DATA STRUCTURES

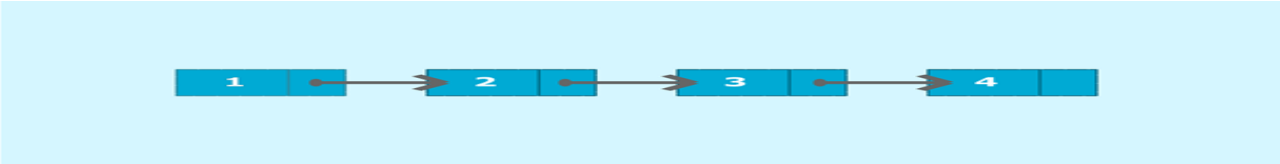
UNIT II:

Linked Lists: Singly linked lists, representation and operations, doubly linked lists and circular linked lists, Comparing arrays and linked lists, Applications of linked lists.

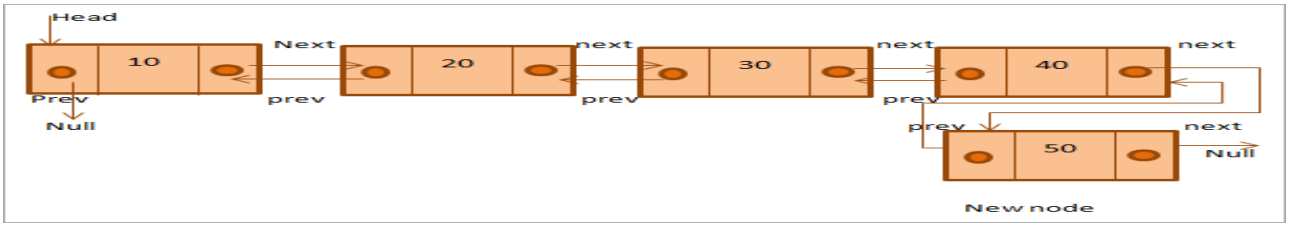
Definition: A linked list is a collection of data elements, called nodes, which the linear order is given by means of pointers. That is each node is divided into two parts: the first part contains the information of the element and the second part called the link filed or next field, contains the address of the next node in the list.

Different Types of Linked List:

1. **Singly Linked List:** Singly Linked List contains nodes which have a data part as well as an address part i.e. Next, which points to the next node in sequence of nodes. The operations we can perform on singly linked list are insertion, deletion and traversal.

