DATE : / /
PAGE NO. :

no info. of previous value)

* CONSTRUCTING A NODE :

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

Q. WAP to create a linked list ?

Ans:

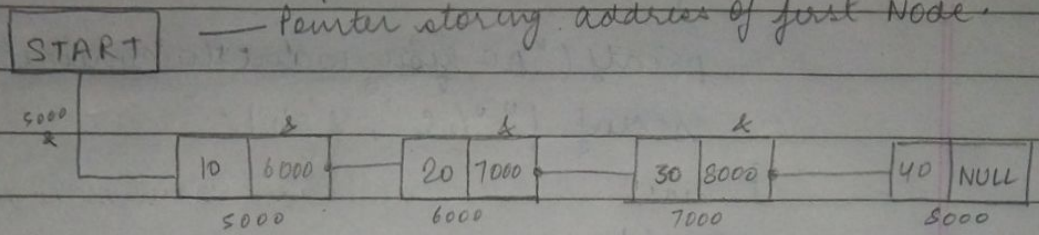
```
# include < stdio . h >          # include < stdlib . h >   for Exit ()
# include < conio . h >
# include < malloc . h >   → HEADER FILES CLASS for Malloc ()
void insert () ;  void display () ;
struct node
{ int info ;
  struct node * link ;
};                                  } STRUCTURE for LINKED LIST
typedef struct node Node   ( instead of struct node Node can be used further

void main ()
{ int choice ; char ch = 'Y' ;
  node * start = NULL , * ptr , * nw ;   → nw ✓ new is not accepted by the compiler
  while ( ch == 'y' || ch == 'Y' )
  { printf (" Enter 1 for creating linked list \n ");
    printf (" Enter 2 for displaying linked list \n");
    printf (" Enter 3 for Exit :");
    printf ("\n Enter your choice :");
    scanf ("%d", & choice );
    switch ( choice )
    { case 1 : insert (); break ;
```

DATE : / /
PAGE NO. :

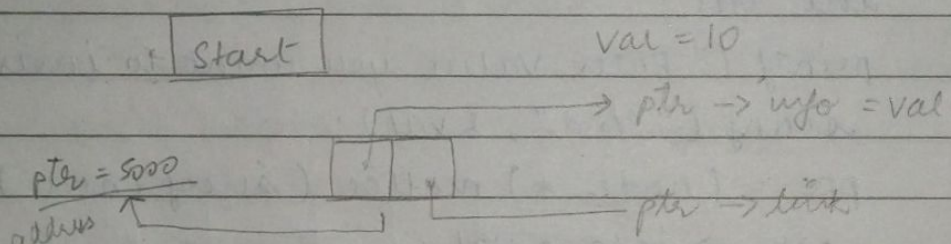
```
            case 2 : display (); break;
            case 3 :  exit ();
            default : printf (" INVALID CHOICE ");
          }
          printf (" DO you want to continue. Press y/Y : ");
          scanf (" %c ", &ch);
        }
        getch ();
    }
    void insert ()                    start = first Node's Address
    {  node *start = NULL , *ptr , *nw ;
       int val ;
       printf (" Enter Value you want to insert in Node :");
       scanf (" %d ", &val);
       ptr = (node *) malloc (sizeof (node)) ;
       ptr -> info = val ;                          DATATYPE
       ptr -> link = NULL;
       if ( start == NULL)
        { start = ptr ; }
       else
        { nw = start ; }
       while (nw -> link != NULL)
        { nw = nw -> link ; }
        nw -> link = ptr ;
        }
    }
    void display ()
    {  node * temp ;
       temp = start ;
       while (temp != NULL)
        { printf ("%d \t ", temp -> info) ;
```

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 :

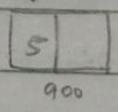
returns node's address to ptr ←

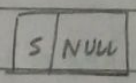
```
void insert_at_begin (start)
{ ptr = (node *) malloc (sizeof (node));
```

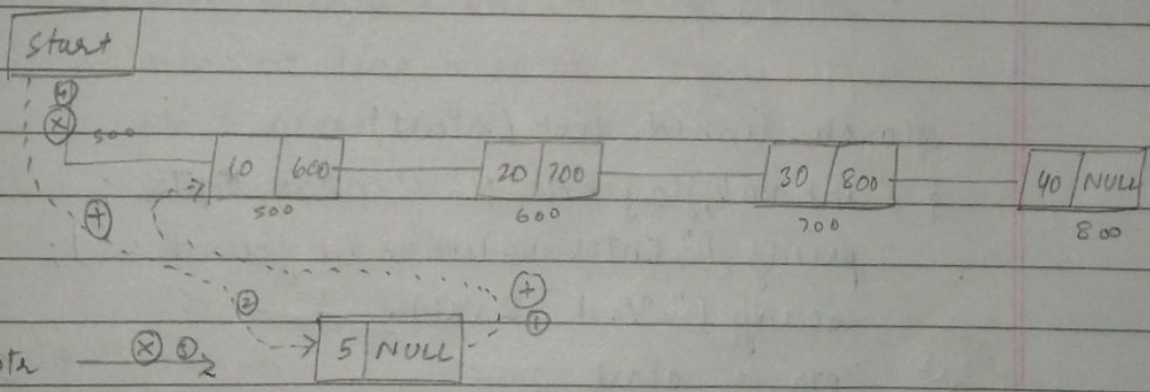
ptr (node*)

```
printf ("Enter value you want to insert in Node");
scanf ("%d", &val); ⑤
ptr -> info = val;
```

