Unit 4 : Chapter 8

Linked List

8.0. Objective

This chapter would make you understand the following concepts:

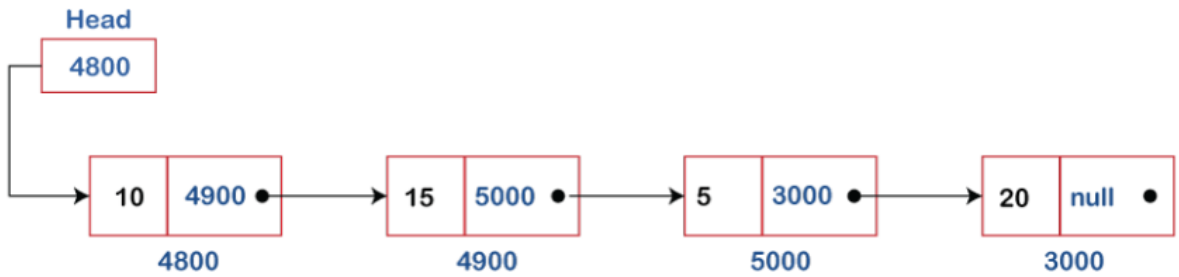
- **To understand the concept of Linked List**
- **To understand Types of Linked List**
- **To Singly Linked list**
- **To Doubly Linked list**

8.1. What is Linked List?

A linked list is also a collection of elements, but the elements are not stored in a consecutive location.Suppose a programmer made a request for storing the integer value then size of 4-byte memory block is assigned to the integer value. The programmer made another request for storing 3 more integer elements; then, three different memory blocks are assigned to these three elements but the memory blocks are available in a random location. So, how are the elements connected?.

These elements are linked to each other by providing one additional information along with an element, i.e., the address of the next element. The variable that stores the address of the next element is known as a pointer. Therefore, we conclude that the linked list contains two parts, i.e., the first one is the data element, and the other is the pointer. The pointer variable will occupy 4 bytes which is pointing to the next element.

A linked list can also be defined as the collection of the nodes in which one node is connected to another node, and node consists of two parts, i.e., one is the data part and the second one is the address part, as shown in the below figure:



In the above figure, we can observe that each node contains the data and the address of the next node. The last node of the linked list contains the NULL value in the address part.

8.2. How can we declare the Linked list?

The need line can be actualized in four different ways that incorporate clusters, connected rundown, stack information construction and twofold pursuit tree. The load information structure is the most productive method of executing the need line, so we will actualize the need line utilizing a store information structure in this

subject. Presently, first we comprehend the motivation behind why pile is the most productive route among the wide range of various information structures.

The structure of a linked list can be defined as:

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

In the above declaration, we have defined a structure named as a node consisting of two variables: an integer variable (data), and the other one is the pointer (next), which contains the address of the next node.

8.3. Advantages of using a Linked list over Array

The following are the advantages of using a linked list over an array:

Dynamic data structure:

The size of the linked list is not fixed as it can vary according to our requirements.

Insertion and Deletion:

Insertion and deletion in linked list are easier than array as the elements in an array are stored in a consecutive location. In contrast, in the case of a linked list, the elements are stored in a random location. The complexity for insertion and deletion of elements from the beginning is O(1) in the linked list, while in the case of an array, the complexity would be O(n). If we want to insert or delete the element in an array, then we need to shift the elements for creating the space. On the other hand, in the linked list, we do not have to shift the elements. In the linked list, we just need to update the address of the pointer in the node.

Memory efficient

Its memory consumption is efficient as the size of the linked list can grow or shrink according to our requirements.

Implementation

Both the stacks and queues can be implemented using a linked list.

Disadvantages of Linked list

The following are the disadvantages of linked list:

Memory usage

The node in a linked list occupies more memory than array as each node occupies two types of variables, i.e., one is a simple variable, and another is a pointer variable that occupies 4 bytes in the memory.

Traversal

In a linked list, the traversal is not easy. If we want to access the element in a linked list, we cannot access the element randomly, but in the case of an array, we can randomly access the element by index. For example, if we want to access the 3rd node, then we need to traverse all the nodes before it. So, the time required to access a particular node is large.