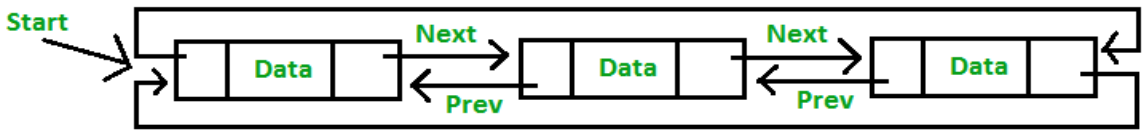


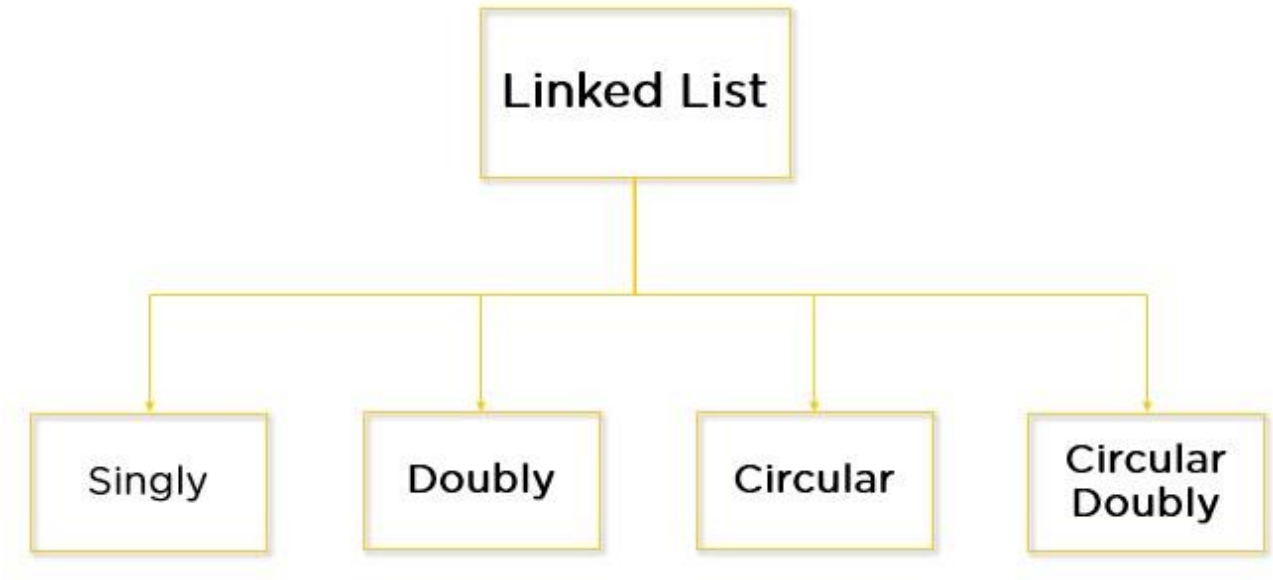
2. Doubly Linked List: In Doubly Linked List, each node contains two links the first link pointer to the previous node and next link pointer to the next node in the sequence.



3. Circular Linked List (or) Singly Circular Linked List: In Circular Linked List, the last node of the list contains the address of the first node and forms a circular chain.

4. Doubly Circular Linked List: In Doubly circular Linked List, the last node of the list contains the address of the first node and first node contains the address of last node and forms a circular chain.





Representation of Singly Linked List in the memory: There are two ways to represent singly linked list in the memory.

1. Static representation using Array
2. Dynamic representation using free pool of storage

1. Static representation using Array:

In static representation of a single linked list, two arrays are maintained: one array for data and the other for links. The static representation of the linked list in the below Figure-1 is shown in below Figure-2.

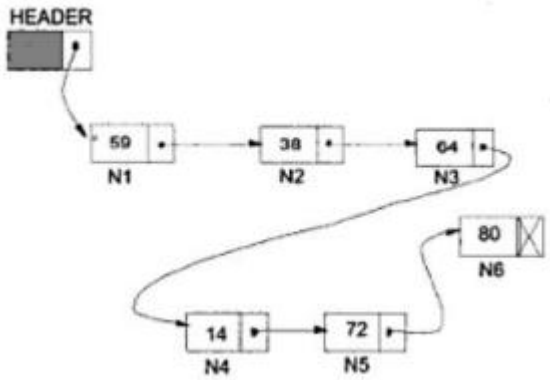


Figure-1: Singly Linked List with six nodes.

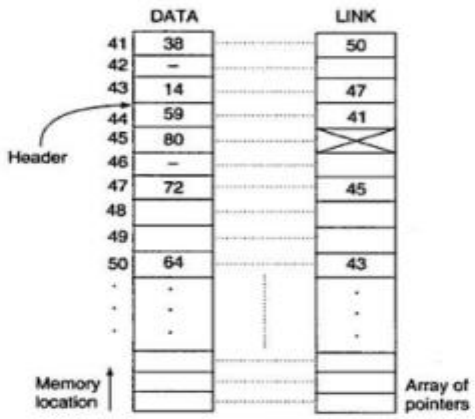


Figure-2: Static representation using arrays of the single linked list of Figure-1.

Two parallel arrays of equal size are allocated which should be sufficient to store the entire linked list. Nevertheless, this contradicts the idea of the linked list (that is non-contagious location of elements).

2. Dynamic representation using free pool of storage

The efficient way of representing a linked list is using the free pool of storage. In this method, there is a memory bank (which "is nothing but a collection of free memory spaces) and a memory manager (a program, in fact). During the creation of a linked list, whenever a node is required, the request is placed to the memory manager; the memory manager will then search the memory bank for the block requested and, if found, grants the desired block to the caller. Again, there is also another program called the garbage collector; it plays whenever a node is no more in use; it returns the unused node to the memory bank. It may be noted that memory bank is basically a list of memory' spaces which is available to a programmer. Such a memory management is known as dynamic memory management. The dynamic representation of linked list uses the dynamic memory management policy.

