Linked-List

We know that array requires contiguous or consecutive memory locations but this is the major drawback of an array.

Suppose we want to create an array of 100 elements of integer type.

```
int a[100];
```

Total memory requirement for array 'a' will be 100 * 2 = 200 Bytes. Here we do not need only 200 Bytes but we need 200 Bytes which are contiguous.

If compiler does not find 200 Bytes in contiguous way, allocation fails.

Now, consider a situation in which we have 200 Bytes available but they are not contiguous in nature i,e they are distributed.

In such case we cannot use "Array", we need a different data structure called "Linked-list".

Limitations of an Array

(1) Size of array cannot be changed after its declaration.

(2) Memory storage space is wasted as the memory remains allocated to the array throughout the program execution even few elements are stored.

(3) Requires contiguous memory space for execution.

(2) These limitations are overcome by using linked-list data structure.

NOTE :-> Array is known as static data structure because once memory space is allocated it cannot be extended.

Linked-list is known as dynamic data structure because memory space allocated for the elements of the list can be extended at any time.

Advantages of Linked-List

(i) Linked-lists are dynamic data structures.
(ii) Efficient memory utilization
(iii) Insertion and deletion are easier and efficient.
(iv) No unused memory.

Dis-advantages of Linked-list

(i) Cost of accessing an element is O(n).
(ii) Extra memory required for NEXT field.
(iii) Access to an arbitrary data item is little bit cumbersome and time-consuming.

Definition of Linked-list

Linked-list is a non-sequential collection of data item called Nodes.

Each node in the linked-list has two fields :

(1) Data Field (or) Info Field

(2) Next Field (or) Link Field

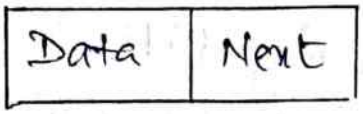


fig. Structure of a Node.

Data Field contains actual value to be stored.

Next Field contains address of the next node in list.

Types of Linked-list

(1) Singly-linked list (or) Linear linked list

(2) Doubly-linked list

(3) Circular-linked list

(4) Circular-doubly linked list

(1) Singly-linked list

- Also known as linear linked list.
- In this, all nodes are linked together in some sequential manner.

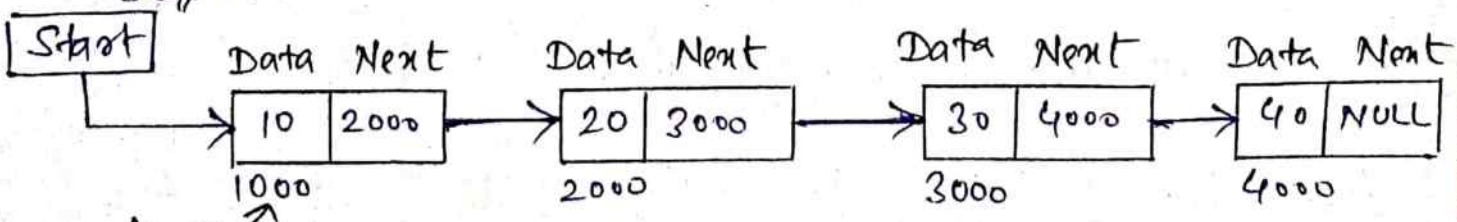


fig. Singly Linked list

(4) Start Node points to the First Node of the list.
The problem with singly-linked list is that we cannot access the predecessor of node from the current node.
This problem is solved by Doubly-linked list.

(2) Doubly Linked List

- Also known as two-way lists.
- In the case of doubly linked list, two link fields are maintained for accessing both the successor and predecessor nodes.

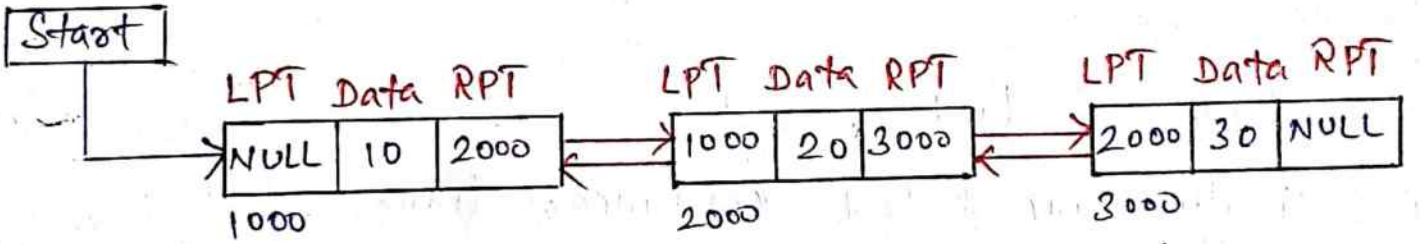


fig. Doubly-linked list

Here, LPT : Left Pointer (contains address of predecessor)
RPT : Right Pointer (contains address of successor)

Advantages of Doubly Linked List

(a) We can traverse in both directions i.e from starting to end and as well as from end to starting.

(b) It is easy to reverse the linked list.

(c) If we are at a node, we can go to any node. But in Linear linked list, it is not possible to reach

the previous node.

Dis-advantages of Doubly Linked list

(1) It requires more space per node because one extra field is required for pointer to previous node.

(2) Insertion and deletion take more time than linear list because more pointer operations are required than linear linked list.

(3) Circular Linked List

A circular linked list has no beginning and no end. If the Next field of the last node contains address of First Node, then the list is called circular linked list.

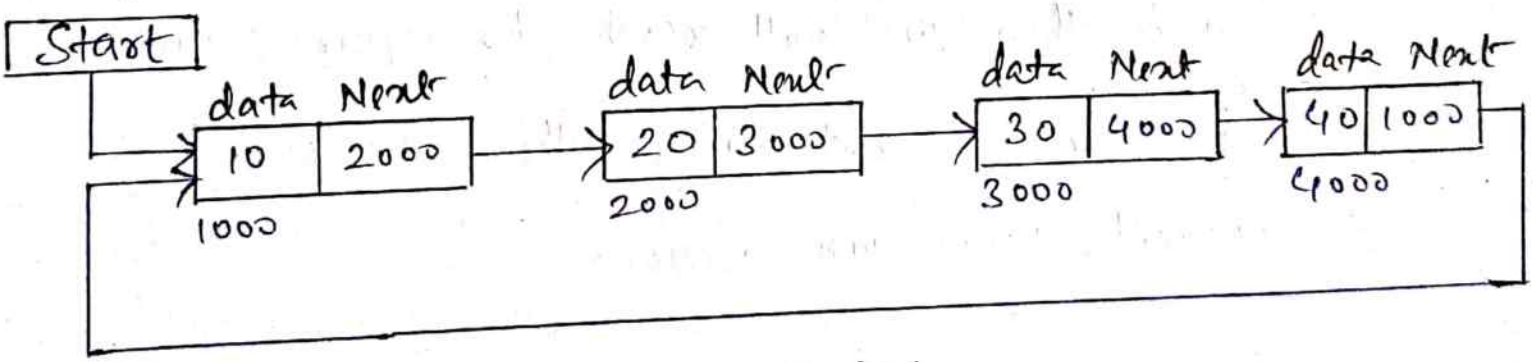


fig. Circular Linked list

Advantages of Circular Linked list

(a) If we are at a node, then we can go to any node. But in Singly linked list, it is not possible to go to previous node.

(b) It saves time when we have to go to first node from the last node. It can be done in single step because there is no need to traverse inbetween nodes.
But in doubly linked list, we will have to go through in between nodes.

Disadvantages of Circular Linked List

(a) It is not easy to reverse the linked list.

(b) If we are at a node, and want to go back to previous node, then we cannot do it in single step, Instead we have to complete the entire circle by going through the inbetween nodes and then we will reach the required node.

(c) If proper care is not taken, then the problem of infinite loop can occur.

(4) Circular Doubly Linked List

A circular doubly linked list is one which has both successor and predecessor pointer in circular manner.

