DATE : / /
PAGE NO. :

LINEAR SEARCHING

```
{ int a[10], i, n, pos, val, r=0;
printf (" Enter the elements : ");
for (i=0; i<n; i++)
{ scanf ("%d", &a[i]);
} printf (" Enter the element to find ; ");
scanf ("%d", &val);
for (i=0; i<n; i++)
  { if (a[i] == val)
      { r=1;
        pos = i;
        break;
      }
  }
  if (r==0)
   { (" Element not found ");
  else
    { (" Element is at %d position ", pos+1);
   }
  getch ();
}
```

BINARY SEARCHING

```
{ int a[10], high, low, flag=0, n, i, val, mid, pos;
  printf (" Enter the elements : "); for (i=0; i<n; i++)
 { scanf (" %d ", & a[i]);
  }
  printf (" Enter element to be found : ");
  scanf (" %d", &val);
  high = n-1;
  low = 0;
  while (low <= high)
```

DATE : / /
PAGE NO. :

```
{  mid = (low + high) / 2;
   if ( a[mid] == val)
     { pos = mid;
       flag = 1;
       break;
     }
   else if (val > a[mid)
     { low = mid + 1;
     }
   else
     { high = mid - 1;
     }
  }
  if (flag == 1)
    { printf (" value is at %d position", pos+1);
    }
  else
    { printf ("Element not found ");
    }
  getch();
}
```

06
09
2019

DATE : / /
PAGE NO. :

SORTING

i) Simple sorting
ii) Bubble sorting
iii) Selection sorting
iv) Insertion sorting
v) Quick sorting
vi) Merge sorting

* BUBBLE SORTING METHOD :

```
for (i=0; i<n-1; i++)
{ for (j=0; j<n-1-j; j++)
  { if (a[j] > a[j+1])
    { temp = a[j];
      a[j] = a[j+1];
      a[j+1] = temp;
    }
  }
}
```

n = 5

①	10	2	8	1	3
	0	1	2	3	4

② 2 8 1 3 10
2 8
1 8
3 8 10

③ 2 1 3 8 10

④ 1 2 3 8 10

- Outer loop is used for PROCESSES. → n-1 times
  (i=0 ; i < 5-1 ; i++)
  i < 4
- Inner loop is used for comparisons. for (j=0; j < 5-1-i ; j++)

Process 1	Process 2	Process 3	Process 4
i = 0	i = 1	i = 2	i = 3
4 times loop	3 times	2 times	1 times

0	10	2	2	1
1	2	8	1	2
2	8	1	3	3
3	1	3	8	8
4	3	10	10	10

SWAPPING TAKES PLACE

DATE : / /
PAGE NO. :

* SELECTION SORTING METHOD :

```
for (i=0 ; i<n-1 ; i++)
{ min = a[i];
  pos = i ;
  for ( j = i+1 ; j<= n-1 ; j++)
  { if (min > a[j])
    { min = a[j] ;
      pos = j ;
    }
  }
  temp = a[pos] ;
  a[pos] = a[i];
  a[i] = temp ;
}
```

* INSERTION SORTING METHOD :

Algorithm :

Insertion_sort (a, n)

Step 1 : declare i, j, key ;
Step 2 : Repeat Step 3 to 6 for i = 1 to n-1
Step 3 : set key = a[i] ; j = i-1 ;
Step 4 : Repeat step 5 while (j >= 0 && a[j] > key)
Step 5 : set a[j+1] = a[j] ; j = j-1 ;
Step 6 : set a[j+1] = key ;
Step 7 : write a[]
Step 8 : exit

Code :

```
for (i=1 ; i<n ; i++)
{ key = a[i];
  j = i-1;
```

DATE : / /

PAGE NO. :

```
while (j>=0 && a[j] > key)
    { a[j+1] = a[j];
      j = j-1;
    }
    a[j+1] = key;
}
```

10
9
19

DATE : / /
PAGE NO. :

