ASSIGNMENT NO. 4 (LINKED LIST)

1) Write a menu driven program to implement singly linear linked list with its different operations.

->

```
#include <iostream>
using namespace std;

class Node {
    int info;
    Node *t3;
    public:
        Node   *create(int);
        void   insert_beg(int);
        void   insert_end(int);
        void   insert_bet(int, int);
        int    remove_beg();
        int    remove_end();
        int    remove_bet(int after);
        void   count();
        void   reverse();
        void   display();
}* list;

Node*  Node::create(int ele) {
    Node *ptr = new Node;
    ptr->info = ele;
```

```
    ptr->t3 = nullptr;
    return ptr;
}

void   Node::insert_beg(int ele) {
    Node *ptr = create(ele);
    ptr->t3 = list;
    list = ptr;
    cout << "\nNode inserted at beginning..." << endl;
}

void   Node::insert_end(int ele) {
    Node *ptr = create(ele);
    if (!list)
      list = ptr;
    else {
      Node *p = list;
      while (p->t3) p = p->t3;
      p->t3 = ptr;
    }
    cout << "\nNode inserted at end..." << endl;
}

void   Node::insert_bet(int after, int ele) {
    Node *p = list;
    while (p && p->info != after) p = p->t3;
    if (p) {
```

```
        Node *ptr = create(ele);
        ptr->t3 = p->t3;
        p->t3 = ptr;
        cout << "\nNode inserted..." << endl;
     } else
        cout << "\nNode not found..." << endl;
}

int   Node::remove_beg() {
    if (!list) return 0;
    Node *p = list;
    int ele = p->info;
    list = list->t3;
    delete p;
    return ele;
}

int   Node::remove_end() {
     if (!list) return 0;
     if (!list->t3) {
        int ele = list->info;
        delete list;
        list = nullptr;
        return ele;
     }
     Node *p = list;
     while (p->t3->t3) p = p->t3;
```

```
    int ele = p->t3->info;
    delete p->t3;
    p->t3 = nullptr;
    return ele;
}

int    Node::remove_bet(int after) {
    Node *p = list;
    while (p && p->t3 && p->info != after) p = p->t3;

    if (p && p->t3) {
      Node *tmp = p->t3;
      int ele = tmp->info;
      p->t3 = tmp->t3;
      delete tmp;
      return ele;
    }
    return 0;
}

void    Node::count() {
    int cnt = 0;
    for (Node *p = list; p; p = p->t3) cnt++;
    cout << "\nNodes count: " << cnt << endl;
}

void    Node::reverse() {
```

```
    Node *t1 = nullptr, *t2 = list, *t3;
    while (t2) {
      t3 = t2->t3;
      t2->t3 = t1;
      t1 = t2;
      t2 = t3;
    }
    list = t1;
    cout << "\nList reversed.\n";
}

void    Node::display() {
    cout << "\nElements: ";
    for (Node *p = list; p; p = p->t3) cout << p->info << " ";
    cout << endl;
}

int main() {
   Node obj;
   int ele, after, ch;

   do {
      cout << "\n1: Insert begin\n2: Insert end\n3: Insert after\n4: Remove begin\n5:
Remove end\n6: Remove after\n7: Count\n8: Reverse\n9: Display\n10: Exit\nEnter
choice: ";
      cin >> ch;

      switch (ch) {
```

```
case 1:
    cout << "\nEnter number: ";
    cin >> ele;
    obj.insert_beg(ele);
    break;

case 2:
    cout << "\nEnter number: ";
    cin >> ele;
    obj.insert_end(ele);
    break;

case 3:
    cout << "\nEnter after which number: ";
    cin >> after;
    cout << "\nEnter number: ";
    cin >> ele;
    obj.insert_bet(after, ele);
    break;

case 4:
    cout << "\nRemoved: " << obj.remove_beg() << endl;
    break;

case 5:
    cout << "\nRemoved: " << obj.remove_end() << endl;
    break;
```

```
    case 6:
        cout << "\nEnter after which number: ";
        cin >> after;
        cout << "\nRemoved: " << obj.remove_bet(after) << endl;
        break;

    case 7:
        obj.count();
        break;

    case 8:
        obj.reverse();
        break;

    case 9:
        obj.display();
        break;

    case 10:
        cout << "Exited.\n";
        break;

    default:
        cout << "\nInvalid choice..." << endl;
    }
} while (ch != 10);
```

```
    return 0;
}
```

Output =>

```
1: Insert begin
2: Insert end
3: Insert after
4: Remove begin
5: Remove end
6: Remove after
7: Count
8: Reverse
9: Display
10: Exit
Enter choice: 1

Enter number: 10

Node inserted at beginning...

1: Insert begin
2: Insert end
3: Insert after
4: Remove begin
5: Remove end
6: Remove after
7: Count
```

8: Reverse

9: Display

10: Exit

Enter choice: 2

Enter number: 20

Node inserted at end...

