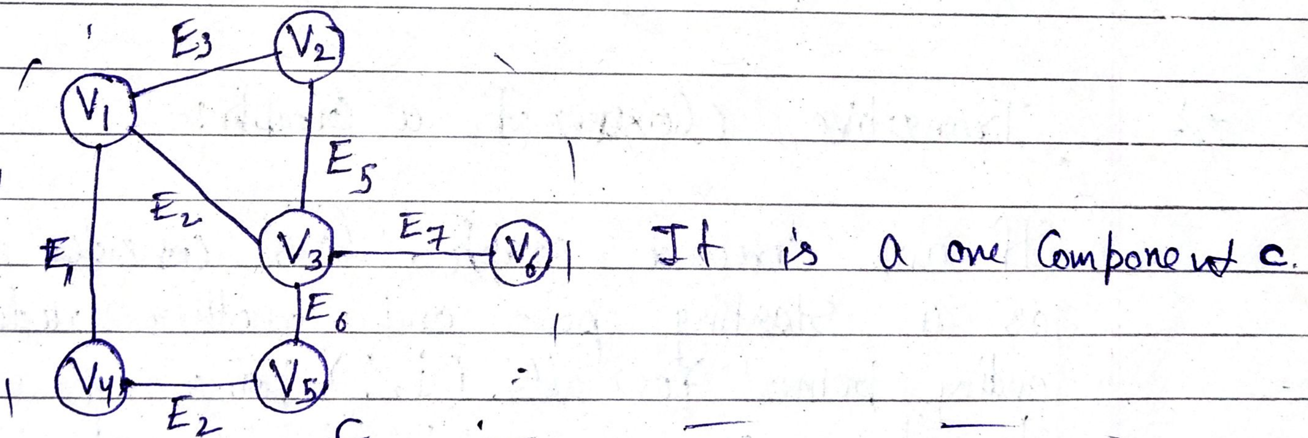
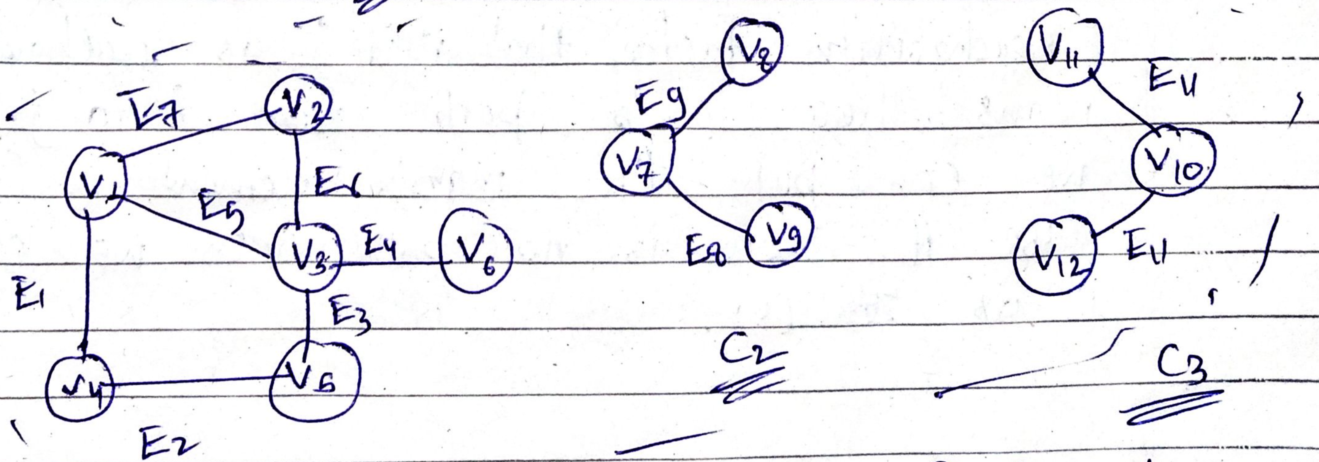
Date 01/12/22
Page

★ Connected Component :-

A Connected Component or simply component of an undirected graph is a sub-graph in which each pair of nodes is connected with each other via a path.

A set of nodes forms a connected component in an undirected graph if any node from the set of node can reach any other node by traversing edges.
The main point here is reachability.
In connected components all the nodes are always reachable from each other





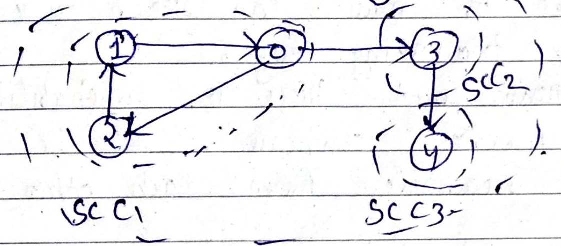
In which three Components C1, C2, C3

SHOT ON OPPO

See in directed graph

★ Strongly Connected Component:-

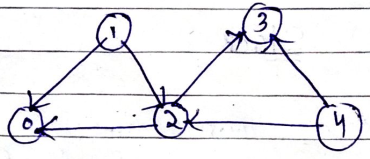
A directed graph is strongly connected if there is a path b/w all pair of vertices.
A strongly connected component of a directed graph is a maximal strongly connected sub-Graph.
For exp:-
There are three strongly connected component in the following graph.



★ Transitive Closure of a Graph:-

If any directed graph, lets consider a node 'i' as a starting point and another node j as ending point for all (i,j) pairs in a graph, transitive closure matrix is formed by the reachability factor, that is if j is reachable from i means there is a path from i to j then we can put the matrix element as '1'. OR else if there is no path then we can put it as zero (0).

For exp:-



Then, the reachability matrix of the given graph can be given by:-

	0	1	2	3	4
0	1	0	0	0	0
1	1	1	1	1	0
2	1	0	1	1	0
3	0	0	0	0	0
4	1	0	1	1	1

hence it is a transitive closure matrix

SHOT ON OPPO
By Ghanendra Kumar

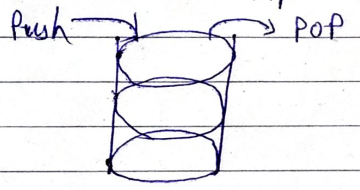
Date 05/12/22
Page

Unit-4th

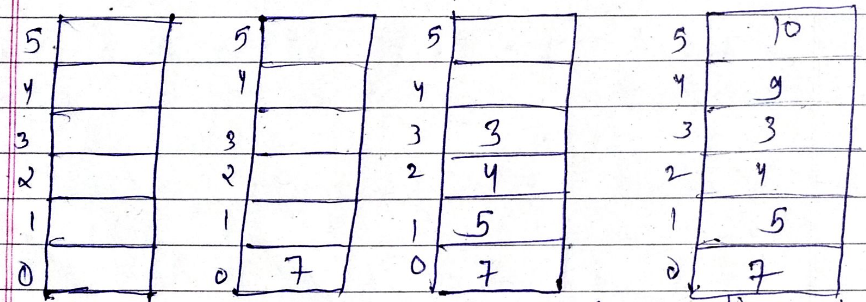
Stack

★ Stack :- A stack is a non-primitive linear data structure. It is an ordered list in which addition of new data item and deletion of already existing data item is done from only one end known as 'Top' of Stack (TOP).
Stack is also called last in first out (LIFO)

Exp ÷
A common model of a stack is plates in a marriage party or coin stacker. Fresh plates are pushed on to the top and popped from the top.



