Linked List

Before understanding the linked list concept, we first look at ***why there is a need for a linked list.***

If we want to store the value in a memory, we need a memory manager that manages the memory for every variable. For example, if we want to create a variable of integer type like:

```
1. int x;
```

In the above example, we have created a variable 'x' of type integer. As we know that integer variable occupies 4 bytes, so 'x' variable will occupy 4 bytes to store the value.

Suppose we want to create an array of integer type like:

```
1. int x[3];
```

In the above example, we have declared an array of size 3. As we know, that all the values of an array are stored in a continuous manner, so all the three values of an array are stored in a sequential fashion. The total memory space occupied by the array would be **3*4 = 12 bytes**.

There are two major drawbacks of using array:

- We cannot insert more than 3 elements in the above example because only 3 spaces are allocated for 3 elements.
- In the case of an array, lots of wastage of memory can occur. For example, if we declare an array of 50 size but we insert only 10 elements in an array. So, in this case, the memory space for other 40 elements will get wasted and cannot be used by another variable as this whole space is occupied by an array.

In array, we are providing the fixed-size at the compile-time, due to which wastage of memory occurs. The solution to this problem is to use the **linked list**.

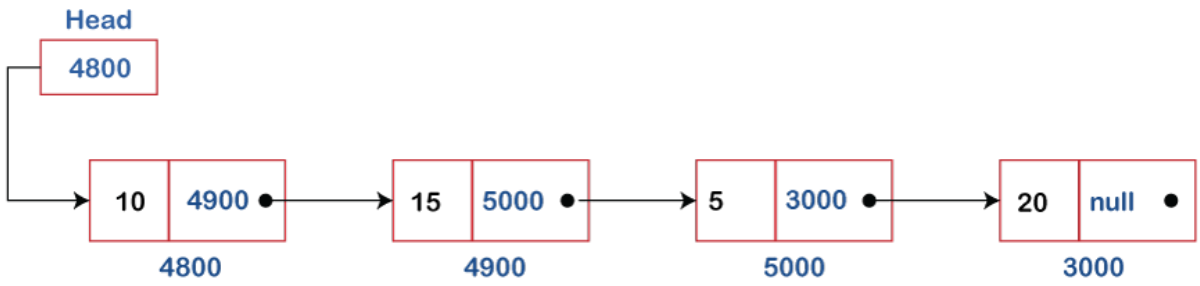
What is Linked List?

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.

How can we declare the Linked list?

The declaration of an array is very simple as it is of single type. But the linked list contains two parts, which are of two different types, i.e., one is a simple variable, and the second one is a pointer variable. We can declare the linked list by using the user-defined data type known as structure.

The structure of a linked list can be defined as:

```
1. struct node
2. {
3.    int data;
4.    struct node *next;
5. }
```

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.

Advantages of using a Linked list over Array

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

- **Dynamicdatastructure:**
  The size of the linked list is not fixed as it can vary according to our requirements.
- **InsertionandDeletion:**
  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.
- **Memoryefficient**
  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

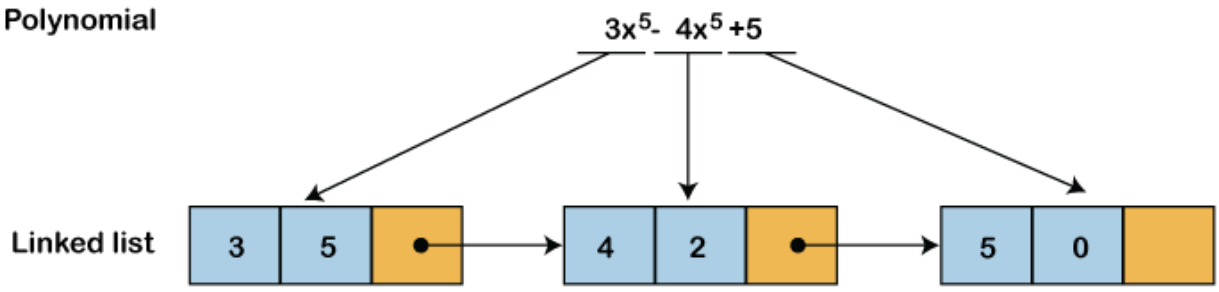
The following are the disadvantages of linked list:

- **Memoryusage**
  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 3^{rd} node, then we need to traverse all the nodes before it. So, the time required to access a particular node is large.
- **Reversetraversing**
  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.

Applications of Linked List

The applications of the linked list are given below:

- With the help of a linked list, the polynomials can be represented as well as we can perform the operations on the polynomial. We know that polynomial is a collection of terms in which each term contains coefficient and power. The coefficients and power of each term are stored as node and link pointer points to the next element in a linked list, so linked list can be used to create, delete and display the polynomial.



- A sparse matrix is used in scientific computation and numerical analysis. So, a linked list is used to represent the sparse matrix.
- The various operations like student's details, employee's details or product details can be implemented using the linked list as the linked list uses the structure data type that can hold different data types.
- Stack, Queue, tree and various other data structures can be implemented using a linked list.
- The graph is a collection of edges and vertices, and the graph can be represented as an adjacency matrix and adjacency list. If we want to represent the graph as an adjacency matrix, then it can be implemented as an array. If we want to represent the graph as an adjacency list, then it can be implemented as a linked list.

- To implement hashing, we require hash tables. The hash table contains entries that are implemented using linked list.
- A linked list can be used to implement dynamic memory allocation. The dynamic memory allocation is the memory allocation done at the run-time.

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

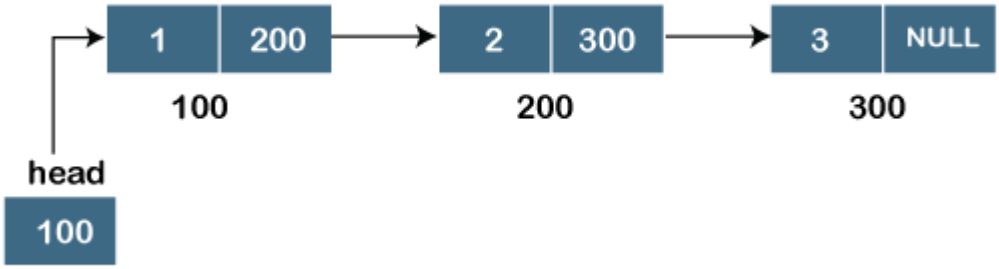
The following are the types of linked list:

- Singly Linked list
- Doubly Linked list
- Circular Linked list
- Doubly Circular Linked list

Singly Linked list

It is the commonly used linked list in programs. If we are talking about the linked list, it means it is a singly linked list. The singly linked list is a data structure that contains two parts, i.e., one is the data part, and the other one is the address part, which contains the address of the next or the successor node. The address part in a node is also known as a **pointer**.

