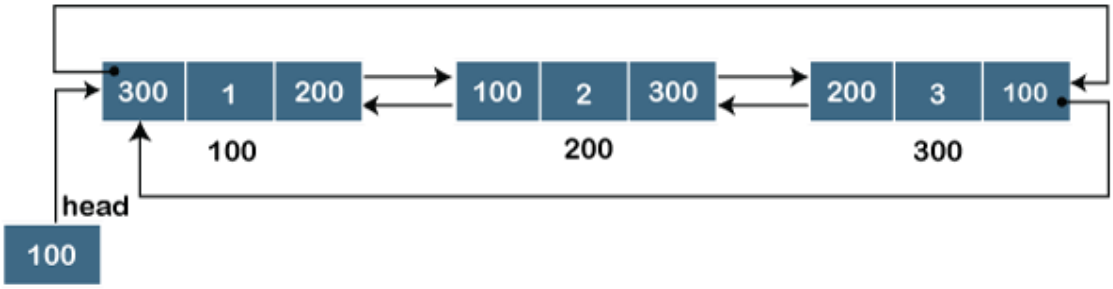
8.5.4.Doubly Circular linked list

The doubly circular linked list has the features of both the circular linked list and doubly linked list.



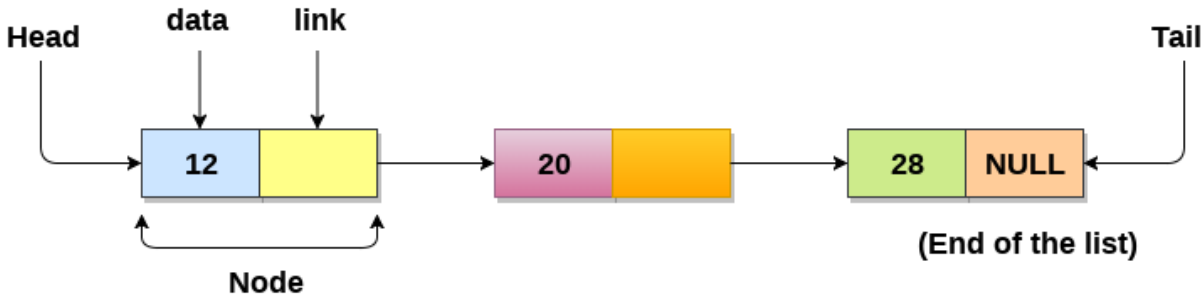
The above figure shows the portrayal of the doubly round connected rundown wherein the last hub is appended to the principal hub and consequently makes a circle. It is a doubly connected rundown likewise in light of the fact that every hub holds the location of the past hub too. The primary distinction between the doubly connected rundown and doubly roundabout connected rundown is that the doubly roundabout connected rundown doesn't contain the NULL incentive in the past field of the hub. As the doubly roundabout connected contains three sections, i.e., two location parts and one information part so its portrayal is like the doubly connected rundown.

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

8.6.Linked List

- Linked List can be defined as collection of objects called nodes that are randomly stored in the memory.
- A node contains two fields i.e. data stored at that particular address and the pointer which contains the address of the next node in the memory.

- The last node of the list contains pointer to the null.



8.7.Uses of Linked List

- The rundown isn't needed to be adjoiningly present in the memory. The hub can dwell anyplace in the memory and connected together to make a rundown. This accomplishes advanced usage of room.
- list size is restricted to the memory size and shouldn't be announced ahead of time.
- Void hub can not be available in the connected rundown.
- We can store estimations of crude sorts or items in the separately connected rundown.

8.8.Why use linked list over array?

Till now, we were utilizing cluster information construction to sort out the gathering of components that are to be put away separately in the memory. Nonetheless, Array has a few points of interest and hindrances which should be known to choose the information structure which will be utilized all through the program.

Array contains following limitations:

- The size of cluster should be known ahead of time prior to utilizing it in the program.
- Expanding size of the cluster is a period taking cycle. It is practically difficult to grow the size of the exhibit at run time.
- All the components in the cluster require to be adorningly put away in the memory. Embedding's any component in the cluster needs moving of every one of its archetypes.