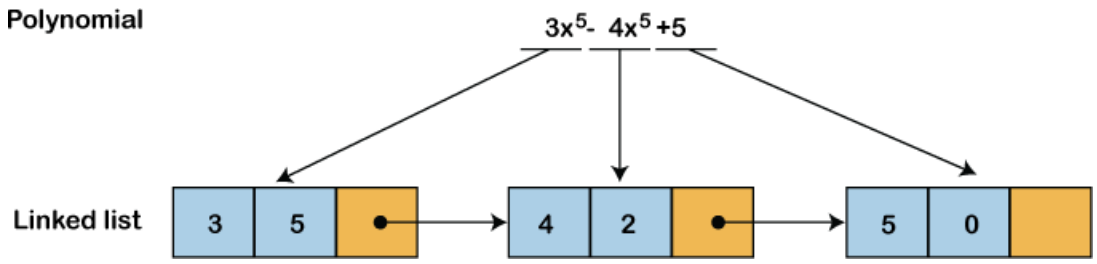
Reverse traversing

In a linked list, backtracking or reverse traversing is difficult. In a doubly linked list, it is easier but requires more memory to store the back pointer.

8.4. Applications of Linked List

The applications of the linked list are given below:

- With the assistance of a connected rundown, the polynomials can be addressed just as we can play out the procedure on the polynomial. We realize that polynomial is an assortment of terms where each term contains coefficient and force. The coefficients and force of each term are put away as hub and connection pointer focuses to the following component in a connected rundown, so connected rundown can be utilized to make, erase and show the polynomial.



- An inadequate grid is utilized in logical calculation and mathematical investigation. In this way, a connected rundown is utilized to address the scanty grid.
- The different tasks like understudy's subtleties, worker's subtleties or item subtleties can be executed utilizing the connected rundown as the connected rundown utilizes the design information type that can hold distinctive information types.
- Stack, Queue, tree and different other information constructions can be executed utilizing a connected rundown.
- The chart is an assortment of edges and vertices, and the diagram can be addressed as a nearness lattice and contiguousness list. In the event that we need to address the diagram as a nearness network, at that point it very well may be actualized as an exhibit. In the event that we need to address the diagram as a nearness list, at that point it tends to be actualized as a connected rundown.
- To actualize hashing, we require hash tables. The hash table contains sections that are executed utilizing connected rundown.
- A connected rundown can be utilized to execute dynamic memory designation. The powerful memory distribution is the memory designation done at the run-time.

8.5. Types of Linked List

Before knowing about the types of a linked list, we should know what is linked list. So, to know about the linked list, click on the link given below:

Types of Linked list

The following are the types of linked list:

Singly Linked list

Doubly Linked list

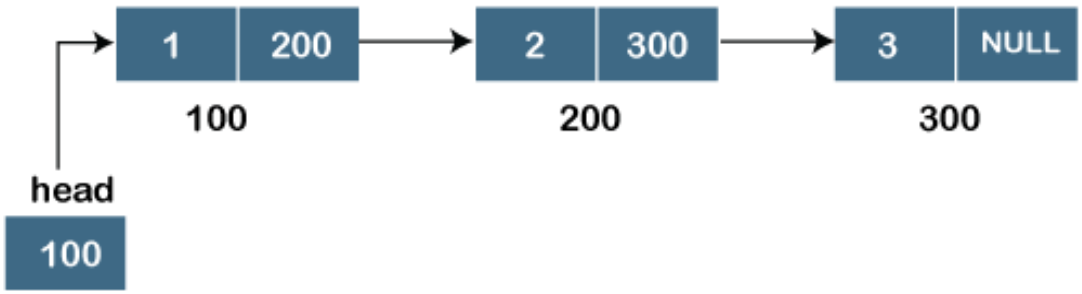
Circular Linked list

Doubly Circular Linked list

8.5.1. Singly Linked list

It is the normally utilized connected rundown in projects. In the event that we are discussing the connected show, it implies it is a separately connected rundown. The separately connected rundown is an information structure that contains two sections, i.e., one is the information part, and the other one is the location part, which contains the location of the following or the replacement hub. The location part in a hub is otherwise called a pointer.

Assume we have three hubs, and the locations of these three hubs are 100, 200 and 300 separately. The portrayal of three hubs as a connected rundown is appeared in the underneath figure:



We can see in the above figure that there are three unique hubs having address 100, 200 and 300 individually. The principal hub contains the location of the following hub, i.e., 200, the subsequent hub contains the location of the last hub, i.e., 300, and the third hub contains the NULL incentive in its location part as it doesn't highlight any hub. The pointer that holds the location of the underlying hub is known as a head pointer.

The connected rundown, which is appeared in the above outline, is referred to as an independently connected rundown as it contains just a solitary connection. In this rundown, just forward crossing is conceivable; we can't navigate the regressive way as it has just one connection in the rundown.

