Algo for Create a node in single linked list:-

step1 Include alloc.h
We will be allocating m/o using dynamic memory allocation in this library all functions are included

step2 Define node structure
```
struct node
{
    int data;
    struct node * ~~next~~;
} *start = null;
```

step3 We create a node using DMA method:-
⇒ We don't have prior knowledge about no. of nodes. So we are call "malloc function to create node at run time

newnode = (struct node *) malloc (size of (struct node));

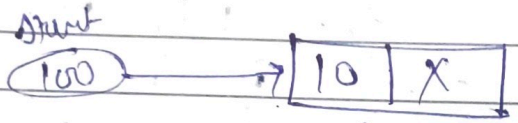
step4 Fill the information in newly created node b-
⇒ whenever we create a new node, make its next field Null.
newnode → next = null;

Name of Lecturer: .....

Steps:- Create very first node :-

if node created in the above step is very first node then

we need to assign it as starting node.



newnode = 100

Set start = temp = newnode

Step 6 Create second or n^{th} node :-

→ Let assume, we have 1 node already created i.e. we have first node.

⇒ Now we have called Create() function again

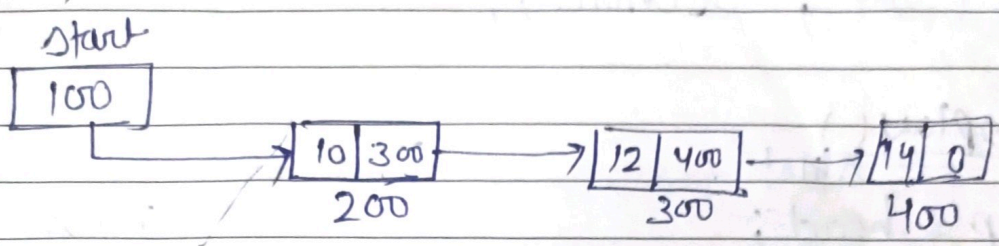
```
else
{
    temp → next = newnode;
    temp = newnode;
}
```

in which we create link between temp & newnode and move temp pointer to the newnode pointer.

Step 7 Stop.

Name of Lecturer: .....

Singly linked list :-




```
void main()
{
struct node
{
  int data;
  struct node *next;
};
struct node *start, *newnode, *temp
start = 0;
int choice;  while(choice)
{
newnode = (struct node*)malloc(sizeof(struct node))

printf("Enter data");
scanf("%d", &newnode->data);

newnode->next = 0
if(start==0)
{  start = newnode;  |  start = temp = newnode;
}
else
  start->next = newnode  |  temp->next = newnode
                            temp = newnode;
```

// want to choice to continue or not.

```
printf(" Do you want to continue 0,1");
scanf("%d", &choice);
}
```

display()

```
temp = head;
while (temp != 0)
{
  printf("%d", temp->data);
  temp = temp->next;
}
getch();
}
```

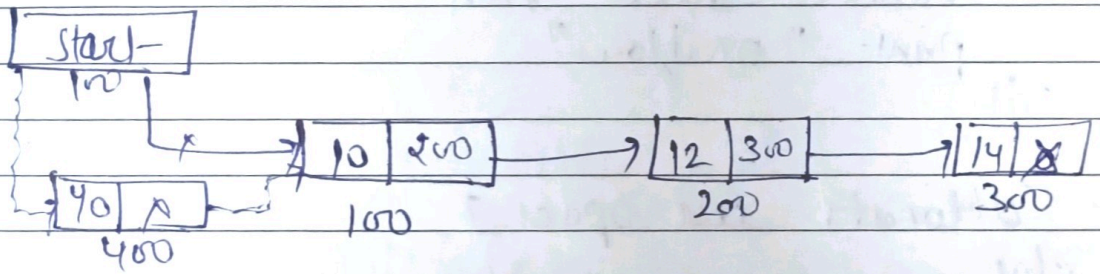
Traversing:- means to process node of linklist, enter display value, count node, sum of value.

```
Void sum (node *p)
{
  int S=0;
  while (p != null)
  {
    S = S+p->data;
    p = p->next;
  }
  printf(" Sum of all node value is=", S);
}
```