★ Working of Stack :-



(a) Top = -1
Initially stack is empty

(b) Insert element 7
Top = Top+1
Top = 0

(c) Insert element 5, 4 and 3
Top = 3

(d) Insert element 9 and 10
Top = 10

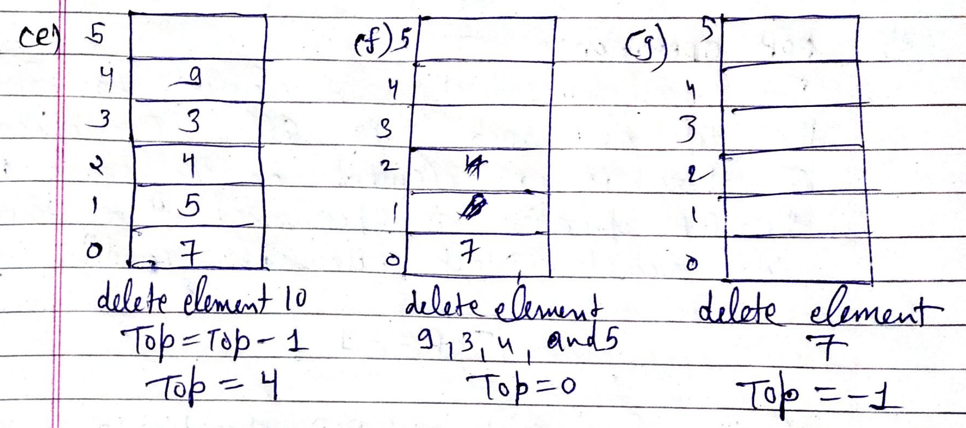
Stack overflow Condition :-

Top = MAX Size - 1

Stack underflow Condn

Top = -1

Date
Page



★ operations on Stack :-

(i) Initialize() (it is for create a stack)
(ii) IsEmpty()
(iii) IsFull()
(iv) Push()] Imp
(v) POP()

The basic operation that can be perform on stack are as follows

1) PUSH operation :-
The process of adding a new element into the top of stack is called as Push operation. In case the array (or stack) is full and no new element can be added, it is called stack overflow condn.

TOP = MAXSIZE - 1

SHOT ON OPPO
By Ghanendra Kumar

2) POP operation:-
The process of deleting an element ~~fo~~ from the top of stack is called POP operation. If there is no element on the stack and the POP operation is performed then this will result stack underflow Condⁿ.

TOP = -1

★ Algorithm for Push and POP operation:-

Push operation:-

1) If TOP = MAXSIZE - 1, then
   Print " STACK OVERFLOW and STOP"
2) READ DATA
3) TOP = TOP + 1
4) STACK [TOP] ← DATA
5) STOP

Pop operation:-

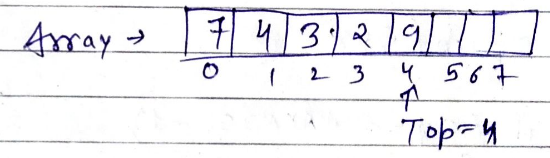
1) If TOP = -1 , then
   print " STACK UNDERFLOW and STOP"
2) DATA ← STACK [TOP]
3) TOP = TOP - 1
4) Print " DATA"
5) STOP

★ Representation of stack:-
A stack may be represented in two ways:-

1) Array Representation of stack
2) Linked list representation of stack.

1) Array Representation of stack:-

To impliment a stack using array we need a variable called TOP, which holds the index of the Top element of the stack. And an array ITEM[] which store the elements of stack.




```
#define MAXSIZE 20
int Top;
int ITEM [MAXSIZE];
```

'C' function for primitive operations on a stack using Array:-

① Void Initialize ()
```
{
    Top=-1;
}
```

② int IsEmpty ()
```
{
```

SHOT ON OPPO

Date
Page

```
    if (Top==-1)
      return(1);
    else
      return(0);
    }
```

③ int IsFull()

```
{
    if (Top == MAXSIZE-1)
        return(1);
    else
        return(0);
}
```

④ void push(int x)

```
{
    if (Top == MAXMISE-1)
    {
        printf(" stack is overflow");
        exit(0);
    }
    Top = Top+1;
    ITEM [POP] =x;
}
```

⑤ int pop()

```
{
    int x;
    if (Top==-1)
    {
        printf(" Stack is underflow");
        exit(0);
    }
```

Date
Page

```
    X=ITEM[Top];
    Top = Top-1;
    return(x);
}
```

② Linked-list Repⁿ of stack:

Singly linked list structure is sufficient to repⁿ any stack. Here, the info field is for the ITEM and next field as usual to point to the next ITEM. The figure show linked-list repⁿ of Stack.

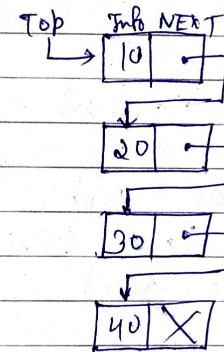


fig - Linked list Repⁿ of Stack.

'C' functions for primitive operation on a stack using linked list -

1) void initialize (stack *s)

```
{
    S= NULL;
}
```

SHOT ON OPPO
By Ghanendra Kumar

Date
Page

(ii)
```
int IsEmpty ( Stack *s)
{
    if (s = NULL)
      return(1);
    else
      return(0);
}
```

(iii)
```
int IsFULL ( Stack *s)
{
    if ( PTR == NULL)
      return(1)
    else
      return(0)
}
```

(iv)
```
Void push ( Stack *s int x)
{
    stack = Stack * temp ;
  temp = ( Stack*) malloc ( size of(node));
    if ( temp == NULL)
  {
    printf (" out of memory space");
    exit (0);
  }
    temp -> info = x ;
    temp -> next = s ;
     s = temp ;
}
```

(v)
```
int pop ( stack * s)
{
    stack * temp;
    int x ;
    temp = s ;
  if ( temp == NULL)
   {
     printf (" stack is empty");
   }
     x = temp -> info ;
     s = temp -> next
     free ( temp);
     return (x);
}
```

Date
Page

★ C program for implementation of stack :-

```
# include< stdio.h >
# process include< process.h >
int stack [100], choice, n, top, x, i;
void push();
void pop ();
Void display () ;
int main ()
{
    top = -1;
  printf (" Enter the size of Stack \n");
    scanf (" %d ", &n);
  printf ("\n\t Stack operation using array \n");
  printf (" \n \t Stack oper 1. push \n \t 2. pop \n
     \t 3. display \n \t 4. Exit \n");
  do
   {
```

SHOT ON OPPO
By Ghanendra Kumar

Date
Page

```
printf(" Enter the choice");
Scanf ("%d", & choice);
  Switch (choice):
{
Case 1:  push();
         break;
Case 2:  pop();
         break;
Case 3:  display();
         break;
Case 4:  ~~Exit~~ printf (" \n\t Exit point\n");
         break;
default : printf (" Invalid Choice");
}
} while ( Choice != 4)
}
void push()
{
 if ( top == n-1)
 {
    printf (" STACK is full ");
 }
else
{
 printf (" Enter a value to be inserted:");
 Scanf(" %d", & x);
 top ++;
 stack [top] = x
}
}
```