Representation of the node in a singly linked list

```
struct node
{
int data;
```

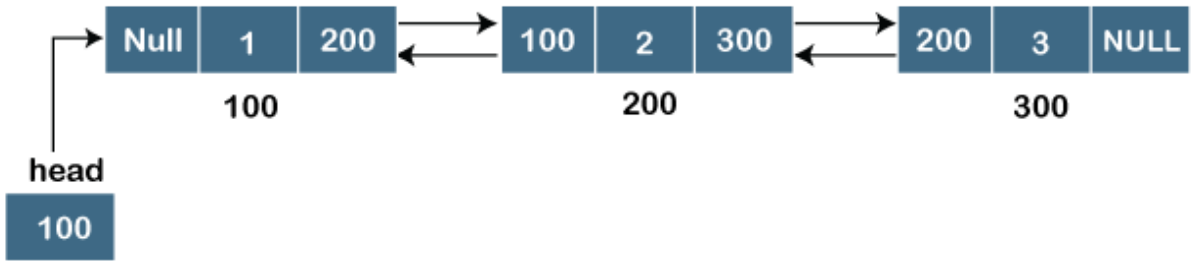
struct node *next;

}

8.5.2. Doubly linked list

As the name recommends, the doubly connected rundown contains two pointers. We can characterize the doubly connected rundown as a straight information structure with three sections: the information part and the other two location part. All in all, a doubly connected rundown is a rundown that has three sections in a solitary hub, incorporates one information section, a pointer to its past hub, and a pointer to the following hub.

Assume we have three hubs, and the location of these hubs are 100, 200 and 300, separately. The portrayal of these hubs in a doubly-connected rundown is appeared beneath:



As we can see in the above figure, the hub in a doubly-connected rundown has two location parts; one section stores the location of the following while the other piece of the hub stores the past hub's location. The underlying hub in the doubly connected rundown has the NULL incentive in the location part, which gives the location of the past hub.

Representation of the node in a doubly linked list

struct node

{

int data;

struct node *next;

struct node *prev;

}

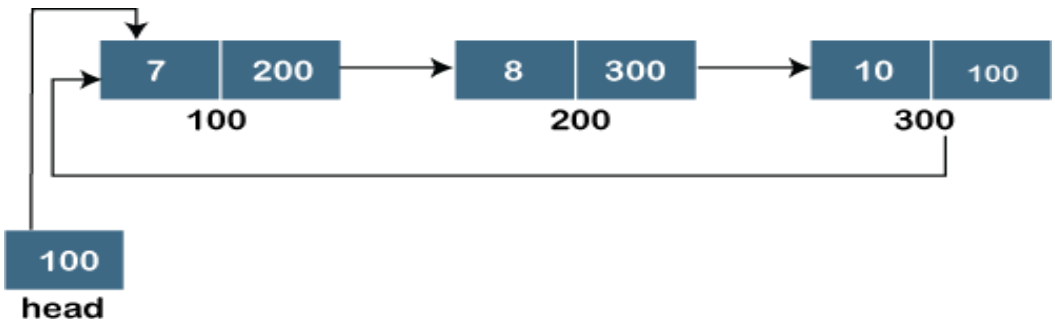
In the above portrayal, we have characterized a client characterized structure named a hub with three individuals, one is information of number sort, and the other two are the pointers, i.e., next and prev of the hub type. The following pointer variable holds the location of the following hub, and the prev pointer holds the location of the past hub. The sort of both the pointers, i.e., next and prev is struct hub as both the pointers are putting away the location of the hub of the struct hub type.

8.5.3. Circular linked list

A round connected rundown is a variety of an independently connected rundown. The lone contrast between the separately connected rundown and a round connected rundown is that the last hub doesn't highlight any hub in an independently connected rundown, so its connection part contains a NULL worth. Then again, the roundabout connected rundown is a rundown where the last hub interfaces with the principal hub, so the connection a piece of the last hub holds the main hub's location. The round connected rundown has no beginning and finishing hub. We can navigate toward any path, i.e., either in reverse or forward. The diagrammatic portrayal of the round connected rundown is appeared underneath:

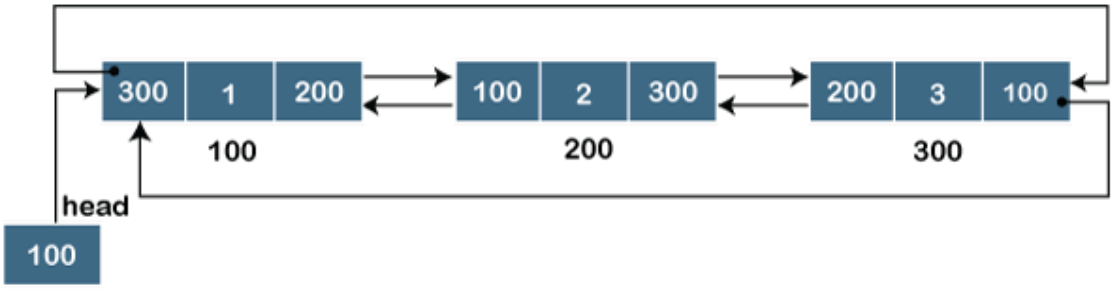
```
struct node
{
int data;
struct node *next;
}
```

A circular linked list is a sequence of elements in which each node has a link to the next node, and the last node is having a link to the first node. The representation of the circular linked list will be similar to the singly linked list, as shown below:



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.



