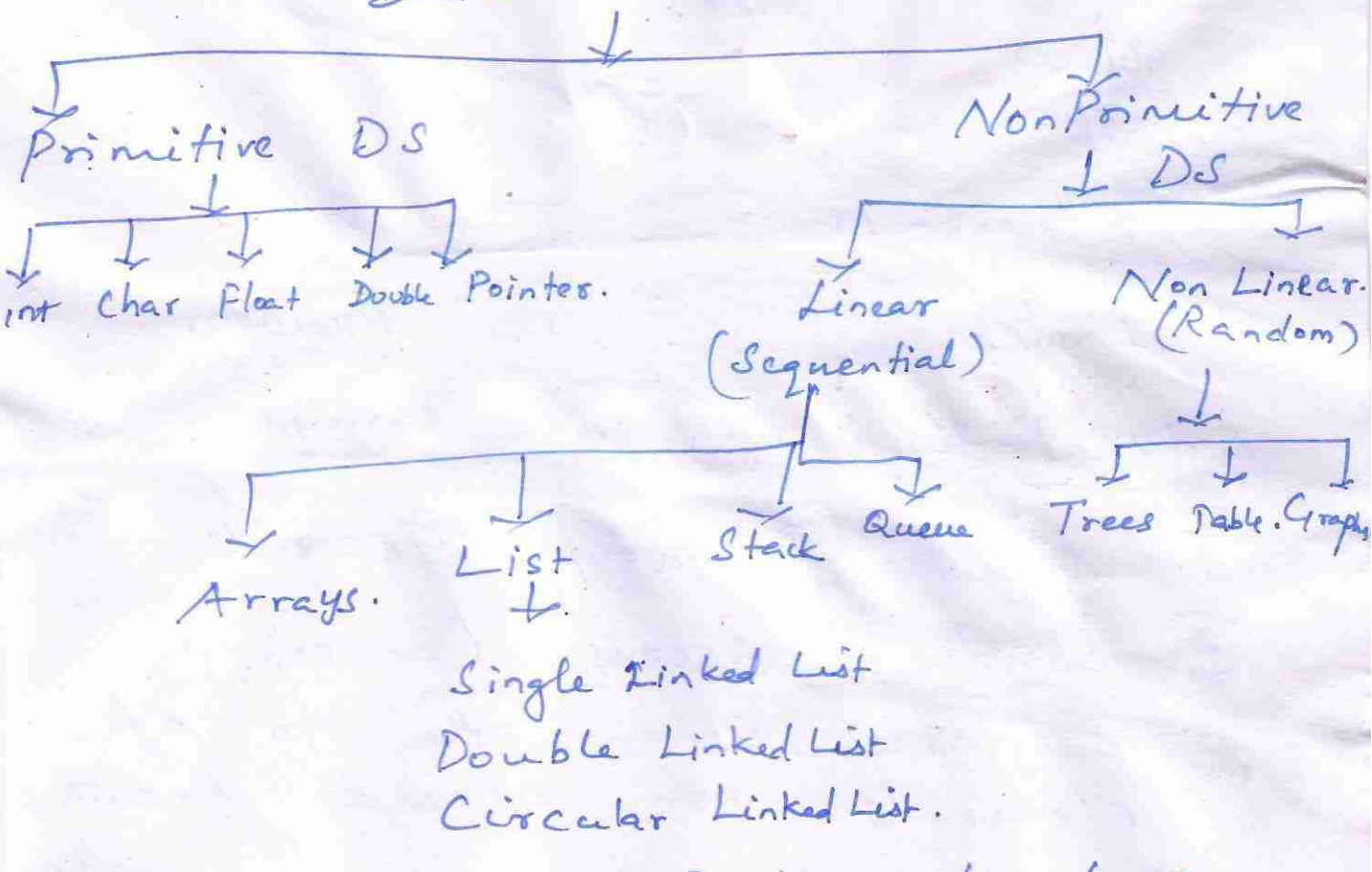
UNIT I

Data Structure:

A data structure is a named location that can be used to store and organize data.

Types of Data Structures.



Operations on Data structure.

1. Searching.
2. Sorting
3. Insertions
4. Deletions
5. Updations.

Arrays

Collection of same data type Stored in a single name is Called as an array.

Ex. Name

int a[10] — Index.

Type — Size.

Elements — The value

Index. — The Position.

Advantages of an Arrays

1. Similar datatype elements
2. Random Access (Index).
3. Suitable when the no of elements are already known.
4. Implementation of Stacks and Queues.

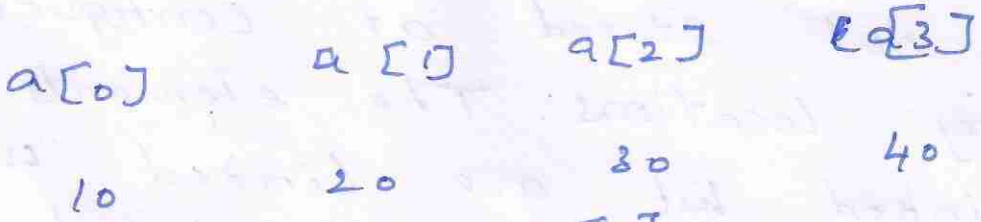
Disadvantages of an Arrays.

1. Static Memory Allocation.

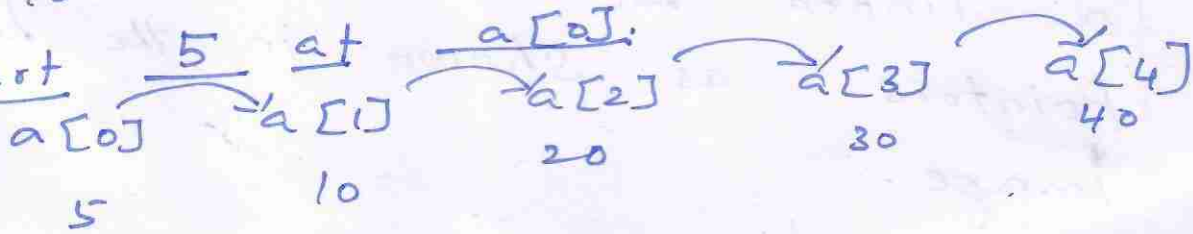
A[10] — No Possible to increase size at compile time.

A[100] — 90 positions

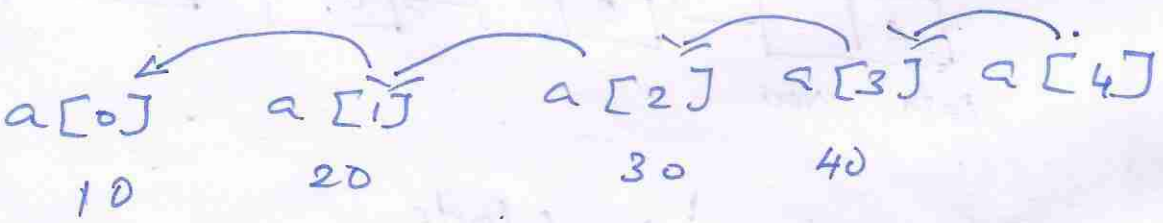
2. Insertion & Deletions are expensive Because It takes more time and more operations.



