SUBMITTED TO: SIR USMAN JOYIA

DATA STRUCTURES

ASSIGNEMENT 2

KHUBAIB SHABBIR
F219634
CS-4E

QUESTION NO 1:

CODE:

```
// Khubaib Shabbir
// Question no 1
// PROGRAM TO PERFORM THE CURD OPERATIONS ON THE LINKED LIST

#include<iostream>
using namespace std;



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

class linked_list {

public:
	node* head;
	node* current;

	linked_list()
	{
		head = NULL;
		current = NULL;
	}

	// the function to make head at the start of the link list.

	void insertNodeAtBeginning(int value)
	{
		node* temp = new node;
		temp->data = value;
		temp->next = head;
		head = temp;
		current = head;
		display();
	}

	// the function to make insertion at the middle of the two nodes.

	void insertNodeAtmiddle(int value, int key)
	{
		node* temp = new node;
		temp->data = value;
		node* curr;
		curr = head;

		while (curr->data != key) // while will operate till the data reached the
required node.
		{
			curr = curr->next;

		}
```

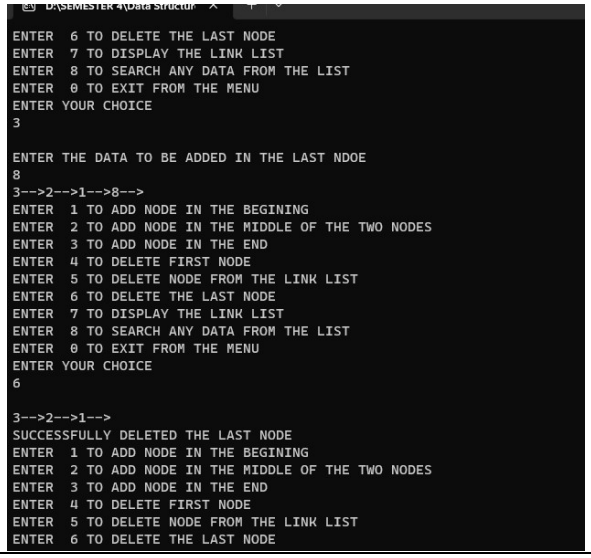
```
		temp->next = curr->next;
		curr->next = temp;
		display();
	}

	void insertnodeatend(int val)
	{
		node* temp = new node;
		temp->data = val;
		node* curr;
		curr = head;
		while ( curr->next!= NULL)
		{
			curr = curr->next;
		}
		temp->next = curr->next;
		curr->next = temp;
		display();
	}
	bool deleteFirstNode()
	{

		if (head == NULL)
		{
			cout << "LIST IS EMPTY" << endl;
			return false;
		}


		else
		{
		    if (head->next == NULL)
			{
				delete head;
			}
			else
			{
				node* temp;
				temp = head;
				current = current->next;
				head = current;
				delete temp;
				display();
			}
			return true;

		}
	}

	bool deleteNode(int key)
	{

		if (head == NULL)
		{
			cout << "LIST IS EMPTY" << endl;
			return false;
		}
		else
```

```
	{
		node* cur;
		cur = head;
		node* temp;
		temp = head;
			while (cur)
			{
			if (cur->next->data == key)
			{
				temp = cur->next;
				cur->next = temp->next;
				delete temp;
				display();
				return true;
			}
			cur = cur->next;

			}
				return false;

			}
	}
	bool deleteLastNode()
	{
		if (head == NULL)
		{
			cout << "LIST IS EMPTY" << endl;
			return false;
		}
		else
		{
			node* cur;
			cur = head;
			node* temp;

			while (cur)
			{
				if (cur->next->next== NULL)
				{
					temp = cur->next;
					cur->next = temp->next;
					delete temp;
					display();
					return true;
				}
				cur = cur->next;
			}
			return false;
		}
	}
	void display()
	{
		node* cur;
		cur = head;
		while (cur!= NULL)
		{
			cout << cur->data << "-->";
			cur = cur->next;
```

```
                }
                cout << endl;
        }
        // Method to search for a node with a given value in the list
        void search(int value)
        {
                node* curr1;
                curr1 = head;
                while (curr1->next != NULL)
                {
                        if (curr1->data == value)
                        {
                                cout << " value found : " << value << endl; // If we find
the node with the given value, print a message
                                break;
                                curr1 = curr1->next; // Traverse the list
                        }
                }
        }




};

int main()
{
        linked_list obj;
        int choice, value, key;

        do{
        cout << "ENTER  1 TO ADD NODE IN THE BEGINING" << endl;
        cout << "ENTER  2 TO ADD NODE IN THE MIDDLE OF THE TWO NODES" << endl;
