Aim: Linked List.

Objectives: The main objective is to understand and implement the linked list data structure and its fundamental operations such as insertion, deletion, traversal, searching, sorting and display in C++.

Tools Used: Visual Studio Code.

Concept:

A linked list is a linear data structure in which elements (called nodes) are stored non-contiguously in memory. Each node consists of two parts:

1. Data: The value or information stored in the node.
2. Next Pointer: A reference to the next node in the sequence.

Types of Linked Lists

1. Singly Linked List: Each node points to the next node, and the last node points to NULL.
   Example: 10 -> 20 -> 30 -> NULL.
2. Doubly Linked List: Each node contains two pointers, one pointing to the next node and one to the previous node.
   Example: NULL <- 10 <-> 20 <-> 30 -> NULL.
3. Circular Linked List: The last node points back to the first node, forming a circle.
   Example: 10 -> 20 -> 30 -> 10.

Operation

1. Insertion

Insertion involves adding a new node at a specific position in the linked list.
Example:
Initial List: 10 -> 20 -> 30 -> NULL
Insert 15 after 10:
Algorithm:

- Create a new node with data 15.
- Update the next pointer of node 10 to the new node.
- Set the next pointer of the new node to node 20.
  Result: 10 -> 15 -> 20 -> 30 -> NULL.

2. Deletion

Deletion involves removing a node from the linked list.
Example:
Initial List: 10 -> 20 -> 30 -> NULL

Delete node 20:
Algorithm:

- Update the next pointer of node 10 to point to node 30.
- Free the memory of node 20.
  Result: 10 -> 30 -> NULL.

3. Display

Display involves visiting each node sequentially.
Example:
List: 10 -> 20 -> 30 -> NULL
Algorithm:

- Start at the list node (10).
- Visit each node in sequence, printing its data: 10 20 30.

4. Search

Searching involves finding whether a specific value exists in the list.
Example:
List: 10 -> 20 -> 30 -> NULL
Search for 20:
Algorithm:

- Start at the list node (10).
- Compare each node’s data with the target (20).
- Stop when a match is found or the end is reached.
  Output: Found at position 2.

5. Sorting

Sorting in a linked list involves arranging the elements in a specific order, such as ascending or descending. The goal is to rearrange the nodes in such a way that the data in the nodes follow a particular order.

Algorithm:

- Start from the list node of the list.
- Iterate through the list: Compare the current node's data with the other nodes to find the correct position for the current node's data.
- Find the minimum or maximum value: Depending on the sorting order (ascending or descending), identify the node with the smallest or largest value in the unsorted portion of the list.
- Swap the data: Swap the data between the current node and the identified node (minimum or maximum).
- Move to the next node and repeat the process until the entire list is sorted.

6. Reverse

Reversing a linked list involves changing the direction of the pointers so that the last node becomes the first node, and the first node becomes the last.

Algorithm:

- Initialize three pointers: prev (set to NULL), current (set to the list of the list), and next (set to NULL).
- Iterate through the list:
    - For each node, store the next node (next = current->next).
    - Change the current node's next pointer to point to prev (this reverses the direction of the pointer).
    - Move prev to the current node, and move current to the next node (the one stored in next).
- Repeat the process until the entire list is traversed and reversed.
- Update the list pointer to the prev node, which now points to the new list of the list.

Problem Statement:

Implement

1) Insert at the Beginning
2) Insert at the End
3) Insert Before a Particular Element
4) Insert After a Particular Element
5) Delete at the Beginning
6) Delete at the End
7) Delete a Particular Element
8) Sort the Linked List
9) Reverse the Linked List
10) Display
11) Search a particular

Using Singly, Doubly and Circular LinkedList.

Solution:

1.) Singly Linked List.

```
#include <iostream>

using namespace std;

int value, ch, key, target;

struct node

{
```

```
    int data;
    struct node *next;
}
    *list = NULL,
    *p, *q, *r, *s, *temp;
class LinkedList
{
public:
    void menu()
    {
        do
        {
            cout << endl;
            cout << "Enter you choice:" << endl;
            cout << "1) Insert at the Beginning" << endl;
            cout << "2) Insert at the End" << endl;
            cout << "3) Insert Before a Particular Element" << endl;
            cout << "4) Insert After a Particular Element" << endl;
            cout << "5) Delete at the Beginning" << endl;
            cout << "6) Delete at the End" << endl;
            cout << "7) Delete a Particular Element" << endl;
            cout << "8) Sort the Linked List" << endl;
            cout << "9) Reverse the Linked List" << endl;
            cout << "10) Display" << endl;
            cout << "11) Search a particular " << endl;
            cout << "15) Exit" << endl;
            cout << endl;

            cin >> ch;
            switch (ch)
            {
            case 1:
```

```
            insertAtBeginning();
            break;
        case 2:
            insertAtEnd();
            break;
        case 3:
            insertBeforeParticularElement();
            break;
        case 4:
            insertAfterParticularElement();
            break;
        case 5:
            deleteAtBeginning();
            break;
        case 6:
            deleteAtEnd();
            break;
        case 7:
            deleteAParticularElement();
            break;
        case 8:
            sortLinkedList();
            break;
        case 9:
            reverse();
            break;

        case 10:
            display();
            break;
        case 11:
            Search();
```

```
            break;

        default:
            cout << "Enter a proper Value";
            break;
        }

    } while (ch != 15);
}

void insertAtBeginning()
{
    p = (struct node *)malloc(sizeof(node));
    cout << "Enter the element you want to Insert:" << endl;
    cin >> key;
    p->data = key;
    if (list == NULL)
    {
        p->next = NULL;
    }
    else
    {
        p->next = list;
    }
    list = p;
}

void insertAtEnd()
{
    p = (struct node *)malloc(sizeof(node));
    cout << "Enter the element you want to Insert:" << endl;
    cin >> key;
```

```
    p->data = key;
    p->next = NULL;
    if (list == NULL)
    {
        cout << "List is Empty Inserting, so Inserting at the Start" << endl;
        list = p;
    }
    else
    {
        q = list;
        while (q->next != NULL)
        {
            q = q->next;
        }
        q->next = p;
    }
}

void insertBeforeParticularElement()
{
    if (list == NULL)
    {
        cout << "List is Empty" << endl;
    }
    else
    {
        cout << "Enter the Element before which you want to Insert the Element" << endl;
        cin >> target;
        if (list->data == target)
        {
            p = (struct node *)malloc(sizeof(node));
            cout << "Enter the element you want to insert: ";
```

```
            cin >> key;
            p->data = key;
            p->next = list;
            list = p;
            return;
        }

        q = list;
        while (q != NULL && q->data != target)
        {
            r = q;
            q = q->next;
        }
        if (q != NULL && q->data == target)
        {
            struct node *p = (struct node *)malloc(sizeof(node));
            cout << "Enter the element you want to insert: ";
            cin >> key;
            p->data = key;
            p->next = q;
            r->next = p;
        }
        else
        {
            cout << "Element not found!" << endl;
        }
    }
}

void insertAfterParticularElement()
{
    if (list == NULL)
```

```
    {
        cout << "List is Empty" << endl;
        return;
    }

    cout << "Enter the element after which you want to insert the new element: ";
    cin >> target;

    r = list;

    while (r != NULL && r->data != target)
    {
        r = r->next;
    }

    if (r != NULL)
    {
        p = (struct node *)malloc(sizeof(node));
        cout << "Enter the element you want to insert: ";
        cin >> key;
        p->data = key;
        p->next = r->next;
        r->next = p;
    }
    else
    {
        cout << "Element not found!" << endl;
    }
}

void deleteAtBeginning()
{
```

```
    if (list == NULL)
    {
        cout << "Linked List is Empty" << endl;
    }
    else
    {
        q = list;
        if (q->next != NULL)
        {
            cout << "Deleted " << q->data << endl;
            list = q->next;
        }
        else
        {
            cout << "Deleted " << q->data << endl;
            list = NULL;
        }
        free(q);
    }
}

void deleteAtEnd()
{
    if (list == NULL)
    {
        cout << "Linked List is Empty" << endl;
    }
    else
    {
        q = list;
        if (q->next == NULL)
        {
```

```
            cout << "Deleted " << q->data << endl;
            list = NULL;
        }
        else
        {
            while (q->next != NULL)
            {
                r = q;
                q = q->next;
            }
            cout << "Deleted " << q->data << endl;
            r->next = NULL;
        }
        free(q);
    }
}

void deleteAParticularElement()
{
    if (list == NULL)
    {
        cout << "Linked List is Empty" << endl;
    }
    else
    {
        cout << "Enter the Element which you want to Delete" << endl;
        cin >> target;
        q = list;
        while (q->next != NULL && q->data != target)
        {
            r = q;
            q = q->next;
```

```
        }
        if (q->next != NULL)
        {
            cout << "Deleted " << q->data << endl;
            r->next = q->next;
        }
        else
        {
            cout << "Deleted " << q->data << endl;
            r->next = NULL;
        }
        free(q);
    }
}

void sortLinkedList()
{
    if (list == NULL)
    {
        cout << "Linked List is Empty" << endl;
    }
    q = list;

    while (q != NULL)
    {
        r = q->next;
        while (r != NULL)
        {
            if (q->data > r->data)
            {
                int temp = q->data;
                q->data = r->data;
```

```
            r->data = temp;
          }

          r = r->next;
        }
        q = q->next;
    }

    cout << "The Sorted Linked List is :" << endl;
    display();
}

  void reverse()
  {
    if (list == NULL)
    {
      cout << "Linked List is Empty" << endl;
    }

    q = s = list;
    temp = NULL;
    r = q->next;
    while (r != NULL)
    {
      temp = q;
      q = r;
      r = q->next;
      q->next = temp;
    }

    list = q;
```

```
    s->next = NULL;
}

void display()
{
    if (list == NULL)
    {
        cout << "Linked List is Empty" << endl;
    }
    else
    {
        q = list;
        while (q != NULL)
        {
            cout << q->data << "-->";
            q = q->next;
        }
    }
}

void Search()
{
    if (list == NULL)
    {
        cout << "List is Empty" << endl;
    }
    else
    {
        cout << "Enter the Element you want to Search" << endl;
        cin >> target;
        if (list->data == target)
        {
```

```
            cout << "Element Found at 0th Index ";
            return;
        }

        q = list;
        int i = 0;
        while (q != NULL && q->data != target)
        {
            q = q->next;
            i++;
        }
        if (q != NULL && q->data == target)
        {
            cout << "Element Found at " << i << "th Index";
        }
        else
        {
            cout << "Element not found!" << endl;
        }
      }
   }
};
int main()
{
   LinkedList ll;
   ll.menu();
   return 0;
}
```