Linked list is the data structure which can overcome all the limitations of an array. Using linked list is useful because

- It dispenses the memory progressively. All the hubs of connected rundown are non-adjacently put away in the memory and connected along with the assistance of pointers.
- Measuring is not, at this point an issue since we don't have to characterize its size at the hour of affirmation. Rundown develops according to the program's interest and restricted to the accessible memory space.

8.8.1.Singly linked list or One way chain

Separately connected rundown can be characterized as the assortment of requested arrangement of components. The quantity of components may shift as indicated by need of the program. A hub in the independently connected rundown comprise of two sections: information part and connection part. Information some portion of the hub stores real data that will be addressed by the hub while the connection a piece of the hub stores the location of its nearby replacement.

One way chain or separately connected rundown can be crossed distinctly one way. As such, we can say that every hub contains just next pointer, hence we can not cross the rundown the opposite way.

Consider a model where the imprints acquired by the understudy in three subjects are put away in a connected rundown as demonstrated in the figure.

In the above figure, the bolt addresses the connections. The information a piece of each hub contains the imprints acquired by the understudy in the diverse subject. The last hub in the rundown is recognized by the invalid pointer which is available in the location part of the last hub. We can have as numerous components we need, in the information part of the rundown.

Complexity

Data Structure	Time Complexity								Space Compleity
	Average				Worst				Worst
	Access	Search	Insertion	Deletion	Access	Search	Insertion	Deletion	
Singly Linked List	θ(n)	θ(n)	θ(1)	θ(1)	O(n)	O(n)	O(1)	O(1)	O(n)

8.8.2.Operations on Singly Linked List

There are various operations which can be performed on singly linked list. A list of all such operations is given below.

Node Creation

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

Insertion

The insertion into a singly linked list can be performed at different positions. Based on the position of the new node being inserted, the insertion is categorized into the following categories.

SN	Operation	Description
1	Insertion at beginning	It involves inserting any element at the front of the list. We just need to a few link adjustments to make the new node as the head of the list.

2	Insertion at end of the list	It involves insertion at the last of the linked list. The new node can be inserted as the only node in the list or it can be inserted as the last one. Different logics are implemented in each scenario.
3	Insertion after specified node	It involves insertion after the specified node of the linked list. We need to skip the desired number of nodes in order to reach the node after which the new node will be inserted. .

Deletion and Traversing

The Deletion of a node from a singly linked list can be performed at different positions. Based on the position of the node being deleted, the operation is categorized into the following categories.

SN	Operation	Description
1	Deletion at beginning	It involves deletion of a node from the beginning of the list. This is the simplest operation among all. It just need a few adjustments in the node pointers.
2	Deletion at the end of the list	It involves deleting the last node of the list. The list can either be empty or full. Different logic is implemented for the different scenarios.
3	Deletion after specified node	It involves deleting the node after the specified node in the list. we need to skip the desired number of nodes to reach the node after which the node will be deleted. This requires traversing through the list.
4	Traversing	In traversing, we simply visit each node of the list at least once in order to perform some specific operation on it, for example, printing data part of each node present in the list.

5	Searching	In searching, we match each element of the list with the given element. If the element is found on any of the location then location of that element is returned otherwise null is returned. .