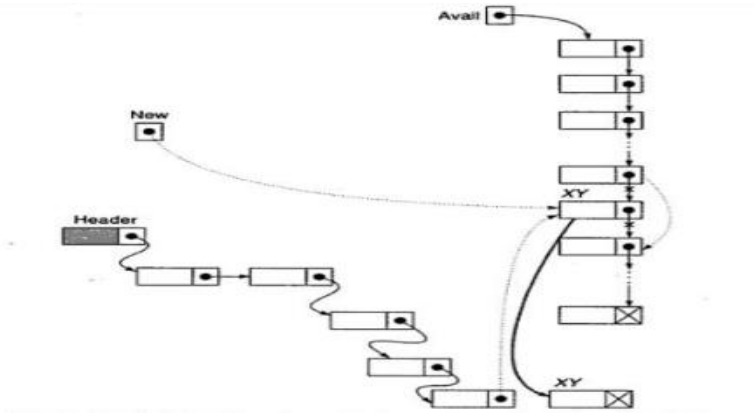


Figure-3: Allocation of a node from memory bank to a linked list.

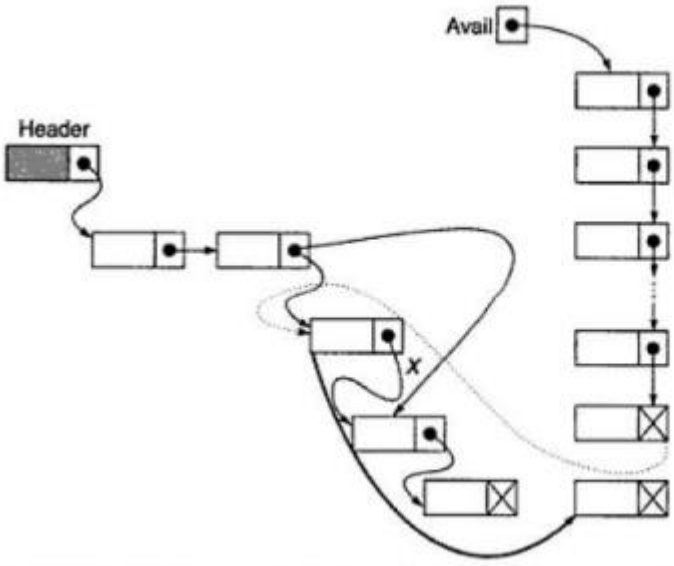


Figure-4: Returning a node from a linked list to memory bank.

Operations on Singly Linked List:

All operations performed in the list using arrays same operations can be performed on the linked list. Now linked list is created using dynamic memory allocation. The main advantage of this kind of implementation is that we can create a list of nodes as per our needs and requirements Hence there won't be any wastage or lack of memory. Various primary operations of linked list are:

- **Creation:** This operation is performed for Creation of a Singly Linked List
- **Display:** This operation is performed to traversal of a Singly Linked List
- **Insertion:** This operation is performed to add an element into a list
- **Deletion:** This operation is performed to delete an element from a Singly Linked List
- **Search:** This operation is performed to search an element from the list using the given key.

Algorithm for creation of Singly Linked List:

Algorithm has three parts (a) Declaration (b) initial Condition (c) Steps for Algorithm

(a) Declaration

```
struct node {

int data;

struct node *next;

} *head, *current, *temp\
```

(b) initial Condition

head=NULL, temp=NULL, current=NULL;

(c) Steps for Algorithm

Begin

Step 1: Read the element into x

Step 2: Create a temp node in the memorytemp = (struct node) sizeof (node)

Step 3: Assign the values in temp node as follows

```
temp -> data = x
temp ->next=NULL
```

Step 4: check whether head is NULL or not

```
if (head=NULL)
 {
 head=temp current=temp
 }
else
 {
  current -> next =temp;
  current = current ->next
 }
```

Step 5: follow step 1 to 4 to insert remaining element in the list.

Step 6: End.

Write an algorithm to traverse or print elements of a single linkedlist

```
void display ()
Begin
        current=head
        while (current! = NULL)
        {
        Print    "current -> info"
        current =current ->next
        }
End
```

Write an algorithm to search an element in single linked list.