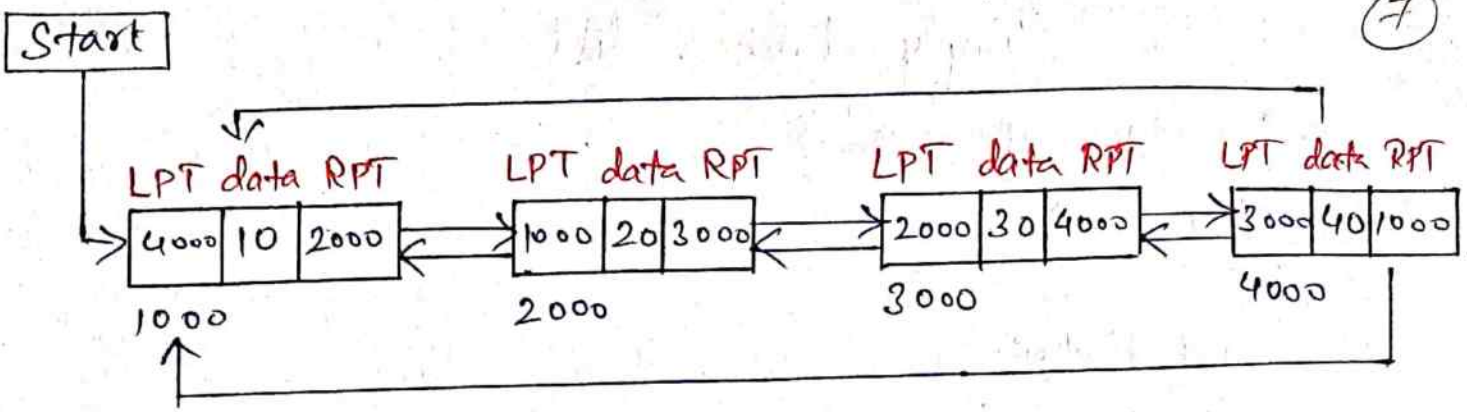


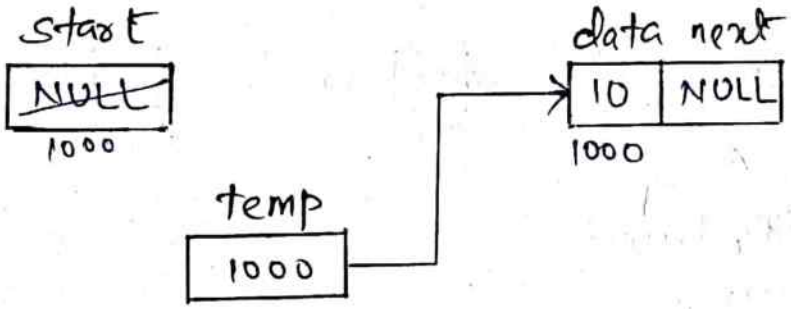
fig. Circular Doubly Linked List.

Here, LPT of first node contains the address of last node. and the RPT of last node contains the address of first node.

(1) Append

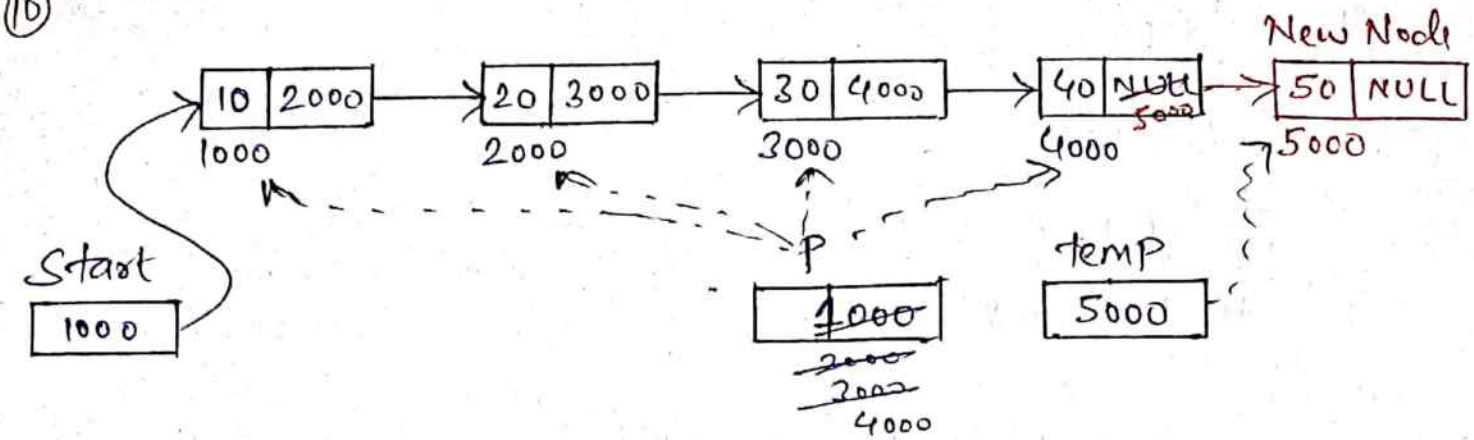
```
struct node
{
  int data;
  struct node * next;
};
struct node * start = NULL;

void append()
{
  struct node * temp;
  temp = (struct node *) malloc (sizeof (struct node));
```



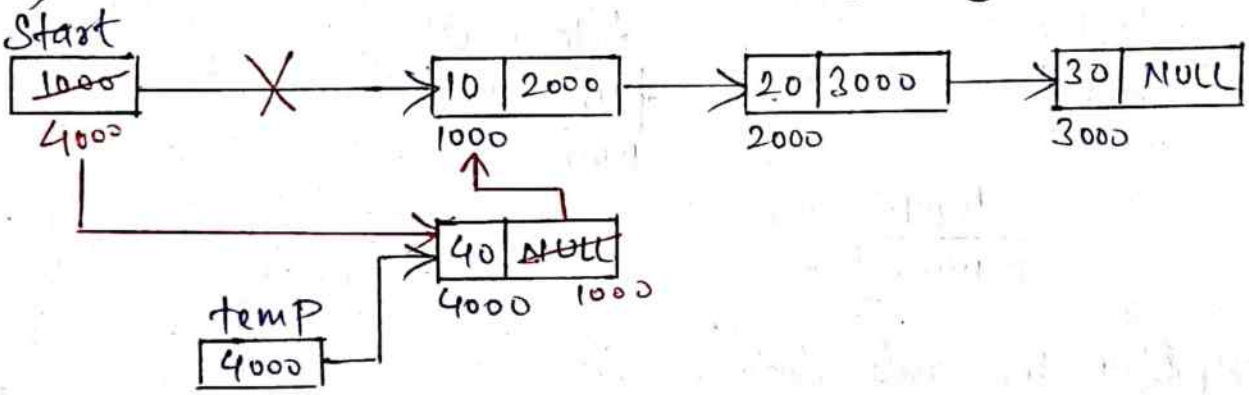

```
  printf ("Enter node data : ");
  scanf ("%d", &temp -> data);
  temp -> next = NULL;
  if (start == NULL)
    {
      start = temp;
    }
  else {
        struct node * p;
        p = start;
      while ( p -> next != NULL)
        {
```

(10)




```
            p= p→next;
        }
        p→next = temp;
    }
}
```

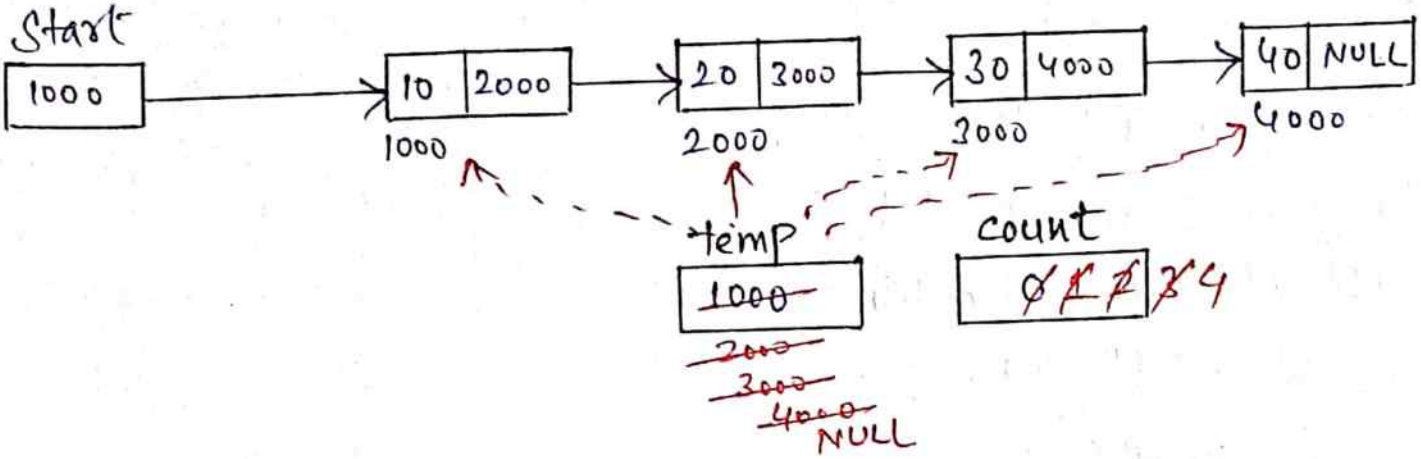
(2) How to add a node at beginning of list.




```
void addatbegin()
{
 struct node * temp ;
 temp = (struct node*) malloc (sizeof (struct node));
 printf("Enter node data : ");
 scanf("%d", &temp→data);
  temp→next = NULL;
 if (start == NULL)
   {
    start = temp;
   }
```

```
else {
    temp -> next = start ;
    start = temp ;
  }
}
```

