data को systematically और efficient access करने के लिए manipulation की एक organizational format

Data structure

Unit - 1 Introduction to Data structure & Array in C

Data structure computer memory में Data को organize करने का तरीका है जिसे Data को efficently store, process or retrieve किया जा सकता है।

- Data structure computer memory में Data को arrangement को बताता है।
- Data structure is the structurely representation of logical relationship between elements.
- Data structure is group of data elements, grouped together under one name.

Type of Data structure

(1) prinitive

(2) Non - primite

(1) **primitive Data structure** :- जो fundamental data type को रखते हैं।

जैसे :- Integer, float, character, pointer

★ → ये basic data type होते हैं जो दूसरे किसी data type से मिलकर नहीं बने होते हैं। is also known as basic data type

और उन types के data types single types की value ही store कर सकते है

जैसे - Integer variable Integer type की value store कर सकता है, float variable float types की value store कर सकता है

(2) Non-primitive Data structure :

Homogenous या Heterogenous Data items के group के structure को represent करता है। यह primitive Data structure से derived होते है।

Ex → Array, stack, Tree, graph, list etc

A → जो primitive से मिलकर बनते है उसे Non-primitive कहते है।

Non primitive Data structure को Two types में बाटाँ गया है।

(1) linear Data structure

(2) non linear Data structure

Page No. / Date / /

linear data structure :

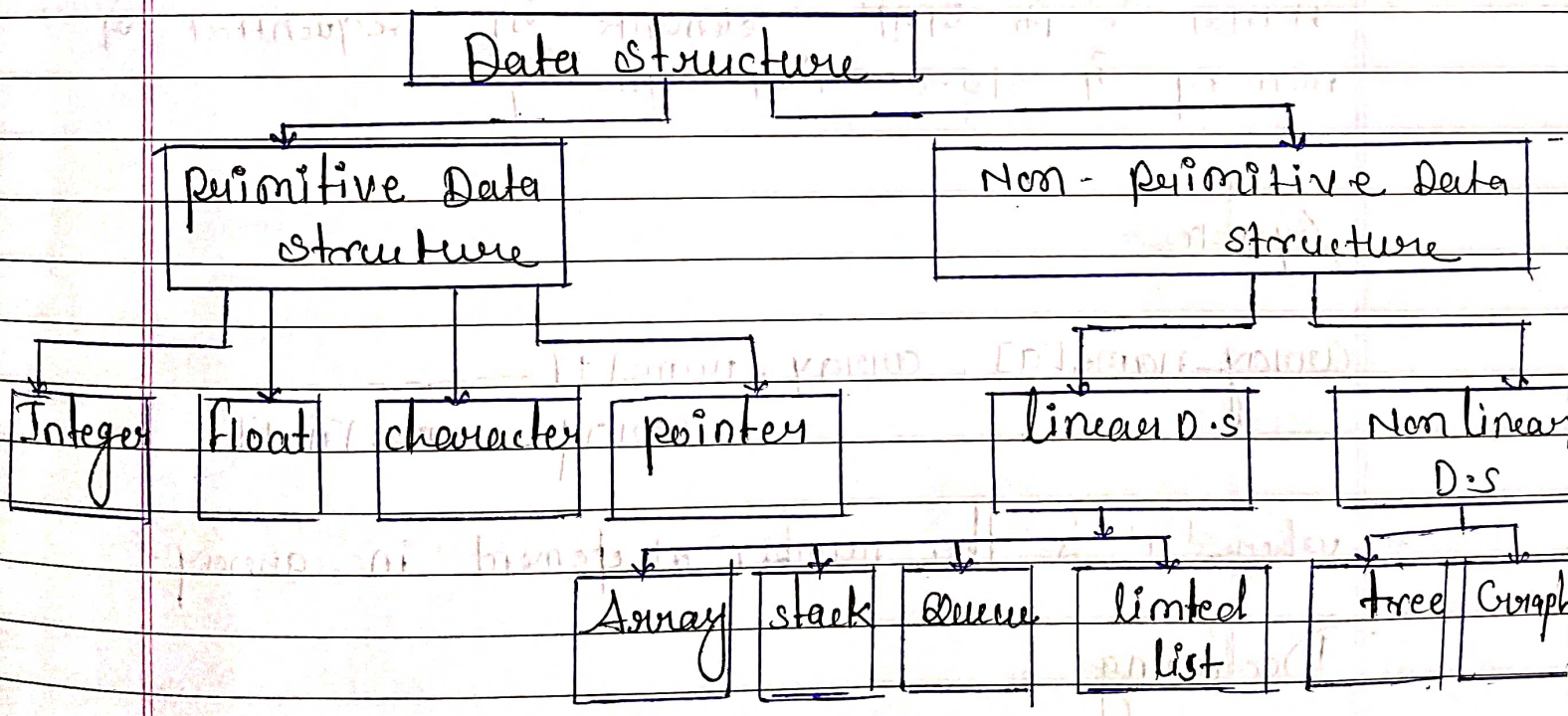
linear D.S में Data elements memory में sequencetial manner में store रहते है।

ex→ Array, linked list, stack, Queue

Non - linear data structure

Random type का Data structure है जिसमें Data non-sequential types में store रहते है।

ex→ tree, Graph



Array :

- Array एक Homogeneous Data type है।
- Array एक प्रकार का Linear Data type structure है।
- Array fixed length वाले होते है जो Data को store करते है।
- Array में Homogeneous का मतलब है कि integer array केवल integer value, character ect को ही store करते है।
- Array को sequential भी कहा जाता है जिसका मतलब है कि सभी elements को sequential से memory में store किया जाता है।

Syntax :

array_name[0], array_name[1] ---------- array_name[n-1]

where 'n' is the number of element in array.

Declaring :

जब हम जानते है कि Program में सभी Variable का use किये जाने से पहले घोषित किये जाते है।

Page No. /
Date / /

उसी तरह, एक array को इसका use करने से पहले घोषित किया जाना चाहिए।

- घोषणा के दौरान, array में size को declare करना आवश्यक है।
- array की घोषणा के दौरान उपयुक्त value का use करना compiler को बताता है कि declaring memory location को allocate या reserve किया जाए।

Syntax :-

General syntax for declaring an array in 'c'

Data type array name [size];

{ This is called one dimentional array }

ex :- int a[10];

Initilizing elements in array :-

data type array name [size] = { elmt, e.l 2 c.l..n }

1) ex :- int a[3] = {1, 2, 3}; data ty :- basic

2) ex :- int a[] = {10, 20, 30, 40}; size = max no of el.

3) ex :- int a[4] : {10, 20, 30};

Page No.
Date / /

Basic Operation :-

(1) Traversing :- print all array element one by one.

(2) Insertion :- Adds an elements at given index.

(3) Search :- Search an elements using the given index or by in value.

(4) Deletion :- Delete an elements at the given index.

(5) Update :- Update an elements at the given index.

Insertion is one-Dimentional array :-

Insertion :-

Array में Data elements को Insert करना ही Insertion कहलाता है।

Array में insertion दो प्रकार से किया गया है :-

1) Insertion at the end of array.
2) Insertion at required position.

Page No. / Date / /

Array के end में element को easily से insert कराया जा सकता है, यदि array के पास उस element को accommed करने के लिए enough large space है।

लेकिन array में element को किसी required position में करने के लिए Array के element को new location में downword move करना होता है।

ताकि new element को accommed किया जा सके।

Ex:-

0	100		0	100	
1	120	new insertion	1	120	
2	40		2		← new element
3	60		3	40	
4	80		4	60	
5			5	80	
6			6		
7			7		
8			8		
9			9		
10			10		

Algorithm for inserting element in array :

INSERT (Arr , N , K , ITEM)

Arr : Linear Array

N : Total number of element in array.

K : Any position integer which is equal less than N.

यह algorithm ITEM को Arr के K+K position में insert करता है।

1) set : J = N // [Initialize Counter]

2) Repeat step 3 & 4 while $j \geq K$

3) set Arr [J+1] = Arr [J] [move the element downward]

4) set J = J - 1 [Decrease Counter]

5) set Arr [K] = ITEM [insert element]

6) Set N = N + 1

7) Exit

Page No. / Date / /

Develop a Program in c to display the list of value in reverse order :

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

for(i=0; i<n; i++)
scanf("%d", &a[i]);

for(i=0; i<n; i++)
printf("%d", a[i]);
printf("Element of array in reverse order");

for(i=n-1; i>=0; i--)
{printf("%d", a[i]);}
}
getch();
}
```

Output

Enter the size of array 3
Enter the element of array
1
2
3
123 element of array in Reverse order 321

String Reverse Program

```
#include <stdio.h>
#include <conio.h>
#include <string.h>
void main()
{
char str1[20] = "tutorial";
clrscr();
printf("given string = %s\n", str1);
printf("reversed string = %s", strrev(str1));
getch();
}
```

Output : given string : tutorial
reverse string - loirotut

String Reverse Program

```
void main()
{
clrscr()
char str[20]
int i, len;
printf("enter string");
scanf("%s", &str);
len= strlen(str);
for(i=len-1; i>=0; i--)
{
Printf("%c", str[i]);
}getch();
}
```

Size of Array Sum

Page No. / Date / /

```
#include <stdio.h>
#include <conio.h>
void main()
{
clrscr();
int a[20];
int i, n;
int sum = 0;
printf (" Enter the size of array ");
scanf (" %d ", &n);

printf (" Enter the element of array ");

for(i=0 ; i<N ; i++)

Scanf (" &%d", &a[i]);

for(i=0 ; i<N ; i++)

{ sum = sum + a[i];

printf ( " Sum of array elemant s %d", sum);

}
  getch();
}
```

Page No.
Date / /

Searching :

Searching, list में से किसी particular element को find करने का process होता है।

यदि element list में present है तब process successful होता है, और process, उस element का location return करता है otherwise search unsuccessful कहलाता है / होता है।

दो type के search use होते हैं :

1. Linear search
2. Binary search

1. Linear search :

Linear search को sequential search algorithm भी कहते हैं।

Linear search में हम list को completely traverse करते हैं और list के element को (ITEM) (जिसे search करना है) उसके साथ match करते हैं। यदि match find होता है तो ITEM का location, Return हो जाती है otherwise null Return करता है।

Page No. /
Date / /

Algorithm

Linear_Search (Arr, N, ITEM, LOC)

Arr : linear array ही।
N : Total number of element in array.
ITEM : given item of information in array.

Algorithm finds the (location) loc of Item in array (arr) or loc = 0 if search is unsuccessful

1. Set Arr[N] = ITEM
2. LOC = 0
3. Repeat while Arr[LOC] ≠ ITEM

   Set loc = loc + 1

4. If loc = N ; the Search is unsuccessful

   Exit

5) print LOC

② Binary Search :

Binary search की Pre-condition Array shorted होना चाहिए।

0	1	2	3	4
5	10	15	25	30

यह एक search techniq है, जो shorted list में efficiantly work करती है।

Binary Search techniq का use list में किसी element के search करने के लिए हमें ensure कर लेना चाहिए कि list shorted है या नहीं।

Binary search devide and conquer approach को follow करता है। जिसमें list को दो parts में devide होती है। और item को list के middle element से compare किया जाता है। तो middle element की location return की जाती है। otherwise matching के द्वारा product Result के अनुसार दोनो parts में से किसी एक part को search करते है।

Algorithm :

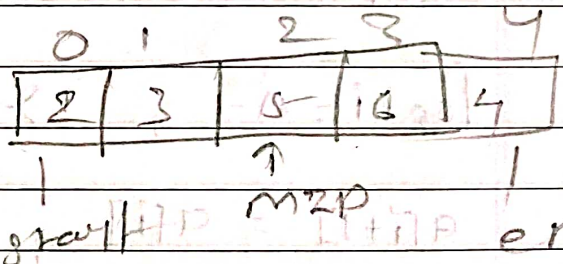
Binary_search(A, LB, UB, ITEM, LOC)

A → shorted array
LB → lower Band
UB → Upper Bound
ITEM → Given item or information
LOC → Location

Page No. / Date / /

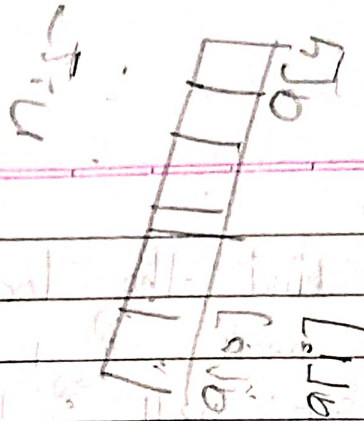
Algorithm find the location LOC of ITEM जो search करती है। otherwise LOC को null में set करता है।

1. Set BEG = LB, END = UB and MID = int ((BEG + END) / 2)
2. Repeat steps 3 & 4 while BEG ≤ END and A[MID] ≠ ITEM
3. If ITEM > A[MID] then set END = MID - 1
   else BEG = MID + 1
4. MID = INT ((BEG + END) / 2)
5. If A[MID] = ITEM then set
   LOC = MID
   else
   LOC = Null
6. Exit



(1) Set Start = LB
End = UB
MID = int[start + end / 2]

Mid = 4/2 = 2



Page No.
Date / /

Insert Program