Singly linked list- insertion operation

=> Insert at begining
=> Insert at end
=> Insert after a given position

(1) Insert node at begining :-



at begining

```
struct node
{
  int data;
  struct node *next;
};

struct node *start, *newnode;

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

printf("Enter the value u want");
scanf("%d", &newnode->data);
```

newnode → next = start;

start = newnode;

}

Algo INSERTFIRST :-

Step1 [Check for free space.]
if newnode = null then
print "overflow"
Exit

Step2 [allocate free space.]
else
newnode = Create a new node;

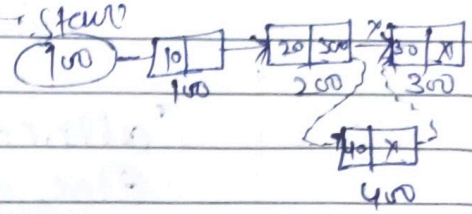
Step3 :- [Read information part of newnode]
data[newnode] = value.

Step4 :- [link address of currently created node with the address start is point.
next[newnode] = start

Step5 : Now assign address of newly created node to the start
start = new_node.

Step 6 : Exit

Insert after a given position:-




```
printf(" Enter the position");
scanf(" %d", &pos);
if ( pos > count)
{
   printf(" Invalid position");
}
else
      temp = start;
      while ( i < pos)
      {
         temp = temp -> next;
         i++;
      }
printf(" Enter data");
scanf(" %d", &newnode -> data);
newnode -> next = temp -> nxt;
temp -> next = newnode;
}
```

Algo INSPOS :-

step1 [Check for free space]
if newnode = null then
print "overflow" and
Exit

step2 [allocate free space]
else
newnode = Create a newnode

step3 [Read value of info part of newnode]
data[newnode] = value

step4 [Move the pointer to desired location]
temp = start;

for (i=0; i<pos; i++) // pos is desired
temp = temp→next position

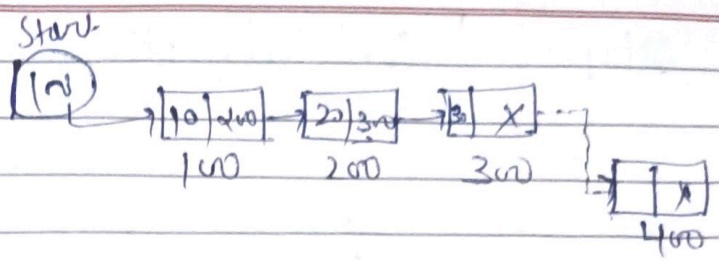
step5 set
newnode → next = temp → next
temp → next = newnode;

step6 Exit.

Name of Lecturer: .....

Excellence in Education

Insert at end :-




```
Struct node
{
    int data
    Struct node *next
};
Struct node *start, *newnode, *temp

newnode = (Struct node*) malloc (Size of (Struct node));

printf (" Enter the value");
scanf ("%d", newnode → data);

newnode → next = 0;
start = newnode

while( temp → next != 0)
{
    temp = temp → next ;
}

    temp → next = newnode
```

Deletion from the linked List :-

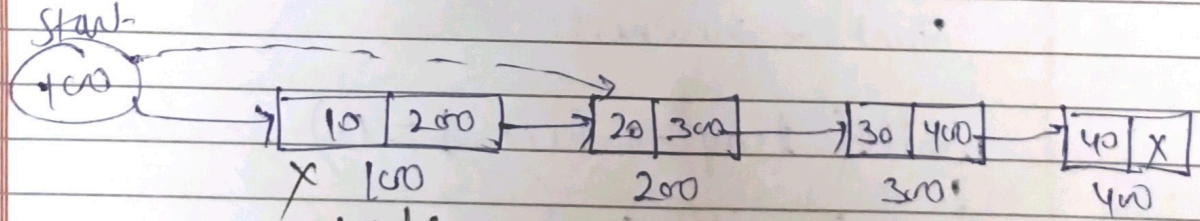
Algo DelFrstNode :-

Step1 (Initialization)
node = Start → next // points to first node in the list.
pre = Start // points at Start