1: Insert begin

2: Insert end

3: Insert after

4: Remove begin

5: Remove end

6: Remove after

7: Count

8: Reverse

9: Display

10: Exit

Enter choice: 3

Enter after which number: 100

Enter number: 400

Node not found...

1: Insert begin

2: Insert end

3: Insert after

4: Remove begin

5: Remove end

6: Remove after

7: Count

8: Reverse

9: Display

10: Exit

Enter choice: 7

Nodes count: 2

1: Insert begin

2: Insert end

3: Insert after

4: Remove begin

5: Remove end

6: Remove after

7: Count

8: Reverse

9: Display

10: Exit

Enter choice: 9

Elements: 10 20

1: Insert begin

2: Insert end

3: Insert after

4: Remove begin

5: Remove end

6: Remove after

7: Count

8: Reverse

9: Display

10: Exit

Enter choice: 10

Exited.

2) Write a menu program to implement stack by using linked list. (Dynamic implementation of stack)

->

```
#include<iostream>
using namespace std;

class stack {
    int info;
    stack *next;
    public:
        stack* create();
        void   isempty();
       void   push(int);
        int    pop();
         void   display();
}*list;
```

```
stack*  stack::create() {
    return new stack;
}

void  stack::isempty() {
    cout << (list ? "\nstack is Not Empty..\n" : "\nstack is Empty..\n");
}

void  stack::push(int ele) {
     stack *p = create();
     p->info = ele;
     p->next = list;
     list = p;
     cout << "\nElement is Inserted..." << endl;
}

int  stack::pop() {
     if (!list) {
       cout << "\nstack Underflows.." << endl;
       return 0;
     }
     int   ele = list -> info;
     stack *temp = list;
     list = list -> next;
     delete temp;
     return ele;
```

```
}

void  stack::display() {
    cout << "\nElements:\n\t";
    for (stack* p = list; p; p = p->next)
        cout << p->info << "  ";
    cout << endl;
}

int main() {

    int ch, ele;
    stack obj;

    while (true) {
        cout << "\n1: Push \n2: Pop \n3: Display \n4: isEmpty \n5: Exit \nEnter your ch: ";
        cin >> ch;

        switch (ch) {
            case 1: cout << "\nEnter the element to push: "; cin >> ele; obj.push(ele); break;
            case 2: cout << "\nPopped element: " << obj.pop() << endl; break;
            case 3: obj.display(); break;
            case 4: obj.isempty(); break;
            case 5: cout << "\nExited.." << endl; return 0;
            default: cout << "Invalid input\n";
        }
    }
```

```
    return 0;
}
```

Output =>

```
1: Push
2: Pop
3: Display
4: isEmpty
5: Exit
Enter your ch: 1

Enter the element to push: 100

Element is Inserted...

1: Push
2: Pop
3: Display
4: isEmpty
5: Exit
Enter your ch: 1

Enter the element to push: 200

Element is Inserted...

1: Push
2: Pop
```

3: Display

4: isEmpty

5: Exit

Enter your ch: 1

Enter the element to push: 300

Element is Inserted...

1: Push

2: Pop

3: Display

4: isEmpty

5: Exit

Enter your ch: 3

Elements:

300 200 100

1: Push

2: Pop

3: Display

4: isEmpty

5: Exit

Enter your ch: 2

Popped element: 300

1: Push

2: Pop

3: Display

4: isEmpty

5: Exit

Enter your ch: 2

Popped element: 200

1: Push

2: Pop

3: Display

4: isEmpty

5: Exit

Enter your ch: 3

Elements:

100

1: Push

2: Pop

3: Display

4: isEmpty

5: Exit

Enter your ch: 4

stack is Not Empty..

1: Push

2: Pop

3: Display

4: isEmpty

5: Exit

Enter your ch: 5

Exited..

3) Write a menu program to implement queue by using linked list. (Dynamic implementation of queue)