Insert 5 at a[0].



Delete 10 at a[0].



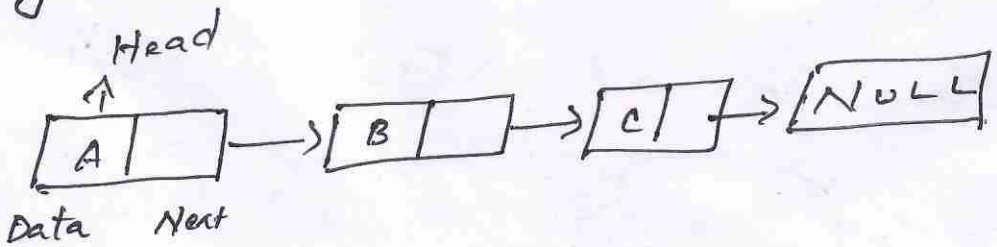
Operations on Arrays

- searching
- Sorting
- Insertion
- Deletion
- Updation.

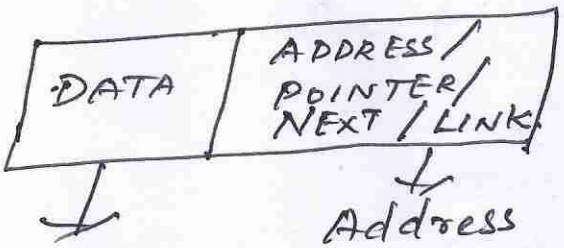
Linked List

Linked List.

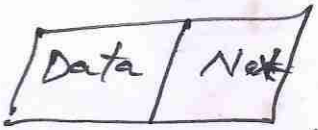
A Linked list is a linear data structure, in which the elements are not stored at contiguous memory locations. The elements in a linked list are linked using Pointers as shown in the below image.



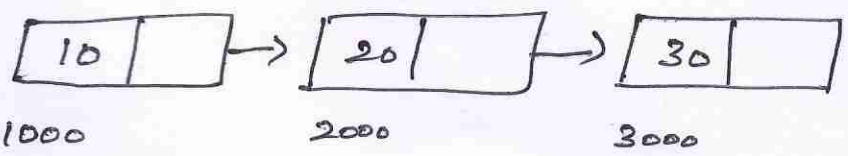
Element → Node
+
For every Node Two Fields. are there.

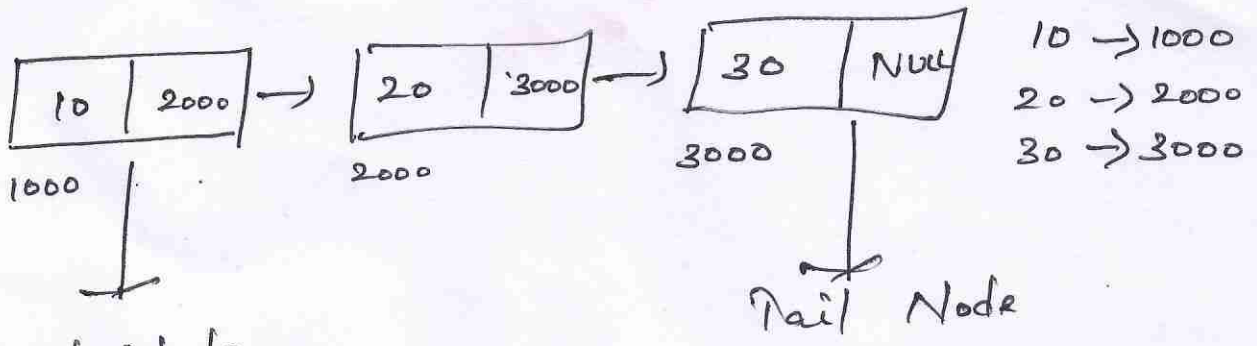


Node Representation.



For with 3 Node Representation.

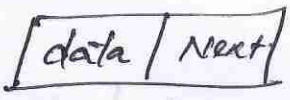




Structure

Collection of different data types stored in a single name is called as structure.

Declaration of a Node.




```
Struct node.
  {
   int data;
   Struct node *next;  (Self Node representing)
  } *new;
```

Struct node *new

DMA – Dynamic Memory Allocation for Structure.

data (value)

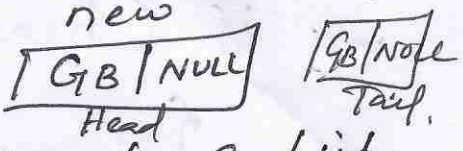
Next (Address of Next value)

```
new = (Struct node *) malloc (sizeof (Struct node));
```

When we create or adding a list we change the tail.

Create a Linked List

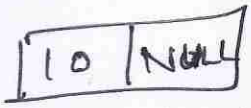
Initialy.



For Create a List.

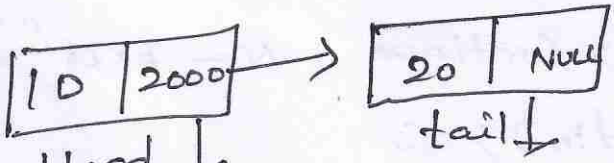
1. Establish a Link
2. Create a Node.

10 → 1000
20 → 2000
30 → 3000

Step 1: 

Head & Tail

It is not an empty List. It has One element 10.

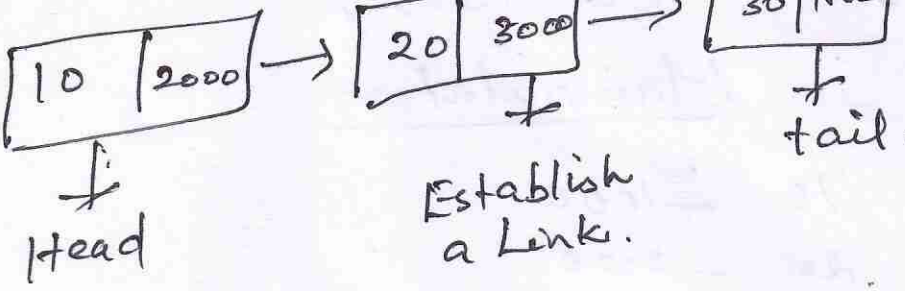
Step 2 : 

(fixed)

Establish Link.

Create Node.

tail->next = new
tail = new

Step 3 : 

Establish a Link.

