, CN=RAJEEV KUMAR UPADHYAY
Reason: I am the author of this document
Location: your signing location here
Date: 2023.09.08 10:44:18+05'30'
Foxit PhantomPDF Version: 10.1.1

---

**Vision of the College**

HCST strives to impart a holistic knowledge-centric environment to serve humanity by providing research-oriented technical education to nurture global leaders and entrepreneurs.

**Mission of the College**

1. Create an ecosystem to foster a culture of innovation, research, academic excellence and entrepreneurship.
2. Nurture technically competent and socially committed global leaders with high moral and ethical values.
3. Impart outcome based education to facilitate students for their holistic development.

# Hindustan College of Science and Technology, Farah-Mathura

## DEPARTMENT OF INFORMATION TECHNOLOGY

**Date: 20th Nov, 2017**

### Vision of the Department

Information Technology undergraduate program empowers students with research and training-based education to become global technical leaders and entrepreneurs, creating positive societal impact.

### Mission of the Department

1. Foster innovation, research, and entrepreneurship through real-world projects and industry collaborations.
2. Develop ethical global leaders with theoretical and practical education, critical thinking, and social responsibility.
3. Provide outcome-based education for holistic development, integrating technical skills with leadership and teamwork.

**PEOs**

1. Equip students with IT knowledge, critical thinking abilities, and leadership qualities through research-oriented education to become global entrepreneurs.
2. Prepare students to solve real-world problems and positively impact society through practical experience and industry collaborations.
3. Develop socially responsible global leaders with high moral and ethical values through community engagement and personal/professional growth opportunities.

**PSOs**

1. Equip students with the latest IT knowledge and skills to tackle real-world challenges.
2. Foster leadership, critical thinking, problem-solving, and communication skills for IT careers.
3. Encourage entrepreneurship and innovation through research, start-up projects, industry collaborations, and business skills.

RAJEEV KUMAR UPADHYAY

Digitally signed by RAJEEV KUMAR UPADHYAY
DN: C=IN, O=Personal, PostalCode=282001, S=Uttar Pradesh, SERIALNUMBER=AA3E8C12CFAA9098785ACF2B07E25E09D7F5B87A4DCA301247D08CBAEE03B9A3, CN=RAJEEV KUMAR UPADHYAY
Reason: I am the author of this document
Location: your signing location here
Date: 2023.09.08 10:44:06+05'30'
Foxit PhantomPDF Version: 10.1.1

HOD-IT
Head
Department of Information Technology
Hindustan College of Science & Technology
Farah, Mathura

# DEPARTMENT OF INFORMATION TECHNOLOGY

## GENERAL LAB INSTRUCTIONS

1. Students are advised to come to the laboratory at least 5 minutes before (to starting time), those who come after 5 minutes will not be allowed into the lab.
2. Plan your task properly much before to the commencement, come prepared to the lab with the synopsis / program / experiment details.
3. Student should enter the laboratory with:
    a. Laboratory observation notes with all the details (Problem statement, Aim, Algorithm, Procedure, Program, Expected Output, etc.,) filled in for the lab session.
    b. Laboratory Record updated up to the last session experiments and other utensils (if any) needed in the lab.
    c. Proper Dress code and Identity card.
4. Sign in the laboratory login register, write the TIME-IN, and occupy the computer system allotted to you by the faculty.
5. Execute your task in the laboratory, and record the results / output in the lab observation note book, and get certified by the concerned faculty.
6. All the students should be polite and cooperative with the laboratory staff, must maintain the discipline and decency in the laboratory.
7. Computer labs are established with sophisticated and high-end branded systems, which should be utilized properly.
8. Students / Faculty must keep their mobile phones in SWITCHED OFF mode during the lab sessions. Misuse of the equipment, misbehaviors with the staff and systems etc., will attract severe punishment.
9. Students must take the permission of the faculty in case of any urgency to go out; if anybody found loitering outside the lab / class without permission during working hours will be treated seriously and punished appropriately.
10. Students should LOG OFF/ SHUT DOWN the computer system before he/she leaves the lab after completing the task (experiment) in all aspects. He/she must ensure the system / seat is kept properly.

RAJEEV KUMAR UPADHYAY

Digitally signed by RAJEEV KUMAR UPADHYAY
DN: C=IN, O=Personal, PostalCode=282001, S=Uttar Pradesh, SERIALNUMBER=AA3E8C12CFAA9098785ACF2B07E25E09D7F5B87A4DCA301247D08CBAEE03B9A3, CN=RAJEEV KUMAR UPADHYAY
Reason: I am the author of this document
Location: your signing location here
Date: 2023.09.08 10:43:49+05'30'
Foxit PhantomPDF Version: 10.1.1

**Head of the Department**

Head
Department of Information Technology
Hindustan College of Science & Technology
Farah, Mathura

# EVALUATION SCHEME

## SEMESTER- III

| Sl. No. | Subject Codes | Subject | Periods | | | Evaluation Scheme | | | | End Semester | | Total | Credit |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | L | T | P | CT | TA | Total | PS | TE | PE | | |
| 1 | KOE031-38/ KAS302 | Engineering Science Course/Maths-IV | 3 | 1 | 0 | 30 | 20 | 50 | | 100 | | 150 | 4 |
| 2 | KAS301/ KVE301 | Technical Communication/Universal Human Values | 2<br>3 | 1<br>0 | 0<br>0 | 30 | 20 | 50 | | 100 | | 150 | 3 |
| 3 | KCS301 | Data Structure | 3 | 1 | 0 | 30 | 20 | 50 | | 100 | | 150 | 4 |
| 4 | KCS302 | Computer Organization and Architecture | 3 | 1 | 0 | 30 | 20 | 50 | | 100 | | 150 | 4 |
| 5 | KCS303 | Discrete Structures & Theory of Logic | 3 | 0 | 0 | 30 | 20 | 50 | | 100 | | 150 | 3 |
| 6 | KCS351 | Data Structures Using C Lab | 0 | 0 | 2 | | | | 25 | | 25 | 50 | 1 |
| 7 | KCS352 | Computer Organization Lab | 0 | 0 | 2 | | | | 25 | | 25 | 50 | 1 |
| 8 | KCS353 | Discrete Structure & Logic Lab | 0 | 0 | 2 | | | | 25 | | 25 | 50 | 1 |
| 9 | KCS354 | Mini Project or Internship Assessment* | 0 | 0 | 2 | | | 50 | | | | 50 | 1 |
| 10 | KNC301/ KNC302 | Computer System Security/Python Programming | 2 | 0 | 0 | 15 | 10 | 25 | | 50 | | | 0 |
| 11 | | MOOCs (Essential for Hons. Degree) | | | | | | | | | | | |
| | | **Total** | | | | | | | | | | **950** | **22** |

*The Mini Project or internship (3-4 weeks) conducted during summer break after II semester and will be assessed during III semester.

RAJEEV KUMAR UPADHYAY

Digitally signed by RAJEEV KUMAR UPADHYAY
DN: C=IN, O=Personal, PostalCode=282001, S=Uttar Pradesh, SERIALNUMBER=AA3E8C12CFAA9098785ACF2B07E25E09D7F5B87A4DCA301247D08CBAEE03B9A3, CN=RAJEEV KUMAR UPADHYAY
Reason: I am the author of this document
Location: your signing location here
Date: 2023.09.08 10:43:35+05'30'
Foxit PhantomPDF Version: 10.1.1

# DATA STRUCTURES LABORATORY

## OBJECTIVE:

The aim of this course is to introduce computer algorithms and data structures, with an emphasis on foundational material

At the end of the course students should

- Understand fundamental data structures, such as arrays, linked lists, stacks, queues, and trees.
- Enhance algorithmic thinking and problem-solving skills through the implementation of various search and sort algorithms, including Linear Search, Binary Search, and Bubble Sort.
- Learn Graph Traversal and Tree Traversals techniques.
- Learn working with dynamic data structures and understand their applications in real-world scenarios.
- Strengthen coding skills by implementing data structures and algorithms in the C programming language.

## COURSE OUTCOMES

| CO No. | Statement of Course Outcome |
|---|---|
| After completion of the course, the student will be able to | |
| CO1 | Implement various operations on Array and Linked List. |
| CO2 | Implement the concept of Stack and Queue using Array and LinkedList. |
| CO3 | Implement the concept of Tree Data Structure using Array and LinkedList. |
| CO4 | Implement various application of Graph data Structure. |
| CO5 | Implement various searching and Sorting Techniques. |

## CO-PO Mapping

| COs | PO 1 | PO 2 | PO 3 | PO 4 | PO 5 | PO 6 | PO 7 | PO 8 | PO 9 | PO1 0 | PO1 1 | PO1 2 | PSO 1 | PSO 2 | PSO 3 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| CO1 | 2 | 3 | 2 | 3 | 2 | | | | | 3 | 2 | 3 | 3 | 3 | |
| CO2 | 3 | 2 | 1 | 2 | 2 | | | | | 3 | 3 | 2 | 3 | 3 | |
| CO3 | 3 | 1 | 3 | 3 | 2 | | | | | 3 | 2 | 3 | 3 | 3 | |
| CO4 | 1 | 2 | 3 | 2 | 3 | | | | | 3 | 3 | 2 | 3 | 3 | |
| CO5 | 3 | 3 | 3 | 2 | 3 | | | | | 2 | 2 | 3 | 3 | 3 | |
| Avg | 2.4 | 2.2 | 2.4 | 2.4 | 2.4 | 0 | 0 | 0 | 0 | 2.8 | 2.4 | 2.6 | 3 | 3 | 0 |

| | |
|---|---|
| PO1 | Engineering Knowledge |
| PO2 | Problem Analysis |
| PO3 | Design/development of solutions |
| PO4 | Conduct investigations of complex Problems |
| PO5 | Modern tool usage |
| PO6 | The engineer and society |
| PO7 | Environment and sustainability |
| PO8 | Ethics |
| PO9 | Individual and team work |
| PO10 | Communication |
| PO11 | Project management and finance |
| PO12 | Life-long learning |

## SOFTWARE REQUIREMENTS

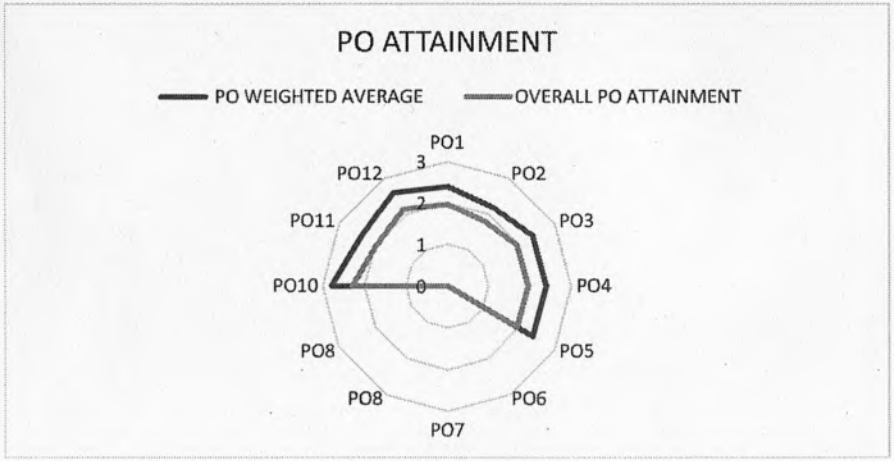
1. C compiler (e.g., GCC, Clang, Microsoft Visual C++ compiler) should be installed on the computer to compile and run C programs.
2. Integrated Development Environment (IDE): An IDE that supports C development can be helpful for writing, debugging, and managing the code. Examples include Code: Blocks, Dev-C++, Visual Studio, or Xcode (for macOS).
3. Text Editor: If an IDE is not used, a text editor like Notepad++, Sublime Text, Visual Studio Code, or Vim can be used for coding C programs.
4. Operating System: The course programs can be developed and executed on Windows, macOS, or Linux operating systems.

