Page No.
Date

* Data Structure

* Data structure is a way to store or organizing a data *

→ The data structure name indicates itself that organizing the data in memory. There are many ways of organizing the data in the memory as we have already seen one of the data structure, i.e array in c language. Array is a collection of memory elements in which data is stored sequentially, i.e, one after ~~stor~~ another. In other world, we can say that array stored the elements in continuous manner. This organization of data is done with the help of an array of data structure.

The data structure is not any programing language like C, C++, java etc. It is a set of algorithms that we can use in any programing language to struct the data in the memory.

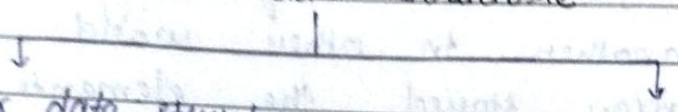
* Types of data structure

- Primitive data structure
- Non - Primitive data structure

- Primitive data structure -

The primitive data structure are primitive data types. The int, char, float, double and pointer are the primitive data structure that can hold a single value.

- Non - primitive data structure -



- Linear data structure

The arrangement of data in a sequential manner is known as a linear data structure. The data structure used for this purpose are array, linked list, stacks, and queue. In these data structure one element is connected to only one another element in a linear form.

- Non - linear data structure -

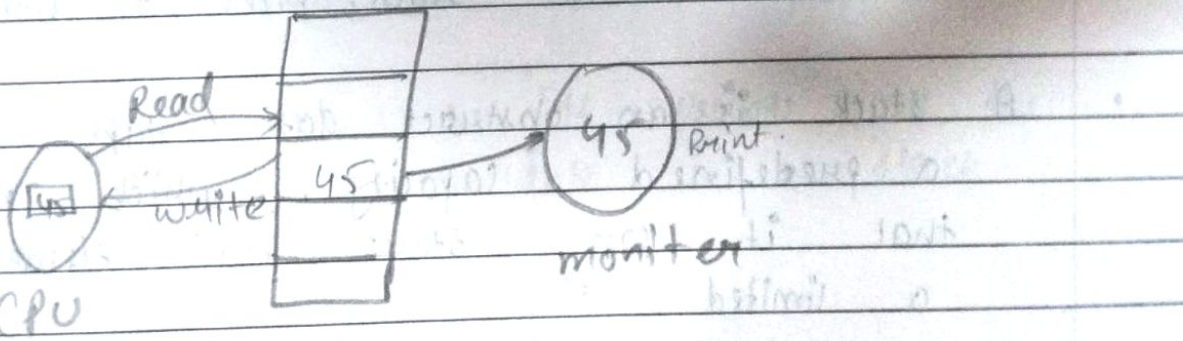
When one element is connected to the 'n' number of elements known as a non - linear data structure. The best example is trees and graphs. In this case, the elements are arranged in a random manner.

* Data structure can also classified as -

- Static data structure - It is a type of data structure, where the size is allocated at the compile time. Therefore, the maximum size is fixed
- Dynamic data structure - It is a type of data structure, where the size of ds is allocated at the run time Therefore, the maximum size is flexible.

→ Static - value cannot be change at run time.

→ dynamic - Able to change value at run time.



Static concept

* Stack *

A stack is a linear data structure that follow the LIFO (Last in First out) principle. Stack has one end. It contain only one pointer top pointer. pointing the topmost element of the stack. whenever an element is added in the stack, it is added on the top of the stack, and the element can be deleted only from the stack. In other words, a stack can be define as a container in which insertion and deletion can be done from the one end known as the top of the stack.

- It is called stack. because it behaves like a real-world stack, piles of book, etc.
- A stack is an abstract data type with a predefined capacity, which means that it can store the element of a limited size.
- It is a data structure that follow some order to insert and delete the elements, and the corder can be ~~fifo~~ LIFO or FILO

* Operations on stack.