```
#include <stdio.h>
#include <conio.h>
void main()
{
int a[20];
int item, i, n, possition;
clrscr();
printf("Enter size of array");
scanf("%d", &n);
printf("Enter array of element");
for(i=0; i<n; i++)
scanf("%d", &a[i]);
Printf("Enter the location where you wish to
        insert an element");
Scanf("%d", &possition);
printf("Enter the item to insert");
Scanf("%d", &item);

for(i=n-1; i>=position; i--)

a[i+1] = a[i]
a[possition] = item

printf("resultant array is");
for(i=0; i<=n; i++)

printf("\n %d", a[i]);
  getch();
}
```

Page No. /
Date / /

Output :

Enter the item to insert

Enter the size of array = 6

Enter array of element = 5, 10, 15, 25, 30, 35

Enter the location where you wish to insert an element = 3

Enter the item to insert 20

resultant array is

5 10 15 20 25 30 35

Search Program

```
#include<stdio.h>
#include<conio.h>
void main()
{
int a[20];
int i, n, item;
clrscr();
printf("enter size of array");
scanf("%d", &n);
```

```
printf("enter the array element");
for (i=0 ; i<n ; i++)
scanf ("%d", &a[i]);
printf("enter the item to search");
scanf ("%d", &item);

for(i=0 ; i<n ; i++)
  if( a[i] == item)
  {
    printf("%d found at position %d",
                    item, i+1);
  }
getch();
}
```

Output

Enter size of array 4

Enter the array element 4, 5, 54, 6

Enter the item to search 6

6 found at position 4

Page No. /
Date / /

2D arrays :

An array consisting of two subscripts is known as two dimensional array. These are often known as array of the array. In two dimensional arrays the array is divided into rows and column. These are well suited to handle a table of Data in 2-D array we can declare an array as:

Declaration :

Syntax : data_type array_name [row_size] [column_size]

Ex : int arr[3][4]

यहाँ 3 Row होंगे और चार Column

$$\begin{bmatrix} 1 & 4 & 7 & 10 \\ 2 & 5 & 8 & 11 \\ 3 & 6 & 9 & 12 \end{bmatrix}$$

Where first index value shows the number of the rows and second index value shows the number of the columns in the array

Initializing two-dimensional arrays :-

int a[2][3] = {0,0,0,1,1,1}

Multi Dimensional Array

→ two dimesinal array में N row तथा m column होते है।

A[M][N]

→ इसे M x N भी कहते है जब array में index integer के बजाय pair (I, K) रूप में होते है. index को subscript भी कहते है।

initialization Two-Dimensional Array :

int A[30][30] = {2,3,4,5,6,7,8,9,10}

n columns → j change

ai, j	a_{11}	a_{12}	a_{13}
m	a_{21}	a_{22}	a_{23}
rows	a_{31}	a_{32}	a_{33}

i ↓ range

fig : M x N matrix

Acces in Array : a[1][1] => 2
a[2][1] => 5

Two dimensional Array Syntex :

Data_type

Array_name [row_size], [column_size];

Page No.
Date

Datatype :

Array में आप जिस type के element store करना चाहते हैं आपका array भी उसी data type (int, float, char) का होना चाहिए।

Array name :

C identifier rules को follow करके array को कोई भी नाम दे सकते हैं।

Array size :

2D array में array size को हम row और column की संख्या में लिखते हैं।

2D array के पहले square brackets [ ] में हम ये बताते हैं कि array में कितनी row चाहिए तथा उसी प्रकार दूसरी square bracket से Columns.

String :

एक sequence of character होता है जो single data item की तरह treat होता है और null character '\0' के द्वारा terminate होता है।

A	i	O	u	\0

6552 18 6552 19 6552 20 6552 21 6552 22

MATRIX ADDITION

```
#include<stdio.h>
#include<conio.h>
void main()
{
int n, m, i, j;
int a[20][20], b[20][20], c[20][20];
clrscr();
printf("enter value of row & column");
scanf("%d %d", &n, &m);

printf("enter value of 1st matrix\n");

for(i=0; i<n; i++)
{
 for(j=0; j<m; j++)
 scanf("%d", &a[i][j]);
}

printf("enter value of 2nd matrix\n");

for(i=0; i<n; i++)
{
  for(j=0; j<m; j++)

scanf("%d", &b[i][j]);