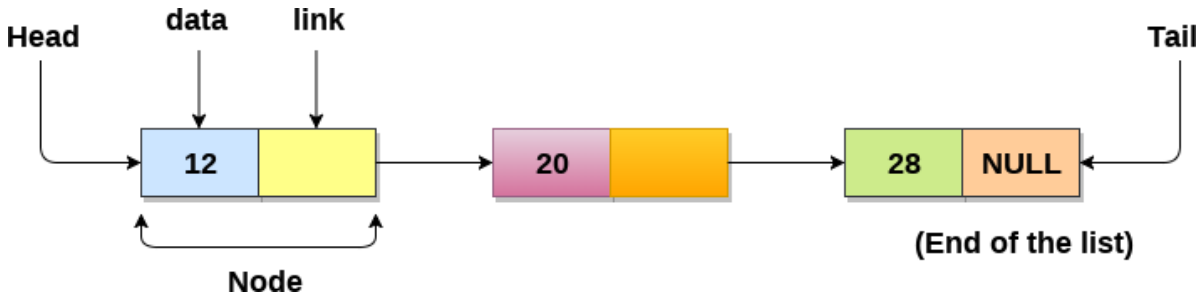
The above figure shows the portrayal of the doubly round connected rundown wherein the last hub is appended to the principal hub and consequently makes a circle. It is a doubly connected rundown likewise in light of the fact that every hub holds the location of the past hub too. The primary distinction 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

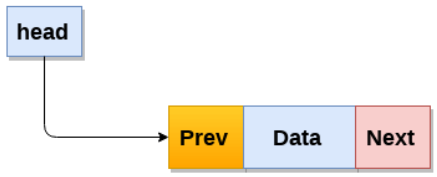
9.Exit

Enter your choice?

9

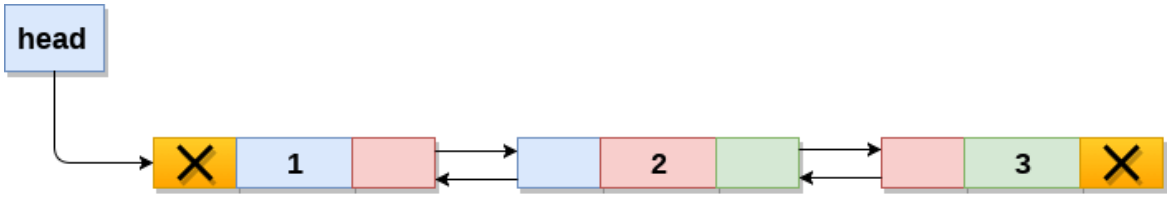
8.9. Doubly linked list

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.

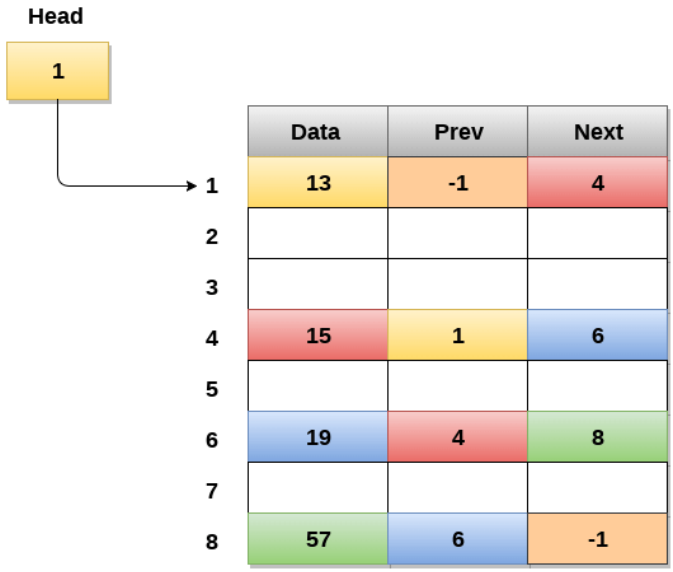
Doubly connected rundown is a mind boggling kind of connected rundown wherein a hub contains a pointer to the past just as the following hub in the arrangement. Subsequently, in a doubly connected rundown, a hub comprises of three sections: hub information, pointer to the following hub in arrangement (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 doublylinked 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?