10 → 1000 → New

20 → 2000 → New

30 → 3000 → New.

do

{

new (struct node*) malloc (sizeof (struct node));

Scanf ("%d", &value.

new → data = value;

new → next = NULL;

```
if (head == NULL)
  {
     head = new;
     tail = new;
  }
else
  {
    tail->next = new;
       tail    = new;
  }
Printf (" y-> Continue , N- Exit ");
 fflush (Stdin);
 Scanf ("%c", &ch)
} while (ch = 'y');
```

Manipulation

10 — 1000
20 — 2000
30 — 3000

value = 10

Step1. new | 10 | NULL

New
head | GB | NULL

New.
Head | GB | NULL

Head →
Tail → | 10 | NULL

Step 2: | 10 | 2000 | → | 20 | NULL

new | 20 | NULL

Head

New = | 30 | NULL

Step 3 | 10 | 2000 | → | 20 | 3000 | → | 30 | NULL

Head

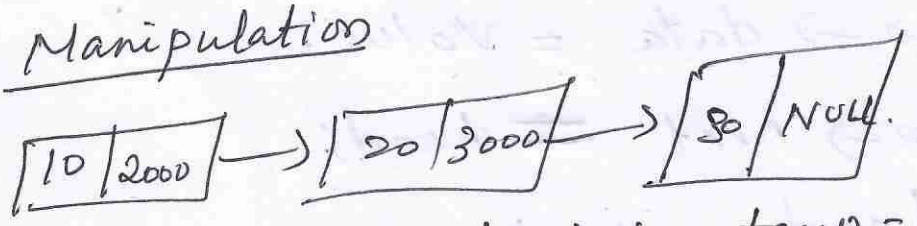
tail

Display the elements in Linked List

```
temp = head;
While (temp != NULL)
{
printf ("%d", temp -> data);
temp = temp -> next;
}.
```

```
struct node
{ int data;
  struct node *next;
} *new, *head, *tail, *temp;
new = (struct node *) malloc (
      sizeof (struct node));
```

Manipulation



head = temp
temp -> data = 10

temp -> temp -> next
temp -> data

temp = temp -> next
temp -> data

10.

20.

30.

temp = temp -> next = NU

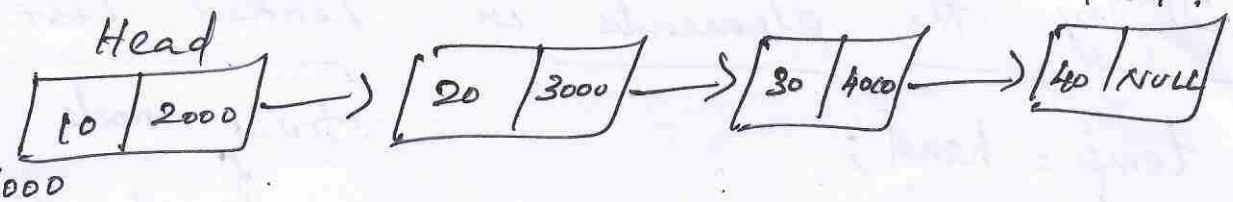
Condition is failed. So the List is Printed with 3 elements.

Insert and elements at Begining.

To insert an element in an existing Linked List in 3 cases.

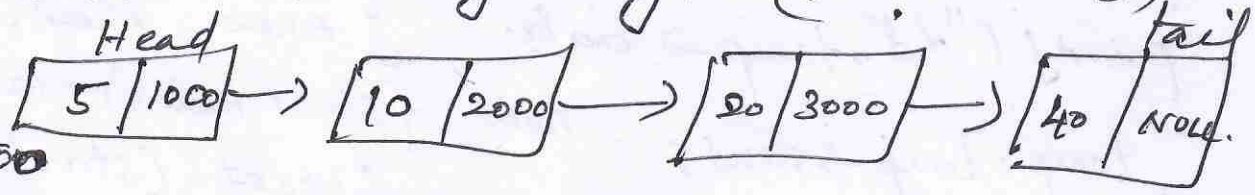
1. To insert an element at Beginning
2. To insert an element at Ending.
3. To insert an element at Specific

For Consider.



new = (Struct node *) Malloc (Size of (Struct node))

1) Insert at Beginning. (Element = 5)



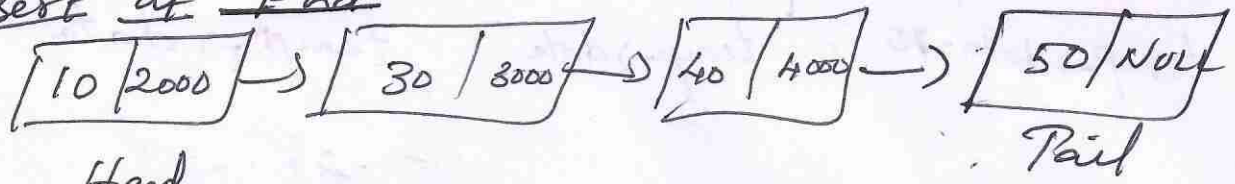
Scanf ("%d", & value); — value = 5

new -> data = value.

new -> next = head;

head = new

2) Insert at End



Scanf ("%d", & value);

new -> data = value;

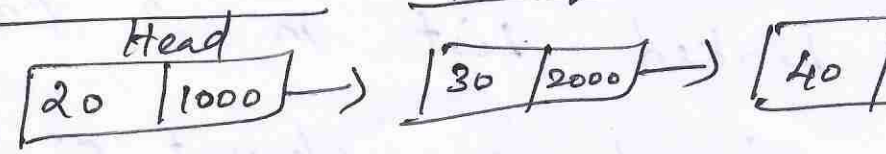
tail -> next = new;

new -> next = NULL;

tail = new;

3) Insert at Specific Position:

Insert Position (2) = 35

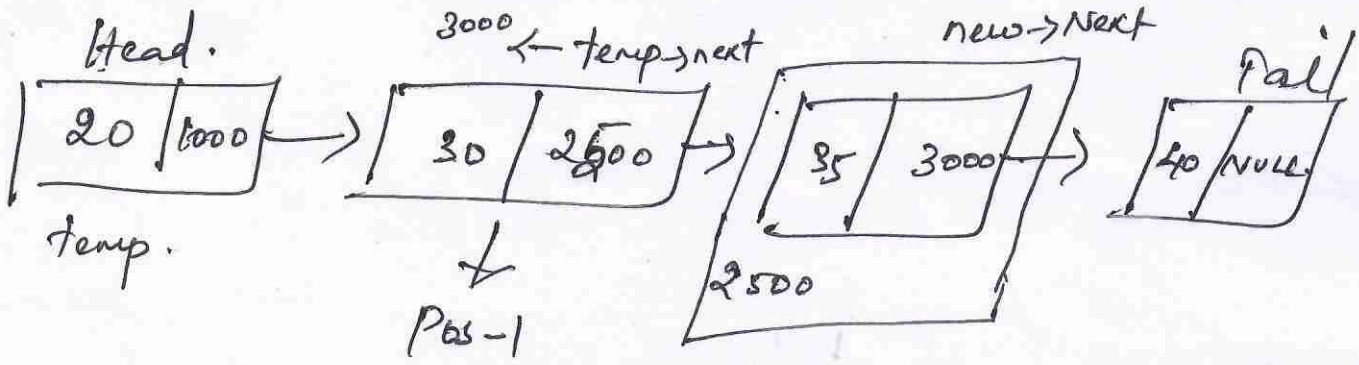


scanf ("%d", & value); value = 35

scanf ("%d", & pos); 2

```
temp = head;
for (i=0 ; i<pos-1 ; i++)
    temp = temp->next;
    new->data = value;
```