}
```

```
printf("resultant matrix is \n");
for(i=0; i<n; i++)
for (j=0; j<m; j++)
  c[i][j] = a[i][j] + b[i][j];
for (i=0; i<n; i++)
{
for(j=0; j<m; j++)
printf("%d", c[i][j]);
printf("\n");
}
getch();
}
```

* Implementation of Stack.
* Application of Stack

Page No. _____
Date

UNIT - 02
STACK AND QUEUE

STACK :- एक non-primitive Linear Data Structure है, यह एक Ordered list है, जिसमें element का insertion or deletion केवल ~~left~~ एक सिरे से होता है जिसे Stack का Top कहते हैं, सबसे last में add किया गया element सबसे पहले से remove होता है। इसलिए Stack को Linear (Last in first out) type का list कहा जाता है।

Stack में सबसे ज्यादा access होने वाला element top में रहता है, जबकि सबसे कम access होने वाला element bottom में रहता है।

Representation of Stack

Basic Operation :-

Push ⇒ Stack के Top में New element को add करने का Process push Operation कहलाता है।

हर एक Push Operation के बाद top एक increment किया जाता है।

- In case जब stack fill हो जाता है और new element को insert कराया जा रहा है तो उस condition को "overflow" कहा जाता है।

② Pop :- stack के top से element को delete करने का process pop कहलाता है। हर एक pop operation के बाद top एक decrement होते हैं यदि stack में कोई element नहीं है और pop operation perform किया जा रहा है। तब वह "stack underflow" की condition आयेगी।

"Stack underflow" वह situation होती है जब stack में कोई element नहीं होती और इसी के लिए कोई element delete भी नहीं हो सकता है।

इस case में stack का Top bottom में होता है।

"Stack overflow" वह situation होती है जब stack full हो चुका होता है और कोई नया element stack में push नहीं किया जा सकता है।

Top :- यह stack के top को बताता है। stack में top under flow और overflow की condition को check करने के लिए करते हैं।

※ Algorithm for push operation :-

Push (STACK , MAX , TOP, ITEM)

STACK – जिसमें हमें ITEM Push करना है।

TOP – Stack के Top ITEM को point या denote कर रहा है।

MAX – stack of size

ITEM – एक variable है जिसमें stack के top element को store करना है।

1) If Top = MAX, then write overflow & exit

2) Set Top = Top + 1 [in this we check over flow]

3) set STACK [TOP] = ITEM

4) EXIT.

Algorithm for pop operation.

POP (STACK, TOP, ITEM)

1. If Top = 0 ; then write "Underflow" & exit
2. ITEM = STACK [TOP]
3. Set Top = Top - 1
4. EXIT

इस point में stack के Top stack के सबसे highest Location में रहता है।

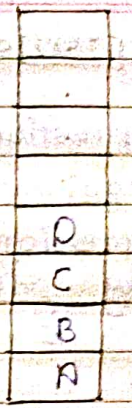
Ex. ∴ N = 8 (STACK size)

STACK — A, B, C, D, F, K ----

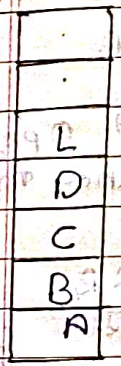
(a) POP (stack, ITEM)
(b) POP
(c) PUSH (stack, L)
(d) PUSH (STACK, P)
(e) POP
(f) PUSH (stack R)
(g) PUSH (stack S)
(h) POP

(a)

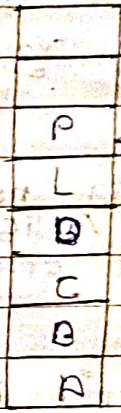
(b)



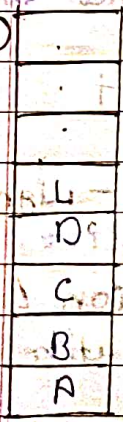
(c)



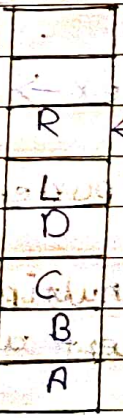
(d)



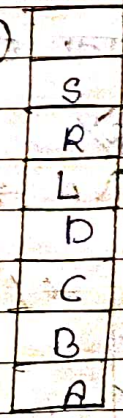
(e)



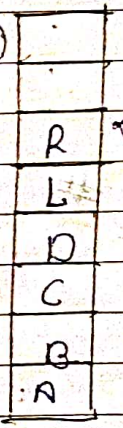
(f)



(g)



(h)



Evaluation

Expression :-

- Expression operands और Operators का legal combination है।
- Expression को लिखने के लिए basically तीन Type notation होते हैं।

① infix notation – इसमें Operator operands के बीच होता है।
ex – A+B

② Prefix notation /Polish – इसमें Operator Operands के पहले होता है।
ex → +AB

③ Postfix notation /reverse polish – इसमें Operator Operand के बाद होता है।
ex → AB+

#
higher level ↑ precedence level

^ → exponential – Right associative

*/ → multiplication / division – left associative

+- → Addition / subtraction – left associative

---

① Infix → Postfix

A-B / (C*D/E)
= A-B / (CD* E/)
A-(BCD*E//)
ABCD*E//-

② Infix → Prefix

(A+B)* (C-D)
= (+AB)* (-CD)
*+AB-CD

① Prefix convert. (Infix →)

A-B / (C*D^E)
= A-B / (*C^DE)
= A-(/B*C^DE)
= -A/B*C^DE

② ↙ Postfix

A+[(B+C)+(D+E)*F]/G

= A+[(BC+)+(DE+)*F]/G
= A+[(BC+)+(DE+F*)]/G
= A+(BC+DE+F*)+/G
= A+(BC+DE+F*+)G/
ABC+DE+F*+G/+

Page No.
Date

⑧ $[(A+B)*C-(D-E)]\uparrow(F+G)$ ~ Post fix

$= [(AB+)*C-(DE-)]\uparrow(FG+)$

$= [(AB+C*)-(DE-)]\uparrow(FG+)$

$(AB+C*DE--)\uparrow(FG+)$

$\boxed{AB+C*DE--FG+\uparrow}$

⑨ $(A+B)*C/D+(E\uparrow F/G$

$= (AB+)*C/D+(EF\uparrow)/G$

$= (AB+C*)/D+(EF\uparrow)/G$

$= (AB+C*D/)+(EF\uparrow)/G$

$= (AB+C*D/)+(EF\uparrow G/)$

$\boxed{AB+C*D/EF\uparrow G/+}$

Ans:

Conversion from Infix to post fix using stack :-

Algorithm :-

Suppose ; Q एक Arithmatic expression है। जो Infix notation में लिखा गया है।

- यह Algorithm Q के लिए Post fix expression P Find करेगा।

1) Push, left parenthesis "(" on stack and "right" parenthesis ")" at the end of Q.

2) Scan Q from left to right and repeat the step 3 to 6 untill stack is empty.

3) If operand is encotered ; then add it to postfix expression P. (यदि Operator encotered आता है तो Postfix expression में "add" किया जाता है।)

4) If left parenthesis "(" is encontered then Push it on stack. (यदि left parenthesis आता है, तो उसे stack में Push कर देंगे।)

5) If Operator is encountered then; (यदि Operator मिलता है तो)

a) First pop the Operators which are already on the stack that have higher or equal precedence than the current operator and append (add) them to P.
पहले stack पर पहले से मौजूद Operator को Pop करें, जो मौजूदा Operator में higher या equal precedence होता है। फिर उसे Postfix (P) में add करें। और उसे append करें।

यदि stack के TOP पर left parenthesis है तो operator push करें।

(b) If left parenthesis "(" is there on top of stack then push the operator on stack.

(6) If right parenthesis ")" is encountered then (repeatedly) POP all the elements of stack until left parenthesis "(" (respectively) is encountered and append each poped elements to postfix expression P.

(7) यदि Input over हो जाता है, तो stack के बचे हुए element को POP करके postfix expression (P) में append कर देते हैं।

(6) यदि right parenthesis मिलता है, तो stack के सभी elements को तब तक POP करें जब तक left parenthesis नहीं मिलता है एक operator मिलता है और POP किए गये elements को Postfix expression P में add किया जाता है।

Ex:- Insertion & Deletion

1) A + B * C

Symbol	Stack	Postfix (P)
A	((push)	A (add)
+	(+	A
B	(+	AB
*	(+*	AB
C	(+*	ABC
)	—	ABC*+

P = ABC*+

Q 2) (A+B) * C

Symbol	Stack	Postfix (P)
(	((	—
A	((	A
+	((+	A
B	((+	AB+
)	(	AB+
*	(*	AB+
C	(*	AB+C
)	—	AB+C*

P = AB+C*

Q A+(B*C-(D/E↑F)*H

Symbol	stack	P
A	(	A
+	(+	A
(	(+(	A
B	(+(	AB
*	(+(*	AB
C	(+(*	ABC
-	(+(-	ABC*
D	(+(+(-	
/	(+(+(	
E	(+(-(/	
↑	(+(-(/	
F	(+(-(/↑	
)	(+(-(/↑	
*	(+(-	
H		
(	(+(-(	ABC*
D	(+(-(	ABC*D
/	(+(-(/	ABC*D
E	(+(-(/	ABC*DE
↑	(+(-(/↑	ABC*DE
F	(+(-(/↑	ABC*DEF
)	(+(-	ABC*DEF↑/
*	(+(-*	ABC*DEF↑/
G	(+(-*	ABC*DEF↑/G
)	(+	ABC*DEF↑/G*-
*	(+*	ABC*DEF↑/G*-
H	(+*	ABC*DEF↑/G*-H
)	empty	*+

E↑F/G)*H)

Q A+B/C-D

Symbol	Stack	P
A	(	A
+	(+	A
B	(+	AB
/	(+/	AB
C	(+/	ABC
-	(-	ABC/+
D	(-	ABC/+D
)	empty	ABC/+D-

Evaluation of postfix expression

* Suppose P एक arithmetic को use करके

- Suppose P एक arithmetic expression है जो Postfix notation में लिखा गया है।

- यह algorithm, stack का use करके Postfix expression P का evaluate करता है।
- यह algorithm expression P का value को find करता है।

1. Add right parathesis ")" at the end of P.
2. Scan P from left to right and repeat steps 3 & 4 for each element of P until ")" is come;
3. If a operand (x) is encountered then ;
4. If operator is encountered, then push it on stack;
5. Remove POP the top two element from stack, where a is top element & b its next top element.
6. Evaluate b (x) a
7. Place result of (6) again on stack
8. Set value to top element of stack

Postfix P :- 5 6 2 + * 12 4 /

Symbol (P)	Stack
5	5
6	5, 6 (b)
2	5, 6, 2 (a)
+	5, 8
*	40
12	40, 12
4	40, 12, 4 (b)
/	40, 3 (a)
-	37
)	

Postfix :- 2, 3 + 4 * 5 /

Symbol	Stack
2	2
3	2, 3
+	5
4	5, 4
*	20
5	20, 5
/	4

Conversion from infix notation to prefix notation.

Step 1 :- Reverse the input string. Reverse करते के time पर "(" become ")" and ")" become "(".

Step 2 :- Input string को left से right scan करेंगे और output string का एक element के लिए step 3 to 6 को Repeat करेंगे।

Step 3 :- यदि Operand आता है, तो उसे output string में add करेंगे।

Step 4 :- यदि "(" Parenthesis आता है तो stack में push करते है।

Step 5 :- यदि ")" Parenthesis आता है तो stack top से Operator को pop करते है और उसे output string में add करते है।

और फिर stack में left parenthesis को discard कर देता है।

Step 6 :- यदि Operator आता है तो;

(a) Stack से वह सभी operator को pop करेंगे, जिसका precedence current operator से greater है।

केवल exponential (^) के case में stack से वह सभी operator pop करेंगे जिनका precedence greater या equal है।

(b) Operator को push करते है stack में।

(c) Stack के remain element को pop करेंगे और output string में add करेंगे।