Date
Page

```
Void pop()
{
  if ( top == -1)
  {
    printf( " STACK is empty");
  }
else
{
   printf(" ~~STACK is~~ The popped element to %d",
    Stack [top]);
   top --;
}
}
Void display()
{
 if (top >= 0)
 {
  printf (" The stack element are \n");
   for (i = top; i >= 0 ; i--)
     printf (" %d \n", stack[i]);
 }
 else
 {
   printf(" stack is empty");
 }
}
```

SHOT ON OPPO
By Ghanendra Kumar

☆ Application of stack :-

There are a no. of application of stack, some of the application of Stack are as follow:

1. Expression Conversion :-
   (a) Infix expression to prefix
   (b) Infix to postfix
   (c) Postfix to Infix
   (d) postfix to prefix

2. Expression Evaluation :-

   3, 4, 6, 5, +, 10, -, /
   (A+B) + (c-d) * E/F

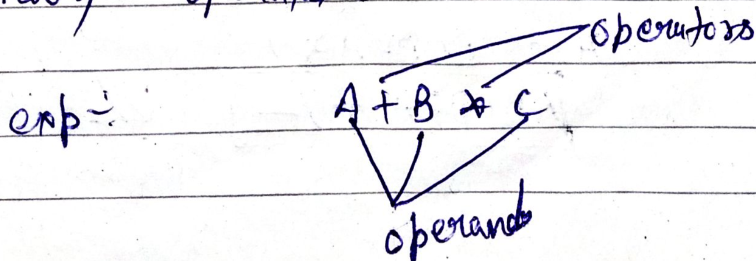
3. Parsing :-

4. Simulation of Recursion :-
5. Function call

☆ Evaluation of airthmatic expression :-

An airthmatic expression complete of operands and operators. operands are variable for Constant and operators are of various types like airthmatic operators, unary operators and binary operators.

exp:-



Date
Page

operator precedence :-

exponential operator	↑ or ^	Highest priority
Multiplication / Division	*, /	Next highest priority
Addition / Subtraction	+, -	least priority

→ There are three popular methods for representing of an airthmatic expression.

1. Infix → x ⊕ y
2. Prefix → ⊕ x y
3. Postfix → x y ⊕

① Conversion infix expression to postfix form -

(i) (A+B) * (C-D)   (LRN)

(AB+) * (CD-)

AB+CD-*

(ii) (A+B) * (C+D) /(E-F/G)

(AB+) * (CD+) / (EFG/-)

(AB+CD+ *) / (EFG/-)

AB+CD+ * EFG/-/

(iii) A-B / (C * D ^ F)

A-B / (C * DF^)

A-B / (CDF^*)

A- BCDF^*/

ABCDF^*/-

Date
Page

(iv) A-B / C * D^F

~~A-B/C*DF^~~

~~A-B/C DF^*~~

~~A-BCDF^*/~~

~~ABCDF^*/-~~

A-B/C * DF^

A- BC/ * DF^

A-BC/ DF^ *

A-BC DF^*/

ABCDF^*/-

☆ Conversion infix expression to prefix form -

(i) (A+B) * (C+D) / (E-F)

(+AB) * (+CD) / (-EF)

(*+AB+CD) / (-EF)

/ * +AB+CD-EF

(ii) (A+B) * C/D + E^F/G

(+AB) * C/D + E^F/G

(+AB) * C/D +(^EF)/ G

(*+ABC)/ D +(^EF)/G

(/*+ABCD) + (^EF) /G

(/*+ABCD) +(/^EFG)

+/*+ABCD +/^EFG

☆ Postfix to infix Conversion

Rule :-

1) Expression is scanned from left to right
2) Whenever an operator is found, operator is put in b/w the previous two operands.

SHOT ON OPPO
By Ghanendra Kumar

Date
Page

Q. Postfix expression

abc - + de - fg - h + / *
a(b-c) + de - fg - h + / *
[a + b-c) de - fg - h + / *
(a+b-c) (d-e) fg - h + / *
(a+b-c)(d-e) (f-g) h + / *
(a+b-c) (d-e) (f-g+h) / *
(a+b-c)(de / f-g+h) *
a+b-c * d-e / f-g+h

Q. ABCDF ∧ * / -
ABC(D∧F) * / -
AB[C * D∧F] / -
A [B / C *(D∧F)] -
A -(B / C *(D∧F))

Q. Convert postfix to ~~in~~ prefix :-

(a) ABCDF ∧ * / -
~~∧ ABCDEA* /~~ *   - ABCDF ∧ * /
~~* ∧ ABCDF~~ -   / - ABCDF ∧ *
- * ∧ ~~ABCDF~~   * / - ABCF∧
∧ * / - ABCDE

(b) abc - + de - fg - h + / *
(a - bc) + de - fg - h + / *
[a + - bc] de - fg - h + / *
(a + - bc) (- de) - fg - h + / *
[- a + - bc - de] fg - h + / *
[- a + - bc - de] (- fg) h + / *

Date
Page

[- a + - bc - de] [+ - fgh] / *
[/ - a + - bc - de + - fgh] *
~~*~~ * / - a + - bc - de + - fgh

SHOT ON OPPO
By Ghanendra Kumar

Date 14/12/22

★ Conversion of Expression from prefix to postfix:-

Rule:-

1) Expression is scanned from right to left.
2) whenever an operator is formed, it is put after the two operands

Q→ * + a − bc / − de + − fgh

* + a − bc / − de + [fg−] h
* + a − bc / − de [fg−h+]
* + a − bc / [de−] [fg−h+]
* + a − bc [(de−)(fg−h+/)]
* + a [bc−] [(de−)(fg−h+/)]
* [abc−+] [(de−)(fg−h+/)]
abc−+ (de−)(fg−h+/) *

★ Conversion of Expression from prefix to Infix:-

Rule:-

1) Expression is scanned from Right to left.
2) Whenever an operator is formed, it is put ~~before~~ b/w two operand.

Q→ * + a − bc / − de + − fgh

* + a − bc / − de + (f−g) h
* + a − bc / − de (f−g+h)
* + a − bc / (d−e) (f−g+h)
* + a − bc [(d−e)/(f−g+h)]
* + a (b−c) [(d−e)/(f−g+h)]
* (a+(b−c)) [(d−e)/(f−g+h)]
(a+(b−c)) * ((d−e)/(f−g+h))

SHOT ON OPPO
By Ghanendra Kumar

Date
Page

★ Conversion of Expression from postfix to prefix:-

Rule:-

1) Expression is scanned from left to Right.
2) whenever an operator is found, it is put before the previous two operands.

Q→ abc − + de − fg − h + / *

a [−bc] + de − fg − h + / *
[+a−bc] de − fg − h + / *
[+a−bc] [−de] fg − h + / *
[+a−bc] [−de] [−fg] h + / *
[+a−bc] [−de] [+−fgh] / *
[+a−bc] [/−de+−fgh] *

* + a − bc / − de + − fgh

Q. Convert the following postfix expression to prefix and infix.

AB + CD + * EFG / − /

Postfix to prefix.