Output:

```
Enter you choice:
1) Insert at the Beginning
2) Insert at the End
```

```
3) Insert Before a Particular Element
4) Insert After a Particular Element
5) Delete at the Beginning
6) Delete at the End
7) Delete a Particular Element
8) Sort the Linked List
9) Reverse the Linked List
10) Display
11) Search a particular
15) Exit

1
Enter the element you want to Insert:
10

Enter you choice:
1) Insert at the Beginning
2) Insert at the End
3) Insert Before a Particular Element
4) Insert After a Particular Element
5) Delete at the Beginning
6) Delete at the End
7) Delete a Particular Element
8) Sort the Linked List
9) Reverse the Linked List
10) Display
11) Search a particular
15) Exit

1
Enter the element you want to Insert:
20

Enter you choice:
1) Insert at the Beginning
2) Insert at the End
3) Insert Before a Particular Element
4) Insert After a Particular Element
5) Delete at the Beginning
6) Delete at the End
7) Delete a Particular Element
8) Sort the Linked List
9) Reverse the Linked List
10) Display
11) Search a particular
15) Exit

1
```

```
Enter the element you want to Insert:
30

Enter you choice:
1) Insert at the Beginning
2) Insert at the End
3) Insert Before a Particular Element
4) Insert After a Particular Element
5) Delete at the Beginning
6) Delete at the End
7) Delete a Particular Element
8) Sort the Linked List
9) Reverse the Linked List
10) Display
11) Search a particular
15) Exit

10
30-->20-->10-->
Enter you choice:
1) Insert at the Beginning
2) Insert at the End
3) Insert Before a Particular Element
4) Insert After a Particular Element
5) Delete at the Beginning
6) Delete at the End
7) Delete a Particular Element
8) Sort the Linked List
9) Reverse the Linked List
10) Display
11) Search a particular
15) Exit

2
Enter the element you want to Insert:
5

Enter you choice:
1) Insert at the Beginning
2) Insert at the End
3) Insert Before a Particular Element
4) Insert After a Particular Element
5) Delete at the Beginning
6) Delete at the End
7) Delete a Particular Element
8) Sort the Linked List
9) Reverse the Linked List
10) Display
```

```
11) Search a particular
15) Exit

10
30-->20-->10-->5-->
Enter you choice:
1) Insert at the Beginning
2) Insert at the End
3) Insert Before a Particular Element
4) Insert After a Particular Element
5) Delete at the Beginning
6) Delete at the End
7) Delete a Particular Element
8) Sort the Linked List
9) Reverse the Linked List
10) Display
11) Search a particular
15) Exit

3
Enter the Element before which you want to Insert the Element
10
Enter the element you want to insert: 15

Enter you choice:
1) Insert at the Beginning
2) Insert at the End
3) Insert Before a Particular Element
4) Insert After a Particular Element
5) Delete at the Beginning
6) Delete at the End
7) Delete a Particular Element
8) Sort the Linked List
9) Reverse the Linked List
10) Display
11) Search a particular
15) Exit

10
30-->20-->15-->10-->5-->
Enter you choice:
1) Insert at the Beginning
2) Insert at the End
3) Insert Before a Particular Element
4) Insert After a Particular Element
5) Delete at the Beginning
6) Delete at the End
7) Delete a Particular Element
```

```
8) Sort the Linked List
9) Reverse the Linked List
10) Display
11) Search a particular
15) Exit

4
Enter the element after which you want to insert the new element: 30
Enter the element you want to insert: 25

Enter you choice:
1) Insert at the Beginning
2) Insert at the End
3) Insert Before a Particular Element
4) Insert After a Particular Element
5) Delete at the Beginning
6) Delete at the End
7) Delete a Particular Element
8) Sort the Linked List
9) Reverse the Linked List
10) Display
11) Search a particular
15) Exit

10
30-->25-->20-->15-->10-->5-->
Enter you choice:
1) Insert at the Beginning