Let x be the element to search

```
void SEARCH(x)

Begin
        found =0
        current =head
        while (current != NULL)
          {
                if(current ->data=x)

                {

                found=1;
                break;
                }
                current=current->next
        }

                if(found=1)     print "Element found"
                  else          print "Not found"
End.
```

Write a Code for Implmenetation of Singly Linked List and its operations

```
#include<stdio.h>
#include<alloc.h>
struct node
{
int data;
struct node *next;
};
struct node *newnode,*first,*last,*i,*p,*p1;
int n;
void main()
{
int ch;
void create();
void insert();
void del();
void display();
do
{
printf("\n1.Creation of Singly Linked List\n");
printf("\n2.Inserting a New node in SLL\n");
printf("\n3.Deleting a Node from SLL\n");
printf("\n4.Displaying SLL\n");
printf("\n5.Exit\n");
printf("\nEnter Your Choice\n");
scanf("%d",&ch);
switch(ch)
{
case 1: create(); break;
case 2: insert(); break;
case 3: del(); break;
case 4: display(); break;
case 5: exit(0);
}
}while(ch!=5);
}
void create()
{
char ch;
int j;
printf("How Many Node:\n");
scanf("%d",&n);
for(j=0;j<n;j++)
{
newnode=(struct node*)malloc(sizeof(struct node));
```

```
printf("\nEnter Value\n");
scanf("%d",&newnode->data);
newnode->next=NULL;
if(first==NULL)
{
first=newnode;
last=newnode;
}
else
{
last->next=newnode;
last=newnode;
}
}
}
void display()
{
if(n<=0) printf("Empty Linked List\n");
else
{
printf("\nLinked List\n");
for(i=first;i->next!=NULL;i=i->next)
printf("%d->",i->data);
printf("%d",i->data);
}
}
void insert()
{
int c,v;
printf("\n1.Front  2.Middle  3.Last\n");
printf("\nYour Choice\n");
scanf("%d",&c);
switch(c)
{
case 1:
newnode=(struct node*)malloc(sizeof(struct node));
printf("\nEnter Data Value\n");
scanf("%d",&newnode->data);
newnode->next=NULL;
newnode->next=first;
first=newnode;
break;
case 2:
newnode=(struct node*)malloc(sizeof(struct node));
printf("\nEnter Data Value\n");
```

```
scanf("%d",&newnode->data);
newnode->next=NULL;
printf("\nEnter after which element\n");
scanf("%d",&v);
p1=p=first;
while((p->data<=v)&&(p!=NULL))
{
p1=p;
p=p->next;
}
if(p!=NULL)
{
newnode->next=p1->next;
p1->next=newnode;
}
else
{
printf("element not found\n");
n--;
}
break;
case 3:
newnode=(struct node*)malloc(sizeof(struct node));
printf("\nEnter Data Value\n");
scanf("%d",&newnode->data);
newnode->next=NULL;
last->next=newnode;
last=newnode;
break;
}
n++;
}
void del()
{
int c,v,flag=0;
printf("\n1.Front  2.Middle  3.Last\n");
printf("\nYour Choice\n");
scanf("%d",&c);
switch(c)
{
case 1:
p=first;
first=first->next;
p->next=NULL;
free(p);
```

