Data Structure Final Solution

1) What is stack DS

A stack is a linear data structure that follows the Last In, First Out (LIFO) principle. This means the insertion and deletion are done at one end, called top.

2) Why we call stack is a LIFO DS

We call stack a LIFO (Last In, First Out) data structure because the last element that is added (pushed) into the stack is removed (popped) first.

3) Explain all operations of stack

Operation	Description
push(int data)	inserts *data* into stack
int pop()	removes and returns last inserted element
int top()	returns last inserted element
int size()	returns number of elements in stack
int isEmpty()	indicate if any element is stored or not
int isFull()	indicates if stack is full or not

4) Implementation all operations of Stack (code)

```c
#include <stdio.h>
#define size 10

int stack[10], top = -1;

void push(int element)
{

    if (top == size - 1)
    {
        printf("The stack is full");
        return;
    }
    else
    {
        top++;
        stack[top] = element;
    }
}

void pop()
{

    if (top == -1)
    {
        printf("The stack is empty");
```

```
            return;
        }
        else
        {
            printf("poped element is %d\n", stack[top]);
            top--;
        }
    }

    void display()
    {
        if (top == -1)
        {
            printf("The stack is empty");
            return;
        }

        else
        {
            int p = top;

            while (p >= 0)
            {
                printf("%d ", stack[p]);
                p--;
            }
        }
    }

    int main()
    {
        push(34);
        push(23);
        push(58);
        push(95);

        pop();

        display();

        return 0;
    }
```

5) What is queue DS

A queue is a linear data structure that follows the First In, First Out (FIFO) principle. This means the first element added to the queue will be the first one to be removed.

There are 4 type of queue

1) Linear queue
2) Circular queue
3) Double ended queue
4) Priority queue

6) Why we call queue is a FIFO DS

We call queue a FIFO (First In, First Out) data structure because the first element added to the queue is the first one to be removed**.**

7) Explain all operations of queue

- **Create**– Creates empty queue
- **Enqueue** – Add an item at the rear (end) of the queue.
- **Dequeue** – Remove an item from the front of the queue.
- **Front/Peek** – View the item at the front without removing it.
- **isEmpty** – Check if the queue is empty.
- **isFull**– Check if the queue is full

8) Implementation all operations of Queue (code)

```c
#include <stdio.h>
#define size 10

int queue[size];
int front = -1;
int rear = -1;

void insert(int element)
{
    if ((front == 0 && rear == size - 1) || (front > 0 && rear == front - 1))
    {
        printf("The Queue is full\n");
        return;
    }
    else if (rear == -1 && front == -1)
    {
        front = 0;
        rear = 0;
        queue[rear] = element;
    }
    else if (front != 0 && rear == size - 1)
    {
        rear = 0;
        queue[rear] = element;
    }
    else
    {
        rear++;
        queue[rear] = element;
 }
}

void delete()
{
    if (rear == -1 && front == -1)
```

```
    {
        printf("The Queue is empty\n");
    }
    else if (rear == front)
    {
        rear = front = -1;
    }
    else if (front == size - 1)
    {
        front = 0;
    }
    else
    {
        front++;
    }
}

void display()
{
    if (rear == -1 && front == -1)
    {
        printf("The Queue is empty\n");
    }
    else if (rear >= front)
    {
        for (int i = front; i <= rear; i++)
        {
            printf("%d ", queue[i]);
        }
    }
    else
    {
        for (int i = front; i < size; i++)
        {
            printf("%d ", queue[i]);
        }
        for (int i = 0; i <= rear; i++)
        {
            printf("%d ", queue[i]);
        }
    }
    printf("\n");
}

int main()
{
    insert(87);
    insert(69);
    insert(37);
    insert(97);
    delete();
    delete();
    insert(12);

    display();

    return 0;
}
```

9) Difference between stack and queue DS

Stack	Queue
Stack is a LIFO (Last in first out) data structure.	Queue is a FIFO (First in first out) data structure.
In a stack, all insertion and deletions are permitted only at one end of stack called top of stack.	In a queue items are inserted at one end called the rear and deleted from the other end called the front.
Stack is widely used in recursion.	Queue is more useful in many of the real life applications.
Example: A stack of plates, a stack of coin etc.	Example: A queue of people waiting to purchase tickets, in a computer system process waiting for line printer etc.
Function calls, undo, backtracking	Scheduling, Keyboard Input Buffer, messaging