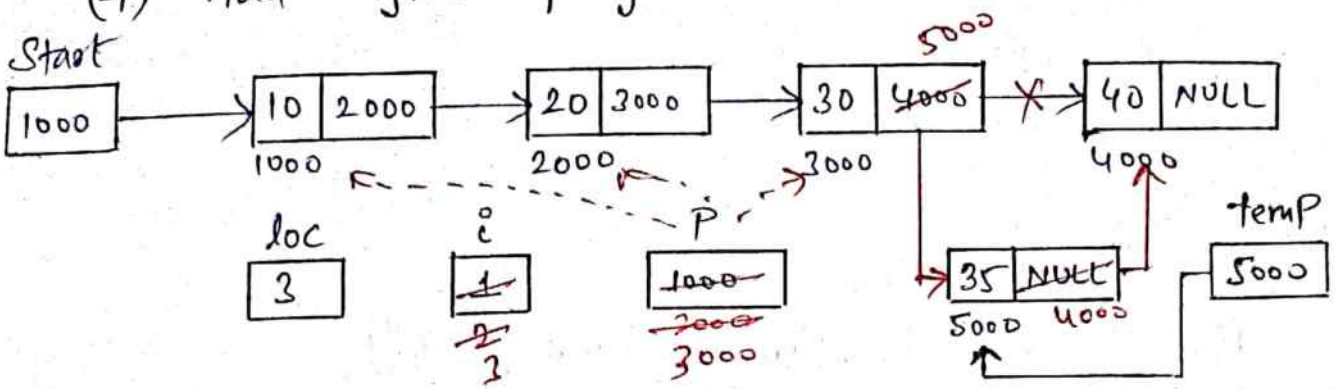
(3) Length of a Singly-Linked list




```
int length()
{
  int count =0;
  struct node * temp;
  temp = start ;
  while ( temp != NULL)
  {
     count ++ ;
     temp = temp -> next ;
  }
  return count ;
}
```

(12)

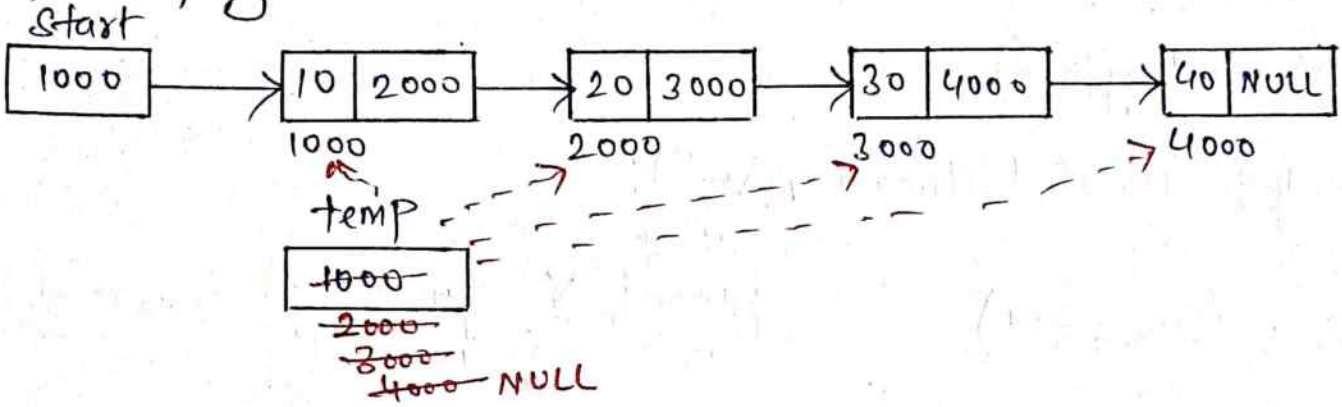
(4) Add After Specified Node.




```
void addafter()
{
  struct node * temp, *p;
  int loc, len, i=1;
  printf("Enter location: ");
  scanf("%d", &loc);
  len = length();
  if (loc > len)
    {
      printf(" Invalid location ");
      printf("Currently list is having %d nodes", len);
    }
  else {
        p = start;
      while (i < loc)
        {
          p = p->next;
          i++;
        }
temp = (struct node *) malloc (sizeof (struct node));
printf("Enter node data : ");
scanf("%d", &temp->data);
```

```
        temp -> next = NULL;
      temp -> next = p -> next ; // Right Connection
       p -> next = temp ;    // Left Connection
    }
  }
```

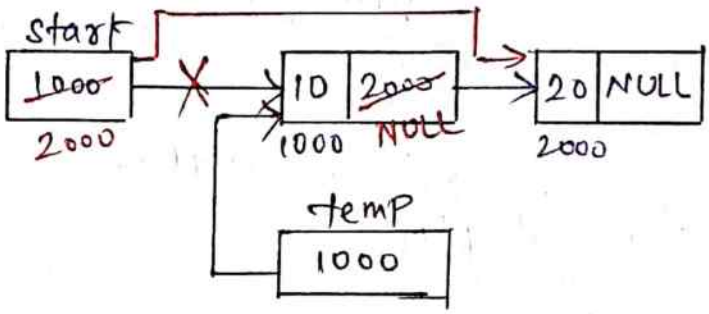
(5) Display all elements of the linked-list.

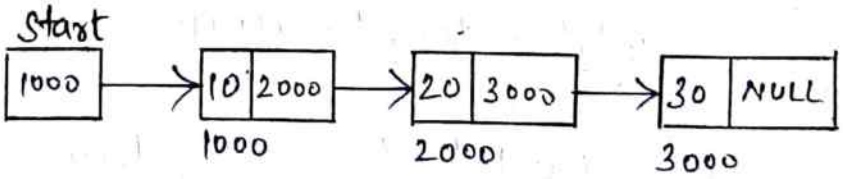


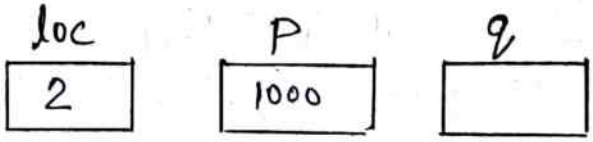

```
void display()
{
 struct node * temp;
  temp = start ;
  if ( temp == NULL)
   {
    printf (" No nodes in the list ");
   }
  else {
       while ( temp != NULL)
        {
          printf ("%d -> ", temp -> data);
          temp = temp -> next ;
        }
     }
  }
```

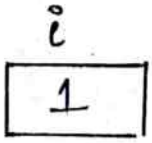
(14) Deleting a node in Singly - Linked List

```
void delete()
{
struct node * temp;
int loc ;
printf (" Enter Location to delete: ");
scanf ("%d", & loc);
if ( loc > length())
{ printf (" Invalid location \n");
}
else if ( loc == 1)
{
   temp = start;
   start = temp -> next;
   temp -> next = NULL;
   free (temp);
}
else {
   struct node * p = start , * q;
   int i = 1 ;
   while ( i < loc - 1)
   { p = p -> next;
   } i++;
   q = p -> next;
   p -> next = q -> next;
   q -> next = NULL;
   free (q);
} }
```



