UNIT-3 -

DATA STRUCTURES

Introduction to Data Structures :- When Large amount of data is organize in a memory, we need to maintain The data Efficently by using The Concept of "Data Structures". In present days, Each and Every application process a Large amount of data then Every applications Needs to apply The data Structures technique.

Definition :- Data Structure is a Memory organization Technique. where The data can be stored, processed, retrieved efficiently from The memory

Advantages :-

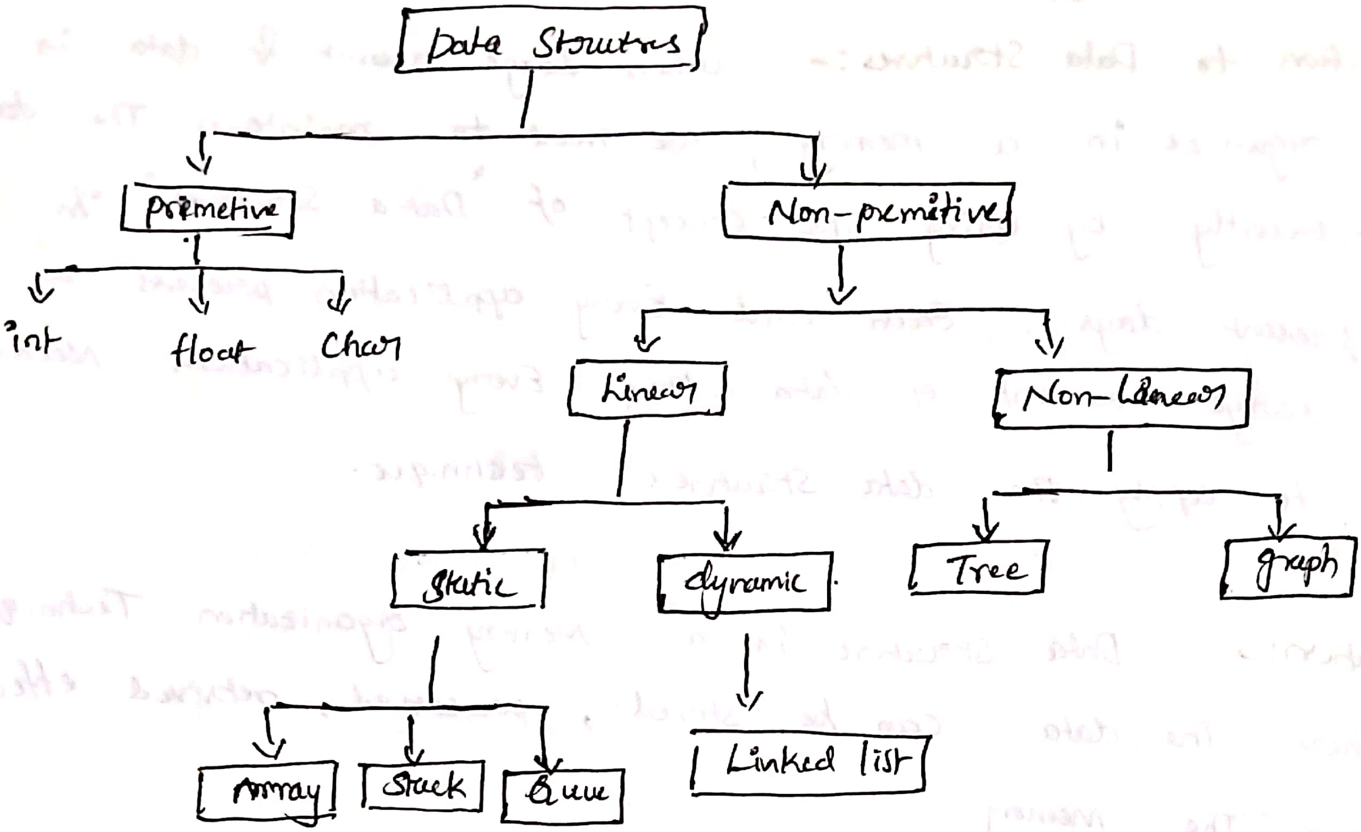
1] Data Structures are used to efficiently organizing and accesing The data quickly

2] Data Structures are always used to reduce the time complexity of a Data processing.

3] In data Structures data always organizing with dynamic approach rather Than Static approach.