->

```
#include<iostream>
using namespace std;

class  queue {
       int    info;
      queue  *next;
      public:
           queue*  create();
           void    isempty();
           void    insert(int);
          int     remove();
          void    display();
}*list;

queue*  queue::create() {
     return new queue;
```

```
}

void    queue::isempty() {
    cout << (list ? "\nQueue is Not Empty..\n" : "\nQueue is Empty..\n");
}

void    queue::insert(int ele) {
      queue* q = create();
      q->info = ele;
      q->next = NULL;

      if (!list) {
        list = q;
      } else {
        queue* p = list;
        while (p->next) p = p->next;
        p->next = q;
      }
      cout << "\nElement is inserted in queue.." << endl;
}

int  queue::remove() {
      if (!list) {
        cout << "\nQueue Underflows.." << endl;
        return 0;
      }
      int   ele = list -> info;
```

```
    queue *temp = list;
    list = list -> next;
    delete temp;
    return ele;
}

void  queue::display() {
   cout << "\nElements:\n\t";
   for (queue* p = list; p; p = p->next)
      cout << p->info << "  ";
   cout << endl;
}

int main() {

   int ch, ele;
   queue obj;

   while (true) {
     cout << "\n1: Insert\n2: Remove\n3: Display\n4: isEmpty\n5: Exit\nEnter your choice:
";
      cin >> ch;

      switch (ch) {
         case 1: cout << "\nEnter the element to insert: "; cin >> ele; obj.insert(ele); break;
         case 2: cout << "\nRemoved element: " << obj.remove() << endl; break;
         case 3: obj.display(); break;
         case 4: obj.isempty(); break;
```

```
            case 5: cout << "\nExited.." << endl; return 0;
            default: cout << "Invalid input\n";
        }
    }
    return 0;
}
```

Output ->

```
1: Insert
2: Remove
3: Display
4: isEmpty
5: Exit
Enter your choice: 1

Enter the element to insert: 100

Element is inserted in queue..

1: Insert
2: Remove
3: Display
4: isEmpty
5: Exit
Enter your choice: 1

Enter the element to insert: 200
```

Element is inserted in queue..

1: Insert

2: Remove

3: Display

4: isEmpty

5: Exit

Enter your choice: 1

Enter the element to insert: 300

Element is inserted in queue..

1: Insert

2: Remove

3: Display

4: isEmpty

5: Exit

Enter your choice: 3

Elements:

100 200 300

1: Insert

2: Remove

3: Display

4: isEmpty

5: Exit

Enter your choice: 4

Queue is Not Empty..

1: Insert

2: Remove

3: Display

4: isEmpty

5: Exit

Enter your choice: 2

Removed element: 100

1: Insert

2: Remove

3: Display

4: isEmpty

5: Exit

Enter your choice: 2

Removed element: 200

1: Insert

2: Remove

3: Display

4: isEmpty

5: Exit

Enter your choice: 3

Elements:

300

1: Insert

2: Remove

3: Display

4: isEmpty

5: Exit

Enter your choice: 5

Exited..

4) Write a menu program to implement doubly linear linked list with its different operations.