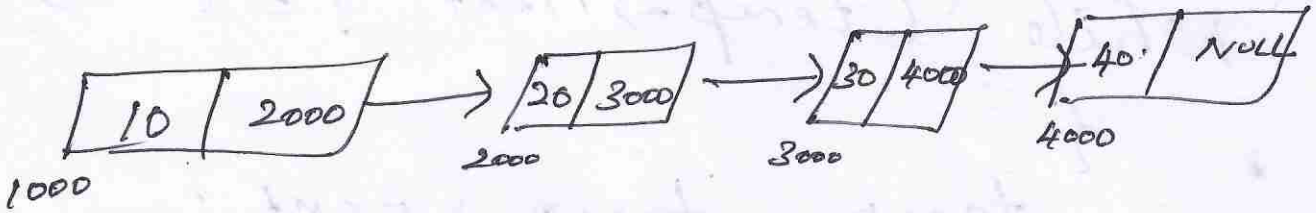
```
new->next = temp->next.
temp->next = new;
```



Deleting an element from Linked List.

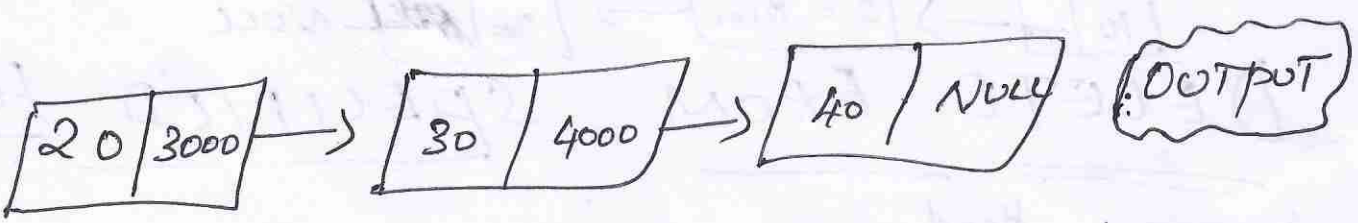
1. DELETE from Beginning.
2. DELETE from Ending.
3. DELETE from the Specified Position.

Consider the list.



DELETE FROM BEGINNING.

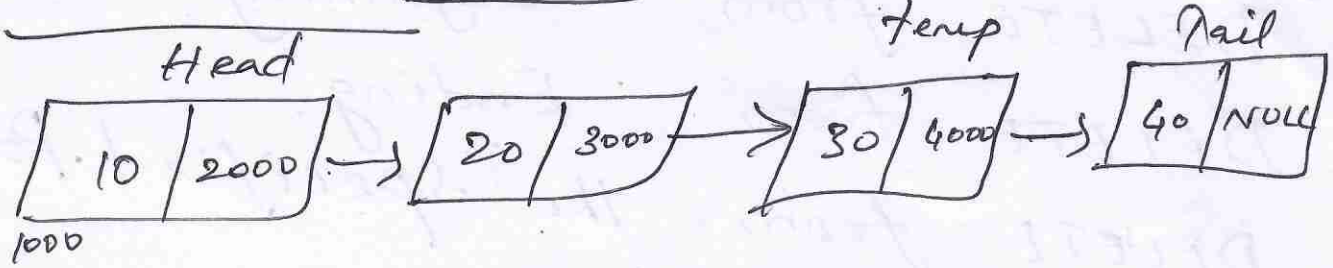
DELETE → DISCONNECT THE LINK + SET NULL at Address Field.



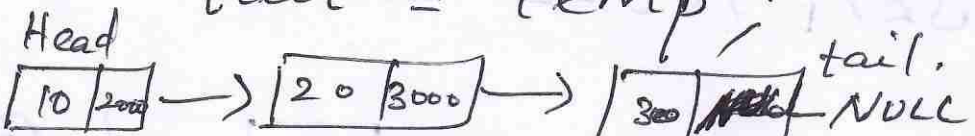

```
temp = head.
head->next = head;
temp->next = NULL;
```

} Don't change the order.

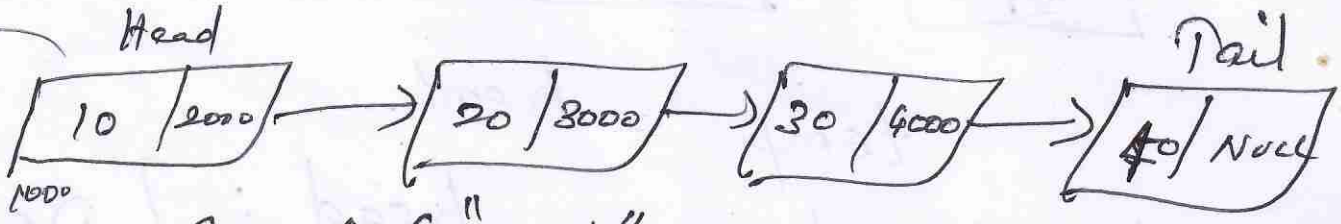
DELETE FROM ENDING.



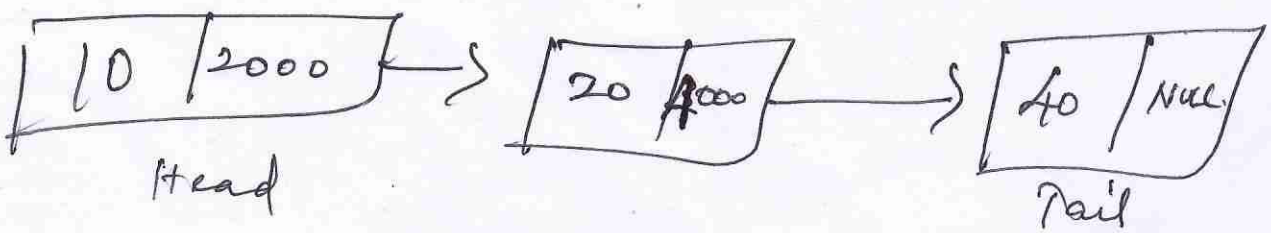

```
temp = head
while (temp -> next != tail)
  {
    temp = temp -> next;
  }
temp -> next = NULL;
tail = temp;
```