4] Data Structures organizing with The data with different operations like Searching, Sorting, inserting, deleting, etc...

5) The following is The Hirarchy for data Structures concept



* The term premitive data structure means where the data structure can hold ony single value. This type of data structures are mainly used in programming language ~~Constructs~~ Con structs

The term Non-premitive data structure means where the data structure can hold more than one value. This type of data structures are mainly used for both programming language constructs and Application constructs.

Linear Search with Recursion :- Linear Search is called as Sequential Search. This is oldest and slow searching Tehnique. for finding the search element The main objective of Linear Search is to Search The element from whole list in Seruentical approach. That means One element after The another element. In This Searching process first we need to a collect a input of elements Then after we need to set a Search element. For finding Search element, we will compare each and every element with Search element until The element is found. If The Search element is not found. Then we can display Search element is not Existing in a list

Ex:- Let us consider The following array of elements. and find The Search element.

a[5] = | 10 | 20 | 30 | 40 | 50 |

a[0] a[1] a[2] a[3] a[4]

Here Search element is : se = 40

Ans:-

if (a[0] == se)   ∴ [a[0] = 10 == se]

10 == 40.

Here The Search element is not found. Then control will go Next

if (a[1] == se)

20 == 40.

∴ Here The Search element is not found.

if (a[2] == se)

30 == 40

∴ Here The Search element is not found.

if (a[3] == se)

40 == 40

break;

Then Search element is found.

programming with recursion :-

```
int lin_recursion (int [], int, int);
void main ()
{
int a[10] ; n, se, i, pos;
printf ("Enter the array size\n");
  scanf ("%d", &n);
printf ("Enter elements in array\n");
  scanf ("%d",
  for (i=0 ; i<=n-1; i++).
  {
  scanf ("%d", &a[i]);
  }.
  printf (" Enter Search element\n");
   scanf ("%d", &se);

pos = lin_recursion (a, n-1, se);
 if (pos <0)
printf ("Element is not found");
else
printf (" Search element %d is located at %d position\n",
         se, pos+1);
}
```

Scanned with OKEN Scanner

}
```
int lin_recursion(int a[10], int n, int se)
{
  if (n>0)
    return -1;

  if (a[n] == se)
    return n;
  else
    return lin_recursion(a, n-1, se);
}.
```

III Binary Search using recursion:- Binary Search is efficient searching technique as compare to Linear Search. Binary Search is working based on Divide and Conquer approach. In This approach The input of elements are partioned into two different parts based on Calculation of mid value. Once the mid value is return we can perform The following steps for finding Searching element.

Here mid = $\frac{low + high}{2}$

{∴ low represents starting position}
{∴ high represent ending position}

i) If (a[mid] == se)
   return mid;

ii) else if (a[mid] > se)
   return high = mid-1;

iii) else
   return low = mid+1;

The above process will be continued until the search element is found.

Note: Binary searching is only working with ordered elements. ∴ That is accessing order & descending order.

Ex:- Let us consider the following array and find a search element (Here search is 14.).

2	4	6	8	10	12	14	16	18
0	1	2	3	4	5	6	7	8

To Calculate Mid = $\frac{low + high}{2}$

$= \frac{0+8}{2} = 4$

low = mid+1. = 4+1 = 5

here a[mid] < se satisfyied that is 10<14.

Then low = mid+1.

12	14	16	18
5	6	7	8.
low			high

mid = $5+8/2 = 13/2 = 6.5 \cong 6.$

Here a[mid] = = se Satisfied.

ie : 14 = =14 Then. Search element is found.

[illegible] :

```
#include < stdio.h>
#include < conio.h >

int binary_rec(int [], int, int, int);

void main ()
{
int a[20]; n, i , se, pos;
clrscr ();
printf (" Enter The size of array\n");
scanf ("%d", &n);
printf (" Enter The elements in array\n");
for (i=0; i<=n-1; i++):
{
scanf ("%d", & a[i]);
}
printf (" Enter The Search element \n");
scanf (" %d ", &se);
pos = binary_rec(a, 0, n-1, se)
if (pos < 0)
printf(" Search element is not found");
```

```
else
    printf (" search element %d occurus. at %d location
            se , pos+1 );

    getch ();

}

int binary_rec ( int a[ ], int d, int h, int se ).

    {
        int mid;
        if (d > h)
        {
        return -1;
        }.
    mid = d+h/2;

    if (a [mid ]== se )
    {
        return mid;
    else if (a [mid ] > se )
        return binary_rec (a, d, mid -1 , se);
    else
        return binary_rec (a, mid+1, h, se );
    }.
```

1. Analysis of algorithms using Time Complexity :-

In data Structures Every algorithm efficiency is calculated by using two factors. That is time complexity and space complexity.

