DREAM STUDY

Smart Way of Easy Learning

Data Structure Using C & C++

(DSC)

www.dreamstudy.tk

dreamstudy123@gmail.com

Send us your query anytime!

BCA 2nd Year

Data Structure

using

C & C++

UNIT - 1

Introduction to data structure and its characteristics array

* Data Structure :- Data structure is representation of the logical relationship existing between individual elements of data.

Data structure is the way of organising data items. Data structure mainly specify the following four things -

1- Organisation of data
2- Accessing methods
3- Degree of associatibility
4- Processing alternatives for information.

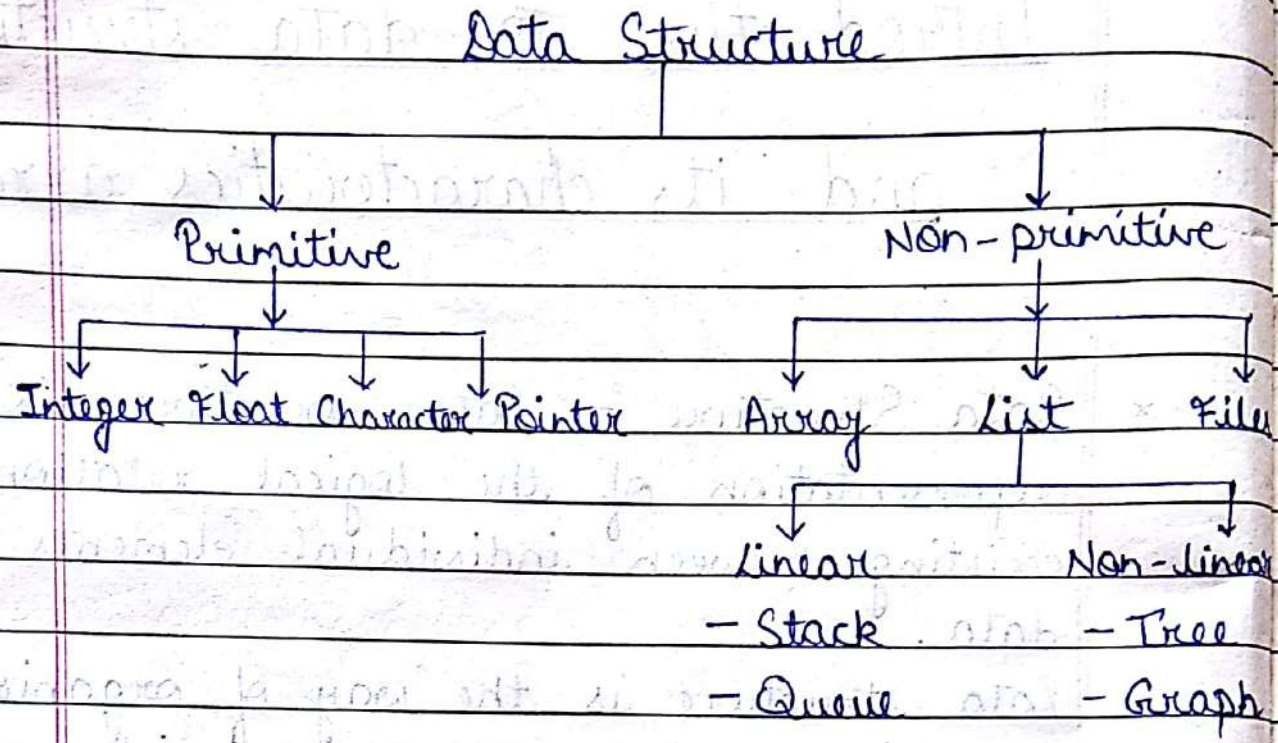
Operations of data structure :-

(i) Insertion
(ii) Deletion
(iii) Updation
(iv) Selection

Other operations -

(i) Searching
(ii) Sorting
(iii) Merging

Classification of data structure :-



* Array :- Array forms an important part of almost all programming languages. It provides a powerful feature and can be used as such or can be used to form complex data structure like - stack and queue.

An array can be define as an infinite collection of homogeneous (similar type) elements.

1. Arrays are always stored in consequtive memory locations.
2. An array can be store multiple values which can be refrenced by a single name unlike a simple variable which store one value at a time.
3. Array name is actually a pointer to

Date / /

the first location of the memory block allocated to the name of the array.

4. An array either can be an integer, character or floating data types, can be initialized only during declaration time and not afterward.

Types of array :-

1- One-dimensional array
2- Two-dimensional array
3- Multi-dimensional array

1- One-dimensional array :-

Initialization -

data_type variable_name [Expression];

Ex- int ex[10];
char word['H' 'E' 'L' 'L' 'O' '\0'];

Accessing One-dimensional array -

To read a value

scanf("%d", &[3]);

Ex-
```
void main()
{
int a[10], i;
clrscr();
printf("Enter the array");
for(i=0; i<=9; i++)
```

Date / /

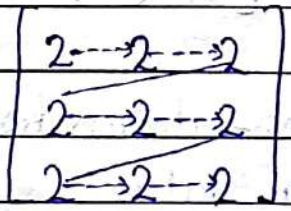
```
{
scanf ("%d", &a[i]);
}
for (i=0; i<9; i++)
{
printf ("%d\n", a[i]);
}
getch ();
}
```

2- Two-dimensional array:- Two-dimensional array is of two types - row major and column major.

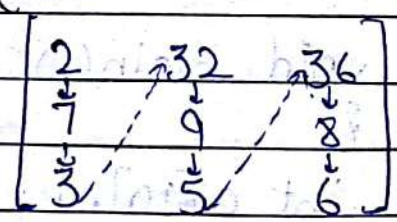
Initialization of 2-D array -

```
int a[2][2];
```

1) Row-major -



2) Column major -



Date / /

* Triangular matrix :-

1- Upper triangular matrix — A square matrix, all of whose elements below 'principal diagonal are zero, is called an upper triangular matrix.

$$\begin{bmatrix} 1 & 2 & 3 \\ 0 & 4 & 5 \\ 0 & 0 & 6 \end{bmatrix}$$

2- Lower triangular matrix — A square matrix, all of whose elements above principal diagonal are zero, is called a lower triangular matrix.

$$\begin{bmatrix} 1 & 0 & 0 \\ 2 & 3 & 0 \\ 4 & 5 & 6 \end{bmatrix}$$

* Sparse matrix :- In numerical analysis, a sparse matrix is in which most of the elements are 0.

Ex-

$$\begin{bmatrix} 0 & 1 & 0 & 0 \\ 3 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

* Dense matrix :-

$$\begin{bmatrix} 3 & 0 & 9 \\ 2 & 1 & 6 \\ 1 & 1 & 0 \end{bmatrix}$$

Date / /

* Tridiagonal matrix :- Matrix that has non-zero elements on the main diagonal.

Ex-

$$\begin{bmatrix} 2 & 3 & 4 & 0 \\ 1 & 5 & 6 & 7 \\ 0 & 1 & 2 & 3 \\ 0 & 1 & 6 & 1 \end{bmatrix} \qquad \begin{bmatrix} 3 & 4 & 0 \\ 1 & 2 & 5 \\ 0 & 2 & 3 \end{bmatrix}$$

Date / /

UNIT -2

Stack & Queue

★ **Stack :-**

1- Stack is a data structure used to store a collection of objects. Individual items can be added and stored in a stack using a push operation.

2- Objects are retrieved using a pop operation.

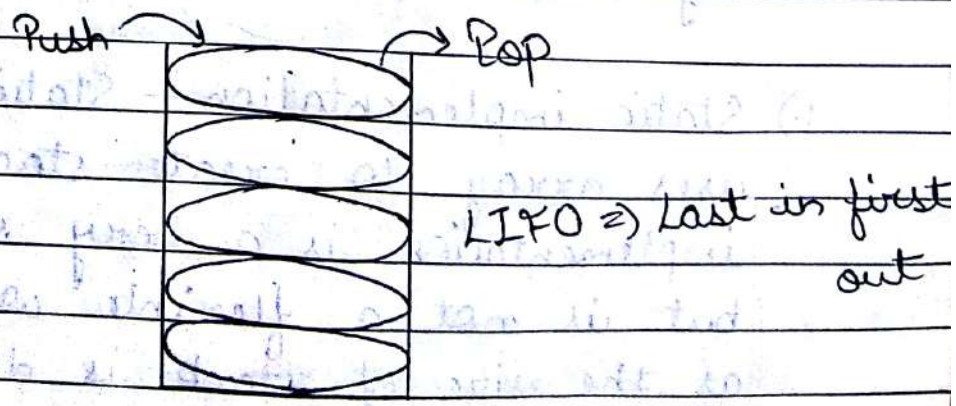
3- When an object is added to a stack, it is placed on the top of all previously entered items. A stack in which items are removed from the top in LIFO.

4- Stacks have several applications in computer programming. Stack can be used to retrieve recently used object from a cache memory.