2) Insert at the End
3) Insert Before a Particular Element
4) Insert After a Particular Element
5) Delete at the Beginning
6) Delete at the End
7) Delete a Particular Element
8) Sort the Linked List
9) Reverse the Linked List
10) Display
11) Search a particular
15) Exit

5
Deleted 30

Enter you choice:
1) Insert at the Beginning
2) Insert at the End
3) Insert Before a Particular Element
4) Insert After a Particular Element
```

```
5) Delete at the Beginning
6) Delete at the End
7) Delete a Particular Element
8) Sort the Linked List
9) Reverse the Linked List
10) Display
11) Search a particular
15) Exit

10
25-->20-->15-->10-->5-->
Enter you choice:
1) Insert at the Beginning
2) Insert at the End
3) Insert Before a Particular Element
4) Insert After a Particular Element
5) Delete at the Beginning
6) Delete at the End
7) Delete a Particular Element
8) Sort the Linked List
9) Reverse the Linked List
10) Display
11) Search a particular
15) Exit

6
Deleted 5

Enter you choice:
1) Insert at the Beginning
2) Insert at the End
3) Insert Before a Particular Element
4) Insert After a Particular Element
5) Delete at the Beginning
6) Delete at the End
7) Delete a Particular Element
8) Sort the Linked List
9) Reverse the Linked List
10) Display
11) Search a particular
15) Exit

10
25-->20-->15-->10-->
Enter you choice:
1) Insert at the Beginning
2) Insert at the End
3) Insert Before a Particular Element
```

```
4) Insert After a Particular Element
5) Delete at the Beginning
6) Delete at the End
7) Delete a Particular Element
8) Sort the Linked List
9) Reverse the Linked List
10) Display
11) Search a particular
15) Exit

7
Enter the Element which you want to Delete
15
Deleted 15

Enter you choice:
1) Insert at the Beginning
2) Insert at the End
3) Insert Before a Particular Element
4) Insert After a Particular Element
5) Delete at the Beginning
6) Delete at the End
7) Delete a Particular Element
8) Sort the Linked List
9) Reverse the Linked List
10) Display
11) Search a particular
15) Exit

10
25-->20-->10-->
Enter you choice:
1) Insert at the Beginning
2) Insert at the End
3) Insert Before a Particular Element
4) Insert After a Particular Element
5) Delete at the Beginning
6) Delete at the End
7) Delete a Particular Element
8) Sort the Linked List
9) Reverse the Linked List
10) Display
11) Search a particular
15) Exit

8
The Sorted Linked List is :
10-->20-->25-->
```

```
Enter you choice:
1) Insert at the Beginning
2) Insert at the End
3) Insert Before a Particular Element
4) Insert After a Particular Element
5) Delete at the Beginning
6) Delete at the End
7) Delete a Particular Element
8) Sort the Linked List
9) Reverse the Linked List
10) Display
11) Search a particular
15) Exit

9

Enter you choice:
1) Insert at the Beginning
2) Insert at the End
3) Insert Before a Particular Element
4) Insert After a Particular Element
5) Delete at the Beginning
6) Delete at the End
7) Delete a Particular Element
8) Sort the Linked List
9) Reverse the Linked List
10) Display
11) Search a particular
15) Exit

10
25-->20-->10-->
Enter you choice:
1) Insert at the Beginning
2) Insert at the End
3) Insert Before a Particular Element
4) Insert After a Particular Element
5) Delete at the Beginning
6) Delete at the End
7) Delete a Particular Element
8) Sort the Linked List
9) Reverse the Linked List
10) Display
11) Search a particular
15) Exit

11
Enter the Element you want to Search
```

```
20
Element Found at 1th Index
Enter you choice:
1) Insert at the Beginning
2) Insert at the End
3) Insert Before a Particular Element
4) Insert After a Particular Element
5) Delete at the Beginning
6) Delete at the End
7) Delete a Particular Element
8) Sort the Linked List
9) Reverse the Linked List
10) Display
11) Search a particular
15) Exit
```