## HARDWARE REQUIREMENTS

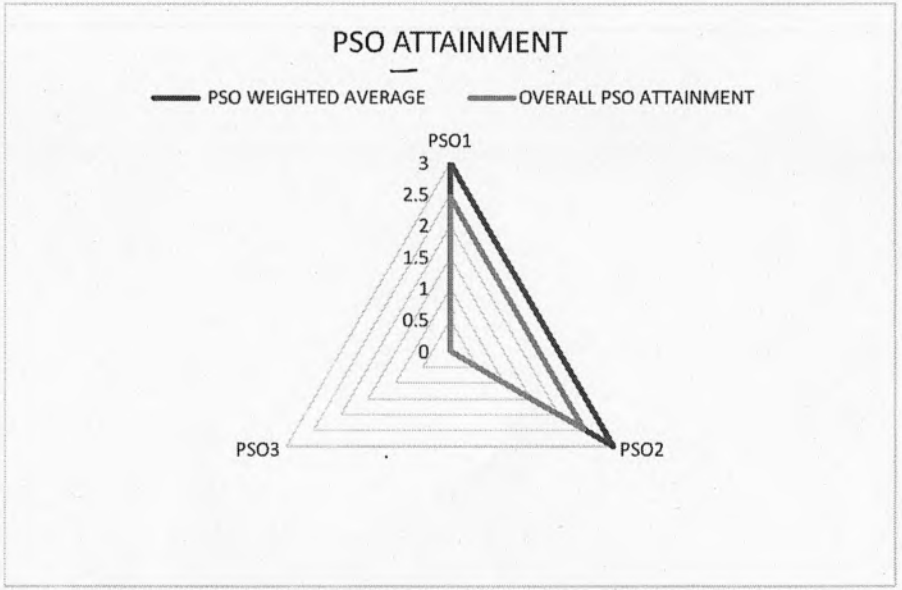
1. Processor: A modern dual-core processor should be sufficient for running the programs efficiently.
2. RAM: At least 4 GB of RAM is recommended for a smooth development experience.
3. Storage: A minimum of 10 GB of available storage space to install the required software and store the programs and data.
4. Display: A standard computer monitor or laptop screen with a resolution of 1366x768 or higher.

## PROGRAM AND PROGRAM SPECIFIC OUTCOME ATTAINMENT CALCULATIONS (2021-2022)

| Evaluation of PO Attainment | | | |
|---|---|---|---|
| AVERAGE CO ATTAINMENT | | | 2.46 |
| | PROGRAM OUTCOME | PO WEIGHTED AVERAGE | OVERALL PO ATTAINMENT |
| | PO1 | 2.4 | 1.97 |
| | PO2 | 2.2 | 1.80 |
| | PO3 | 2.4 | 1.97 |
| | PO4 | 2.4 | 1.97 |
| | PO5 | 2.4 | 1.97 |
| | PO6 | 0 | 0.00 |
| | PO7 | 0 | 0.00 |
| | PO8 | 0 | 0.00 |
| | PO8 | 0 | 0.00 |
| | PO10 | 2.8 | 2.30 |
| | PO11 | 2.4 | 1.97 |
| | PO12 | 2.6 | 2.13 |




| Evaluation of PSO Attainment | | | |
|---|---|---|---|
| AVERAGE CO ATTAINMENT | | | 2.46 |
| | PROGRAM SPECIFIC OUTCOMES | PSO WEIGHTED AVERAGE | OVERALL PSO ATTAINMENT |
| | PSO1 | 3 | 2.46 |
| | PSO2 | 3 | 2.46 |
| | PSO3 | 0 | 0 |




RAJEEV KUMAR UPADHYAY

Digitally signed by RAJEEV KUMAR UPADHYAY
DN: C=IN, O=Personal, PostalCode=282001, S=Uttar Pradesh, SERIALNUMBER=AA3E8C12CFAA9098785ACF2B07E25E09D7F5B87A4DCA301247D08CBAEE03B9A3, CN=RAJEEV KUMAR UPADHYAY
Reason: I am the author of this document
Location: your signing location here
Date: 2023.09.08 10:42:50+05'30'
Foxit PhantomPDF Version: 10.1.1

# LIST OF PROGRAMS

| | | |
|---|---|---|
| **Exp 1** | Write a program in "C" to insert and delete a data at/from location in Array. | **CO1** |
| **Exp 2** | Write a program in "C" to insert and delete a data from I) Beginning II) Last, and III) Particular Location in single linked list and display its elements. | |
| **Exp 3** | Write a program in "C" to implement a stack using Array and Linked List. | **CO2** |
| **Exp 4** | Write a program in "C" to implement a queue using Array and Linked List. | |
| **Exp 5** | Write a program in "C" to implement a Tree Traversals. | **CO3** |
| **Exp 6** | Write a program in "C" to implement BFS Technique. | **CO4** |
| **Exp 7** | Write a program in "C" to implement Linear Search/Binary Search. | **CO5** |
| **Exp 8** | Write a program in "C" to implement Bubble Sort. | |

# PROGRAM NO. 1

**OBJECTIVE:** Write a program in "C" to INSERT and DELETE a data at/from location in Array.

## THEORY:

### ARRAYS

***"An array is a collection of variables of the same type that are referenced by a common name."***

Arrays are a way to group several items into a larger unit. In C, array elements are stored in contiguous (consecutive) memory locations. The lowest address corresponds to the first element and the highest address to the last element. Arrays can have data items of simple types like **int** or **float** or even of user-defined types like structure or objects.

### Types of Arrays

Arrays are of different types:

1. **Single-dimensional arrays** comprised of finite homogenous (same type) elements.
2. **Multi-dimensional arrays** comprised of elements, each of which is itself an array. A two-dimensional array is the simplest of the multi-dimensional arrays. However, C programming allows arrays of more than two dimensions. The exact limit (of dimensions) is determined by the compiler you use.

### SINGLE DIMENSIONAL ARRAYS

- The simplest form of an array is the **single-dimensional array**.
- The array is given a name and its elements are referred to by their subscripts or indices.
- C language array's index numbering starts with zero.
- The index of first element is known as **lower bound** and
- The index of the last element is known as **upper bound**.

### Declaration of Single-dimensional array:

**Syntax:**

```
data_type array-name[size];
```

Where *data_type* declares the base type of array, which is the type of each element in the array. The *array-name* specifies the name with which the array will be referenced and *size* defines how many elements the array will hold. The size must be an integer value or integer constant without any sign.

**For e.g.**

```
int marks[10];
```

The above statement declared array **marks** with 10 elements, marks[0] to marks[9].

### Initialization of array :

```
data_type array-name[size]={element-1,element-2,……..,element-n};
                                    or
data_type array-name[ ]={element-1,element-2,……..,element-n};
```

**For example,**

```
int marks[5]={50,25,72,45,30};
Marks[0]=50;
Marks[1]=25;
```

```
Marks[2]=72;
Marks[3]=45;
Marks[4]=30;
```
or
```
int marks[ ]={50,25,72,45,30};
```

Also float price[ ] = {300.50, 250.50, 500.50, 175.50, 900.50};
and
```
char grade[ ] = {'D' , 'A' , 'C' , 'B' , 'A' , 'C' };
```

**Accessing an element at a particular index for one-dimensional array**
An individual element of an array can be accessed using the following syntax:

**array_name[index or subscript];**

For example, to assign a value to second location of array, we give the following statement
```
marks[1]=90;
```

Similarly, for reading the value of fourth element in array_name **marks**, we give the following statement:
```
scanf("%d",&marks[3]);
```

For writing the value of second element in array_name **marks**, we give the following statement:
```
printf("%d\t",marks[1]);
```

Arrays can always be read or written through loop. If we read a one-dimensional array, it requires one loop for reading and other for writing (printing) the array. For example:

**(a) For reading the array**
For reading the marks of 10 students:
```
for ( i = 0; i <10 ; i++)
{
scanf("%d", &marks [ i ] );
}
```

**(b) For writing the array**
```
for ( i = 0; i <10 ; i++)
{
printf("%d\t", marks [ i ] );
}
```

## ALGORITHM TO INSERT A NUMBER IN AN ARRAY:

1. Start the program.
2. Declare variables `len`, `A`, `num`, `i`, and `pos`.
3. Ask the user to enter the size of the array (`len`) with a maximum limit of .
4. Read the value of `len` from the user.
5. Declare an integer array `A` of size to store the elements.
6. Ask the user to enter `len` elements of the array.
7. Read `len` elements and store them in the array `A`.
8. Print the original array `A` using a loop from to `len-`.
9. Ask the user to enter the element `num` to be inserted.
10. Read the value of `num` from the user.
11. Ask the user to enter the position `pos` at which the element should be inserted.
12. Read the value of `pos` from the user.
13. Decrement `pos` by (to adjust for zero-based indexing).
14. Start a loop from `len-` to `pos`, and shift each element in the array `A` one position down.
15. Assign the value of `num` to `A[pos]` to insert the element at the desired position.
16. Check if the `pos` is greater than the original `len`.
    a. If `pos` is greater, print "Insertion outside the array."
    b. If not, proceed to the next step.
17. Print "New array after insertion".
18. Increment `len` to accommodate the newly inserted element.
19. Print the updated array `A` using a loop from to `len-`.
20. End the program.

### Sample Output:

Enter size of array (max. 10) : 5
Enter 5 elements of array :
2 4 8 10 12
Original array is :
2 4 8 10 12
Enter the element to be inserted : 6
Enter the position of insertion : 3
New array after insertion :
2 4 6 8 10 12

## ALGORITHM TO DELETE A NUMBER FROM A PARTICULAR LOCATION IN AN ARRAY:

1. Declare an integer array A of size 10 to hold the elements.
2. Declare integer variables i, len, num, and f.
3. Set f = 0 (f is a flag variable to indicate whether the number to be deleted is found in the array or not).
4. Display a message to the user to enter the size of the array (max. 10) and read the input into the variable len.
5. Display a message to the user to enter len elements of the array.
6. Use a loop to read len elements one by one and store them in the array A.
7. Display a message indicating the original array.
8. Use a loop to print all elements of the array A.
9. Display a message to the user to enter the number to be deleted and read the input into the variable num.
10. Use a loop to traverse the array A.