->


```
#include <iostream>
using namespace std;

class node {
      int info;
      node *prev, *next;
      public:
         node   *create();
          void   insert_beg(int ele);
         void   insert_end(int ele);
         void   insert_bet(int after, int ele);
         int    remove_beg();
         int    remove_end();
         int    remove_bet(int after);
```

```
        void   display();
        void   search(int num);
         void   count();
         void   reverse();
} *list;

node*  create() {
    return new node;
}
void   node::insert_beg(int ele) {
   node *p = create();
   p->info = ele;
   p->prev = NULL;
   p->next = list;
   if (list) list->prev = p;
   list = p;
   cout << "\nNode is inserted.." << endl;
}

void   node::insert_end(int ele) {
   node *p = list, *q = create();
   q->info = ele;
   q->next = NULL;
   if (!p) {
      q->prev = NULL;
      list = q;
   } else {
```

```
        while (p->next) p = p->next;
        p->next = q;
        q->prev = p;
    }
    cout << "\nNode is inserted.." << endl;
}

void   node::insert_bet(int after, int ele) {
    node *p = list;
    while (p && p->info != after) p = p->next;
    if (!p || !p->next) {
        cout << "\nInsert between is not possible.." << endl;
    } else {
        node *q = create();
        q->info = ele;
        q->prev = p;
        q->next = p->next;
        p->next->prev = q;
        p->next = q;
        cout << "\nNode is inserted.." << endl;
    }
}

int   node::remove_beg() {
    if (!list) return cout << "\nRemove operation failed.." << endl, 0;
    node *p = list;
    int ele = p->info;
```

```
    list = list->next;
    if (list) list->prev = NULL;
    delete p;
    return ele;
}

int   node::remove_end() {
    if (!list) return cout << "\nRemove end failed.." << endl, 0;
    node *p = list;
    while (p->next) p = p->next;
    int ele = p->info;
    if (p->prev) p->prev->next = NULL;
    else list = NULL;
    delete p;
    return ele;
}

int   node::remove_bet(int after) {
    node *p = list;
    while (p && p->info != after) p = p->next;
    if (!p || !p->next) return cout << "\nRemove between failed.." << endl, 0;
    node *tmp = p->next;
    int ele = tmp->info;
    p->next = tmp->next;
    if (tmp->next) tmp->next->prev = p;
    delete tmp;
    return ele;
```

```
}

void   node::display() {
   cout << "\nElements:\n\t";
   for (node *p = list; p; p = p->next) cout << p->info << "   ";
   cout << endl;
}

void   node::search(int num) {
   for (node *p = list; p; p = p->next) {
      if (p->info == num) cout << "\nNode is Found" << endl; break;
   }
   cout << "\nNode is not Found" << endl;
}

void   node::count() {
   int counter = 0;
   for (node *p = list; p; p = p->next) counter++;
   cout << "\nTotal Nodes: " << counter << endl;
}

void   node::reverse() {
   node *current = list, *prev = NULL;
   while (current) {
      node *next = current->next;
      current->next = prev;
      current->prev = next;
```

```
        prev = current;
        current = next;
    }
    list = prev;
    cout << "\nLinked list is Reversed.." << endl;
}

int   main() {
    node obj;
    int num, after, ch;
    do {
        cout << "\n1: Insert begin\n2: Insert end\n3: Insert after\n4: Remove begin\n5:
Remove end\n6: Remove after\n7: Display\n8: Search\n9: Count\n10: Reverse\n11:
Exit\nEnter your choice: ";
        cin >> ch;
        switch (ch) {
        case 1:
          cout << "\nEnter number to insert at beginning: "; cin >> num;
          obj.insert_beg(num);
          break;
        case 2:
           cout << "\nEnter number to insert at end: "; cin >> num;
           obj.insert_end(num);
           break;
        case 3:
           cout << "\nAfter which node do you want to insert: "; cin >> after;
           cout << "Enter number to insert: "; cin >> num; obj.insert_bet(after, num);
           break;
```

```
        case 4:
            cout << "\nRemoved element at first: " << obj.remove_beg() << endl;
            break;
        case 5:
            cout << "\nRemoved element at last: " << obj.remove_end() << endl;
            break;
        case 6:
            cout << "\nAfter which node do you want to remove: "; cin >> after;
            cout << "\nRemoved element after " << after << ": " << obj.remove_bet(after) <<
endl; break;
        case 7: obj.display(); break;
        case 8:
             cout << "\nEnter element to search: "; cin >> num;
             obj.search(num); break;
        case 9:
             obj.count(); break;
        case 10:
             obj.reverse(); break;
        case 11: return 0;
        default: cout << "\nInvalid choice. Try again.\n";
        }
    } while (ch != 11);
    return 0;
}
```


Output =>

1: Insert begin

2: Insert end

3: Insert after

4: Remove begin

5: Remove end

6: Remove after

7: Display

8: Search

9: Count

10: Reverse

11: Exit

Enter your choice: 1

Enter number to insert at beginning: 100

Node is inserted..

1: Insert begin

2: Insert end

3: Insert after

4: Remove begin

5: Remove end

6: Remove after

7: Display

8: Search

9: Count

10: Reverse

11: Exit

Enter your choice: 1

Enter number to insert at beginning: 200

Node is inserted..

1: Insert begin

2: Insert end

3: Insert after

4: Remove begin

5: Remove end

6: Remove after

7: Display

8: Search

9: Count

10: Reverse

11: Exit

Enter your choice: 2

Enter number to insert at end: 300

Node is inserted..

1: Insert begin

2: Insert end

3: Insert after

4: Remove begin

5: Remove end

6: Remove after

7: Display

8: Search

9: Count

10: Reverse

11: Exit

Enter your choice: 7

Elements:

200 100 300

1: Insert begin

2: Insert end

3: Insert after

4: Remove begin

5: Remove end

6: Remove after

7: Display

8: Search

9: Count

10: Reverse

11: Exit

Enter your choice: 3

After which node do you want to insert: 200

Enter number to insert: 400

Node is inserted..

1: Insert begin

2: Insert end

3: Insert after

4: Remove begin

5: Remove end

6: Remove after

7: Display

8: Search

9: Count

10: Reverse

11: Exit

Enter your choice: 7

Elements:

200 400 100 300

1: Insert begin

2: Insert end

3: Insert after

4: Remove begin

5: Remove end

6: Remove after

7: Display

8: Search

9: Count

10: Reverse

11: Exit

Enter your choice: 9

Total Nodes: 4

1: Insert begin

2: Insert end

3: Insert after

4: Remove begin

5: Remove end

6: Remove after

7: Display

8: Search

9: Count

10: Reverse

11: Exit

Enter your choice: 4

Removed element at first: 200

1: Insert begin

2: Insert end

3: Insert after

4: Remove begin

5: Remove end

6: Remove after

7: Display

8: Search

9: Count

10: Reverse

11: Exit

Enter your choice: 4

Removed element at first: 400

1: Insert begin

2: Insert end

3: Insert after

4: Remove begin

5: Remove end

6: Remove after

7: Display

8: Search

9: Count

10: Reverse

11: Exit

Enter your choice: 7

Elements:

100 300

1: Insert begin

2: Insert end

3: Insert after

4: Remove begin

5: Remove end

6: Remove after

7: Display

8: Search

9: Count

10: Reverse

11: Exit

Enter your choice: 5

Removed element at last: 300

1: Insert begin

2: Insert end

3: Insert after

4: Remove begin

5: Remove end

6: Remove after

7: Display

8: Search

9: Count

10: Reverse

11: Exit

Enter your choice: 7

Elements:

100

1: Insert begin

2: Insert end

3: Insert after

4: Remove begin

5: Remove end

6: Remove after

7: Display

8: Search

9: Count

10: Reverse

11: Exit

Enter your choice: 11

5) Write a menu program to implement singly circular linked list with its different operations

->

```
#include<iostream>
using namespace std;

class node {
      int info;
       node *next;
       public:
          node* create() { return new node; }
          void insert_beg(int);
          void insert_end(int);
          void insert_bet(int, int);
          int remove_beg();
          int remove_end();
          int remove_bet(int);
          void count();
          void reverse();
          void search(int);
          void display();
} *list = NULL;

void node::insert_beg(int ele) {
   node *p = list, *q;
```

```
    if (!p) {
        p = create();
        p->info = ele;
        p->next = p;
        list = p;
    }
    else {
        while (p->next != list) p = p->next;
        q = create();
        q->info = ele;
        q->next = p->next;
        p->next = q;
        list = q;
    }
    cout << "\nNode inserted.." << endl;
}

void node::insert_end(int ele) {
    node *p = list, *q;
    if (!p) {
        p = create();
        p->info = ele;
        p->next = p;
        list = p;
    }
    else {
        while (p->next != list) p = p->next;
```

```
        q = create();
        q->info = ele;
        q->next = p->next;
        p->next = q;
    }
    cout << "\nNode inserted.." << endl;
}

void node::insert_bet(int after, int ele) {
    node *p = list, *q;
    if (!p || p->next == p) { cout << "\nInsert between failed.." << endl; return; }
    while (p->next != list) {
        if (p->info == after) {
            q = create();
            q->info = ele;
            q->next = p->next;
            p->next = q;
            cout << "\nNode inserted.." << endl; return;
        }
        p = p->next;
    }
}

int node::remove_beg() {
    node *p = list, *tmp;
    if (!p) { cout << "\nEmpty list.." << endl; return 0; }
    if (p->next == p) {
```

```
        int ele = p->info;
        list = NULL;
        delete p;
        return ele;
    }
    while (p->next != list) p = p->next;
    tmp = p->next;
    int ele = tmp->info;
    p->next = tmp->next;
    list = tmp->next;
    delete tmp;
    return ele;
}

int node::remove_end() {
    node *p = list, *tmp;
    if (!p) { cout << "\nEmpty list.." << endl; return 0; }
    if (p->next == p) {
        int ele = p->info;
        list = NULL;
        delete p;
        return ele;
    }
    while (p->next->next != list) p = p->next;
    tmp = p->next;
    int ele = tmp->info;
    p->next = tmp->next;
```

```
    delete tmp;
    return ele;
}

int node::remove_bet(int after) {
    node *p = list, *tmp;
    if (!p || p->next == p) { cout << "\nEmpty list.." << endl; return 0; }
    while (p->next != list) {
        if (p->info == after) {
            tmp = p->next;
            int ele = tmp->info;
            p->next = tmp->next;
            delete tmp;
            return ele;
        }
        p = p->next;
    }
    return 0;
}

void node::count() {
    node *p = list; int counter = 0;
    if (!p) { cout << "\nTotal Nodes: 0" << endl; return; }
    do { counter++; p = p->next; } while (p != list);
    cout << "\nTotal Nodes: " << counter << endl;
}
```

```
void node::reverse() {
    node *t1 = list, *t2, *t3 = list;
    do {
        t2 = t1->next; t1->next = t3; t3 = t1; t1 = t2;
    } while (t1 != list);
    list = t3; t1->next = t3;
    cout << "\nList reversed.." << endl;
}

void node::search(int ele) {
    node *p = list; do {
        if (p->info == ele) { cout << "\nNode found.." << endl; return; }
        p = p->next;
    } while (p != list);
    cout << "\nNode not found.." << endl;
}

void node::display() {
    node *p = list;
    if (!p) { cout << "Empty list.." << endl; return; }
    cout << "\nElements: ";
    do {
         cout << p->info << " "; p = p->next;
    } while (p != list);
    cout << endl;
}
```

```
int main() {
    node obj; int num, after, ch;
    do {
        cout << "\n1 : Insert begin\n2 : Insert end\n3 : Insert after\n4 : Remove begin\n5 : Remove end\n6 : Remove after\n7 : Display\n8 : Search\n9 : Count\n10 : Reverse\n11 : Exit\nEnter choice: ";
        cin >> ch;
        switch (ch) {
            case 1:
                cout << "\nEnter number to insert at beginning: "; cin >> num;
                obj.insert_beg(num);
                break;
            case 2:
                cout << "\nEnter number to insert at end: "; cin >> num;
                obj.insert_end(num);
                break;
            case 3:
                cout << "\nAfter which node do you want to insert: "; cin >> after;
                cout << "Enter number to insert: "; cin >> num; obj.insert_bet(after, num);
                break;
            case 4:
                cout << "\nRemoved element at beginning: " << obj.remove_beg() << endl;
                break;
            case 5:
                cout << "\nRemoved element at end: " << obj.remove_end() << endl;
                break;
            case 6:
                cout << "\nAfter which node do you want to remove: "; cin >> after;
```

```
            cout << "\nRemoved element: " << obj.remove_bet(after) << endl;
            break;
        case 7: obj.display(); break;
        case 8: cout << "\nEnter element to search: "; cin >> num; obj.search(num);
break;
        case 9: obj.count(); break;
        case 10: obj.reverse(); break;
        case 11: cout << "\nExited.." << endl; return 0;
      }
   } while (ch != 11);
   return 0;
}
```