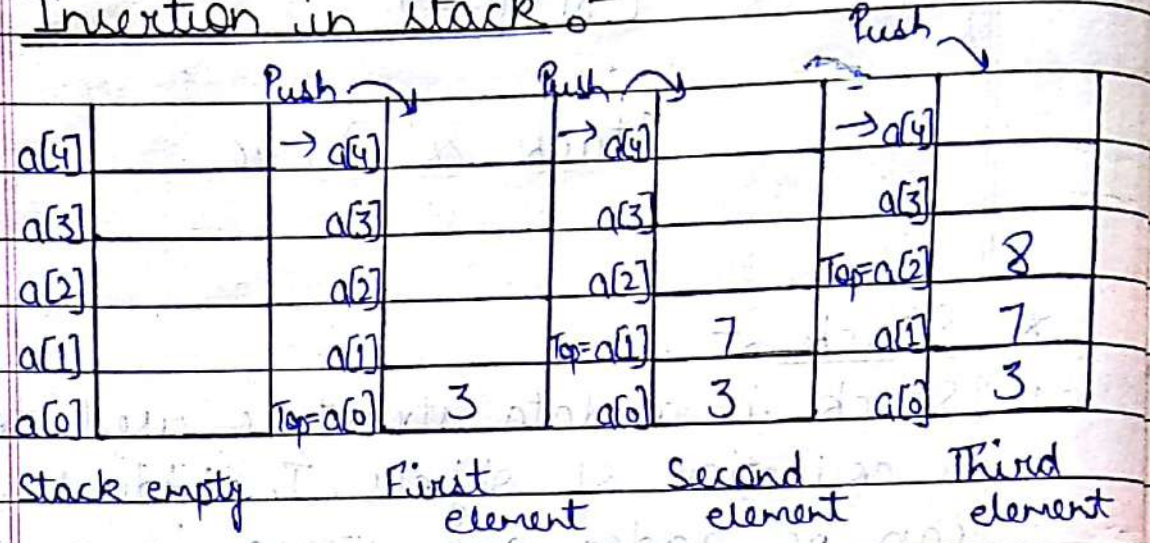
Example -

1- Plates in a marriage

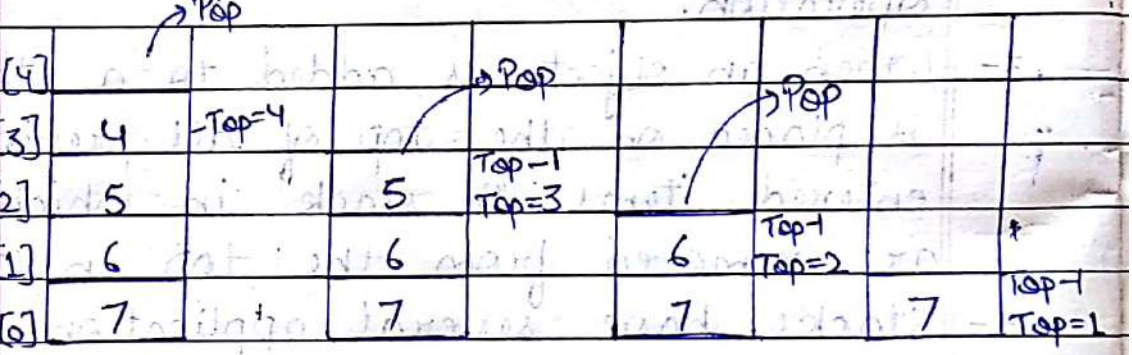
2- Coins stacker



* Insertion in stack :-



* Deletion in stack :-



* Stack implementation :-

1.) Static implementation
2.) Dynamic implementation

1.) Static implementation — Static implementation uses array to create stack. Static implementation is a very simple technique but is not a flexible way of creation, as the size of stack is declared during program design, after that size cannot

be varied.

2.) Dynamic implementation – Dynamic implementation is also called link-list representation and uses the pointers to implement the stack type of data structure.

* Inserting an item :-

Algorithm –

Step-1 Initialize
Set top = -1

Step-2 Repeat step 3 to 5 until
top < maxsize - 1

Step-3 Read item

Step-4 Set top = top + 1

Step-5 Set stack[top] = item

Step-6 Print stack overflow

Function –

```
int stack[5], top = -1;
void push()
{
int item;
if (top < 4)
{
printf("Enter the no.");
scanf("%d", &item);
```

```
top = top + 1;
stack [top] = item;
}
else
printf (" stack overflow");
getch();
}
```

* Pop operation :-

Pop (stack [max size], item);

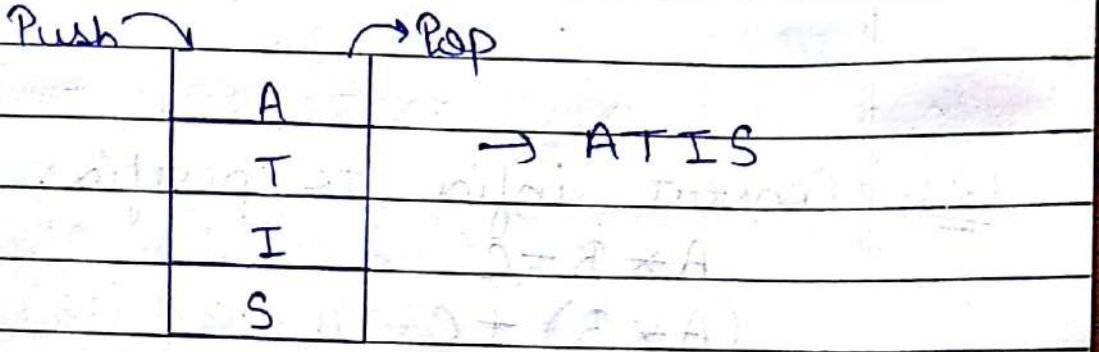
Step 1 - Repeat step 2 to 4 until top > 0
Step 2 - set item = stack [top]
Step 3 - set top = top - 1
Step 4 - print; no deletion
Step 5 - print stack underflow

Function -

```
void pop()
{
int item;
if (top > 0)
{
item = stack [top];
top = top - 1;
printf (" No deletion = %d", item);
}
else
printf (" stack underflow");
}
```

* Application of stack :-

1.) Reversing a string -



- Infix to postfix conversion of expressions-

Expressions -

A+B	AB+	+AB
Infix	Postfix	Prefix

Rules -

1- Parenthesize the expression starting from left to right.
2- During parenthesizing the expression, the operands associated with operator having higher precedence first parenthesized.
3- The sub expression which has been converted into postfix is to be treated as a single operand.
4- Once the expression is converted to postfix form, remove the parenthesis.

Date / /

Operator precedence –

1. Exponential operator ^
2. Multiplication / Division * / /
3. Addition / subtraction + / –

Q1. Convert infix to postfix.

A * B + C

(A * B) + C

(AB *) + C Let AB * = T

T + C

TC +

AB * C +

Q2. A + B / C – D

A + (B / C) – D

A + (BC /) – D Let BC / = T

A + T – D

(A + T) – D

(AT +) – D AT + = U

U – D

U D –

AT + D –

ABC / + D –

A + (B / C) – D

Symbol	Stack	Postfix
A		A
+	+	A
(	+(	A

Date / /

B	+(	AB
/	+(/	AB
C	+(/	ABC
)	+	ABC/
−	−	ABC/+
D	−	ABC/+D−

Q3. (A+B/C * (D+E) − F)

(A + (B/C) * (D+E) − F)

(A + (BC/) * (D+E) − F) BC/ = T

(A + T * (D+E) − F)

(A+T * (DE+) − F) DE+ = P

(A+T * P − F)

(A + (T*P) − F)

(A + (TP*) − F)

(ATP*+) − F

ATP*+F−

ABC/DE+*+F−

((A+(B/C)*(D+E)) − F)

Symbol	Stack	Postfix
(	(	
(	((	
A	((	A
+	((+	A
(	((+(	A
B	((+(	AB
/	((+(/	AB
C	((+(/	ABC
)	((+	ABC/

*	((+*	ABC/
(	((+*(	ABC/
D	((+*(	ABC/D
+	((+*(+	ABC/D
E	((+*(+	ABC/DE
)	((+*	ABC/DE+
)	(	ABC/DE+*+
−	(−	ABC/DE+*+
F	(−	ABC/DE+*+F
)		ABC/DE+*+F−

- Infix to Prefix conversion −

1- Reverse the given string.
2- Convert into postfix.
3- Reverse the output.