Suppose we have three nodes, and the addresses of these three nodes are 100, 200 and 300 respectively. The representation of three nodes as a linked list is shown in the below figure:



We can observe in the above figure that there are three different nodes having address 100, 200 and 300 respectively. The first node contains the address of the next node, i.e., 200, the second node contains the address of the last node, i.e., 300, and the third node contains the NULL value in its address part as it does not point to any node. The pointer that holds the address of the initial node is known as a ***head pointer***.

The linked list, which is shown in the above diagram, is known as a singly linked list as it contains only a single link. In this list, only forward traversal is possible; we cannot traverse in the backward direction as it has only one link in the list.

Representation of the node in a singly linked list

```
1. struct node
2. {
3.    int data;
4.    struct node *next;
5. }
```

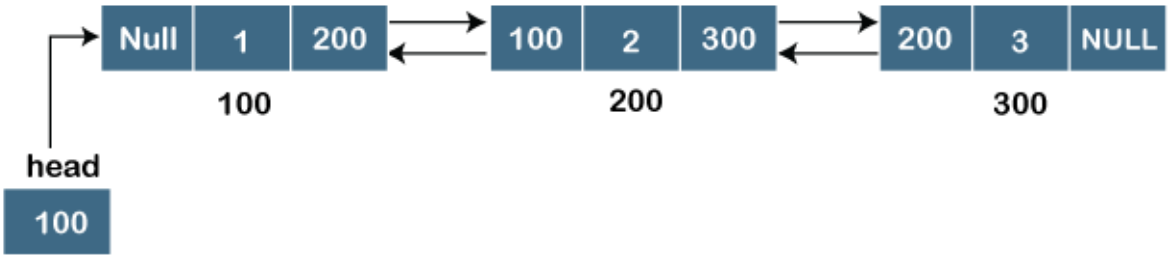
In the above representation, we have defined a user-defined structure named a **node** containing two members, the first one is data of integer type, and the other one is the pointer (next) of the node type.

Doubly linked list

As the name suggests, the doubly linked list contains two pointers. We can define the doubly linked list as a linear data structure with three parts: the data part and the other

two address part. In other words, a doubly linked list is a list that has three parts in a single node, includes one data part, a pointer to its previous node, and a pointer to the next node.

Suppose we have three nodes, and the address of these nodes are 100, 200 and 300, respectively. The representation of these nodes in a doubly-linked list is shown below:



As we can observe in the above figure, the node in a doubly-linked list has two address parts; one part stores the ***address of the next*** while the other part of the node stores the ***previous node's address***. The initial node in the doubly linked list has the **NULL** value in the address part, which provides the address of the previous node.

Representation of the node in a doubly linked list

```
1. struct node
2. {
3.   int data;
4.   struct node *next;
5.  struct node *prev;
6. }
```

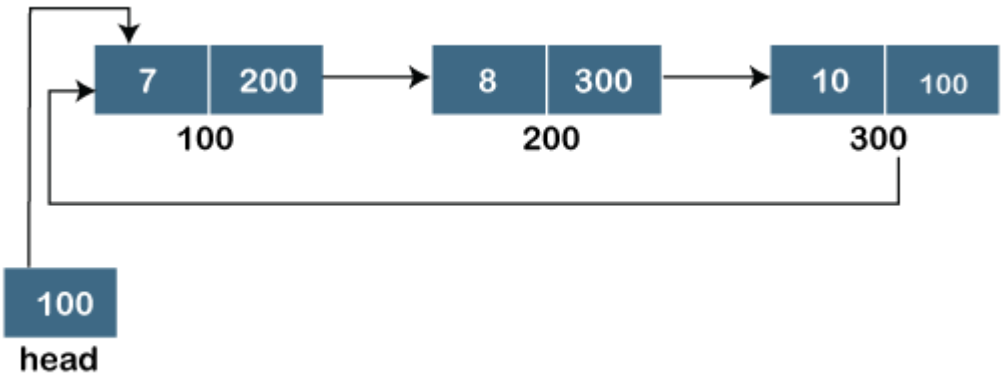
In the above representation, we have defined a user-defined structure named ***a node*** with three members, one is **data** of integer type, and the other two are the pointers, i.e., **next and prev** of the node type. The **next pointer** variable holds the address of the next node, and the **prev pointer** holds the address of the previous node. The type of both the pointers, i.e., **next and prev** is **struct node** as both the pointers are storing the address of the node of the ***struct node*** type.

Circular linked list

A circular linked list is a variation of a singly linked list. The only difference between the ***singly linked list*** and a ***circular linked*** list is that the last node does not point to any node in a singly linked list, so its link part contains a NULL value. On the other hand, the circular linked list is a list in which the last node connects to the first node, so the link part of the last node holds the first node's address. The circular linked list has no starting and ending node. We can traverse in any direction, i.e., either backward or forward. The diagrammatic representation of the circular linked list is shown below:

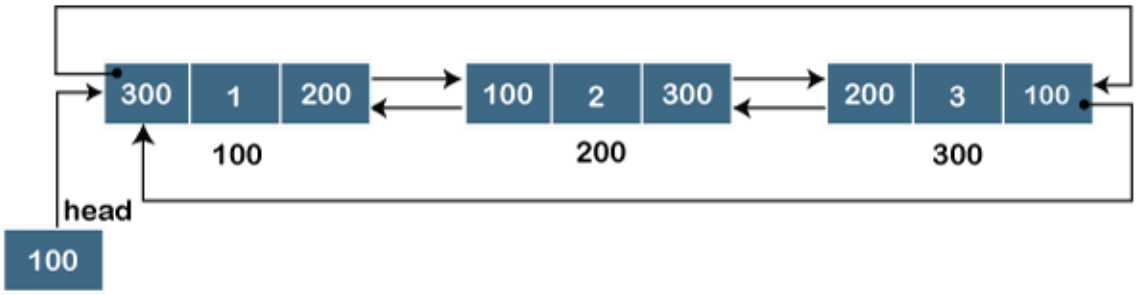
```
1. struct node
2. {
3.    int data;
4.    struct node *next;
5. }
```

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:



Doubly Circular linked list

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

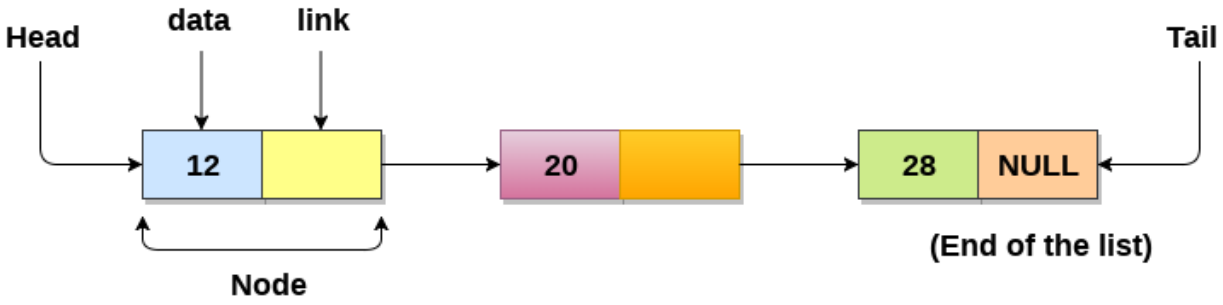


The above figure shows the representation of the doubly circular linked list in which the last node is attached to the first node and thus creates a circle. It is a doubly linked list also because each node holds the address of the previous node also. The main difference between the doubly linked list and doubly circular linked list is that the doubly circular linked list does not contain the NULL value in the previous field of the node. As the doubly circular linked contains three parts, i.e., two address parts and one data part so its representation is similar to the doubly linked list.

```
1. struct node
2. {
3.   int data;
4.   struct node *next;
5.  struct node *prev;
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.



Uses of Linked List

- The list is not required to be contiguously present in the memory. The node can reside any where in the memory and linked together to make a list. This achieves optimized utilization of space.
- list size is limited to the memory size and doesn't need to be declared in advance.
- Empty node can not be present in the linked list.
- We can store values of primitive types or objects in the singly linked list.

Why use linked list over array?

Till now, we were using array data structure to organize the group of elements that are to be stored individually in the memory. However, Array has several advantages and disadvantages which must be known in order to decide the data structure which will be used throughout the program.

Array contains following limitations:

1. The size of array must be known in advance before using it in the program.
2. Increasing size of the array is a time taking process. It is almost impossible to expand the size of the array at run time.
3. All the elements in the array need to be contiguously stored in the memory. Inserting any element in the array needs shifting of all its predecessors.

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

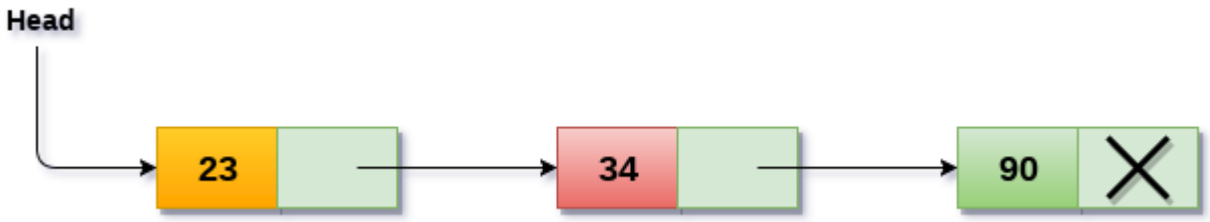
1. It allocates the memory dynamically. All the nodes of linked list are non-contiguously stored in the memory and linked together with the help of pointers.
2. Sizing is no longer a problem since we do not need to define its size at the time of declaration. List grows as per the program's demand and limited to the available memory space.

Singly linked list or One way chain

Singly linked list can be defined as the collection of ordered set of elements. The number of elements may vary according to need of the program. A node in the singly linked list consist of two parts: data part and link part. Data part of the node stores actual information that is to be represented by the node while the link part of the node stores the address of its immediate successor.

One way chain or singly linked list can be traversed only in one direction. In other words, we can say that each node contains only next pointer, therefore we can not traverse the list in the reverse direction.

Consider an example where the marks obtained by the student in three subjects are stored in a linked list as shown in the figure.



In the above figure, the arrow represents the links. The data part of every node contains the marks obtained by the student in the different subject. The last node in the list is identified by the null pointer which is present in the address part of the last node. We can have as many elements we require, in the data part of the list.

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

```
1. struct node
2. {
3.     int data;
4.     struct node *next;
5. };
6. struct node *head, *ptr;
7. 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. .

Linked List in C: Menu Driven Program

```
1. #include<stdio.h>
2. #include<stdlib.h>
3. struct node
4. {
5.     int data;
6.     struct node *next;
7. };
8. struct node *head;
9.
10. void beginsert ();
11. void lastinsert ();
12. void randominsert();
13. void begin_delete();
```

```
14. void last_delete();
15. void random_delete();
16. void display();
17. void search();
18. void main ()
19. {
20.     int choice =0;
21.     while(choice != 9)
22.     {
23.         printf("\n\n*********Main Menu*********\n");
24.         printf("\nChoose one option from the following list ...\n");
25.         printf("\n===============================================\n");
26.         printf("\n1.Insert in begining\n2.Insert at last\n3.Insert at any random location\n4.Delete fro
    m Beginning\n
27.         5.Delete from last\n6.Delete node after specified location\n7.Search for an element\n8.Sho
    w\n9.Exit\n");
28.         printf("\nEnter your choice?\n");
29.         scanf("\n%d",&choice);
30.         switch(choice)
31.         {
32.             case 1:
33.             beginsert();
34.             break;
35.             case 2:
36.             lastinsert();
37.             break;
38.             case 3:
39.             randominsert();
40.             break;
41.             case 4:
42.             begin_delete();
43.             break;
44.             case 5:
45.             last_delete();
```

```
46.             break;
47.             case 6:
48.             random_delete();
49.             break;
50.             case 7:
51.             search();
52.             break;
53.             case 8:
54.             display();
55.             break;
56.             case 9:
57.             exit(0);
58.             break;
59.             default:
60.             printf("Please enter valid choice..");
61.         }
62.     }
63. }
64. void beginsert()
65. {
66.     struct node *ptr;
67.     int item;
68.     ptr = (struct node *) malloc(sizeof(struct node *));
69.     if(ptr == NULL)
70.     {
71.         printf("\nOVERFLOW");
72.     }
73.     else
74.     {
75.         printf("\nEnter value\n");
76.         scanf("%d",&item);
77.         ptr->data = item;
78.         ptr->next = head;
79.         head = ptr;
```

```
80.         printf("\nNode inserted");
81.     }
82.
83. }
84. void lastinsert()
85. {
86.     struct node *ptr,*temp;
87.     int item;
88.     ptr = (struct node*)malloc(sizeof(struct node));
89.     if(ptr == NULL)
90.     {
91.         printf("\nOVERFLOW");
92.     }
93.     else
94.     {
95.         printf("\nEnter value?\n");
96.         scanf("%d",&item);
97.         ptr->data = item;
98.         if(head == NULL)
99.         {
100.                 ptr -> next = NULL;
101.                 head = ptr;
102.                 printf("\nNode inserted");
103.             }
104.             else
105.             {
106.                 temp = head;
107.                 while (temp -> next != NULL)
108.                 {
109.                     temp = temp -> next;
110.                 }
111.                 temp->next = ptr;
112.                 ptr->next = NULL;
113.                 printf("\nNode inserted");
```

```
114.
115.                 }
116.             }
117.         }
118.         void randominsert()
119.         {
120.             int i,loc,item;
121.             struct node *ptr, *temp;
122.             ptr = (struct node *) malloc (sizeof(struct node));
123.             if(ptr == NULL)
124.             {
125.                 printf("\nOVERFLOW");
126.             }
127.             else
128.             {
129.                 printf("\nEnter element value");
130.                 scanf("%d",&item);
131.                 ptr->data = item;
132.                 printf("\nEnter the location after which you want to insert ");
133.                 scanf("\n%d",&loc);
134.                 temp=head;
135.                 for(i=0;i<loc;i++)
136.                 {
137.                     temp = temp->next;
138.                     if(temp == NULL)
139.                     {
140.                         printf("\ncan't insert\n");
141.                         return;
142.                     }
143.
144.                 }
145.                 ptr ->next = temp ->next;
146.                 temp ->next = ptr;
147.                 printf("\nNode inserted");
```

```
148.            }
149.        }
150.        void begin_delete()
151.        {
152.            struct node *ptr;
153.            if(head == NULL)
154.            {
155.                printf("\nList is empty\n");
156.            }
157.            else
158.            {
159.                ptr = head;
160.                head = ptr->next;
161.                free(ptr);
162.                printf("\nNode deleted from the begining ...\n");
163.            }
164.        }
165.        void last_delete()
166.        {
167.            struct node *ptr,*ptr1;
168.            if(head == NULL)
169.            {
170.                printf("\nlist is empty");
171.            }
172.            else if(head -> next == NULL)
173.            {
174.                head = NULL;
175.                free(head);
176.                printf("\nOnly node of the list deleted ...\n");
177.            }
178.
179.            else
180.            {
181.                ptr = head;
```

```
182.             while(ptr->next != NULL)
183.             {
184.                 ptr1 = ptr;
185.                 ptr = ptr ->next;
186.             }
187.             ptr1->next = NULL;
188.             free(ptr);
189.             printf("\nDeleted Node from the last ...\n");
190.         }
191.     }
192.     void random_delete()
193.     {
194.         struct node *ptr,*ptr1;
195.         int loc,i;
196.         printf("\n Enter the location of the node after which you want to perform deletion \n");

197.         scanf("%d",&loc);
198.         ptr=head;
199.         for(i=0;i<loc;i++)
200.         {
201.             ptr1 = ptr;
202.             ptr = ptr->next;
203.
204.             if(ptr == NULL)
205.             {
206.                 printf("\nCan't delete");
207.                 return;
208.             }
209.         }
210.         ptr1 ->next = ptr ->next;
211.         free(ptr);
212.         printf("\nDeleted node %d ",loc+1);
213.     }
214.     void search()
```

```
{
    struct node *ptr;
    int item,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");
        }
    }

}
```

```
249.        void display()
250.        {
251.            struct node *ptr;
252.            ptr = head;
253.            if(ptr == NULL)
254.            {
255.                printf("Nothing to print");
256.            }
257.            else
258.            {
259.                printf("\nprinting values . . . . .\n");
260.                while (ptr!=NULL)
261.                {
262.                    printf("\n%d",ptr->data);
263.                    ptr = ptr -> next;
264.                }
265.            }
266.        }
267.
```