11. If the current element A[i] is equal to num, it means the number is found in the array.
12. Set f = 1 to indicate that the number is found.
13. Use another loop to shift all elements after the found element one position to the left.
14. Decrement len to indicate that the array size has reduced by 1 due to deletion.
15. Exit the loop after the deletion is done.
16. If f is still 0, it means the number was not found in the array. In this case, print "Number not found in array."
17. If f is 1, it means the number was found and deleted successfully. In this case, display a message indicating the new array after deletion.
18. Use a loop to print all elements of the updated array A.

## Sample Output:

```
Enter the size of array (max. 10) : 5
Enter 5 elements of an array :
2 4 6 8 10
Original array is :
2 4 6 8 10
Enter the element to delete : 6
New Array after deletion :
2 4 8 10
```

## VIVA QUESTIONS:

**a)** What is an array, and how is it different from a linked list?
**b)** Explain how you would insert an element at a specific location in an array.
**c)** How would you delete an element from an array given its location?
**d)** What is the time complexity for inserting an element at a particular location in the array?
**e)** Can you explain a scenario where using an array would be advantageous over a linked list and vice versa?

## PROGRAM NO. 2

**OBJECTIVE:** Write a program to create a single linked list and perform INSERTION / DELETION operation.

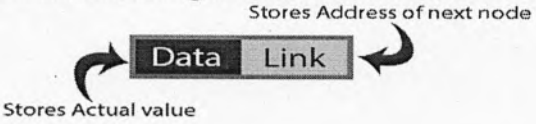
**THEORY:** When we want to work with unknown number of data values, we use a linked list data structure to organize that data. Linked list is a linear data structure that contains sequence of elements such that each element links to its next element in the sequence. Each element in a linked list is called as **"Node"**.

***"A linked list is a linear collection of data elements, called node pointing to the next nodes by means of pointers."***

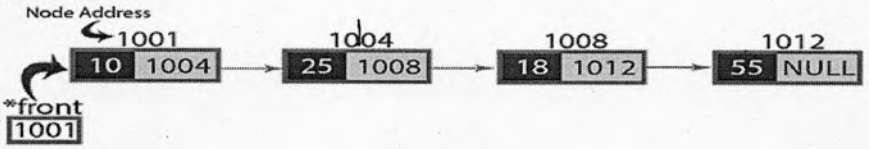
In a linked list, the individual element is called as **"Node".** Each Node is divided into two parts:

- The first part containing the information of the element called as **"data"**, and
- The second part called the **"link or next"** pointer containing the address of the next node in the list.

The graphical representation of a node in a single linked list is as follows:




**Example of a Linked List:**




## ALGORITHM:

Before we implement actual operations, first we need to setup empty list. First perform the following steps before implementing actual operations.

- **Step 1:** Include all the **header files** which are used in the program.
- **Step 2:** Declare all the **user defined** functions.
- **Step 3:** Define a **Node** structure with two members **data** and **next**
- **Step 4:** Define a Node pointer **'head'** and set it to **NULL**.
- **Step 5:** Implement the **main** method by displaying operations menu and make suitable function calls in the main method to perform user selected operation.

## INSERTION

In a single linked list, the insertion operation can be performed in three ways. They are as follows...

1. Inserting At Beginning of the list
2. Inserting At End of the list
3. Inserting At Specific location in the list

**Inserting At Beginning of the list**

We can use the following steps to insert a new node at beginning of the single linked list...

- **Step 1:** Create a **newNode** with given value.
- **Step 2:** Check whether list is **Empty (head == NULL)**
- **Step 3:** If it is **Empty** then, set **newNode→next = NULL** and **head = newNode.**
- **Step 4:** If it is **Not Empty** then, set **newNode→next = head** and **head = newNode.**

**Inserting At End of the list**

We can use the following steps to insert a new node at end of the single linked list...

- **Step 1:** Create a **newNode** with given value and **newNode → next** as **NULL**.
- **Step 2:** Check whether list is **Empty (head == NULL)**.
- **Step 3:** If it is **Empty** then, set **head = newNode**.
- **Step 4:** If it is **Not Empty** then, define a node pointer **temp** and initialize with **head**.
- **Step 5:** Keep moving the **temp** to its next node until it reaches to the last node in the list (until **temp → next** is equal to **NULL**).
- **Step 6:** Set **temp → next = newNode**.

**Inserting At Specific location in the list (After a Node)**
We can use the following steps to insert a new node after a node in the single linked list...

- **Step 1:** Create a **newNode** with given value.
- **Step 2:** Check whether list is **Empty (head == NULL)**
- **Step 3:** If it is **Empty** then, set **newNode → next = NULL** and **head = newNode**.
- **Step 4:** If it is **Not Empty** then, define a node pointer **temp** and initialize with **head**.
- **Step 5:** Keep moving the **temp** to its next node until it reaches to the node after which we want to insert the newNode (until **temp → data** is equal to **location**, here location is the node value after which we want to insert the newNode).
- **Step 6:** Every time check whether **temp** is reached to last node or not. If it is reached to last node then display **'Given node is not found in the list!!! Insertion not possible!!!'** and terminate the function. Otherwise move the **temp** to next node.
- **Step 7:** Finally, Set **'newNode → next = temp → next'** and **'temp → next = newNode'**

SAMPLE OUTPUT

```
Enter the No. of Nodes in Linked List: 3

Enter data for Head Node: 10

Enter data for 2 Node: 20

Enter data for 3 Node: 30

Select your choice
Select:
1 For Insertion at Beginning,
2 For Insertion at End,
3 For Insertion Between two nodes,
4 For Exit
1

Enter the value to be inserted: 40

One Node Inserted at Beginning Successfully
Elements of Linked List are: 40 -> 10 -> 20 -> 30 ->
Select your choice
Select:
1 For Insertion at Beginning,
2 For Insertion at End,
3 For Insertion Between two nodes,
4 For Exit
```

## DELETION

In a single linked list, the deletion operation can be performed in three ways. They are as follows:

1. Deleting from Beginning of the list
2. Deleting from End of the list
3. Deleting a Specific Node

**Deleting from Beginning of the list**

We can use the following steps to delete a node from beginning of the single linked list...

- **Step 1:** Check whether list is **Empty (head == NULL)**
- **Step 2:** If it is **Empty** then, display **'List is Empty!!! Deletion is not possible'** and terminate the function.
- **Step 3:** If it is **Not Empty** then, define a Node pointer **'temp'** and initialize with **head**.
- **Step 4:** Check whether list is having only one node **(temp → next == NULL)**
- **Step 5:** If it is **TRUE** then set **head = NULL** and delete **temp** (Setting **Empty** list conditions)
- **Step 6:** If it is **FALSE** then set **head = temp → next**, and delete **temp**.

**Deleting from End of the list**

We can use the following steps to delete a node from end of the single linked list...

- **Step 1:** Check whether list is **Empty (head == NULL)**
- **Step 2:** If it is **Empty** then, display **'List is Empty!!! Deletion is not possible'** and terminate the function.
- **Step 3:** If it is **Not Empty** then, define two Node pointers **'temp1'** and **'temp2'** and initialize **'temp1'** with **head**.
- **Step 4:** Check whether list has only one Node **(temp1 → next == NULL)**
- **Step 5:** If it is **TRUE**. Then, set **head = NULL** and delete **temp1**. And terminate the function. (Setting **Empty** list condition)
- **Step 6:** If it is **FALSE**. Then, set **'temp2 = temp1 '** and move **temp1** to its next node. Repeat the same until it reaches to the last node in the list. (until **temp1 → next == NULL**)
- **Step 7:** Finally, set **temp2 → next = NULL** and delete **temp1**.

**Deleting a Specific Node from the list**

We can use the following steps to delete a specific node from the single linked list...

- **Step 1:** Check whether list is **Empty (head == NULL)**
- **Step 2:** If it is **Empty** then, display **'List is Empty!!! Deletion is not possible'** and terminate the function.
- **Step 3:** If it is **Not Empty** then, define two Node pointers **'temp1'** and **'temp2'** and initialize **'temp1'** with **head**.
- **Step 4:** Keep moving the **temp1** until it reaches to the exact node to be deleted or to the last node. And every time set **'temp2 = temp1'** before moving the **'temp1'** to its next node.
- **Step 5:** If it is reached to the last node then display **'Given node not found in the list! Deletion not possible!!!'**. And terminate the function.
- **Step 6:** If it is reached to the exact node which we want to delete, then check whether list is having only one node or not
- **Step 7:** If list has only one node and that is the node to be deleted, then set **head = NULL** and delete **temp1 (free(temp1))**.
- **Step 8:** If list contains multiple nodes, then check whether **temp1** is the first node in the list **(temp1 == head)**.
- **Step 9:** If **temp1** is the first node then move the **head** to the next node **(head = head → next)** and delete **temp1**.
- **Step 10:** If **temp1** is not first node then check whether it is last node in the list **(temp1 → next == NULL)**.
- **Step 11:** If **temp1** is last node then set **temp2 → next = NULL** and delete **temp1 (free(temp1))**.
- **Step 12:** If **temp1** is not first node and not last node then set **temp2 → next = temp1 → next** and delete **temp1 (free(temp1))**.

