CIRCULAR LINKED LIST :

```
#include<iostream>

using namespace std;

class Node {
  public:
    int key;
  int data;
  Node * next;
  Node * previous;

  Node() {
    key = 0;
    data = 0;
    next = NULL;
    previous = NULL;
  }
  Node(int k, int d) {
    key = k;
    data = d;
  }
};

class DoublyLinkedList {

  public:
    Node * head;

  DoublyLinkedList() {
    head = NULL;
  }
  DoublyLinkedList(Node * n) {
    head = n;
  }

  // 1. CHeck if node exists using key value

  Node * nodeExists(int k) {
    Node * temp = NULL;
    Node * ptr = head;

    while (ptr != NULL) {
      if (ptr - > key == k) {
        temp = ptr;
      }
      ptr = ptr - > next;
    }

    return temp;
  }
```

```
  // 2. Append a node to the list

  void appendNode(Node * n) {
    if (nodeExists(n - > key) != NULL) {
      cout << "Node Already exists with key value : " << n - > key
<< ". Append another node with different Key value" << endl;
    } else {
      if (head == NULL) {
        head = n;
        cout << "Node Appended as Head Node" << endl;
      } else {
        Node * ptr = head;
        while (ptr - > next != NULL) {
          ptr = ptr - > next;
        }
        ptr - > next = n;
        n - > previous = ptr;
        cout << "Node Appended" << endl;
      }
    }
  }

  // 3. Prepend Node - Attach a node at the start
  void prependNode(Node * n) {
    if (nodeExists(n - > key) != NULL) {
      cout << "Node Already exists with key value : " << n - > key
<< ". Append another node with different Key value" << endl;
    } else {
      if (head == NULL) {
        head = n;
        cout << "Node Prepended as Head Node" << endl;
      } else {
        head - > previous = n;
        n - > next = head;
        head = n;
        cout << "Node Prepended" << endl;
      }

    }
  }

  // 4. Insert a Node after a particular node in the list
  void insertNodeAfter(int k, Node * n) {
    Node * ptr = nodeExists(k);
    if (ptr == NULL) {
      cout << "No node exists with key value: " << k << endl;
    } else {
      if (nodeExists(n - > key) != NULL) {
        cout << "Node Already exists with key value : " << n - > key
<< ". Append another node with different Key value" << endl;
      } else {
        Node * nextNode = ptr - > next;
        // inserting at the end
        if (nextNode == NULL) {
```

```
        ptr - > next = n;
        n - > previous = ptr;
        cout << "Node Inserted at the END" << endl;
      }

      //inserting in between
      else {
        n - > next = nextNode;
        nextNode - > previous = n;
        n - > previous = ptr;
        ptr - > next = n;

        cout << "Node Inserted in Between" << endl;

      }

    }
  }
}

// 5. Delete node by unique key. Basically De-Link not delete
void deleteNodeByKey(int k) {
  Node * ptr = nodeExists(k);
  if (ptr == NULL) {
    cout << "No node exists with key value: " << k << endl;
  } else {

    if (head - > key == k) {
      head = head - > next;
      cout << "Node UNLINKED with keys value : " << k << endl;
    } else {
      Node * nextNode = ptr - > next;
      Node * prevNode = ptr - > previous;
      // deleting at the end
      if (nextNode == NULL) {

        prevNode - > next = NULL;
        cout << "Node Deleted at the END" << endl;
      }

      //deleting in between
      else {
        prevNode - > next = nextNode;
        nextNode - > previous = prevNode;
        cout << "Node Deleted in Between" << endl;

      }
    }
  }
}

// 6th update node
void updateNodeByKey(int k, int d) {
```

```
    Node * ptr = nodeExists(k);
    if (ptr != NULL) {
      ptr - > data = d;
      cout << "Node Data Updated Successfully" << endl;
    } else {
      cout << "Node Doesn't exist with key value : " << k << endl;
    }

  }

  // 7th printing
  void printList() {
    if (head == NULL) {
      cout << "No Nodes in Doubly Linked List";
    } else {
      cout << endl << "Doubly Linked List Values : ";
      Node * temp = head;

      while (temp != NULL) {
        cout << "(" << temp - > key << "," << temp - > data << ")
<--> ";
        temp = temp - > next;
      }
    }

  }

};

int main() {

  DoublyLinkedList obj;
  int option;
  int key1, k1, data1;
  do {
    cout << "\nWhat operation do you want to perform? Select Option
number. Enter 0 to exit." << endl;
    cout << "1. appendNode()" << endl;
    cout << "2. prependNode()" << endl;
    cout << "3. insertNodeAfter()" << endl;
    cout << "4. deleteNodeByKey()" << endl;
    cout << "5. updateNodeByKey()" << endl;
    cout << "6. print()" << endl;
    cout << "7. Clear Screen" << endl << endl;

    cin >> option;
    Node * n1 = new Node();
    //Node n1;

    switch (option) {
    case 0:
      break;
    case 1:
      cout << "Append Node Operation \nEnter key & data of the Node
```

```
to be Appended" << endl;
      cin >> key1;
      cin >> data1;
      n1 - > key = key1;
      n1 - > data = data1;
      obj.appendNode(n1);
      //cout<<n1.key<<" = "<<n1.data<<endl;
      break;

    case 2:
      cout << "Prepend Node Operation \nEnter key & data of the Node
to be Prepended" << endl;
      cin >> key1;
      cin >> data1;
      n1 - > key = key1;
      n1 - > data = data1;
      obj.prependNode(n1);
      break;

    case 3:
      cout << "Insert Node After Operation \nEnter key of existing
Node after which you want to Insert this New node: " << endl;
      cin >> k1;
      cout << "Enter key & data of the New Node first: " << endl;
      cin >> key1;
      cin >> data1;
      n1 - > key = key1;
      n1 - > data = data1;

      obj.insertNodeAfter(k1, n1);
      break;

    case 4:

      cout << "Delete Node By Key Operation - \nEnter key of the
Node to be deleted: " << endl;
      cin >> k1;
      obj.deleteNodeByKey(k1);

      break;
    case 5:
      cout << "Update Node By Key Operation - \nEnter key & NEW data
to be updated" << endl;
      cin >> key1;
      cin >> data1;
      obj.updateNodeByKey(key1, data1);

      break;
    case 6:
      obj.printList();

      break;
    case 7:
      system("cls");
```

```
      break;
    default:
      cout << "Enter Proper Option number " << endl;
    }

  } while (option != 0);

  return 0;
}
```