Time complexity is defined as the amount of time required for executing the instructions of algorithm. The time complexity of algorithm can be calculated based on the following factors

a) Type of a processor used. [single processor, (or) multi processor)

b) Type of Architecture used [8 bit, (or). 32 bit (or) 64 bit ..etc]

c) The unit time cost of time Execution required for Executing the operations of Algorithm.

ie: Assignment operator, Arithmetic, logical, etc.

Ex:- Let us consider the single processor and 32 bit machine Architecture. To calculate the time complexity for the following piece of code (or) Algorithm.

```
Sum_of_numbers (int a [], int n)
{
    int sum= 0;
for (i=0 ; i<n ; i++)
 {
    sum = sum + a[i];
 }
  return sum;
```

cost	repetation	total
1	1	1
1+1+1	1+(n+1)+n	2n+2
1	2n	2n
1	1	1

The above

The time complexity for the above code. $T(n) = 4n+4$

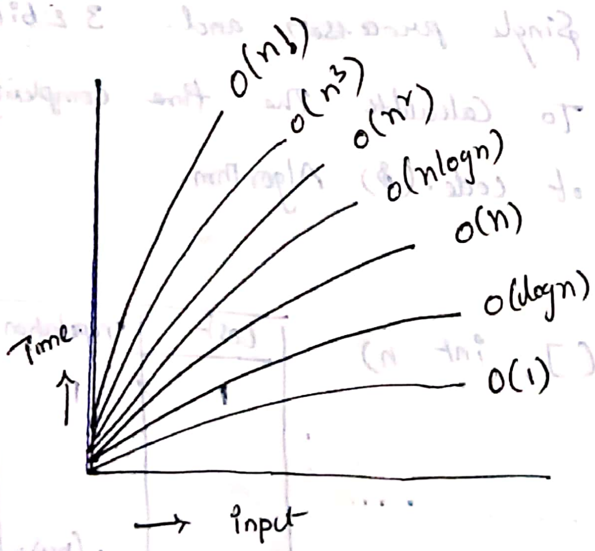
Generally the time complexity of Algorithm is mesured by using Big-oh notation ('O'). According to notation., The time complexity is representing with polynomial Terms.

$\therefore \ T(n) = c \cdot n + c'$

where c, c' are constants for a polynomial.

$\therefore \ T(n) = O(n)$

The following diagram shows. The performance of analysis of algorithms with time complexity



what is Linked List ? Types of Linked List :-

Linked List is also called as Linear List and which Supports Linear data Structure. In Linked List the elements are organized in memory dynamically with inter connection of a addresses by using Sequntial approach. Thats why Linked list is ealso called as Linear List.

In Linked List The elements are organizing in memory with a data-node representation. where The data node. Consists of two parts That is First part Storing a data. and the Second part storing address of a node.

The following Shows The Structure of a data node.

data node.

data	Address

There are Three types of Linked list

1) Single Linked list

2) Double Linked list

3) Circular linked list

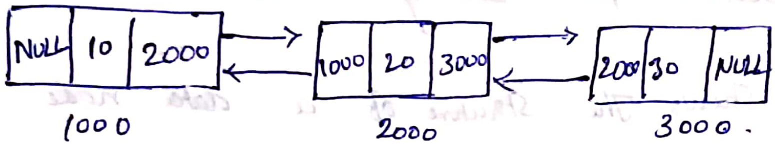
1) Single Linked List :- This is a default Linked list for implementing The list of elements. where The Elements are organizing in a Sequential fashion with connecting of address d next node in memory. In This type of Linked List the elements are only traversing in uni direction

Ex :- head

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

2) Double Linked List :- In doble linked list. The elements are organized in memory with a representation of data node. but here The data node. consists of a Three parts.. The First part indicates storing The address of a previous node. The Second part indicates data, The third part indicates storing address of next node. In This type of list The elements are inter connect to each other. in a by directional way.