ASYMPTOTIC NOTATIONS

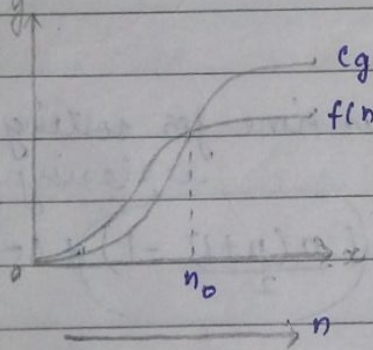
* Asymptotic Notations are used to describe complexity (TIME) of a Algorithm.
* These gives us a method for classifying functions according to their rate of growth.
* We need to develop a way to talk about rate of growth of functions so that we can compare algorithms.
* We are usually interested in the growth of the running time of a Algorithm not in the exact running time.
* This is also referred as Asymtotic running time.
* Asymptotic means tends to Infinity.

- Notations :

I) BIG OH (O) :

- Upper Bound
- Indirectly used for Worst Case
- Graph :

f(n) - Equation generated



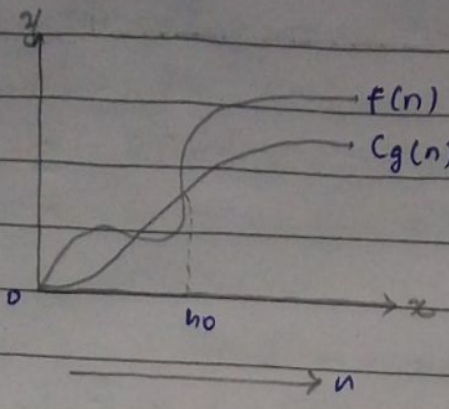
- $f(n) \leq cg(n)$
- for all $n \geq n_0$
  $c > 0$

then $f(n) = O(g(n))$

II) OMEGA (Ω)

- Lower Bound
- Graph :

DATE : / /
PAGE NO. :

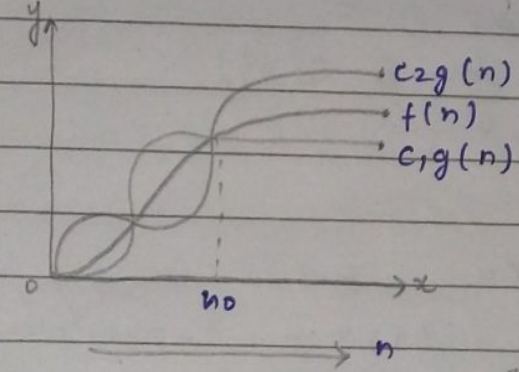


- $f(n) \geq cg(n)$
- for all $n \geq n_0$ and $c > 0$

then $f(n) = \Omega(g(n))$

iii) THETA (θ) :

- Tight Bound
- Graph :



- $c_1 g(n) \leq f(n) \leq c_2 g(n)$
- for all $n \geq n_0$ and $c > 0$

then $f(n) = \theta(g(n))$

θ: $f(n) = 3n - 2 = O(n)$ $\quad cg(n) = cn$

- $f(n) = 3n - 2$
- $gn = n$
- $f(n) \leq cg(n)$
- $3n - 2 \leq cn$

$cg(n) = cn^2$

Ω: $f(n) = 5n^2 + 3n - 2 = O(n^2)$

- $5n^2 + 3n - 2 \leq cn^2$
- $c = 6$

* True for $c \geq 6$

$n_0 \geq 2$

DATE : / /
PAGE NO. :

Q: $f(n) = 4n + 3$

$c \geq 5$

$f(n) \leq cg(n)$ $\quad g(n) = n$ $\quad n_0 \geq 3$

$\downarrow$

$f(n) \leq c(n)$

$n=1$	$4n+3 \leq cn$	$= 4\times1+3 \leq 5\times1$	$=$ false
$n=2$	$4n+3 \leq cn$	$= 4\times2+3 \leq 5\times2$	$=$ false
$n=3$	$4n+3 \leq cn$	$= 4\times3+3 \leq 5\times3$	$=$ TRUE
$n=4$	$4n+3 \leq cn$	$= 4\times4+3 \leq 5\times4$	$=$ TRUE