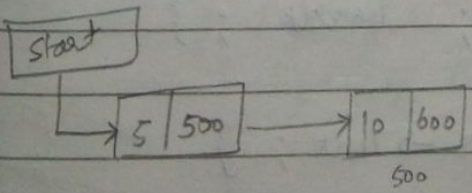
→ value goes at info part



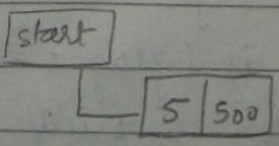
① `ptr -> link = NULL;` → 



② `ptr -> link = start;` // 5's address goes to start and now it points 500 addressed value.



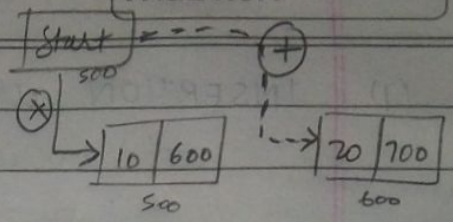
③ `start = ptr;`



④ `ptr = NULL;`
or
`free (Ptr);`

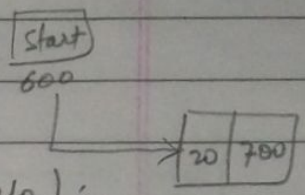
}

DATE : / /
PAGE NO. :

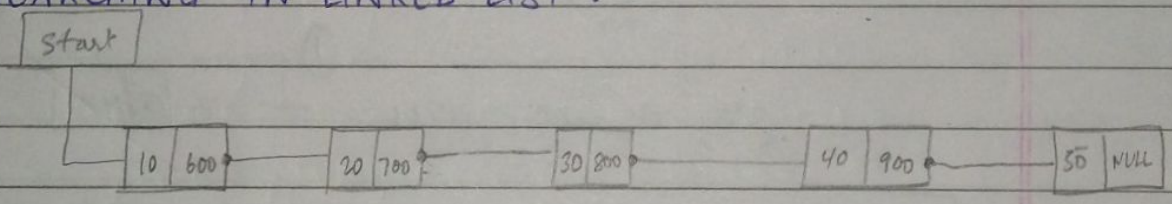


(iii) TO DELETE first NODE :

```
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 != NULL

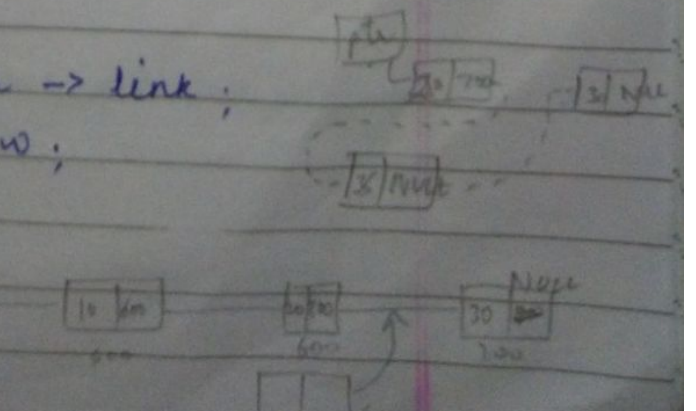
DATE : / /
PAGE NO. :

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

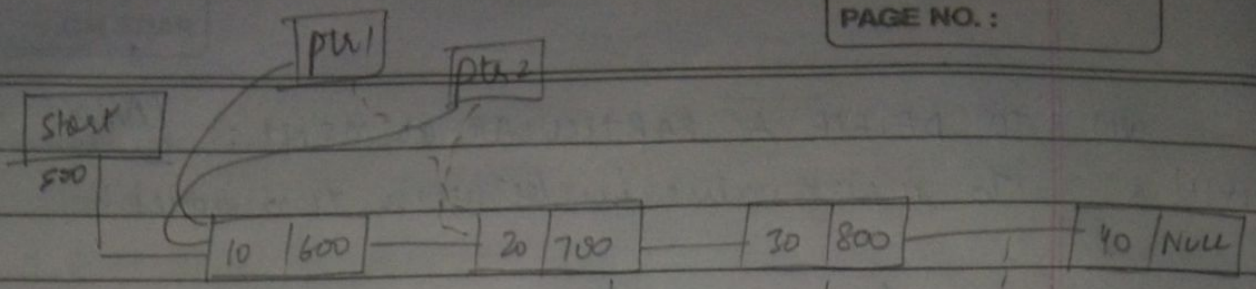
(VII) * To insert value in between two nodes :

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