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
```

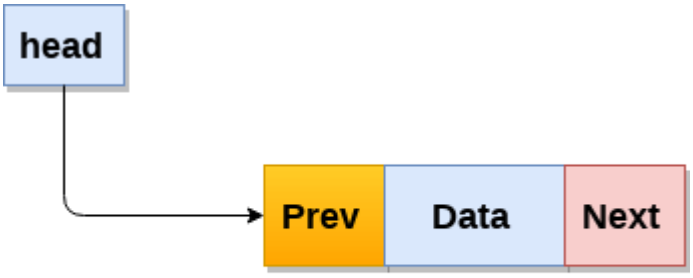
```
6.Delete node after specified location
7.Search for an element
8.Show
9.Exit

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
```

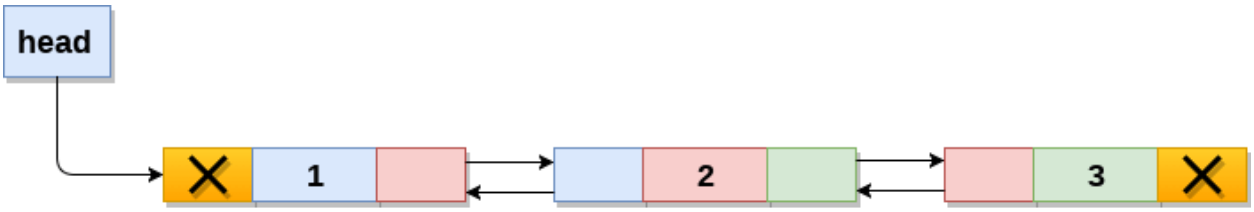
Doubly linked list

Doubly linked list is a complex type of linked list in which a node contains a pointer to the previous as well as the next node in the sequence. Therefore, in a doubly linked list, a node consists of three parts: node data, pointer to the next node in sequence (next pointer) , pointer to the previous node (previous pointer). A sample node in a doubly linked list is shown 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 :

```
1. struct node
2. {
3.     struct node *prev;
4.     int data;
5.     struct node *next;
6. }
```

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

In a singly linked list, we could traverse only in one direction, because each node contains address of the next node and it doesn't have any record of its previous nodes. However, doubly linked list overcome this limitation of singly linked list. Due to the fact that, each node of the list contains the address of its previous node, we can find all the details about the previous node as well by using the previous address stored inside the previous part of each node.

Memory Representation of a doubly linked list

Memory Representation of a doubly linked list is shown in the following image. 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 the following image, the first element of the list that is i.e. 13 stored at address 1. The head pointer points to the starting address 1. Since this is the first element being added to the list therefore the **prev** of the list **contains** null. The next node of the list resides at address 4 therefore the first node contains 4 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.

Memory Representation of a Doubly linked list

Operations on doubly linked list

Node Creation

```
1. struct node
2. {
3.     struct node *prev;
4.     int data;
5.     struct node *next;
6. };
7. 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.