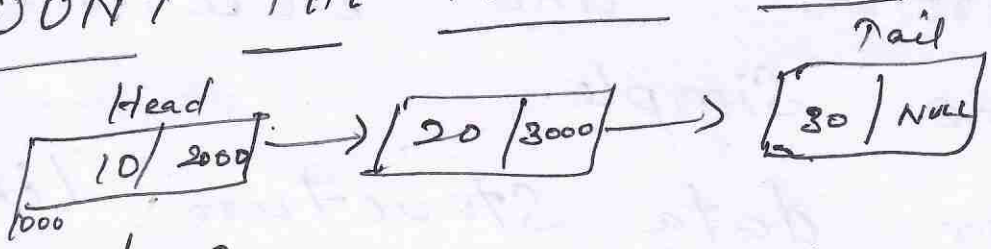
DELETE FROM SPECIFIED POSITION




```
scanf ("%d", &pos);        pos = 2
temp = head;
for (i=0 ; i<pos-1 ; i++)
  temp = temp -> next;
temp->next = temp-> next -> next.
```



COUNT THE NO OF NODES IN LL.



temp.

```
int count=0;
temp = head;
while (temp != NULL)
  {
    count ++;
    temp = temp -> next;
  }
printf ("%d", count);
```

TYPES OF LINKED LIST

1. SINGLE LINKED LIST
2. DOUBLE LINKED LIST
3. CIRCULAR LINKED LIST.

ADVANTAGES OF SINGLE LL

1. No wastage of Memory.
2. No need to use contiguous memory locations.

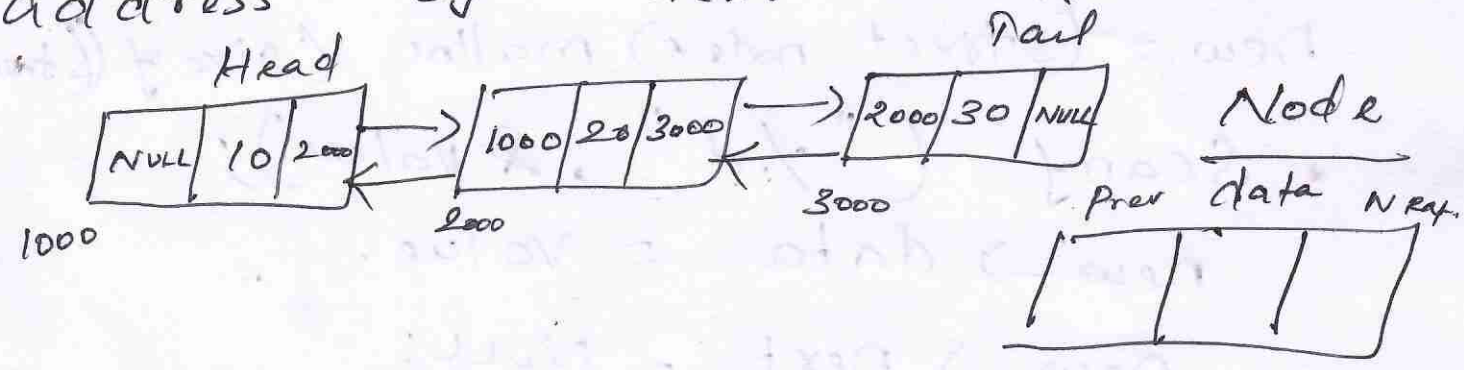
3. Insertion and Deletion become simple.
4. Linear data structure like stack and Queue can be implemented.
5. Can store any type of variable.

DISADVANTAGES OF LINKED LL

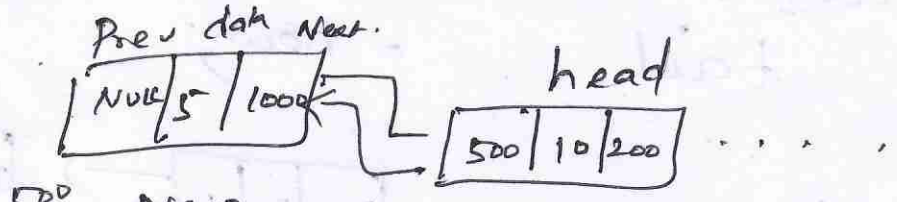
1. Random Accessing is not possibl
2. Only Forward travesal is possibl
3. Binary search Algorithm is not possible.
4. Reversing Linked List is Compl
5. Extra pointer is required.

Double Linked List

Every Nodes have two links that holds the address of Previous node and holds the address of next node.

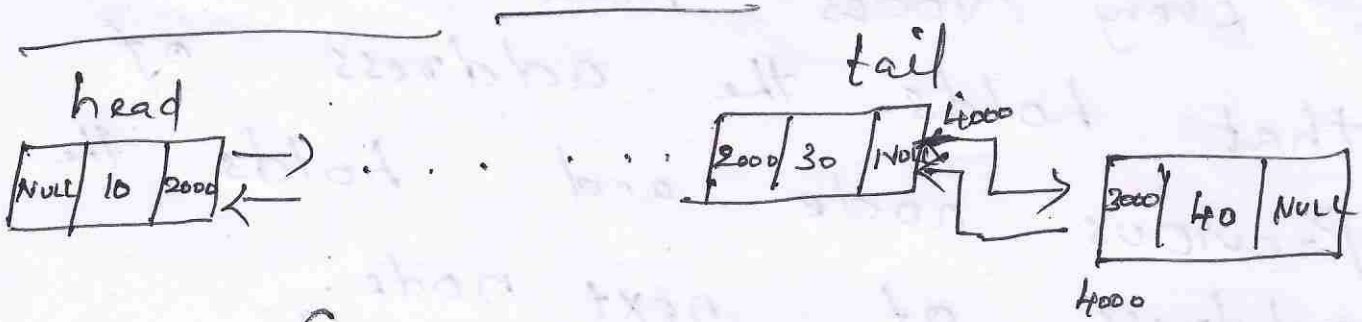


Insert an Element at Beginning

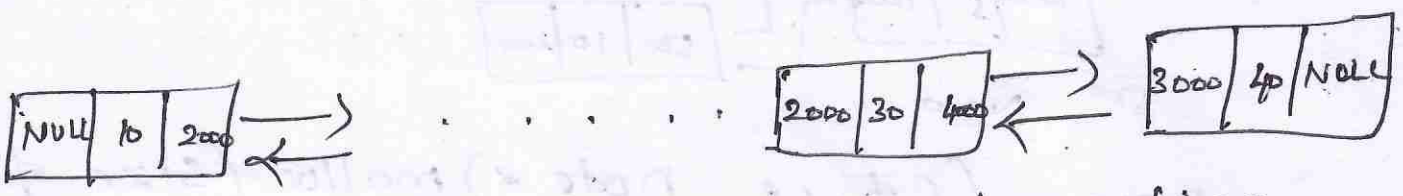



```
new = (Struct node *) malloc (size of
                              Struct node));
Scanf ("%d" &value);
new -> data = value.
new -> prev = NULL
new -> next = head
head -> prev = new;
head = new;
```