8.8.3.Linked List in C: Menu Driven Program

```
#include<stdio.h>
#include<stdlib.h>
struct node
{
int data;
struct node *next;
};
struct node *head;

voidbeginsert ();
voidlastinsert ();
voidrandominsert();
voidbegin_delete();
voidlast_delete();
voidrandom_delete();
void display();
void search();
void main ()
{
int choice =0;
while(choice != 9)
   {
printf("\n\n*********Main Menu*********\n");
printf("\nChoose one option from the following list ...\n");
printf("\n===============================================\n");
```

```
printf("\n1.Insert in begining\n2.Insert at last\n3.Insert at any random location\n4.Delete from Beginning\n
5.Delete from last\n6.Delete node after specified location\n7.Search for an element\n8.Show\n9.Exit\n");
printf("\nEnter your choice?\n");
scanf("\n%d",&choice);
switch(choice)
	{
case 1:
beginsert();
break;
case 2:
lastinsert();
break;
case 3:
randominsert();
break;
case 4:
begin_delete();
break;
case 5:
last_delete();
break;
case 6:
random_delete();
break;
case 7:
search();
break;
case 8:
display();
```

```
break;
case 9:
exit(0);
break;
default:
printf("Please enter valid choice..");
        }
    }
}
voidbeginsert()
{
struct node *ptr;
int item;
ptr = (struct node *) malloc(sizeof(struct node *));
if(ptr == NULL)
    {
printf("\nOVERFLOW");
    }
else
    {
printf("\nEnter value\n");
scanf("%d",&item);
ptr->data = item;
ptr->next = head;
head = ptr;
printf("\nNode inserted");
    }

}
voidlastinsert()
{
```

```
struct node *ptr,*temp;
int item;
ptr = (struct node*)malloc(sizeof(struct node));
if(ptr == NULL)
   {
printf("\nOVERFLOW");
   }
else
   {
printf("\nEnter value?\n");
scanf("%d",&item);
ptr->data = item;
if(head == NULL)
      {
ptr -> next = NULL;
head = ptr;
printf("\nNode inserted");
      }
else
      {
temp = head;
while (temp -> next != NULL)
         {
temp = temp -> next;
         }
temp->next = ptr;
ptr->next = NULL;
printf("\nNode inserted");

      }
   }
```

```
}
voidrandominsert()
{
inti,loc,item;
struct node *ptr, *temp;
ptr = (struct node *) malloc (sizeof(struct node));
if(ptr == NULL)
   {
printf("\nOVERFLOW");
   }
else
   {
printf("\nEnter element value");
scanf("%d",&item);
ptr->data = item;
printf("\nEnter the location after which you want to insert ");
scanf("\n%d",&loc);
temp=head;
for(i=0;i<loc;i++)
       {
temp = temp->next;
if(temp == NULL)
           {
printf("\ncan't insert\n");
return;
           }

       }
ptr ->next = temp ->next;
temp ->next = ptr;
printf("\nNode inserted");
```

```
    }
}
voidbegin_delete()
{
struct node *ptr;
if(head == NULL)
    {
printf("\nList is empty\n");
    }
else
    {
ptr = head;
head = ptr->next;
free(ptr);
printf("\nNode deleted from the begining ...\n");
    }
}
voidlast_delete()
{
struct node *ptr,*ptr1;
if(head == NULL)
    {
printf("\nlist is empty");
    }
else if(head -> next == NULL)
    {
head = NULL;
free(head);
printf("\nOnly node of the list deleted ...\n");
    }
```

```
else
   {
ptr = head;
while(ptr->next != NULL)
       {
          ptr1 = ptr;
ptr = ptr ->next;
       }
       ptr1->next = NULL;
free(ptr);
printf("\nDeleted Node from the last ...\n");
   }
}
voidrandom_delete()
{
struct node *ptr,*ptr1;
intloc,i;
printf("\n Enter the location of the node after which you want to perform deletion
\n");
scanf("%d",&loc);
ptr=head;
for(i=0;i<loc;i++)
   {
       ptr1 = ptr;
ptr = ptr->next;

if(ptr == NULL)
       {
printf("\nCan't delete");
return;
       }
```

```
    }
    ptr1 ->next = ptr ->next;
free(ptr);
printf("\nDeleted node %d ",loc+1);
}
void search()
{
struct node *ptr;
intitem,i=0,flag;
ptr = head;
if(ptr == NULL)
    {
printf("\nEmpty List\n");
    }
else
    {
printf("\nEnter item which you want to search?\n");
scanf("%d",&item);
while (ptr!=NULL)
        {
if(ptr->data == item)
            {
printf("item found at location %d ",i+1);
flag=0;
            }
else
            {
flag=1;
            }
i++;
ptr = ptr -> next;
```

```
        }
if(flag==1)
        {
printf("Item not found\n");
        }
    }

}

void display()
{
struct node *ptr;
ptr = head;
if(ptr == NULL)
    {
printf("Nothing to print");
    }
else
    {
printf("\nprinting values . . . . .\n");
while (ptr!=NULL)
        {
printf("\n%d",ptr->data);
ptr = ptr -> next;
        }
    }
}
```

Output:

**********Main Menu**********

Choose one option from the following list ...