Q: PROVE THAT : for

$f(n) = 5n^2 + 4n + 3$ is $O(n^2)$ $\quad g(n) = n^2$

C . S

$n_0 = 1$ $\quad f(n) \leq cn$

$\Rightarrow 5(1)^2 + 4(1) + 3 \leq 5(1)^2$

$\Rightarrow 5 + 4 + 3 \leq 5(1)$ FALSE

$5(2)^2 + 4(2) + 3 \leq 5(2)^2$

$\Rightarrow 20 + 8 + 3 \leq 20$ FALSE

$c = 7$, $n_0 = 1$ ANSWER

① $5(1)^2 + 4(1) + 3 \leq 7(1)^2$

$\Rightarrow 5 + 4 + 3 \leq 7$

② $5(2)^2 + 4(1) + 3 \leq 7\times(2)^2$

$20 + 4 + 3 \leq 28$ TRUE

③ $5(3)^2 + 4(1) + 3 \leq 7\times(3)^2$

$45 + 4 + 3 \leq 63$

$52 \leq 63$ TRUE

Q: $f(n) = 4n+3$ is $\Omega(n)$

Q: $f(n) = 3n^2 - 5n + 2$ is $\Theta(n)$

Q: $f(n) = \frac{1}{2}n^2 - 3n$ for $\Theta(n^2)$:

$\Theta : c_1 g(n) \leq f(n) \leq c_2 g(n)$

$\Rightarrow \frac{c_1 n^2 \leq \frac{1}{2}n^2 - 3n \leq c_2 n^2}{n^2} \Rightarrow c_1 \leq \frac{1}{2} - \frac{3}{n} \leq c_2$

assume $c_2 = \frac{1}{2}$ $\Rightarrow$ $c_1 \leq \frac{1}{2} - \frac{3}{n} \leq \frac{1}{2}$

$n_0 = 7$ $\Rightarrow$ $c_1 \leq \frac{1}{2} - \frac{3}{7} \leq \frac{1}{2}$ $\Rightarrow$ $c_1 \leq \frac{7-6}{14} \leq \frac{1}{2}$

Q: $f(n) = 5n^2 + 3n - 5 = \Theta(n^2)$

$\Theta \Rightarrow c_1 g(n) \leq f(n) \leq c_2 g(n)$

$\Rightarrow c_1 n^2 \leq 5n^2 + 3n - 5 \leq c_2 n^2$ $\quad n_0 = 1$

$\Rightarrow c_1 (1)^2 \leq 5(1)^2 + 3(1) + 5 \leq c_2 (1)^2$

$\Rightarrow c_1 \leq 5 + 3 + 5 \leq c_2$

$\Rightarrow c_1 \leq 13 \leq c_2$

$\Rightarrow c_1 (2)^2 \leq 5(2)^2 + 3(2) + 5 \leq c_2 (2)^2$ $\quad n_0 = 2$

$\Rightarrow c_1 (4) \leq 20 + 6 + 5 \leq c_2 (4)$

$\Rightarrow c_1 (4) \leq 31 \leq c_2 (4)$ $\quad$ assume: $c_1 = 3$, $c_2 = 6$

DATE : / /
PAGE NO. :

$c_1 n^2 \leq 2n^2 + \frac{5}{n} - \frac{3}{n} \leq c_2 n^2$

$n_0 = 1$
$c_2 = 5$
$c_1 = 1$

$c_1 (1)^2 \leq 2(1)^2 + 5 - 3 \leq c_2 (1)^2$

$c_1 \leq 2 + 5 - 3 \leq c_2$

$c_1 \leq 2 + 5 - 3 \leq 5$

$1 \leq 2 + 5 - 3 \leq 5$

$1 \leq 4 \leq 5$

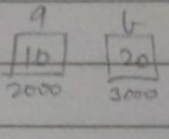
16
09
19

POINTER

* Pointer is a special variable that is used to store the address of another variable of similar datatype.

* **POINTER TO POINTER :**
Special variable i.e. used to store address of another pointer of similar datatype.