## SAMPLE OUTPUT:

```
Enter the No. of Nodes in Linked List: 4

Enter data for Head Node: 1

Enter data for 2 Node: 2

Enter data for 3 Node: 3

Enter data for 4 Node: 4

Elements of Linked List are: 1 -> 2 -> 3 -> 4 ->
Select your choice
Select:
1 For Deletion from Beginning,
2 For Deletion from End,
3 For Deletion from Location,
4 For Exit
3

Enter the node value to be deleted: 3

Node deleted successfully!!!
Elements of Linked List are: 1 -> 2 -> 4 ->
Select your choice
Select:
1 For Deletion from Beginning,
2 For Deletion from End,
3 For Deletion from Location,
4 For Exit
```

## VIVA QUESTIONS:

**a)** What is a singly linked list, and how is it different from a doubly linked list?

**b)** How do you insert a new node at the beginning of a singly linked list?

**c)** Explain the process of deleting a node from the end of the singly linked list.

**d)** How would you delete a node from the singly linked list, given a specific value it holds?

**e)** Discuss the advantages and disadvantages of using a linked list over an array.

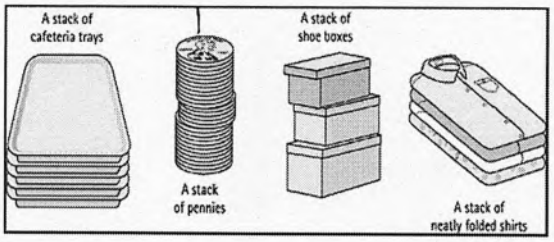
# PROGRAM NO. 3

**OBJECTIVE:** Write a program to demonstrate STACK primitive operations PUSH and POP using **Array (Static Implementation) and Linked List (Dynamic Implementation).**

## THEORY:

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

**Examples:**



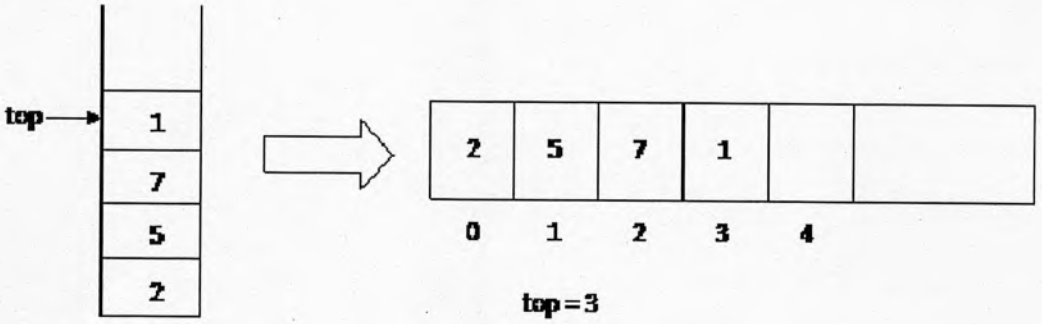

## IMPLEMENTATION OF STACK

It can be implemented in two ways:

- Static Implementation
- Dynamic Implementation

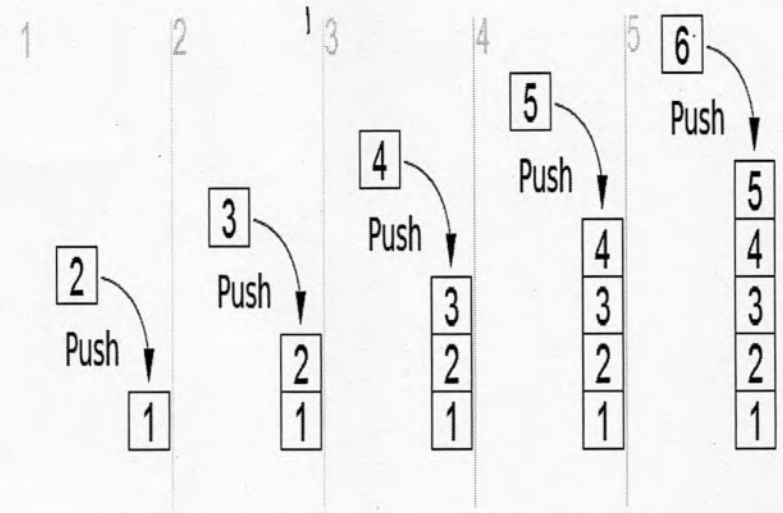
### STATIC IMPLEMENTATION

- Static implementation uses **arrays** to create stack.
- Static implementation is a very simple technique, but is not a flexible way of creation, as the size of stack has to be declared during program design, after that the size cannot be varied.
- Moreover, static implementation is not too efficient w.r.t. memory utilization. As the declaration of array (for implementing stack) is done before the start of the operation (at program design time).



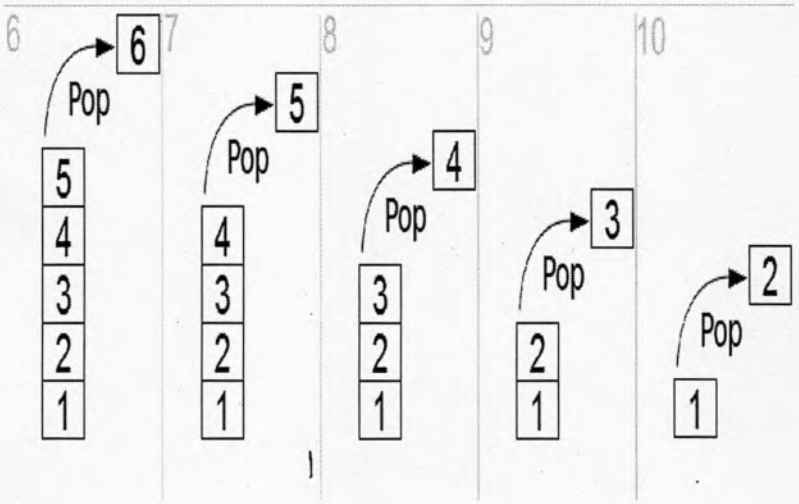

**PUSH Operation**:

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




**POP Operation:**

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




## ALGORITHM (Static Implementation):

**Algorithm for inserting an item into the stack (PUSH) using Arrays**

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

**Step 1:** [Check for stack overflow ? ]
If TOP = MAXSIZE – 1, then:
Write: 'Stack Overflow' and return.
[End of If Structure]
**Step 2:** Read NUM to be pushed in stack.
**Step 3:** Set TOP = TOP + 1 [Increases TOP by 1]
**Step 4:** Set STACK[TOP] = NUM
[Inserts new number NUM in new TOP Position]
**Step 5:** Exit

**Algorithm for deleting an item from the stack (POP) using Arrays**

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

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

**Step 2:** Set NUM = STACK[TOP] [Assign Top element to NUM]

**Step 3:** Write 'Element popped from stack is: ',NUM.

**Step 4:** Set TOP = TOP - 1 [Decreases TOP by 1]

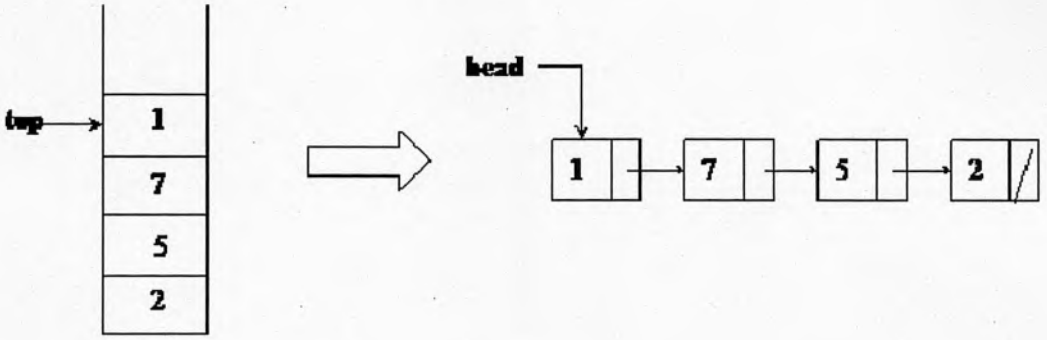
**Step 5:** Exit

**SAMPLE OUTPUT**

```
Enter the Choice:
1. For PUSH
2. For POP
3. EXIT 1

Enter the value to be pushed in stack: 10

 Value Pushed in Stack Successfully
Elements in STACK :10
Enter the Choice:
1. For PUSH
2. For POP
3. EXIT 1

Enter the value to be pushed in stack: 20

 Value Pushed in Stack Successfully
Elements in STACK :10   20
Enter the Choice:
1. For PUSH
2. For POP
3. EXIT 2

Value deleted is 20
Elements in STACK :10
Enter the Choice:
1. For PUSH
2. For POP
3. EXIT
```

**DYNAMIC IMPLEMENTATION**

- This problem associated with static implementation can be overcome if we implement a stack using **a linked list.**
- In case of a linked list, we shall push and pop nodes from one end of a linked list.
- Linked list representation is commonly known as **Dynamic implementation** and uses pointers to implement the stack type of data structure.
- The stack as linked list is represented as a singly connected list. Each node in the linked list contains the data and a pointer that gives location of the next node in the list.




## ALGORITHM (Dynamic Implementation):

**Algorithm for inserting an item into the stack (PUSH) using Linked List**

Let **NODE** is the structure pointer which allocates memory for the **newnode** & **NUM** is the element to be pushed into stack, **TOP** represents the address of node at the top of the **stack (head)**, **DATA** represents the information part of the node and **NEXT** represents the link or next pointer pointing to the address of next node.

**Step 1:** Allocate memory for the newnode using node structure.
**Step 2:** Read NUM to be pushed into stack.
**Step 3:** Set newnode->data=NUM
**Step 4:** Set newnode->next=TOP
**Step 5:** Set TOP = newnode
**Step 6:** Exit

**Algorithm for deleting an item from the stack (POP) using Linked List**

Let **TEMP** is the structure pointer which deallocates memory of the node at the top of stack & **NUM** is the element to be popped from stack, **TOP** represents the address of node at the top of the stack, **IDATA** represents the information part of the node and **NEXT** represents the next pointer pointing to the address of next node.

**Step 1:** [Check for Stack Underflow ?]
If TOP = NULL: then
Write 'Stack Underflow' & Return.
[End of If Structure]
**Step 2:** Set TEMP=TOP.
**Step 3:** Set NUM=TEMP->DATA
**Step 4:** Write 'Element popped from stack is: ',NUM
**Step 5:** Set TOP=TOP->NEXT
**Step 6:** Deallocate memory of the node at the top using
FREE(TEMP).
**Step 5:** Exit

**SAMPLE OUTPUT**

```
Enter the Choice:
1. For PUSH
2. For POP
3. EXIT 1

Enter the value to be pushed in stack: 10

 Value Pushed in Stack Successfully
Elements in STACK :10
Enter the Choice:
1. For PUSH
2. For POP
3. EXIT 1

Enter the value to be pushed in stack: 20

 Value Pushed in Stack Successfully
Elements in STACK :10   20
Enter the Choice:
1. For PUSH
2. For POP
3. EXIT 2

Value deleted is 20
Elements in STACK :10
Enter the Choice:
1. For PUSH
2. For POP
3. EXIT
```

## VIVA QUESTIONS:

**a)** Define what a stack is and explain its LIFO (Last-In-First-Out) property.
**b)** How do you push an element onto the stack using an array implementation?
**c)** Explain the process of popping an element from the stack using a linked list implementation.
**d)** Discuss the time complexity of various stack operations using the array and linked list.
**e)** Compare the advantages and disadvantages of using an array-based stack versus a linked-list-based stack.

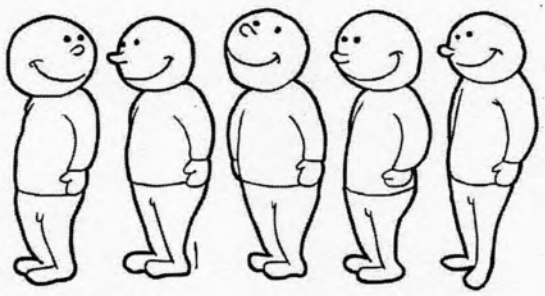
## PROGRAM NO. 4

**OBJECTIVE:** Write a program to demonstrate QUEUE primitive operations ENQUEUE and DEQUEUE using **Array (Static Implementation) and Linked List (Dynamic Implementation)**.