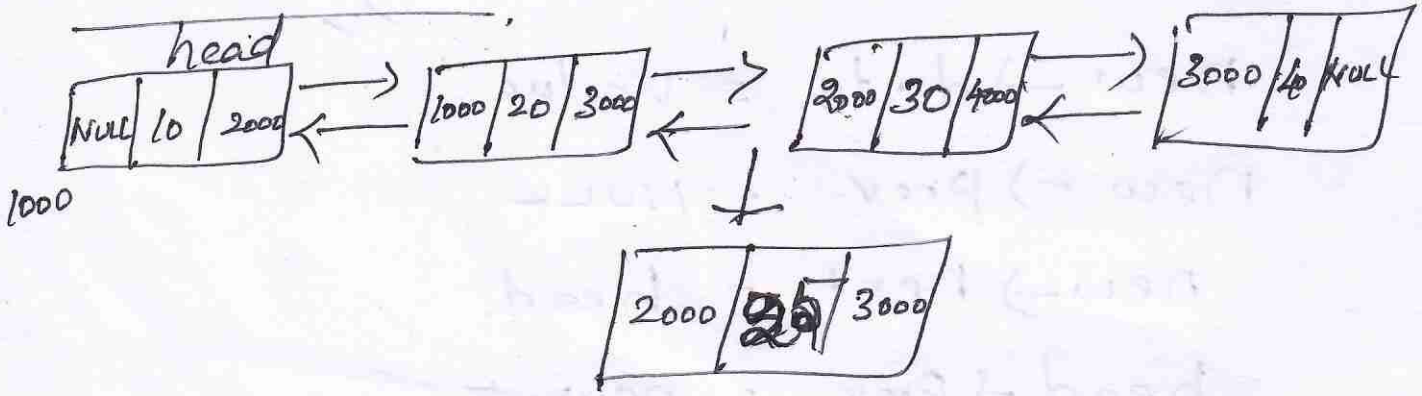
Insert an Element at End.



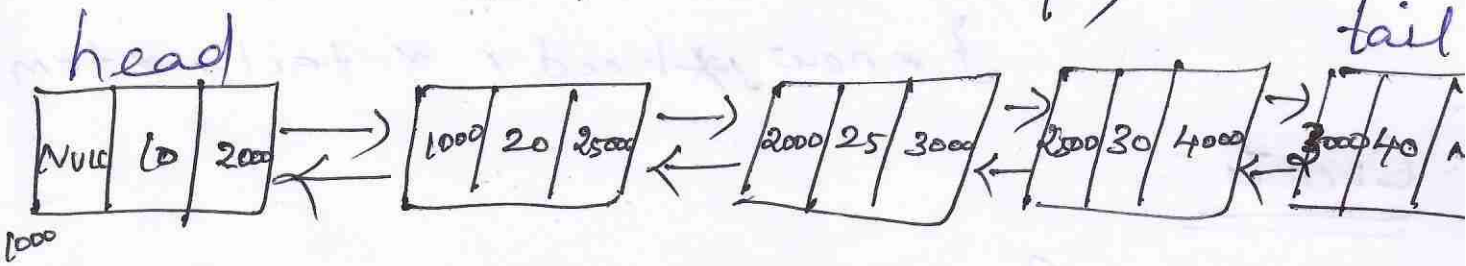

```
new = (struct node*) malloc (sizeof (struct node));
scanf ("%d", &value);
new -> data = value.
new -> next = NULL;
new -> prev = tail;
tail -> next = new;
tail = new;
```



Insert at Specified Location.



```
new = (struct node *) malloc (sizeof struct node);
    scanf ("%d", &value);
    scanf ("%d", &pos);
      temp = head;
for (i=0; i< pos-1; i++)
    temp = temp -> next;
  new->data = value;
new -> next = temp -> next
temp -> next -> prev = new;
      temp -> next = new;
      new -> prev = temp;
```



Delete at Beginning

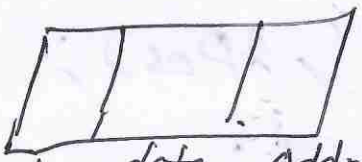
```
temp = head;
head = head -> next
temp -> next = NULL
hed -> prev = NULL;
```

& End

```
temp = tail
tail = tail -> pr
temp -> prev = NUL
tail -> next = NU
```

DOUBLE LINKED LIST CREATION

Node



Address of Previous Node

If it is head

Must Prev = NULL.

Address of Next Node.

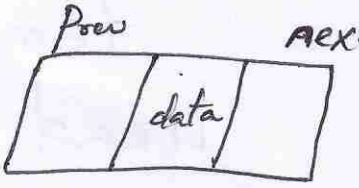
If it is tail

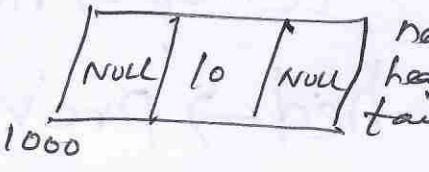
Next = NULL

```
Struct node
{
    Struct node * Prev.
    int data;
    Struct node * next;
} *new, *head, * tail, *temp;
```

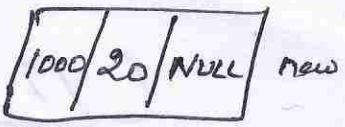
Create

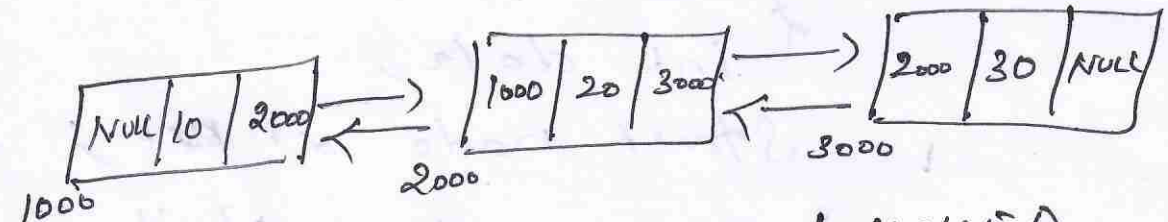
```
new = (Struct node *) malloc (sizeof (structnode));
Scanf ("%d", &value);
new -> prev = NULL;
new -> data = value;
new -> next = NULL;
```





```
if (head == NULL)
{
    head = new;
    tail = new;
}
else
{
    tail->next = new;
    new->prev = tail;
    tail = new;
}
```