* Example :




```
void main ()
{ int *ptr1, a = 10, b = 20;
  float *ptr2, c = 5.5;
  ptr1 = &a;
  printf ("Value of a = %d ", a); // direct
  printf ("Value of a = %d", *ptr1);      *(a);
  printf ("Value of a = %d", *(&a));      *(2000);
  printf ("Address of a = %d", &a);       Random (2000)
  printf ("Address of a = %d", ptr1);     (2000)
  ptr1 = &b;
  printf ("Value of b = %d", b);
  printf ("Value of b = %d", *ptr1);
  ptr2 = &c;
```

DATE : / /
PAGE NO. :

```
printf (" value of c = %f", c);
printf (" value of c : %f", *ptr2);
getch ();
}
```

int **ptp; ** declaration (pointer to pointer)

pp = &ptr1;

printf (" value of pp

ARRAY USING POINTER:

```
void main ()
{ int *p , a[5] , i ;
  p=a;
  printf (" Enter values (5) in Array : ");
  for (i=0 ; i<5 ; i++)
  { scanf (" %d", p+i);
  }
  printf (" values in the Array are : ");
  for (i=0 ; i<5 ; i++)
  { printf (" %d ", *(p+i));
  }
  getch ();
}
```

base address of array.

→ address of 1st element

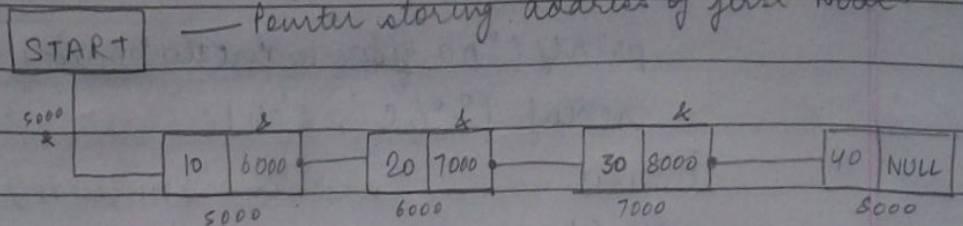
POINTER can be ++ , -- ptr1++;

	a[0]	a[1]	a[2]	a[3]	a[4]
a[5]	10	20	30	40	50
	2000	2002	2004	2006	2008

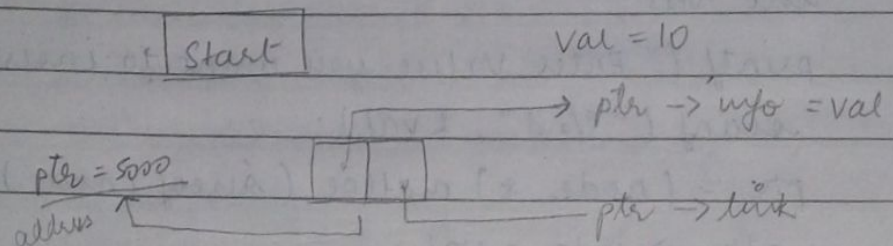
Address

DATE : / /
PAGE NO. :

```
    temp = temp -> link;
  }
}
```



* **Malloc**, when gives memory, the default value is garbage value and with calloc the value is 0.



* CASES : OPERTIONS

i) Insertion at the beginning
ii) Insertion at the end
iii) To delete the first node
iv) To delete the last node
v) searching in linked list
vi) To delete a particular node (by value)
vii) To insert a new node after a particular value.
viii) To reverse the linked list
ix) To sort the linked list.

DATE: / /
PAGE NO.:

(1) INSERTION at the BEGINNING :

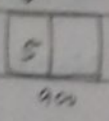
returns address to ptr

```
void insert_at_begin (start)
{ ptr = (node *) malloc (sizeof (node));
```

ptr → (node*)

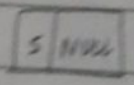
```
printf ("Enter value you want to insert in node");
scanf ("%d", &val);
ptr -> info = val;
```