2.) Doubly Linked List.

```
#include <iostream>

using namespace std;

int value, ch, key, target;
struct node
{
   struct node *prev;
   int data;
   struct node *next;
}
   *list = NULL,
   *p, *q, *r, *s, *temp;
class DoublyLinkedList
{
public:
   void menu()
   {
      do
      {
         cout << endl;
         cout << "Enter you choice:" << endl;
         cout << "1) Insert at the Beginning" << endl;
         cout << "2) Insert at the End" << endl;
         cout << "3) Insert Before a Particular Element" << endl;
         cout << "4) Insert After a Particular Element" << endl;
         cout << "5) Delete at the Beginning" << endl;
         cout << "6) Delete at the End" << endl;
         cout << "7) Delete a Particular Element" << endl;
         cout << "8) Sort the Linked List" << endl;
         cout << "9) Reverse the Linked List" << endl;
```

```
cout << "10) Display" << endl;
cout << "11) Search a particular " << endl;
cout << "15) Exit" << endl;
cout << endl;

cin >> ch;
switch (ch)
{
case 1:
   insertAtBeginning();
   break;
case 2:
   insertAtEnd();
   break;
case 3:
   insertBeforeParticularElement();
   break;
case 4:
   insertAfterParticularElement();
   break;
case 5:
   deleteAtBeginning();
   break;
case 6:
   deleteAtEnd();
   break;
case 7:
   deleteAParticularElement();
   break;
case 8:
   sortLinkedList();
   break;
```

```
        case 9:
            reverse();
            break;
        case 10:
            display();
            break;
        case 11:
            Search();
            break;

        default:
            cout << "Enter a proper Value";
            break;
        }

    } while (ch != 15);
}

void insertAtBeginning()
{
    p = (struct node *)malloc(sizeof(node));
    cout << "Enter the element you want to Insert:" << endl;
    cin >> key;
    p->data = key;
    p->prev = NULL;

    if (list == NULL)
    {
        p->next = NULL;
    }
    else
    {
```

```
        p->next = list;
        list->prev = p;
    }

    list = p;
}

void insertAtEnd()
{
    p = (struct node *)malloc(sizeof(node));
    cout << "Enter the element you want to Insert:" << endl;
    cin >> key;
    p->data = key;

    p->next = NULL;
    if (list == NULL)
    {
        cout << "List is Empty Inserting, so Inserting at the Start" << endl;
        list = p;
        list->prev = NULL;
    }
    else
    {
        q = list;
        while (q->next != NULL)
        {
            // cout << q->data << endl;
            q = q->next;
        }
        q->next = p;
        p->prev = q;
    }
```

```
}

void insertBeforeParticularElement()
{
   if (list == NULL)
   {
      cout << "List is Empty" << endl;
   }
   else
   {
      cout << "Enter the Element before which you want to Insert the Element" << endl;
      cin >> target;
      if (list->data == target)
      {
         p = (struct node *)malloc(sizeof(node));
         cout << "Enter the element you want to insert: ";
         cin >> key;
         p->data = key;
         p->next = list;
         p->prev = NULL;
         list->prev = p;
         list = p;
         return;
      }

      q = list;
      while (q != NULL && q->data != target)
      {
         r = q;
         q = q->next;
      }
      if (q != NULL && q->data == target)
```

```
        {
            struct node *p = (struct node *)malloc(sizeof(node));
            cout << "Enter the element you want to insert: ";
            cin >> key;
            p->data = key;
            p->next = q;
            p->prev = q->prev;

            if (q->prev != NULL)
            {
                q->prev->next = p;
            }
            q->prev = p;
        }
        else
        {
            cout << "Element not found!" << endl;
        }
    }
}

void insertAfterParticularElement()
{
    if (list == NULL)
    {
        cout << "List is Empty" << endl;
        return;
    }

    cout << "Enter the element after which you want to insert the new element: ";
    cin >> target;
```

```
    q = list;
    while (q != NULL && q->data != target)
    {
        q = q->next;
    }

    if (q != NULL)
    {

        p = new node;
        cout << "Enter the element you want to insert: ";
        cin >> key;

        p->data = key;

        p->prev = q;
        p->next = q->next;

        if (q->next != NULL)
        {
            q->next->prev = p;
        }
        q->next = p;
    }
    else
    {
        cout << "Element not found!" << endl;
    }
}

void deleteAtBeginning()
{
```

```
    if (list == NULL)
    {
        cout << "Linked List is Empty" << endl;
    }
    else
    {
        q = list;
        if (q->next != NULL)
        {
            cout << "Deleted " << q->data << endl;
            list = q->next;
            list->prev = NULL;
        }
        else
        {
            cout << "Deleted " << q->data << endl;
            list = NULL;
        }
        free(q);
    }
}

void deleteAtEnd()
{
    if (list == NULL)
    {
        cout << "Linked List is Empty" << endl;
    }
    else
    {
        q = list;
        if (q->next == NULL)
```

```
        {
            cout << "Deleted " << q->data << endl;
            list = NULL;
        }
        else
        {
            while (q->next != NULL)
            {
                r = q;
                q = q->next;
            }
            cout << "Deleted " << q->data << endl;
            r->next = NULL;
        }
        free(q);
    }
}

void deleteAParticularElement()
{
    if (list == NULL)
    {
        cout << "Linked List is Empty" << endl;
    }
    else
    {
        cout << "Enter the Element which you want to Delete" << endl;
        cin >> target;
        q = list;
        if (q->data == target)
        {
            cout << "Deleted " << q->data << endl;
```

```
            list = q->next;
            if (list != NULL)
            {
                list->prev = NULL;
            }
            free(q);
            return;
        }
        while (q != NULL && q->data != target)
        {
            r = q;
            q = q->next;
        }
        if (q != NULL)
        {
            cout << "Deleted " << q->data << endl;
            r->next = q->next;
            if (q->next != NULL)
            {
                q->next->prev = r;
            }
            free(q);
        }
        else
        {
            cout << "Element not found!" << endl;
        }
    }
}

void sortLinkedList()
{
```

```
    if (list == NULL)
    {
        cout << "Linked List is Empty" << endl;
    }
    q = list;

    while (q != NULL)
    {
        r = q->next;
        while (r != NULL)
        {
            if (q->data > r->data)
            {
                int temp = q->data;
                q->data = r->data;
                r->data = temp;
            }

            r = r->next;
        }
        q = q->next;
    }

    cout << "The Sorted Linked List is :" << endl;
    display();
}

void reverse()
{
    if (list == NULL)
    {
        cout << "Linked List is Empty" << endl;
```

```
    }

    q = list;
    temp = NULL;
    while (q != NULL)
    {
        temp = q->prev;
        q->prev = q->next;
        q->next = temp;
        q = q->prev;
    }
    if (temp != NULL)
    {
        list = temp->prev;
    }
}

void Search()
{
    if (list == NULL)
    {
        cout << "List is Empty" << endl;
    }
    else
    {
        cout << "Enter the Element you want to Search" << endl;
        cin >> target;


        q = list;
        int i = 0;
        while (q != NULL)
```

```
        {
            if (q->data == target)
            {
                cout << "Element Found at " << i << "th Index" << endl;
                return;
            }
            q = q->next;
            i++;
        }

        cout << "Element not found!" << endl;
    }
  }

  void display()
  {
    if (list == NULL)
    {
        cout << "Linked List is Empty" << endl;
    }
    else
    {
        q = list;
        while (q != NULL)
        {
            cout << q->data << "-->";
            q = q->next;
        }
    }
  }
};
```

```
int main()
{
   DoublyLinkedList dll;
   dll.menu();
   return 0;
}
```