DISPLAY DOUBLE LINKED LIST

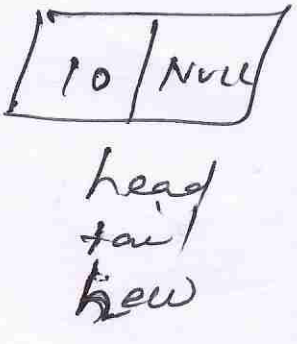
```
temp = head;
while (temp != NULL)
{
    printf ("%d", temp->data);
    temp = temp->next;
}
```

CIRCULAR LINKED LIST

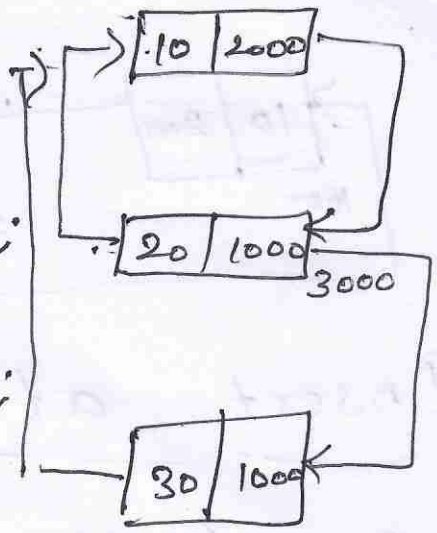
A circular linked list is similar to the single Linked List except that the tail node Points to the first node in the List.

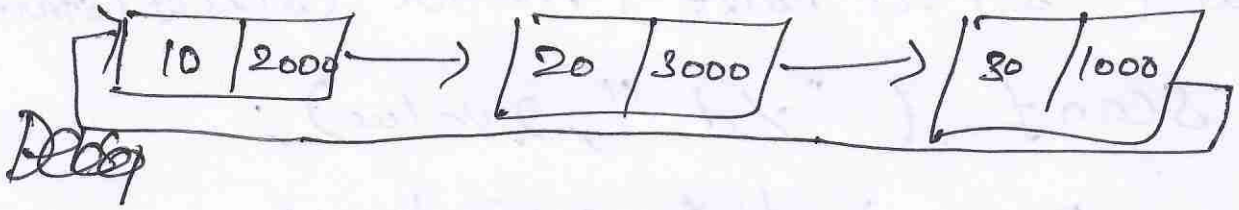
Creation

```
struct node
{ int data;
  Struct node *next;
} *new, *head, *tail, *temp;
do{
new = (Struct node*) malloc (sizeof(struct node))
scanf ("%d", &value);
new->data = value;
new->next = NULL;
if (head == NULL)
{
   head = new;
   tail = new;
}
else {
```



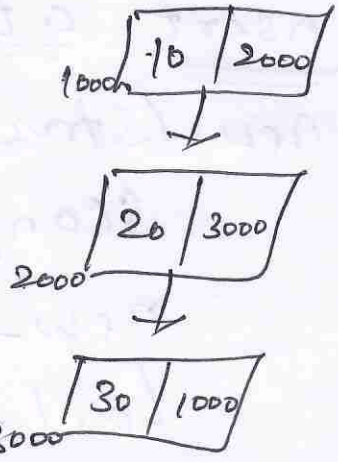
```
tail -> next = new;
tail         = new;
tail -> next = head;
}
printf ('y to Exit, N to cod);
while (ch='y');
```

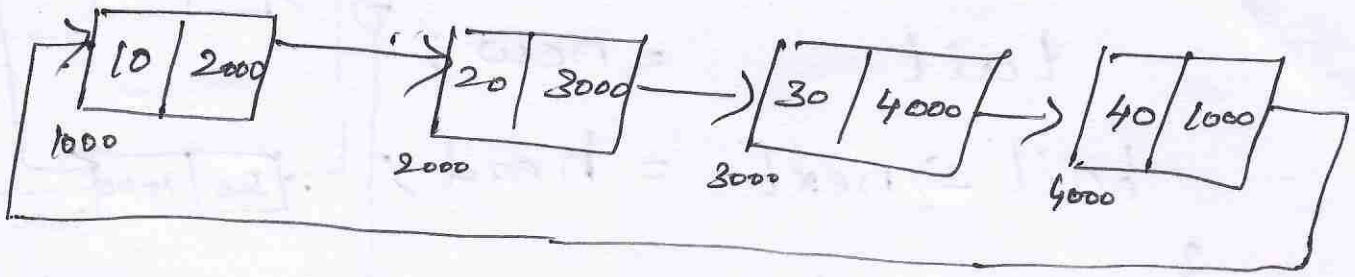




DISPLAY CIRCULAR LINKED LIST

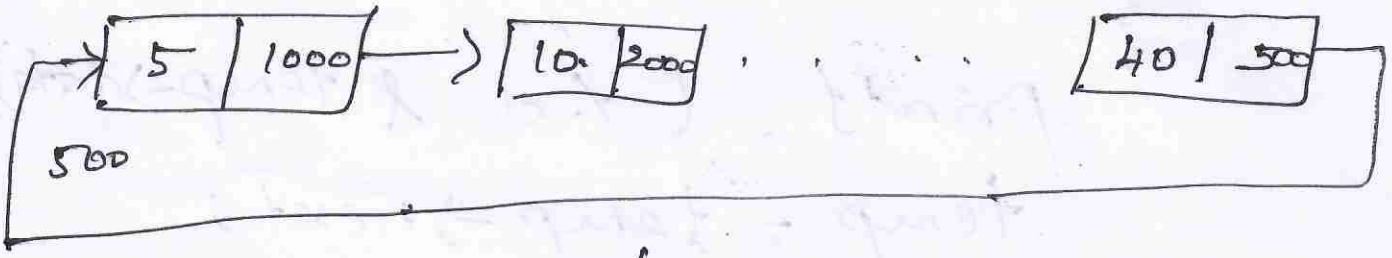
```
temp = head.
While (temp -> next != head)
    {
        printf ("%d" & temp->data);
        temp = temp -> next;
    }
Printf("%d, temp -> data.);
```





Insert at Beginning.

```
new=( struct node* ) malloc (Sizeof (Structnode));
    Scanf ("%d", &value);
        new -> data = value;
        new -> next = head;
        tail -> next = new
        head = new;
```