→ [+AB] CD + * EFG / − /
[+AB] [+CD] * EFG / − /

Date
Page

[* +AB+CD] EFG/ - /

[* +AB+CD] E [/FG] - /

[* +AB+CD] [E/FG] /

/ * +AB+CD-E/FG

Postfix to infix.

AB+CD + * EFG/-/

[A+B] CD+ * EFG/-/

[A+B] [C+D] * EFG/-/

[A+B * C+D] EFG/-/

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

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

A+B* C+D/ E-F/G

Date 16/12/22
Page

Evaluation of postfix expression:-

Algorithm:-
This algorithm points the value of an airthmatic expression (P) written in postfix notation.

1) Add a right paranthesis ')' at the end of 'P'.
2) Scan "P" from left to right and repeat step ③ and ④ and each element of 'P' until the right paranthesis ")" is encountered.

3) If an operand is encountered put it on stack.
4) If an operator ⊗ is encounted then.

(a) Remove the top two elements from stack where 'A' is the top element and 'B' is the next to top element. [∵ ⊗ → operator]

(b) Evaluate Ⓑ ⊗ Ⓐ

(c) Place the result of the step 4(b) back on stack.
[End of If structure]
[End of Step ② loop]

5) Set value equal to the top element on stack.

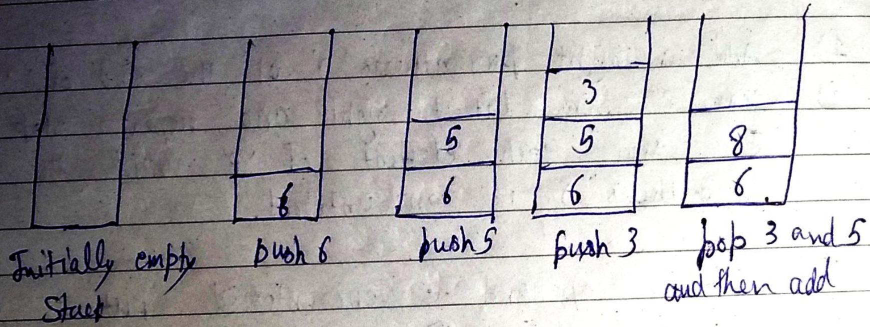
6) Exit.

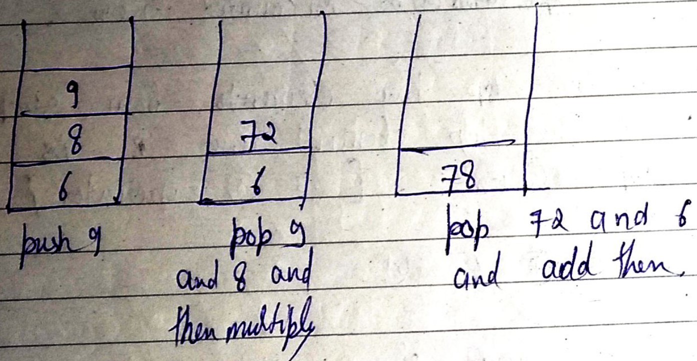
SHOT ON OPPO
By Ghanendra Kumar

Date
Page

Q. Evaluate the following postfix expression.
6, 5, 3, +, 9, *, +

Sol: First method :—
P: 6 5 3 + 9 * +)





Second Method : —

P: 6 5 3 + 9 * +)

Date
Page

Symbol scanned	Stack	Remark
6	6	Push 6
5	6, 5	Push 5
3	6, 5, 3	Push 3
+	6, 8	pop 3, 5 and add them
9	6, 8, 9	push 9
*	6, 72	pop 8 and 9 and multiply them.
+	78	pop 6, 72 and add then
)	78	postfix

Q. Evaluate the following expression :-
3, 16, 2, +, *, 12, 6, /, −

Sol: P: 3, 16, 2, +, *, 12, 6, /, −,)

Symbol scanned	stack	Remark
3	3	push 3
16	16	push 16
2	2	push 2
+	3, 18	pop, 16, 2 and add them
*	54	pop 3, 18 and multiply then
12	54, 12	push 12
6	54, 12, 6	push 6
/	54, 2	pop 12, 6 and divide then
−	52	pop 54, 2 and Subtract then
)	52	postfix

SHOT ON OPPO

Date ______ Page ______

Q→ 4, 6, 2, +, *, 12, 3, /, −

symbol scanned	Stack	Remark
4	4	push 4
6	4, 6	push 6
2	4, 6, 2	push 2
+	4, 8	pop 6, 2 and add then
*	32	pop 4, 8 and multiply them
12	32, 12	push 12
3	32, 12, 3	push 3
/	32, 4	pop 12, 3 and divide them
−	28	pop 32, 4 and sub them
)	28	postfix

NOTE:- Tabular form or stack both are same.

Convert infix expression to postfix expression using stack or tabular form.

→ Algorithm :-

Let 'x' is an airthmatic expression written in infix notation. This algorithm find the equivalent postfix expression y.

① Push left paranthesis "(" on stack and add a right paranthesis ")" to the end of

SHOT ON OPPO
By Ghanendra Kumar

Date ______ Page ______

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

③ If an operand is encountered, add it to 'Y'.

④ If a left paranthesis '(' is encountered, push it on to stack.

⑤ If an operator is encountered than :-

a) Repeatedly pop from stack and add to y each operator (on the top of stack) which has the same precedence as or higher precedence then operator.

b) Add operator to stack [End of If]

⑥ If a right paranthesis is encountered then :-

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

(b) Remove the left paranthesis [end of If]
[end of loop].

⑦ Exit.

Date
Page

Q→ Infix expression Convert into postfix expression
: A+(B*C-(D/E^F)*G)*H

Sol→ X : A+(B*C-(D/E^F)*G)*H)

Symbol Scanned	Stack	Postfix expression (Y)
	C	A
A	C	A
+	C+	A
C	C+C	A
B	C+C	AB
*	C+C*	AB
C	C+C*	ABC
-	C+C-	ABC*
C	C+C-C	ABC*
D	C+C-C	ABC*D
/	C+C-C/	ABC*D
E	C+C-C/	ABC*DE
^	C+C-C/^	ABC*DEF
F	C+C-C/^	ABC*DEF
)	C+C-	ABC*DEF^/
*	C+C-*	ABC*DEF^/
G	C+C-*	ABC*DEF^/G
)	C+	ABC*DEF^/G*-
*	C+*	ABC*DEF^/G*-
H	C+*	ABC*DEF^/G*-H
)	—	ABC*DEF^/G*-H*+

Date
Page

Q→ Convert it into infix to postfix form.
A+(B*C-(D/E)*G)*H.