Step2 Perform deletion
if node = null then
print "Underflow"
and Exit
else
pre → next = node → next;

Step3 Free the memory space associated with node
Free(node)

Step4 Exit.



node = Start → next
200

pre = Start
100

fig :- delete first node.

Name of Lecturer: .....

Algo for deleting last node :-

Step 1 Initialization
```
node = start → next;
pre = start;
```

Step 2 If node = Null
print "underflow"
and exit.

Step 3 Reaching to the last node
Repeat while node ≠ null.
```
node = node → next;
pre = pre → next;
```

Step 4
```
pre → next = node → next;
```

Step 5 free the memory space associated with node.
free(node)

Step 6 Exit.

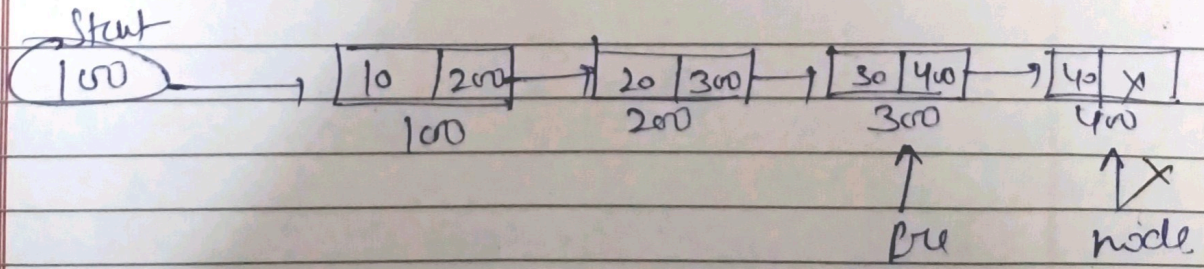
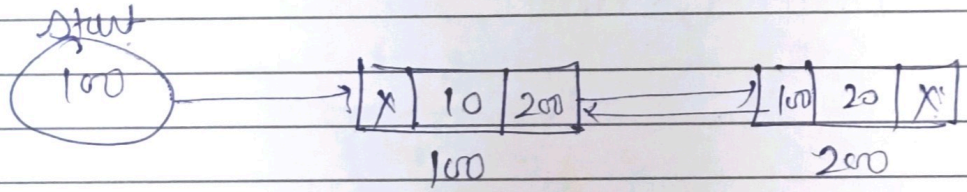


fig :- delete last node

doubly linked list :

```
Struct nodenode
{
    Struct nodenode *pre;
    Struct nodenode *next;
    int data;
};
Struct nodenode *start = null;
```




```
void creat(nodenode *p, int d)
{
    if(p==null)
    {
        start = (nodenode *) malloc (sizeof(nodenode));

        Start → pre → null;
        Start → data → d;
        Start → next → null;

    }
    else
    {   While(P→next != null)
        {
            P = P→next;
        }
```