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

```
1. #include<stdio.h>
2. #include<stdlib.h>
3. struct node
4. {
5.     struct node *prev;
6.     struct node *next;
```

```
7.      int data;
8.  };
9.  struct node *head;
10. void insertion_beginning();
11. void insertion_last();
12. void insertion_specified();
13. void deletion_beginning();
14. void deletion_last();
15. void deletion_specified();
16. void display();
17. void search();
18. void main ()
19. {
20. int choice =0;
21.     while(choice != 9)
22.     {
23.         printf("\n*********Main Menu*********\n");
24.         printf("\nChoose one option from the following list ...\n");
25.         printf("\n===============================================\n");
26.         printf("\n1.Insert in begining\n2.Insert at last\n3.Insert at any random location\n4.Delete fro
    m Beginning\n
27.         5.Delete from last\n6.Delete the node after the given data\n7.Search\n8.Show\n9.Exit\n");
28.         printf("\nEnter your choice?\n");
29.         scanf("\n%d",&choice);
30.         switch(choice)
31.         {
32.             case 1:
33.             insertion_beginning();
34.             break;
35.             case 2:
36.                     insertion_last();
37.             break;
38.             case 3:
39.             insertion_specified();
```

```
40.             break;
41.             case 4:
42.             deletion_beginning();
43.             break;
44.             case 5:
45.             deletion_last();
46.             break;
47.             case 6:
48.             deletion_specified();
49.             break;
50.             case 7:
51.             search();
52.             break;
53.             case 8:
54.             display();
55.             break;
56.             case 9:
57.             exit(0);
58.             break;
59.             default:
60.             printf("Please enter valid choice..");
61.         }
62.     }
63. }
64. void insertion_beginning()
65. {
66.     struct node *ptr;
67.     int item;
68.     ptr = (struct node *)malloc(sizeof(struct node));
69.     if(ptr == NULL)
70.     {
71.         printf("\nOVERFLOW");
72.     }
73.     else
```

```
74.    {
75.     printf("\nEnter Item value");
76.     scanf("%d",&item);
77.
78.    if(head==NULL)
79.    {
80.        ptr->next = NULL;
81.        ptr->prev=NULL;
82.        ptr->data=item;
83.        head=ptr;
84.    }
85.    else
86.    {
87.        ptr->data=item;
88.        ptr->prev=NULL;
89.        ptr->next = head;
90.        head->prev=ptr;
91.        head=ptr;
92.    }
93.    printf("\nNode inserted\n");
94. }
95.
96. }
97. void insertion_last()
98. {
99.    struct node *ptr,*temp;
100.            int item;
101.            ptr = (struct node *) malloc(sizeof(struct node));
102.            if(ptr == NULL)
103.            {
104.                printf("\nOVERFLOW");
105.            }
106.            else
107.            {
```

```
108.             printf("\nEnter value");
109.             scanf("%d",&item);
110.              ptr->data=item;
111.             if(head == NULL)
112.             {
113.                 ptr->next = NULL;
114.                 ptr->prev = NULL;
115.                 head = ptr;
116.             }
117.             else
118.             {
119.               temp = head;
120.               while(temp->next!=NULL)
121.               {
122.                   temp = temp->next;
123.               }
124.               temp->next = ptr;
125.               ptr ->prev=temp;
126.               ptr->next = NULL;
127.               }
128.
129.             }
130.           printf("\nnode inserted\n");
131.          }
132.       void insertion_specified()
133.       {
134.          struct node *ptr,*temp;
135.          int item,loc,i;
136.          ptr = (struct node *)malloc(sizeof(struct node));
137.          if(ptr == NULL)
138.          {
139.             printf("\n OVERFLOW");
140.          }
141.          else
```

```
142.         {
143.             temp=head;
144.             printf("Enter the location");
145.             scanf("%d",&loc);
146.             for(i=0;i<loc;i++)
147.             {
148.                 temp = temp->next;
149.                 if(temp == NULL)
150.                 {
151.                     printf("\n There are less than %d elements", loc);
152.                     return;
153.                 }
154.             }
155.             printf("Enter value");
156.             scanf("%d",&item);
157.             ptr->data = item;
158.             ptr->next = temp->next;
159.             ptr -> prev = temp;
160.             temp->next = ptr;
161.             temp->next->prev=ptr;
162.             printf("\nnode inserted\n");
163.         }
164.     }
165.     void deletion_beginning()
166.     {
167.         struct node *ptr;
168.         if(head == NULL)
169.         {
170.             printf("\n UNDERFLOW");
171.         }
172.         else if(head->next == NULL)
173.         {
174.             head = NULL;
175.             free(head);
```

```
176.             printf("\nnode deleted\n");
177.         }
178.         else
179.         {
180.             ptr = head;
181.             head = head -> next;
182.             head -> prev = NULL;
183.             free(ptr);
184.             printf("\nnode deleted\n");
185.         }
186.
187.     }
188.     void deletion_last()
189.     {
190.         struct node *ptr;
191.         if(head == NULL)
192.         {
193.             printf("\n UNDERFLOW");
194.         }
195.         else if(head->next == NULL)
196.         {
197.             head = NULL;
198.             free(head);
199.             printf("\nnode deleted\n");
200.         }
201.         else
202.         {
203.             ptr = head;
204.             if(ptr->next != NULL)
205.             {
206.                 ptr = ptr -> next;
207.             }
208.             ptr -> prev -> next = NULL;
209.             free(ptr);
```

```
210.             printf("\nnode deleted\n");
211.          }
212.       }
213.       void deletion_specified()
214.       {
215.          struct node *ptr, *temp;
216.          int val;
217.          printf("\n Enter the data after which the node is to be deleted : ");
218.          scanf("%d", &val);
219.          ptr = head;
220.          while(ptr -> data != val)
221.          ptr = ptr -> next;
222.          if(ptr -> next == NULL)
223.          {
224.             printf("\nCan't delete\n");
225.          }
226.          else if(ptr -> next -> next == NULL)
227.          {
228.             ptr ->next = NULL;
229.          }
230.          else
231.          {
232.             temp = ptr -> next;
233.             ptr -> next = temp -> next;
234.             temp -> next -> prev = ptr;
235.             free(temp);
236.             printf("\nnode deleted\n");
237.          }
238.       }
239.       void display()
240.       {
241.          struct node *ptr;
242.          printf("\n printing values...\n");
243.          ptr = head;
```

```
244.         while(ptr != NULL)
245.         {
246.             printf("%d\n",ptr->data);
247.             ptr=ptr->next;
248.         }
249.     }
250.     void search()
251.     {
252.         struct node *ptr;
253.         int item,i=0,flag;
254.         ptr = head;
255.         if(ptr == NULL)
256.         {
257.             printf("\nEmpty List\n");
258.         }
259.         else
260.         {
261.             printf("\nEnter item which you want to search?\n");
262.             scanf("%d",&item);
263.             while (ptr!=NULL)
264.             {
265.                 if(ptr->data == item)
266.                 {
267.                     printf("\nitem found at location %d ",i+1);
268.                     flag=0;
269.                     break;
270.                 }
271.                 else
272.                 {
273.                     flag=1;
274.                 }
275.                 i++;
276.                 ptr = ptr -> next;
277.             }
```

```
278.             if(flag==1)
279.             {
280.                 printf("\nItem not found\n");
281.             }
282.         }
283.
284.     }
```

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 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 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 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 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

*********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
```

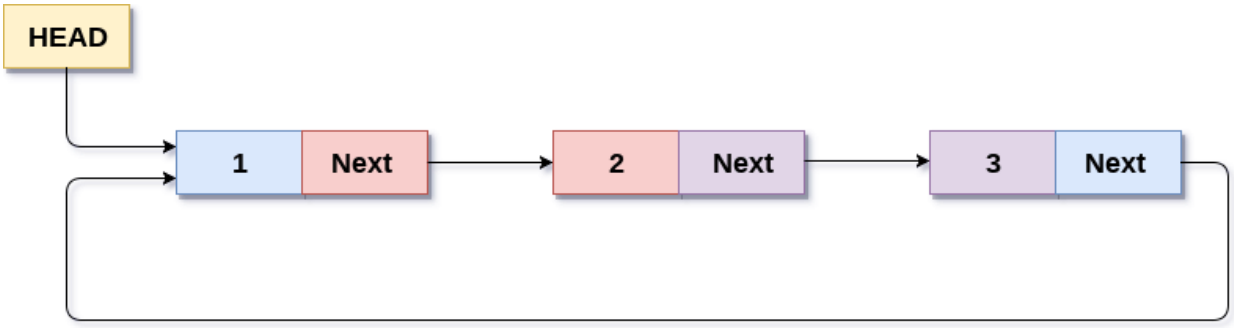
```
4.Delete from Beginning
5.Delete from last
6.Delete the node after the given data
7.Search
8.Show
9.Exit