Qus1: A + B/C − D (Infix to prefix)
Reversing the string
D − (C/B) + A

Symbol	Stack	Prefix
D		D
−	−	D
(	−(	D
C	−(	DC
/	−(/	DC
B	−(/	DCB
)	−	DCB/

+	−	DCB/+
A	−	DCB/+A−

−A+/BCD (Prefix)

Ques2. A+(B*C−(D/E*F)*G)

Reversing the string -

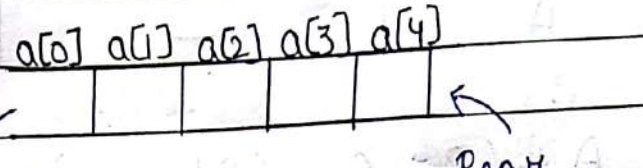
(G*(F*(E/D))−C*B)+A

Symbol	Stack	Prefix
(	(	
G	(	G
*	(*	G
(	(*(	G
F	(*(	GF
*	(*(*	GF
(	(*(*(	GF
E	(*(*(	GFE
/	(*(*(/	GFE
D	(*(*(/	GFED
)	(*(*	GFED/
)	(*	GFED/*
−	(*−	GFED/*
C	(*−	GFED/*C
*	(*−*	GFED/*C
B	(*−*	GFED/*CB
)	(*	GFED/*CB*−
+	(*+	GFED/*CB*−
A	(*+	GFED/*CB*−A*+

+*A−*BC*/DEFG ⇒ Prefix

Date / /

* Queue :- FIFO ⇒ First in first out



"A queue is a non-primitive linear data structure. It is an homogeneous collection of elements in which new elements are added at one end called the rear and the existing elements are deleted from other end called the front end.

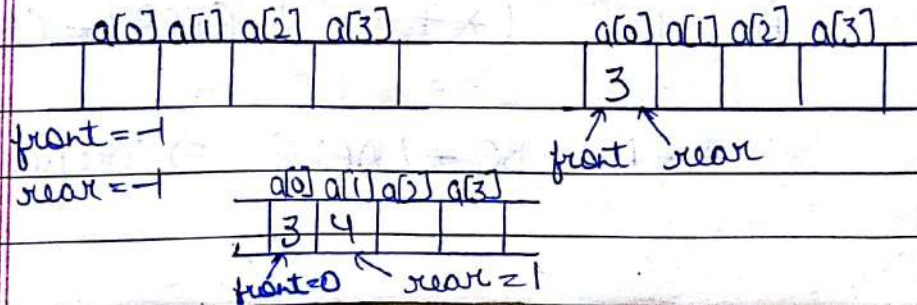
* Implementation of Queue :-

1- Static implementation - With the help of array.

2- Dynamic implementation - With the help of pointers.

* Operations in queue :-

1.) Insertion Operation -



Date / /

Algo-

[Queue, [Maxsize] item]

Step 1- Initialization
set front = -1
set rear = -1

Step 2- Repeat step 3 to 5 until Rear < Maxsize-1

Step 3- Read item

Step 4- if front = = -1 then
front = 0
rear = 0
else
rear = rear + 1
End if

Step 5- set queue[rear] = item

Step 6- Print queue overflow

Queue is declared as int queue[5]; front=-1, rear = -1

```
void queue()
{
int item;
if (rear < 4)
{
printf ("Enter the number");
scanf ("%d", &item);
if (front == -1)
{
front = 0;
rear = 0;
}
```

```
else
{
rear = rear + 1;
}
queue[rear] = item;
}
else
printf("Queue is full");
}
```

2.) Queue deletion –

Queue[Maxsize]

Step 1- Repeat stages 2 to 4 until front >= 0

Step 2- Set item = queue[front]

Step 3- if front == rear
set front = -1.
set rear = -1
else
front = front + 1
end if

Step 4- printf ; no item for delete

Step 5- printf, "queue is empty".

Function for deletion in queue –

```
void delete()
{
int item;
if (front != -1)
{
item = queue[front];
```

```
if (front == rear)
{
front = -1;
rear = -1;
}
else
front = front + 1;
printf ("no deleted is :%d", item );
}
else
printf ("Queue is empty");
}
```

* Circular queue :-

1- A circular queue is one in which the insertion of a new element is done at the very first location of the queue if full.

2- A circular queue overcomes the problem of unutilized space in linear queue implemented as array.

3- A circular queue also has a front and a rear to keep the track of the elements to be deleted and therefore to maintain the unique characteristics of the queue.

The below assumptions are made -

1- Front will always be pointing the field element.

2- If front = rear, the queue will be empty.

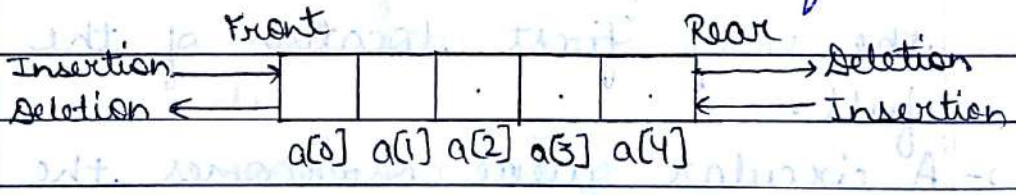
3- Each time a new element is inserted into the queue, the rear is increment-ed by 1.

rear = rear + 1.

4- Each time an element is deleted from the queue, the value of front is incremented by 1.

front = front + 1.

* **Double-ended queue** :- It is also a homogeneous list of elements in which insertion and deletion operations are performed from both the end i.e. we can insert element from the rear end and from the front end. Hence it is called double ended queue.



* **Priority queue** :- A priority queue is a collection of elements such that each element has been assigned a priority and there order in which elements are deleted and processed from the following rules -

An element of higher priority is processed according to the order in which they were added to the queue.

Date / /

★ Applications of priority queue –

1- Round Robin technique for processor scheduling is implemented using queue.
2- All types of costumer services (like – railway ticket reservation) Costumer services centre of software are designed using queues to store costumer information
3- Printer server routines are designed using queue.

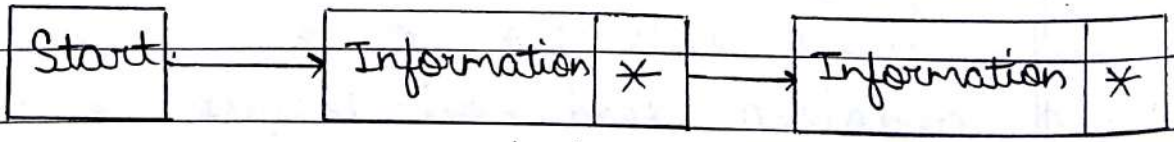
UNIT – 3

Link – List

* **Link – List** – If the memory is allocated before the execution of program, it is fixed and cannot be changed. We have to adopt an alternative strategy to allocate memory when it is required.

There is a special data-structure called link-list that provides a more flexible storage system and it does not require the use of array.

"Link-lists are special lists of some data elements link to one another. The logical ordering is represented by having each element pointing to the next element.



Nodes

In singly link-list nodes have one pointer (next) pointing to the next node

Advantages of Link-list – Link-list have many advantages –

1- Link-list are dynamic data structure, that is they can grow and shrink during the execution of program.

2- Effecient memory utiliration. Here memory is not preallocated. Memory is allocated whenever it is require and it is de-allocated (removed) when it is no longer needed.

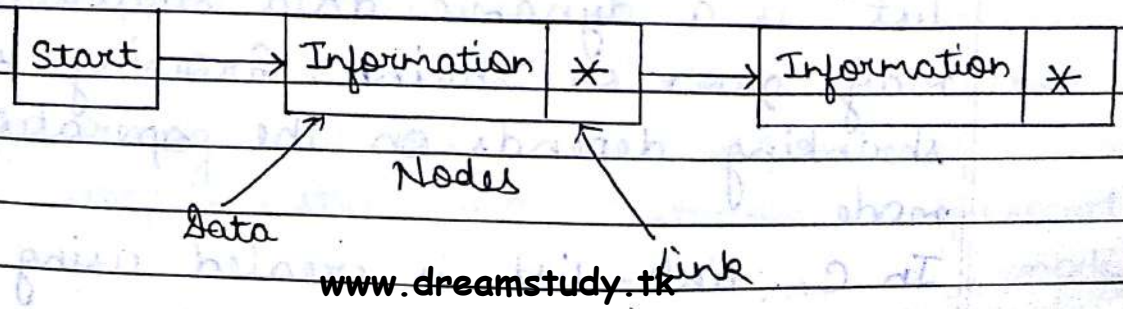
3- Insertion and deletion are easier and efficient. Link-lists provide flexibility in inserting a data item at a specified position and deletion of data-items from the given position.

Disadvantages of Link-list —

1- More memory, if number of feeds are more then more space is required.

2- Access to an orbitatory data-item is little-bit difficult and time consuming.

- **Nodes** :- A link-list is a non-sequential collection of data-items called nodes. Each node in a link-list have basically two fields —



1.) The data fields contains an actual value to be stored and process.

2.) The link field contains the address of the next data item in the link-list. The address used to access a particular node is known as pointer.

3.) The logical and physical ordering of data-items in a link-list need to be same. But in sequential representation these ordering are the same.

Initialization Of link-list -

```
struct node
{
int info;
struct node * link;
};
struct node * tmp;
tmp = (struct node *) malloc (sizeof (struct node));
```

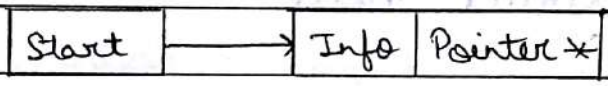
* Types of link-list :-

1- Singly link-list - A singly link-list is a dynamic data structure. It may grow or shrink. Growing or shrinking depends on the operations made.