        cout << "ENTER  3 TO ADD NODE IN THE END" << endl;
        cout << "ENTER  4 TO DELETE FIRST NODE " << endl;
        cout << "ENTER  5 TO DELETE NODE FROM THE LINK LIST" << endl;
        cout << "ENTER  6 TO DELETE THE LAST NODE" << endl;
        cout << "ENTER  7 TO DISPLAY THE LINK LIST" << endl;
        cout << "ENTER  8 TO SEARCH ANY DATA FROM THE LIST " << endl;
        cout << "ENTER  0 TO EXIT FROM THE MENU " << endl;
        cout << "ENTER YOUR CHOICE " << endl;
        cin >> choice;
        cout << endl;
                switch (choice)
                {
                case 1:
                        cout << "ENTER THE DATA TO BE ENTER IN THE NODE" << endl;
                        cin >> value;
                        obj.insertNodeAtBeginning(value);
                        break;
                case 2:
                        cout << "ENTER THE DATA TO BE ENTER IN THE MIDDLE OF TWE  NODES "
<< endl;
                        cin >> value;
                        cout << "ENTER THE VALUE OF THE DATA OF THE NODE TO ADD NEW NODE
AFTER IT " << endl;
                        cin >> key;
                        obj.insertNodeAtmiddle(value, key);
```

```
                        break;
                case 3:
                        cout << "ENTER THE DATA TO BE ADDED IN THE LAST NDOE" << endl;
                        cin >> value;
                        obj.insertnodeatend(value);
                        break;
                case 4:
                        obj.deleteFirstNode();
                        cout << "SUCCESSFULLY DELETED THE FIRST NODE " << endl;
                        break;
                case 5:
                        cout << "ENTER THE VALUE OF THE DATA OF THE NODE TO DELETE THAT
NODE " << endl;
                        cin >> key;
                        obj.deleteNode(key);
                        break;
                case 6:
                        obj.deleteLastNode();
                        cout << "SUCCESSFULLY DELETED THE LAST NODE " << endl;
                        break;
                case 7:
                        obj.display();
                        break;
                case 8:
                        cout << "ENTER THE DATA TO SEARCH IN THE LINK LIST " << endl;
                        cin >> value;
                        obj.search(value);
                        break;

                default:
                        cout << "PLEASE ENTER THE VALID INPUT " << endl;
                        cin >> choice;

                }

        } while (choice != 0);
        return 0;
}
```

SCREENSHOT:



```
ENTER  1 TO ADD NODE IN THE BEGINING
ENTER  2 TO ADD NODE IN THE MIDDLE OF THE TWO NODES
ENTER  3 TO ADD NODE IN THE END
ENTER  4 TO DELETE FIRST NODE
ENTER  5 TO DELETE NODE FROM THE LINK LIST
ENTER  6 TO DELETE THE LAST NODE
ENTER  7 TO DISPLAY THE LINK LIST
ENTER  8 TO SEARCH ANY DATA FROM THE LIST
ENTER  0 TO EXIT FROM THE MENU
ENTER YOUR CHOICE
8

ENTER THE DATA TO SEARCH IN THE LINK LIST
3
 value found : 3
```

```
ENTER YOUR CHOICE
6

3-->2-->
SUCCESSFULLY DELETED THE LAST NODE
ENTER  1 TO ADD NODE IN THE BEGINING
ENTER  2 TO ADD NODE IN THE MIDDLE OF THE TWO NODES
ENTER  3 TO ADD NODE IN THE END
ENTER  4 TO DELETE FIRST NODE
ENTER  5 TO DELETE NODE FROM THE LINK LIST
ENTER  6 TO DELETE THE LAST NODE
ENTER  7 TO DISPLAY THE LINK LIST
ENTER  8 TO SEARCH ANY DATA FROM THE LIST
ENTER  0 TO EXIT FROM THE MENU
ENTER YOUR CHOICE
```