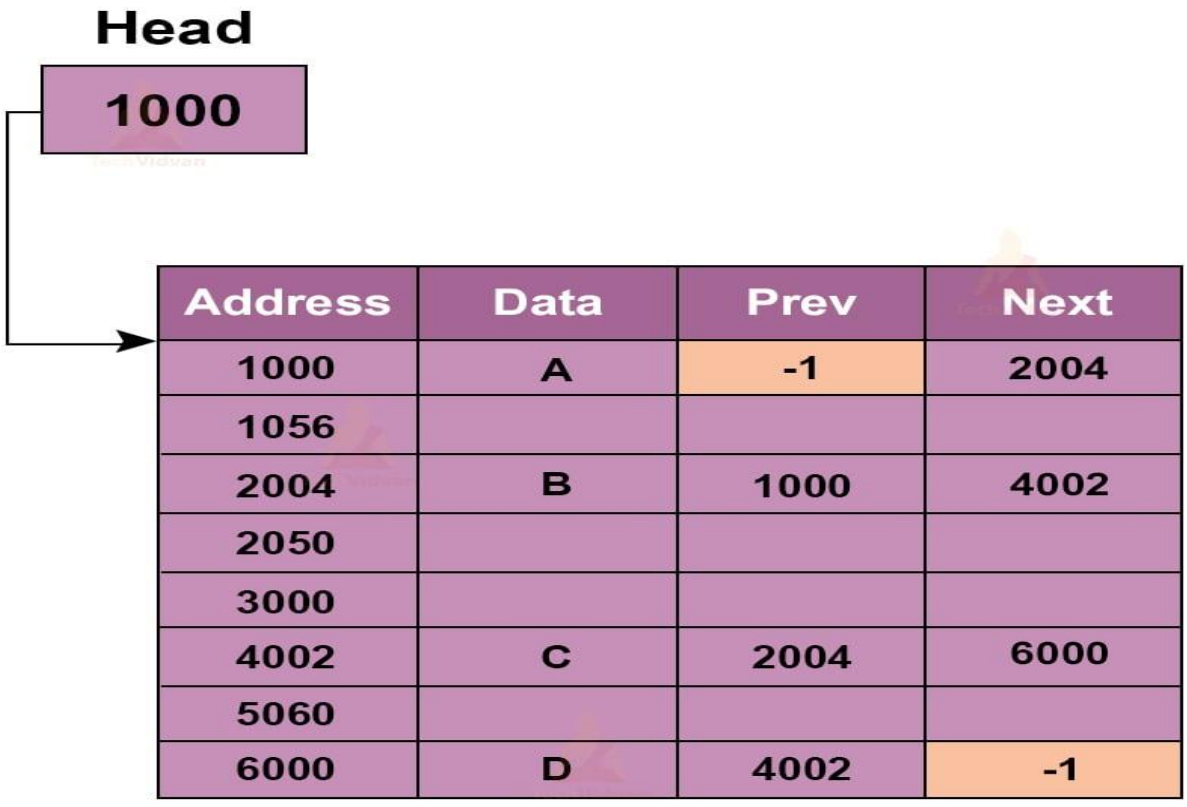
```
break;
case 2:
printf("\nEnter Element to Delete\n");
scanf("%d",&v);
p=p1=first;
while(p->data!=v)
{
if(p->next!=NULL)
{
p1=p;
p=p->next;
}
else
{
flag=1;
break;
}
}
if(flag!=1)
p1->next=p->next;
else
{
printf("\nNo Element\n");
n++;
}
free(p);
break;
case 3:
p=p1=first;
while(p->next!=NULL)
{
p1=p;
p=p->next;
}
p1->next=NULL;
last=p1;
free(p1);
break;
}
n--;
}
```

Memory Representation of a doubly linked list

- Memory Representation of a doubly linked list is shown in Figure-5. Generally, doubly linked list consumes more space for every node and therefore, causes more expansive basic operations such as insertion and deletion. However, we can easily manipulate the elements of the list since the list maintains pointers in both the directions (forward and backward).

- In Figure-5, the first element of the list that is i.e. A stored at address 1000. The head pointer points to the starting address 1000. Since this is the first element being added to the list therefore the prev of the list contains NULL (-1). The next node of the list resides at address 2004 therefore the first node contains 2004 in its next pointer.

- We can traverse the list in this way until we find any node containing null or -1 in its next part.



Address	Data	Prev	Next
1000	A	-1	2004
1056			
2004	B	1000	4002
2050			
3000			
4002	C	2004	6000
5060			
6000	D	4002	-1

Figure-5: Memory representation of Doubly linked list.