Output =>

1 : Insert begin

2 : Insert end

3 : Insert after

4 : Remove begin

5 : Remove end

6 : Remove after

7 : Display

8 : Search

9 : Count

10 : Reverse

11 : Exit

Enter choice: 1

Enter number to insert at beginning: 100

Node inserted..

1 : Insert begin

2 : Insert end

3 : Insert after

4 : Remove begin

5 : Remove end

6 : Remove after

7 : Display

8 : Search

9 : Count

10 : Reverse

11 : Exit

Enter choice: 2

Enter number to insert at end: 200

Node inserted..

1 : Insert begin

2 : Insert end

3 : Insert after

4 : Remove begin

5 : Remove end

6 : Remove after

7 : Display

8 : Search

9 : Count

10 : Reverse

11 : Exit

Enter choice: 7

Elements: 100 200

1 : Insert begin

2 : Insert end

3 : Insert after

4 : Remove begin

5 : Remove end

6 : Remove after

7 : Display

8 : Search

9 : Count

10 : Reverse

11 : Exit

Enter choice: 3

After which node do you want to insert: 100

Enter number to insert: 300

Node inserted..

1 : Insert begin

2 : Insert end

3 : Insert after

4 : Remove begin

5 : Remove end

6 : Remove after

7 : Display

8 : Search

9 : Count

10 : Reverse

11 : Exit

Enter choice: 7

Elements: 100 300 200

1 : Insert begin

2 : Insert end

3 : Insert after

4 : Remove begin

5 : Remove end

6 : Remove after

7 : Display

8 : Search

9 : Count

10 : Reverse

11 : Exit

Enter choice: 9

Total Nodes: 3

1 : Insert begin

2 : Insert end

3 : Insert after

4 : Remove begin

5 : Remove end

6 : Remove after

7 : Display

8 : Search

9 : Count

10 : Reverse

11 : Exit

Enter choice: 8

Enter element to search: 100

Node found..

1 : Insert begin

2 : Insert end

3 : Insert after

4 : Remove begin

5 : Remove end

6 : Remove after

7 : Display

8 : Search

9 : Count

10 : Reverse

11 : Exit

Enter choice: 10

List reversed..

1 : Insert begin

2 : Insert end

3 : Insert after

4 : Remove begin

5 : Remove end

6 : Remove after

7 : Display

8 : Search

9 : Count

10 : Reverse

11 : Exit

Enter choice: 7

Elements: 200 300 100

1 : Insert begin

2 : Insert end

3 : Insert after

4 : Remove begin

5 : Remove end

6 : Remove after

7 : Display

8 : Search

9 : Count

10 : Reverse

11 : Exit

Enter choice: 4

Removed element at beginning: 200

1 : Insert begin

2 : Insert end

3 : Insert after

4 : Remove begin

5 : Remove end

6 : Remove after

7 : Display

8 : Search

9 : Count

10 : Reverse

11 : Exit

Enter choice: 5

Removed element at end: 100

1 : Insert begin

2 : Insert end

3 : Insert after

4 : Remove begin

5 : Remove end

6 : Remove after

7 : Display

8 : Search

9 : Count

10 : Reverse

11 : Exit

Enter choice: 7

Elements: 300

1 : Insert begin

2 : Insert end

3 : Insert after

4 : Remove begin

5 : Remove end

6 : Remove after

7 : Display

8 : Search

9 : Count

10 : Reverse

11 : Exit

Enter choice: 11

Exited..

