# Mahad Ghauri

233523

DSA Lab Task

    #ifndef NODE\_H

    #define NODE\_H

    #include <iostream>

    #include <string>

    using namespace std;

    class Node

    {

    private:

        int data;

        Node\* next;

    public:

        Node(int x)

        {

            data = x;

            next = NULL;

        }

        void setNext(Node\* other)

        {

            next = other;

        }

        void setData(int x)

        {

            data = x;

        }

        Node\* getNext()

        {

            return next;

        }

        int getData()

        {

            return data;

        }

    };

    #endif

#ifndef LIST\_H

#define LIST\_H

#include "NODE.H"

class List

{

public:

    Node \*head = NULL;

    Node \*tail = NULL;

    List()

    {

        head = NULL;

        tail = NULL;

    }

    void AddNode(int value)

    {

        if (head == NULL)

        {

            head = new Node(value);

            tail = head;

        }

        else

        {

            tail->setNext(new Node(value));

            tail = tail->getNext();

        }

    }

    void Add\_NewNode(int value)

    {

        Node \*addingNode = new Node(value);

        if (head == NULL)

        {

            head = addingNode;

            tail = addingNode;

            tail->setNext(head);

        }

        else

        {

            tail->setNext(addingNode);

            tail = tail->getNext();

            tail->setNext(head);

        }

    }

    void Display()

    {

        Node \*temp = head;

        cout << " List: ";

        while (temp != NULL)

        {

            cout << temp->getData() << " ";

            temp = temp->getNext();

        }

        cout << endl;

    }

    void Swap\_Nodes(int x)

    {

        Node \*beforePrevious = NULL;

        Node \*previous = NULL;

        Node \*current = head;

        x--;

        while (x--)

        {

            beforePrevious = previous;

            previous = current;

            current = current->getNext();

        }

        if (previous == head)

        {

            head = current;

        }

        else if (beforePrevious != NULL)

        {

            beforePrevious->setNext(current);

        }

        previous->setNext(current->getNext());

        current->setNext(previous);

        tail->setNext(head);

    }

    List \*CircularList()

    {

        if (head == NULL)

        {

            return this;

        }

        tail->setNext(head);

        return this;

    }

    void Circular\_Display()

    {

        Node \*temp = head;

        cout << " List: ";

        do

        {

            cout << temp->getData() << " ";

            temp = temp->getNext();

        } while (temp != head);

        cout << endl;

    }

    void Delete\_Specific\_Node(int node\_deletion)

    {

        Node \*current = head;

        Node \*previous = NULL;

        if (head != NULL)

        {

            if (head->getNext() == NULL)

            {

                head = NULL;

                delete head;

            }

            else

            {

                node\_deletion--;

                while (node\_deletion--)

                {

                    previous = current;

                    current = current->getNext();

                }

                previous->setNext(current->getNext());

                if (current == tail)

                {

                    tail = previous;

                }

                delete current;

            }

        }

    }

    List \*Update\_Node(int oldValue , int newValue)

    {

        Node \*current = head;

        do

        {

            if (current->getData() == oldValue)

            {

                current->setData(newValue   );

                break;

            }

            current = current->getNext();

        } while (current != head);

        tail->setNext(head);

        return this;

    }

};

#endif

#include "NODE.H"

#include "LIST.H"

int main()

{

    List firstList;

    int oddArray[] = {1, 3, 5, 7, 9};

    for (int i = 0; i < 5; i++)

    {

        firstList.AddNode(oddArray[i]);

    }

    firstList.Display();

    List \*circularList = firstList.CircularList();

    cout << "\n Adding a new element in the circular linked list: " << endl;

    circularList->Add\_NewNode(11);

    circularList->Circular\_Display();

    cout << "\n";

    int position = 3 , oldValue = 9, newValue = 5;

    cout << "\n Deleting Node a specific node in the list:" << endl;

    circularList->Delete\_Specific\_Node(position);

    circularList->Circular\_Display();

    cout << "\n";

    cout << "\n Swap Nodes : " << endl;

    circularList->Swap\_Nodes(position);

    circularList->Circular\_Display();

    cout << "\n Updating Node : " << endl;

    List \*updatedList = circularList->Update\_Node(oldValue , newValue);

    updatedList->Circular\_Display();

}