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..
```

Circular Singly Linked List

In a circular Singly linked list, the last node of the list contains a pointer to the first node of the list. We can have circular singly linked list as well as circular doubly linked list.

We traverse a circular singly linked list until we reach the same node where we started. The circular singly liked list has no beginning and no ending. There is no null value present in the next part of any of the nodes.

The following image shows a circular singly linked list.



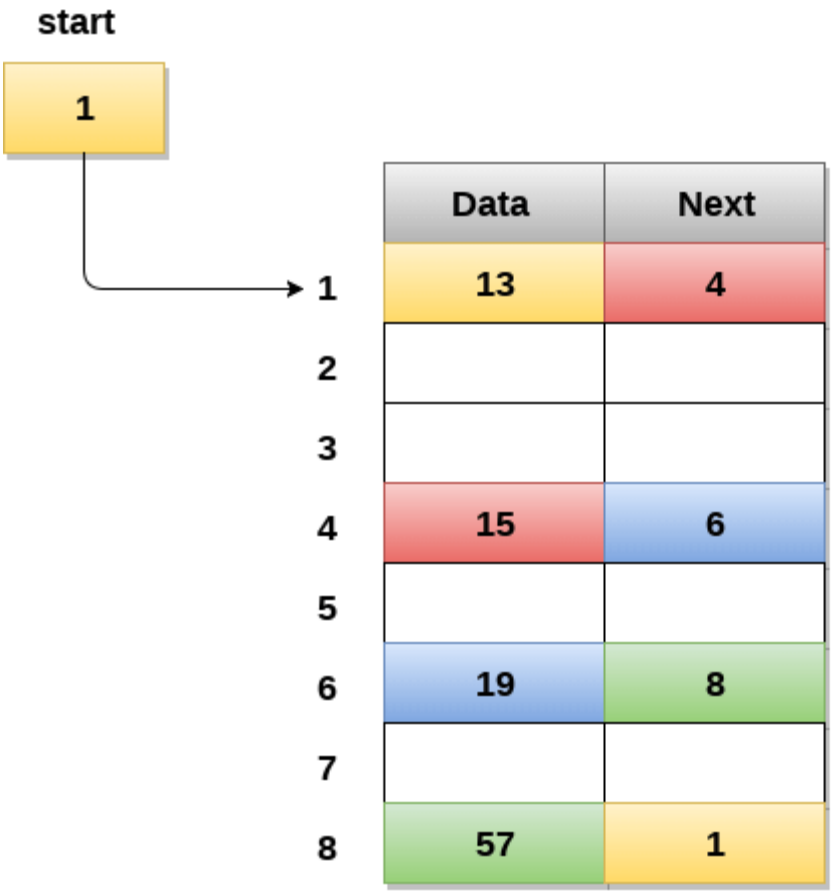
Circular Singly Linked List

Circular linked list are mostly used in task maintenance in operating systems. There are many examples where circular linked list are being used in computer science including browser surfing where a record of pages visited in the past by the user, is maintained in the form of circular linked lists and can be accessed again on clicking the previous button.

Memory Representation of circular linked list:

In the following image, memory representation of a circular linked list containing marks of a student in 4 subjects. However, the image shows a glimpse of how the circular list is being stored in the memory. The start or head of the list is pointing to the element with the index 1 and containing 13 marks in the data part and 4 in the next part. Which means that it is linked with the node that is being stored at 4th index of the list.

However, due to the fact that we are considering circular linked list in the memory therefore the last node of the list contains the address of the first node of the list.



Memory Representation of a circular linked list

We can also have more than one number of linked list in the memory with the different start pointers pointing to the different start nodes in the list. The last node is identified by its next part which contains the address of the start node of the list. We must be able to identify the last node of any linked list so that we can find out the number of iterations which need to be performed while traversing the list.

Operations on Circular Singly linked list:

Insertion

SN	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.

Deletion & Traversing

SN	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.

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

```
1.  #include<stdio.h>
2.  #include<stdlib.h>
3.  struct node
4.  {
5.      int data;
6.      struct node *next;
7.  };
8.  struct node *head;
9.
10. void beginsert ();
11. void lastinsert ();
12. void randominsert();
13. void begin_delete();
14. void last_delete();
15. void random_delete();
16. void display();
17. void search();
18. void main ()
19. {
20.     int choice =0;
21.     while(choice != 7)
22.     {
23.         printf("\n*********Main Menu*********\n");
24.         printf("\nChoose one option from the following list ...\n");
25.         printf("\n===============================================\n");
26.         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");
27.         printf("\nEnter your choice?\n");
28.         scanf("\n%d",&choice);
29.         switch(choice)
```

```
30.         {
31.             case 1:
32.             beginsert();
33.             break;
34.             case 2:
35.             lastinsert();
36.             break;
37.             case 3:
38.             begin_delete();
39.             break;
40.             case 4:
41.             last_delete();
42.             break;
43.             case 5:
44.             search();
45.             break;
46.             case 6:
47.             display();
48.             break;
49.             case 7:
50.             exit(0);
51.             break;
52.             default:
53.             printf("Please enter valid choice..");
54.         }
55.     }
56. }
57. void beginsert()
58. {
59.     struct node *ptr,*temp;
60.     int item;
61.     ptr = (struct node *)malloc(sizeof(struct node));
62.     if(ptr == NULL)
63.     {
```

```
64.         printf("\nOVERFLOW");
65.     }
66.     else
67.     {
68.         printf("\nEnter the node data?");
69.         scanf("%d",&item);
70.         ptr -> data = item;
71.         if(head == NULL)
72.         {
73.             head = ptr;
74.             ptr -> next = head;
75.         }
76.         else
77.         {
78.             temp = head;
79.             while(temp->next != head)
80.                 temp = temp->next;
81.             ptr->next = head;
82.             temp -> next = ptr;
83.             head = ptr;
84.         }
85.         printf("\nnode inserted\n");
86.     }
87.
88. }
89. void lastinsert()
90. {
91.     struct node *ptr,*temp;
92.     int item;
93.     ptr = (struct node *)malloc(sizeof(struct node));
94.     if(ptr == NULL)
95.     {
96.         printf("\nOVERFLOW\n");
97.     }
```