QUESTION NO 2:

CODE:

```
// Khubaib Shabbir
// Question no 2
// PROGRAM TO PERFORM THE CURD OPERATIONS ON THE DOUBLY CIRCULAR LINKED LIST

#include<iostream>
using namespace std;



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

class linked_list {

public:
	node* head;
	node* tail;
```

```
	linked_list()
	{
		head = NULL;
		tail = NULL;
	}

	// the function to make head at the start of the link list.

	void insertNodeAtBeginning(int value)
	{
		if (head == NULL)
		{
			node* temp = new node;
			temp->data = value;
			temp->next = head;
			temp->previous = head;
			head = temp;
			tail = head;
			tail->next = head;
			head->previous = tail;
			display();
		}
		else
		{
			node* temp = new node;
			temp->data = value;
			temp->next = head;
			temp->previous = head;
			head = temp;
			tail->next = head;
			head->previous = tail;
			display();

		}
	}

	// the function to make insertion at the middle of the two nodes.

	void insertNodeAtmiddle(int value, int key)
	{
		node* temp = new node;
		temp->data = value;
		node* curr;
		curr = head;

		while (curr->data != key) // while will operate till the data reached the
required node.
		{
			curr = curr->next;

		}
		temp->previous = curr;
		temp->next = curr->next;
		temp->previous->next = curr->next;
		temp->next->previous = temp;
		curr = temp;
		display();
	}
```