```
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);
  new = (node *) malloc (sizeof (node));
  new -> info = val1;
  new -> 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
   { new -> link = ptr -> link;
     ptr -> link = new;
   }
}
```



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;
        }
    }
else { ptr1 = ptr2;
       ptr2 = ptr2 -> link;
  }
  ptr1 -> link = ptr2 -> link;
    free (ptr1);
    free (ptr2);
}
```

12-10-19

DATE : / /
PAGE NO. :

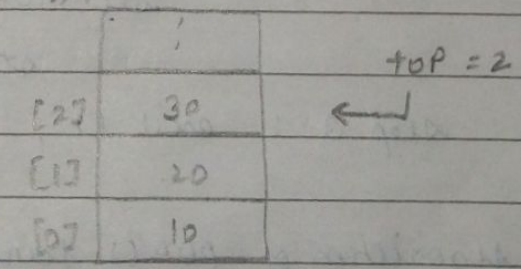
STACK

* stack is a linear data structure in which insertion and deletion can take place on a single end
* This end will be known as top. TOP.
* stack works on LIFO principle (last in first out)

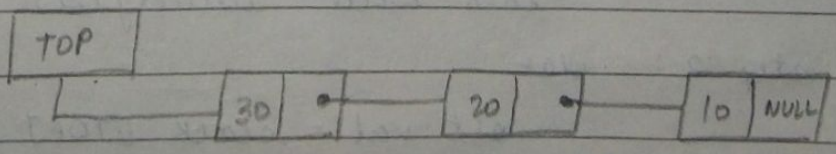
- Applications of stack :

I) stack is used to implement recursive functions.
II) stack is used to solve arithmetic expression, a+b*c/d
III) PUSH() → it is a function which is used to insert a new element in the stack.
IV) POP() → it is a function which is used to delete a element from the stack.
V) stack can be created by using :
   a) array and
   b) linked list
VI) example of stack is :

• by using array



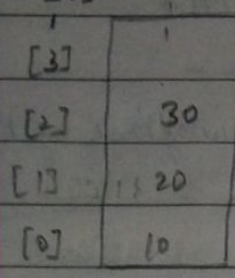
• by using linked list



* TOP contains the position of last inserted element in stack.

DATE : / /
PAGE NO. :

** stack by using Array : * ARRAY



TOP = -1
TOP = 0
TOP = 1
TOP = 2
TOP != 9

▲ 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 " ;

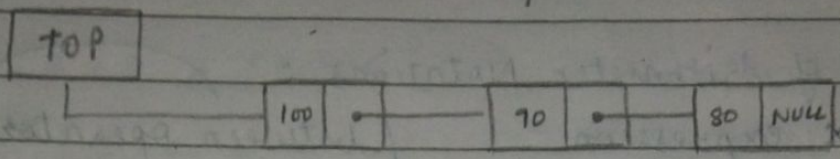
step 4 : exit () ;

* 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)

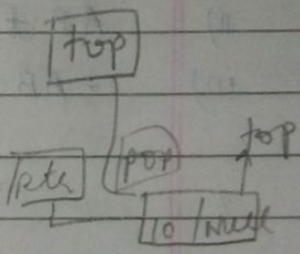
step 1 : if ~~TOP~~ ≠ ∄ ptr = new node;

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 () ;

14.10
2019

ARITHMATIC NOTATIONS

DATE : / /
PAGE NO. :

* Types of Arithmetic Notations :
  - I) INFIX expression (between operator in operands)
  - II) POSTFIX expression (operator after operand)
  - III) PREFIX expression (operator before operand)

I) A+B → between operands
II) AB+ → after operands
III) +AB → before operands

* INFIX to POSTFIX :
* POSTFIX evaluation

POSTFIX - Example :

INFIX ⇒ A+B*C/D

POSTFIX ⇒ A+[BC*]/D

⇒ [A+[BC*D/]]

- Priority based solution
- if same, associative rule will be followed.
- considered as a combined operand.

FINAL ⇒ ABC*D/+

Q: (A+B*(C/D)^E)

= (A+B*[CD/]^E) ⇒ (A+[B]*[CD/E^])

= ([A+[BCD/E^]*]) ⇒ ABCD/E^*+

PREFIX - Example :

A+B/C*D/E

⇒ A+[/BC]*D/E ⇒ A+*/BCD/E

⇒ A+/*/BCDE ⇒ +A/*/BCDE