In C, link-list is created using structures

pointers and dynamic memory allocation function malloc (memory allocation). We consider head as an external pointer. This help in creating and accessing other nodes in the link-list.

When the statement tmp = (struct node *) malloc (sizeof (struct node)); is executed, a block of memory sufficient to store the node is allocated and assigns head as the starting address of the node (now head is an external pointer). This activity can be pictorially represented as-



Node

Now we can assign values to the respective field of node.

tmp = info = 10;

tmp → 10;

tmp link → NULL

Any number of nodes can be created and linked to the existing nodes.

Suppose we want to add another node to the above list, when the following statement is required -

tmp = (struct node *) malloc (sizeof (struct node));

Date / /

start → tmp · info = 100;
start → tmplink · tmp link = NULL.

* Insertion in singly link-list :-

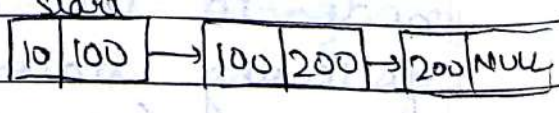
1. Insertion in an empty link-list.
2. Insertion at the beginning.
3. Insertion at the end.
4. Insertion at the n^{th} position.

1. Insertion in empty link-list —

```
struct node
{
int info;
struct node * link;
}
struct node * tmp;
tmp = (struct node*) malloc(sizeof(struct node));
tmp → info = info;
tmp → link = link;
tmp → link = NULL;
start = tmp;
```

2- Insertion at the beginning —

```
struct node
{
int info;
struct node * link;
```

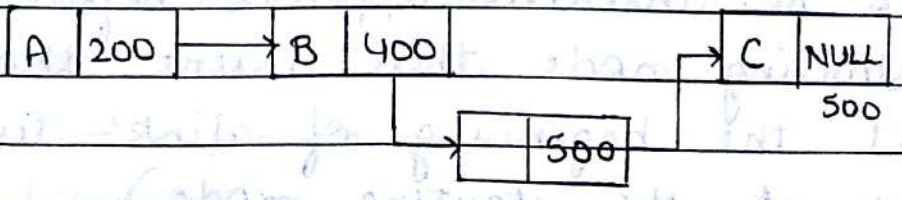


```
}
struct node * tmp;
tmp = (struct node *) malloc (sizeof (struct node));
tmp → info = 10;
tmp → link = start;
start = tmp;
```

3. Insertion at the end —

```
struct node
{
int info;
struct node * link;
}
struct node * tmp;
tmp = (struct node *) malloc (sizeof (struct
                                    node));
struct node * p;
p = start;
while (p != NULL)
p = p → link;
p → link = tmp;
tmp → link = NULL;
```

4. Insertion at n^{th} position —




```
struct node
{
```

```
int info;
struct node * link;
}
struct node * tmp;
tmp = (struct node *) malloc(sizeof(struct node));
struct node *p
p = start;
for (i = 1; i < position - 1 && p != NULL; i++)
p = p → link;
tmp → link = p → link;
p → link = tmp;
```

Inserting a new node into the link-list has the following four instances-

1. Insertion in an empty link-list.
2. Insertion at the beginning.
3. Insertion at the end.
4. Insertion at n^{th} position.

The following steps involved in deciding the position of insertion are as follows-

1.) If a link-list is empty or the node to be inserted appears before the starting node then insert that node at the beginning of link-list. (that is at the starting node).

2.) If the node to be inserted appears

Date / /

after the last node in the link-list then insert that node at the end of the link-list.

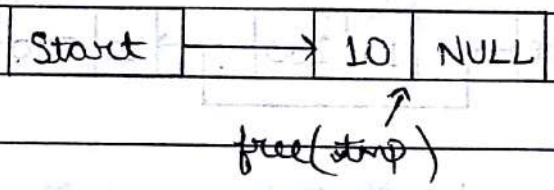
3.) If the above two conditions do not hold true then insert the new node at the specified position within the list.

* <u>Deletion in link-list</u> :- Deleting a node from link-list has the following three instances —

1- Deleting the first node of the link-list.
2- Deleting the last node of the link-list.
3- Deleting the specified node within the link-list.

~~Deleting the last node of the link-list.~~
~~Deleting the specified node within the link-list.~~

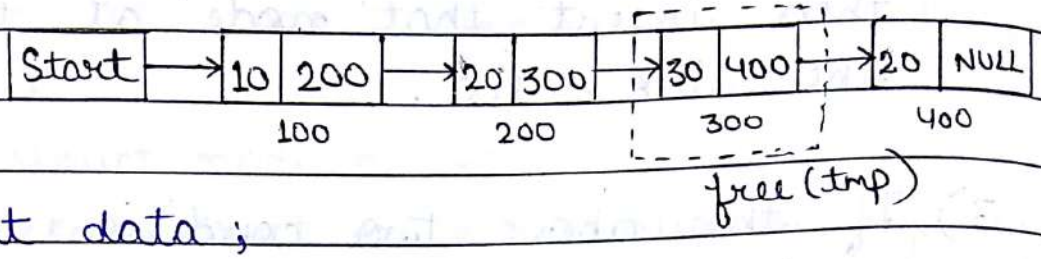
1- Deletion of one node -




```
struct node *tmp
tmp = start;
start = null;
free (tmp);
```

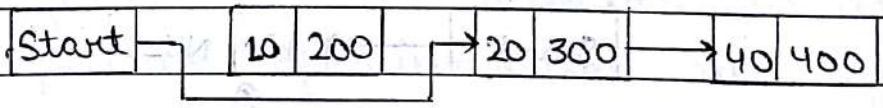
Date / /

2- Deletion of n^{th} node -




```
int data;
struct node *p;
struct node *tmp;
  p = start;
while (p→link != NULL)
  {
if (p→link→info == data)
  {
tmp = p→link;
p→link = tmp→link;
 free (tmp);
  }
p = p→link;
  }
```

3- Deletion at the beginning -




```
struct node *tmp;
 tmp = start;
start = start→link;
free (tmp);
```

Date / /

* Program for matrix multiplication :-

```
#include <stdio.h>
#include <conio.h>
void main()
{
clrscr();
int a[3][3], b[3][3], c[3][3], i, j, k;
printf("Enter the data in matrix A");
for (i=0; i<3; i++)
{
for (j=0; j<3; j++)
{
scanf ("%d", &a[i][j]);
}
}
printf("Enter the data in matrix B");
for( i=0; i<3; i++)
{
for(j=0; j<3; j++)
{
scanf ("%d", &b[i][j]);
}
}
printf("Multiplication of matrices A and B is:
\n");
for (i=0; i<3; i++)
{
for(j=0; j<3; j++)
{
c[i][j] =0;
```

Date / /

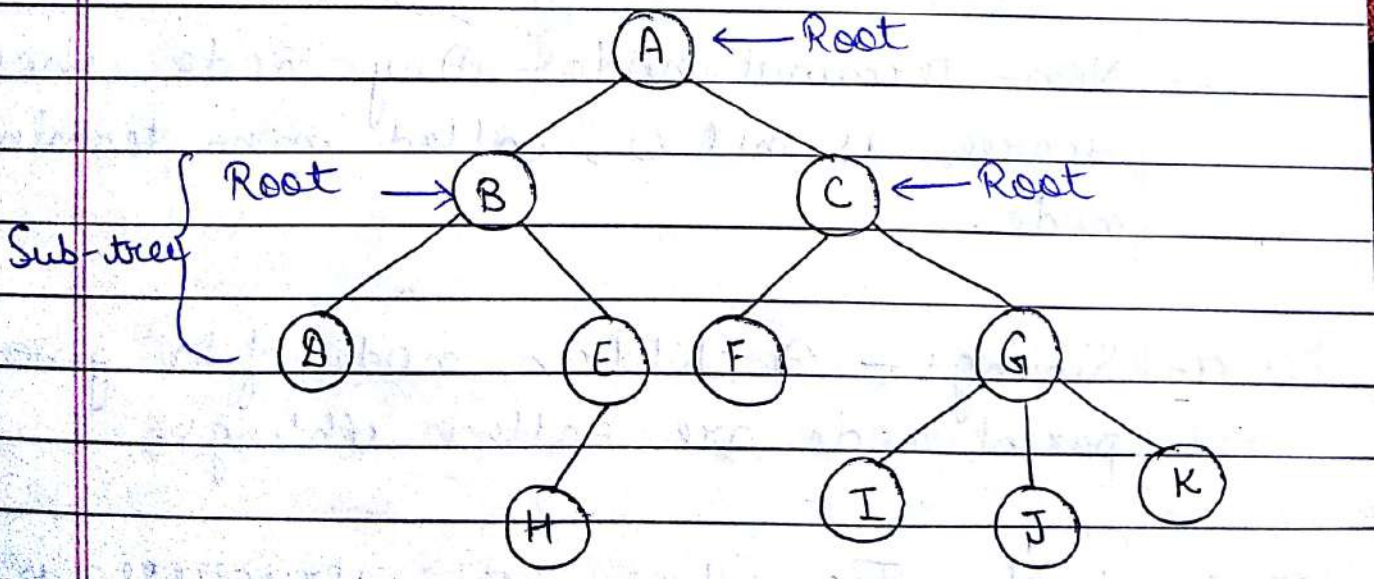
```
for(k=0 ; k<3; k++)
{
c[i][j] = a[i][j] + a[i][k] + b[k][j];
}
}
printf ("%d\n");
for( i=0; i<3; i++)
{
for(j=0; j<3; j++)
{
printf ("%.d\t", c[i][j]);
}}
getch();
}
```

UNIT - 4

Tree

- A tree is a non-linear data-structure in which items are arranged in a sorted sequence.
- It is used to represent hierarichal relationship existing among several data-structures.