Output:

```
Enter you choice:
1) Insert at the Beginning
2) Insert at the End
3) Insert Before a Particular Element
4) Insert After a Particular Element
5) Delete at the Beginning
6) Delete at the End
7) Delete a Particular Element
8) Sort the Linked List
9) Reverse the Linked List
10) Display
11) Search a particular
15) Exit

1
Enter the element you want to Insert:
30

Enter you choice:
1) Insert at the Beginning
2) Insert at the End
3) Insert Before a Particular Element
4) Insert After a Particular Element
5) Delete at the Beginning
6) Delete at the End
7) Delete a Particular Element
8) Sort the Linked List
9) Reverse the Linked List
10) Display
11) Search a particular
15) Exit

1
Enter the element you want to Insert:
50

Enter you choice:
1) Insert at the Beginning
```

```
2) Insert at the End
3) Insert Before a Particular Element
4) Insert After a Particular Element
5) Delete at the Beginning
6) Delete at the End
7) Delete a Particular Element
8) Sort the Linked List
9) Reverse the Linked List
10) Display
11) Search a particular
15) Exit

2
Enter the element you want to Insert:
80

Enter you choice:
1) Insert at the Beginning
2) Insert at the End
3) Insert Before a Particular Element
4) Insert After a Particular Element
5) Delete at the Beginning
6) Delete at the End
7) Delete a Particular Element
8) Sort the Linked List
9) Reverse the Linked List
10) Display
11) Search a particular
15) Exit

10
50-->30-->80-->
Enter you choice:
1) Insert at the Beginning
2) Insert at the End
3) Insert Before a Particular Element
4) Insert After a Particular Element
5) Delete at the Beginning
6) Delete at the End
7) Delete a Particular Element
8) Sort the Linked List
9) Reverse the Linked List
10) Display
11) Search a particular
15) Exit

3
Enter the Element before which you want to Insert the Element
```

```
80
Enter the element you want to insert: 25

Enter you choice:
1) Insert at the Beginning
2) Insert at the End
3) Insert Before a Particular Element
4) Insert After a Particular Element
5) Delete at the Beginning
6) Delete at the End
7) Delete a Particular Element
8) Sort the Linked List
9) Reverse the Linked List
10) Display
11) Search a particular
15) Exit

10
50-->30-->25-->80-->
Enter you choice:
1) Insert at the Beginning
2) Insert at the End
3) Insert Before a Particular Element
4) Insert After a Particular Element
5) Delete at the Beginning
6) Delete at the End
7) Delete a Particular Element
8) Sort the Linked List
9) Reverse the Linked List
10) Display
11) Search a particular
15) Exit

4
Enter the element after which you want to insert the new element: 50
Enter the element you want to insert: 8

Enter you choice:
1) Insert at the Beginning
2) Insert at the End
3) Insert Before a Particular Element
4) Insert After a Particular Element
5) Delete at the Beginning
6) Delete at the End
7) Delete a Particular Element
8) Sort the Linked List
9) Reverse the Linked List
10) Display
```

```
11) Search a particular
15) Exit

10
50-->8-->30-->25-->80-->
Enter you choice:
1) Insert at the Beginning
2) Insert at the End
3) Insert Before a Particular Element
4) Insert After a Particular Element
5) Delete at the Beginning
6) Delete at the End
7) Delete a Particular Element
8) Sort the Linked List
9) Reverse the Linked List
10) Display
11) Search a particular
15) Exit

5
Deleted 50

Enter you choice:
1) Insert at the Beginning
2) Insert at the End
3) Insert Before a Particular Element
4) Insert After a Particular Element
5) Delete at the Beginning
6) Delete at the End
7) Delete a Particular Element
8) Sort the Linked List
9) Reverse the Linked List
10) Display
11) Search a particular
15) Exit

10
8-->30-->25-->80-->
Enter you choice:
1) Insert at the Beginning
2) Insert at the End
3) Insert Before a Particular Element
4) Insert After a Particular Element
5) Delete at the Beginning
6) Delete at the End
7) Delete a Particular Element
8) Sort the Linked List
9) Reverse the Linked List
```

```
10) Display
11) Search a particular
15) Exit

6
Deleted 80

Enter you choice:
1) Insert at the Beginning
2) Insert at the End
3) Insert Before a Particular Element
4) Insert After a Particular Element
5) Delete at the Beginning
6) Delete at the End
7) Delete a Particular Element
8) Sort the Linked List
9) Reverse the Linked List
10) Display
11) Search a particular
15) Exit

10
8-->30-->25-->
Enter you choice:
1) Insert at the Beginning
2) Insert at the End
3) Insert Before a Particular Element
4) Insert After a Particular Element
5) Delete at the Beginning
6) Delete at the End
7) Delete a Particular Element
8) Sort the Linked List
9) Reverse the Linked List
10) Display
11) Search a particular
15) Exit

7
Enter the Element which you want to Delete
30
Deleted 30

Enter you choice:
1) Insert at the Beginning
2) Insert at the End
3) Insert Before a Particular Element
4) Insert After a Particular Element
5) Delete at the Beginning
```

```
6) Delete at the End
7) Delete a Particular Element
8) Sort the Linked List
9) Reverse the Linked List
10) Display
11) Search a particular
15) Exit

10
8-->25-->
Enter you choice:
1) Insert at the Beginning
2) Insert at the End
3) Insert Before a Particular Element
4) Insert After a Particular Element
5) Delete at the Beginning
6) Delete at the End
7) Delete a Particular Element
8) Sort the Linked List
9) Reverse the Linked List
10) Display
11) Search a particular
15) Exit

3
Enter the Element before which you want to Insert the Element
25
Enter the element you want to insert: 80

Enter you choice:
1) Insert at the Beginning
2) Insert at the End
3) Insert Before a Particular Element
4) Insert After a Particular Element
5) Delete at the Beginning
6) Delete at the End
7) Delete a Particular Element
8) Sort the Linked List
9) Reverse the Linked List
10) Display
11) Search a particular
15) Exit

10
8-->80-->25-->
Enter you choice:
1) Insert at the Beginning
2) Insert at the End
```

```
3) Insert Before a Particular Element
4) Insert After a Particular Element
5) Delete at the Beginning
6) Delete at the End
7) Delete a Particular Element
8) Sort the Linked List
9) Reverse the Linked List
10) Display
11) Search a particular
15) Exit

8
The Sorted Linked List is :
8-->25-->80-->
Enter you choice:
1) Insert at the Beginning
2) Insert at the End
3) Insert Before a Particular Element
4) Insert After a Particular Element
5) Delete at the Beginning
6) Delete at the End
7) Delete a Particular Element
8) Sort the Linked List
9) Reverse the Linked List
10) Display
11) Search a particular
15) Exit

9

Enter you choice:
1) Insert at the Beginning
2) Insert at the End
3) Insert Before a Particular Element
4) Insert After a Particular Element
5) Delete at the Beginning
6) Delete at the End
7) Delete a Particular Element
8) Sort the Linked List
9) Reverse the Linked List
10) Display
11) Search a particular
15) Exit

10
80-->25-->8-->
Enter you choice:
1) Insert at the Beginning
```

```
2) Insert at the End
3) Insert Before a Particular Element
4) Insert After a Particular Element
5) Delete at the Beginning
6) Delete at the End
7) Delete a Particular Element
8) Sort the Linked List
9) Reverse the Linked List
10) Display
11) Search a particular
15) Exit

11
Enter the Element you want to Search
8
Element Found at 2th Index

Enter you choice:
1) Insert at the Beginning
2) Insert at the End
3) Insert Before a Particular Element
4) Insert After a Particular Element
5) Delete at the Beginning
6) Delete at the End
7) Delete a Particular Element
8) Sort the Linked List
9) Reverse the Linked List
10) Display
11) Search a particular
15) Exit
```