Sol→ X: A+(B*C-(D/E)*G)*H)

Symbol scanned	Stack	Postfix expression (Y)
	C	
A	C	A
+	C+	A
C	C+C	A
B	C+C	AB
*	C+C*	AB
C	C+C*	ABC
-	C+C-	ABC*
C	C+C-C	ABC*
D	C+C-C	ABC*D
/	C+C-C/	ABC*D
E	C+C-C/	ABC*DE
)	C+C-	ABC*DE/
*	C+C-*	ABC*DE/
G	C+C-*	ABC*DE/G
)	C+	ABC*DE/G*-
*	C+*	ABC*DE/G*-
H	C+*	ABC*DE/G*-H
)	—	ABC*DE/G*-H*+

SHOT ON OPPO
By Ghanendra Kumar

Infix to prefix Conversion:-

Q→ Infix expression:- A+(B*C-(D/E)*G)*H

Sol→ Reverse of Infix Expression:-

H*(G*(E/D)-C*B)+A)

Symbol Scanned	Stack	Prefix expression
	C	
H	C	H
*	C*	H
C	C*C	H
G	C*C	HG
*	C*C*	HG
C	C*C*C	HG
E	C*C*C	HGE
/	C*C*C/	HGE
D	C*C*C/	HGED
)	C*C*	HGED/
-	C*C-	HGED/*
C	C*C-	HGED/*C
*	C*C-*	HGED/*C
B	C*C-*	HGED/*CB
)	C*	HGED/*CB*-
+	C+	HGED/*CB*-*
A	C+	HGED/*CB*-*A
)	-	HGED/*CB*-*A+

Now
Reverse the result:-

+A*-*BC*/DEGH

Q→ Convert the following infix expression into equivalent postfix expression using stack.

(a) A∧B*(C+D)+(E-F)+G/(H+W)

(b) A*(B+D)/E-F*(G+H/K)

Sol→ (a) A∧B*(C+D)+(E-F)+G/(H+W))

Symbol Scanned	Stack	Postfix expression
	C	
A	C	A
∧	C∧	A
B	C∧	AB
*	C∧*	AB
C	C∧*C	AB
C	C∧*C	ABC
+	C∧*C+	ABC
D	C∧*C+	ABCD
)	C∧*	ABCD+
+	C∧*+	ABCD+
C	C∧*+C	ABCD+
E	C∧*+C	ABCD+E
-	C∧*+C-	ABCD+E
F	C∧*+C-	ABCD+EF
)	C∧*+	ABCD+EF-
+	C∧*++	ABCD+EF-
G	C∧*++	ABCD+EF-G
/	C∧*++/	ABCD+EF-G

C	C∧*++/C	ABCD+EF−G
H	C∧*++/C	ABCD+EF−GH
+	C∧*++/C+	ABCD+EF−GH
W	C∧*++/C+	ABCD+EF−GHW
)	C∧*++/	ABCD+EF−GHW+
)	—	ABCD+EF−GHW+/++*∧

A∧B * C C+D) + (E−F) + G/(H+W))

Symbol scanned	Stack	Post Expression.
	C	
A	C	A
∧	C∧	A
B	C∧	AB
*	C*	AB∧
C	C*C	AB∧
C	C*C	AB∧C
+	C*C+	AB∧C
D	C*C+	AB∧CD
)	C*	AB∧CD+
+	C*+	AB∧CD+*
C	C+C	AB∧CD+*
E	C+C	AB∧CD+*E
−	C+C−	AB∧CD+*E
F	C+C−	AB∧CD+*EF
)	C+	AB∧CD+*EF
+	C++	AB∧CD+*EF
G	C++	AB∧CD+*EFG
/	C++/	AB∧CD+*EFG
C	C++/C	AB∧CD+*EFG

H	C++/C	AB∧CD+*EFGH
+	C++/C+	AB∧CD+*EFGH
W	C++/C+	AB∧CD+*EFGHW
)	C++/	AB∧CD+*EFGHW+
)	—	AB∧CD+*EFGHW+/++

A → AB∧CD+*EFGHW+/++

Q (b) A * (B+D)/E−F * (G+H/K))

Symbol Scanned	Stack	Post expression
	C	
A	C	A
*	C*	A
C	C*C	A
B	C*C	AB
+	C*C+	AB
D	C*C+	ABD
)	C*	ABD+
/	C*/	ABD+
E	C*/	ABD+E
−	C−	ABD+E/*
F	C−	ABD+E/*F
*	C−*	ABD+E/*F
C	C−*C	ABD+E/*F
G	C−*C	ABD+E/*FG
+	C−*C+	ABD+E/*FG
H	C−*C+	ABD+E/*FGH
/	C−*C+/	ABD+E/*FGH

SHOT ON OPPO
By Ghanendra Kumar

Date
Page

K	C-*C+/	ABD+E/*FGHK
)	C-*	ABD+E/*FGHK/+
)	-	ABD+E/*FGHK/+*

A → ABD+E/*FGHK/+*- A

Date
Page

*	(- * (+ /	ABD + E / * FGHK
)	(- *	ABD + E / * FGHK / +
)	—	ABD + E / * FGHK / + *

A → ABD + E / * FGHK / + * — A

☆ **Recursion:-** Recursion is a fundamental concept in mathematics, when a function is define in terms of itself, it is called recursion or recursive function.

Factorial without recursion	with Recursion
int n;	= 5 * factorial (4)
int f=1;	= 5 * 4 * factorial (3)
for (i=n; i<=1; i--)	= 5 * 4 * 3 * factorial (2)
fact = fact * i;	= 5 * 4 * 3 * 2 * factorial (1)
	= 5 * 4 * 3 * 2 * 1
	= 120
	if ((n=0) or (n=1))
	return (1);
	else
	return (n * factorial (n-1))

Date
Page

☆ **Recursive function for factorial:-**

```
int factorial (int n)
{
    if ((n==1)!! (n==0))
        return(1);
    else
        return (n * factorial (n-1) );
}
```

Finding n^{th} term of fibonacci series:-
Fibonacci series - 1 1 2 3 5 8 13 21

Recursive defination:-

fib(n) = { return n if ((n=1)or(n=0)) else return (fib(n-1) + fib(n-2))

0 1	n=3
n=2	fib(2) + fib(1)
fib(1) + fib(0)	f(1) + fib(1) + fib(1)
1+0=1	1+0+1=2

☆ **Types of recursion:-**
Basically there are two types of recursion:-

1. Direct recursion:- Wheather function call itself from with in itself its call direct recursion.

2. Indirect recursion:- Wheather two function call on another mutually it is called indirect recursion

exp:-

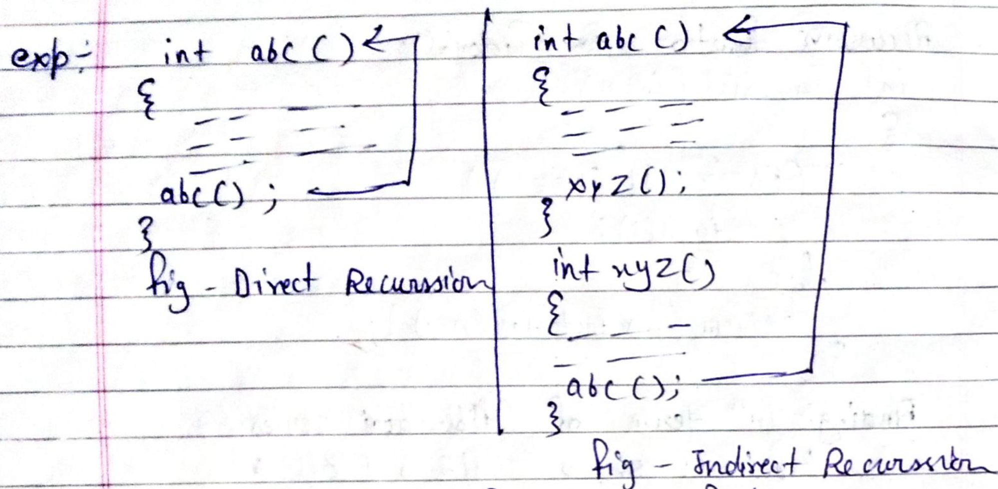


fig - Direct Recursion

fig - Indirect Recursion

→ Recursion may be further classified as :-

1. Linear Recursion
2. Binary "
3. Multiple "

☆ Advantages of Recursion:-

→ We can solve complex problems by using recursion.