Tree:- It is a finite set of one or more data-items, such that -
There is a special data-item called the root of the tree.
And its remaining data-items are partitioned into number of mutually exclusive subsets, each of which is itself a tree, and they are called subtree.



Date / /

★ Tree terminology :-

1- Root – It is specially designed data-item in a tree. It is at the first in the hierarchical arrangement of data-items.

2- Node – Each data-item in the tree is called a node. It is the basic structure in the tree. It specifies the data information and link to other data items.

3- Degree of a node – It is the number of sub-tree of a node in a given tree.

4- Degree of a tree – It is the maximum degree of node in the given tree.

5- Terminal node – A node with degree 0 is called terminal node.

6- Non-terminal node – Any node whose degree is not 0, called non-terminal node.

7- Siblings – A children node of a given parent node are called siblings.

8- Level – The entire tree structure is leveled in such a way that the root node is always at level 0.

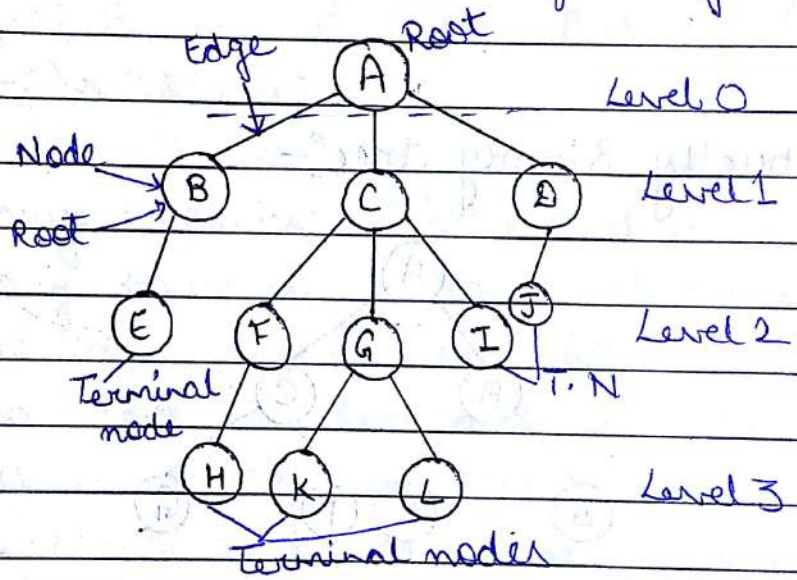
Date / /

9- Edges – It is a connecting line of two nodes.

10- Path – It is a sequence of consequtive edges from the source node to the destination node.

11- Depth – It is the maximum level of any node in a given tree. It is also known as height of a tree.

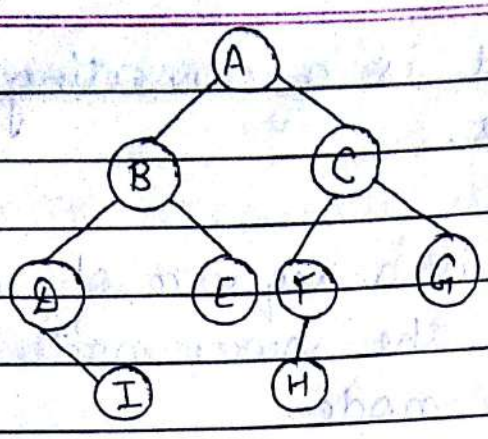
12- Forest – It is a set of disjoint tree.



Degree of tree = 3

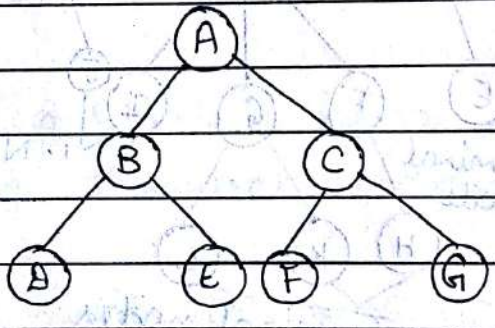
* Binary tree :- A binary tree is a set of finite data items which are either empty or consist of a single item called the root and two disjoint binary tree called the left sub tree and right sub tree.

Date / /



1. A binary tree is a very important and the most commonly used non-linear data-structure.
2. In a binary tree, the maximum degree of any node is atmost 2.
3.

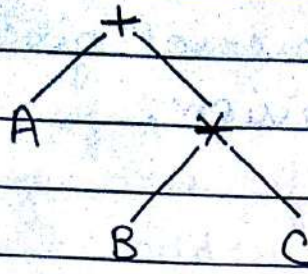
* Strictly Binary tree:-



- If every non-terminal node in a binary tree consist of non-empty left sub-tree and right sub-tree, then such a tree is called strictly binary tree.
- In the above binary tree all the non-terminal nodes having non-empty left or right sub-tree.
- An expression can be represented with the help of binary tree.

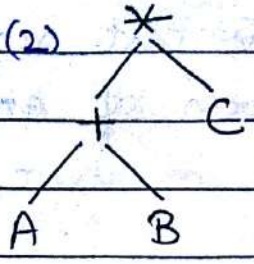
Date / /

(1)



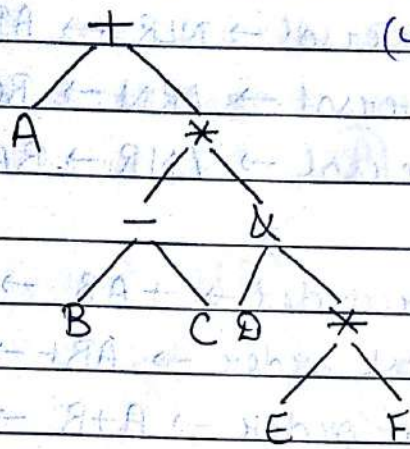
A + (B * C)

(2)



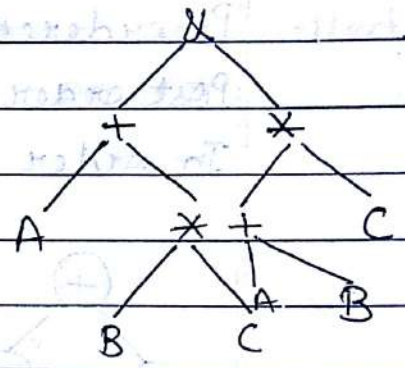
(A+B) * C

(3)



A + (B - C) * D & (E * F)

(4)



(A + (B * C)) & ((A + B) * C)

★ Binary Expression Traversal :-

Binary expression traversal gives the prefix and infix expression.

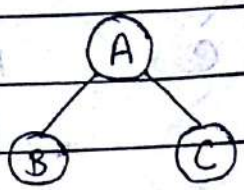
When tree is traverse in pre-order means that the operator precides to operate. Thus the pre-order traversal gives the prefix for the expression.

1. + A * BC
2. * + ABC
3. + A * - BC $ D * EF
4. $ + A * BC * + ABC

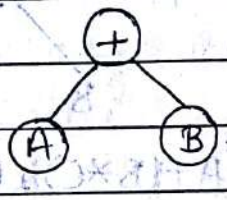
Traversing a binary expression in post order places an operator in postfix form.

Date / /

The post order traversal gives the postfix form of an expression.



Preordered traversal → NLR → ABC
Post order traversal → LRN → BCA
In order traversal → LNR → BAC

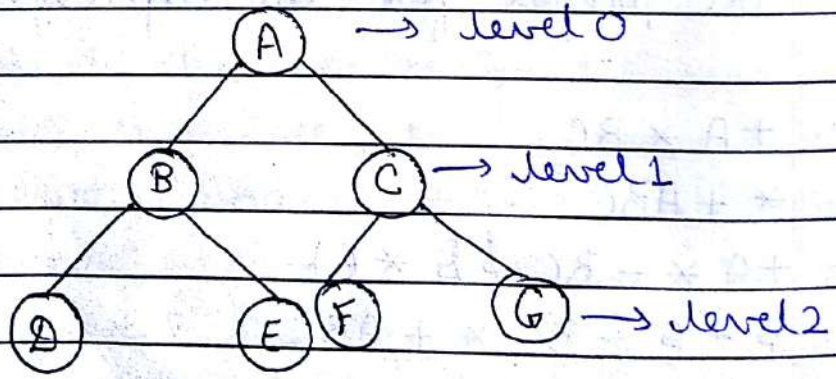


Preorder → +AB → NLR
Post order → AB+ → LRN
In order → A+B → LNR

* Complete binary tree :-

1- A binary tree with n nodes and of depth d is a strictly binary tree, all of whose terminal nodes are at level d.