```
98.     else
99.     {
100.                printf("\nEnter Data?");
101.                scanf("%d",&item);
102.                ptr->data = item;
103.                if(head == NULL)
104.                {
105.                   head = ptr;
106.                   ptr -> next = head;
107.                }
108.                else
109.                {
110.                   temp = head;
111.                   while(temp -> next != head)
112.                   {
113.                      temp = temp -> next;
114.                   }
115.                   temp -> next = ptr;
116.                   ptr -> next = head;
117.                }
118.
119.                printf("\nnode inserted\n");
120.             }
121.
122.          }
123.
124.          void begin_delete()
125.          {
126.             struct node *ptr;
127.             if(head == NULL)
128.             {
129.                printf("\nUNDERFLOW");
130.             }
131.             else if(head->next == head)
```

```
132.         {
133.             head = NULL;
134.             free(head);
135.             printf("\nnode deleted\n");
136.         }
137.
138.         else
139.         {   ptr = head;
140.             while(ptr -> next != head)
141.                 ptr = ptr -> next;
142.             ptr->next = head->next;
143.             free(head);
144.             head = ptr->next;
145.             printf("\nnode deleted\n");
146.
147.         }
148.     }
149.     void last_delete()
150.     {
151.         struct node *ptr, *preptr;
152.         if(head==NULL)
153.         {
154.             printf("\nUNDERFLOW");
155.         }
156.         else if (head ->next == head)
157.         {
158.             head = NULL;
159.             free(head);
160.             printf("\nnode deleted\n");
161.
162.         }
163.         else
164.         {
165.             ptr = head;
```

```
166.             while(ptr ->next != head)
167.             {
168.                preptr=ptr;
169.                ptr = ptr->next;
170.             }
171.             preptr->next = ptr -> next;
172.             free(ptr);
173.             printf("\nnode deleted\n");
174.
175.          }
176.       }
177.
178.       void search()
179.       {
180.          struct node *ptr;
181.          int item,i=0,flag=1;
182.          ptr = head;
183.          if(ptr == NULL)
184.          {
185.             printf("\nEmpty List\n");
186.          }
187.          else
188.          {
189.             printf("\nEnter item which you want to search?\n");
190.             scanf("%d",&item);
191.             if(head ->data == item)
192.             {
193.             printf("item found at location %d",i+1);
194.             flag=0;
195.             }
196.             else
197.             {
198.             while (ptr->next != head)
199.             {
```

```
200.                 if(ptr->data == item)
201.                 {
202.                     printf("item found at location %d ",i+1);
203.                     flag=0;
204.                     break;
205.                 }
206.                 else
207.                 {
208.                     flag=1;
209.                 }
210.                 i++;
211.                 ptr = ptr -> next;
212.             }
213.             }
214.             if(flag != 0)
215.             {
216.                 printf("Item not found\n");
217.             }
218.         }
219.
220.     }
221.
222.     void display()
223.     {
224.         struct node *ptr;
225.         ptr=head;
226.         if(head == NULL)
227.         {
228.             printf("\nnothing to print");
229.         }
230.         else
231.         {
232.             printf("\n printing values ... \n");
233.
```

```
234.             while(ptr -> next != head)
235.             {
236.
237.                 printf("%d\n", ptr -> data);
238.                 ptr = ptr -> next;
239.             }
240.             printf("%d\n", ptr -> data);
241.         }
242.
243.     }
```

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 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?
```

```
2

Enter Data?20

node inserted

*********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?
2

Enter Data?30

node inserted

*********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?
3

node deleted

*********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
```

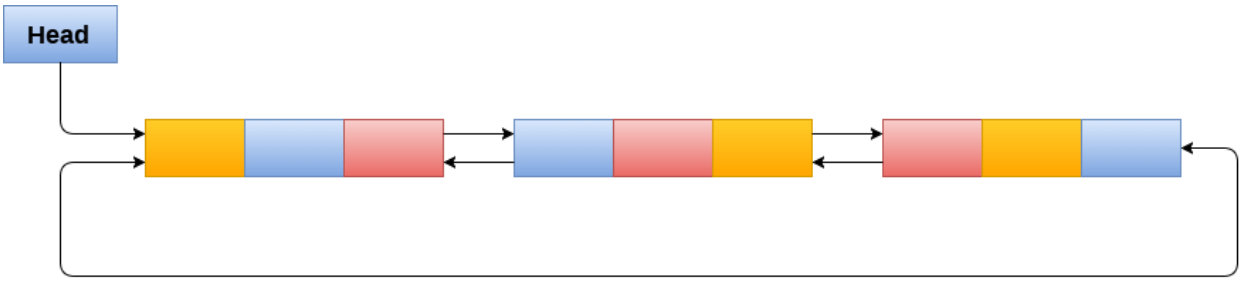
```
6.Show
7.Exit

Enter your choice?
4

node deleted

*********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?
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 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?
6

 printing values ...
20

*********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?
7
```

Circular Doubly Linked List

Circular doubly linked list is a more complexed type of data structure in which a node contain pointers to its previous node as well as the next node. Circular doubly linked list doesn't contain NULL in any of the node. The last node of the list contains the address of the first node of the list. The first node of the list also contain address of the last node in its previous pointer.

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



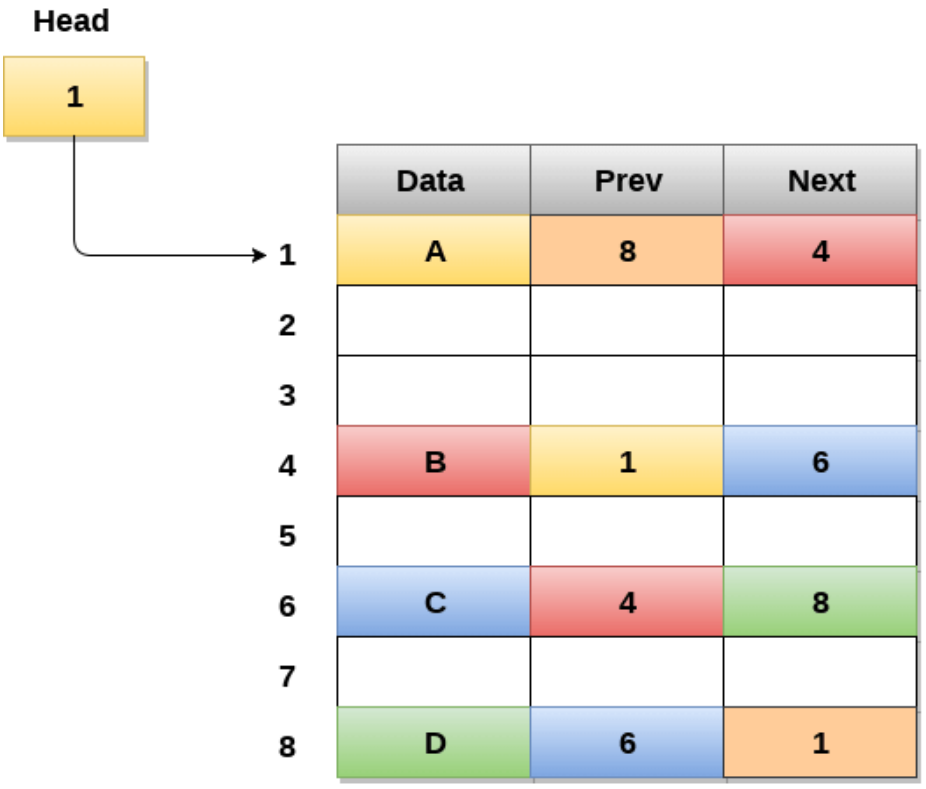
Circular Doubly Linked List

Due to the fact that a circular doubly linked list contains three parts in its structure therefore, it demands more space per node and more expensive basic operations. However, a circular doubly linked list provides easy manipulation of the pointers and the searching becomes twice as efficient.