==========

1.Insert in begining

2.Insert at last

3.Insert at any random location

4.Delete from Beginning

5.Delete from last

6.Delete node after specified location

7.Search for an element

8.Show

9.Exit

Enter your choice?

1

Enter value

1

Node inserted

*********Main Menu*********

Choose one option from the following list ...

==========

1.Insert in begining

2.Insert at last

3.Insert at any random location

4.Delete from Beginning

5.Delete from last

6.Delete node after specified location

7.Search for an element

8.Show

9.Exit

Enter your choice?

2

Enter value?

2

Node inserted

*********Main Menu*********

Choose one option from the following list ...

==========

1.Insert in begining

2.Insert at last

3.Insert at any random location

4.Delete from Beginning

5.Delete from last

6.Delete node after specified location

7.Search for an element

8.Show

9.Exit

Enter your choice?

3

Enter element value1

Enter the location after which you want to insert 1

Node inserted

*********Main Menu*********

Choose one option from the following list ...

==========

1.Insert in begining

2.Insert at last

3.Insert at any random location

4.Delete from Beginning

5.Delete from last

6.Delete node after specified location

7.Search for an element

8.Show

9.Exit

Enter your choice?

8

printing values . . . . .

1

2

1

*********Main Menu*********

Choose one option from the following list ...

==========

1.Insert in begining

2.Insert at last

3.Insert at any random location

4.Delete from Beginning

5.Delete from last

6.Delete node after specified location

7.Search for an element

8.Show

9.Exit

Enter your choice?

2

Enter value?

123

Node inserted

*********Main Menu*********

Choose one option from the following list ...

==========

1.Insert in begining

2.Insert at last

3.Insert at any random location

4.Delete from Beginning

5.Delete from last

6.Delete node after specified location

7.Search for an element

8.Show

9.Exit

Enter your choice?

1

Enter value

1234

Node inserted

*********Main Menu*********

Choose one option from the following list ...

==========

1.Insert in begining

2.Insert at last

3.Insert at any random location

4.Delete from Beginning

5.Delete from last

6.Delete node after specified location

7.Search for an element

8.Show

9.Exit

Enter your choice?

4

Node deleted from the begining ...

*********Main Menu*********

Choose one option from the following list ...

==========

1.Insert in begining

2.Insert at last

3.Insert at any random location

4.Delete from Beginning

5.Delete from last

6.Delete node after specified location

7.Search for an element

8.Show

9.Exit

Enter your choice?

5

Deleted Node from the last ...

**********Main Menu**********

Choose one option from the following list ...

==========

1.Insert in begining

2.Insert at last

3.Insert at any random location

4.Delete from Beginning

5.Delete from last

6.Delete node after specified location

7.Search for an element

8.Show

9.Exit

Enter your choice?
6

Enter the location of the node after which you want to perform deletion
1

Deleted node 2

*********Main Menu*********

Choose one option from the following list ...

==========

1.Insert in begining
2.Insert at last
3.Insert at any random location
4.Delete from Beginning
5.Delete from last
6.Delete node after specified location
7.Search for an element
8.Show
9.Exit

Enter your choice?
8

printing values . . . . .

1

1

*********Main Menu*********

Choose one option from the following list ...

==========

1.Insert in begining

2.Insert at last

3.Insert at any random location

4.Delete from Beginning

5.Delete from last

6.Delete node after specified location

7.Search for an element

8.Show

9.Exit

Enter your choice?

7

Enter item which you want to search?

1

item found at location 1

item found at location 2

*********Main Menu*********

Choose one option from the following list ...