(d) EXIT.

Example

Infix I/P string :- A*B+C

I/P string	stack	output string
C	(	C
+	(+	C
B	(+	CB
*	(+*	CB
A	(+*	CBA
)		CBA*+

o/p string C B A ↑ + +

Reverse o/p string → + + ↑ A B C

Q Infix I/P string :-

A + B / C - D

Reverse string → D - C / B + A

I/P string	Stack	o/p String
D	(	D
-	—	D
C	—	DC
/	— /	DC
B	— /	DCB
+	— +	DCB/
A	— +	DCB/A
)	—	DCB/A+-

o/p string — DCB/A+-

Reverse o/p string → -+A/BCD

Q Infix I/P string - (A+B) / (C-D)

Reverse string ./+AB-CD

reverse string (D-C)/(B+A)

I/P string	Stack	output string
(	(	-
D	((	D
-	(-	D
C	(-	DC
)	empty	DC-
/	/	DC-
(	/(	DC-
B	/(	DC-B
+	/(+	DC-B
A	/(+	DC-BA
)	/	DC-BA+

o/p string → DC-BA+/

Reverse o/p string → /+AB-CD

Q Infix I/P string → A/B ^ C+D

Reverse string → D+C ^ B/A

I/P string	Stack	output string
D	empty	D
+	+	D
C	+	DC
^	+^	DC
B	+^	DCBDCB
/	+/	DCB^DCB^A
A	+/	DCB^A/DCB^A/+

Application of STACK

Assignment

Queue

- Queue एक linear list है, जिसके दो simple end होते हैं :-
  1. elements को Insert करने के लिए।
  2. elements को Deletion करने के लिए।

जिस end से element को insert किया जाता है उसे end को rear end कहते हैं। और

जिस end से elements को Queue से Delete किया जाता है, उसे end को Front end कहते हैं।

// Queue में दो Types के Basic Operation किये जा सकता है :-

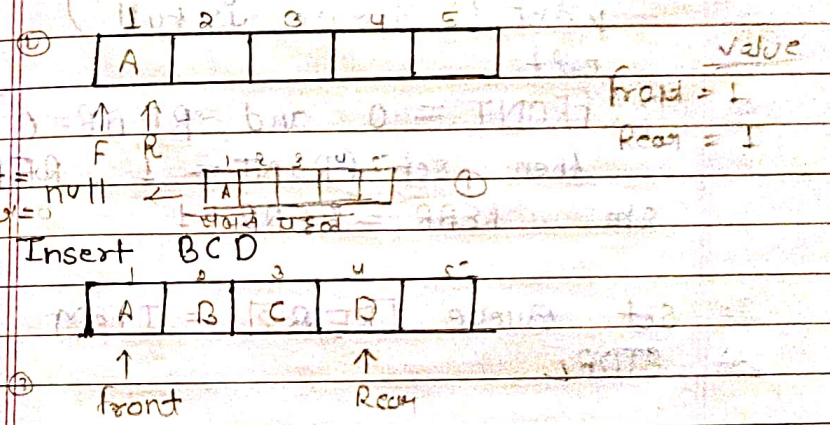
1. Insert an element on queue (Enqueue)
2. Delete an element from queue. (Dequeue)

- real time example of queue is people standing queue for Billigon in shop
- Computer application में Job scheduling में Queue का use होता है।

क्योंकि Queue में पहले insert होने वाला element पहले बाहर होता है इसलिए Queue (First in first out) (FIFO) List कहलाता है।

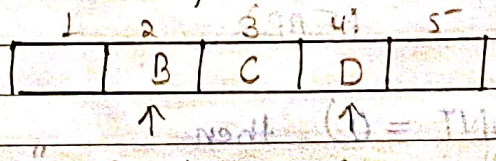
Front :- Front Delete item को show करता है।

Rear :- Rear में बताता है, कि next element insert होगा।

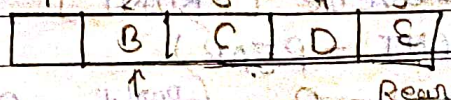


front = 1 Rear = 4

Deletion of one element



Insert E



Output string → DCB^AA/+
reverse +/A^BCD

Evaluation of prefix expression

Algorithm

① Scan the expression from last element.

② Check the current element :

ⓐ यदि वह operand है तो उसे stack में push करेंगे।

ⓑ यदि वह operator ⊕ है तो stack के top से दो elemat pop करेंगे ; यदि A⊗B preform करेंगे जहाँ — A top operand है और B second top operand है।

③ Step 2 repeat करते जाएंगे जब तक traverse नहीं हो जाते है और end में stack में operation का result रहेगा।

Q. Prefix (aythmetic expression)
→ x+69-31

Second.

Scanned elemat	Stack
1	1
3	1, 3
−	2
9	2, 9
6	2, 9, 6
+	2, 15
*	30

Q + 9 * 25

Scanned element	Stack
6	6
2	6, 2
*	12
9	12, 9
+	21

Q −+8/632

Scanned element	Stack
2	2
3	2, 3
6	2, 3, 6
/	2, 2
8	2, 2 8
+	2, 10
−	8

(SIMPLE Queue)

Algorithm for Queue for Insert.

Insert (Queue, N, ITEM, FRONT, REAR)

1. If (REAR = N)
   print ("Queue is full")
   exit
2. If (FRONT = 0 and REAR = 0)
   then set FRONT = 1, REAR = 1
   else: REAR = REAR + 1
3. Set Queue [REAR] = ITEM
4. STOP

Algorithm Queue for Delet.

Delete (Queue, N, ITEM, FRONT, REAR)

1. If (FRONT = 0) then
   print "Queue is empty"
   exit
2. ITEM = Queue [FRONT]
3. If (FRONT = REAR)
   Set REAR = 0, FRONT = 0
   Else
   FRONT = FRONT + 1
4. stop.

Circular queue.

Circular queue में New element को Queue के सबसे पहले location में insert करवाया जा सकता है, यदि Queue का last location full है और Queue के पास New element को insert करवाने के लिए space है।

FRONT = REAR + 1, FRONT = 1 & REAR = N

Algorithm for Circular Queue.

(a) INSERT (QUEUE, N, ITEM, FRONT, REAR)

Rear जिसको point करता है, उसके बाद Rear को update करना है।

1. If FRONT = 1 & REAR = N or FRONT = REAR + 1;
   write "Queue is full"
   EXIT (overflow)
2. If REAR = NULL (खाली) then;
   set FRONT = 1, REAR = 1

Else

If REAR = ~~REAR + 1;~~ N

REAR = 1;

ELSE

REAR = REAR + 1

3) Set QUEUE [REAR] = ITEM

4) EXIT

(Q) DELETE (QUEUE, N, ITEM, FRONT, REAR)

Front (जिसको) point करता है उसे Delete करते हैं

1. If FRONT = ~~Null~~ 0 , REAR = ~~Null~~ 0

then

write " Queue is empty ";

EXIT

2) ITEM = QUEUE [FRONT] ;

3) If FRONT = REAR , then

set FRONT = ~~Null~~ 0 ; REAR = ~~Null~~ 0

Else

if FRONT = N , then

FRONT = 1

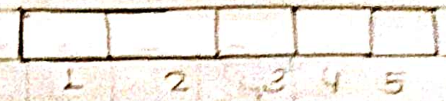
ELSE

FRONT = FRONT + 1

4) EXIT.

Circular Queue

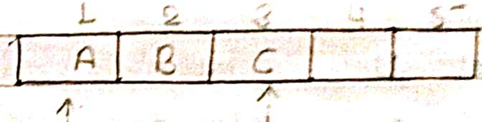
Ex:



F = 0

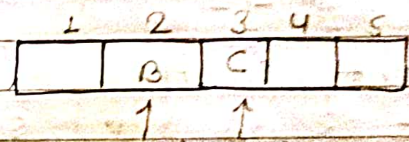
R = 0

① Insert A, B, C



F = 1 R = 3

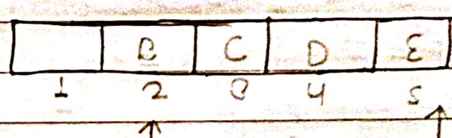
② Delete A



F = 2 R = 3

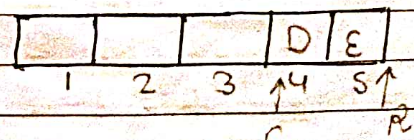
underflow की condition check करते हैं

③ Insert D, E,



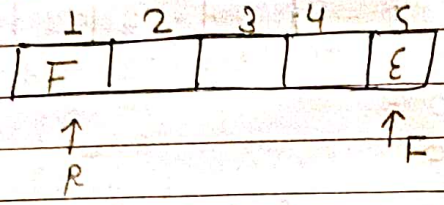
F = 2 R = 5

④ Delete B, C.



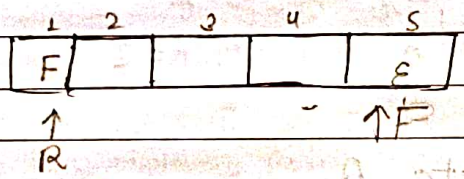
F = 4, R = 5

5) Insert F



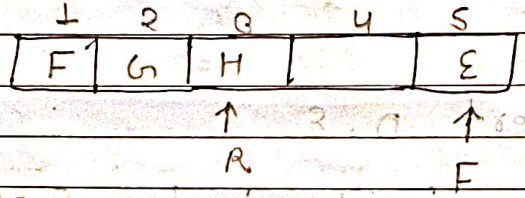
F = 5 , R = 1

6) Delet D



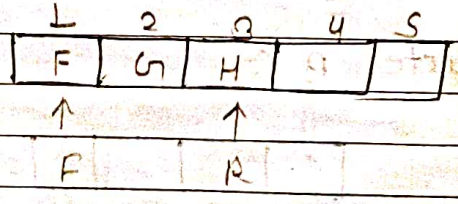
F = 5 , R = 1

7) Insert G, H



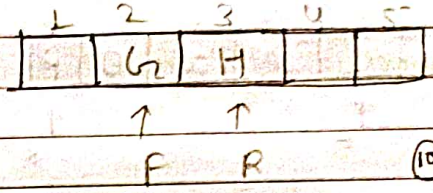
F = 5 , R = 3

8) Delete E



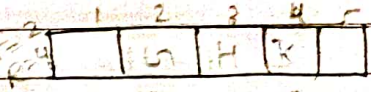
F = 1 , R = 3

9) Delete F

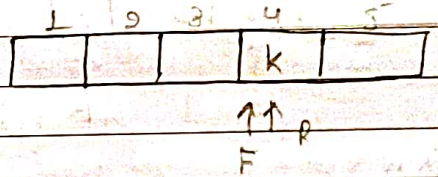


F = 2 , R = 3

10) Insert K

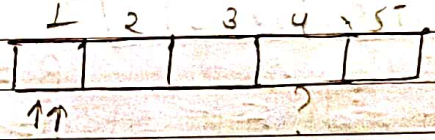


11) Delete G, H



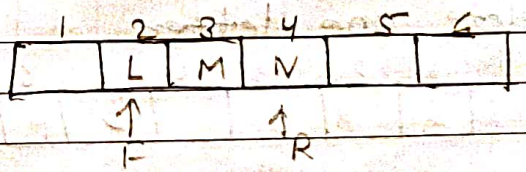
F = 4 , R = 4

12) Delete K



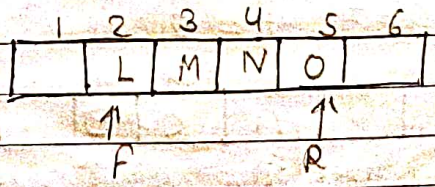
F = 0, R = 0 F-R

Q.2



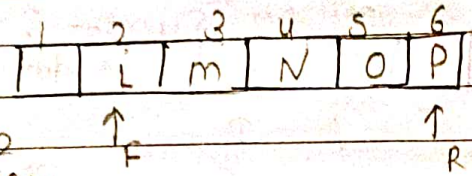
F = 2
R = 4

a) Add O



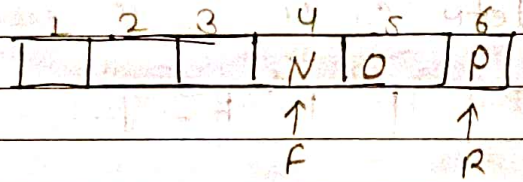
F = 2
R = 5

(b) Add P



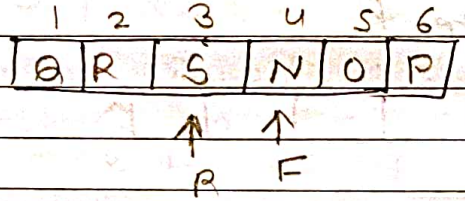
F = 2
R = 6

(c) Delete two letters.



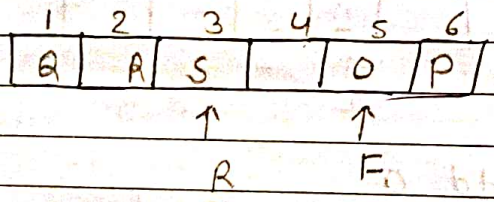
F = 4
R = 6

(d) Add Q,R,S



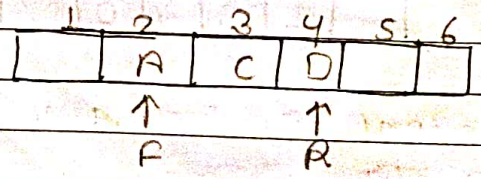
R = 3
F = 4

(e) Delete one letter



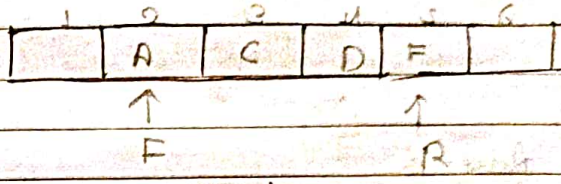
R = 3, F = 5

Q3.



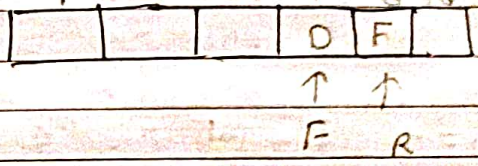
F = 2
R = 4

(i) Add F



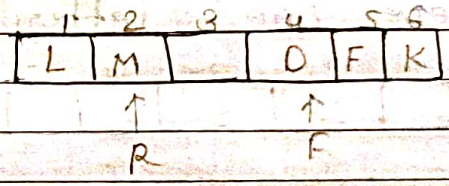
F = 2
R = 5

(ii) Delete 2 letters



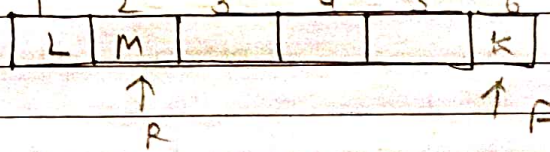
F = 4
R = 5

(iii) Add K, L, M.



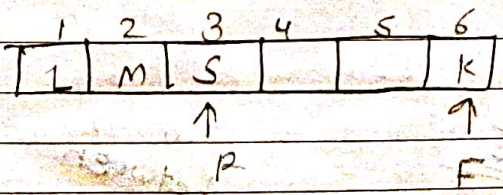
F = 4
R = 2

(iv) Delete 2 letter.



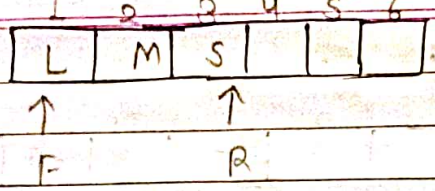
F = 6
R = 2

(v) Add S



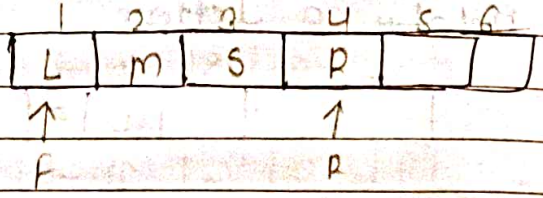
R = 3
F = 6

(vi) Delete one letter.



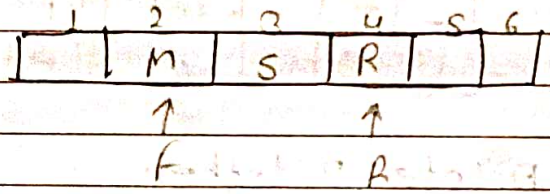
F = 1
R = 3

(vii) Add R



F = 1 , R = 4

(viii) Delete one letter.



F = 2 , R = 4

★ NOTE

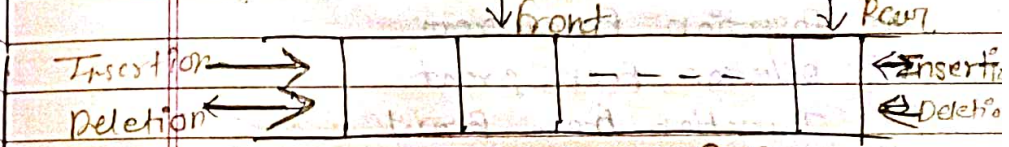
Comparision between of Queue and Stack

Application of Queue - CPU, Disk and printer.

Double ended Queue :- (DEQUE)

DEQUE एक linear list है, जिसमें element को किसी भी end से Insert या Delete करवाया जा सकता है, पर middle से नहीं।

- DEQUE में element के Rear end या front end से Insert करवा सकते है, साथ ही किसी element का Delete Front end या Rear end से करवाया जा सकता है।



Structure of Queue

Type of ~~Queue~~ DEQUEUE :-

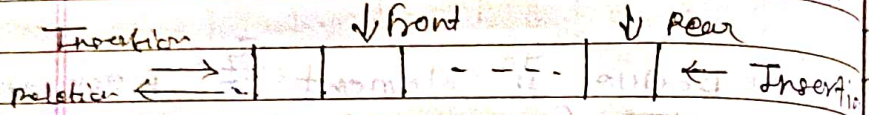
① Input Restricted deque

यह Deque Insertion को केवल एक ही end से allow करता है, जबकि Deletion की list को दोनों end से allow करता है।

Input Restricted deque

② Output Restricted deque

वह Deque Deletion को लिए केवल एक ही end से allow करता है लेकिन Insertion दोनों end से allow करता है।



Output Restricted deque

Possible Operation in deque :-

1) Insertion from rear
2) Deletion from front
3) Insertion from front
4) Deletion from rear.

Algorithm for Insert (in DEQUE)

DQ Insert Rear (Queue, N, ITEM, FRONT, REAR).

1 If REAR = N, then
write "Overflow", exit,

2 If FRONT = Null, REAR = Null
set FRONT = 1
REAR = 1

Else
If REAR = N
set REAR = 1
REAR = REAR + 1;

3. Queue [REAR] = ITEM;
4. End

DQ Insert front (Queue, N, ITEM, FRONT, REAR)

1. If FRONT = 1, then
"Overflow" & exit

2. If FRONT = Null
set FRONT = 1, REAR = 1

Else if FRONT = 1
set FRONT = N

else
FRONT = FRONT - 1

3. Queue [FRONT] = ITEM

4. EXIT

Page No.
Date

Algorithm for Delete (in Queue)

Do Delete front (queue, N, ITEM, FRONT, REAR)

① If FRONT = Null, then
write: "underflow" & exit

② ITEM = QUEUE [FRONT]

③ If FRONT = REAR then,
set FRONT = Null, REAR = Null

else
if FRONT = N, then
set FRONT = 1

④ else, FRONT = FRONT + 1

⑤ end

Front - left
rear - Right

Do Delete Rear (Queue, N, ITEM, FRONT, REAR)

① If FRONT = Null then,
write "underflow!" & exit.

② ITEM = Queue [REAR];

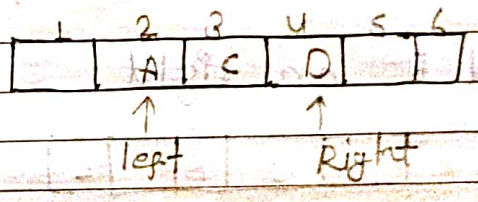
③ If FRONT = = REAR
set FRONT = NULL, REAR = Null

Else if REAR = 1 then,
set REAR = N

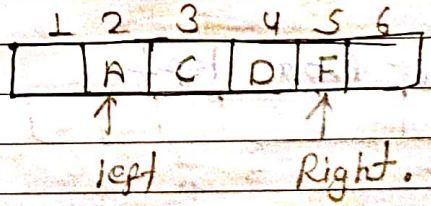
Else REAR = REAR - 1

④ EXIT

Question

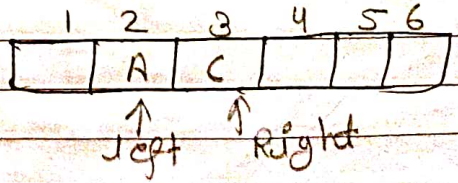


① add & front right (Rear)



Deleteform

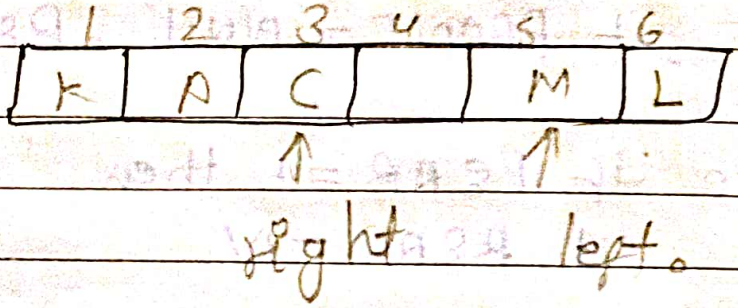
② & Jeftov Right (Rear)