## THEORY:

- Queue is a linear data structure in which the insertion and deletion operations are performed at two different ends.
- In a queue data structure, the insertion operation is performed at a position which is known as '**rear**' and the deletion operation is performed at a position which is known as '**front**'.
- The first element that gets added into the queue is the first one to get removed from the list. Hence, Queue is also referred to as ***First-In-First-Out (FIFO)*** list.
- Consider a railway reservation booth, at which we have to get into the reservation queue.

**Examples:**



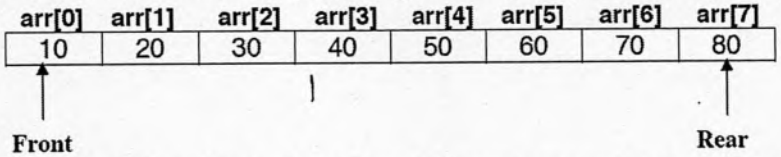
### Implementation of Queue

It can be implemented in two ways:

- Static Implementation
- Dynamic Implementation

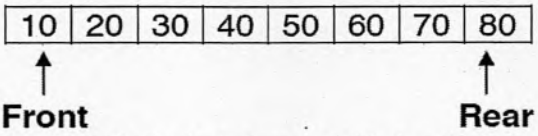
**STATIC IMPLEMENTATION**

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




- The following relation gives the total number of elements present in the queue, when implemented using arrays: **rear – front + 1**
- Also note that if ***front* > *rear***, then there will be no element in the queue or queue is empty.

**ENQUEUE Operation:**




In fig (1), 10 is the first element and 80 is the last element added to the Queue. Similarly, 10 would be the first element to get removed and 80 would be the last element to get removed. Figures 2(a) to 2(d) shows queue graphically during insertion operation:

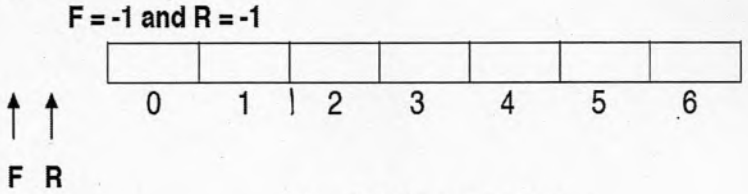



**Fig. 2(a) Empty Queue**

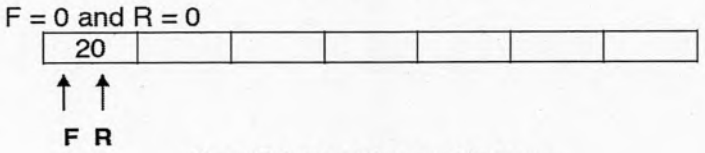



**Fig. 2(b) One Element Queue**

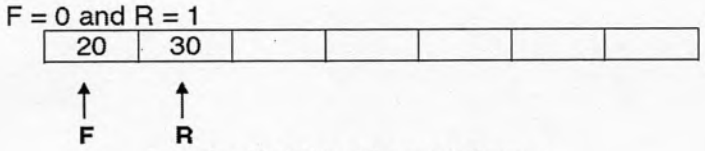



**Fig. 2(c) Two Element Queue**

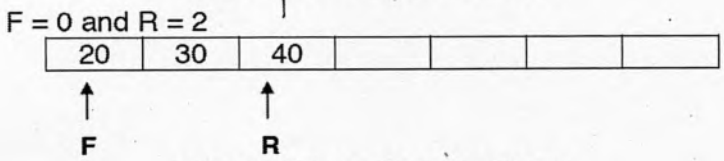



**Fig. 2(d) Three Element Queue**

**DEQUEUE Operation:**

The following figures show Queue graphically during deletion operation:

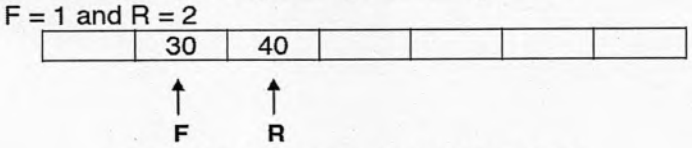



**Fig. 2(e) One Element (20) Deleted from Front**

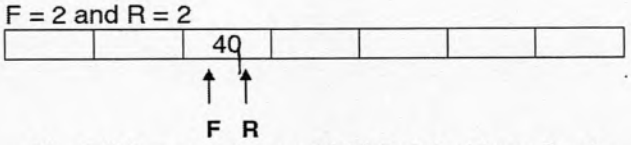



**Fig. 2(f) Second Element (30) Deleted from Front**

This is clear from Fig. 2(e) and 2(f), that whenever an element is removed from the queue, the value of Front is incremented by one i.e.,

**Front = Front + 1**

## ALGORITHM:

**Algorithm for Insertion in a Linear Queue (Using Arrays)**

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

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

**Algorithm for Deletion in a Linear Queue (Using Arrays)**

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

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

**SAMPLE OUTPUT**

```
***** MENU *****
1. Insertion
2. Deletion
3. Display
4. Exit
Enter your choice: 1
Enter the value to be insert: 10

Insertion success!!!

***** MENU *****
1. Insertion
2. Deletion
3. Display
4. Exit
Enter your choice: 1
Enter the value to be insert: 20

Insertion success!!!
```

```
***** MENU *****
1. Insertion
2. Deletion
3. Display
4. Exit
Enter your choice: 3

Queue elements are:
10      20

***** MENU *****
1. Insertion
2. Deletion
3. Display
4. Exit
Enter your choice: 2

Deleted : 10
```

## DYNAMIC IMPLEMENTATION ALGORITHM:

**Algorithm for Insertion in a Linear Queue (Using Linked Lists)**

- Let **PTR** is the structure pointer which allocates memory for the new node &
- **NUM** is the element to be inserted into linear queue,
- **DATA** represents the information part of the node and **NEXT** represents the link or next pointer pointing to the address of next node.
- **FRONT** represents the address of first node, REAR represents the address of the last node.
- Initially, Before inserting first element in the queue, **FRONT=REAR=NULL.**

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

**Algorithm for Deletion in a Linear Queue (Using Linked Lists)**

- Let **PTR** is the structure pointer which allocates memory for the new node &
- **NUM** is the element to be inserted into linear queue,
- **DATA** represents the information part of the node and **NEXT** represents the link or next pointer pointing to the address of next node.
- **FRONT** represents the address of first node, REAR represents the address of the last node.
- Initially, before inserting first element in the queue, **FRONT=REAR=NULL.**

**Step 1:** If FRONT = NULL: then
Write 'Queue is Empty (Queue Underflow)' and return.
[End of If structure]
**Step 2:** Set PTR = FRONT
**Step 3:** Set NUM = PTR->INFO
**Step 4:** Write 'Deleted element from linear queue is: ', NUM.
**Step 5:** Set FRONT = FRONT->LINK
**Step 6:** If FRONT = NULL: then
Set REAR = NULL.
[End of If Structure].
**Step 7:** Deallocate memory of the node at the beginning of
queue using PTR.
**Step 8:** Exit.

**SAMPLE OUTPUT**

```
***** MENU *****
1. Insertion
2. Deletion
3. Display
4. Exit
Enter your choice: 1
Enter the value to be insert: 10

Insertion success!!!

***** MENU *****
1. Insertion
2. Deletion
3. Display
4. Exit
Enter your choice: 1
Enter the value to be insert: 20

Insertion success!!!
```

```
***** MENU *****
1. Insertion
2. Deletion
3. Display
4. Exit
Enter your choice: 3

Queue elements are:
10      20

***** MENU *****
1. Insertion
2. Deletion
3. Display
4. Exit
Enter your choice: 2

Deleted : 10
```

## VIVA QUESTIONS:

**a)** Define what a queue is and explain its FIFO (First-In-First-Out) property.

**b)** How do you enqueue an element into the queue using an array implementation?

**c)** Explain the process of dequeuing an element from the queue using a linked list implementation.

**d)** Discuss the time complexity of various queue operations using the array and linked list.

**e)** Compare the advantages and disadvantages of using an array-based queue versus a linked-list-based queue.

## PROGRAM NO. 5

**OBJECTIVE:** Write a program in "C" to implement a Tree Traversals.

## THEORY:

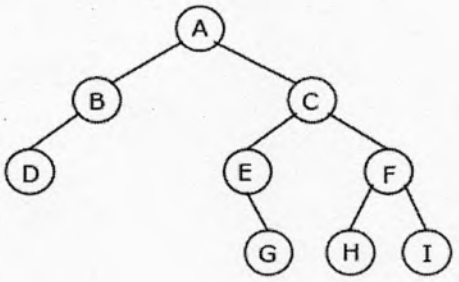
When we wanted to display a binary tree, we need to follow some order in which all the nodes of that binary tree must be displayed. In any binary tree displaying order of nodes depends on the traversal method.

***"Displaying (or) visiting order of nodes in a binary tree is called as Binary Tree Traversal."***

The traversal of a binary tree involves visiting each node in the tree exactly once.
There are three types of binary tree traversals.

1. **In - Order Traversal**
2. **Pre - Order Traversal**
3. **Post - Order Traversal**

Consider the following binary tree:




The Traversals are as follows:

- Preorder traversal yields:
  A, B, D, C, E, G, F, H, I
- Postorder traversal yields:
  D, B, G, E, H, I, F, C, A
- Inorder traversal yields:
  D, B, A, E, G, C, H, F, I

We will assume that the tree is implemented using a linked data structure (binary tree) with the following structure:

```
struct Node {
  int data;
  struct Node* left;
  struct Node* right;
};
```

## ALGORITHM FOR TREE TRAVERSALS:

1. Define the structure of a tree node (as shown above).
2. Create functions for creating a new node, inserting nodes into the tree, and for performing the three types of tree traversals (pre-order, in-order, and post-order).
3. Implement the function for creating a new node. It should take an integer value as input and return a pointer to the newly created node.
4. Implement the function for inserting a node into the tree. This function should take the root node and the value to be inserted as inputs.
5. Implement the function for pre-order traversal. This function should take the root node as input and print the elements in the tree in pre-order (root-left-right) sequence.

   **Algorithm for Pre-order Traversal Function:**

   **a)** Start the function for pre-order traversal, passing the root node of the binary tree as the input.
   **b)** Check if the current node (root) is not NULL.
   **c)** If the current node is not NULL, print the value of the current node.
   **d)** Recursively call the pre-order traversal function for the left subtree (i.e., pass the left child of the current node as the input to the pre-order function).
   **e)** Recursively call the pre-order traversal function for the right subtree (i.e., pass the right child of the current node as the input to the pre-order function).
   **f)** End the function.