3) Circular Linked List.

```
#include <iostream>

using namespace std;

int value, ch, key, target;
struct node
{
   int data;
   struct node *next;
}
   *list = NULL,
   *p, *q, *r, *s, *temp;
class CircularLinkedList
{
public:
   void menu()
   {
      do
      {
         cout << endl;
         cout << "Enter you choice:" << endl;
         cout << "1) Insert at the Beginning" << endl;
         cout << "2) Insert at the End" << endl;
         cout << "3) Insert Before a Particular Element" << endl;
         cout << "4) Insert After a Particular Element" << endl;
         cout << "5) Delete at the Beginning" << endl;
         cout << "6) Delete at the End" << endl;
         cout << "7) Delete a Particular Element" << endl;
         cout << "8) Sort the Linked List" << endl;
         cout << "9) Reverse the Linked List" << endl;
         cout << "10) Display" << endl;
```

```
cout << "11) Search a particular " << endl;
cout << "15) Exit" << endl;
cout << endl;

cin >> ch;
switch (ch)
{
case 1:
   insertAtBeginning();
   break;
case 2:
   insertAtEnd();
   break;
case 3:
   insertBeforeParticularElement();
   break;
case 4:
   insertAfterParticularElement();
   break;
case 5:
   deleteAtBeginning();
   break;
case 6:
   deleteAtEnd();
   break;
case 7:
   deleteAParticularElement();
   break;
case 8:
   sortLinkedList();
   break;
```

```
        case 9:
            reverse();
            break;
        case 10:
            display();
            break;
        case 11:
            Search();
            break;

        default:
            cout << "Enter a proper Value";
            break;
        }

    } while (ch != 15);
}

void insertAtBeginning()
{
    p = new node();
    cout << "Enter the element you want to Insert:" << endl;
    cin >> key;
    p->data = key;
    if (list == NULL)
    {
        list = p;
        p->next = list;
    }
    else
    {
```

```
        q = list;
        while (q->next != list)
        {
            q = q->next;
        }
        p->next = list;
        q->next = p;
        list = p;
    }
}

void insertAtEnd()
{
    p = new node();
    cout << "Enter the element you want to Insert:" << endl;
    cin >> key;
    p->data = key;
    if (list == NULL)
    {
        list = p;
        p->next = list;
    }
    else
    {
        q = list;
        while (q->next != list)
        {
            q = q->next;
        }
        q->next = p;
        p->next = list;
```

```
    }
  }

  void insertBeforeParticularElement()
  {
    if (list == NULL)
    {
      cout << "List is Empty" << endl;
    }
    else
    {
      cout << "Enter the Element before which you want to Insert the Element" << endl;
      cin >> target;
      if (list->data == target)
      {
        insertAtBeginning();
        return;
      }

      q = list;
      do
      {
        r = q;
        q = q->next;
      } while (q != list && q->data != target);

      if (q->data == target)
      {
        p = new node();
        cout << "Enter the element you want to insert: ";
        cin >> key;
```

```
            p->data = key;
            p->next = q;
            r->next = p;
        }
        else
        {
            cout << "Element not found!" << endl;
        }
    }
}

void insertAfterParticularElement()
{
    if (list == NULL)
    {
        cout << "List is Empty" << endl;
        return;
    }

    cout << "Enter the element after which you want to insert the new element: ";
    cin >> target;

    r = list;
    do
    {
        if (r->data == target)
        {
            p = new node();
            cout << "Enter the element you want to insert: ";
            cin >> key;
            p->data = key;
```

```
            p->next = r->next;
            r->next = p;
            return;
        }
        r = r->next;
    } while (r != list);

    cout << "Element not found!" << endl;
}

void deleteAtBeginning()
{
    if (list == NULL)
    {
        cout << "Linked List is Empty" << endl;
        return;
    }
    q = list;
    if (q->next == list)
    {
        cout << "Deleted " << q->data << endl;
        delete q;
        list = NULL;
    }
    else
    {
        while (q->next != list)
        {
            q = q->next;
        }
        p = list;
```

```
        cout << "Deleted " << p->data << endl;
        list = list->next;
        q->next = list;
        delete p;
    }
}

void deleteAtEnd()
{
    if (list == NULL)
    {
        cout << "Linked List is Empty" << endl;
        return;
    }
    q = list;
    if (q->next == list)
    {
        cout << "Deleted " << q->data << endl;
        delete q;
        list = NULL;
    }
    else
    {
        while (q->next->next != list)
        {
            q = q->next;
        }
        p = q->next;
        cout << "Deleted " << p->data << endl;
        q->next = list;
        delete p;
```

```
    }
  }

  void deleteAParticularElement()
  {
    if (list == NULL)
    {
      cout << "Linked List is Empty" << endl;
      return;
    }
    cout << "Enter the Element which you want to Delete" << endl;
    cin >> target;

    if (list->data == target)
    {
      deleteAtBeginning();
      return;
    }

    q = list;
    do
    {
      r = q;
      q = q->next;
    } while (q != list && q->data != target);

    if (q->data == target)
    {
      r->next = q->next;
      cout << "Deleted " << q->data << endl;
      delete q;
```

```
    }
    else
    {
        cout << "Element not found!" << endl;
    }
}

void sortLinkedList()
{
    if (list == NULL)
    {
        cout << "Linked List is Empty" << endl;
        return;
    }

    q = list;
    do
    {
        r = q->next;
        while (r != list)
        {
            if (q->data > r->data)
            {
                int temp = q->data;
                q->data = r->data;
                r->data = temp;
            }
            r = r->next;
        }
        q = q->next;
    } while (q != list);
```

```
    cout << "The Sorted Linked List is :" << endl;
    display();
}

void reverse()
{
    if (list == NULL)
    {
        cout << "Linked List is Empty" << endl;
        return;
    }

    q = list;
    p = NULL;
    r = NULL;

    do
    {
        r = q->next;
        q->next = p;
        p = q;
        q = r;
    } while (q != list);

    list->next = p;

    list = p;
    cout << "The Reversed Linked List is :" << endl;
    display();
}
```

```
void Search()
{
    if (list == NULL)
    {
        cout << "List is Empty" << endl;
    }
    else
    {
        cout << "Enter the Element you want to Search: ";
        cin >> target;

        q = list;
        int i = 0;

        do
        {
            if (q->data == target)
            {
                cout << "Element Found at " << i << "th Index" << endl;
                return;
            }
            q = q->next;
            i++;
        } while (q != list);

        cout << "Element not found!" << endl;
    }
}

void display()
```

```
    {
        if (list == NULL)
        {
            cout << "The list is empty." << endl;
            return;
        }

        q = list;
        do
        {
            cout << q->data << "-->";
            q = q->next;
        } while (q != list);
        cout << endl;
    }
};

int main()
{
    CircularLinkedList cll;
    cll.menu();
    return 0;
}
```