☆ Disadvantages of Recursion:-

→ It consumes more storages space because the recursive calls along with automatic variable are stored in stack.

→ It is not more efficient in terms of speed and time.

☆ Ackermann function :- A (m, n)

1) If m=0, then A(m, n) = n+1
2) If m≠0 but n=0, then A(m, n) = A(m-1, 1)
3) If m≠0 and n≠0, then A(m, n) = A(m-1, A(m, n-1))

Q. Find the Value of A(1,2) using Ackermann function

Sol:

A(1,2) = ?

Now, A(m, n) = A(m-1, A(m, n-1))

A(1,2) = A(0, A(1,1)) — ①

A(1,1) = A(0, A(1,0)) — ②

A(1,0) = A(0,1) — ③

put eqⁿ ④ in eqⁿ ①

A(0,1) = 1+1 = 2 — ④

Now,

A(1,2) = A(0,2)
= 2+1 = 3 Ans

A(1,2) = A(0,3)
= 3+1 = 4 Ans

M. Imp

☆ Tower of Hanoi problem :-

Tower of Hanoi problem can be defined as :

A Tower of 'n' disk is stacked in decreasing order of peg 'A'. These disk are to be

transfered to peg 'C' with the help of peg 'B'.

Following rule must be followed by transfering disk from 1 peg to another peg.

a) only one disk can be transfered at a time
b) At no time a larger disk can not be placed on a smaller disk.

★ n-disk problem :-

Move n-disk from peg 'A' to peg 'C' using peg 'B'.

1) move 'n-1' disk from peg 'A' to peg 'B'
2) move the only one disk from peg 'A' to peg 'C'.
3) Move the 'n-1' disk from peg 'B' to peg 'C'

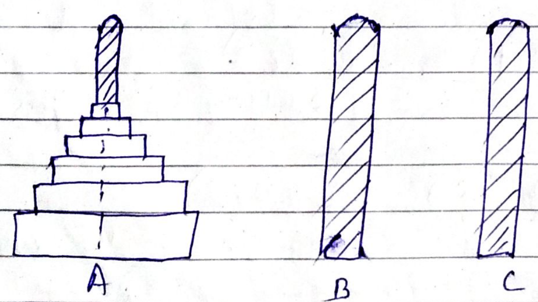


fig :- Disks are stacked on peg 'A' in decreasing order of size.

Given 'n' disks :-

$$\text{No. of moves} = 2^n - 1$$

n = no. of disk

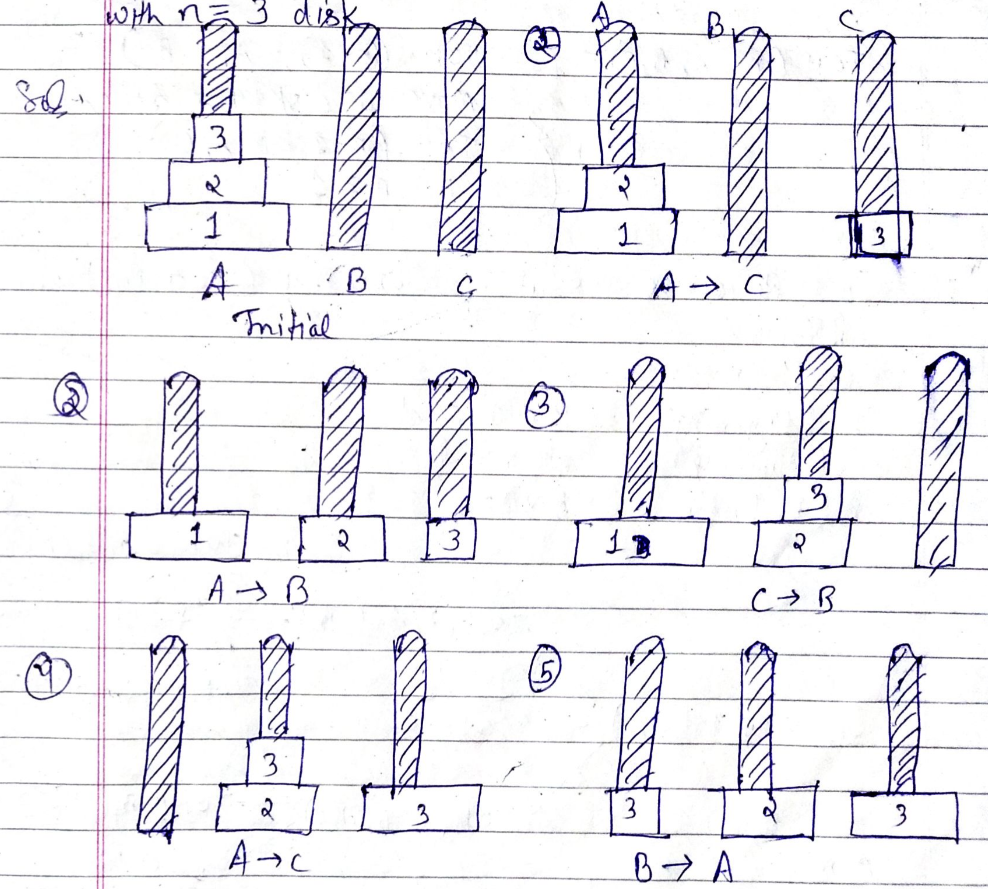
Q → Calculate total no. of moves for Tower of hanoi having 10 disk.

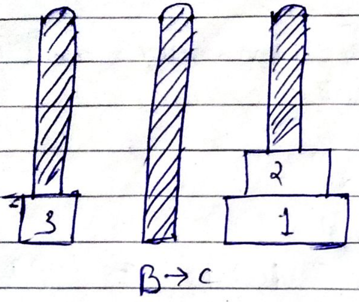
Sol →

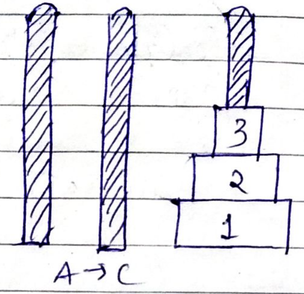
$$\text{no. of moves} = 2^n - 1$$
$$= 2^{10} - 1$$
$$= 1024 - 1 = 1023$$

Q → Solution to the Tower of hanoi problem with n = 3 disk.