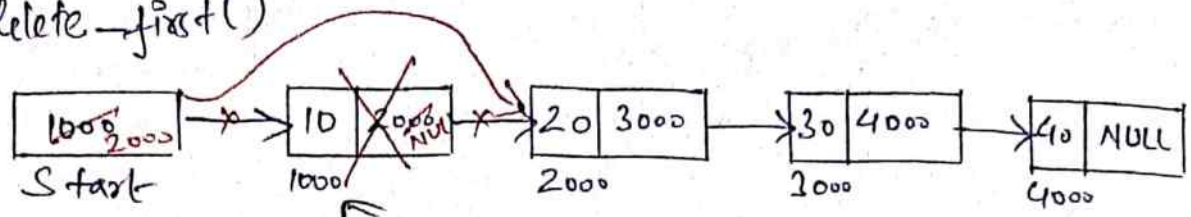






Delete First Node in Singly Linked list

```
void delete_first()
{
```

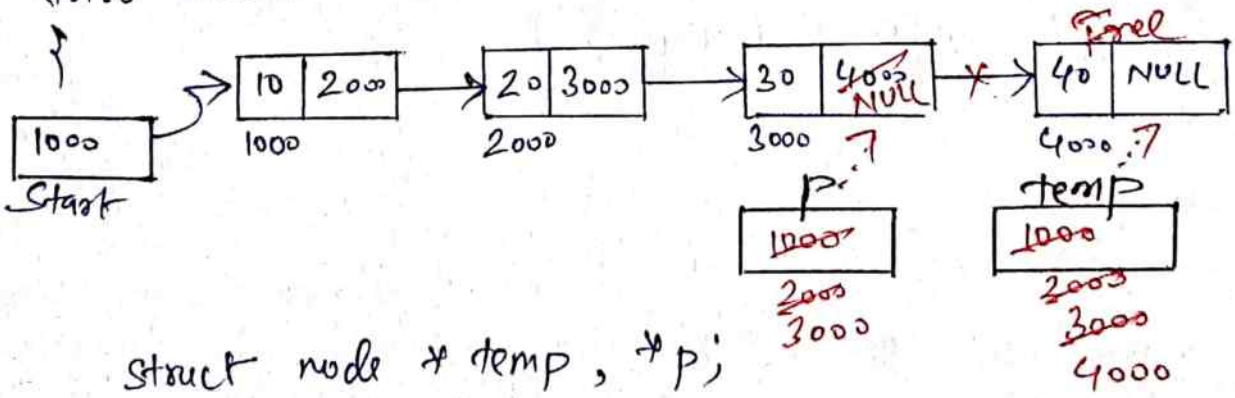



```
Struct node *temp;
temp = start;
if (temp == NULL)
   {printf("Singly Undir Flow \n");
   }
else{
    start = temp -> next;
    temp->next = NULL;
    free (temp);
  }
}
```

temp
1000

Delete Last Node in Singly linked list

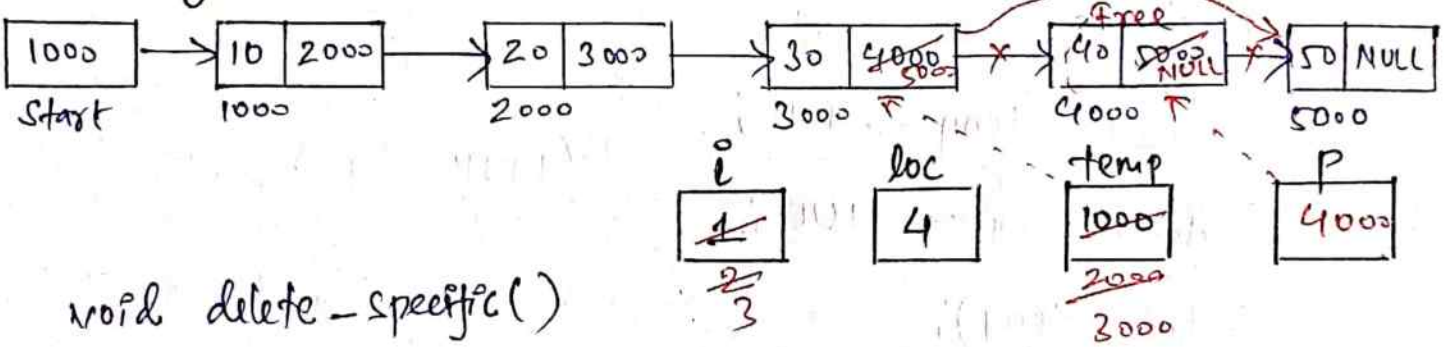
```
void delete_last()
{
```




```
Struct node * temp, *p;
    temp = start;
if (temp == NULL)
   {printf("Singly UnderFlow \n");
   }
```

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

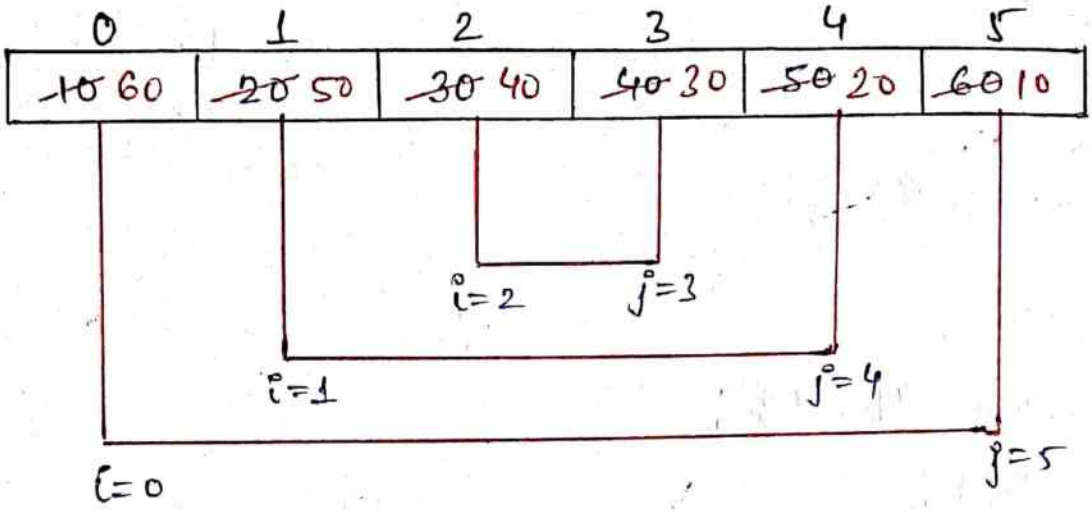
Deleting a Specific Node in Singly LinKed list




```
void delete_specific()
{
   struct node * temp, *p;
     temp = start;
   if (temp == NULL)
   {
   printf(" Singly Underflow \n");
   }
   else {
        int i = 1;
   while ( int loc;
   printf (" Enter location to delete : ");
    Scanf ("%d", &loc);
```

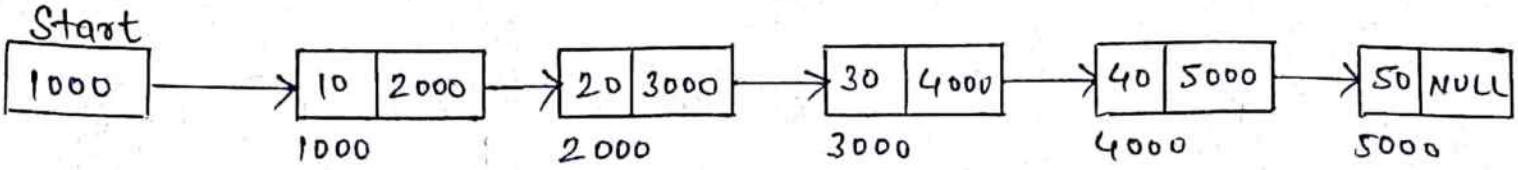
```
while ( i < loc -1)
{
    temp = temp -> next;
    i++;
}
p= temp -> next;
temp -> next = p -> next;
p -> next = NULL;
free ( p);
}
}
```

How to Reverse all the elements of an array.