==========

1.Insert in begining

2.Insert at last

3.Insert at any random location

4.Delete from Beginning

5.Delete from last

6.Delete node after specified location

7.Search for an element

8.Show

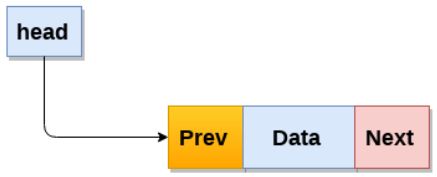
9.Exit

Enter your choice?

9

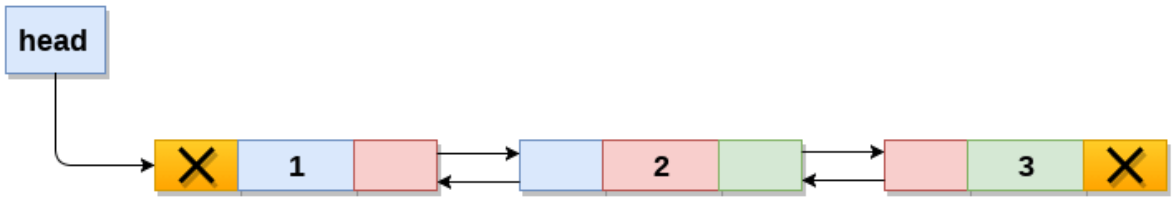
8.9.Doubly linked list

Doubly connected rundown is a mind boggling kind of connected rundown wherein a hub contains a pointer to the past just as the following hub in the arrangement. Subsequently, in a doubly connected rundown, a hub comprises of three sections: hub information, pointer to the following hub in arrangement (next pointer) , pointer to the past hub (past pointer). An example hub in a doubly connected rundown is appeared in the figure.



Node

A doubly linked list containing three nodes having numbers from 1 to 3 in their data part, is shown in the following image.



Doubly Linked List

In C, structure of a node in doubly linked list can be given as :

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

The prev part of the first node and the next part of the last node will always contain null indicating end in each direction.

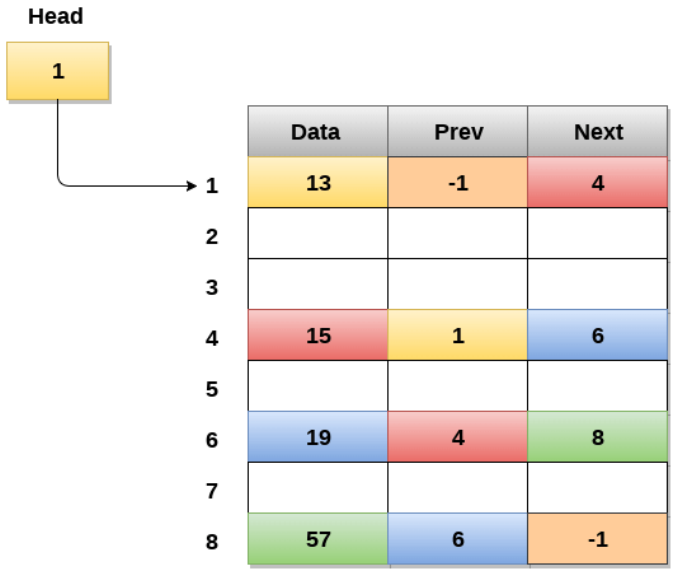
Doubly connected rundown is a mind boggling kind of connected rundown wherein a hub contains a pointer to the past just as the following hub in the arrangement. Subsequently, in a doubly connected rundown, a hub comprises of three sections: hub information, pointer to the following hub in arrangement (next pointer) , pointer to the past hub (past pointer). An example hub in a doubly connected rundown is appeared in the figure.

8.9.1.Memory Representation of a doubly linked list

Memory Representation of a doubly connected rundown is appeared in the accompanying picture. For the most part, doubly connected rundown burns-through more space for each hub and in this way, causes more sweeping fundamental activities, for example, inclusion and erasure. Notwithstanding, we can without much of a stretch control the components of the rundown since the rundown keeps up pointers in both the ways (forward and in reverse).