2- In a complex binary tree, there is exactly 1 node at level 0, 2 nodes at level 1 and 4 nodes at level 2.

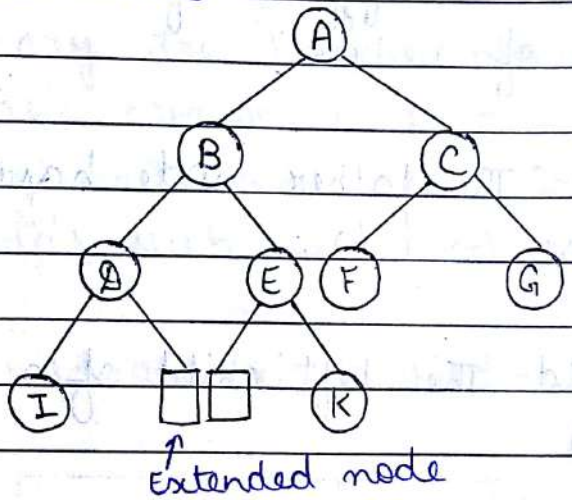


Complete binary tree

Date / /

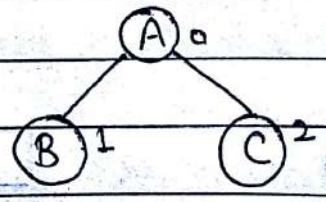
* Extended binary tree -

- A binary tree is said to be extended binary tree, if each node n has either 0 or 2 children.
- In such a case the nodes with two children are called internal nodes and the nodes with 0 children are called external nodes.



* Binary tree representation :-

1- Array representation - An array can be used to store the nodes of a binary tree. The nodes stored in an array are accessible sequentially.

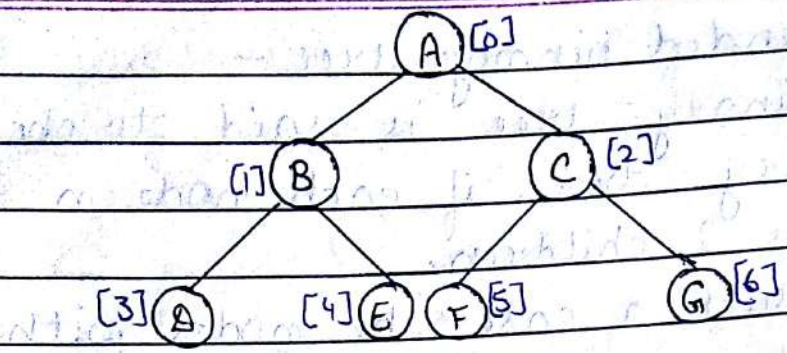


Binary tree

a[0]	A
a[1]	B
a[2]	C

Date / / BT

*



0	A
1	B
2	C
3	D
4	E
5	F
6	G

How to identify the father, children and siblings of a tree?

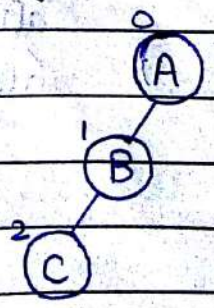
1. Father – The father node have an index is at floor $(n-1)/2$.

2. Left child – The left child of a node n is at $(2n+1)$.

3. Right child – The right child of a node is $(2n+2)$.

4. Sibling – If the left child at index n is given, then its right sibling is an $(n+1)$. Similarly, if right child is given at index n, then its left sibling is at $(n-1)$.

- Disadvantage –



A
B
C

Date / /

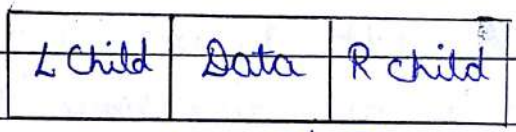
The array representation is more ideal for complete binary tree, but it is not suitable for other than complete binary tree as it results in unnecessary wastage of memory space.

2. Link-list representation of binary tree -

→ To avoid memory loss link-list is used for binary tree representation.

→ The basic component to be representate in a binary tree is a node. The node consist of 3 fields such as -

- Data
- Left child and right child

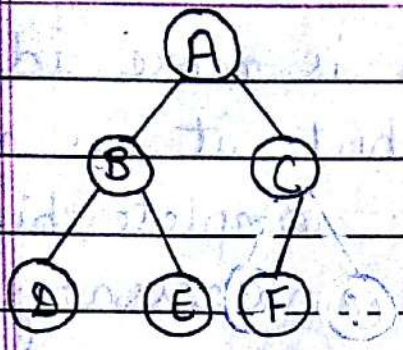


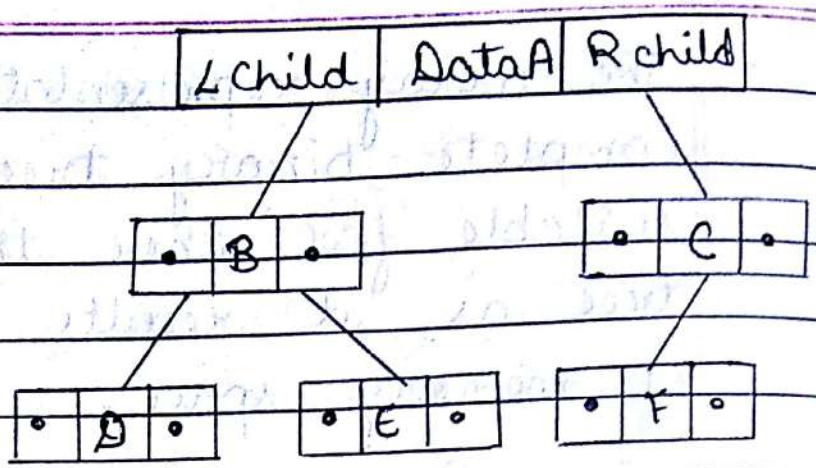
(Node structure Of Binary Tree)

Logical representation -

```
struct node
{
char data;
struct node *l child;
struct node *r child;
};
typef struct node node;
```

Date / /





Date / /

UNIT – 5
B-TREE

* B-Tree :-

- A B-tree is also known as the balanced sort tree. It finds its use in external sorting. It is not a binary tree. To reduce this access, several conditions of the tree must be used –

1. The height of the tree must be kept to be minimum.
2. There must be no empty sub-tree above the leaves of the tree.
3. The leaves of the tree must all be on the same level.
4. All nodes except the leaves must have at least some minimum number of children.

- B-tree of order M has following properties-

1. Each node has a maximum of M children and a minimum of M/2 children or any number from 2 to the maximum.
2. Each node has one fewer keys then children with maximum (M-1) key.
3. Keys are arranged in a defined order within the node.

All keys in the sub-tree do the key of a predecessors of the key and that on right are successors of the key.

Example –

Date / /

1, 2, 3, 7, 8, 9, 10, 15, 18, 20, 25, 26

Order = 4 (M)

Max key = (M-1) = 4-1 = 3

Min key = (M/2) = 4/2 = 2

* Searching :-

- Searching is a process of finding an element within the list of elements stored in any order or randomly.
- Searching is divided into 2 categories - linear search and binary search.
- Linear searching is the basic and simple method of searching.
- Binary searching is more efficient than linear searching.

1-) Linear searching -

- In linear search we access each element of an array one by one sequentially and see whether it is desired element or not.
- A search will be unsuccessful if all the elements are accessed and the desired element are not found.

2-) Binary search -

- Binary search is an extremely efficient algorithm. This search techniques searches the given item in minimum possible comparision.

Date / /

- To do the binary search first we had to sort the array elements.
- The logic behind the binary search is -

1. Find the middle element of the array.
2. Compare the mid element with an item.
3. There are three cases -

(a) If it is a desired element then search is successful.

(b) If it is less than desired item then search only the first half of the array.

(c) If it is greater than the desired element search in the second half of the array.

Example -

8	10	11	13	25	86	1	3
0	1	2	3	4	5	6	7

Middle = $\frac{0+7}{2}$

* Hashing :-

- A very common technique of data processing involves storing information in table and then later retrieving the information stored there.
- The data comprises of a collection of key and values.
- Information is retrieved from the data base by searching for a given key.

Hash table - Hash table is a dictionary in which keys are mapped to array positions by hash functions.

Date / /

Index

1	→ Searching
	Mod = 25/2 = 1

Hashing function — A hashing function transforms an identifier x into a hash table. The address computate is known as hash address of the identifier x. If more than one record have same hashing address, they are said to collide. The Phenomena is called address colligen. The various hashing functions are —

1. The mid square hash function
2. Division hash function.
3. Multiplication method

* Algorithm for Infix to Postfix conversion:-
Q is infix expression, P is postfix expression.

Step 1 - Push ("onto stack and add") to the end of Q.

Step 2 - Scan Q from left to right and repeat step 3 to 6 for each element of Q until the stack is empty.

Step 3 - If an operand is encountered, add it to P.

Step 4 - If the left parenthesis is encountered

then

(a) Add ⊕ to stack.