- push() : when we insert an element in a stack then the operation is known as a push. If the stack is full then the overflow condition occurs.
- pop() : when we delete an element from the stack, the operation is known as pop. If stack is empty means that no element exists in stack the state is known as ~~over~~. underflow state.
- isEmpty() : It determines whether the stack is empty or not.
- isFull() : It determine whether the is full or not
- peek() : It return the element at given position.
- size() ; It return the size of stack.
- Display() / Transverse() : It prints all the element available in a stack.

Date

Stack program

```
int stack[10];
int top=-1;
int size=10;

void push(int data)
{
    if(top==size-1)
    {
        cout<<"stack is full";
    }
    else
    {
        stack[++top]=data;
    }
}
int pop()
{
    int temp;
    if(top==-1)
    {
    cout<<"stack is empty";
    }
    else
    {
        temp= stack[top--]
    }
        return temp;

}
```

```
void display()
{   int i;
    if (top==-1)
        cout<<" stack is empty";

    else
    {
        for(i=0 ; i<=top ; i++)
        {
            cout<< stack[i];
        }
    }

}
int size()
{
        return top+1;
}

void main()
{   char ch;
    int choise;
do {
    cout<<" Enter 1 for Push \n Enter 2 for pop \n
            Enter 3 for display \n Enter 4 for size";

        cin>> choise;

  switch (choise)
  {
     case 1:
     {  int data;
```

```
        cout << " Enter a data " ;
        cin >> data ;

        Push (data) ;
        break ;
    }
    case 2 :
    {   int data ;
        data = pop() ;
        cout << " Deleted data is " << data ;
        break ;
    }
    case 3 :
    {
        display () ;
        break ;
    }
    case 4 :
    {
    cout << " size of stack is " << size() ;
        break ;
    }
    default ;

    cout << " Enter c for continue " ;
    cin >> ch ;
}
}while (ch == 'c') ;
```

Page No.
Date / /

Queue

- A queue can be defined as an ordered list which enables insert operations to be performed at one end called REAR and delete operations to be performed at another end called FRONT.
- Queue is referred to be as First in First out list. (FIFO)
- For example, People waiting in line for a rail ticket form a queue.

→ Whatever comes first will go out first. It follow FIFO policy. In queue insertion is done from one end known as the Rear or end or tail of the queue. Whereas the deletion is done from another end known as the front end or the head of the queue.

* Operations on Queue -

- Enqueue - The enqueue operation is used to insert the element at the rear end of the queue

- Dequeue : The dequeue operation perform the deletion from the front-end of the queue. It also return the element which has been removed from the front-end. It ~~int~~ return an integer value. The dequeue operation can also be designed to void.

- Queue Overflow (isfull) : when the queue is completely full, then it show the overflow condition.

- Queue underflow (isempty) : when the queue is empty. ie no element are in the queue then it throws the underflow condition.

- Peek : This is the * return the element, which is pointed by the front pointer in the queue but does not delete it.

There are four types of Queue.

1. linear queue - In linear Queue, an insertion takes place from one end while the deletion occurs from another end. The end at which the insertion takes place known as the rear end. and the end at which the deletion takes place is known as front end. It strictly follow the FIFO rule.

Circular queue - In circular queue, all the nodes are represented as circular. It is similar to the linear Queue except that the last element of the queue is connected to the first element. It is also known as Ring Buffer as all the end are connected to another end. The drawback that occurs in a linear queue is overcome by using the circular queue If the empty space is available in a circular queue, the new element can be added in an empty space by simply incrementing the value of rear.

Priority Queue -

Deque

* Linked list *

↳ is a dynamic array

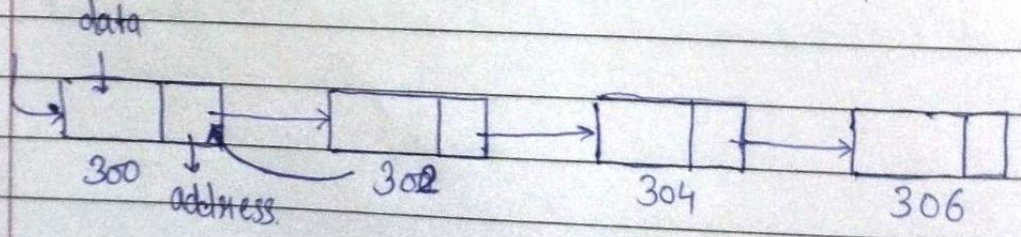
A linked list is also a collection of element but the elements are not stored in a consecutive location.