   **Pseudocode for Pre-order Traversal Function:**

```
PreorderTraversal(node):
  // Step 2
  if node is not NULL:
    // Step 3
    print node's value

    // Step 4
    PreorderTraversal(node's left child)

    // Step 5
    PreorderTraversal(node's right child)

  // Step 6
  return
```

6. Implement the function for in-order traversal. This function should take the root node as input and print the elements in the tree in in-order (left-root-right) sequence.

   **Algorithm for In-order Traversal Function:**

   **a)** Start the function for in-order traversal, passing the root node of the binary tree as the input.
   **b)** Check if the current node (root) is not NULL.
   **c)** If the current node is not NULL, recursively call the in-order traversal function for the left subtree (i.e., pass the left child of the current node as the input to the in-order function).
   **d)** Print the value of the current node.
   **e)** Recursively call the in-order traversal function for the right subtree (i.e., pass the right child of the current node as the input to the in-order function).
   **f)** End the function.

**Pseudocode for In-order Traversal Function:**

```
InorderTraversal(node):
  // Step 2
  if node is not NULL:
    // Step 3
    InorderTraversal(node's left child)

    // Step 4
    print node's value

    // Step 5
    InorderTraversal(node's right child)

  // Step 6
  return
```

7. Implement the function for post-order traversal. This function should take the root node as input and print the elements in the tree in post-order (left-right-root) sequence.

**Algorithm for Post-order Traversal Function:**

**a)** Start the function for post-order traversal, passing the root node of the binary tree as the input.
**b)** Check if the current node (root) is not NULL.
**c)** If the current node is not NULL, recursively call the post-order traversal function for the left subtree (i.e., pass the left child of the current node as the input to the post-order function).
**d)** Recursively call the post-order traversal function for the right subtree (i.e., pass the right child of the current node as the input to the post-order function).
**e)** Print the value of the current node.
**f)** End the function.

**Pseudocode for Post-order Traversal Function:**

```
PostorderTraversal(node):
  // Step 2
  if node is not NULL:
    // Step 3
    PostorderTraversal(node's left child)

    // Step 4
    PostorderTraversal(node's right child)

    // Step 5
    print node's value

  // Step 6
  return
```

8. In the main function, create a sample binary tree by inserting elements.
9. Call the pre-order, in-order, and post-order traversal functions with the root of the sample tree to see the output.

**Sample Output:**

Suppose we have the following binary tree:

**Output of the program will be:**

Pre-order traversal: 5 3 2 4 8 9

In-order traversal: 2 3 4 5 8 9

Post-order traversal: 2 4 3 9 8 5

## VIVA QUESTIONS:

**a)** What is a binary tree, and how is it different from a binary search tree?

**b)** Explain the preorder traversal of a binary tree with an example.

**c)** How is an inorder traversal different from a postorder traversal of a binary tree?

**d)** How would you implement a level-order traversal of a binary tree using a queue?

**e)** Discuss the applications of different tree traversals in various algorithms and data structures.

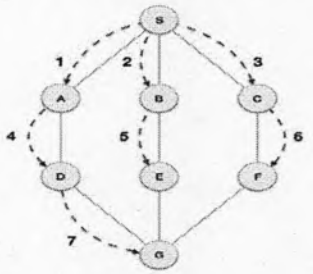
## PROGRAM NO. 6

**OBJECTIVE:** Write a program in "C" to implement BFS Technique.

## THEORY:

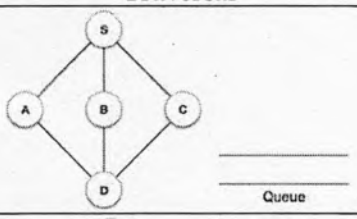
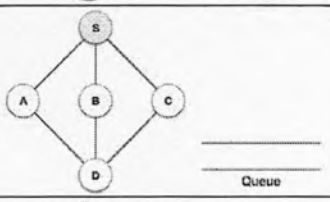
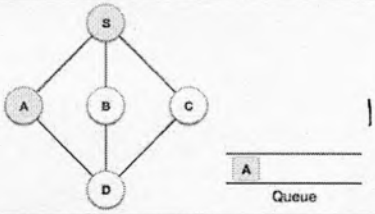
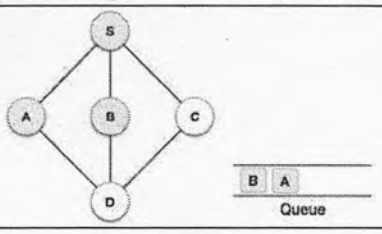
### Breadth First Search (BFS)

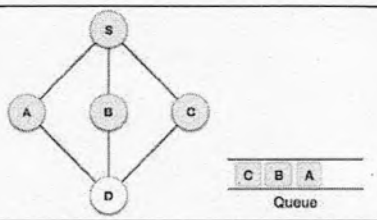
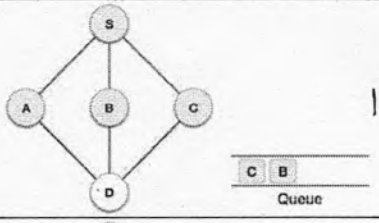
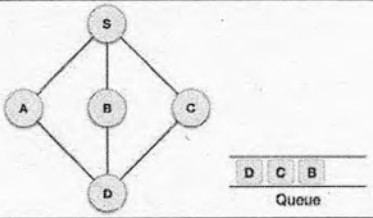
Breadth First Search (BFS) algorithm traverses a graph in a breadthward motion and uses a queue to remember to get the next vertex to start a search, when a dead end occurs in any iteration.




As in the example given above, BFS algorithm traverses from A to B to E to F first then to C and G lastly to D. It employs the following rules.

- **Rule 1** – Visit the adjacent unvisited vertex. Mark it as visited. Display it. Insert it in a queue.
- **Rule 2** – If no adjacent vertex is found, remove the first vertex from the queue.
- **Rule 3** – Repeat Rule 1 and Rule 2 until the queue is empty.

| Step | Traversal | Description |
|---|---|---|
| 1. |   | Initialize the queue. |
| 2. |   | We start from visiting **S** (starting node), and mark it as visited. |
| 3. |   | We then see an unvisited adjacent node from **S**. In this example, we have three nodes but alphabetically we choose **A**, mark it as visited and enqueue it. |
| 4. |   | Next, the unvisited adjacent node from **S** is **B**. We mark it as visited and enqueue it. |

| | | |
|---|---|---|
| 5. |  | Next, the unvisited adjacent node from **S** is **C**. We mark it as visited and enqueue it. |
| 6. |  | Now, **S** is left with no unvisited adjacent nodes. So, we dequeue and find **A**. |
| 7. |  | From **A** we have **D** as unvisited adjacent node. We mark it as visited and enqueue it. |

At this stage, we are left with no unmarked (unvisited) nodes. But as per the algorithm we keep on dequeuing in order to get all unvisited nodes. When the queue gets emptied, the program is over.

## ALGORITHM FOR BFS IMPLEMENTATION:

1. Initialize necessary data structures:
   - **a)** Create a graph data structure to represent the graph. It can be implemented using an adjacency list or an adjacency matrix.
   - **b)** Create a queue data structure to store nodes during the BFS traversal.
2. Create a function for adding edges to the graph:
   - **a)** The function should take the graph, source node, and destination node as parameters.
   - **b)** Add the destination node to the adjacency list of the source node (or update the corresponding cell in the adjacency matrix).
3. Create a function for BFS traversal:
   - **a)** The function should take the graph and the starting node as parameters.
   - **b)** Create a visited array to keep track of visited nodes. Initialize it with false for all nodes.
   - **c)** Create a queue and enqueue the starting node.
   - **d)** While the queue is not empty:
   - **e)** Dequeue a node from the queue.
   - **f)** Mark the dequeued node as visited.
   - **g)** Process the node (e.g., print it or perform other required operations).
   - **h)** Enqueue all unvisited neighbors of the dequeued node into the queue.
   - **i)** Continue this process until the queue becomes empty.
4. Call the BFS traversal function with the starting node as the input.

**Sample output:**

Let us consider a simple undirected graph and demonstrate the BFS traversal using the algorithm above. The graph can be represented as an adjacency list.

**Example graph:**

Graph:

1 -- 2
| |
3 -- 4

1. Initialize the data structures:
    a. Create an adjacency list to represent the graph.
    b. Create a queue to store nodes during BFS traversal.
2. Add edges to the graph:
    a. Add edges: (1, 2), (1, 3), (2, 1), (2, 4), (3, 1), (3, 4), (4, 2), (4, 3).
3. Perform BFS traversal starting from node 1:
    a. Mark node 1 as visited and enqueue it.
    b. Dequeue 1, print it, and enqueue its unvisited neighbors 2 and 3.
    c. Dequeue 2, print it, and enqueue its unvisited neighbor 4.
    d. Dequeue 3, print it, and enqueue its unvisited neighbor 4.
    e. Dequeue 4, print it (no unvisited neighbors), and the queue becomes empty.
4. Sample output for the BFS traversal starting from node 1: 1 -> 2 -> 3 -> 4
5. This output represents the order in which nodes are visited during BFS traversal starting from node 1.

## VIVA QUESTIONS:

**a)** What is BFS (Breadth-First Search) and where is it used in computer science?
**b)** Explain the data structure used to implement BFS.
**c)** How does BFS differ from DFS (Depth-First Search)?
**d)** What is the time complexity of BFS on a graph with V vertices and E edges?
**e)** Describe an application where BFS can be used effectively.