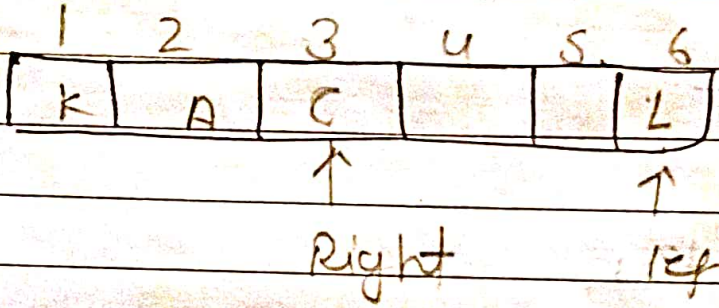
Implement of stack / queue
Application of stack
Application of queue

Page No.
Date

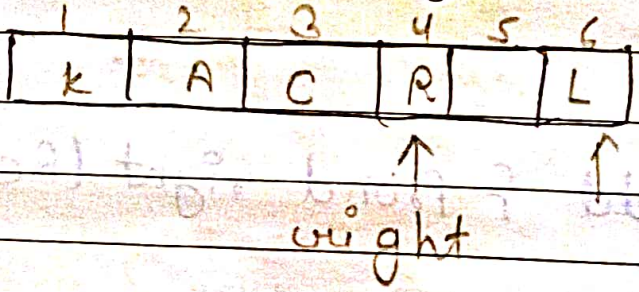
④ Add K, L, M on Front



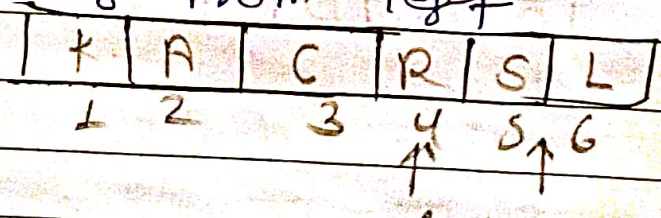
⑤ Delete one letter from left



⑥ Add R on right



⑦ add S from left



Unit 04
Linked List

4.1 Introduction of Linked List =>

- Data processing में सामतौर पर data को Frequently store किया और Data processing को list में organize किया जाता है। इसे एक तरीके से "Array" के रूप में store किया जा सकता है।
- लेकिन Array में कुछ Disadvantages Certain (कुछ) होते हैं।
Array में elements को Insert and delete relatively expensive हो सकते हैं क्योंकि Array सामान्यतः एक memory space का block ले लेता है। • इस स्थायी Data structure का का size (सीधे) simply double or triple नहीं किया जा सकता है।

Linked List

Memory में एक list को store करने का एक और तरीका है।

जिसमें, list के प्रत्येक element में एक "link" or "Pointer" contain एक field होता है जिसमें list के अगले element का पता होता है।

इस प्रकार, list के अनुक्रमित elements को memory में समाप्त space ~~का ले~~ लेने की आवश्यकता नहीं होती है। इससे list में elements को ~~adjust~~ Insert और delete करना easy हो जाता है। इस प्रकार की data structure को "Linked list" कहा जाता है।

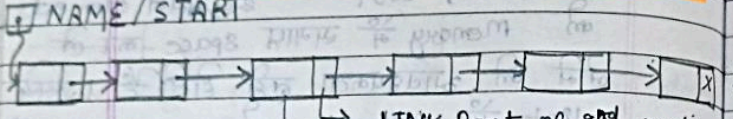
Linked list का use करके हम Array को remove कर सकते हैं

- linked list or one-way list, जिसे nodes कहा जाता है,
- A linear collection of data elements जिसमें linear order, pointers के माध्यम से दिया जाता है। अर्थात
- एक node दो हिस्सों में divide होता है।
  - The first part contains the information of the element.
  - The second part contains the address of the next node in the list

LINK Part == NEXT Pointer

So the 1st Part is known as "INFO Part" and 2nd Part is known as "LINK" Part.

Fig below is a schematic diag of linked list with 6th nodes



① Each node is pictured with two parts, जिसमें left part node represent करता है: the Information part of the node, जिसमें कुछ Record की Data items हो सकते हैं।
और,
Second part node represent करता है: the next pointer filed or link field. और एक तीर होता है, जिससे list में अगले node को और इशारा किया जाता है। या किसी node के Pointer में एक विशेष मूल्य होता है जिसे null pointer कहा जाता है। which is any invalid address. जो कोई भी invalid address को व्यक्त करता है।

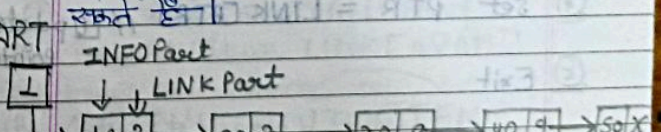
② NULL Pointer :- Last node के LINK field में कोई भी contain NULL नहीं होता, rather than यह NULL होता है। NULL Pointer list के end का address को रखता है / पता करता है।

Care of Single link list

* Representation of linked list in Memory

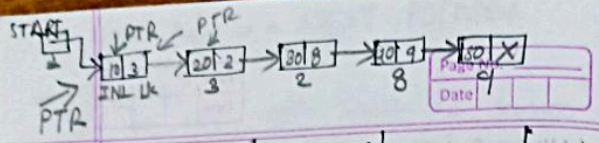
LINK list को memory में बनाये रखने के लिए दो linear arrays की help ली जाती है -
① INFO ② LINK !

Example :- अगर हमें five nodes वाली एक linked list को कैसे store करना है, तो हम memory में इसे निम्नलिखित रूप में store कर सकते हैं।



INFO No.	INFO
1	10
2	30
3	20
4	
5	

LINK No.	LINK
1	3
2	8
3	2
4	
5	



6		6	
7		7	
8	40	8	9
9	50	9	0
10		10	

Write an algorithm to print an element of link list

PRINT LIST (INFO, LINK, START)

1. set PTR = START
2. Repeat steps 3 & 4 while PTR ≠ NULL
3. Write INFO [PTR]
4. set PTR = LINK [PTR] // [Updates Pointer]
5. Exit

सबसे पहले Pointer Start को Point करेगा PTR = START फिर steps 3&4 Repeat होगा जब तक कि PTR ≠ NULL न हो जाये। फिर INFO [PTR] होगा फिर set करेंगे PTR = LINK [PTR] उसके बाद ये Process Repeat होते रहेगा फिर जब तक NULL न हो जाय फिर EXIT.

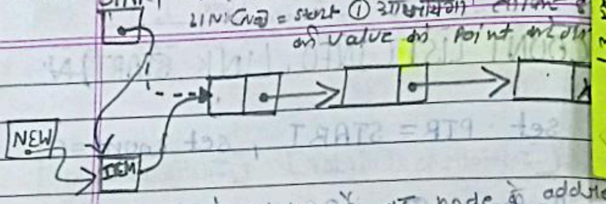
COUNT LIST (INFO, LINK, START)

1. set PTR = START, set count = 0 [Initializes Pointer] [Initialize counter]
2. Repeat step 3&4 while PTR ≠ NULL
3.
4. count = count + 1 ; [Increase count by 1]
4. PTR = LINK [PTR] [Updates pointer]
5. Write count , EXIT.

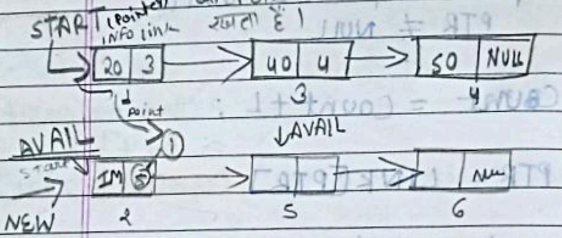
INFO FIRST (INFO, LINK, START, ITEM)

1. IF AVAIL = NULL overflow & exit.
2. set NEW = AVAIL and AVAIL = LINK [AVAIL]
3. set INFO [NEW] = ITEM
4. set LINK [NEW] = START
5. Set START = NEW
6. EXIT

तो पहले start pointer को save करेंगे LINK[NEW] में (start की value) ① को लेकर START में इसकी value डालेंगे LINK[NEW] = START ① आ जायेगा की value को point करेगा



जो pointer है वह node के address को रखता है।



सबसे पहले यहाँ पर यदि AVAIL = X (null है क्या) then overflow & Exit फिर पहले AVAIL को NEW point करेगा NEW जो है ITEM को point करता है। इसमें INFO[NEW] = ITEM ① और LINK[NEW] = START को ② point कर रहा है तो फिर AVAIL आगे बढ़ेगा। तो AVAIL 5 होगा तो NEW का AVAIL को point करेगा इसके बाद NEW = ITEM रहेगा INFO[NEW] = ITEM और LINK[NEW] = START फिर start में NEW रहेगा उसके बाद इसी तरह Process repeat होगा जब तक AVAIL = NULL होगा फिर overflow होकर EXIT.

(margin notes: AVAIL को point करता है। AVAIL में जो पुरानी value है जैसे यहाँ 2 है उसे secure करके रखेंगे और फिर हमें एक new pointer मिलेगा और ये AVAIL की value को point करता है new AVAIL में आ जाएगा AVAIL की value save होगी उसके बाद AVAIL आगे बढ़ेगा।)

AVAIL आगे बढ़ने फिर इसकी value 5 हो जायेगी क्योंकि AVAIL में 5 की value थी AVAIL = LINK[AVAIL]

अब हमारे पास इस node का Access है new pointer के through हमें Item डालना है। हर node में two part हैं। INFO & LINK हमें पता है कि हर node को access pointer के through ही करते हैं। ③ जैसे ही ③ point Item save हो जाते हैं, new में AVAIL आगे बढ़ेगा अब हमें एक पहला node बनाना है।

INSLOC (INFO, LINK, START, AVAIL, LOC, ITEM)

(first node में insert करना है।)

① IF AVAIL = NULL
overflow & EXIT,

(हमें linklist बनाना है और इसमें खाली node चाहिए जिसमें AVAIL को set करना होगा)

② Set NEW = AVAIL and AVAIL = LINK[AVAIL].

③ Set INFO[NEW] = ITEM

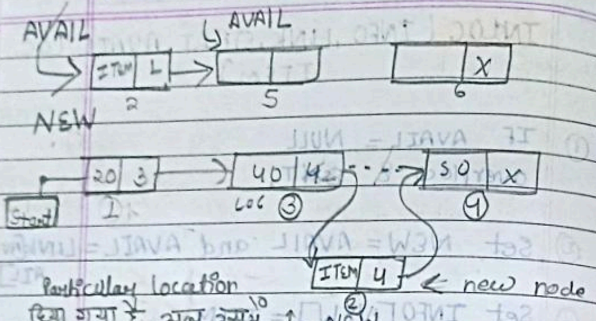
④ If LOC = NULL; then:
set LINK[NEW] = START and
START = NEW.

Else.
set LINK[NEW] = LINK[LOC]
& LINK[LOC] = NEW

⑤ Exit

सबसे पहले set करना है AVAIL की value NEW में फिर AVAIL = LINK[AVAIL] (आगे बढ़ेगा) ~~स्टोर रहेगा इसके बाद set करना है~~ INFO[NEW] = ITEM में फिर Condition लगाना है जिसमें LOC = NULL हो तो set करना है LINK[NEW] = START में और START में NEW रहेगा फिर set LINK[LOC] में LINK[NEW] रहेगा और LINK[LOC] = NEW आ जायेगा।

यह Process तब तक होगा कि AVAILL = NULL में होगा तब overflow होकर exit हो जायेगा।



Particular location दिया गया है, अब उसमें insertion कैसे करना है।

INSLAST (INFO, LINK, START, AVAIL, LOC, ITEM)

→ LAST मे insert करायेगा तो Traverse करते-करते जायेंगे।

① IF AVAIL = NULL
overflow & Exit

② Set NEW = AVAIL &
AVAIL = LINK[AVAIL]

③ Set INFO[NEW] = ITEM
& set LINK[NEW] = NULL

④ PTR = START

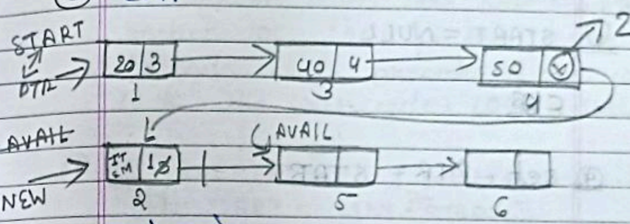
⑤ Repeat step ⑥ while LINK[PTR] ≠ NULL

⑥ Set PTR = LINK[PTR]

इसमें जब तक LINK[PTR] NULL न हो जाये। तब तक

⑦ LINK[PTR] = NEW

⑧ EXIT



सबसे पहले

④ ← If PTR START को point कर रहा है। तो Repeat ⑥ step जब तक कि LINK[PTR] ≠ NULL न हो जाये। then set करेंगे LINK[PTR] को PTR में रखेंगे और NEW को LINK[PTR] में रहेगा। (element)

→ पहले INFO[NEW] = ITEM रहेगा और set LINK[NEW] = NULL तब

- DELETE FIRST (INFO, LINK, START, AVAIL)

① If START = NULL
underflow & EXIT. (एक से ज्यादा node/ अधिक LINK[Start] = NULL)

② ITEM = INFO[START] (Start → 20 | X)

③ IF LINK[START] = NULL (link of start में AVAIL की value point कर रहा है।)
पहला ITEM को delete

AVAIL = freenode निकालने के लिए
START = pointer को point करने के लिए
pointer = [PTR] को दर्शाया जाता है।

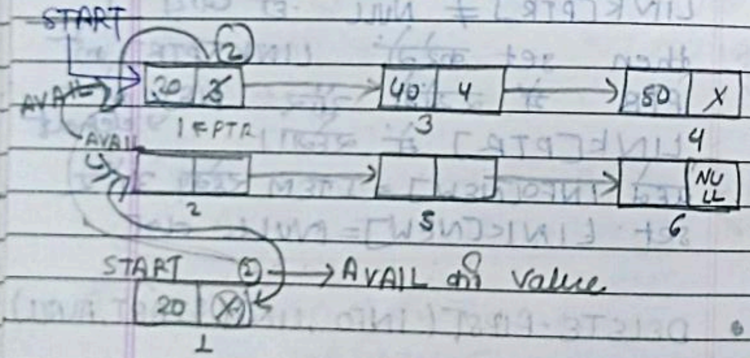
(a) LINK [START] = AVAIL &
AVAIL = START

(b) START = NULL

. Else

(a) set PTR = START

(b) START = LINK [START]
set [LINK[PTR] = AVAIL &
AVAIL = PTR



- LINK[PTR] 1 AVAIL start को point कर रहा है।
AVAIL 2 अब (Start 1); AVAIL 1 हो जायेगा
AVAIL की value 3 अगर यदि START = NULL है तो,
को LINK में point करायेंगे (4) • सेट करेंगे START जो है pointer को point करेगा और LINK[START] में AVAIL की
value START में है वो set करेंगे।
pointer AVAIL की value को point करेगा।

IMP जो location (LOC) N को Delete करता है।
LOC is a Location of (node) node N in link list.
LOCP is a location of node preceding N.

DEL (INFO, LINK, START, AVAIL, LOC, LOCP)

This algorithm deletes the node N with location LOC. LOCP is the location of the node which precedes N or, when N is the first node, LOCP = NULL.

(1) If LOCP = NULL [Delete first node]
set START = LINK [START]

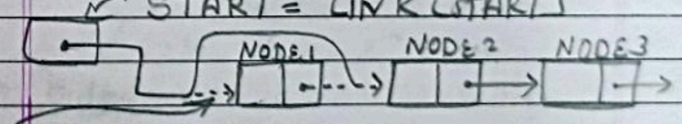
Else
Set LINK[LOCP] = LINK[LOC]
[Deletes node N]

(2) Set LINK[LOC] = AVAIL &
AVAIL = LOC

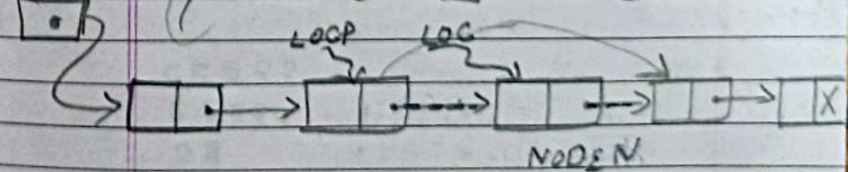
LOCP = NULL
पहला node delete हो रहा है।

(3) Exit

START = LINK [START]



LINK[LOCP] = LINK[LOC]



NOTE:— &, address %d
*, value %u
~~int~~ %s
• Pointer → इसमें address को रखता है।

Pointer.CPP