Sol →





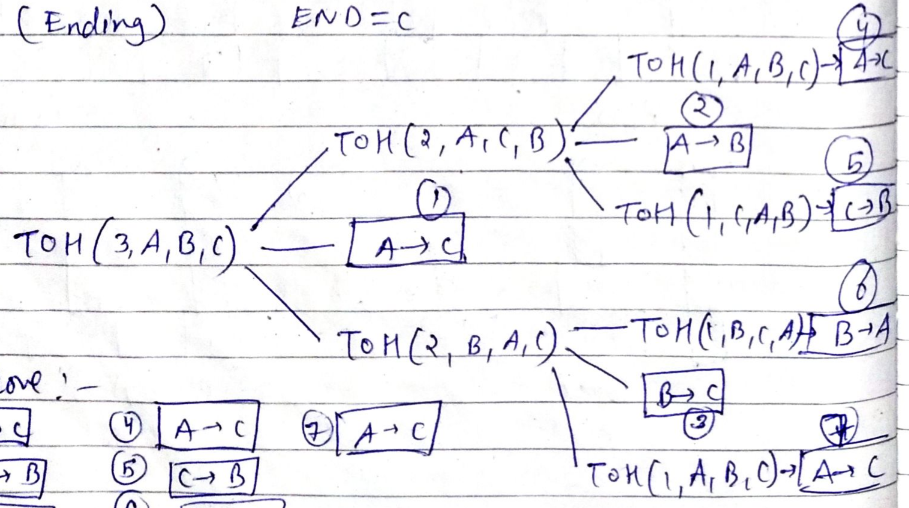


★ Recursive Solⁿ of Tower of Hanoi problem for n = 3.

TOH (n, A, B, C) = { TOH ((n−1), A, C, B)
Move a disk from A to C
TOH (n−1, B, A, C)
if n ≥ 1 }

Solⁿ :- The Recursive solution for TOH (3, A, B, C) as follows.

Here, n = 3
(Begining) BEG = A
(Auxilary) AUX = B
(Ending) END = C



★ C-program for TOH problem using recursion :-

```
Void TOH ( int, char, char, char );
     Void main()
  {
     int n;
   Char A='A', B='B', C='C';
   printf (" Enter the no. of disks \n");
    Scanf ("%d", &n);
   printf (" Tower of Hanoi problem with %d
        disk ", n);
   printf (" sequence is \n");
        TOH (n, A, B, C);
  }
Void TOH (int n, char A, char B, char C)
  {
     if (n! = 0)
   {
      ToH (n-1, A, C, B);
   printf (" move disk %d from %c to %c";
          n, A, C);
      TOH (n-1, B, A, C);
  }
```

Iteration	Recursion
① It is the process of executing a statement or set of statement repeated by until some specific condition is satisfied.	① Recursion is a technique of defining anything in terms of itself.
② Iteration involves four char cut steps. Initialization, condition { updation and executation }	② There must be an exclusive if statement inside the recursive function specifying stoping condition.
③ Any recursive problem can be -	③ Not all problem have recursive solution
④ Iterative counter part of a problem is more efficient in terms of memory utilization and execution speed.	④ Recursion is generally a worst option to go for simple problems or problems not recursive in Nature

Tail recursion:— It is a situation where a single recursive call is consider by a function and it is a final statement to be executed. It can be replaced by Iteration.
Tail recursion can be eliminated by changing the recursive call to a goto statement preceded by a set of assignment per function call.

Using tail recursion:—

```
Void TOH ( int n, char x, Char y, Char z)
{
    if (n>0)
    {
        TOH(n-1, x, z, y)
        printf ("%c → %c\n", x, z);
        TOH( n-1, y, x, z); /* Tail Recursion */
    }
}
```

Without using Tail recursion:—

```
Void TOH ( int n, char x, Char y, Char z)
{
    Char Temp;
    label :0   if n
    {
        TOH (n-1, x, z, y);
        printf("%c→ %c \n", x, z);

        temp = z;
        z = y;
        x = temp;
        n = n-1;

        goto label 0;
}
```

Date
Page

★ Principle of Recursion :-
For implementing and designing the good recursive program we must make certain assumption as follows:-

① Base Case:- Base case is the terminating condⁿ for the problem by designing any recursive algorithm.
We must choose a proper terminating condⁿ for problem.

② IF Condition:- If condition is the recursive algorithm define the terminating condition.
Every time new recursive call is made a new memory space alocated to each automatic variable used by recursive routine.
Each time recursive call is there, the duplicated values of the local variables of a recursive call are pushed on to the stack. within its respective call and there values are available to the respective function call when it is poped off from the stack.

③ Recursive case:- Else part of the recursive definition calls the function recursively.

exp:-
```
int factorial (int n)
{
    if (n==1)||(n==0))   → if Condⁿ
        return(1);
    else
        return (n * factorial(n-1);   → Recursive case
}
```

Date
Page

★ Removal of Recursion:- Any recursive function can be converted into non-recursive function so use of stack.

① A recursive call is similar to a call to another function.

② Any call to a function requires that the function has storage ~~to~~ area where it can store its local variables and actual parameters.

③ Return address must be saved before a call is made to a function.

④ In case of a recursive call, the value of local variables parameters and return address must be saved on the stack.

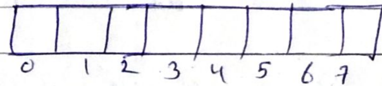
⑤ While returning from a nested call the previous other call must be recalled with resetting all the local variables and operation must resume from where it was suspended.

★ **Queue :-** A Queue is a linear list of element in which insertion can take place only at one end called rear and deletion can be take place only at other end called front.

Queue is also called first - in- first- out (FIFO) type of list. since the first element in a queue will be the first element out of the queue.

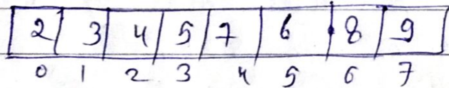
Working

① Initially



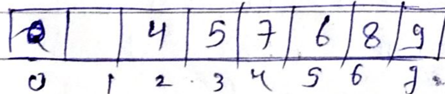
Front = -1 }
Rear = -1 } underflow Condⁿ

②



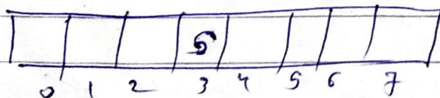
Front = 0 }
Rear = 7 } At insertion only rear pointer increase

③



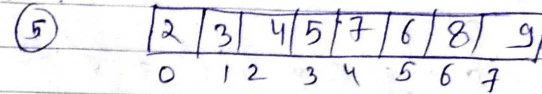
Rear = 7 }
Front = 2 } At deletion only front pointer increase

④



Front = 3 }
Rear = 3 } ⇒ front = Rear } when there is a single element in queue

⑤



Front = 0 }
Rear = 7 } Rear = Max_Size - 1
overflow Condⁿ

★ **operation on Queue :-**

① Create / Initialize( )
② Insert( )
③ Delete ( )
④ Is Empty( )
⑤ Is full ( )

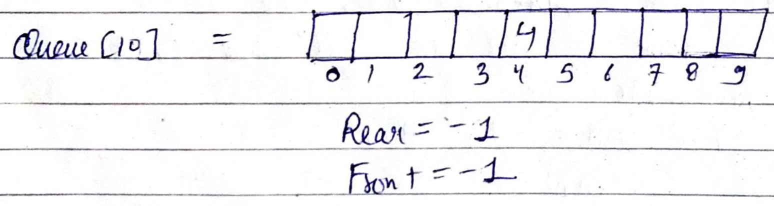
Algorithm for insertion in Queue :-

1) Initialization
   set front = -1
   set rear = -1
2) If (Rear = MAXSIZE - 1), then,
   print " Queue is full" and Exit
3) Read Item
4) If (front = -1), then
   Front = 0
   Rear = 0
Else,
   Rear = Rear + 1
   [End if]