Write a Code for Implmenetation of Doubly Linked List and its operations

```
#include<alloc.h>
#include<stdio.h>
struct node
{
struct node *prev;
int data;
struct node *next;
};
struct node *first,*last,*newnode,*curr,*i,*x,*y,*p,*k;
void create()
{
int n1,j;
printf("How many nodes do you want to create:\n");
scanf("%d",&n1);
for(j=0;j<n1;j++)
{
newnode=(struct node *)malloc(sizeof(struct node));
scanf("%d",&newnode->data);
newnode->prev=NULL;
newnode->next=NULL;
if(first==NULL)
{
first=newnode;
last=newnode;
curr=newnode;
}
else
{
curr->next=newnode;
newnode->prev=last;
last=newnode;
curr=newnode;
}
}
}
void dinsertf()
{
printf("Enter A Number To Insert At First Node:");
newnode=(struct node*)malloc(sizeof(struct node));
scanf("%d",&newnode->data);
newnode->prev=NULL;
newnode->next=NULL;
newnode->next=first;
```

```
first->prev=newnode;
first=newnode;
}
void dinsertm()
{
int pos,b;
printf("Enter A Number To Insert At Middle Node:");
newnode=(struct node*)malloc(sizeof(struct node));
scanf("%d",&newnode->data);
newnode->prev=NULL;
newnode->next=NULL;
printf("Enter Position:");
scanf("%d",&pos);
b=1;
x=first;
while(b<pos)
{
x=x->next;
b++;
}
newnode->prev=x;
newnode->next=x->next;
(x->next)->prev=newnode;
x->next=newnode;
}
void dinsertl()
{
printf("Enter A Number To Insert At Last Node:");
newnode=(struct node*)malloc(sizeof(struct node));
scanf("%d",&newnode->data);
last->next=newnode;
newnode->prev=last;
last=newnode;
}
void ddeletef()
{
int x;
x=first->data;
p=first;
first=first->next;
p->next=NULL;
free(p);
printf("The Deleted Node Is %d\n",x);
}
```

```
void ddeletem()
{
int pos,i=1;
printf("Enter Position Number To Delete:");
scanf("%d",&pos);
x=first;
while(i<pos)
{
y=x;
x=x->next;
i++;
}
y->next=x->next;
(x->next)->prev=(y->next)->prev;
free(x);
printf("The Deleted Node Is %d\n",x->data);
}
void ddeletel()
{
x=y=first;
while(x->next!=NULL)
{
y=x;
x=x->next;
}
y->next=NULL;
x->prev=NULL;
free(x);
printf("The Deleted Node Is %d\n",x->data);
}
void display()
{
for(i=first;i!=NULL;i=i->next)
{
printf("%d->",i->data);
}
}
void main()
{
int n;
void create();
void insert();
void del();
void display();
```

```
do
{
printf("\n1. Create a Doubly Linked List:\n");
printf("2. Insert a node into Doubly linked list:\n");
printf("3. Delete a node from Doubly linked list:\n");
printf("4. Display DLL\n");
printf("5. Exit\n");
printf("Enter your choice:\n");
scanf("%d",&n);
switch(n)
{
case 1: create(); break;
case 2: insert(); break;
case 3: del(); break;
case 4: display(); break;
case 5: exit(0);
}
}while(n!=5);
}
void insert()
{
int n2;
do
{
printf("1. Insert at front node:\n");
printf("2. Insert at middle node:\n");
printf("3. Insert at Last node:\n");
printf("4. Exit\n");
printf("Enter your choice:\n");
scanf("%d",&n2);
switch(n2)
{
case 1: dinsertf();
break;
case 2: dinsertm();
break;
case 3: dinsertl();
break;
case 4: return;
}
}while(n2!=4);
}
```

```
void del()
{
int n3;
do
{
printf("1. Delete at front node:\n");
printf("2. Delete at middle node:\n");
printf("3. Delete at Last node:\n");
printf("4. Exit\n");
printf("Enter your choice:\n");
scanf("%d",&n3);
switch(n3)
{
case 1: ddeletef();
break;
case 2: ddeletem();
break;
case 3: ddeletel();
break;
case 4: return;
}
} while(n3!=4);
}
```