10) Uses of stack and queue DS

Uses of stack data structure:

- Memory management: Store function info in stack memory
- History of a web browser
- Implementing function calls
- Recursion
- Tree Traversal
- Undo/Redo : Track user actions
- Backtracking: Explore and revert steps
- Data reversing: Reverse strings or arrays

Uses of Queue data structure:

- CPU Scheduling : Manage tasks waiting for processor
- Keyboard Input Buffer: Store key presses in order
- Messaging Systems: Deliver messages in order
- Streaming/Network: Process incoming data packets
- Breadth-First Search(BFS): Traverse graphs level by level

11) What is structure variable

In programming , a structure is a user-define data type that allow to combine data items of different data type . Its like a custom container that can hold multiple variables

12) What is Dynamic memory allocation? (malloc, calloc, free function)

Dynamic memory allocation is the process of allocating memory during the runtime of a program instead of at compile time.

Malloc: The malloc( ) function in C is used to **dynamically allocate memory** during program execution.

Calloc: calloc is another memory allocation function , similar to malloc . But it not only allocates memory but also initializes all the allocation memory to zero.

Free: The free( ) function is used to **release** **dynamically allocated memory** back to the system.

13) What is Link List? Types of Linked List

A **linked list** is a **linear data structure** where elements (called **nodes**) are stored in **separate memory locations**, and each node **points to the next one**.

There are 3 types of Link List

i. Single Linked List
ii. Double Link List
iii. Circular Link List

14) All operations for single Linked List (code)

15) Basic theory of Double , Circular Linked List

16) Difference between array and Linked List

Array	Link List
Array is fixed size	Link List is dynamic size
Insertion and deletion is inefficient	Insertion and deletion is efficient
Random access is efficient	Random access is inefficient
Sequential access is faster because its elements in contiguous memory location	Sequential access is slow because its element in not contiguous memory location
Element accessing time O(1)	Element accessing time O(n)
No memory waste if the array is full otherwise it waste memory to much	No chance to west memory because is allocate memory dynamically
Memory uses is efficient because it store only elements	Memory uses is inefficient because it store also pointer
Types: 1d, 2d , Multidimensional	Single , double, circular Link List

17) Difference between Linear DS and Non-Linear DS

Linear Data Structure

i) Data is stored in a sequential order.
ii) Traversed in a single level (one after another).
iii) Simple to implement and understand.
iv) Uses contiguous memory.
v) One-to-one relationship between elements.

Non-Linear Data Structure

i) Data is stored in a hierarchical or interconnected form.
ii) Traversal can be done in multiple levels or directions.
iii) More complex to implement and understand.
iv) Uses non-contiguous memory.
v) One-to-many or many-to-many relationships between elements.

18) What is tree DS

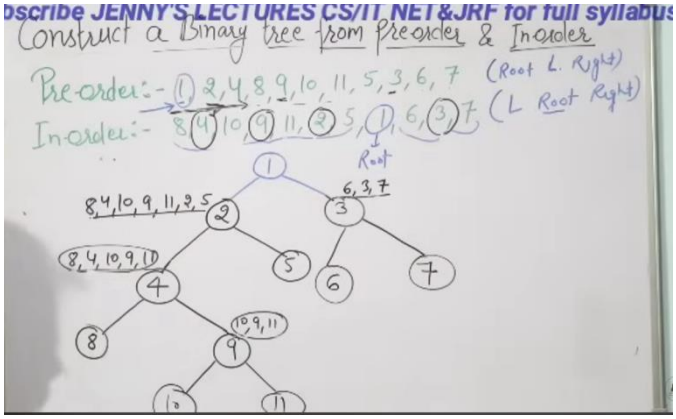
Tree ds is a type of data structure that organized data in a hierarchical manner .it start from a special node called the root , and each node can have one parent and multiple children node.