A linked list is can also be define as the collection of the nodes in which one node is connected to another node, and node consists of two parts, ie one is the data part and the second one is the address part.

We give the address of the first node a special name called head. Also, the last node in the linked list can be indentified because its next portion points to NULL.

Types of linked list -

- Singly linked list
- Doubly linked list
- Circular linked list
- Doubly circular linked list.



- singly linked list -
  It is the com most common.
  Each node has data and a pointer to the next node.

ode is represent as -

```
struct node {
    int data;
    struct node * next;
};
```

- Doubly linked list -
  we add a pointer to the previous node in a doubly linked list. Thus we can go in either direction: forward or backward

esentation -

```
struct node
{   int data;
    struct node * next;
    struct node * prev;
};
```

Circular linked list -
A circular linked list is a variation of a linked list in which the element is linked to the first element. This forms a circular loop.

Linked List -

```
struct node
{
        int data;
        node * next;
} head = NULL;

void finsertion ( int data)
{
    struct node * newnode ; * head ;
newnode = (struct node*) malloc (size of (struct node));

if (head == NULL)
  {
      head=newnode ;

  }
else
  {
        newnode -> data = data
         newnode -> next = head ;
           head = newnode ;

  }

int fdeletion ()
{   int data ;
   if (head == NULL)
     {
        cout << " list is Empty " ;

     }
```

free(node)

```
else
{
    data = head -> data;
    head = head -> next;
    return data;
}

void Insertion (int data)
{
    node * temp;
    temp = head;

newnode = (struct node *) malloc (size of (struct node));

if ( temp == NULL)
{
  headtemp = newnode;
}
else
{
  while ( temp -> next != NULL)
  {
      temp = temp -> next;
  }
    temp -> next = newnode;
    newnode -> data = data;
    newnode -> next = NULL;
}
```

```
int Ldeletion()
{   int data;
    node* temp;
    temp = head;

    if (temp == NULL)
    {   cout << "list is empty";
    }

    else if (temp -> next == NULL)
    {
        temp -> next = newnode;
    }
    else
    {   while (temp -> next -> next != NULL)
        {
            temp = temp -> next;
        }
            data = temp -> next -> data;
            temp -> next = NULL;
            return temp;
}
```

Page No.
Date

Doubly linked list -

Doubly linked list is a complex type of linked list in which a node contains a pointer to the previous as well as the next node in the sequence. Therefore in a doubly linked list, a node contains of three parts: node data, pointer to the next node in sequence (next pointer), pointer to the previous node (previous pointer). A sample

→
```
struct node
{
        node * prev ;
        int    data ;
        node * next ;
} head ;

void insertatbegin (int data)
{
    node * newnode
newnode = (struct node *) malloc (sizeof (struct node));

if (head == NULL)
{
    newnode → data = data ;
     newnode → prev = NULL ;
     newnode → next = NULL ;
        head = newnode ;
}
```

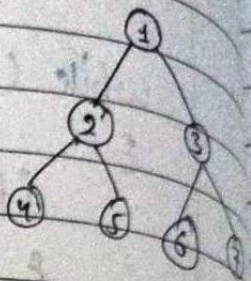
```
else
{       newnode -> data = data;
        newnode -> prev = NULL;
        newnode -> next = head;
        head -> prev = newnode;
        head = newnode;
}

int delete_at_first()
{
    int item;
if ( head == NULL)
{
    cout << "List is Empty";
}
else
if( head -> next == NULL)
{
    head item = head -> data;
        head = NULL;
}
    return item;
else
{
    item = head -> data;
    head = head -> next;
    head -> prev = NULL;
}
    return item;
}
```

```
void insert at last ( int data)
{
    node * newnode , *temp = head ;
newnode = (struct node *) malloc (size of (struct node));
if (head == NULL)
{
        newnode -> data = data ;
        newnode -> prev = NULL ;
        newnode -> next = NULL ;
}
else
{
      while ( temp -> next != NULL)
      {
          temp = temp -> next ;
      }
          temp -> next = newnode ;
           newnode -> prev = temp ;
            newnode -> next = NULL ;
}

int delete at last ()
{
    int item ;

if ( head == NULL)
{
    cout << " List is Empty " ;
}
```

```
else
{
    while ( temp -> next -> next != NULL)
    {
        temp = temp -> next;
    }
    item = temp -> next -> data;
    temp -> next = NULL;
}
return item;
```