In the accompanying picture, the principal component of the rundown that is for example 13 put away at address 1. The head pointer focuses to the beginning location 1. Since this is the primary component being added to the rundown along these lines the prev of the rundown contains invalid. The following hub of the rundown lives at address 4 accordingly the first hub contains 4 in quite a while next pointer.

We can cross the rundown in this manner until we discover any hub containing invalid or - 1 in its next part.



Memory Representation of a Doubly linked list

8.9.2.Operations on doubly linked list

Node Creation

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

All the remaining operations regarding doubly linked list are described in the following table.

SN	Operation	Description
1	Insertion at beginning	Adding the node into the linked list at beginning.
2	Insertion at end	Adding the node into the linked list to the end.
3	Insertion after specified node	Adding the node into the linked list after the specified node.
4	Deletion at beginning	Removing the node from beginning of the list
5	Deletion at the end	Removing the node from end of the list.
6	Deletion of the node having given data	Removing the node which is present just after the node containing the given data.
7	Searching	Comparing each node data with the item to be searched and return the location of the item in the list if the item found else return null.
8	Traversing	Visiting each node of the list at least once in order to perform some specific operation like searching, sorting, display, etc.

8.9.3.Menu Driven Program in C to implement all the operations of doubly linked list

```
#include<stdio.h>
#include<stdlib.h>
struct node
{
```

```
struct node *prev;
struct node *next;
int data;
};
struct node *head;
voidinsertion_beginning();
voidinsertion_last();
voidinsertion_specified();
voiddeletion_beginning();
voiddeletion_last();
voiddeletion_specified();
void display();
void search();
void main ()
{
int choice =0;
while(choice != 9)
  {
printf("\n*********Main Menu*********\n");
printf("\nChoose one option from the following list ...\n");
printf("\n===============================================\n");
printf("\n1.Insert in begining\n2.Insert at last\n3.Insert at any random
location\n4.Delete from Beginning\n
5.Delete from last\n6.Delete the node after the given
data\n7.Search\n8.Show\n9.Exit\n");
printf("\nEnter your choice?\n");
scanf("\n%d",&choice);
switch(choice)
      {
case 1:
insertion_beginning();
```

```
break;
case 2:
insertion_last();
break;
case 3:
insertion_specified();
break;
case 4:
deletion_beginning();
break;
case 5:
deletion_last();
break;
case 6:
deletion_specified();
break;
case 7:
search();
break;
case 8:
display();
break;
case 9:
exit(0);
break;
default:
printf("Please enter valid choice..");
        }
    }
}
voidinsertion_beginning()
```

```
{
struct node *ptr;
int item;
ptr = (struct node *)malloc(sizeof(struct node));
if(ptr == NULL)
  {
printf("\nOVERFLOW");
  }
else
  {
printf("\nEnter Item value");
scanf("%d",&item);

if(head==NULL)
  {
ptr->next = NULL;
ptr->prev=NULL;
ptr->data=item;
head=ptr;
  }
else
  {
ptr->data=item;
ptr->prev=NULL;
ptr->next = head;
head->prev=ptr;
head=ptr;
  }
printf("\nNode inserted\n");
}
```

```
}
voidinsertion_last()
{
struct node *ptr,*temp;
int item;
ptr = (struct node *) malloc(sizeof(struct node));
if(ptr == NULL)
  {
printf("\nOVERFLOW");
  }
else
  {
printf("\nEnter value");
scanf("%d",&item);
ptr->data=item;
if(head == NULL)
     {
ptr->next = NULL;
ptr->prev = NULL;
head = ptr;
     }
else
     {
temp = head;
while(temp->next!=NULL)
       {
temp = temp->next;
       }
temp->next = ptr;