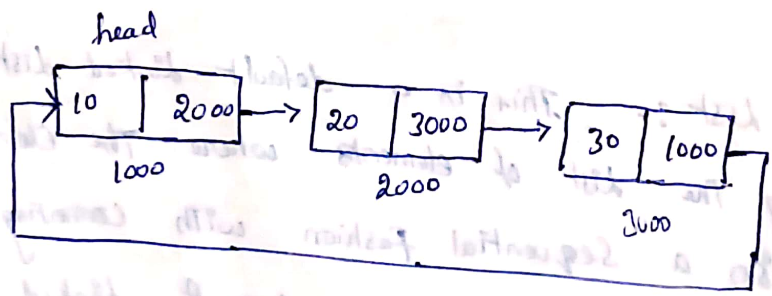
Ex:-

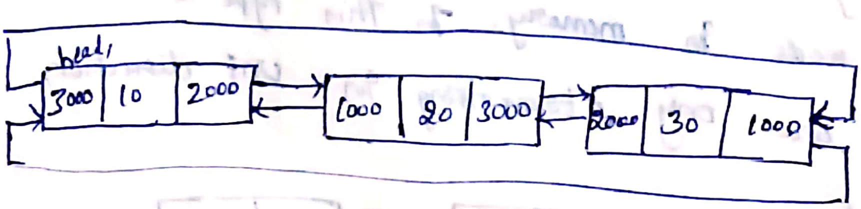


3) Circular Linked List :-

The Circular Linked list also represented with Circular Linked list and Circular double Linked list In Circular Single linked list The starting address of node in Connected to last node in a Circular fashion. In Circular double linked list The Last and First address node are mutualy inter connected. in a Circular fashion.

Ex 1:-





operations On single Linked List :- There are Three operations on Single Linked List (i) insertion

(ii) deletion

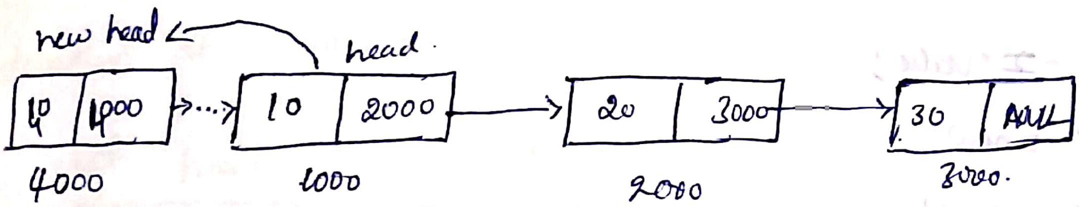
(iii) Display.

1) insertion :- There are Three ways of insertion.

(i) insertion at begining.

(ii) insertion at Ending

(iii) insertion at any specific position.



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

printf ("Enter The value.");

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

new → data = value;

new → next = head;

new = head;

struct node.

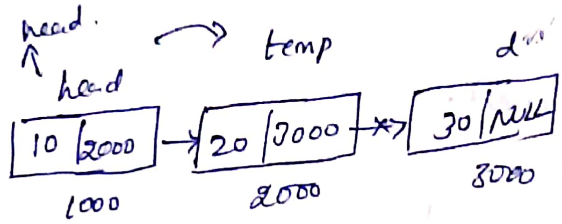
{

int data;

struct node *next;

} *new, * head;

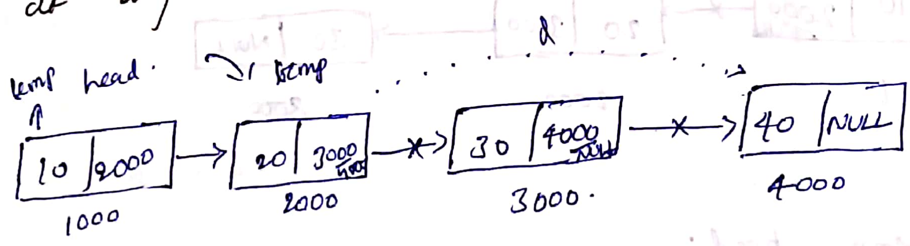
Deletion at ending:-




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

```
Struct node
{
int data;
Struct node *next;
}*temp, *head, *d;
```