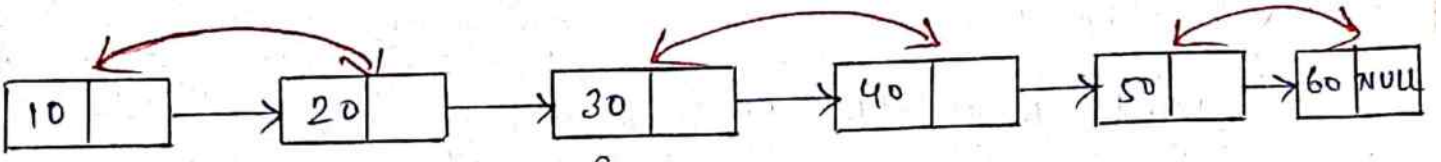

```
i=0;
j=n-1;
while (i<j)
 {
  temp = a[i];
  a[i] = a[j];
  a[j] = temp;
  i++;
  j--;
 }
```

Reverse a Singly Linked List

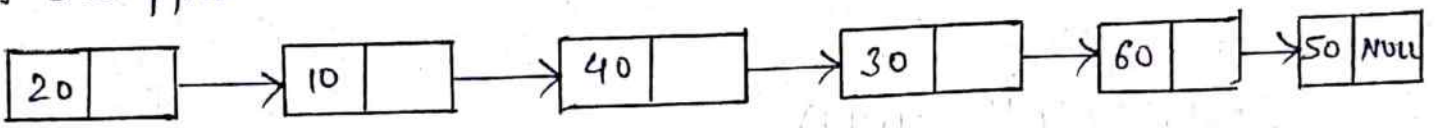



```
void reverse_list()
{
 int i, j, len ;
 struct node *p, *q ;
 len = length() ;
 i=0;
 j=n-1;
 p=q= start;
 while ( i<j)
 {
  int k=0;
  while (k < j)
    {
      q = q → next ;
      k++;
    }
  temp = p → data ;
  p→ data = q→ data;
  q→ data = temp;
    i++;
    j--;
   p= p→ next;
   q= start;
}
```

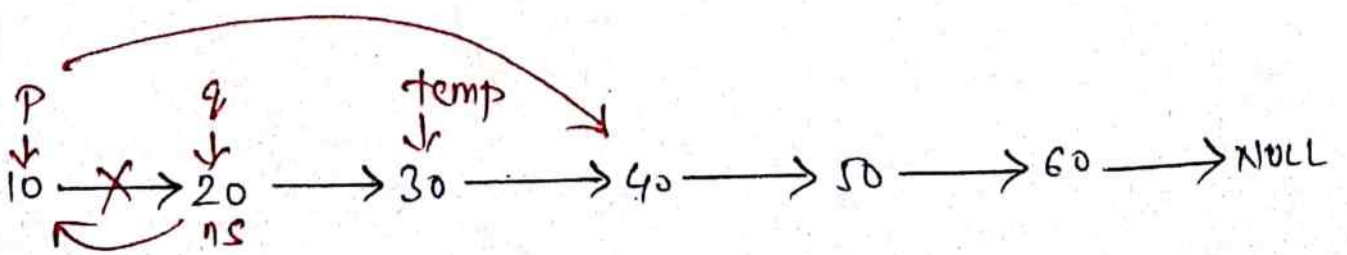
Swap pair-wise nodes in Singly Linked list



∴ Swapped linked list is




```
void swap_pairs()
{
struct node *p, *new_start;
p= start;
new_start = p->next;
while(1)
{
 q = p->next;
 temp = q->next;
 q->next = p;
 if (temp == NULL || temp->next == NULL)
  {
    p->next = NULL;
    break;
  }
 p->next = temp->next;
 p = temp;
}
}
```



Sort a Singly Linked list in C

```
void sort()
{
struct node *p, *q;
int temp;
q = start;
while (q != NULL)
{
  p = q -> next;
  while (p != NULL)
    {
      if (q -> data > p -> data)
        {
          temp = q -> data;
          q -> data = p -> data;
          p -> data = temp;
        }
      p = p -> next;
    }
    q = q -> next;
  }
}
```

Singly Linked List Program

```
#include <stdio.h>
#include <stdlib.h>
 struct node
 {
   int data;
   struct node * next;
 };
struct node * start = NULL;
 int len;
 void append (void);
 void addatbegin (void);
 void addafter (void);
 int length (void);
 void display (void);
 void delete (void);
 void main()
 {
   int ch;
   while (1)
   {
    printf ("Singly Linked List Operations: \n");
    printf ("1. Append \n");
    printf ("2. Addat begin \n");
    printf ("3. Addafter \n");
    printf ("4. Length \n");
    printf ("5. Display \n");
    printf ("6. Delete \n");
    printf ("7. Quit \n");
```

(16)

```
printf("Enter your choice : ");
scanf("%d", &ch);
switch(ch)
{
case 1 : append();
         break;
case 2 : addatbegin();
         break;
case 3 : addafter();
         break;
case 4 : len = length();
         printf("Length = %d \n", len);
         break;
case 5 : display();
         break;
case 6 : delete();
         break;
case 7 : exit(1);
default : printf("Invalid choice \n");
}
}
}
```

```
void append()
{
}
```

```
void addatbegin()
{
}
```

```
void addafter()
{
}
```

```
int length()
{
}
```

```
void display()
{
}
```

```
void delete()
{
}
```

Double Linked List

Problem with Singly-Linked list is that we cannot traverse or move in backward direction.

But in doubly linked list, either we can move in forward direction or backward direction.

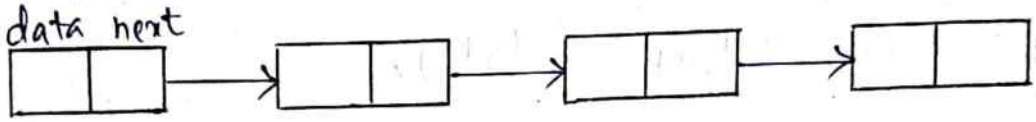


fig. Single-Linked list

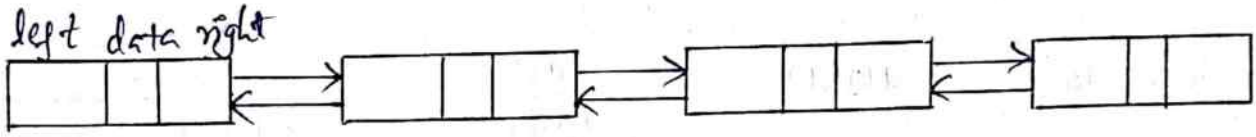
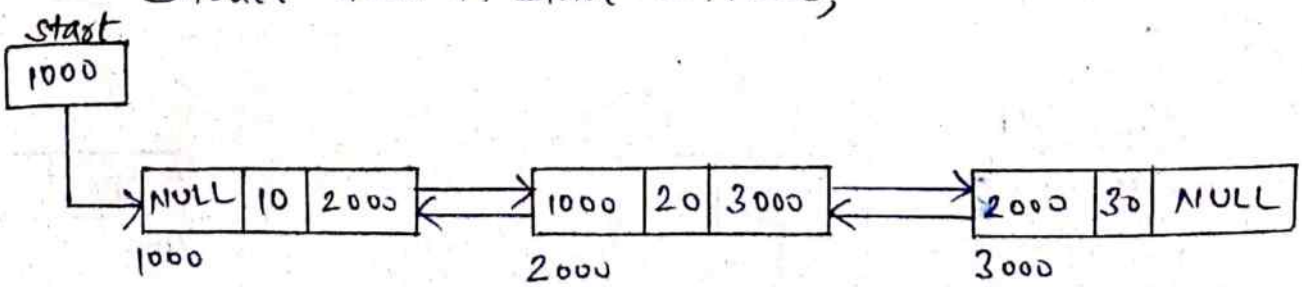


fig Double-linked list

Double Linked list :-> Node Structure

```
Struct node
{
 Struct node * left;
 int data;
 Struct node * right;
};
```