6) Write a menu driven program to implement doubly circular linked list with its different operations.

->

```
#include<iostream>
using namespace std;

class  node {
    int   info;
    node  *prev, *next;
    public:
      node* create();
```

```
        void  insert_beg(int);
        void  insert_end(int);
        void  insert_bet(int, int);
        int   remove_beg();
        int   remove_end();
        int   remove_bet(int);
        void  count();
        void  reverse();
        void  search(int);
        void  display();
}*list;

node*  node::create() {
    return new  node;
}

void  node::insert_beg(int ele) {
     node *p = list, *q;

     if (p == NULL) {
        p = create();
        p -> prev = p;
        p -> info = ele;
        p -> next = p;
        list = p;
     }
     else {
```

```
        while (p -> next != list) {
            p = p -> next;
        }
        q = create();
        q -> prev = p;
        q -> info = ele;
        q -> next = p -> next;
        p -> next = q;
        list = q;
    }
    cout << "\nNode is Created.." << endl;
}

void  node::insert_end(int ele) {
    node *p = list, *q;

    if (p == NULL) {
        p = create();
        p -> prev = p;
        p -> info = ele;
        p -> next = p;
        list = p;
    }
    else {
        while (p -> next != list) {
            p = p -> next;
        }
```

```
        q = create();
        q -> prev = p;
        q -> info = ele;
        q -> next = p -> next;
        p -> next -> prev = q;
        p -> next = q;
    }
    cout << "\nNode is Created.." << endl;
}

void  node::insert_bet(int after, int ele) {
    node *p = list, *q;

    if (p == NULL) {
        cout << "\nLinked list Empty.." << endl;
    }
    else if(p -> next == p) {
        cout << "\nInsert between Failed.." << endl;
    }
    else {
        while (p -> next != list) {
            if (p -> info == after) {
                q = create();
                q -> prev = p;
                q -> info = ele;
                q -> next = p -> next;
                p -> next -> prev = q;
```

```
                p -> next = q;
                cout << "\nNode is Created.." << endl;
            }

            p = p -> next;
        }

    }

}

int  node::remove_beg() {
    node  *p = list, *tmp;
    int ele;

    if (p == NULL) {
        cout << "\nLinked list is Empty.." << endl;
        return 0;
    }
    else if(p -> prev == p && p -> next == p) {
        ele = p -> info;
        list = NULL;
        delete p;
        return ele;
    }
    else {
        while (p -> next != list) {
```

```
            p = p -> next;
        }
        tmp = p -> next;
        ele = tmp -> info;
        p -> next = tmp -> next;
        list = tmp -> next;
        tmp -> next -> prev = p;
        delete tmp;
        return ele;
    }
}

int  node::remove_end() {
    node  *p = list, *tmp;
    int ele;

    if (p == NULL) {
       cout << "\nLinked list is Empty.." << endl;
       return 0;
    }
    else if(p -> prev == p && p -> next == p) {
        ele = p -> info;
        list = NULL;
        delete p;
        return ele;
    }
    else {
```

```
        while (p -> next -> next != list) {
            p = p -> next;
        }
        tmp = p -> next;
        ele = tmp -> info;
        p -> next = tmp -> next;
        tmp -> next -> prev = p;
        delete tmp;
        return ele;
    }
}

int  node::remove_bet(int after) {
    node  *p = list, *tmp;
    int ele;

    if (p == NULL) {
       cout << "\nLinked list is Empty.." << endl;
       return 0;
    }
    else if(p -> prev == p && p -> next == p) {
        cout << "\nRemove after Failed.." << endl;
    }
    else {
        while (p -> next != list) {
            if (p -> info == after) {
                tmp = p -> next;
```

```
                ele = tmp -> info;
                p -> next = tmp -> next;
                tmp -> next -> prev = p;
                delete tmp;
                return ele;
            }

            p = p -> next;
        }
    }
    return 0;
}

void  node::count() {
     node *p = list;
     int counter = 0;

     if(p == NULL) {
        counter = 0;
     }
     else {
        do
        {
          counter ++;
          p = p -> next;
        } while (p != list);
     }
```

```
    cout << "\nTotal Nodes: " << counter << endl;
}

void  node::reverse() {
    node *t1 = list,*t2, *t3 = list;
    do
    {
       t2 = t1 -> next;
       t1 -> next = t3;
       t1 -> prev = t2;
       t3 = t1;
       t1 = t2;
    } while (t1 != list);
    list = t3;
    t1 -> next = t3;
    cout << "\nLinked list is reversed.." << endl;
}

void  node::search(int ele) {
    node *p = list;
    int check = 0;

    do {
      if (p -> info == ele) {
          check = 1;
          break;
      }
```

```
        p = p -> next;
      } while (p != list);
      if (check) {
         cout << "\nNode is Found.." << endl;
      }
      else {
         cout << "\nNode is not Found.." << endl;
      }
}

void  node::display() {
       node *p = list;

       cout << "\nElements: \n\t";
       if (p != NULL) {
           do {
              cout << p -> info << "   ";
              p = p -> next;
           } while (p != list);
           cout << endl;
       }
       else {
           cout << "Empty.." << endl;
       }
}

int  main()
```

```
{

    node obj;
    int num, after, ch;
    do
    {
       cout << "\n1 : Insert begin\n2 : Insert end\n3 : Insert after\n4 : Remove begin\n5 :
Remove end\n6 : Remove after\n7 : Display\n8 : Search\n9 : Count \n10 : Reverse
L.L.\n11 : Exit\n\nEnter your choice : ";
       cin >> ch;
       switch(ch)
       {
        case 1:
          cout << "\nEnter number to insert at begging : ";
          cin >> num;
          obj.insert_beg(num);
          break;

        case 2:
          cout << "\nEnter number to insert at end : ";
          cin >> num;
          obj.insert_end(num);
          break;

        case 3:
          cout << "\nAfter which node do you want to insert : ";
          cin >> after;
          cout << "Enter number to insert : ";
```

```
  cin >> num;
  obj.insert_bet(after, num);
  break;

case 4:
  cout << "\nRemoved element at first: "<< obj.remove_beg() << endl;
  break;

case 5:
  cout << "\nRemoved element at last : "<< obj.remove_end() << endl;
  break;

case 6:
  cout << "\nAfter which node do you want to remove : ";
  cin >> after;
  cout << "\nRemoved element after "<<after <<" : "<< obj.remove_bet(after)<< endl;
  break;

case 7:
  obj.display();
  break;

case 8:
  cout << "\nEnter element do you want to search : ";
  cin >> num;
  obj.search(num);
  break;
```

```
        case 9:
          obj.count();
          break;

        case 10:
          obj.reverse();
          break;

        case 11:
          cout << "\nExited.." << endl;
          exit(1);

      }
    } while (ch != 11);
    return 0;
}
```