Write a Code for Implmenetation of Circular Linked List and its operations

```
#include<stdio.h>
#include<alloc.h>
struct node
{
int data;
struct node *next;
};
struct node *newnode,*first,*last,*i,*p,*p1;
int n;
void main()
{
int ch;
void create();
void insert();
void del();
void display();
do
{
printf("\n1.Creation of Cicular Linked List\n");
printf("\n2.Inserting a New node in CLL\n");
printf("\n3.Deleting a Node from CLL\n");
printf("\n4.Displaying CLL\n");
printf("\n5.Exit\n");
printf("\nEnter Your Choice\n");
scanf("%d",&ch);
switch(ch)
{
case 1: create(); break;
case 2: insert(); break;
case 3: del(); break;
case 4: display(); break;
case 5: exit(0);
}
}while(ch!=5);
}
void create()
{
char ch;
int j;
printf("How Many Node:\n");
scanf("%d",&n);
for(j=0;j<n;j++)
{
```

```
newnode=(struct node*)malloc(sizeof(struct node));
printf("\nEnter Value\n");
scanf("%d",&newnode->data);
newnode->next=NULL;
if(first==NULL)
{
first=newnode;
last=newnode;
first->next=first;
}
else
{
last->next=newnode;
last=newnode;
last->next=first;
}
}
}
void display()
{
if(n<=0) printf("Empty Linked List\n");
else
{
printf("\nLinked List\n");
for(i=first;i->next!=first;i=i->next)
printf("%d->",i->data);
printf("%d",i->data);
}
}
void insert()
{
int c,v;
printf("\n1.Front  2.Middle  3.Last\n");
printf("\nYour Choice\n");
scanf("%d",&c);
switch(c)
{
case 1:
newnode=(struct node*)malloc(sizeof(struct node));
printf("\nEnter Data Value\n");
scanf("%d",&newnode->data);
newnode->next=NULL;
newnode->next=first;
first=newnode;
last->next=first;
```

```
break;
case 2:
newnode=(struct node*)malloc(sizeof(struct node));
printf("\nEnter Data Value\n");
scanf("%d",&newnode->data);
newnode->next=NULL;
printf("\nEnter after which element\n");
scanf("%d",&v);
p1=p=first;
while((p->data<=v)&&(p!=first))
{
p1=p;
p=p->next;
}
if(p!=NULL)
{
newnode->next=p1->next;
p1->next=newnode;
}
else
{
printf("element not found\n");
n--;
}
break;
case 3:
newnode=(struct node*)malloc(sizeof(struct node));
printf("\nEnter Data Value\n");
scanf("%d",&newnode->data);
newnode->next=NULL;
last->next=newnode;
last=newnode;
last->next=first;
break;
}
n++;
}
void del()
{
int c,v,flag=0;
printf("\n1.Front  2.Middle  3.Last\n");
printf("\nYour Choice\n");
scanf("%d",&c);
switch(c)
{
```

```
case 1:
p=first;
first=first->next;
p->next=NULL;
free(p);
last->next=first;
break;
case 2:
printf("\nEnter Element to Delete\n");
scanf("%d",&v);
p=p1=first;
while(p->data!=v)
{
if(p->next!=first)
{
p1=p;
p=p->next;
}
else
{
flag=1;
break;
}
}
if(flag!=1)
p1->next=p->next;
else
printf("\nNo Element\n");
free(p);
break;
case 3:
p=p1=first;
while(p->next!=first)
{
p1=p;
p=p->next;
}
p1->next=NULL;
last=p1;
last->next=first;
break;
}
n--;
}
```

Why do we choose linked list over Arrays?

Need of choosing Linked List over Arrays

- Linked Lists are used when the number of elements is not known in advance i.e. size is not known as linked lists support dynamic memory allocation.
- Linked lists are simple and can be used to implement other data structures like stack, queue, and tree.
- Linked Lists can be used for the manipulation of polynomials.
- Linked lists are used for performing arithmetic operations on long integers.
- Linked List can be used in cases when faster insertion and deletion are required.

Arrays Vs Linked List

SNO	Array	Linked list
1	An array is a collection of elements of a similar data type.	A linked list is a collection of objects known as a node where node consists of two parts, i.e., data and address.
2	Array elements store in a contiguous memory location.	Linked list elements can be stored anywhere in the memory or randomly stored.
3	Array works with a static memory. Here static memory means that the memory size is fixed and cannot be changed at the run time.	The Linked list works with dynamic memory. Here, dynamic memory means that the memory size can be changed at the run time according to our requirements.
4	Array elements are independent of each other.	Linked list elements are dependent on each other. As each node contains the address of the next node so to access the next node, we need to access its previous node.
5	Array takes more time while performing any operation like insertion, deletion, etc.	Linked list takes less time while performing any operation like insertion, deletion, etc.
6	Accessing any element in an array is faster as the element in an array can be directly accessed through the index.	Accessing an element in a linked list is slower as it starts traversing from the first element of the linked list.