1

Enter Item value1234

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?

8

printing values...

1234

123

12

*********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?

2

Enter value89

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?

3

Enter the location1

Enter value12345

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?

8

printing values...

1234

123

12345

12

89

*********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?

4

node deleted

*********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?

5

node deleted

*********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?

8

printing values...

123

12345

*********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?

6

Enter the data after which the node is to be deleted : 123

*********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?

8

printing values...

123

*********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?

7

Enter item which you want to search?

123

item found at location 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 the node after the given data

7.Search

8.Show

9.Exit

Enter your choice?

6

Enter the data after which the node is to be deleted : 123

Can't delete

*********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?

9

Exited..

Unit 4 : Chapter 9

Operations on Linked List

9.0. Objective

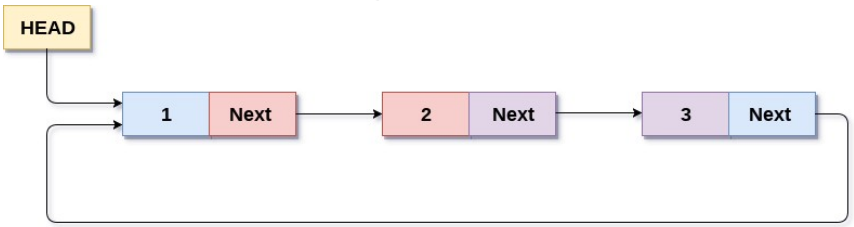
- **To Understand the concept of Circular Singly Linked List**
- **Operations on Circular Singly linked list**
- **To Understand the concept of Circular Doubly Linked List**
- **Memory Management of Circular Doubly linked list**

9.1. Circular Singly Linked List

In a roundabout Singly connected rundown, the last hub of the rundown contains a pointer to the primary hub of the rundown. We can have roundabout separately connected rundown just as roundabout doubly connected rundown.

We navigate a roundabout separately connected rundown until we arrive at a similar hub where we began. The roundabout separately loved rundown has no start and no consummation. There is no invalid worth present in the following piece of any of the hubs.

The accompanying picture shows a round separately connected rundown.

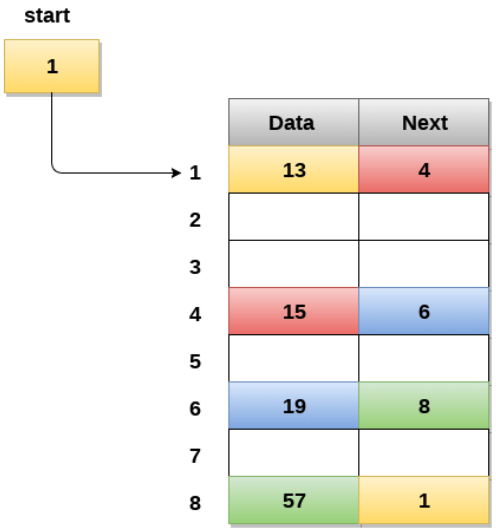


Circular Singly Linked List

Circular linked list are generally utilized in errand support in working frameworks. There are numerous models where round connected rundown are being utilized in software engineering including program riding where a record of pages visited in the past by the client, is kept up as roundabout connected records and can be gotten to again on tapping the past catch.

9.2. Memory Representation of circular linked list:

In the accompanying picture, memory portrayal of a round connected rundown containing signs of an understudy in 4 subjects. Nonetheless, the picture shows a brief look at how the round rundown is being put away in the memory. The beginning or top of the rundown is highlighting the component with the file 1 and containing 13 imprints in the information part and 4 in the following part. Which implies that it is connected with the hub that is being put away at fourth list of the rundown. Notwithstanding, because of the way that we are thinking about roundabout connected rundown in the memory in this manner the last hub of the rundown contains the location of the primary hub of the rundown.



Memory Representation of a circular linked list

We can likewise have more than one number of connected rundown in the memory with the distinctive beginning pointers highlighting the diverse beginning hubs in the rundown. The last hub is distinguished by its next part which contains the location of

the beginning hub of the rundown. We should have the option to recognize the last hub of any connected rundown with the goal that we can discover the quantity of cycles which should be performed while navigating the rundown.