## PROGRAM NO. 7

**OBJECTIVE:** Write a program to implement Linear Search and Binary Search Techniques.

**ALGORITHM:**

**LINEAR SEARCH**

```
Algorithm linsrch (a[], x)
{
  // a[1:n] is an array of n elements index:= 0; flag:= 0;
  while (index < n) do
  {
  if (x = a[index]) then
  {  flag:= 1; break;
  }
  index ++;
  }
  if(flag =1)
  write("Data found ");
  else
  write("Data not found ");
}
```

**Example:** Given a list of n elements and search a given element x in the list using linear search.

**a.** Start from the leftmost element of list a[] and one by one compare x with each element of list a[].

**b.** If x matches with an element, return the index.

**c.** If x doesn't match with any of elements, return -1.

Consider a list with 10 elements and search for 9.

a = [56, 3, 249, 518, 7, 26, 94, 651, 23, 9]

| **Index →** | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** |
|---|---|---|---|---|---|---|---|---|---|---|
| **Iteration 1** | 56 | 3 | 249 | 518 | 7 | 26 | 94 | 651 | 23 | 9 |
| **Iteration 2** | 56 | 3 | 249 | 518 | 7 | 26 | 94 | 651 | 23 | 9 |
| **Iteration 3** | 56 | 3 | 249 | 518 | 7 | 26 | 94 | 651 | 23 | 9 |
| **Iteration 4** | 56 | 3 | 249 | 518 | 7 | 26 | 94 | 651 | 23 | 9 |
| **Iteration 5** | 56 | 3 | 249 | 518 | 7 | 26 | 94 | 651 | 23 | 9 |
| **Iteration 6** | 56 | 3 | 249 | 518 | 7 | 26 | 94 | 651 | 23 | 9 |
| **Iteration 7** | 56 | 3 | 249 | 518 | 7 | 26 | 94 | 651 | 23 | 9 |
| **Iteration 8** | 56 | 3 | 249 | 518 | 7 | 26 | 94 | 651 | 23 | 9 |
| **Iteration 9** | 56 | 3 | 249 | 518 | 7 | 26 | 94 | 651 | 23 | 9 |

| Iteration 10 | 56 | 3 | 249 | 518 | 7 | 26 | 94 | 651 | 23 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|

**SAMPLE OUTPUT:**

```
Enter list:
21 2 43 13 5 46 42 63
Enter the search element:43
The element found at 3 position
```

```
Enter list:
21 423 5231 32 12 52 13
Enter the search element:323
Element not found
```

**ALGORITHM:**

**BINARY SEARCH**

```
Algorithm binsrch (a[], n, x)
{
// a[1:n] is an array of n elements low = 1;
high = n;
while (low < high) do
{
mid = (low + high)/2 ;
if (x < a[mid]) then
high = mid – 1;
else if (x > a[mid]) then
low = mid + 1;
else
return mid;
}
return 0;
}
```

**Example:** Given a sorted list of a[] of n elements, search a given element x in list.

a. Search a sorted list by repeatedly dividing the search interval in half. Begin with an interval covering the whole list.
b. If the search key is less than the item in the middle item, then narrow the interval to the lower half. Otherwise narrow it to the upper half.
c. Repeat the procedure until the value is found or the interval is empty.

Consider a sorted list a[] with 9 elements and the search key is 31.

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| 11 | 23 | 31 | 33 | 65 | 68 | 71 | 89 | 100 |

Let the search key = 31.

First low = 0, high = 8, mid = (low + high) = 4, a[mid] = 65 is the center element, but 65 > 31.

So now high = mid - 1= 4 - 1 = 3, low = 0, mid = (0 + 3) / 2 = 1

a[mid] = a[1] = 23, but 23 < 31.

Again low = mid +1 = 1 +1 =2, high = 3, mid = (2 + 3) /2 = 2

a[mid] = a[2] = 31 which is the search key, so the search is successful.

**SAMPLE OUTPUT**

```
Enter list:
12 32 14 53 5 767 52 24 46
The sorted list is [5, 12, 14, 24, 32, 46, 52, 53, 767]
Enter the search element:24
The element found at 4 position
```

```
Enter list:
12 32 14 53 5 767 52 24 46
The sorted list is [5, 12, 14, 24, 32, 46, 52, 53, 767]
Enter the search element:12
The element found at 2 position
```

**VIVA QUESTIONS:**

**a)** What is linear search, and how does it work on an array?

**b)** Explain the concept of binary search and its requirements for operation.

**c)** Discuss the time complexity of linear search and binary search on a sorted array.

**d)** Can binary search be used on an unsorted array? Why or why not?

**e)** When would you prefer to use linear search over binary search or vice versa?

# PROGRAM NO. 8

**OBJECTIVE:** Write a program to implement following Bubble Sort Technique.

## THEORY:

Sorting refers to arranging data in a particular format. Sorting algorithm specifies the way to arrange data in a particular order. Most common orders are in numerical or lexicographical order.
The importance of sorting lies in the fact that data searching can be optimized to a very high level, if data is stored in a sorted manner. Sorting is also used to represent data in more readable formats.

Following are some of the examples of sorting in real-life scenarios −

- **Telephone Directory** − The telephone directory stores the telephone numbers of people sorted by their names, so that the names can be searched easily.
- **Dictionary** − The dictionary stores words in an alphabetical order so that searching of any word becomes easy.

## Bubble Sort Algorithm

```
Algorithm  bubblesort ( x[],  n)
        {
        // x[1:n] is an array of n elementsfor i:= 0 to n do
                {
                        for j:= 0 to  n–i-1 do
                        {
                          if (x[j] > x[j+1])
                           {
                                   temp = x[j];
                                   x[j] = x[j+1];
                                   x[j+1] = temp;
                           }
                }     }

        }
```

**Example:** Bubble Sort is the simplest sorting algorithm that works by repeatedly swapping the adjacent elements if they are not in order.

| | | | | | |
|---|---|---|---|---|---|
| **First Pass** | | | | | |
| 5 | 1 | 4 | 2 | 8 | Compare the first two elements, and swaps since 5 > 1 |
| 1 | 5 | 4 | 2 | 8 | Compare 5 and 4 and swap since 5 > 4 |
| 1 | 4 | 5 | 2 | 8 | Compare 5 and 2 and swap since 5 > 2 |
| 1 | 4 | 2 | 5 | 8 | Compare 5 and 8 and since 5 < 8, no swap. |
| **Second Pass** | | | | | |
| 1 | 4 | 2 | 5 | 8 | Compare the first two elements, and as 1 < 4, no swap. |
| 1 | 4 | 2 | 5 | 8 | Compare 4 and 2, swap since 4 > 2 |
| 1 | 2 | 4 | 5 | 8 | Compare 4 and 5, no swap. |
| 1 | 2 | 4 | 5 | 8 | Compare 5 and 8 and no swap. |
| **Third Pass** | | | | | |
| 1 | 2 | 4 | 5 | 8 | Compare the first two elements and no swap. |
| 1 | 2 | 4 | 5 | 8 | Compare 2 and 4, no swap. |
| 1 | 2 | 4 | 5 | 8 | Compare 4 and 5, no swap. |
| 1 | 2 | 4 | 5 | 8 | Compare 5 and 8, no swap. |
| **Fourth Pass** | | | | | |
| 1 | 2 | 4 | 5 | 8 | Compare the first two elements and no swap. |
| 1 | 2 | 4 | 5 | 8 | Compare 2 and 4, no swap. |
| 1 | 2 | 4 | 5 | 8 | Compare 4 and 5, no swap. |

1 2 4 5 8 Compare 5 and 8, no swap.

**SAMPLE OUTPUT**

**BUBBLE SORT:**

```
Enter elements into list:
1 4 2 3 5 7 2 11 23 45 231
The sorted list is [1, 2, 2, 3, 4, 5, 7, 11, 23, 45, 231]
```

## VIVA QUESTIONS:

**a)** Define bubble sort and explain how it sorts an array.

**b)** What is the time complexity of bubble sort on an array of N elements?

**c)** Can you describe a scenario where bubble sort performs better than other sorting algorithms like merge sort or quicksort?

**d)** Explain the idea of swapping adjacent elements in the array during each pass in bubble sort.

**e)** Discuss the limitations of bubble sort and suggest ways to improve its efficiency.

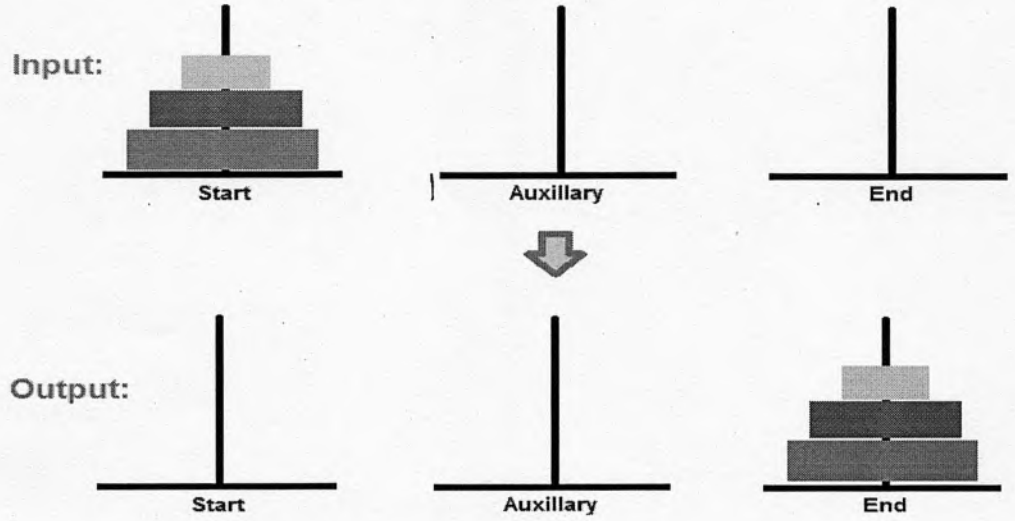
## ADDITIONAL PROGRAM

**OBJECTIVE:** Write a program to demonstrate working of Tower of Hanoi.

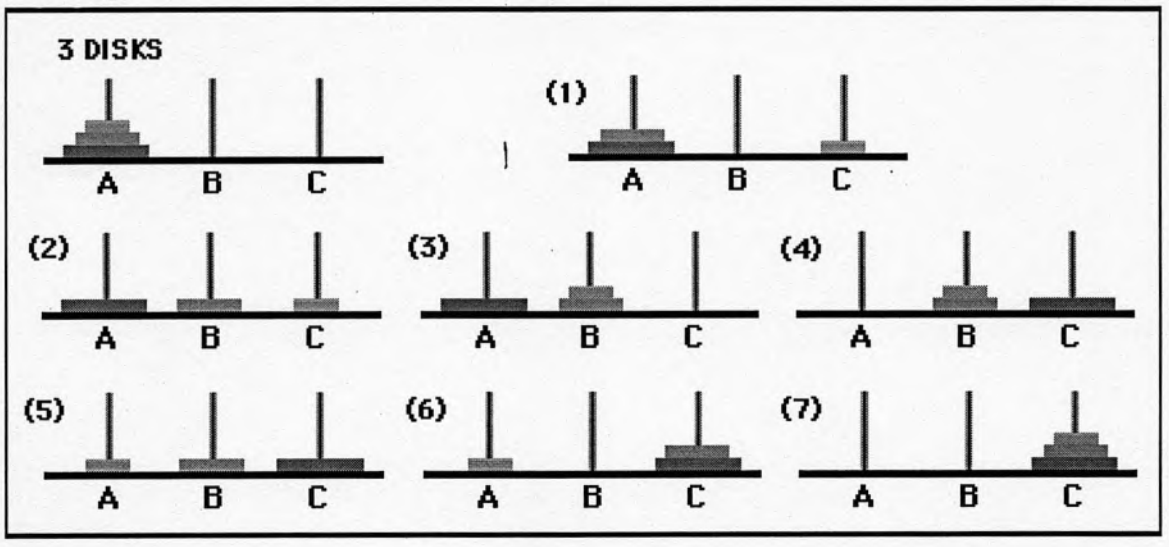
## THEORY:

Tower of Hanoi is a mathematical puzzle where we have three towers and n disks. The objective of the puzzle is to move the entire stack to another tower, obeying the following simple rules:

1. Only one disk can be moved at a time.
2. Each move consists of taking the upper disk from one of the stacks and placing it on top of another stack i.e. a disk can only be moved if it is the uppermost disk on a stack.
3. No disk may be placed on top of a smaller disk.