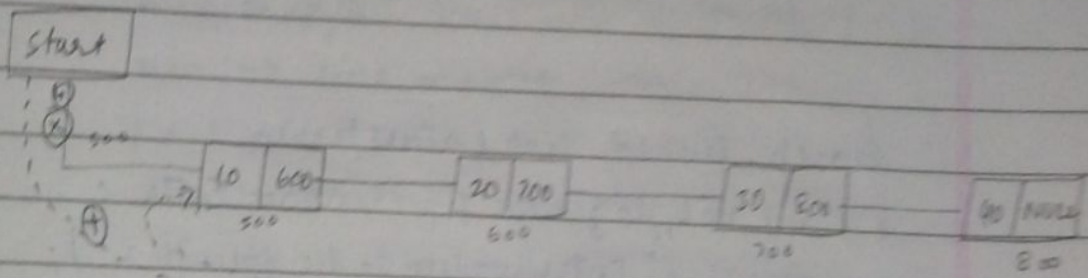
→ value goes at info part



①

```
ptr -> link = NULL;
```

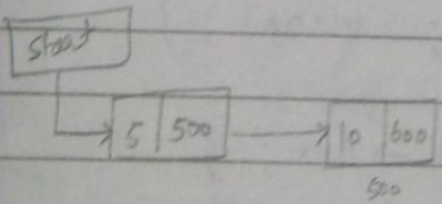
→ 



②

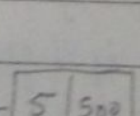
```
ptr -> link = start;
```

// 5's address goes to start and now it points 500 addressed value



③

```
start = ptr;
ptr = NULL;
```

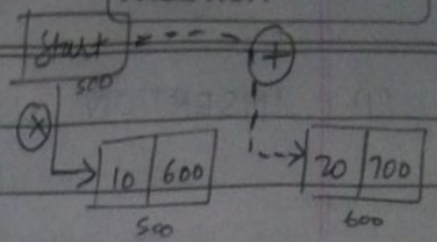


Or ④

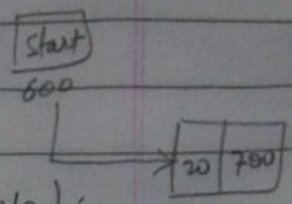
```
free (ptr);
}
```

DATE : / /
PAGE NO. :

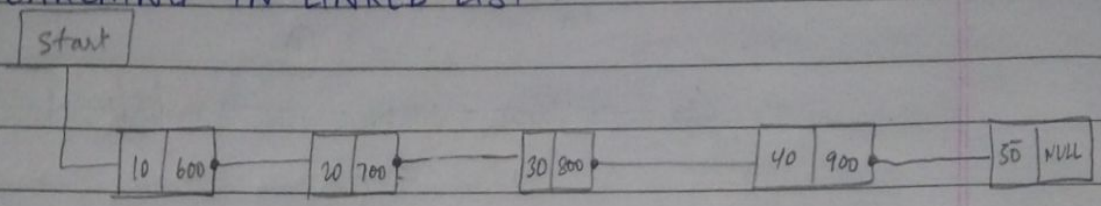
(iii) TO DELETE First NODE :




```
void delete_first_node (start)
{   node * temp;
    temp = start;
    start = start -> link;
    temp -> link = NULL;
printf ("Deleted Value is %d", temp -> info);
    free (temp);
}
```



(v) SEARCHING IN LINKED LIST :




```
search_linked_list (start)
{ int val, flag = 0 ;     node *ptr ;
  printf ("Enter value to be found : ");
  scanf ("%d", &val);
  ptr = start ;                    ptr pointing (10).
  while (ptr != NULL)
    { if (ptr -> info == val)
        { flag = 1 ; break ; }
      else
        { ptr = ptr -> link ; }      // ptr = 600 address changes.
    }
  if (flag == 0)
    { printf ("value is not Present ");
  else
    { printf ("value's address is %d", ptr);   or ptr -> info
                                       prints 600.
}
```

when it reaches last NODE ⇒ ptr = NULL so (NULL != NULL)

start 500 | ptr 500 | =N

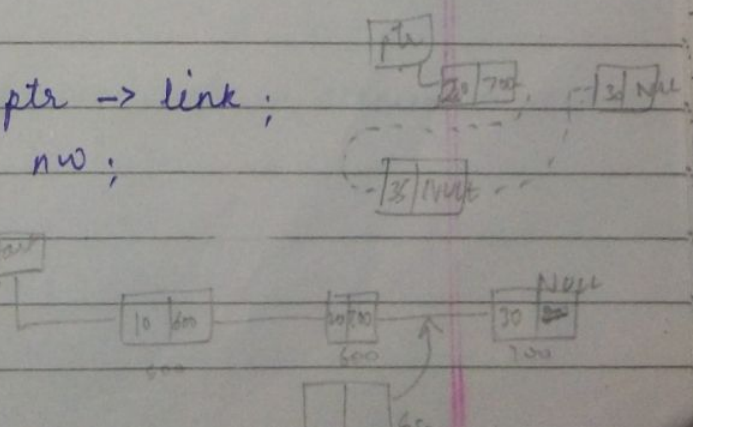
DATE : / /
PAGE NO. :