9.3. Operations on Circular Singly linked list:

Insertion

SNO	Operation	Description
1	Insertion at beginning	Adding a node into circular singly linked list at the beginning.
2	Insertion at the end	Adding a node into circular singly linked list at the end.

9.4. Deletion& Traversing

SNO	Operation	Description
1	Deletion at beginning	Removing the node from circular singly linked list at the beginning.
2	Deletion at the end	Removing the node from circular singly linked list at the end.
3	Searching	Compare each element of the node with the given item and return the location at which the item is present in the list otherwise return null.
4	Traversing	Visiting each element of the list at least once in order to perform some specific operation.

9.5. Menu-driven program in C implementing all operations on circular singly linked list

```
#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 != 7)
  {
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.Delete from Beginning\n4.Delete
from last\n5.Search for an element\n6.Show\n7.Exit\n");
printf("\nEnter your choice?\n");
scanf("\n%d",&choice);
switch(choice)
    {
case 1:
beginsert();
break;
case 2:
lastinsert();
break;
case 3:
begin_delete();
break;
```

```
case 4:
last_delete();
break;
case 5:
search();
break;
case 6:
display();
break;
case 7:
exit(0);
break;
default:
printf("Please enter valid choice..");
        }
    }
}
voidbeginsert()
{
struct node *ptr,*temp;
int item;
ptr = (struct node *)malloc(sizeof(struct node));
if(ptr == NULL)
    {
printf("\nOVERFLOW");
    }
else
    {
printf("\nEnter the node data?");
scanf("%d",&item);
ptr -> data = item;
```

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

```
if(head == NULL)
        {
head = ptr;
ptr -> next = head;
        }
else
        {
temp = head;
while(temp -> next != head)
            {
temp = temp -> next;
            }
temp -> next = ptr;
ptr -> next = head;
        }
printf("\nnode inserted\n");
    }
}
voidbegin_delete()
{
struct node *ptr;
if(head == NULL)
    {
printf("\nUNDERFLOW");
    }
else if(head->next == head)
    {
head = NULL;
free(head);
printf("\nnode deleted\n");
    }
```

```
else
    {   ptr = head;
while(ptr -> next != head)
ptr = ptr -> next;
ptr->next = head->next;
free(head);
head = ptr->next;
printf("\nnode deleted\n");

    }
}
voidlast_delete()
{
struct node *ptr, *preptr;
if(head==NULL)
    {
printf("\nUNDERFLOW");
    }
else if (head ->next == head)
    {
head = NULL;
free(head);
printf("\nnode deleted\n");
    }
else
    {
ptr = head;
while(ptr ->next != head)
        {
preptr=ptr;
ptr = ptr->next;
```

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

```
flag=0;
break;
            }
else
            {
flag=1;
            }
i++;
ptr = ptr -> next;
        }
        }
if(flag != 0)
        {
printf("Item not found\n");
        }
    }
}
void display()
{
struct node *ptr;
ptr=head;
if(head == NULL)
    {
printf("\nnothing to print");
    }
else
    {
printf("\n printing values ... \n");
while(ptr -> next != head)
        {
printf("%d\n", ptr -> data);
```

```
ptr = ptr -> next;
        }
printf("%d\n", ptr -> data);
    }
}
```

Output:

```
*********Main Menu*********
Choose one option from the following list ...
================================================
1.Insert in begining
2.Insert at last
3.Delete from Beginning
4.Delete from last
5.Search for an element
6.Show
7.Exit
Enter your choice?
1
Enter the node data?10
node inserted
*********Main Menu*********
Choose one option from the following list ...
================================================
1.Insert in beginning
2.Insert at last
3.Delete from Beginning
4.Delete from last
5.Search for an element
6.Show
7.Exit
Enter your choice?
```

2

Enter Data?20

node inserted

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

Choose one option from the following list ...

==========

1.Insert in beginning

2.Insert at last

3.Delete from Beginning

4.Delete from last

5.Search for an element

6.Show

7.Exit

Enter your choice?

2

Enter Data?30

node inserted

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

Choose one option from the following list ...

==========

1.Insert in beginning

2.Insert at last

3.Delete from Beginning

4.Delete from last

5.Search for an element

6.Show

7.Exit

Enter your choice?

3

node deleted

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

Choose one option from the following list ...

==========

1.Insert in beginning

2.Insert at last

3.Delete from Beginning

4.Delete from last

5.Search for an element

6.Show

7.Exit

Enter your choice?

4

node deleted

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

Choose one option from the following list ...

==========

1.Insert in beginning

2.Insert at last

3.Delete from Beginning

4.Delete from last

5.Search for an element

6.Show

7.Exit

Enter your choice?

5

Enter item which you want to search?