```
# include<stdio.h>
# include<conio.h>
void main()
{
 clrscr();
 int i=3;
 int *j;
 j=&i;
 printf("%u",&i);
 printf("\n%d",i);
 printf("%u",&j);
 printf("\n%u",j);
 printf("\n%d",*(&i));
 printf("\n%d",*j);
 getch();
}
```

Output

```
65524
3
65522
65524
38
3
```

Second Pointer.cpp

```
# include<stdio.h>
# include<conio.h>
void main()
{
clrscr();
int i = 3;
int *j;
int **k;
j = &i
k = &j;

printf("%d", &i);
printf("\n%d", j);
printf("\n%u", *k);
printf("\n%u", &j);
printf("\n%u", k);
printf("\n%u", &k);
printf("\n%d", *(&i));
printf("\n%d", *j);
printf("\n%d", **k);
getch();
}
```

Output

```
65524
65524
65524
65522
65522
65520
```

3
3
3

Array of Pointer, to Array

```
# include<stdio.h>
# include<conio.h>
void main()
{
clrscr();
int a[5] = {10, 15, 20, 25, 30};
int *p; i=0;
p = &a[0]
while(i<5)
{
printf("%u\n", &a[i]);
printf("%d\n", *p);
i++;
p++;
```

Output

```
65516
10
65518
15
65520
20
65522
25
65524
30
```

All Pointer.CPP / ~~Float.CPP~~

```
# include<stdio.h>
# include<conio.h>
void main()
{
clrscr();
float n, *p;
int k, *q;
char s, *a;
n = 3.4;
p = &n;
k = 7;
q = &k;
s = 'n';
a = &s;

printf(" address of Float \n%u",
                    p);
p++;
printf("\n%u", p);
p--;
printf("\n%u", p);
printf("\n%u", q);
q--;
printf("\n%u", q);
q++;
printf("\n%u", a);
a++;
printf("\n%u", a);
getch();
}
```

o/p

Float.CPP