(b) Repeatedly, pop from stack and add P each operator (on the top of the stack) which has the same precedence are as higher precedence than ⊕.

Step5- If right parenthesis is encountered, then

(a) Repeatedly pop from stack and add to P each operator (on the top of stack until a left parenthesis is encountered.

(b) Remove left parenthesis (do not add left parenthesis to P.

Step7- End.

UNIT – 6

Sorting

* Types of sorting –
1. Insertion sort
2. Selection sort
3. Merge sort
4. Heap sort

Insertion sort :-

1- Insertion sort is based on the idea that one element from the input elements is consume in each iteration to find a correct position to which it belongs in a sorted array.

2- It iterates the input element by growing the sorted array at each iteration. It compares the current element with the largest value in the sorted array.

3- If the current element is greater, then it leaves the element in its place and moves on the next element else to find its correct position in the sorted array, and moves it to that position.

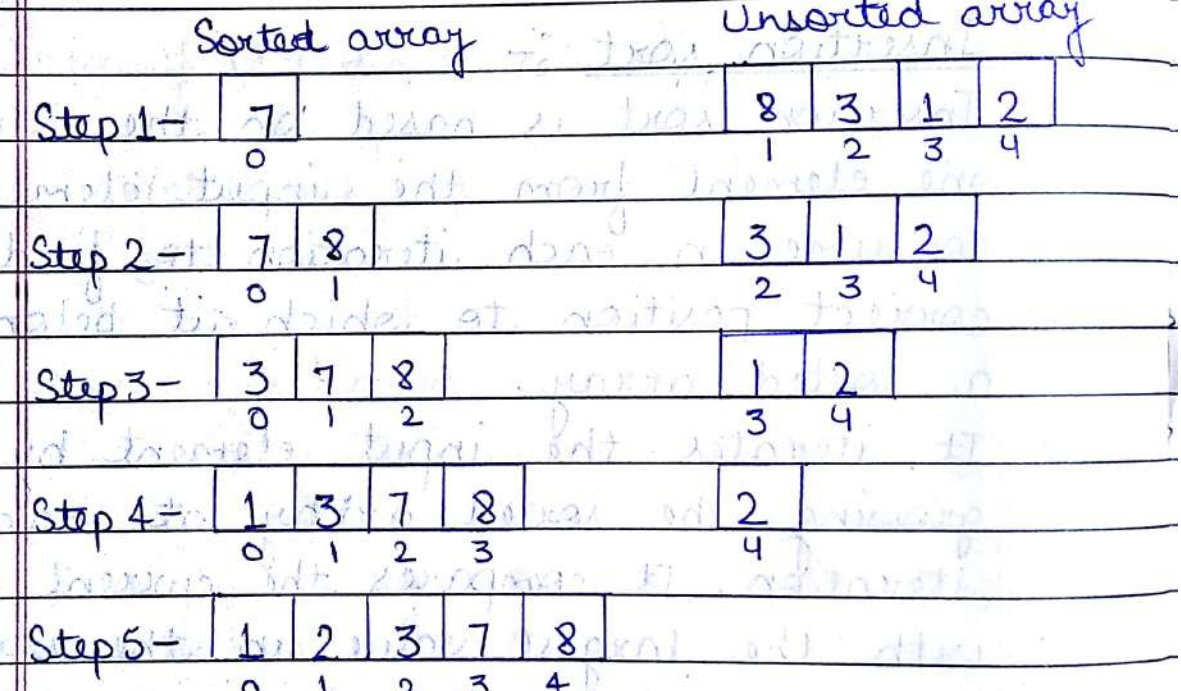
4- This is done by shifting all the elements, which are larger than the current element, in the sorted array to one position ahead.

Date / /

Insertion sort is used when number of elements is small.
It can also be useful when input array is sorted only for elements are misplace.
In complete big array, use binary search to reduce the number of comparisions in normal insertion sort.

Example:

7	8	3	1	2



Methods of insertion sort :-

1- A sub-list (or sorted array) is maintained which is always sorted.
2- Not suitable for large data set.
3- Average and worst case complexity is n^2 (n is no. of lines).
4- $n-1$ steps are required to sort elements.
5- In each pass, we insert current element

Date / /

at appropriate place so that the element in current range are in order.

6- Each pass has K comparision where k is the pass number.

```
#include <stdio.h>
#include <conio.h>
void insert (int[], int)
void main()
{
int a[20], i, n;
clrscr();
printf("Enter the no. of items");
for (i=0; i<n; i++)
{
scanf("%d", &a[i]);
}
insert (a, n);
getch();
}
void insert (int a[], int n)
{
int i, j, temp;
for (i=1; i<n; i++)
{
temp = a[i];
for (j=i-1; j>=0; j--)
{
if (a[j] > temp)
{
```

```
        a[j+1] = a[j];
    }
    else break;
    }
    a[j+1];
    }
    printf ("Data after insertion sort");
    for(i=0; i<n; i++)
    printf ("\n%.d", a[i]);
    }
```

Selection sort :-

1- The selection sort technique is based upon the extension of the minimum and maximum technique.

2- By means of nest of for loops, a pass through the array is made to locate the minimum value.

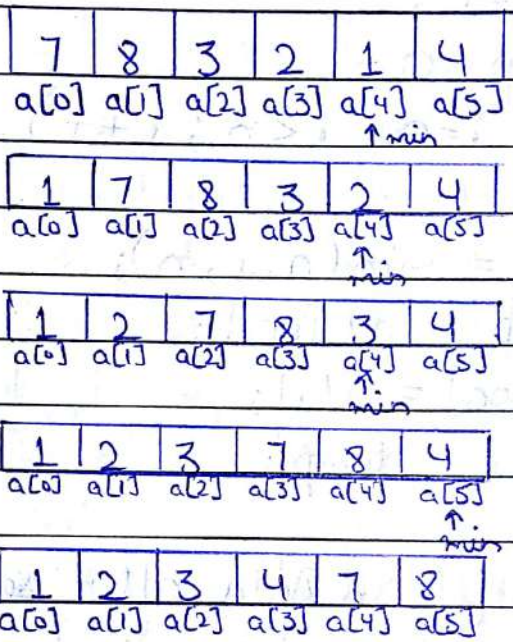
3- Once this element is found, it is placed in the first position of the array (position 0).

4- Another the remaining elements, is made to find the next smallest element, which is placed to the second position of array (position 1) and so on.

Date / /

5- Once the next two last elements is compared with the last element, all the elements have been sorted into the ascending order.

Example:




```
#include <stdio.h>
#include <conio.h>
void select(int[], int);
void main()
{
int a[20], i, n;
clrscr();
printf("Enter the no. of items in array");
scanf("%d", &n);
printf("Enter the data in the array");
for(i=0; i<n; i++)
{
scanf("%d", &a[i]);
select(a,n);
getch();
```

```
}
void select(int a[], int n)
{
int i, loc, temp;
loc = 0;
temp = 0;
for(i=0; i<n; i++)
 {
loc = min(a, i, n);
temp = a[loc];
a[loc] = a[i];
 a[i] = temp;
 }
printf("\n Data after selection sort");
 for(i=0; i<n; i++)
printf("\n%d", a[i]);
}
```

★ Merge sort :-

1- Merge sort is a sorting technique which divides the array into some arrays of size 2 and merge adjecent pairs.

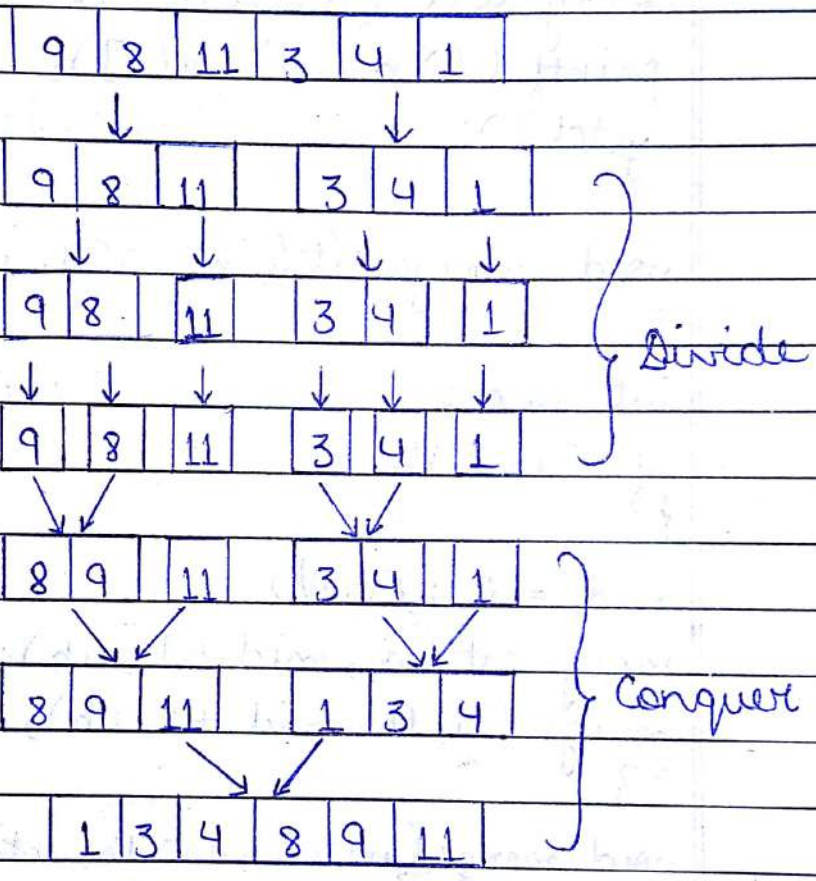
2- We then have approximately $\frac{n}{2}$ array of size 2.