ptr ->prev=temp;
ptr->next = NULL;
```

```
        }

      }
printf("\nnode inserted\n");
   }
voidinsertion_specified()
{
struct node *ptr,*temp;
intitem,loc,i;
ptr = (struct node *)malloc(sizeof(struct node));
if(ptr == NULL)
  {
printf("\n OVERFLOW");
  }
else
  {
temp=head;
printf("Enter the location");
scanf("%d",&loc);
for(i=0;i<loc;i++)
      {
temp = temp->next;
if(temp == NULL)
         {
printf("\n There are less than %d elements", loc);
return;
         }
      }
printf("Enter value");
scanf("%d",&item);
ptr->data = item;
```

```
ptr->next = temp->next;
ptr ->prev = temp;
temp->next = ptr;
temp->next->prev=ptr;
printf("\nnode inserted\n");
   }
}
voiddeletion_beginning()
{
struct node *ptr;
if(head == NULL)
   {
printf("\n UNDERFLOW");
   }
else if(head->next == NULL)
   {
head = NULL;
free(head);
printf("\nnode deleted\n");
   }
else
   {
ptr = head;
head = head -> next;
head ->prev = NULL;
free(ptr);
printf("\nnode deleted\n");
   }
}
voiddeletion_last()
{
```

```
struct node *ptr;
if(head == NULL)
   {
printf("\n UNDERFLOW");
   }
else if(head->next == NULL)
   {
head = NULL;
free(head);
printf("\nnode deleted\n");
   }
else
   {
ptr = head;
if(ptr->next != NULL)
       {
ptr = ptr -> next;
       }
ptr ->prev -> next = NULL;
free(ptr);
printf("\nnode deleted\n");
   }
}
voiddeletion_specified()
{
struct node *ptr, *temp;
intval;
printf("\n Enter the data after which the node is to be deleted : ");
scanf("%d", &val);
ptr = head;
while(ptr -> data != val)
```

```
ptr = ptr -> next;
if(ptr -> next == NULL)
   {
printf("\nCan't delete\n");
   }
else if(ptr -> next -> next == NULL)
   {
ptr ->next = NULL;
   }
else
   {
temp = ptr -> next;
ptr -> next = temp -> next;
temp -> next ->prev = ptr;
free(temp);
printf("\nnode deleted\n");
   }
}
void display()
{
struct node *ptr;
printf("\n printing values...\n");
ptr = head;
while(ptr != NULL)
   {
printf("%d\n",ptr->data);
ptr=ptr->next;
   }
}
void search()
{
```

```
struct node *ptr;
intitem,i=0,flag;
ptr = head;
if(ptr == NULL)
   {
printf("\nEmpty List\n");
   }
else
   {
printf("\nEnter item which you want to search?\n");
scanf("%d",&item);
while (ptr!=NULL)
      {
if(ptr->data == item)
         {
printf("\nitem found at location %d ",i+1);
flag=0;
break;
         }
else
         {
flag=1;
         }
i++;
ptr = ptr -> next;
      }
if(flag==1)
      {
printf("\nItem not found\n");
      }
   }
```

}

Output

*********Main Menu*********

Choose one option from the following list ...

==========

1.Insert in beginning

2.Insert at last

3.Insert at any random location

4.Delete from Beginning

5.Delete from last

6.Delete the node after the given data

7.Search

8.Show

9.Exit

Enter your choice?

8

printing values...

*********Main Menu*********

Choose one option from the following list ...

==========

1.Insert in beginning

2.Insert at last

3.Insert at any random location

4.Delete from Beginning

5.Delete from last

6.Delete the node after the given data

7.Search

8.Show

9.Exit

Enter your choice?

1

Enter Item value12

Node inserted

*********Main Menu*********

Choose one option from the following list ...

==========

1.Insert in begining

2.Insert at last

3.Insert at any random location

4.Delete from Beginning

5.Delete from last

6.Delete the node after the given data

7.Search

8.Show

9.Exit

Enter your choice?

1

Enter Item value123

Node inserted

*********Main Menu*********

Choose one option from the following list ...

==========

1.Insert in begining

2.Insert at last

3.Insert at any random location

4.Delete from Beginning

5.Delete from last

6.Delete the node after the given data

7.Search

8.Show

9.Exit

Enter your choice?