Output ->

1 : Insert begin

2 : Insert end

3 : Insert after

4 : Remove begin

5 : Remove end

6 : Remove after

7 : Display

8 : Search

9 : Count

10 : Reverse L.L.

11 : Exit

Enter your choice : 1

Enter number to insert at begging : 100

Node is Created..

1 : Insert begin

2 : Insert end

3 : Insert after

4 : Remove begin

5 : Remove end

6 : Remove after

7 : Display

8 : Search

9 : Count

10 : Reverse L.L.

11 : Exit

Enter your choice : 2

Enter number to insert at end : 300

Node is Created..

1 : Insert begin

2 : Insert end

3 : Insert after

4 : Remove begin

5 : Remove end

6 : Remove after

7 : Display

8 : Search

9 : Count

10 : Reverse L.L.

11 : Exit

Enter your choice : 2

Enter number to insert at end : 100

Node is Created..

1 : Insert begin

2 : Insert end

3 : Insert after

4 : Remove begin

5 : Remove end

6 : Remove after

7 : Display

8 : Search

9 : Count

10 : Reverse L.L.

11 : Exit

Enter your choice : 200

1 : Insert begin

2 : Insert end

3 : Insert after

4 : Remove begin

5 : Remove end

6 : Remove after

7 : Display

8 : Search

9 : Count

10 : Reverse L.L.

11 : Exit

Enter your choice : 7

Elements:

100 300 100

1 : Insert begin

2 : Insert end

3 : Insert after

4 : Remove begin

5 : Remove end

6 : Remove after

7 : Display

8 : Search

9 : Count

10 : Reverse L.L.

11 : Exit

Enter your choice : 9

Total Nodes: 3

1 : Insert begin

2 : Insert end

3 : Insert after

4 : Remove begin

5 : Remove end

6 : Remove after

7 : Display

8 : Search

9 : Count

10 : Reverse L.L.

11 : Exit

Enter your choice : 10

Linked list is reversed..

1 : Insert begin

2 : Insert end

3 : Insert after

4 : Remove begin

5 : Remove end

6 : Remove after

7 : Display

8 : Search

9 : Count

10 : Reverse L.L.

11 : Exit

Enter your choice : 7

Elements:

100 300 100

1 : Insert begin

2 : Insert end

3 : Insert after

4 : Remove begin

5 : Remove end

6 : Remove after

7 : Display

8 : Search

9 : Count

10 : Reverse L.L.

11 : Exit

Enter your choice : 11

Exited..