Deletion at any specific position:-

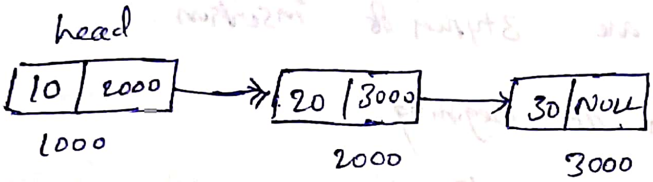



```
int pos, i;
temp = head;
printf(" Enter a position you want del");
Scanf(" %d", & pos);
for (i=0; i< pos -1 & i++)
{
    temp = temp ->next;
}
```

② Struct

```
d = temp -> next;
temp -> next = d -> next;
d -> next = NULL;
free (d);
```

Display :-




```
if (head == NULL)
{
    printf ("List is Empty");
    exit(0);
}
else
{
    temp = head;
    while (temp -> next -> next != NULL)
    {
        printf ("%d\n", temp -> data);
    }
    temp = temp -> next;
    }
}
```

```
struct node
{
  int data;
  struct node *next;
} *temp, *head;
```

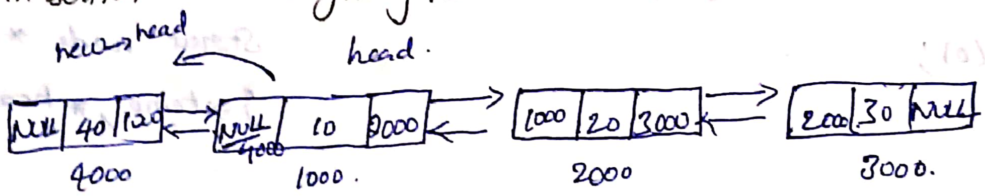
VII operation on double Linked List :- There are 3 operations in double Linked List

(i) Insertion
(ii) deletion
(iii) Display.

(i) Insertion :- There are 3 types of insertion.

(i) insertion at Begining
(ii) insertion at ending
(iii) insertion at any or specific position.

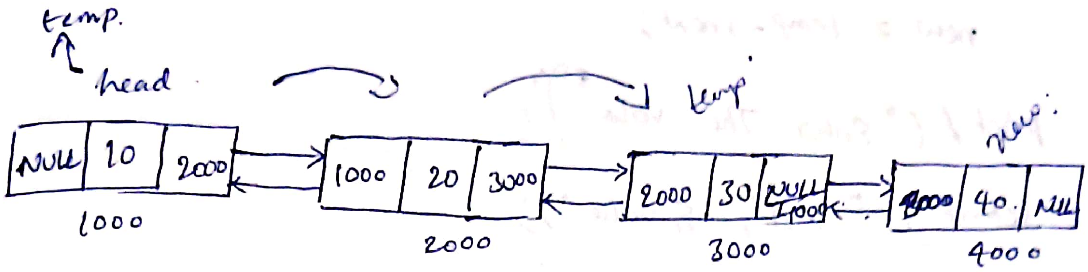
(i) insertion at Begining :-




```
int Value;

new = (Struct node*) malloc (Sizeof (Struct node));

printf("Enter Value\n");
scanf("%d", &value);

new -> data = value;

new -> next = head;
head -> prev = new;
new -> prev = NULL;
head = new;
```

```
Struct node
{
int data;
Struct node *Next
Struct node *prev
};
*new, *head;
```

(ii) insertion at ending :-



int value ;

temp = head;

while (temp ->next != NULL)

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

print ("Enter value\n");

scanf ("%d", &value);

new -> data = value ;

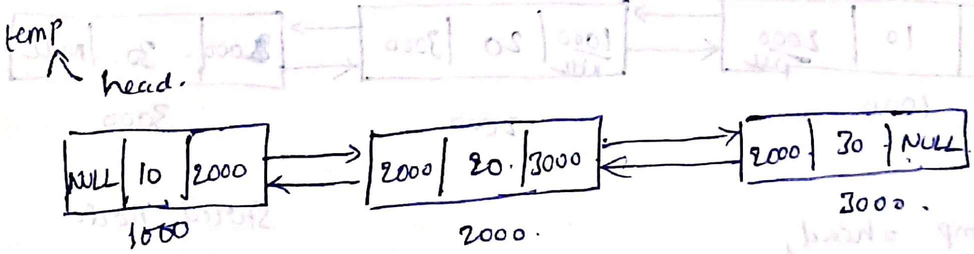
temp ->next = new;

new ->prev = temp;

new ->next = NULL.

struct node
{
int data;
struct node * next
struct node * prev.
} *new, * head, * temp.
~~struct *new~~

(iii) Insertion at any specific position :-



int pos, value, i;

temp = head;

printf (" Enter the position you want insert\n");

scanf("%d", &pos); ②.

for (i=0; pos-1; i++);

temp = temp ->next;

struct node
{
int data;
struct node * next;
struct node * prev;
} * new , * head
* temp.