```
# include <stdio.h>
# include <conio.h>
void main()
{
 clrscr();
 float n, *p;
 int k, *q;
 n = 3.4;
 p = &n;
 k = 7;
 q = &k;
 printf("\n%u", p);
 p++;
 printf("\n%u", p);
 p--;
 printf("\n%u", p);
 printf("\n%u", q);
 q--;
 printf("\n%u", q);
 q++;
 printf("\n%u", q);
 getch();
}
```

output

LINK SEARCH (ITEM, PTR, LOC, LOCP, INFO, LINK, START)

* यह LINK SEARCH Algorithm node N का Location LOC को find (search) करता है, जो ITEM को रखता है। LOCP node N का preceding node का location है।
* यदि ITEM LINK List में नहीं है, तो Algorithm LOC = NULL set कर देता है।
* यदि ITEM, first node में है, तो LOCP = NULL पे set कर देता है। तो LOC START होगा।

① If START = NULL, then set LOC = NULL and LOCP = NULL and exit.

② If INFO[START] = ITEM then, set LOC = START and LOCP = NULL and exit.

③ Set SAVE = START & PTR = LINK[START]
// Initialize

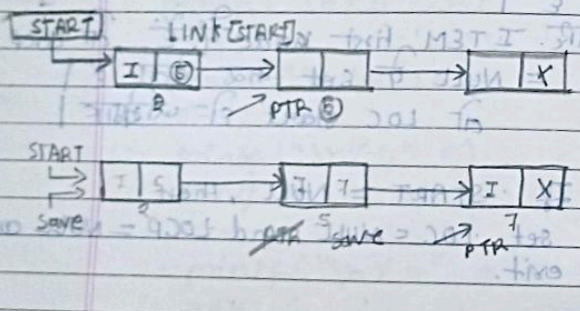
④ Repeat step ⑤ while PTR ≠ NULL

⑤ If INFO[PTR] = ITEM
LOC = PTR, LOCP = SAVE and return

⑥ SAVE = PTR and
PTR = LINK[PTR]

⑥ LOC = NULL

⑦ Exit.



Two way link list (Doubly linked list)

Two way link list Data element का linear collection है, जिसे nodes कहते हैं। यहाँ हर node N तीन parts में devided रहता है।

① INFORMATION field 'INFO' जो Node N का Data रखता है।

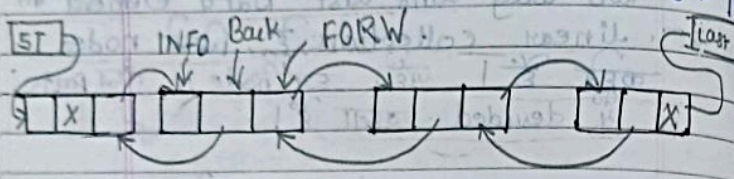
② LINK field 'Forward' "FORW" जो List में next node का location रखता है।

③ LINK field 'Back', जो Preceding node का address / location रखती है।

Two way link list को दो direction में Traverse किया जा सकता है, Forward direction जो list के beginning से end तक जाता है या Back ward direction जो list के end से Bigning तक जाता है।

Two way linklist में यदि किसी node N का location LOC दिया गया है तो उस node के next node और Preceding node को तुरंत access किया जा सकता है।

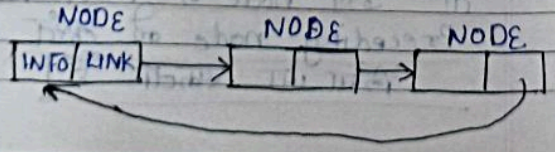
जिसमें उस node N को list से Delete करने के लिए हमें list को Traverse करने की आवश्यकता नहीं पड़ती।



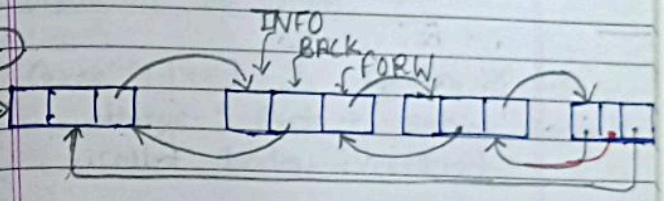
Circular linked list

यह link list का Variation है, जिसमें last node का link field, list के first node का address रखता है।
Means Last node का link field null को Point करने के जगह link list के Bigning node को Point करता है।
दोनों Single link list को Double link list में बनाया जा सकता है।

Single link list as a circular link list.



Doubling link list as a circular link list.



Unit 05

Graph And Tree ⟶ non linear DS. है।

* Definition of graph ⇒

Graph 'G = [V, E]' को Consists of two things : First → set of elements called nodes (vertices).

Second :- Set E of edges, such that each edge e in E is identified with unique pair [u, v] of nodes in V, denoted by e = [u, v]

Graph vertex, Edge का collection है। यही है set.

V = { V_1, V_2, V_3, V_4 }

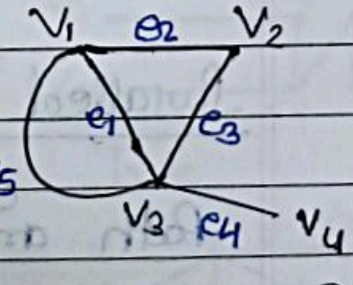
E = { e_1, e_2, e_3, e_4 }

e_1 = { V_1, V_3 }

e_2 = { V_1, V_2 }

e_3 = { V_2, V_3 }

e_4 = { V_3, V_4 }



V के अंदर unique Pair Identify होती है।

Connected Graph → एक Vertices से दूसरे Vertices तक जाने के लिए Path होता है, means simple path accessable / reachable होती है।

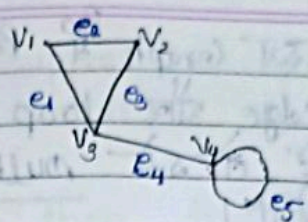
Graph जो है, Connected Graph कहलाती है, यदि graph 'G' के हर किसी भी दो node / Vertices के बीच simple path होता है।

disconnected Graph →

एक Vertices से दूसरे Vertices तक जाने के लिए Path नहीं होता है, means simple Path accessable / reachable नहीं होता है इसे ही disconnected graph कहा जाता है।

Parallel edge → Same pair of vertices के बीच Path को Parallel edge कहते है।

loop edge → जिसके identical शुरुआत और समाप्त (Start / end) Point होती है उसे ही loop edge कहते है।

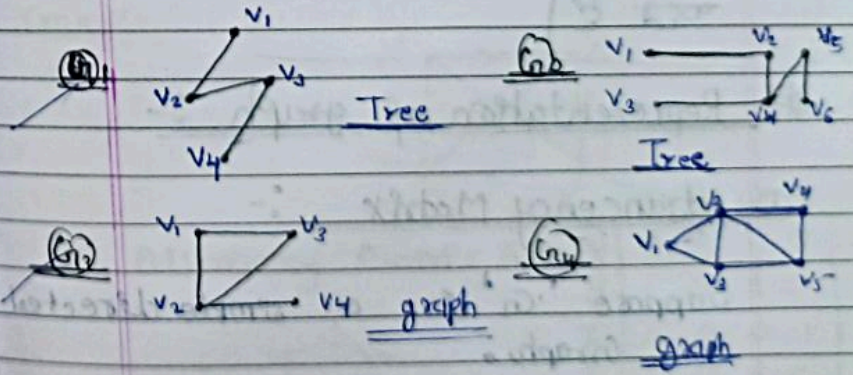


यहाँ पर V4 start and end Point है means ये loop edge कहेंगे।

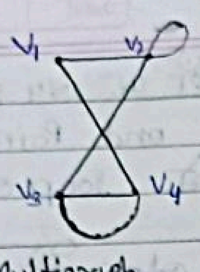
edge → हर एक edge में two unique vertices के pair की Path होती है, जिसे edge कहते है।

Introduction to Tree ⇒ connected and cyclick नहीं होगा तथा Parent and child वाला directory बनेगी। इसे Tree कहेंगे।
इसमें V_1 से V_2 जा सकते है लेकिन V_2 से V_1 नहीं जा / जा सकते है।
इसमें cyclick नहीं बनना चाहिए।

* A connected graph without any cycle is called tree graph



(G₂)



इस Graph में Parallel edge और loop edge भी हैं, तो इसे multigraph कहेंगे।

Multigraph

Direction graph →

Direction graph, जब उसमें graph में edge हों, direction show करता हो, उसमें तब उसे Direction graph कहेंगे।

Non Direction graph →

जब Graph में edge use नहीं होती या direction show नहीं करती तो उसे Non direction graph कहते हैं।

Representation of graph :-

(1) Adjancency Matrix :-

Suppose 'G' is a simple directed Graph.

nodes (vertices) of

Suppose Graph 'G' have been called $(V_1, V_2, V_3, V_4 \ldots V_m)$

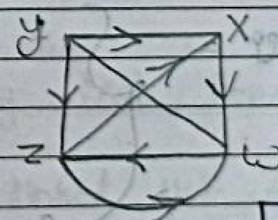
then the adjancecy matrix $A-(a_{ij})$ of Graph G is $m \times m$ matrix.
Graph G जिसका adjancecy matrix A जिसका a_{ij} हो तो एक $m \times m$ matrix है।
which is defined as follows:-

* v_i और v_j के बीच edge है तो a_{ij} की value 1 होगी।

$$a_{ij} = \begin{cases} 1 & \text{if } v_i \text{ is adjancecy to } v_j \text{ i.e. there is an edge } (v_i, v_j) \\ 0 & \text{otherwise} \end{cases}$$

इस Matrics में 0 और 1 होते हैं और जिसका Tree matrix कहलाता है, इसे bit Matrix or Boolean Matrix भी कहा जाता है।

Example



Adjancecy Matrix A =

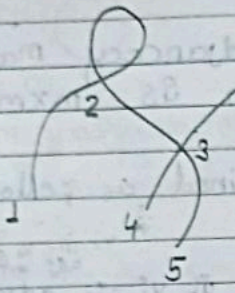
	x	y	w	z
x	0	0	1	0
y	1	0	1	0
w	0	0	0	1
z	1	0	1	0

4×4

Boolean Matrix

Adjacency Matrix A :-

	1	2	3	4	5	6
1	0	1	0	0	0	0
2	1	1	1	0	0	0
3	0	1	0	1	1	1
4	0	0	1	0	0	0
5	0	0	1	0	0	0
6	0	0	1	0	0	0



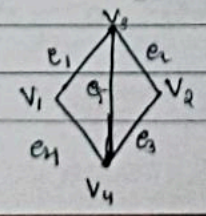
② Incidence Matrix :-

Suppose G is undirected a graph having n vertices & m edges.

Incidence matrix निम्नलिखित $C=[c_{ij}]$ से तो $n \times m$ matrix defined होते हैं।

$$c_{ij} = \begin{cases} 1, & \text{if line vertex } v_i \text{ is incident by edge } e_j \\ 0, & \text{otherwise.} \end{cases}$$

Example :-



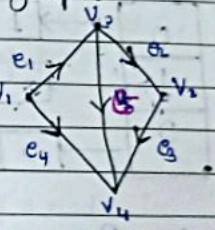
Incidence matrix में (column m(edge)
Row Vertices (n) Represent

	e_1	e_2	e_3	e_4	e_5
v_1	1	0	0	1	0
v_2	0	1	1	0	0
v_3	1	1	0	0	1
v_4	0	0	1	1	1

4×5

In case of directed graph

c_{ij} = 1, if v_i is initial vertex of e_j
−1, if v_i is final vertex of e_j
0, if v_i is not incident on edge e_j



	e_1	e_2	e_3	e_4	e_5
v_1	1	0	0	1	0
v_2	0	−1	1	0	0
v_3	−1	1	0	0	1
v_4	0	0	−1	−1	−1

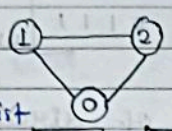
4×5

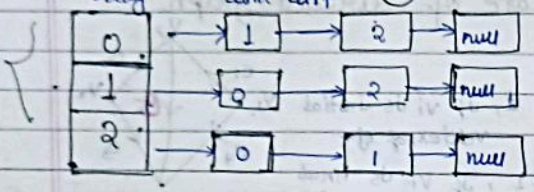
⇒ Adjacency List Representation.

- Array of linked list is used to store edges between two vertices.
- Size of array is equal to the number of vertices.

(8) यदि graph में number of vertices है, तब Array का size n है।

Undirected Graph :



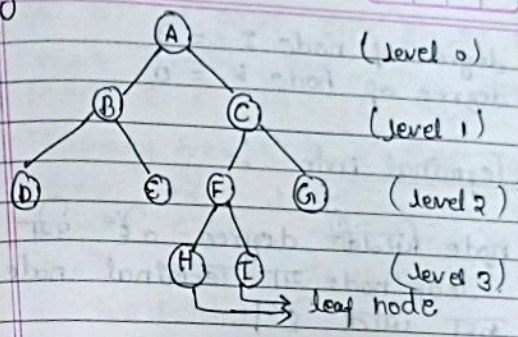


* Adjacency list, Array of linklist

Tree एक non linear Data structure है, जो different Data items के बीच hierarical relationship को represented करने के लिए use होता है। Tree में एक root node होता है, जो Tree के Top में होता है। इस node का Parent node नहीं होता है केवल यही एक node है जिसका Parent node नहीं होता है।

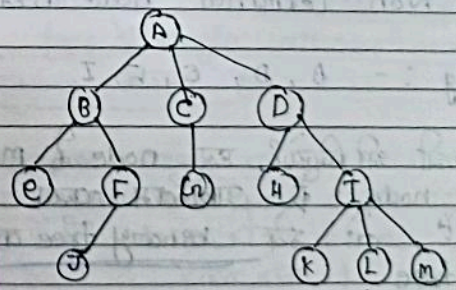
हर Graph Tree नहीं होता है लेकिन हर tree Graph होता है।

Eg →



* degree of node :
number of subtree of a node :
degree of node A = 2
degree of node B = 2
degree of node C = 2
degree of node F = 2

Eg →



* degree of node, number of subtree of a node
• Maximum degree of node = 3.
degree of node A = 3
degree of node B = 2
degree of node C = 1

degree of node I = 3
degree of node K = 0

* Terminal node :-

node जिसकी degree 0 हो उसे leaf node या Terminal node कहा जाता है।

Eg :→ E, J, G, K, L, M.

* Non Terminal node :-

वह root node को छोड़कर कोई भी node जिसकी Degree 0 नहीं है उसे ही Non Terminal node कहते है।

Eg :- B, D, C, F, I.

~~कभी भी~~ जिसमें, हर node के maximum node के अधिकतम चलन 2 होते है, तो उसे binary tree कहते है

* Type of tree

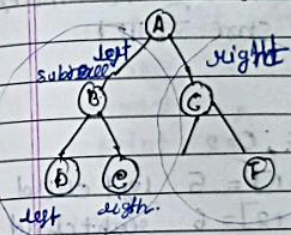
#① Binary tree :-

Data Items का एक tree का एक finite set है।

होता

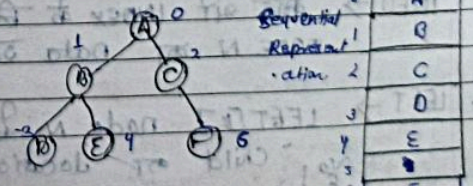
जो या तो खाली होता है, या एक root कहलाने वाले एक Items से मिलता है। और दो अलग-अलग Binary tree होते है जिन्हें Binary tree के left subtree और right subtree कहा जाता है। इसमें अधिकतम दो child होंगे।

two disjoint



Sequential Representation of binary tree ~~के~~ single linear array को use करता है।

- Suppose array का नाम (T) Tree है। means वो root होगा तो Tree[0] में store होगा means root node save होगा।



* यदि कोई भी NODE N Array में Tree[k] है तब उसके लिए —

In left child
Tree [2*k+1] &

In right child
Tree [2*k+2] वाले index array में store रहेगा।

Example

Tree[K] = C = 2
Tree[2*2+1] = 5 left child
Tree[2*2+2] = 6 right child

Tree[K] = B = 1
Tree[2*1+1] = 3 left child
Tree[2*1+2] = 4 right child

इस Binary tree में तीन, Parallel array का use किया जाता है-
(INFO, LEFT, RIGHT)

INFO → कोई भी INDEX के लिए INFO[k] NODE N का Data रखता है।

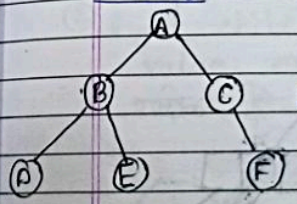
LEFT → LEFT[k] node N के Left Child का location रखता है।

RIGHT → Right of k, RIGHT[k] node N के right child का location रखता है।

Example :-

* Array में Data को sequential store कर सकते हैं और Link list का यह जरूरी नहीं है कि उसे sequential store करे but link list में sequence, pointer के through point करते हैं।

Linked



	INFO	LEFT	RIGHT
1		1	1
2	E	2 NULL	2 2
3	B	3 8	3 2
Root → 4	A	4 3	4 14
5	F	5 NULL	5 NULL
6		6	6
7	C	7 NULL	7 5
8	D	8 NULL	8 NULL
9		9	9
AVAIL 10		10	10

• AVAIL freelink list का link list बनाता है।
• Like किसी को जाने के लिए Point करते हैं।

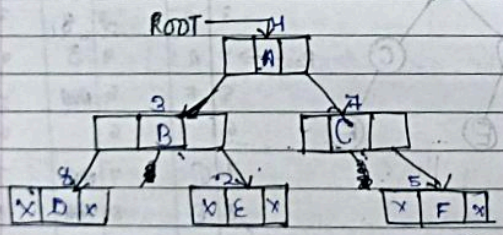
A का left child B है।
right child C है।

B का left child D है।
right child E है।

C का left child Null है।
right child F है।

D का left child Null है।
right child null है।

E का left child null है।
right child null है।



#(2) Binary Search Tree

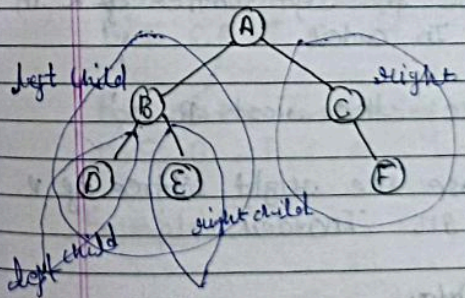
Binary search tree which has a property है, जिसकी यह property होती है; कि किसी भी node n के left sub tree के सभी elements इस node n के content / value उससे छोटे होते है व इस node n के right sub tree के सभी elements value / content से Greater होते है।

Binary search tree एक Binary tree है, जो या तो empty होती है या निम्न Rules को follow करती है।

Rules

① left child या left subtree की value Root tree से कम होगी।

② Right child या Right subtree के node की value root tree के बराबर या अधिक होगी।



Traverse means elements ko access karna.

★ Traversing in Binary Tree.

1. Pree order
2. In Order
3. Post order

(1) Pree Order :-

(i) Process root node R.

(ii) Traverse left subtree of R in Pree order

(iii) Traverse right subtree of R in Pree order.

(2) In Order

(i) Traverse the left subtree of R in In order.

(ii) Traverse the root R.

(iii) Traverse the right subtree of R in Inorder.

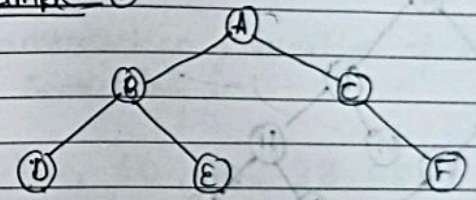
(3) Post Order.

(i) Traverse the left subtree of R in Post order

(ii) Traverse the right subtree of R in Post order.

(iii) Process the root R.

Example (1)

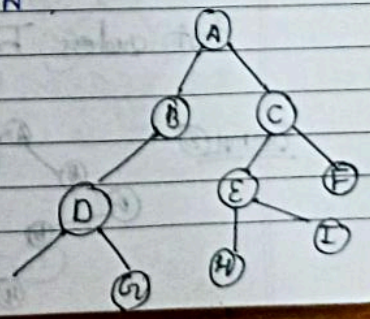


Preorder Traversing
A, B, D, E, C, F

Inorder Traversing
D, B, E, A, F, C

Post order Traversing.
D, E, B, F, C, A

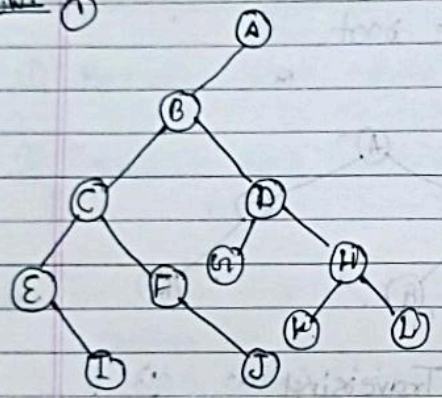
Example (2)



Pree order Traversing
A, B, D, G, C, E, H, I, F

Inorder Traversing
D, G, B, A, H, E, I, C, F

Ans (1)

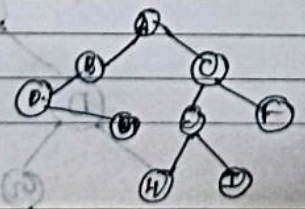


Preorder Traversing
A, B, C, E, I, F, J, D, G, H, K, L

Inorder Traversing
E, I, C, F, J, B, G, D, H, K, L, A

Post order Traversing.
I, E, J, F, C, G, K, L, H, D, B, A

Exam (2)



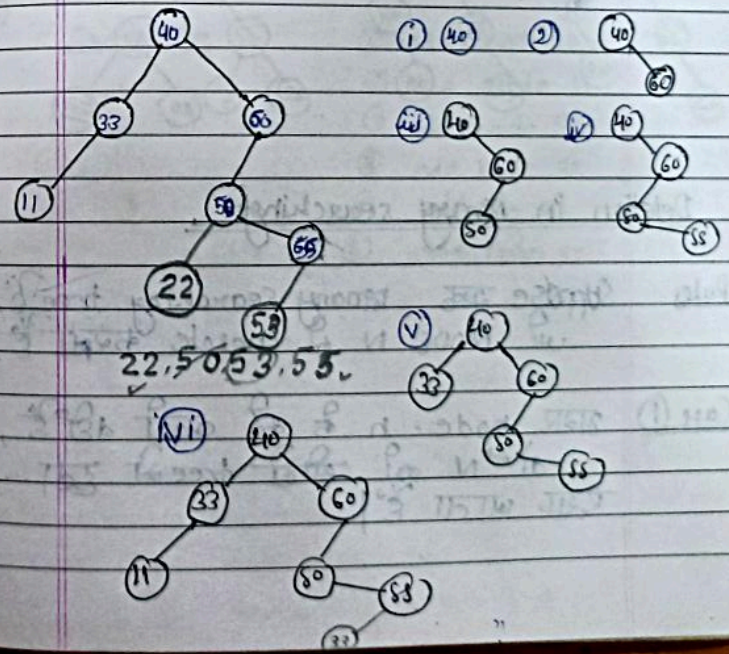
Pree order Traversing
A, B, D, G, C, E, H, I, F

Inorder Traversing.
D, G, B, A, H, E, I, C, F

Post order Traversing
G, D, B, H, I, E, F, C, A

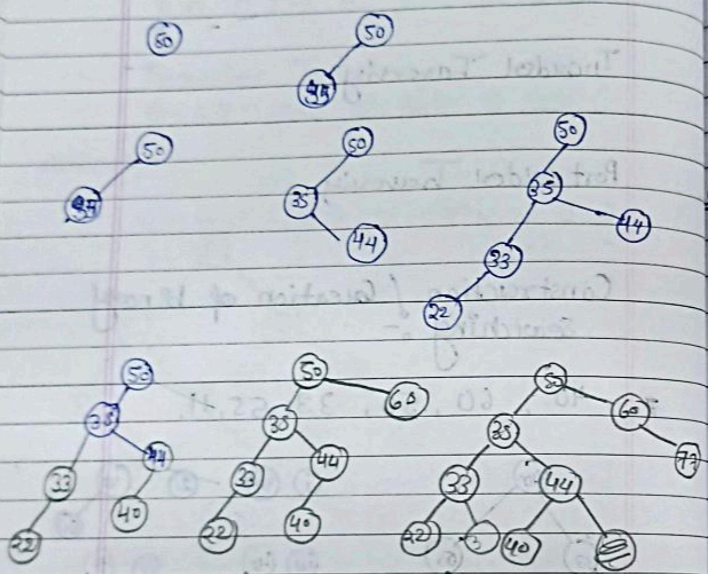
Construction / Creation of binary searching :-

40, 60, 50, 33, 55, 11.



22, 50, 53, 55.

50, 33, 44, 22, 77, 35, 60, 40,



Deletion in Binary searching.

Rules Suppose एक Binary searching tree है, जो NODE N को Delete करना है।

Case (1) अगर node n के को बच्चे नहीं है, तो node N को सीधे tree से हटा दिया जाता है।

Case (2) अगर Node N के पास केवल एक child है, तो Node n को सीधे पेड़ से हटाया जाता है। और Node-n के स्थान पर Node n के केवल child के location को allocate किया जाता है।

Case (3) अगर Node N के पास दो बच्चे हैं, तो मान लें कि, N का in-order successor ~~S[N]~~ है, तो Node N को tree से हटाया जाता है, Node-n को tree से ~~...~~ S[N] को पहले हटाकर और फिर node N को tree में node S[n] के location पर रखकर।

Example :

1. add node 20
2. add node 15
3. add node 88
4. add node 22
5. add node 22
6. add node 75

Pointer :-

यह एक Variable है, जो किसी अन्य Variable के Memory Location को Point करता है।

Pointer एक Programming Concept है, जो कि memory location को refer करता है। हर Pointer एक Particular data type का address store करता है, जैसे => integer, Character, float etc.

Pointer का use memory management, data structures, और low level languages में Pointer का इस्तेमाल किया जाता है।

Operation on pointers. (और detail example Pointer chapter में है।)

⊥ Pointer पर कुछ मुख्य Operations होते हैं :-

① Dereferencing (*) : Pointer के through memory address में value को retrieve करने का Process dereferencing कहलाता है।

इसके लिए Pointer के पहले * Operator का use होता है। जैसे कि :-

Example