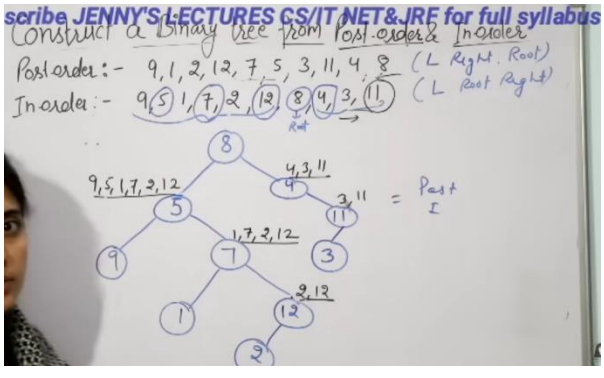
19) What is tree Traversal algorithm? Types of tree traversal

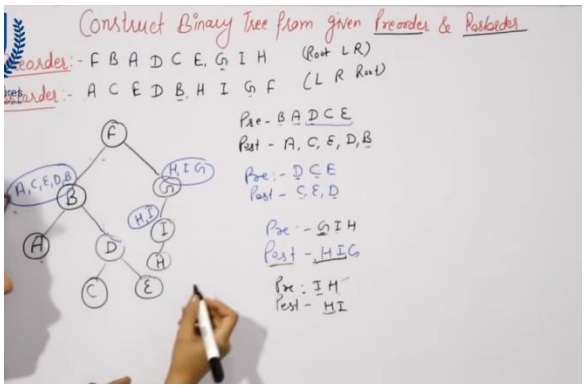
A **tree traversal algorithm** is a method used to **visit all the nodes** in a tree **in a specific order**.there are three types of tree traverse

1) Preorder
2) Inorder
3) Postorder

20) Convert one traversal order to another order. & draw tree diagram by given traversal order







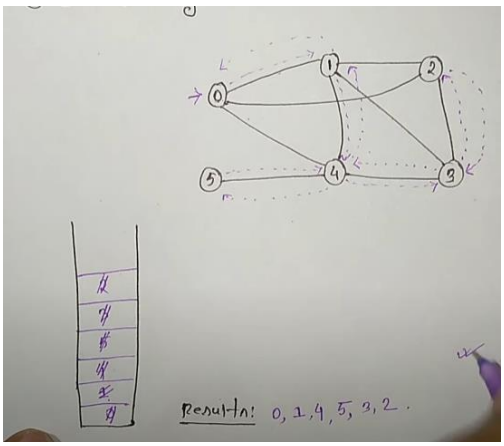
21) What is Graph DS

A graph is a type of data structure that consist of set of nodes and edges that connect pairs of nodes.

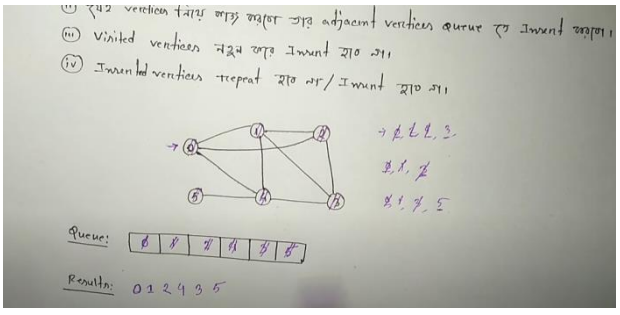
22) What is DFS, BFS algorithm

DFS (Depth-First Search) and **BFS (Breadth-First Search)** are two fundamental graph and tree traversal algorithms used in **data structures** and **algorithms**.

23) Determine the DFS & BFS from Graph



adjacent nodes



24) Why Tree & Graph considered a non-linear DS

The data in a tree are not stored in a sequential manner i.e., they are not stored linearly. Instead, they are arranged on multiple levels or we can say it is a hierarchical structure

```
void reverse()
{
    struct node *previourNode = NULL;
    struct node *currentNode = head;
    struct node *nextNode = head;

    while (nextNode != NULL)
    {
        nextNode = nextNode->next;
        currentNode->next = previourNode;
        previourNode = currentNode;
        currentNode = nextNode;
    }

    head = previourNode;
}
```