Date / /

3- This process is repeated until there is only one array remaining of size n.

4- Suppose an array A with n elements A_1, A_2 to A_n is in memory.

Example:




```
#include <stdio.h>
#include <conio.h>
void mergesort(int a[], int, int);
void merge(int a[], int, int, int);
void main()
{
int a[20], i, n;
clrscr();
printf("Enter the number of elements");
scanf("%d", &n);
printf("Enter the elements");
```

Date / /

```
for(i=0; i<n; i++)
{
scanf("%d", &a[i]);
}
mergesort(a, 0, n-1);
printf("\n%d", a[i]);
getch();
}
void mergesort(int a[], int lb, int ub)
{
int mid;
if(lb<ub)
{
mid = (lb+ub)/2
mergesort(a, mid+1, ub);
merge(a, lb, mid+1, ub);
}}
void merge(int a[], int lb, int mid, int ub)
{
int k, p1, p2, p3, b[20];
p1 = lb;
p3 = ub;
p2 = mid;
while((p1<mid)&&(p2<=ub))
{
if(a[p1]<=[p2])
b[p3++] = a[p1++];
else
b[p3++] = a[p2++];
}
```

Date / /

```
while (p1 <= mid)
{
b[p3++] = a[p1++];
}
while (p2 <= ub)
{
b[p3++] = a[p2++];
}
for (k = lb; k < p3; k++)
{
a[k] = b[k];
}
}
```

★ Heap Sort :-

1- A heap is define as an almost complete binary tree of n nodes such that the value of each node is less than or equal to the value of the parent. In sequential representation of an almost complete binary tree. This implies that the root of the binary tree (i.e. the first array element) has the largest key in the heap.

2- This type of heap is usually called descending heap or max heap, as the path from the root node to the terminal node from an

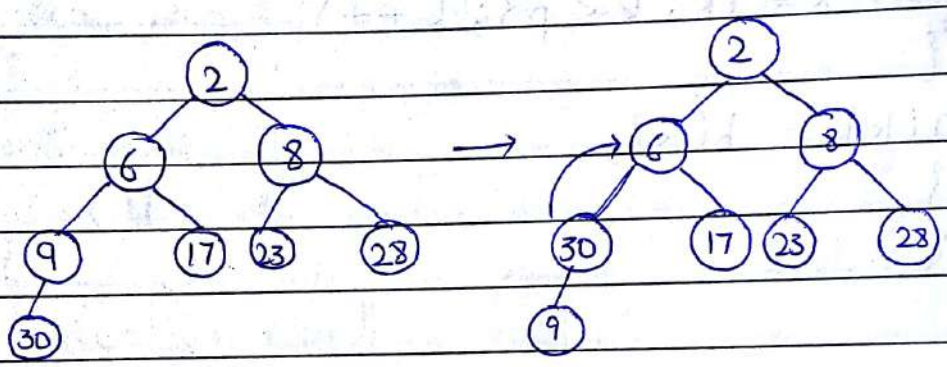
Date / /

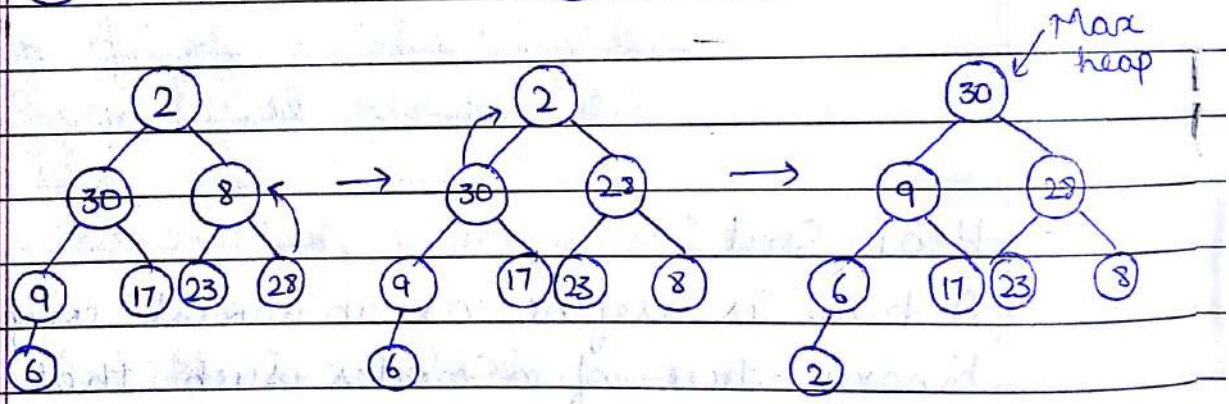
ordered list of elements arranged in descending order.

We can also define an ascending heap as an almost complete binary tree in which value of each node is greater than or equal to the value of its father.

Example-

2	6	8	9	17	23	28	30





Array after sorting - | 30 | 28 | 23 | 17 | 9 | 8 | 6 | 2 |

A heap is very useful in implementing priority queues. The priority queue is a data structure in which the intrinsic ordering of the data items determines the result of its basic operation. Primarily queues can be classified as -

1- Ascending priority queue

Date / /

2- Descending priority queue :-

1- Ascending priority queue – An ascending priority queue can be define as a group of elements in which new elements are inserted orbiterily but only the smallest element is deleted from it.

2- Descending priority queue – A descending priority queue can be define as a group of elements in which new elements are inserted orbiterily but only the largest element is deleted from it.

* Sorting :- Sorting is the basic operation in computer science. Sorting refers to the operation of arranging data in some given sequence i.e. increasing order or decreasing order.

Sorting is categorized as internal sorting and external sorting. By internal sorting, means we are arranging the numbers within the array which is in computer primary memory, whereas the external sorting is the sorting of numbers from the external file by reading in from secondary memory.

There are various sorting methods –

1- Insertion sort

2- Selection sort

Date / /

3- Merge sort

4- Heap sort

Date / /

* Insertion sort algorithm –

Algorithm: INSERTION_SORT (ARR, N)

1- Repeat step 2 to 7 for A = 2 to N:
2- Set TEMP = ARR[A] and B = A-1
3- Repeat step 4 to 6 while TEMP < ARR[B]:
4- Set ARR[B+1] = ARR[B]. [Moves element forward]
5- Set B = B-1
6- If B = 0; then goto step 4.
[End of Step 3 loop].
7- Set ARR[B+1] = TEMP.
[Inserts element in proper place]
[End of Step 1 loop]
8- Exit

* Illustration of Insertion Sort Algorithm –
Let 3, 6, 2, 7, 1, 8, 9, 4, 0 be any list

Pass 1: ARR[2] (i.e. 6) is greater than ARR[1] (i.e. 3) so ARR[1] and ARR[2] (i.e., 3 and 6) are sorted.

Pass 2: ARR[3] will be inserted before ARR[1] (i.e. 3) so that the list from ARR[1] to ARR[3] is sorted.

Date / /

* Selection-sort method —

Pass 1: Step 1: If ARR[1] > ARR[2] then Swap (ARR[1], ARR[2])
Step 2: If ARR[1] > ARR[3] then Swap (ARR[1], ARR[3])
⋮
Step N-1: If ARR[1] > ARR[N] then Swap (ARR[1], ARR[N])

Pass 2: Step 1: If ARR[2] > ARR[3] then Swap (ARR[2], ARR[3])
Step 2: If ARR[2] > ARR[4] then Swap (ARR[2], ARR[4])
⋮
Step N-2: If ARR[2] > ARR[N] then swap (ARR[2], ARR[N])
⋮

Pass N-1: Step 1: If ARR[N-1] > ARR[N] then Swap (ARR[N-1], ARR[N])

* Algorithm for Selection Sort :—

Algorithm: SELECTION_SORT (ARR, N)

1- Repeat step 2 to 3 for A = 1 to N-1.
2- Repeat step 3 for B = A+1 to N:
3- If ARR[A] > ARR[B] then:
Swap (ARR[A], ARR[B]).
[End of if structure]
[End of step 2 loop]
[End of step 1 loop]
4. Exit

END

Hope the study material was helpful , to stay connected with us : Visit Us

www.dreamstudy.tk

https://www.facebook.com/dreamstudy

https://www.instagram.com/dream.study/

https://www.youtube.com/channel/UC8l0Dfy2YekfiigaXEtbAow or
search on youtube with - dreamstudy website

https://twitter.com/DreamStudy_