20

item found at location 1

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

Choose one option from the following list ...

==========

1.Insert in beginning

2.Insert at last

3.Delete from Beginning

4.Delete from last

5.Search for an element

6.Show

7.Exit

Enter your choice?

6

printing values ...

20

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

Choose one option from the following list ...

==========

1.Insert in beginning

2.Insert at last

3.Delete from Beginning

4.Delete from last

5.Search for an element

6.Show

7.Exit

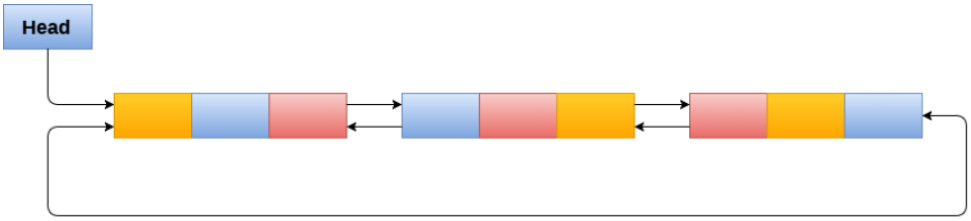
Enter your choice?

7

9.6. Circular Doubly Linked List

Circular Doubly Linked List rundown is a more complexed kind of information structure in which a hub contain pointers to its past hub just as the following hub. Round doubly connected rundown doesn't contain NULL in any of the hub. The last hub of the rundown contains the location of the main hub of the rundown. The main hub of the rundown additionally contain address of the last hub in its past pointer.

A circular doubly linked list is shown in the following figure.

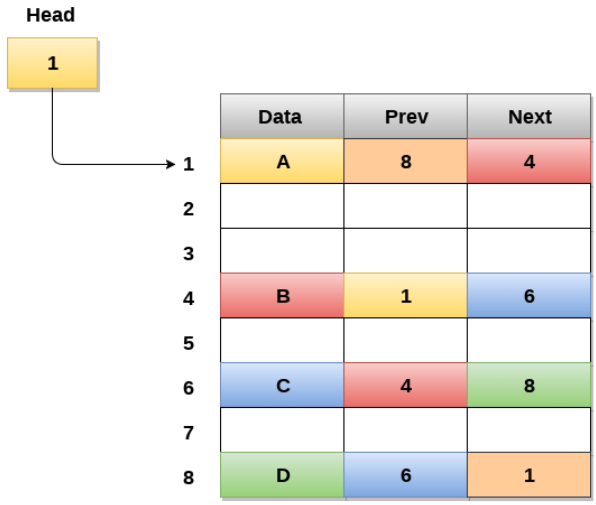


Circular Doubly Linked List

Because of the way that a round doubly connected rundown contains three sections in its design hence, it requests more space per hub and more costly essential activities. Be that as it may, a round doubly connected rundown gives simple control of the pointers and the looking turns out to be twice as proficient.

9.7. Memory Management of Circular Doubly linked list

The accompanying figure shows the manner by which the memory is designated for a round doubly connected rundown. The variable head contains the location of the principal component of the rundown for example 1 consequently the beginning hub of the rundown contains information An is put away at address 1. Since, every hub of the rundown should have three sections along these lines, the beginning hub of the rundown contains address of the last hub for example 8 and the following hub for example 4. The last hub of the rundown that is put away at address 8 and containing information as 6, contains address of the primary hub of the rundown as demonstrated in the picture for example 1. In roundabout doubly connected rundown, the last hub is recognized by the location of the main hub which is put away in the following piece of the last hub hence the hub which contains the location of the principal hub, is really the last hub of the rundown.



Memory Representation of a Circular Doubly linked list

9.8. Operations on circular doubly linked list :

There are different tasks which can be performed on round doubly connected rundown. The hub design of a roundabout doubly connected rundown is like doubly connected rundown. Be that as it may, the procedure on round doubly connected rundown is portrayed in the accompanying table.

SN	Operation	Description
1	Insertion at beginning	Adding a node in circular doubly linked list at the beginning.
2	Insertion at end	Adding a node in circular doubly linked list at the end.
3	Deletion at beginning	Removing a node in circular doubly linked list from beginning.
4	Deletion at end	Removing a node in circular doubly linked list at the end.

Traversing and searching in circular doubly linked list is similar to that in the circular singly linked list.