5) Set Queue [Rear] = Item
6) Exit

Algorithm for deletion in Queue:-

1) If (front == -1), then
   print " Queue is empty" and Exit
2) Set Item = Queue [front];
3) If (front = Rear)
   set front = -1
   set Rear = -1
   Else
   front = front + 1
   [End if]
4) Print " No. deleted is", Item
5) Exit



★ C functions for different Queue operation :-

```
#define MAX 100
int, front, Rear;
int Queue [MAX0];
```

①
```
Void Create()
{
    front = -1;
    Rear = -1;
}
```

②
```
int IsEmpty ()
{
    if (front == -1)
        return(1);
    else
        return (0);
}
```

③
```
int IsFull()
{
    if ( Rear = MAXSIZE - 1)
        return (1);
    else
        return (0);
}
```

④
```
Void Insert ()
{
    if (Rear == MAXSIZE - 1)
    {
        printf(" Queue is full");
```

```
    exit(0);
  }
  if (front == -1)     → Check underflow
  {
    front = 0;
    Rear = 0;
  }
  else {
    Rear = Rear + 1;
    Queue [Rear] = Item;
  }
}
```

⑤

```
int Delete ()
{
    int x;
    if (front == -1)
    {
      printf("Queue is empty");
      exit(0);
    }
    if (front == Rear)   → when single element in Queue
    {
      front = -1;
      Rear = -1;
    }
    else {
      x = Queue [front]
      front = front + 1;
    } return (x);
```

☆ Application of Queue :-

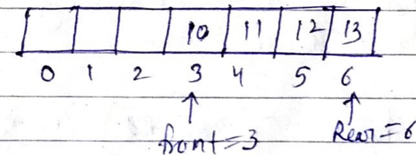
1) Round Robin technique for processor scheduling is implementing using queue.

2) All types of Customer services like railway ticket reservation, centre softwares are design using queue to store customers information.

3) Printer server routines are design using Queue.

☆ Limitations of Queue :-

If the last position of the queue is occupied then it is not possible to insert any more elements in the Queue even those some positions are vacant towards the front position of the Queue.



☆ Circular Queue :-

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

We can say that a circular Queue is one in which the first element comes just after the last element. It can be viewed as a

mesh or loop of wire in which two ends of the wire are connected together.

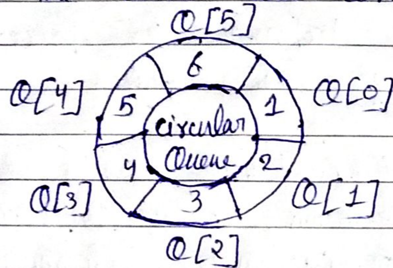


fig :- Circular Queue

overflow condition :-

| front = (Rear+1) % Max_size |

underflow:-

| front = -1 |

☆ Algorithm for Inserting an element in Circular Queue:

1) If (front = (Rear +1) % MAX_SIZE), then
   print " Queue is overflow " and EXIT

2) Else
   Read [ITEM]
3) If (front = -1) or If ((front = -1) or (Rear = -1))
   set front = 0
   set Rear = 0
Else
   Rear = (Rear +1) % Max_size
4) Set Queue [Rear] = Item
   [End If]
5) Exit

☆ Algorithm for Deleting an element in Circular Queue:-

1) If (front = -1) , then
   print " Queue is underflow or Empty " and Exit.

2) Else
   ITEM = Queue [front]

3) If (front = Rear)
   Set front = -1
   set Rear = -1
   Else
   Front = (Front + 1) % Max_size
   [End if]
4) print " No. deleted is ", Item.
5) Exit

Q. Consider the following circular Queue capable of accomodating maximum 6 elements.

front = 2 , Rear = 4,

Queue :- _, L, M, N, _, _

↑ Front = 2 ↑ Rear = 4

Date
Page

(ii) Add O

Queue :- __, L, M, N, O, __

Front = 2 , Rear = 5

(vi) Add P :-

Queue :-

__ L, M, N, O, P

Front = 2 Rear = 6

(iii) Delete two letters :-

Queue :-

__, __, __, N, O, P

Front = 4 . Rear = 6

(iv) Add Q, R, S

Queue :- Q, R, S, N, O, P

Rear = 3 front = 4

(v) Delete one letter

Queue :- Q, R, S, __, O, P

Rear = 3 , front = 5

Front = (Rear + 1) % MAXSIZE overflow

Circular queue front = -1 underflow

Date
Page

★ Dequeue (or Double Ended Queue) :-

A Dequeue is a linear list in which elements can be added or removed at either end but not in the middle, the term Dequeue stands for double Ended Queue.

There are two variations of Dequeue. These two variations are due to the restrictions put to perform either the insertion or deletion only at one end.

① Input restricted dequeue
② output restricted dequeue

① Input Restricted Dequeue :-

An input Restricted Dequeue is a dequeue which allows insertion at only one end of the list but allows deletion at both ends of the list.

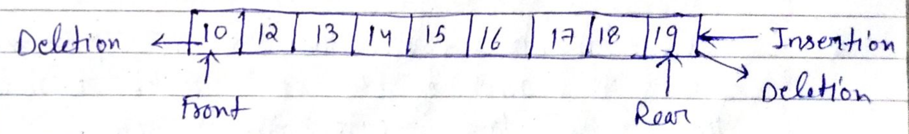


Fig :- Input Restricted Dequeue.

Date
Page

② Output Restricted Dequeue :-

An output Restricted Dequeue is a Dequeue which allows deletion at only one end of the list but allows insertion at both end of the list.

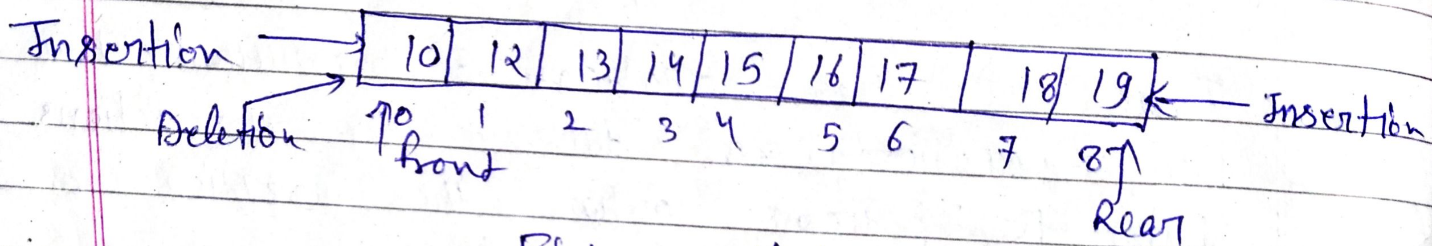


Fig :- Output Restricted Dequeue.

☆ Priority Queue :-

It is the collection of elements such that each element has been assign a priority and the order inwhich elements are deleted and processed comes from the following rules.

Rules :-

1. An elements of higher priority is processed before any element of lower priority.

2. Two elements with the same priority are processed according to the order inwhich they were added into the queue.