7	In the case of an array, memory is allocated at compile-time.	In the case of a linked list, memory is allocated at run time.
8	Memory utilization is inefficient in the array. For example, if the size of the array is 6, and array consists of 3 elements only then the rest of the space will be unused.	Memory utilization is efficient in the case of a linked list as the memory can be allocated or deallocated at the run time according to our requirement.

Applications of Linked List

- Linked Lists are used to implement stacks and queues.
- It is used for the various representations of trees and graphs.
- It is used in dynamic memory allocation(linked list of free blocks).
- It is used for representing sparse matrices.
- It is used for the manipulation of polynomials.
- It is also used for performing arithmetic operations on long integers.
- It is used for finding paths in networks.
- In operating systems, they can be used in Memory management, process scheduling and file system.
- Linked lists can be used to improve the performance of algorithms that need to frequently insert or delete items from large collections of data.
- Implementing algorithms such as the LRU cache, which uses a linked list to keep track of the most recently used items in a cache.

Applications of Linked List in real world

- The list of songs in the music player are linked to the previous and next songs.
- In a web browser, previous and next web page URLs are linked through the previous and next buttons.
- In image viewer, the previous and next images are linked with the help of the previous and next buttons.
- Switching between two applications is carried out by using “alt+tab” in windows and “cmd+tab” in mac book. It requires the functionality of circular linked list.
- In mobile phones, we save the contacts of the people. The newly entered contact details will be placed at the correct alphabetical order. This can be achieved by linked list to set contact at correct alphabetical position.
- The modifications that we make in documents are actually created as nodes in doubly linked list. We can simply use the undo option by pressing Ctrl+Z to modify the contents. It is done by the functionality of linked list.

Advantages of Linked Lists:

Linked lists are a popular data structure in computer science, and have a number of advantages over other data structures, such as arrays. Some of the key advantages of linked lists are:

- **Dynamic size:** Linked lists do not have a fixed size, so you can add or remove elements as needed, without having to worry about the size of the list. This makes linked lists a great choice when you need to work with a collection of items whose size can change dynamically.
- **Efficient Insertion and Deletion:** Inserting or deleting elements in a linked list is fast and efficient, as you only need to modify the reference of the next node, which is an O(1) operation.
- **Memory Efficiency:** Linked lists use only as much memory as they need, so they are more efficient with memory compared to arrays, which have a fixed size and can waste memory if not all elements are used.
- **Easy to Implement:** Linked lists are relatively simple to implement and understand compared to other data structures like trees and graphs.
- **Flexibility:** Linked lists can be used to implement various abstract data types, such as stacks, queues, and associative arrays.
- **Easy to navigate:** Linked lists can be easily traversed, making it easier to find specific elements or perform operations on the list.

Disadvantages of Linked Lists:

Linked lists are a popular data structure in computer science, but like any other data structure, they have certain disadvantages as well. Some of the key disadvantages of linked lists are:

- **Slow Access Time:** Accessing elements in a linked list can be slow, as you need to traverse the linked list to find the element you are looking for, which is an O(n) operation. This makes linked lists a poor choice for situations where you need to access elements quickly.
- **Pointers:** Linked lists use pointers to reference the next node, which can make them more complex to understand and use compared to arrays. This complexity can make linked lists more difficult to debug and maintain.
- **Higher overhead:** Linked lists have a higher overhead compared to arrays, as each node in a linked list requires extra memory to store the reference to the next node.
- **Cache Inefficiency:** Linked lists are cache-inefficient because the memory is not contiguous. This means that when you traverse a linked list, you are not likely to get the data you need in the cache, leading to cache misses and slow performance.
- **Extra memory required:** Linked lists require an extra pointer for each node, which takes up extra memory. This can be a problem when you are working with large data sets, as the extra memory required for the pointers can quickly add up.