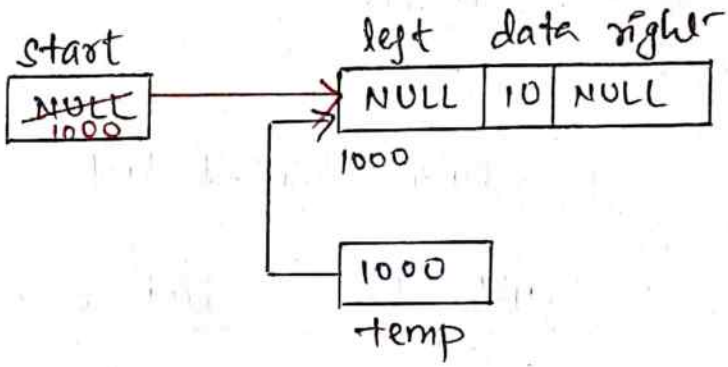

```
Struct node *. start = NULL;
```

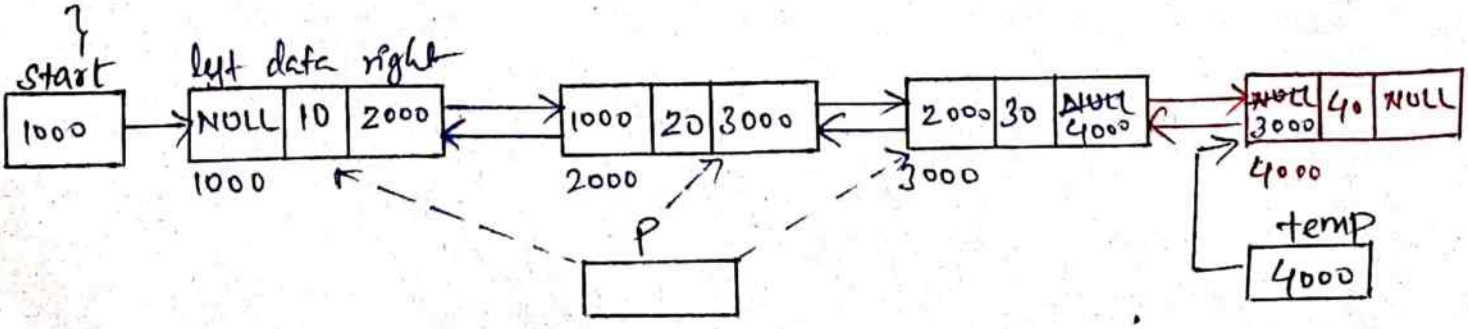


Operations on Doubly Linked list

(1) **Append** :→ means adding a node at the end

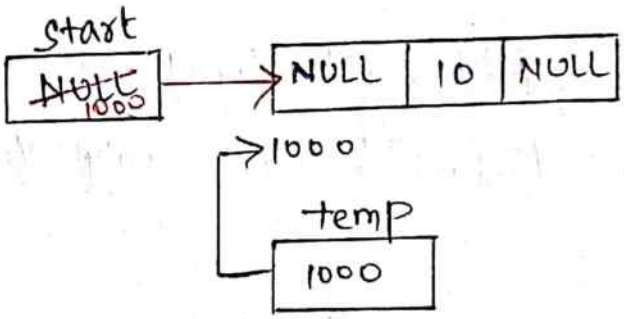
```
void append()
{
 struct node * temp;
 temp= (struct node*) malloc (sizeof(struct node));
 printf("Enter node data : ");
 scanf("%d", &temp → data);
   temp → left = NULL;
   temp → right = NULL;
if (start == NULL)
  {
   start = temp;
  }
else {
      struct node *p;
      p= start;
  while (p → right != NULL)
   {   p= p → right;
    }
    p → right = temp;
    temp → left = p;
  }
}
```

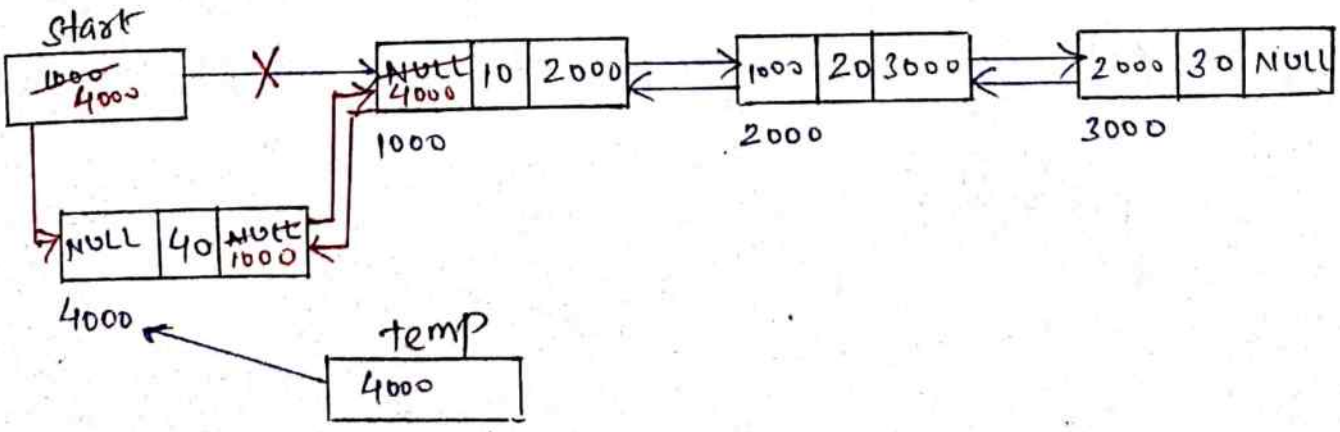




(2) Insertion at the beginning

```
void insert_beg()
{
  struct node *temp;
  temp = (struct node *) malloc (sizeof (struct node));
  printf ("Input node data : ");
  scanf ("%d", &temp -> data);
  temp -> left = NULL;
  temp -> right = NULL;
if (start == NULL)
  {
   start = temp;
  }
else
  {
    temp -> right = start;
    start -> left = temp;
    start = temp;
    printf ("New node is inserted \n");
  }
}
```

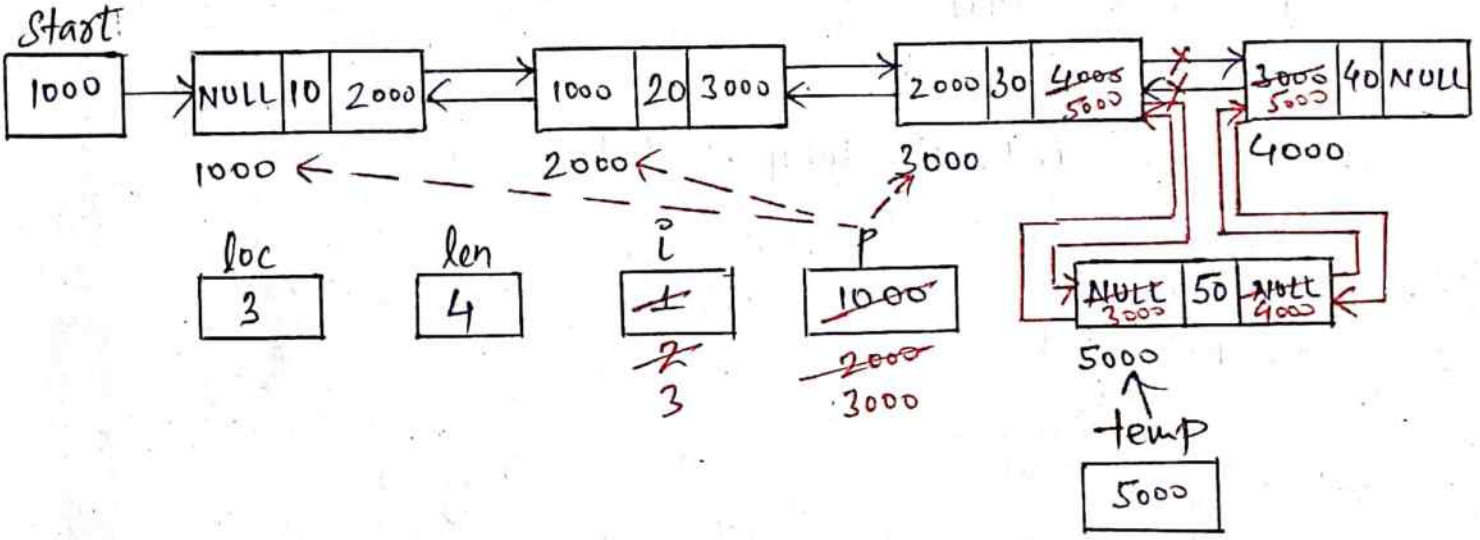




(20) Insertion After a Specified Node in Doubly Linked list.

```
void addatafter()
{
 struct node *temp, *p;
 int loc, len, i=1;
 printf("Enter location to add: ");
 scanf("%d", &loc);
 len = length();
 if (loc > len)
 {
  printf("Invalid location");
  printf("List contains only %d nodes", len);
 }
 else{
  temp = (struct node*) malloc (sizeof (struct node));
  printf("Enter node data: ");
  scanf("%d", &temp->data);
   temp->left = NULL;
   temp->right = NULL;
   p= start;
   while (i< loc)
    {
      p= p->right;
      i++;
    }
```