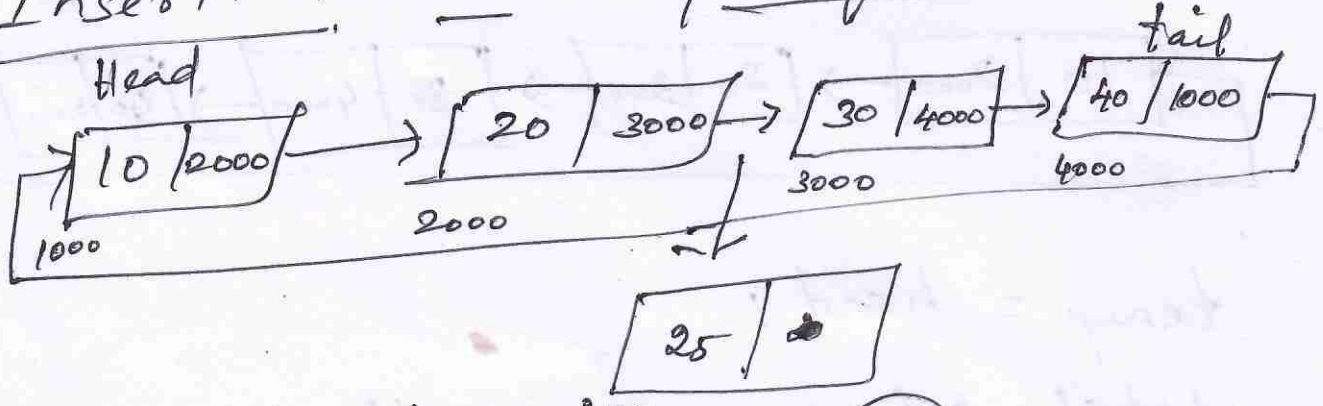


Insert at End:

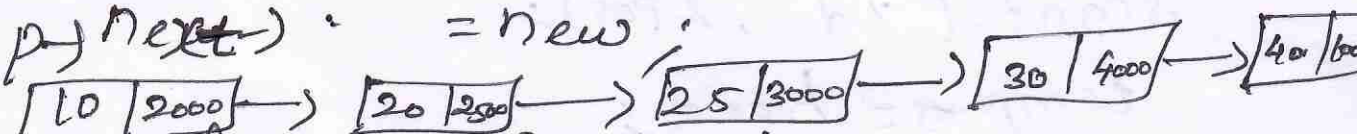
```
new=(Struct node* ) malloc (Size of (Structnode))
    Scanf ("%d", &value);
    new -> data = value;
    tail -> next = new;
    new -> next = tail;
```

tail = new;

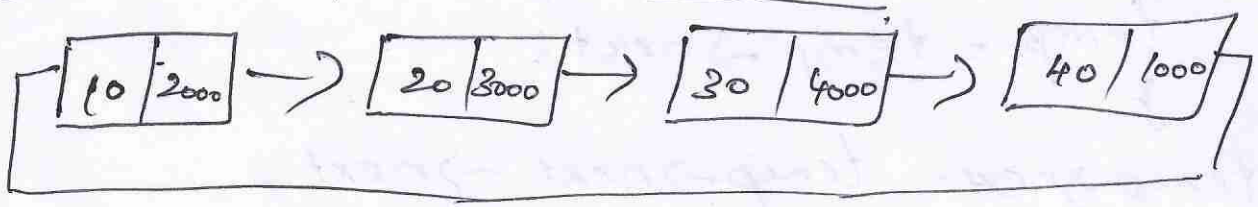
⑧ Insertion at Specified Position.



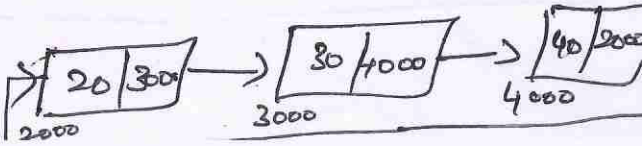

```
Scanf ("%d", &value);   — (25)
Scanf ("%d", &pos);
    temp = head;
for (i=0; i<pos-1; i++)
    temp = temp->next;
    new->data = value
    new->next = temp->next;
temp->next = new;
```



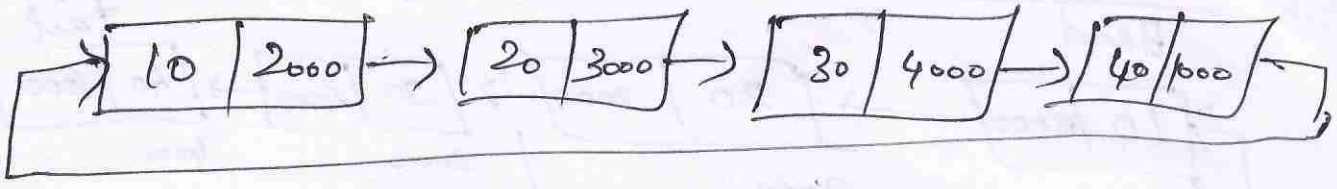
Delete from Beginning.




```
temp = head;
head = head->next.
tail->next = head.
temp->next = NULL;
```



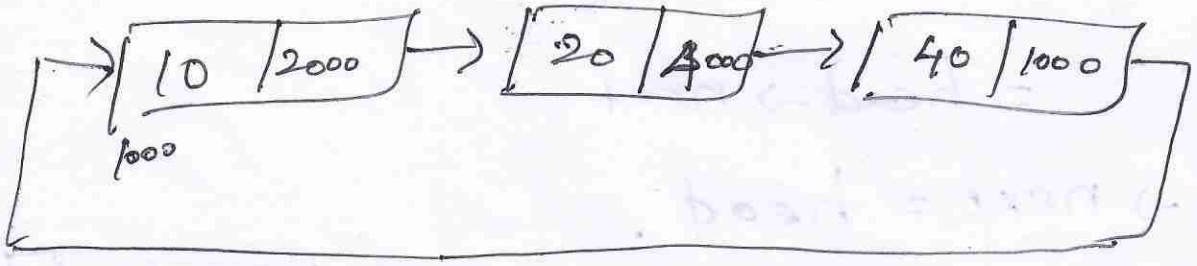
Delete from End.



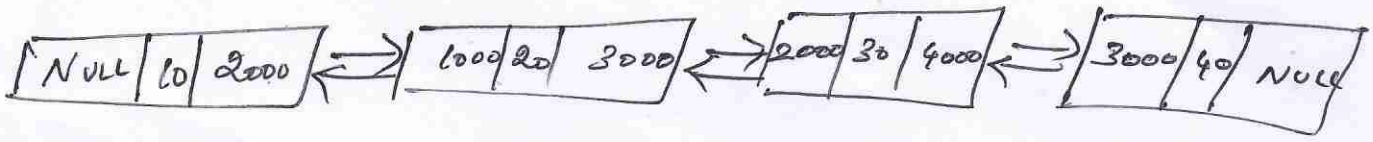

```
temp = head;
While (temp->next != tail)
  {
   temp = temp->next;
  }
   temp->next -> head;
   tail   = temp;
```

Delete from Specified Position.

```
scanf ("%d", &pos);
  temp = head;
for (i=0; i<pos-1; i++)
   {
    temp = temp->next;
   }
 temp->next = temp->next->next;
```



Double Linked List DELETE FROM SPECIFIED POSITION




```
scanf ("%d", &pos)
  temp = head;
for(i=0 ; i < pos-1 ; i++)
  temp = temp -> next;
temp -> next = temp -> next -> next;
temp -> next -> prev = temp;
```