Scanned with OKEN Scanner

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

new = temp -> next;

printf ("Enter The value:");

scanf ("%d", &value);

new -> data = value;

temp -> next = new -> next;

new temp -> next -> prev = new;

temp -> next = new.

new -> prev = temp.

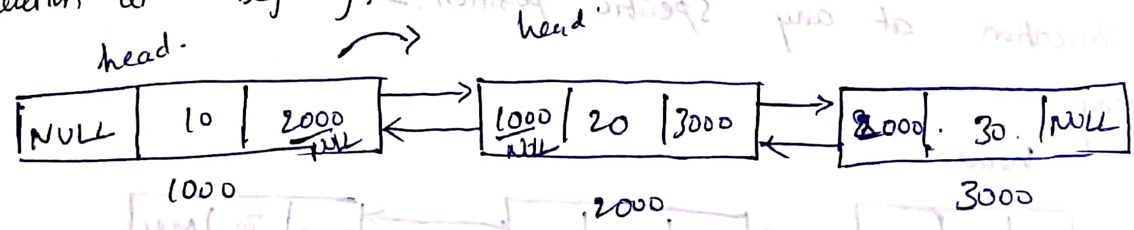
Deletion :- There are Three operations in deletion

(i) deletion at begining

(ii) deletion at ending

(iii) deletion at any specific position.

(i) deletion at begining:-



temp = head;

head = head -> next;

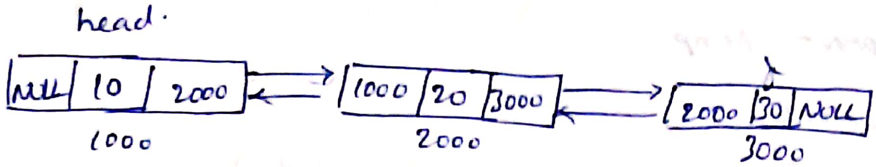
temp -> next = NULL;

head -> pre = 0;

free (temp);

Struct node
{
int data;
Struct node * next;
Struct node * prev;
} * temp, * head;

(ii) Deletion at ending :-



temp = head;

while (temp→ next →next ! = NULL)

{.

temp = temp → next;

}.

d = temp → next;

temp → next = NULL;

d → prev = NULL .

free (d);

Struct node.

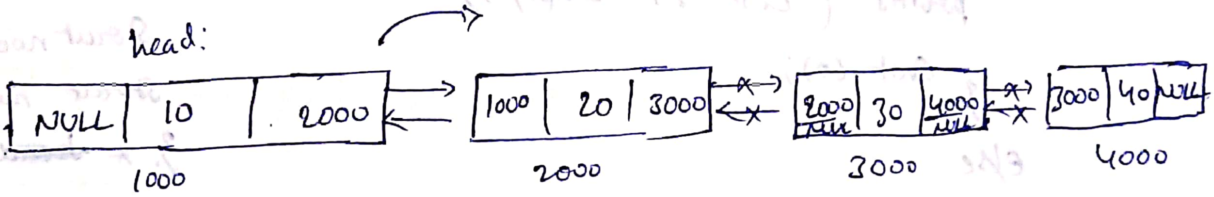
{ int data;

Struct node * next;

Struct node * prev;

} * temp , * head, * d;

(i) Deletion at Specific position :-



int , pos, i

temp = head;

printf (" Enter position you want ");

Scanf (" %d ", & pos); → ②.

for (i = 0; i < pos-1 ; i++);

0 < 1

1 < 1

{.

temp. = temp → next;

}

d = temp → next;

temp → next = d → next;

Struct node.