```
int *P;   // Pointer declare Kiya
int num = 10;
P = & num; // Pointer को num के address से initialize kiya
Count << " value at address pointed by P : " << *P << endl; // Dereferencing
```

इस Example में, *P हमें num की value देता है।

② Address of (&) : Address of Operator (&) का use variable या object का address निकालने के लिए होता है। जैसे कि :

Example :-

```
int num = 10;
int * P = & num; // Pointer को num के address से initialize किया।
```

इस Example में &num variable का address को रखता है।

Page No. ______ Date

(3) Arithmetic Operations : Pointer arithmetic का use Pointers के addresses को manipulate करने के लिए होता है। इसमें Pointer को integers के साथ add, subtract, increment या decrement किया जा सकता है। जिससे Pointer अन्य data type के दूसरे element को Point करे।

Example

```
int a[5] = {1, 2, 3, 4, 5};
int *P = a ;  // Pointer को array के 1st element के address से initialize करना
P ++;         // Pointer को array के दूसरे element पर Point करने के लिए increment करना।
```

(4) Comparision Operations :- Pointer comparison के लिए comparison operators (==, !=, <, >, <=, >=) का use किया जा सकता है। Pointer variables के addresses को compare करने के लिए इन operators का इस्तेमाल होता है।

Example

```
int num1 = 10, num2 = 20;
int *P1 = &num1 ;
int *P2 = &num2;
```

Page No. ______ Date

```
if (p1 == p2)
{
  Count<< "Pointers Point to the same-
  - address." << endl;
}
else
{
  count<< "Pointers Point to different-
  - address." << endl;
}
```

ये सभी operations Pointers को manipulate करने और उनका use करने में help करते हैं।

Union and Structure.

	Union	Structure
Definition	Unions में सिर्फ सबसे बड़े member के लिए memory allocate होती है। यह member को union के सारे members को बारी use किया जाता है।	Structure में, सभी members को एक साथ memory में allocate किया जाता है, जबकि
Memory usage:	~~Unions में भी Same है, but Union के members को access करने के लिए सिर्फ एक ही member~~ को use किया जा सकता है। जो कि union की सबसे बड़ी mem होती है।	~~Structure में, सारे members को अलग-अलग तरीके से access किया जा सकता है, जबकि,~~

Memory Usage:	Unions में सबसेबड़ी-member के लिए ही memory allocate होती है। Union में सिर्फ एक mem. का ही value hold होती है।	Structures में हर member के लिए अलग-अलग memory allocate होती है। but
Size	Unions का size सबसे बड़ी member के size के बराबर होता है।	Structures का size उसके members के sum के बराबर होता है, जबकि
Accessing members:	but union के members को access करने के लिए सिर्फ एक ही mem. को use किया जाता है, जो कि Union के सबसे बड़ी mem. होती है।	Structures में सारे members को अलग-अलग तरीके से access किया जा सकता है। Union में भी same है।
Memory Wastage	Unions में memory Optimization होती है क्योंकि सिर्फ एक ही member को एक समय पर access किया जा सकता है।	Structures में memory wastage हो सकती है, क्योंकि अगर सभी members को एक साथ use नहीं किया जाता है, जबकि

Flexibility:	Unions में सिर्फ एक ही members को एक ही समय पर access किया जाता है।	Structure में सभी member को एक साथ access किया जा सकता है। जबकि
Behaviour:	Unions में एक ही समय पर सिर्फ एक member का behavior किया जा सकता है।	Structures के हर mem. का behaviour अलग होता है और एक साथ उनको modify किया जा सकता है। जबकि
Initialisation:	Unions में सिर्फ किसी member को initialize किया जा सकता है एक समय पर।	Structures के members को initialize करने के लिए अलग-अलग values assign की जा सकती है। but

Example: `Person person = {"Alice", 25, 6000.0};` ← Structure को Initialize

Unions को Initialize → `Data data;` `data.intValue = 10;`

Accessing members का example

```
Person person;
Strcpy(person.name, "John");
person.age = 20;
person.salary = 8000.0;
```

Structure को access

`Data data;` `data.intValue = 20;` Union को access

Array of Pointers

Array of pointers एक ऐसे concept है, जिसमें एक array में pointers store होते हैं। हर pointer एक specific data type का memory address point करते हैं। यह pointer एक array के elements के addresses को point करते हैं।

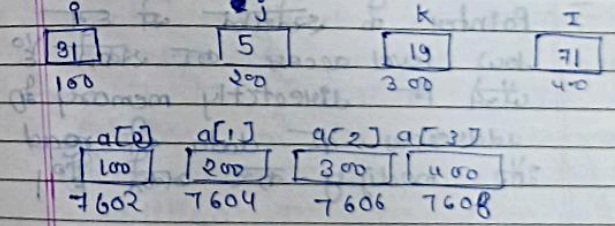
ये concept useful होता है जब हम एक array of strings या complex data structures जैसे कि linked lists, trees, etc. को represent करना होता है। हर एक pointer एक individual element या node के address को point करता है।

Example :

```
# include <stdio.h>
# include <conio.h>
void main()
{
    int *a[4];
    int i = 31, j = 5, k = 19, I = 71,
        m;
    a[0] = &i;
    a[1] = &j;
    a[2] = &k;
    a[3] = &I;
    for (m=0; m<=3; m++)
        printf("\n%d", *(a[m]));
}
```

Output

```
31
5
19
71
```



In this example :- *a[4]; एक array है, जो pointers store करता है। हर pointer एक integer variable का address point करता है, फिर हमने हर एक pointer के corresponding array of pointers के through value को access किया।

Feature of Pointer :-

① Memory Address को store करता है :- Pointer एक memory address को store करता है, जिसे हम किसी variable या object कहते हैं।

② Memory Management :- Pointer का इस्तेमाल memory management में होता है, ये हमें dynamic memory allocation और deallocation की flexibility देता है।

③ Low Level Access :- Pointer के इस्तेमाल से हम low level access कर सकते हैं जैसे कि directly memory के addresses पर data को read और modify कर सकते हैं।

④ Multiple Data Types को Point कर सकता है :- Pointer किसी भी data type के variable object का address store कर सकता है, चाहे वो Integer, Character, Float, या किसी और data type का हो।

⑤ Null Pointer :- Pointer का एक special value होता है जो Null होता है। Null pointer के किसी valid memory address की जगह assign किया जा सकता है जिसमें कोई value नहीं होती।

⑥ Function Pointers :- Pointer का use function addresses को store करने और उन्हें call करने के लिए होता है। इससे हम functions को arguments के रूप में भी pass कर सकते हैं।

⑦ Pointer Arithmatic :- Pointer arithmetic का इस्तेमाल Pointers के addresses को manipulate करने के लिए होता है। इससे Pointers को increment, decrement और arithmetic operations जैसे कि addition या Subtraction करने की capability मिलती है।

⑧ Array Access :- Pointer का इस्तेमाल array को access करने के लिए होता है। Pointer arithmetic का use करके हम array के element को traverse कर सकते हैं।

declaration of Pointer ☆☆☆

ये features pointers को powerful और versatile बनाते हैं, और इनसे programming में flexibility और efficiency आती है।

Introduction and features of structure & union.

Structure :- Structure एक C programming concept है, जो कि various data elements को एक साथ रखने के लिए उपयोग किया जाता है। यह एक composite data type होता है। जिसमें हम अलग-अलग data types के variables को एक साथ रख सकते हैं।

features :-

① अलग-अलग Data types का एक साथ use :- Structures में हम अलग-अलग Data types के variables को एक साथ रख सकते हैं, जैसे कि int, float, char आदि।

② Memory Efficiency :- structures का use memory को अधिक कार्यात्मक बनाने में help करता है, क्योंकि हम एक ही structure में कई variables को संग्रहित कर सकते हैं।

③ Accessing Members :- Structures के members को "." (dot) operator करके access किया जा सकता है।

④ User-defined data Type :- structures user-defined data type होती है, जिसे हम अपनी आवश्यकताओं के अनुसार define कर सकते हैं।

Union :-

Introduction :- Union भी एक programming concept है, जो कि कई variables को एक ही memory location में store करने के लिए उपयोग किया जाता है। यह भी एक composite data type होती है, but Union में सभी variables के साथ में रखने के बजाय, केवल एक ही variable को ही memory में store किया जाता है।

features:-

① Memory Efficiency :- union में सभी variable को एक ही memory में store करने के कारण, यह memory का अधिक वा विभिन्न बनाता है।

② Union size :- Union का size सबसे बड़े variable के आकार के बराबर होता है, जिसे सभी अन्य variables के लिए भी use किया जाता है।

③ Accessing Members :- Union के members को सामान्यतः किसी एक ही समय में एक ही value को access करने के लिए प्रयोग किया जाता है।

④ Data Overwrite :- Union में जब एक variable को update किया जाता है, तो अन्य variables की values over-write हो जाती है।

⑤ Common Use Cases :- Union को अक्सर memory optimization के लिए या किसी object के विभिन्न रूपों में प्रस्तुत करने के लिए इस्तेमाल किया जाता है।

Structure with syntax and example :-

Syntax of structure :-

```
Struct <structure_name>
{
    data_type member1;
    data_type member2;
}; // और अधिक member
```

यहाँ <structure_name> एक unique identifier है जिसे हम structure का नाम कहते हैं, और data_type है वह Data type जो इस member के लिए उपयोग किया जाता है।

Example

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
void main()
{
```

```
struct book    // definition of structure
{
  char name;
  float price;
  int pages;
};
Clrscr();      // use of structure
struct book b1, b2;
Scanf("%c %f %d", &b1.name,
      &b1.price, &b1.pages);
Scanf("%c %f %d", &b2.name,
      &b2.price, &b2.pages);
printf("%c %f %d", b1.name,
      b1.price, b1.pages);
printf("%c %f %d", b2.name,
      b2.price, b2.pages);
getch();
}
```

Output

```
a    56    67
b    68    57

a    56.000000    -67
b    68.000000    57
```

in this example, book एक structure है जिसमें name, price, page आदि members हैं। void main() function में, struct book structure का object b1, b2 बनाया गया है। और उसके members को assign और print किया गया है।

Union of structure:

Syntax of Union:-

```
union <union-name>
{
  data_type member1;
  data_type member2;
  // और अधिक सदस्य।
};
```

यहाँ <union-name> एक unique identifier है जिसे हम Union का Name कहते हैं और data_type है वह datatype हर member के लिए use होता है।

Example of Union...

```
#include<stdio.h>
#include<conio.h>
void main()
{
  clrscr();
  union test1
  {
    int x;
    int y;
  } test1;

  Union test2
  {
    int x;
    Char y;
  } test2;

  Union test3
  {
    int a[2];
    char y;
  } test3;

  Void main();
  {
    Pnt size1 = sizeof(test1);
    Int size2 = sizeof(test2);
    int size3 = sizeof(test3);
    printf("Sizeof test1 : %d\n", Size1);
    printf("Sizeof test2 : %d\n", size2);
    printf("Sizeof test3 : %d\n", Size3);
    getch();
  }
}
```

Output

```
Sizeof test1 : 2
Sizeof test2 : 2
Size of test3 : 20
```

In this example Data एक union है जिसमें intValue, charValue और arrayValue (int a[2]) value, member है। void main() function में data union का एक object test1, test2, test3 बनाया गया है, और उसके member को assign और print किया गया है।

कृपया ध्यान दें कि केवल एक member को ही access किया जा सकता है एक समय में union के।

25

27