Output:

```
Enter you choice:
1) Insert at the Beginning
2) Insert at the End
3) Insert Before a Particular Element
4) Insert After a Particular Element
5) Delete at the Beginning
6) Delete at the End
7) Delete a Particular Element
8) Sort the Linked List
9) Reverse the Linked List
10) Display
```

```
11) Search a particular
15) Exit

1
Enter the element you want to Insert:
20

Enter you choice:
1) Insert at the Beginning
2) Insert at the End
3) Insert Before a Particular Element
4) Insert After a Particular Element
5) Delete at the Beginning
6) Delete at the End
7) Delete a Particular Element
8) Sort the Linked List
9) Reverse the Linked List
10) Display
11) Search a particular
15) Exit

1
Enter the element you want to Insert:
90

Enter you choice:
1) Insert at the Beginning
2) Insert at the End
3) Insert Before a Particular Element
4) Insert After a Particular Element
5) Delete at the Beginning
6) Delete at the End
7) Delete a Particular Element
8) Sort the Linked List
9) Reverse the Linked List
10) Display
11) Search a particular
15) Exit

2
Enter the element you want to Insert:
100

Enter you choice:
1) Insert at the Beginning
2) Insert at the End
3) Insert Before a Particular Element
4) Insert After a Particular Element
```

```
5) Delete at the Beginning
6) Delete at the End
7) Delete a Particular Element
8) Sort the Linked List
9) Reverse the Linked List
10) Display
11) Search a particular
15) Exit

10
90-->20-->100-->

Enter you choice:
1) Insert at the Beginning
2) Insert at the End
3) Insert Before a Particular Element
4) Insert After a Particular Element
5) Delete at the Beginning
6) Delete at the End
7) Delete a Particular Element
8) Sort the Linked List
9) Reverse the Linked List
10) Display
11) Search a particular
15) Exit

3
Enter the Element before which you want to Insert the Element
20
Enter the element you want to insert: 89

Enter you choice:
1) Insert at the Beginning
2) Insert at the End
3) Insert Before a Particular Element
4) Insert After a Particular Element
5) Delete at the Beginning
6) Delete at the End
7) Delete a Particular Element
8) Sort the Linked List
9) Reverse the Linked List
10) Display
11) Search a particular
15) Exit

10
90-->89-->20-->100-->
```

```
Enter you choice:
1) Insert at the Beginning
2) Insert at the End
3) Insert Before a Particular Element
4) Insert After a Particular Element
5) Delete at the Beginning
6) Delete at the End
7) Delete a Particular Element
8) Sort the Linked List
9) Reverse the Linked List
10) Display
11) Search a particular
15) Exit

4
Enter the element after which you want to insert the new element: 20
Enter the element you want to insert: 80

Enter you choice:
1) Insert at the Beginning
2) Insert at the End
3) Insert Before a Particular Element
4) Insert After a Particular Element
5) Delete at the Beginning
6) Delete at the End
7) Delete a Particular Element
8) Sort the Linked List
9) Reverse the Linked List
10) Display
11) Search a particular
15) Exit

10
90-->89-->20-->80-->100-->

Enter you choice:
1) Insert at the Beginning
2) Insert at the End
3) Insert Before a Particular Element
4) Insert After a Particular Element
5) Delete at the Beginning
6) Delete at the End
7) Delete a Particular Element
8) Sort the Linked List
9) Reverse the Linked List
10) Display
11) Search a particular
15) Exit
```

```
5
Deleted 90

Enter you choice:
1) Insert at the Beginning
2) Insert at the End
3) Insert Before a Particular Element
4) Insert After a Particular Element
5) Delete at the Beginning
6) Delete at the End
7) Delete a Particular Element
8) Sort the Linked List
9) Reverse the Linked List
10) Display
11) Search a particular
15) Exit

10
89-->20-->80-->100-->

Enter you choice:
1) Insert at the Beginning
2) Insert at the End
3) Insert Before a Particular Element
4) Insert After a Particular Element
5) Delete at the Beginning
6) Delete at the End
7) Delete a Particular Element
8) Sort the Linked List
9) Reverse the Linked List
10) Display
11) Search a particular
15) Exit

6
Deleted 100

Enter you choice:
1) Insert at the Beginning
2) Insert at the End
3) Insert Before a Particular Element
4) Insert After a Particular Element
5) Delete at the Beginning
6) Delete at the End
7) Delete a Particular Element
8) Sort the Linked List
9) Reverse the Linked List
```

```
10) Display
11) Search a particular
15) Exit

10
89-->20-->80-->

Enter you choice:
1) Insert at the Beginning
2) Insert at the End
3) Insert Before a Particular Element
4) Insert After a Particular Element
5) Delete at the Beginning
6) Delete at the End
7) Delete a Particular Element
8) Sort the Linked List
9) Reverse the Linked List
10) Display
11) Search a particular
15) Exit

7
Enter the Element which you want to Delete
20
Deleted 20

Enter you choice:
1) Insert at the Beginning
2) Insert at the End
3) Insert Before a Particular Element
4) Insert After a Particular Element
5) Delete at the Beginning
6) Delete at the End
7) Delete a Particular Element
8) Sort the Linked List
9) Reverse the Linked List
10) Display
11) Search a particular
15) Exit

10
89-->80-->

Enter you choice:
1) Insert at the Beginning
2) Insert at the End
3) Insert Before a Particular Element
4) Insert After a Particular Element
```

```
5) Delete at the Beginning
6) Delete at the End
7) Delete a Particular Element
8) Sort the Linked List
9) Reverse the Linked List
10) Display
11) Search a particular
15) Exit

8
The Sorted Linked List is :
80-->89-->

Enter you choice:
1) Insert at the Beginning
2) Insert at the End
3) Insert Before a Particular Element
4) Insert After a Particular Element
5) Delete at the Beginning
6) Delete at the End
7) Delete a Particular Element
8) Sort the Linked List
9) Reverse the Linked List
10) Display
11) Search a particular
15) Exit

9
The Reversed Linked List is :
89-->80-->

Enter you choice:
1) Insert at the Beginning
2) Insert at the End
3) Insert Before a Particular Element
4) Insert After a Particular Element
5) Delete at the Beginning
6) Delete at the End
7) Delete a Particular Element
8) Sort the Linked List
9) Reverse the Linked List
10) Display
11) Search a particular
15) Exit

11
Enter the Element you want to Search: 80
Element Found at 1th Index
```

```
Enter you choice:
1) Insert at the Beginning
2) Insert at the End
3) Insert Before a Particular Element
4) Insert After a Particular Element
5) Delete at the Beginning
6) Delete at the End
7) Delete a Particular Element
8) Sort the Linked List
9) Reverse the Linked List
10) Display
11) Search a particular
15) Exit
```

Observation: When performing operations on singly, doubly, and circular linked lists, each type has its own way of handling tasks. Singly linked lists are easy to work with at the beginning, but require going through the list for tasks at the end. Doubly linked lists allow easier manipulation since each node knows both the next and previous node. Circular linked lists make it simple to work at both ends because the last node points back to the first. For all types, sorting, reversing, and searching involve moving through the list and adjusting pointers.