9.9.C program to implement all the operations on circular 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();
voiddeletion_beginning();
voiddeletion_last();
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 Beginning\n2.Insert at last\n3.Delete from Beginning\n4.Delete
from last\n5.Search\n6.Show\n7.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:
```

```
deletion_beginning();
break;
case 4:
deletion_last();
break;
case 5:
search();
break;
case 6:
display();
break;
case 7:
exit(0);
break;
default:
printf("Please enter valid choice..");
        }
    }
}
voidinsertion_beginning()
{
struct node *ptr,*temp;
int item;
ptr = (struct node *)malloc(sizeof(struct node));
if(ptr == NULL)
    {
printf("\nOVERFLOW");
    }
else
    {
printf("\nEnter Item value");
```

```
scanf("%d",&item);
ptr->data=item;
if(head==NULL)
  {
head = ptr;
ptr -> next = head;
ptr ->prev = head;
  }
else
  {
temp = head;
while(temp -> next != head)
   {
temp = temp -> next;
   }
temp -> next = ptr;
ptr ->prev = temp;
head ->prev = ptr;
ptr -> next = head;
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)
      {
head = ptr;
ptr -> next = head;
ptr ->prev = head;
      }
else
      {
temp = head;
while(temp->next !=head)
         {
temp = temp->next;
         }
temp->next  =  ptr;
ptr   ->prev=temp;
head ->prev = ptr;
ptr -> next = head;
      }
   }
printf("\nnode inserted\n");
}
voiddeletion_beginning()
{
struct node *temp;
```

```
if(head == NULL)
   {
printf("\n UNDERFLOW");
   }
else if(head->next == head)
   {
head = NULL;
free(head);
printf("\nnode deleted\n");
   }
else
   {
temp = head;
while(temp -> next != head)
       {
temp = temp -> next;
       }
temp -> next = head -> next;
head -> next ->prev = temp;
free(head);
head = temp -> next;
   }

}
voiddeletion_last()
{
struct node *ptr;
if(head == NULL)
   {
printf("\n UNDERFLOW");
   }
```

```
else if(head->next == head)
   {
head = NULL;
free(head);
printf("\nnode deleted\n");
   }
else
   {
ptr = head;
if(ptr->next != head)
       {
ptr = ptr -> next;
       }
ptr ->prev -> next = head;
head ->prev = ptr ->prev;
free(ptr);
printf("\nnode deleted\n");
   }
}
void display()
{
struct node *ptr;
ptr=head;
if(head == NULL)
   {
printf("\nnothing to print");
   }
else
   {
printf("\n printing values ... \n");
```

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

```
            {
printf("item found at location %d ",i+1);
flag=0;
break;
            }
else
            {
flag=1;
            }
i++;
ptr = ptr -> next;
        }
        }
if(flag != 0)
        {
printf("Item not found\n");
        }
    }
}
```

Output:

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

Choose one option from the following list ...

==========

1.Insert in Beginning

2.Insert at last

3.Delete from Beginning

4.Delete from last

5.Search

6.Show

7.Exit

Enter your choice?

1

Enter Item value123

Node inserted

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

Choose one option from the following list ...

==========

1.Insert in Beginning

2.Insert at last

3.Delete from Beginning

4.Delete from last

5.Search

6.Show

7.Exit

Enter your choice?

2

Enter value234

node inserted

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

Choose one option from the following list ...

==========

1.Insert in Beginning

2.Insert at last

3.Delete from Beginning

4.Delete from last

5.Search

6.Show

7.Exit

Enter your choice?

1

Enter Item value90

Node inserted

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

Choose one option from the following list ...

==========

1.Insert in Beginning

2.Insert at last

3.Delete from Beginning

4.Delete from last

5.Search

6.Show

7.Exit

Enter your choice?

2

Enter value80

node inserted

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

Choose one option from the following list ...

==========

1.Insert in Beginning

2.Insert at last

3.Delete from Beginning

4.Delete from last

5.Search

6.Show

7.Exit

Enter your choice?

3

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

Choose one option from the following list ...

==========

1.Insert in Beginning

2.Insert at last

3.Delete from Beginning

4.Delete from last

5.Search

6.Show

7.Exit

Enter your choice?

4

node deleted

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

Choose one option from the following list ...

==========

1.Insert in Beginning

2.Insert at last

3.Delete from Beginning

4.Delete from last

5.Search

6.Show

7.Exit

Enter your choice?

6

printing values ...

123

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

Choose one option from the following list ...

==========

1.Insert in Beginning

2.Insert at last

3.Delete from Beginning

4.Delete from last

5.Search

6.Show

7.Exit

Enter your choice?

5

Enter item which you want to search?

123

item found at location 1

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

Choose one option from the following list ...

==========

1.Insert in Beginning

2.Insert at last

3.Delete from Beginning

4.Delete from last

5.Search

6.Show

7.Exit

Enter your choice?

7