{.

int data;

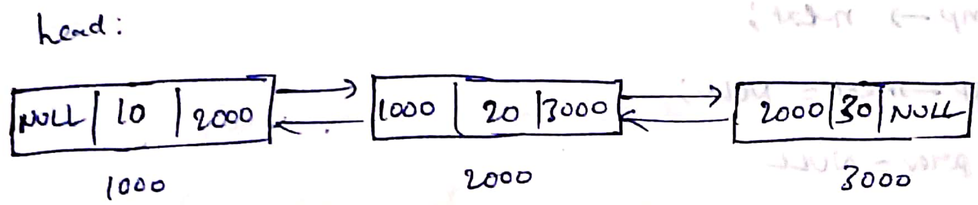
Struct node * next;

Struct node * prev;

} * temp , * head , * d;

```
d-> next ->prev = temp;
d-> next ->prev = temp.
d-> next = NULL;
d-> prev = NULL;
free (d) ;
```

Display :-




```
Struct node
{
  int data.
  Struct node *next
  Struct node * prev;
}; * *temp, *head

if (head ==0(NULL)
 {.
 printf ("List is Empty");
  exit (0);
 }.
else.
 { temp = head;
while. ( temp -> next! =NULL)
 {.
  printf ("%d\t", temp -> data);
  temp = temp -> next;
 }.
```

```
temp = temp -> next;
printf ("%d", temp -> data);
}
```

VIII. Circular linked list :- (circular Singular) :-

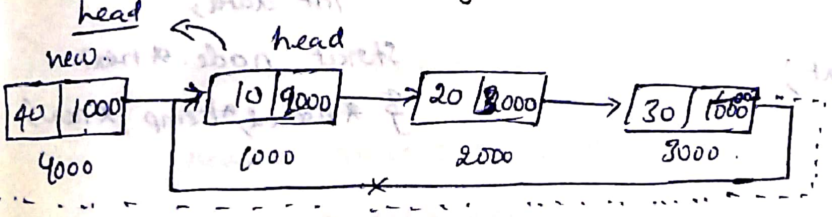
There are three operations in circular linked list

(i) Insertion
(ii) deletion
(iii) Display.

(i) Insertion :-

(a) Insertion at begining
(b) Insertion at ending
(c) Insertion at specific position.

insertion of at begining :

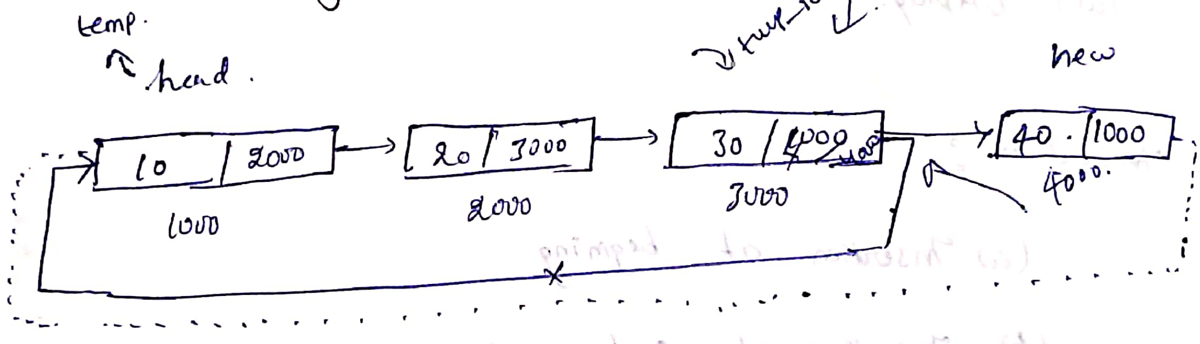



```
int value;
new = (struct node *) malloc (sizeof (struct node));
temp = head;
while (temp -> next != head)
{
    temp = temp -> next;
}
temp -> next = new;
```

```
struct node
{
    int data;
    struct node * next;
} *temp, * head, *new;
```

```
printf ("Enter The value");
scanf ("%d", &value);

new -> data = value;
new -> next = head;
   head = new;
```

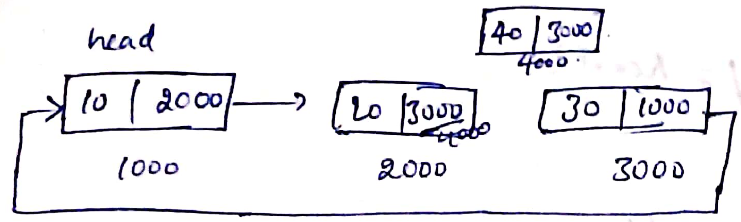
(ii) Insertion at ending :-




```
int value;
temp = head.
while (temp -> next != head)
{
   temp -> next -> next;
}.
new = (Struct node *) malloc (sizeof (Struct node));
   new -> next = head;
printf (" Enter The Value");
scanf(" %d", &value);
new -> data = value;
temp -> next = new;
```