**Internal Steps**




**ALGORITHM:**

```
void TOH(int n, int A, int B, int C)
{
        if (n>0)
        {
```

```
            TOH(n-1, A, C, B);
            printf("Move a disc from %d to %d", A, C);
            TOH(n-1, B, A, C);
        }
}
```

**SAMPLE OUTPUT:**

```
Enter the no of disks:3

Move a disk from A to C
Move a disk from A to B
Move a disk from C to B
Move a disk from A to C
Move a disk from B to A
Move a disk from B to C
Move a disk from A to C
 No of steps is 7:
------------------------
```

## Important VIVA Questions

1. What is a Data Structure?
2. What are linear and non-linear data Structures?
3. What are the various operations that can be performed on different Data Structures?
4. What is Stack and where it can be used?
5. What is a Queue, how it is different from the stack and how is it implemented?
6. What is Infix, prefix, Postfix notations?
7. What is a Linked List and What are its types?
8. Which data structures are used for BFS and DFS of a graph?
9. How to implement a stack using queue?
10. Which Data Structure Should be used for implementing LRU cache?
11. What is the worst-case time for quick sort to sort an array of n elements?
12. When is a binary search best applied?
13. What are multidimensional arrays?
14. What is merge sort?
15. Differentiate NULL and VOID
16. What is Data abstraction?
17. Which sorting algorithm is considered the fastest?
18. What is a dequeue?
19. What is a graph?
20. What is Fibonacci search?

# HINDUSTAN COLLEGE OF SCIENCE & TECHNOLOGY, MATHURA

CLASS B.Tech 3 sem SECTION / BRANCH IT BATCH A1

| S. No. | Roll No. | Name of Student | Attendance Date | 17/9 | 1/10 | 15/10 | 5/11 | 12/11 | 17/11 | 25/11 | 1/12 | 8/12 | 15/12 | 22/12 | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | No. | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| | 2100640130001 | Abhishek Nohwar | | A | A | A | A | A | A | A | A | A | A | A | | | | |
| | 2 | Adarsh Gautam | | A | A | A | A | 1 | 2 | A | A | A | A | 3 | | | | |
| | 3 | Aditya Pratap Singh | | A | A | 1 | A | A | A | 2 | 3 | A | 4 | 5 | | | | |
| | 4 | Aditya Rajpal | | A | A | A | A | A | 1 | 2 | A | 3 | A | 4 | | | | |
| | 5 | Akash Thakur | | A | A | A | A | A | A | A | A | A | A | A | | | | |
| | 6 | Akshit | | A | A | 1 | 2 | 3 | 4 | A | 5 | A | A | 6 | | | | |
| | 7 | Aman Chaudhary | | A | A | A | A | A | A | A | 1 | A | 2 | 3 | | | | |
| | 8 | Aranya Dubey | | 1 | A | 2 | 3 | 4 | 5 | A | 6 | 7 | 8 | 9 | | | | |
| | 9 | Anjali Chaturvedi | | A | A | A | A | A | A | A | A | A | A | A | | | | |
| | 10 | Ankit Arora | | A | A | 1 | A | 2 | A | A | A | 3 | A | 4 | | | | |
| | 11 | Ankit Gautam | | A | A | A | A | A | 1 | A | A | A | A | A | | | | |
| | 12 | Ashish Garg | | A | A | A | A | A | A | A | A | A | A | A | | | | |
| | 13 | Aslam | | A | A | A | A | A | A | A | A | A | A | A | | | | |
| | 14 | Astha | | 1 | A | 2 | 3 | 4 | 5 | A | 6 | A | A | 7 | | | | |
| | 15 | Atharv Jadon | | A | A | A | A | A | A | A | A | A | A | A | | | | |
| | 16 | Bhavnesh Sharma | | A | A | A | A | A | A | A | A | A | A | A | | | | |
| | 17 | Bishakha Kumari Shaw | | 1 | A | 2 | A | 3 | 4 | A | 5 | 6 | A | 7 | | | | |
| | 18 | Chetan Chaudhary | | A | A | A | A | A | A | A | A | A | A | A | | | | |
| | 19 | Devesh Kumar | | 1 | 2 | A | A | 3 | 4 | A | 5 | 6 | A | 7 | | | | |
| | 20 | Dheeraj Yadav | | A | A | A | A | A | 1 | A | A | A | A | A | | | | |
| | 21 | Dhwani Agarwal | | A | A | 1 | 2 | 3 | 4 | 5 | A | 6 | A | 7 | | | | |
| | 22 | Diwakar Goswami | | A | A | A | A | A | A | A | A | A | A | A | | | | |
| | 23 | Diya Gupta | | A | 1 | 2 | A | 3 | 4 | 5 | 6 | 7 | 8 | 9 | | | | |
| | 24 | Gaurav Raj | | A | A | 1 | 2 | 3 | 4 | 5 | 6 | 7 | A | 8 | | | | |
| | 25 | Govind Tyagi | | A | A | 1 | 2 | A | 3 | 4 | A | 5 | A | 6 | | | | |
| | 26 | Harsh Sharma | | A | A | A | 1 | A | A | A | A | A | A | A | | | | |
| | 27 | Harsh Tripathi | | A | A | A | 1 | 2 | A | 3 | A | 4 | A | 5 | | | | |
| | 28 | Harshul Chawla | | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | | | | |
| | 29. | Himanshu Singh | | A | A | A | A | A | A | A | A | A | A | A | | | | |
| | 30 | Kriti Maheshwari | | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | | | | |
| | | | | 6 | 4 | 12 | 10 | 13 | 15 | 9 | 11 | 12 | 6 | 17 | | | | |

Sign. of Faculty

Head
Department of Information Technology
Hindustan College of Science & Technology
Farah, Mathura

Sign. of H.O.D.

# HINDUSTAN COLLEGE OF SCIENCE & TECHNOLOGY, MATHURA

CLASS B.Tech. 3Sem SECTION / BRANCH IT BATCH A2

| S. No. | Roll No. | Name of Student | Attendance | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | Date | 9/9 | 16/9 | 23/9 | 30/9 | 7/10 | 4/10 | 4/11 | 11/11 | 18/11 | 25/11 | 2/12 | 9/12 | 23/12 | | |
| | | No. | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| | 31 | Kunal | A | A | A | A | A | A | A | A | A | A | A | A | A | | |
| | 32 | Manjeet Singh | A | A | A | A | A | A | A | A | A | A | A | A | A | | |
| | 33 | Monu Sikarwar | A | A | A | A | A | A | A | A | A | A | A | A | A | | |
| | 34 | Nikhil Rathore | A | 1 | 2 | 3 | 4 | 5 | 6 | A | A | 7 | 8 | A | A | | |
| | 35 | Nikunj Maheshwari | A | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | A | 11 | | |
| | 36 | Nishant Dixit | A | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | A | 9 | A | 10 | | |
| | 37 | Nishkarsh Rathore | A | 1 | A | A | A | A | A | 2 | 3 | 4 | A | A | A | | |
| | 38 | Nitesh Kr. Sharma | A | A | A | A | A | A | A | A | A | A | A | A | A | | |
| | 39 | Palak Sharma | 1 | A | A | 2 | 3 | 4 | 5 | 6 | A | 7 | 8 | 9 | 10 | | |
| | 41 | Parth Singh Sikarwar | A | A | A | A | A | 1 | 2 | A | 3 | A | A | A | A | | |
| | 42 | Prashant Pal Singh | A | A | A | A | A | A | 1 | 2 | A | 3 | 4 | 5 | 6 | | |
| | 43 | Prashant Upreti | A | 1 | 2 | A | A | 3 | 4 | 5 | A | A | 6 | 7 | 8 | | |
| | 44 | Prateek Chaudhary | A | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | A | A | | |
| | 45 | Pratiom Yadav | A | A | A | A | A | A | A | A | A | A | A | A | A | | |
| | 46 | Priyanshi Gupta | 1 | A | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | | |
| | 48 | Priyanshu Verma | A | A | A | A | A | 1 | 2 | 3 | A | A | 4 | A | A | | |
| | 49 | Rohit Kumar Sharma | A | 1 | A | A | A | 2 | A | 3 | 4 | A | A | 5 | A | | |
| | 50 | Sachin Gautam | A | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | A | 11 | | |
| | 51 | Sahil Ali | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | A | 9 | 10 | 11 | 12 | | |
| | 52 | Sakar Talwar | 1 | A | A | 2 | 3 | A | 4 | 5 | A | A | 6 | 7 | 8 | | |
| | 53 | Satyam Singh | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | | |
| | 54 | Shivani | 1 | 2 | 3 | A | A | A | A | A | 4 | A | 5 | 6 | A | | |
| | 55 | Siddharth Jadon | A | A | A | A | A | A | A | A | A | A | A | A | A | | |
| | 56 | Suryansh Singh | 1 | 2 | 3 | A | A | 4 | 5 | 6 | A | A | A | 7 | A | | |
| | 57 | Vait Kukreti | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | A | 9 | 10 | 11 | 12 | | |
| | 58 | Vikash Solanki | 1 | 2 | 3 | 4 | A | 5 | 6 | 7 | 8 | 9 | 10 | A | 11 | | |
| | 60 | Vishesh Sharma | A | A | A | A | A | A | A | A | A | A | A | A | A | | |
| | | Prateek Mulgal | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | A | A | A | 9 | 10 | | |
| | | Priyanshi Chaudhary | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | 10 | 11 | 12 | | |
| | | Shivam Sarot | A | 1 | 2 | A | 3 | 4 | 5 | A | 6 | A | 7 | A | 8 | | |
| | | Devrishi | A | A | A | A | A | A | A | A | A | A | A | A | A | | |
| | | Kushagra Kulshreshtha | | | | | 1 | 2 | A | 3 | 4 | 5 | 6 | 7 | 8 | | |
| | | Mohit Sharma | | | | | 1 | A | A | 2 | 3 | 4 | 5 | A | A | | |
| | | Gagandeep Baghel | | | | | 1 | 2 | A | 3 | A | A | A | A | A | | |
| | | Sudhanshu Vaibhav | | | | | | | | 1 | 2 | A | 3 | A | | | |
| | | | 11 | 17 | 16 | 14 | 17 | 21 | 20 | 22 | 16 | 15 | 20 | 16 | 16 | | |

Sign. of Faculty

Sign. of H.O.D

Head
Department of Information Technology
Hindustan College of Science & Technology
Farah, Mathura