* Tree

A tree is a nonlinear hierarchical data structure that consists of nodes connected by edges.



→ Tree terminologie -

1) ROOT → It is the topmost node of a tree

2) Node - A node is an entity that contains a key or value and pointers to its child nodes.

3. Edge - It is the link between any two nodes.

4. Height of node - The height of a node is the number of edges from the node of the deepest leaf (i.e. the longest path from the node to a leaf node).

5. Depth of a node - The depth of a node is the number of edges from the root to the node.

6. Height of a tree - The height of a Tree is the height of root node or the depth to the deepest node.

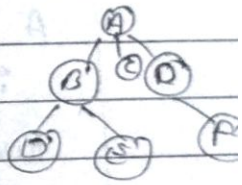
7. Degree of a Node - The degree of a Node is the total number of branches to that nodes.

8. Forest - A Collection of disjoint tree is called a forest
you can create a forest by cutting the root of a tree.

9 Terminal node - whose degree is zero. i.e the leaf node of a tree.

10 Non Terminal node - Nodes having a child nodes which i.e it has a degree.

11 Sibling - Here, siblings of B is C and D.



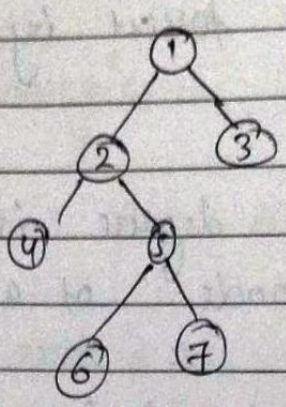
Types of Tree

1 Binary tree - A binary tree is a tree data structure in which each parent node can have at most two childeren. Each node of a binary tree consists of three items.

data item
address of left child
address of right child.

Type of Binary tree -

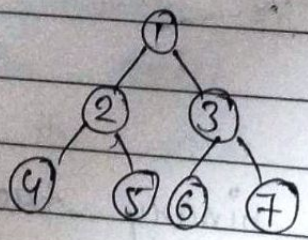
→ Full Binary - Tree - A full binary tree is a special type of binary tree in which every parent node / internal node has either two or no children



full binary tree

2 Perfect binary tree -

A perfect binary tree is a type of binary tree in which every internal node has exactly two child nodes and all the leaf nodes are at the same level.



Perfect binary tree

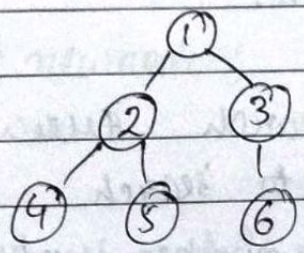
3 Complete binary tree -

A Complete binary tree is just like a full binary tree, but with two major differences.

1. Every level must be completely filled.

2. All the leaf element must lean towards the left.

3. The last leaf element might not have a right sibling i.e a complete binary tree does not have to be a full binary tree.



Complete binary tree.

Binary tree representation -

```
struct node
{
    int data;
    node * left;
    node * Right;
};
```

2. Binary search tree -

Binary search tree is a data structure that quickly allows us to maintain a sorted list of numbers.

- It is called a binary tree because each tree node has a maximum of two children.
- It is called a search tree because it can be used to search for the presence of a number in O(log(n)) time.

The properties that separate a binary search tree from a regular binary tree is

1. All nodes of left subtree are less than the root node.
2. All nodes of right subtree are more than the root node.
3. Bothe subtrees of each node are also BST. ie they have the above two properties.