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

(ii) Insertion at any specific position:-

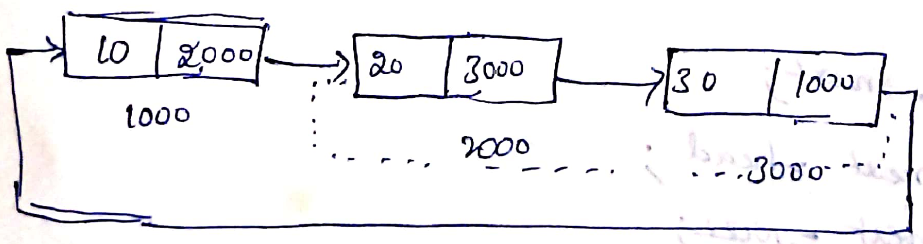



```
int Value, pos; i;

printf(" Enter The Position you want");
 scanf ("%d", &pos);
   temp = head;

for (i=0; i<pos-1, i++)
 {
temp=temp -> neat;
 }
new= (Struct node*) malloc ( Site of ( Struct node ));
 .printf (" Enter The Value.");
  scanf ("%d, & vale);
    new -> data = Value;
    new -> neat = temp -> neat;
     temp -> next = new;
```

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

Deletion :- (a) Deletion at begining.

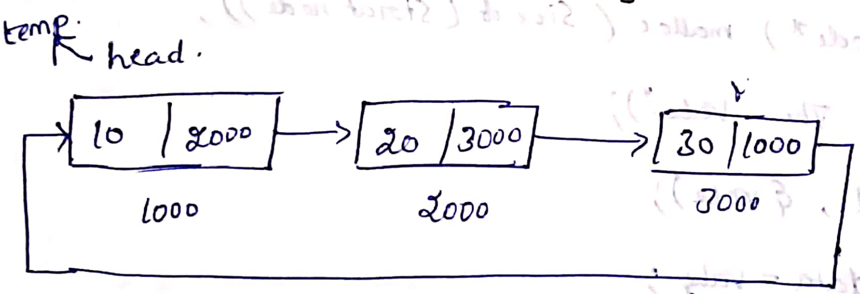
head.




```
Struct node
{
 int data;
Struct node* next;
} * temp, *head, *d;
```

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

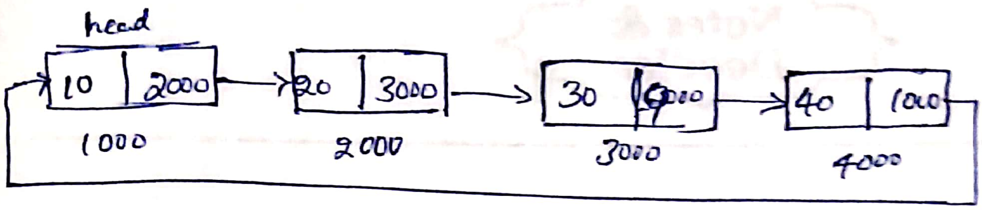
(ii) ~~Insertion~~ deletion at ending point :-




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

```
struct node
{
    int data.
    struct node *next
} temp, *head, *d.
```

(iii) Deletion at any specific position :-




```
int pos, i;
temp = head;
printf ("Enter position you want\n");
scanf("%d", &pos);
for (i=0; i<pos-1; i++)
{
   temp = temp ->next}
}.
d= temp -> next;
temp ->next = d ->next;
d->next = NULL;
free (d);
```

```
Struct node
{.
  int data.
  struct node * next
}. *temp* head* d*
```

End of chapter-3

Notes & Doodles

display operation:-

| 10 | 2000 | | 20 | 3000 | | 30 | 1000 |

```
struct node
{
  int data;
  struct node *next;
} temp*, head*;
```

```
if (head==NULL)
printf("List is Empty.");
  exit(0);

else
 {
  temp = head;

 {
  printf ("%d\t", temp->data);

     temp = temp -> next! = head)

     {
     printf ("%d\t"= temp-> data);

        temp=temp -> next;

     }
     printf ("%d\t"= temp->data);

          temp=temp->next;

     }
```

Refer Note 2 for 4 unit