(VI) TO DELETE A PARTICULAR ELEMENT : Next Page →

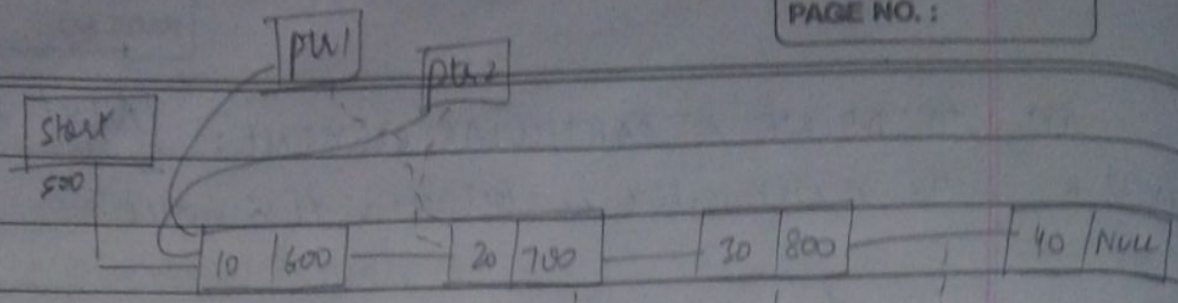
(VII) * To insert value in between two nodes :

```
*
    nw -> link = ptr -> link ;
    ptr -> link = nw ;          // MAIN LOGIC
```

```
void insert_after_value (start)
{ int val1, val2, flag = 0;
  printf ("Enter value you want to be inserted : ");
  scanf ("%d", &val1);
  printf (" Enter value after you want to insert : ");
  scanf ("%d", &val2);
  nw = (node *) malloc (sizeof (node));
  nw -> info = val1;
  nw -> link = NULL;
  ptr = start;
  while (ptr != NULL)
  { if (ptr -> info == val2)
      { flag = 1;
        break; }
      ptr = ptr -> link          // to go forward.
  }
  if (flag == 0)
  { printf ("Connot Insert "); exit ();
  }
  else
  { nw -> link = ptr -> link;
    ptr -> link = nw;
  }
}
```



DATE : / /
PAGE NO. :



TO DELETE A PARTICULAR NODE:

```
void delete_particular_node
{ ptr1 = NULL;
  ptr2 = start;
  while (ptr2 != NULL)
    { if (ptr2 -> info == val)
        { flag = 1; break;
        }
    }
  { ptr1 = ptr2;
    ptr2 = ptr2 -> link;
  }
  ptr1 -> link = ptr2 -> link;
    free (ptr1);
    free (ptr2);
}
```

DATE : / /
PAGE NO. :

** stack by using Array : * ARRAY.

[9]

[3]		TOP = -1
[2]	30	TOP = 0
[1]	20	TOP = 1
[0]	10	TOP = 2
		TOP = 4

[3]

▲ TOP = = n-1
then the stack is full
OVERFLOW = full
UNDERFLOW = empty

TOP = 2

- TOP increases in Push()
- Top decreases in Pop()

* Algorithm of PUSH() function :

n = no. of elements in stack

PUSH (stack , val , n)

step 1 : if TOP = = n-1
then write " overflow and exit "

step 2 : else
set TOP = TOP + 1 ;
stack [TOP] = val ;

step 3 : exit () ;

* Algorithm of POP() function :

POP (stack , n, val)

step 1 : if TOP = = -1
then write "underflow and exit" ;

step 2 : else
set val = stack [TOP] ;
TOP = TOP - 1 ;

step 3 : write " delete value is val " ;