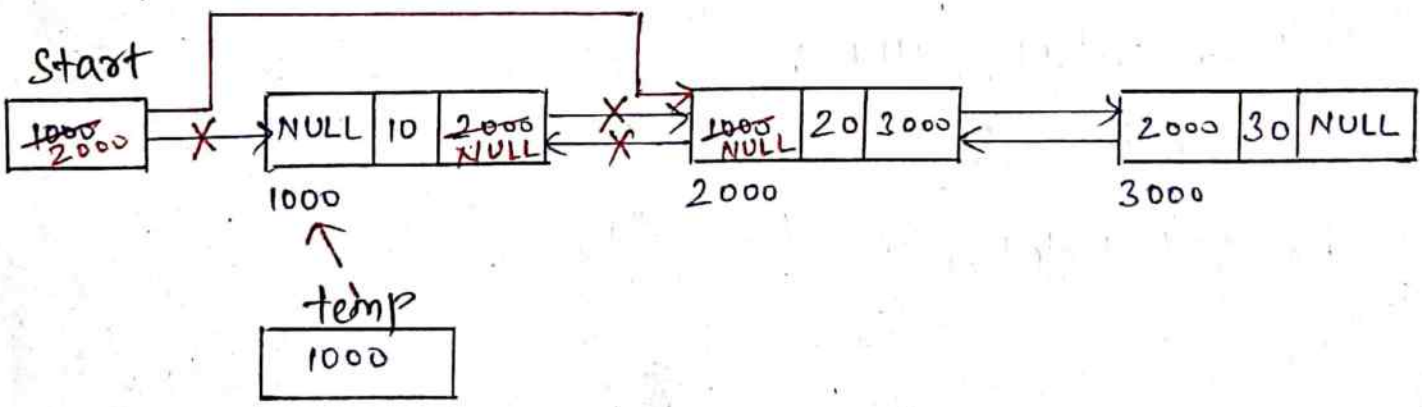
```
temp -> right = p -> right;
p -> right -> left = temp;
temp -> left = p;
p -> right = temp;
    }
}
```



Deletion in Doubly Linked List

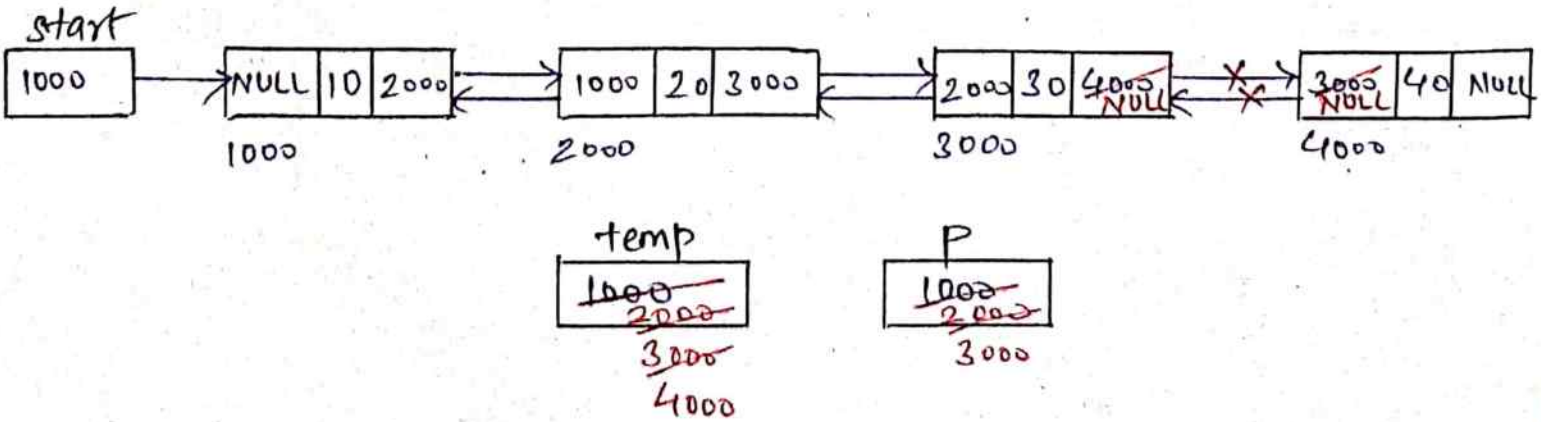
(1) Deletion From Beginning

```
void delete_beg()
{
    struct node *temp;
    temp = start;
    start = temp -> right;
    start -> left = NULL;
    temp -> right = NULL;
    free (temp);
    printf ("Deletion is done \n");
}
```



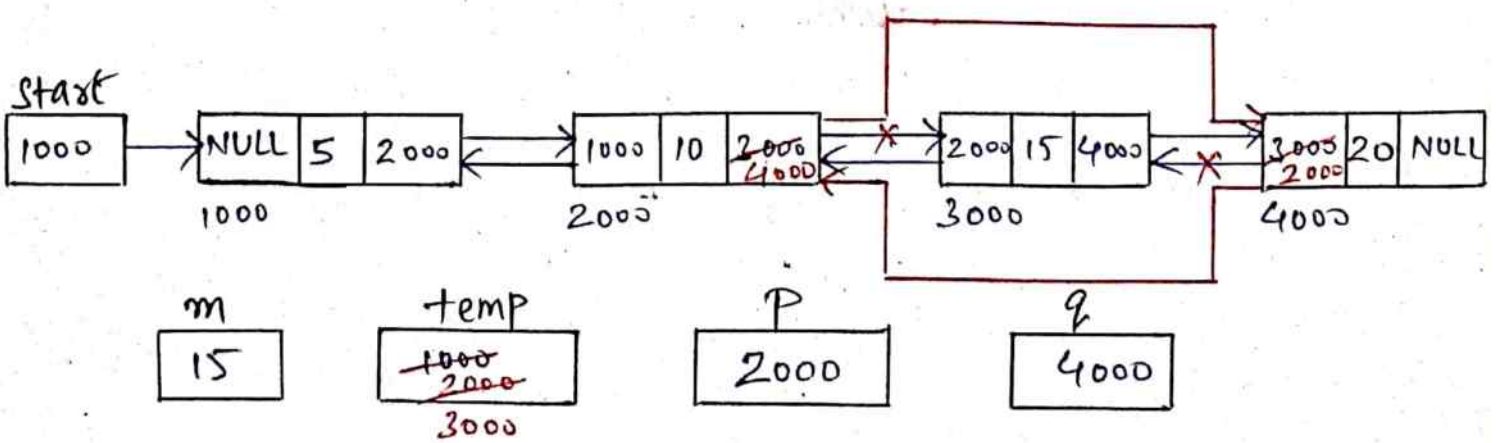
(2) Deletion From End

```
void delete_end()
{
  struct node *temp, *p;
  temp = start;
  if (temp == NULL)
  { printf ("No nodes in the list");
  }
  else{
    while ( temp -> right != NULL)
      {
        p = temp;
        temp = temp -> right;
      }
    p -> right = NULL;
    temp -> left = NULL;
    free (temp);
    printf ("Deletion is done \n");
  }
}
```



(3) Deletion of Specified Node Value

```
void delete_given_node()
{
 struct node *temp, *p, *q;
 int m;
 printf("Enter the data value of node to be deleted: ");
 scanf("%d", &m);
 temp = start;
 while (temp -> data != m)
      {
       temp = temp -> right;
      }
   p = temp -> left;
   q = temp -> right;
   p -> right = q;
   q -> left = p;
   free(temp);
   printf("Deletion is done \n");
}
```