Memory Management of Circular Doubly linked list

The following figure shows the way in which the memory is allocated for a circular doubly linked list. The variable head contains the address of the first element of the list i.e. 1 hence the starting node of the list contains data A is stored at address 1. Since,

each node of the list is supposed to have three parts therefore, the starting node of the list contains address of the last node i.e. 8 and the next node i.e. 4. The last node of the list that is stored at address 8 and containing data as 6, contains address of the first node of the list as shown in the image i.e. 1. In circular doubly linked list, the last node is identified by the address of the first node which is stored in the next part of the last node therefore the node which contains the address of the first node, is actually the last node of the list.



Memory Representation of a Circular Doubly linked list

Operations on circular doubly linked list :

There are various operations which can be performed on circular doubly linked list. The node structure of a circular doubly linked list is similar to doubly linked list. However, the operations on circular doubly linked list is described in the following 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.

C program to implement all the operations on circular doubly linked list

```
1. #include<stdio.h>
2. #include<stdlib.h>
3. struct node
4. {
5.     struct node *prev;
6.     struct node *next;
7.     int data;
8. };
9. struct node *head;
10. void insertion_beginning();
11. void insertion_last();
12. void deletion_beginning();
13. void deletion_last();
14. void display();
15. void search();
16. void main ()
17. {
18. int choice =0;
```

```
19.    while(choice != 9)
20.    {
21.        printf("\n*********Main Menu*********\n");
22.        printf("\nChoose one option from the following list ...\n");
23.        printf("\n==========================================\n");
24.        printf("\n1.Insert in Beginning\n2.Insert at last\n3.Delete from Beginning\n4.Delete from las
    t\n5.Search\n6.Show\n7.Exit\n");
25.        printf("\nEnter your choice?\n");
26.        scanf("\n%d",&choice);
27.        switch(choice)
28.        {
29.            case 1:
30.            insertion_beginning();
31.            break;
32.            case 2:
33.                  insertion_last();
34.            break;
35.            case 3:
36.            deletion_beginning();
37.            break;
38.            case 4:
39.            deletion_last();
40.            break;
41.            case 5:
42.            search();
43.            break;
44.            case 6:
45.            display();
46.            break;
47.            case 7:
48.            exit(0);
49.            break;
50.            default:
51.            printf("Please enter valid choice..");
```

```
52.         }
53.     }
54. }
55. void insertion_beginning()
56. {
57.    struct node *ptr,*temp;
58.    int item;
59.    ptr = (struct node *)malloc(sizeof(struct node));
60.    if(ptr == NULL)
61.    {
62.        printf("\nOVERFLOW");
63.    }
64.    else
65.    {
66.     printf("\nEnter Item value");
67.     scanf("%d",&item);
68.     ptr->data=item;
69.    if(head==NULL)
70.    {
71.       head = ptr;
72.       ptr -> next = head;
73.       ptr -> prev = head;
74.    }
75.    else
76.    {
77.        temp = head;
78.     while(temp -> next != head)
79.     {
80.         temp = temp -> next;
81.     }
82.     temp -> next = ptr;
83.     ptr -> prev = temp;
84.     head -> prev = ptr;
85.     ptr -> next = head;
```

```
86.     head = ptr;
87.    }
88.    printf("\nNode inserted\n");
89. }
90.
91. }
92. void insertion_last()
93. {
94.    struct node *ptr,*temp;
95.    int item;
96.    ptr = (struct node *) malloc(sizeof(struct node));
97.    if(ptr == NULL)
98.    {
99.        printf("\nOVERFLOW");
100.            }
101.            else
102.            {
103.                printf("\nEnter value");
104.                scanf("%d",&item);
105.                 ptr->data=item;
106.                if(head == NULL)
107.                {
108.                    head = ptr;
109.                    ptr -> next = head;
110.                    ptr -> prev = head;
111.                }
112.                else
113.                {
114.                  temp = head;
115.                  while(temp->next !=head)
116.                  {
117.                      temp = temp->next;
118.                  }
119.                  temp->next = ptr;
```

```
                ptr ->prev=temp;
                head -> prev = ptr;
              ptr -> next = head;
                }
            }
             printf("\nnode inserted\n");
        }

        void deletion_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;
            }

```

```
154.        }
155.        void deletion_last()
156.        {
157.            struct node *ptr;
158.            if(head == NULL)
159.            {
160.                printf("\n UNDERFLOW");
161.            }
162.            else if(head->next == head)
163.            {
164.                head = NULL;
165.                free(head);
166.                printf("\nnode deleted\n");
167.            }
168.            else
169.            {
170.                ptr = head;
171.                if(ptr->next != head)
172.                {
173.                    ptr = ptr -> next;
174.                }
175.                ptr -> prev -> next = head;
176.                head -> prev = ptr -> prev;
177.                free(ptr);
178.                printf("\nnode deleted\n");
179.            }
180.        }
181.
182.        void display()
183.        {
184.            struct node *ptr;
185.            ptr=head;
186.            if(head == NULL)
187.            {
```

```
188.             printf("\nnothing to print");
189.         }
190.         else
191.         {
192.             printf("\n printing values ... \n");
193.
194.             while(ptr -> next != head)
195.             {
196.
197.                 printf("%d\n", ptr -> data);
198.                 ptr = ptr -> next;
199.             }
200.             printf("%d\n", ptr -> data);
201.         }
202.
203.     }
204.
205.     void search()
206.     {
207.         struct node *ptr;
208.         int item,i=0,flag=1;
209.         ptr = head;
210.         if(ptr == NULL)
211.         {
212.             printf("\nEmpty List\n");
213.         }
214.         else
215.         {
216.             printf("\nEnter item which you want to search?\n");
217.             scanf("%d",&item);
218.             if(head ->data == item)
219.             {
220.             printf("item found at location %d",i+1);
221.             flag=0;
```

```
222.             }
223.             else
224.             {
225.             while (ptr->next != head)
226.             {
227.                if(ptr->data == item)
228.                {
229.                   printf("item found at location %d ",i+1);
230.                   flag=0;
231.                   break;
232.                }
233.                else
234.                {
235.                   flag=1;
236.                }
237.                i++;
238.                ptr = ptr -> next;
239.             }
240.             }
241.             if(flag != 0)
242.             {
243.                printf("Item not found\n");
244.             }
245.          }
246.
247.       }
```

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
```