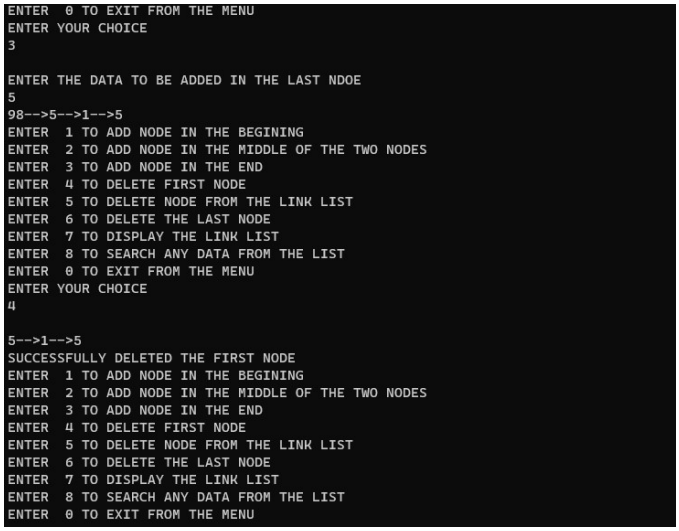
```
void insertnodeatend(int val)
{
	node* temp = new node;
	temp->data = val;
	node* curr;
	curr = head;
	temp->previous = tail;
	tail->next = temp;
	tail = temp;
	tail->next = head;
	head->previous = tail;
	display();
}
bool deleteFirstNode()
{

	if (head == NULL)
	{
		cout << "LIST IS EMPTY" << endl;
		return false;
	}


	else
	{
		if (head->next == NULL)
		{
			delete head;

		}
		else
		{
			node* temp;
			temp = head;
			head = head->next;
			delete temp;
			tail->next = head;
			head->previous = tail;
			display();
		}
		return true;

	}
}

bool deleteNode(int key)
{

	if (head == NULL)
	{
		cout << "LIST IS EMPTY" << endl;
		return false;
	}
	else
	{
		node* cur;
		cur = head;
```

```
                node* temp;
                temp = head;
                while (cur->next !=head)
                {
                        if (cur->next->data == key)
                        {
                                temp = cur->next;
                                cur->next = temp->next;
                                temp->next->previous = cur;
                                delete temp;
                                display();
                                return true;
                        }
                        cur = cur->next;

                }
                return false;

        }
}
bool deleteLastNode()
{
        if (head == NULL)
        {
                cout << "LIST IS EMPTY" << endl;
                return false;
        }
        else
        {
                node* temp = tail;
                tail = tail->previous;
                tail->next = head;
                head->previous = tail;
                delete temp;
                return true;
        }
}
void display()
{
        node* cur;
        cur = head;
        while (cur->next != head)
        {
                cout << cur->data << "-->";
                cur = cur->next;
        }
        cout << cur->data ;
        cout << endl;
}
// Method to search for a node with a given value in the list
void search(int value)
{
        node* curr1;
        curr1 = head;
        if (curr1->data == value)
        {
                cout << " value found : " << value << endl;
                return;
```

```
		}
		curr1 = curr1->next;
		while (curr1!= head)
		{
			if (curr1->data == value)
			{
				cout << " value found : " << value << endl; // If we find
the node with the given value, print a message
				return;

			}
			curr1 = curr1->next; // Traverse the list
		}
		cout << "VALUE NOT FOUND" << endl;

	}




};

int main()
{
	linked_list obj;
	int choice, value, key;

	do {
		cout << "ENTER  1 TO ADD NODE IN THE BEGINING" << endl;
		cout << "ENTER  2 TO ADD NODE IN THE MIDDLE OF THE TWO NODES" << endl;
		cout << "ENTER  3 TO ADD NODE IN THE END" << endl;
		cout << "ENTER  4 TO DELETE FIRST NODE " << endl;
		cout << "ENTER  5 TO DELETE NODE FROM THE LINK LIST" << endl;
		cout << "ENTER  6 TO DELETE THE LAST NODE" << endl;
		cout << "ENTER  7 TO DISPLAY THE LINK LIST" << endl;
		cout << "ENTER  8 TO SEARCH ANY DATA FROM THE LIST " << endl;
		cout << "ENTER  0 TO EXIT FROM THE MENU " << endl;
		cout << "ENTER YOUR CHOICE " << endl;
		cin >> choice;
		cout << endl;
		switch (choice)
		{
		case 1:
			cout << "ENTER THE DATA TO BE ENTER IN THE NODE" << endl;
			cin >> value;
			obj.insertNodeAtBeginning(value);
			break;
		case 2:
			cout << "ENTER THE DATA TO BE ENTER IN THE MIDDLE OF TWE  NODES "
<< endl;
			cin >> value;
			cout << "ENTER THE VALUE OF THE DATA OF THE NODE TO ADD NEW NODE
AFTER IT " << endl;
			cin >> key;
			obj.insertNodeAtmiddle(value, key);
			break;
		case 3:
			cout << "ENTER THE DATA TO BE ADDED IN THE LAST NDOE" << endl;
```

```
                        cin >> value;
                        obj.insertnodeatend(value);
                        break;
                case 4:
                        obj.deleteFirstNode();
                        cout << "SUCCESSFULLY DELETED THE FIRST NODE " << endl;
                        break;
                case 5:
                        cout << "ENTER THE VALUE OF THE DATA OF THE NODE TO DELETE THAT
NODE " << endl;
                        cin >> key;
                        obj.deleteNode(key);
                        break;
                case 6:
                        obj.deleteLastNode();
                        cout << "SUCCESSFULLY DELETED THE LAST NODE " << endl;
                        break;
                case 7:
                        obj.display();
                        break;
                case 8:
                        cout << "ENTER THE DATA TO SEARCH IN THE LINK LIST " << endl;
                        cin >> value;
                        obj.search(value);
                        break;

                default:
                        cout << "PLEASE ENTER THE VALID INPUT " << endl;
                        cin >> choice;

                }

        } while (choice != 0);
        return 0;
}
```

SCREENSHOT:



QUESTION NO 3:

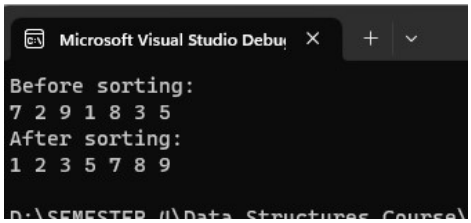
CODE:

```
// Khubaib Shabbir
// Question no 3
// Merge sort for linked list
#include <iostream>

using namespace std;

// Node structure for a singly linked list
struct Node
{
    int data;
    Node* next;
};

// Function to merge two sorted linked lists
Node* Merge(Node* head1, Node* head2)
{
    // If one of the linked lists is empty, return the other list
    if (head1 == nullptr) return head2;
    if (head2 == nullptr) return head1;

    // Merge the two sorted lists recursively
    if (head1->data < head2->data) {
        head1->next = Merge(head1->next, head2);
        return head1;
    }
    else {
        head2->next = Merge(head1, head2->next);
        return head2;
    }
}

// Function to split a linked list into two halves using the slow-fast pointer approach
void Split(Node* head, Node** left, Node** right) {
    // If the linked list is empty or has only one node, return the same list as left
and null as right
    if (head == nullptr || head->next == nullptr) {
        *left = head;
        *right = nullptr;
        return;
    }

    // Use the slow-fast pointer approach to find the middle node
    Node* slow = head;
    Node* fast = head->next;

    while (fast != nullptr) {
        fast = fast->next;
        if (fast != nullptr) {
            slow = slow->next;
            fast = fast->next;
        }
    }
```

```
    // Split the linked list into two halves at the middle node
    *left = head;
    *right = slow->next;
    slow->next = nullptr;
}

// Function to perform Merge Sort on a linked list
void MergeSort(Node** head) {
    // If the linked list is empty or has only one node, return
    if (*head == nullptr || (*head)->next == nullptr) {
        return;
    }

    // Split the linked list into two halves
    Node* left;
    Node* right;
    Split(*head, &left, &right);

    // Recursively apply Merge Sort on the two halves
    MergeSort(&left);
    MergeSort(&right);

    // Merge the two sorted halves into a single sorted list
    *head = Merge(left, right);
}

// Function to insert a new node at the beginning of a linked list
void Insert(Node** head, int data) {
    Node* newNode = new Node;
    newNode->data = data;
    newNode->next = *head;
    *head = newNode;
}

// Function to print the nodes of a linked list
void PrintList(Node* head)
{
    while (head != nullptr)
    {
        cout << head->data << " ";
        head = head->next;
    }
    cout << endl;
}

int main()
{
    // Create a linked list and insert some nodes
    Node* head = nullptr;
    Insert(&head, 5);
    Insert(&head, 3);
    Insert(&head, 8);
    Insert(&head, 1);
    Insert(&head, 9);
    Insert(&head, 2);
    Insert(&head, 7);

    // Print the unsorted linked list
```

```
    cout << "Before sorting:" << endl;
    PrintList(head);

    // Sort the linked list using Merge Sort
    MergeSort(&head);

    // Print the sorted linked list
    cout << "After sorting:" << endl;
    PrintList(head);

    return 0;
}
```

SCREENSHOT:



QUESTION NO 4:

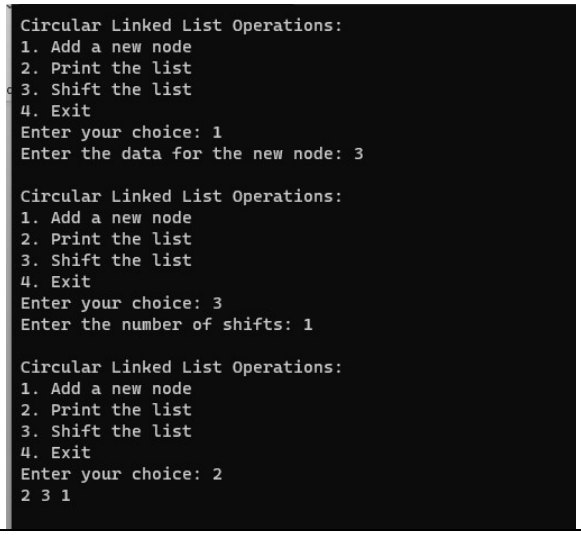
CODE:

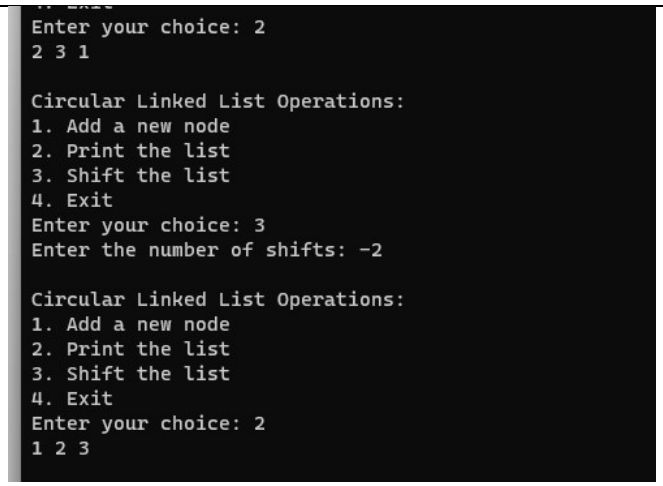
```
/*Khubaib Shabbir
 Question no4
 Program to deal with shift cases in the linked list*/


#include<iostream>
#include<cstdlib>

using namespace std;

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

    Node(int data) {
        this->data = data;
        next = NULL;
    }
};

class CircularLinkedList {
public:
    Node* head;

    CircularLinkedList() {
        head = NULL;
```

```
    }

    // function to add a new node to the list
    void add_node(int data) {
        Node* new_node = new Node(data);
        if (head == NULL) {
            head = new_node;
            new_node->next = head;
        }
        else {
            Node* curr_node = head;
            while (curr_node->next != head) {
                curr_node = curr_node->next;
            }
            curr_node->next = new_node;
            new_node->next = head;
        }
    }

    // function to print the contents of the list
    void print_list() {
        if (head == NULL) {
            cout << "List is empty!" << endl;
        }
        else {
            Node* curr_node = head;
            do {
                cout << curr_node->data << " ";
                curr_node = curr_node->next;
            } while (curr_node != head);
            cout << endl;
        }
    }

    // function to shift the nodes of the list
    void shift(int n) {
        if (head == NULL) {
            return;
        }

        if (n > 0) {
            // right shift
            for (int i = 0; i < n; i++) {
                head = head->next;
            }
        }
        else if (n < 0) {
            // left shift
            n = abs(n);
            for (int i = 0; i < n; i++) {
                head = head->next;
            }
        }
    }
};

int main() {
    CircularLinkedList list;
```

```
    int choice, data, n;
    do {
        cout << "Circular Linked List Operations:" << endl;
        cout << "1. Add a new node" << endl;
        cout << "2. Print the list" << endl;
        cout << "3. Shift the list" << endl;
        cout << "4. Exit" << endl;
        cout << "Enter your choice: ";
        cin >> choice;

        switch (choice) {
        case 1:
            cout << "Enter the data for the new node: ";
            cin >> data;
            list.add_node(data);
            break;
        case 2:
            list.print_list();
            break;
        case 3:
            cout << "Enter the number of shifts: ";
            cin >> n;
            list.shift(n);
            break;
        case 4:
            cout << "Exiting..." << endl;
            break;
        default:
            cout << "Invalid choice! Please try again." << endl;
        }

        cout << endl;
    } while (choice != 4);

    return 0;
}
```

SCREENSHOT:




Enter your choice: 2
2 3 1
Circular Linked List Operations:
1. Add a new node
2. Print the list
3. Shift the list
4. Exit
Enter your choice: 3
Enter the number of shifts: -2
Circular Linked List Operations:
1. Add a new node
2. Print the list
3. Shift the list
4. Exit
Enter your choice: 2
1 2 3