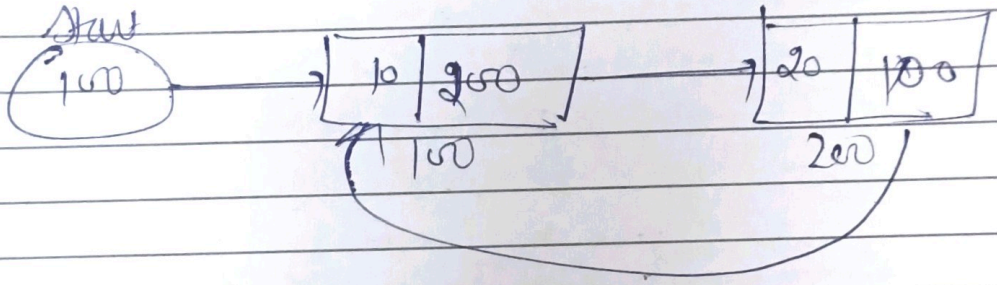
```
p->next = (node *) malloc (size of (node));

p->next->pre = p;
p->next->data = d;
p->next->next = null;

}
}
```

Name of Lecturer: .....

Circular linked list :-

```
Struct node
{
  int data
  Struct node * next;
};
```

```
Struct node * start = null;
```



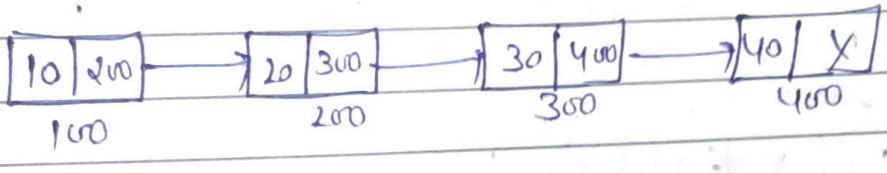

```
void create (node *p, int d)
{
  if (P == null)
  {
    start = (node*) malloc (size of (node));
    start → data = d;
    start → next = start;
  }
  else
  { while (P → next != start)
    { P = P → next.
```

```
p->next = (node *) malloc (sizeof (node));
p->next->data = d;
p->next->next = start;
}
```

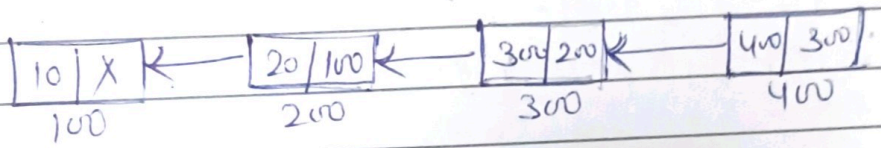
Name of Lecturer: .....

Reversing a Single Linked list :-

input :-

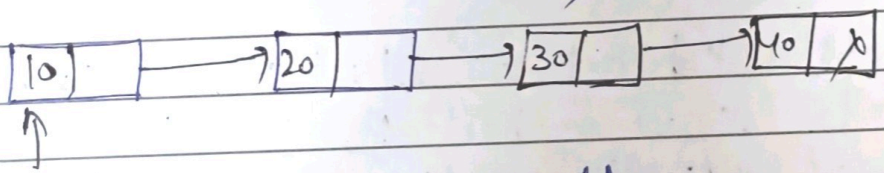


output :-



Step 1 Initialize three pointers :-

```
pre = null;
head = start;
next = null;
```



start
head

pre = null
next = null

Step 2 Traverse / iterate through the links list ; Repeat step (3) (4) (5) until head equals to null.

next = head → next

in which we store the address of next node.

Step 3

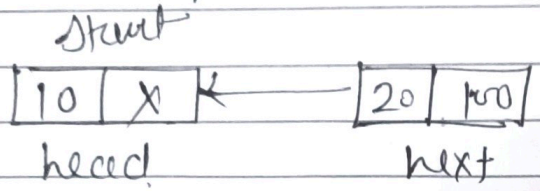
head → next = pre

in which we reverse the link.

step 4 pre = head

step 5 head = next

in which moving pre & head one step forward ← pass!




```
// * C function to reverse the linked list

void reverse()
{
    node *head = start;      // Initialization
    node *pre = null;
    node *next = null;

    while (head != null)
    {
        next = head->next;    // Store next
        head->next = pre;     // Reverse current node
        pre = head;           // Move pointer one
        head = next;          // position ahead.
    }
```