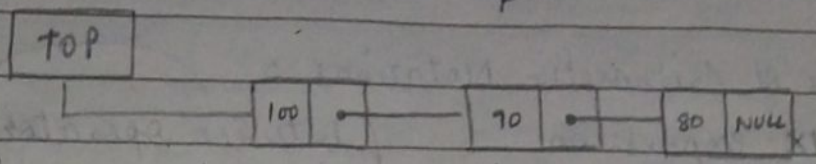
step 4 : exit () ;

* displaying STACK :

for (i= top ; i >=0 ; i--)
{ printf ("%d", stack[i]) ; }

DATE : / /
PAGE NO. :

** stack using linked list : * LINKED LIST



using insertion at the beginning of linked list
" deletion " " " " " " "

* Algorithm for PUSH() function:

PUSH (TOP, val)

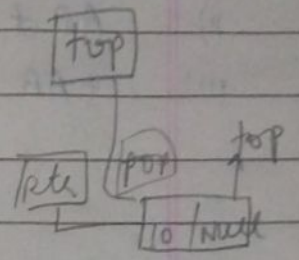
step 1 : if TOP ≠ ≠/ ptr = new node;

step 2 : set ptr -> info = val;
ptr -> link = NULL;

step 3 : ptr -> link = TOP ;
TOP = ptr ;

step 4 : ptr = NULL ;

step 5 : exit



* Algorithm for POP() function:

POP (TOP, val)

step 1 : if TOP = = NULL
then write "underflow and exit " ;

step 2 : else
ptr = TOP ; TOP = TOP -> link ;

step 3 : write " delete value is ptr -> info " ;

step 2+2 : exit () ;

① put parenthesis to right and left
② scan the expression from left to right
③ if left parenthesis come, push it in stack
④ any operand comes, send to postfix part.

DATE: / /
PAGE NO.:

* INFIX to POSTFIX conversion by using STACK.

⇒ (A+B * (C/D)^E) * example

SYMBOL scanned	STACK	POSTFIX exp^n
C	|C|	
A	|C|	A
+	|C|+|	A
B	|C|+|	AB
*	|C|+|*	AB
(	|C|+|*|C|	AB
C	|C|+|*|(|	ABC
/	|C|+|*|(|/|	ABC
D	|C|+|*|(|/|	ABCD
)	|C|+|*|	ABCD/
^	|C|+|*|^|	ABCD/
E	|C|+|*|^|	ABCD/E
)	left removed.	ABCD/E^*+

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

) when right comes, pop out values till left parenthesis comes and left parenthesis also removed.

if operator exists → check priority compare → pop less priority and push to postfix exp^n

no existence → push

do this when less priority operator is present

Algorithm to convert

Step Rule ① • Put left parenthesis into the left side of INFIX exp^n and right parenthesis to the right side of exp^n.

Step Rule ② • scan infix exp^n from left to right and repeat step ③ to ⑥ for each element of infix exp^n.

Step Rule ③ if an operand is scanned, push it to postfix exp^n.

④ if left parenthesis is encountered, push it into Stack.

DATE : / /
PAGE NO. :

(5) (a) if an operator is encountered then
  (a) repeatly pop from stack and and add it to postfix expression.
  (b) if the operator which is on the top of the stack have same priority or higher priority
  (b) add (push) this operator into stack
(6) if a right parenthesis is scanned then repeatly pop from the stack and add into postfix exp^n until a left parenthesis is encountered, also remove the left parenthesis from stack.

$A+(B+C-(D/E\hat{}F)*G)*H$
$A+(B+C-(D/EF\hat{})*G)*H$
$A+(B+C-(DEF\hat{}/)*G)*H$
$\Rightarrow A+(B+C-DEF\hat{}/*G)*H$
$\Rightarrow A+(B+C-DEF\hat{}/G*)*H$
$A+(BC+-DEF\hat{}/G*)*H$
$\Rightarrow A+(BC+DEF\hat{}/G*-)*H$
$\Rightarrow A+BC+DEF\hat{}/G*-H*$
$\Rightarrow ABC+DEF\hat{}/G*-H*+$