C Program to implement a Circular Linked-list

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

struct node
{
 int data;
 struct node * next;
};
struct node * start = NULL;
void display();
void insertatbegin();
void insertatend();
void insert_spec();
int length();
void delete_begin();
void delete_end();

void main()
{
 int ch;
 clrscr();
 while(1)
 {
 printf("\n **** Circular Linked list Operations ****\n");
 printf("\n 1. Display the Circular Linked list \n");
 printf("\n 2. Insert node at beginning of list \n");
 printf("\n 3. Insert node at end of list \n");
 printf("\n 4. Insert node at specified location \n");
 printf("\n 5. Delete a node from beginning of list \n");
```

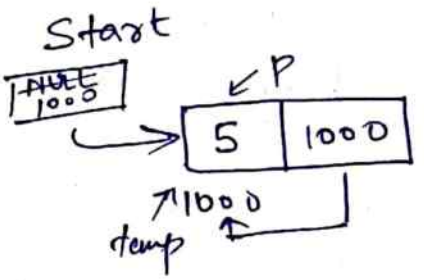
(27)

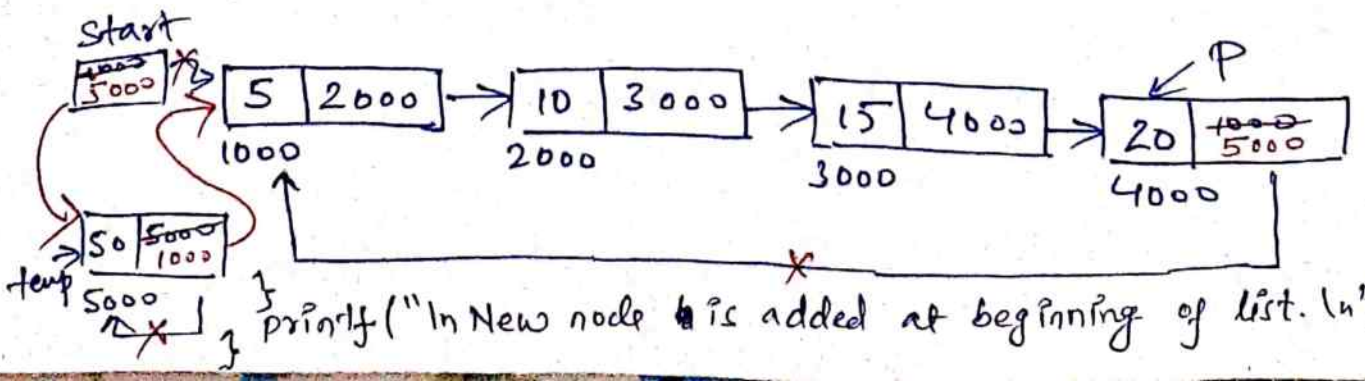
```
printf("\n 6. Delete a node from the end of list \n");
printf("\n 7. Length of the list \n");
printf("\n 8. Quit \n");
printf("\n Enter your choice: ");
scanf("%d", &ch);
switch(ch)
{
case 1: display();
        break;
case 2: insertatbegin();
        break;
case 3: insertatend();
        break;
case 4: insert_spec();
        break;
case 5: delete_begin();
        break;
case 6: delete_end();
        break;
case 7: length();
        break;
case 8: exit(1);
default: printf("\n Invalid choice... Please enter
         correct choice \n");
}
}
}
```

// Dynamic Implementation of Circular Linked-list

(1) Insertion at beginning

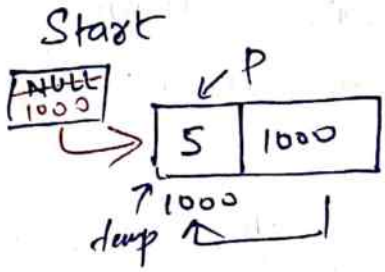
```
void insertatbegin()
{
 struct node *temp, *p;
 temp=(struct node*)malloc(sizeof(struct node));
 printf("Enter node data: ");
 scanf("%d", &temp->data);
 temp->next = temp;
 if(start == NULL)
 {
  start = temp;
  p= start;
 }
 else
 {
   p=start;
 while(p->next != start)
  {
    p=p->next;
  }
 temp->next = start;
 start =temp;
 p->next = start;
 }
 printf("\n New node is added at beginning of list.\n");
}
```

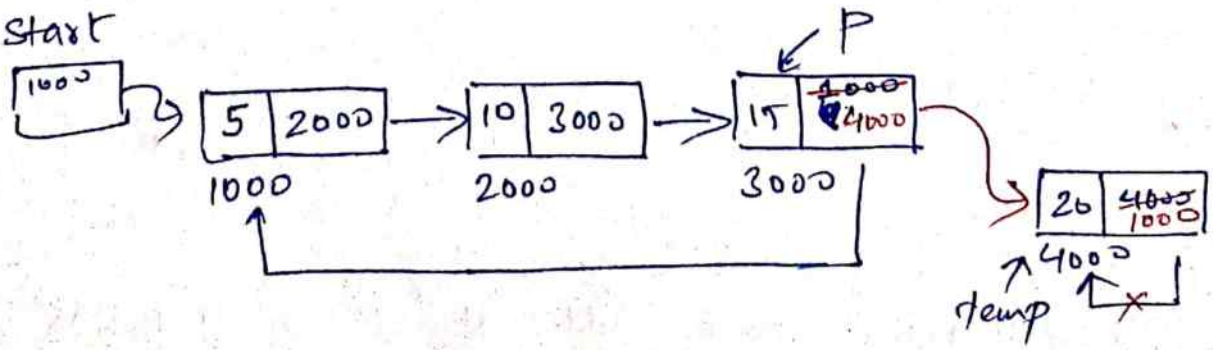




(2) Insertion at End

```
void insertatend()
{
 struct node *temp, *p;
 temp = (struct node*) malloc (sizeof (struct node));
 printf ("Enter node data: ");
 scanf ("%d", &temp->data);
 temp->next = temp;
 if (start == NULL)
 {
   start = temp;
   p= start;
 }
else
 {
  p= start;
 while (p->next != start)
 {
    p= p->next;
 }
  p->next = temp;
  temp->next = start;
 }
 printf ("\n New node is added at end of list.\n");
}
```





(3) Insertion at specified location

```
void insert_spec()
{
struct node *temp, *p;
int loc, i, len;
printf("Enter the location to add newly created
      node: ");
scanf("%d", &loc);
len = length();
if (loc > len)
 {
  printf("\n Invalid location. Please enter correct
       location \n");
  printf("\n currently the list is having %d nodes", len);
 }
else
{
  p = start;
 for (i=1; i<loc-1; i++)
  {
   p = p->next;
  }
 temp = (struct node *) malloc(sizeof(struct node));
 printf("\n Enter node data: ");
 scanf("%d", &temp->data);
 temp->next = p->next;
  p->next = temp;
 printf("\n New node is added at given location \n");
 }
}
```

(4) Length of list

```
int length()
{
  struct node *temp;
  temp= start;
  if( temp == NULL)
  {
   printf("\n No nodes in the list.\n");
  }
  else
  {
   int count =0;
  while( temp->next != start)
  {
  count ++;
  temp = temp -> next;
  }
  count++;
  printf("\n Currently the list is having %d nodes.\n", count);
  return count;
  }
 }
```

(5) Delete from beginning

```
Void delete_begin()
{
struct node *temp, *p;
temp= start;
if (temp == NULL)
{
printf("\n No nodes in the list.\n");
}
else{
if(temp->next == start)
{
printf("\n Node is deleted with data value %d",
temp-> data);
printf("\n Now list is empty.\n");
start = NULL;
free(temp);
}
else{
p=start;
while ( p->next != start)
{
p=p->next;
}
start = temp->next;
p->next = start;
free(temp);
printf("\n First node from the list is deleted\n");
}
}
}
```

(6) Delete from End

```
void delete_end()
{
struct node *temp, *p;
temp= start;
if(temp==NULL)
{
 printf("\n No nodes in the list \n");
}
else{
 if(temp->next==start)
{
 printf("\n Node is deleted with data value %d",
       temp->data);
 printf("\n Now list is empty.\n");
 start=NULL;
 free(temp);
 }
else{
  p=start;
  while(p->next!=start)
  {
    temp=p;
     p=p->next;
   }
   temp->next=start;
    free(p);
   printf("\n Last node from the list is deleted.\n");
  }
 }
}
```

(7) Display the list

```
void display()
{
 struct node *temp;
 temp = start;
if(temp == NULL)
 {
  printf("\n Linked list is empty.\n");
 }
 else
 {
  while(temp->next != start)
  {
   printf("%d -> ", temp->data);
   temp = temp->next;
  }
   printf("%d -> ", temp->data);
  }
}
```