**Ministerul Educaţiei și Cercetării**

**Universitatea Tehnică a Moldovei**

**Faculatea Calculatoare, Informatică și Microelectronică**

**Departamentul Ingineria Software și Automatică**



**RAPORT**

**Lucrare de Laborator nr.2**

**SDA**

**Tema:**

A efectuat:

st. gr. CR-231 Maletchi Mirela

A verificat:

asist. univ

**Chişinău 2024**

**main.cpp**

#include "Header.h"

#include <iostream>

using namespace std;

int main() {

    LinkedList lista;

    int optiune;

    do {

        cout << "\nMeniu:\n";

        cout << "1. Inserare la inceputul listei.\n";

        cout << "2. Inserare la sfarsitul listei.\n";

        cout << "3. Cautare in lista.\n";

        cout << "4. Modificare valoare nod.\n";

        cout << "5. Interschimbare noduri.\n";

        cout << "6. Afișare lista.\n";

        cout << "7. Lungimea listei.\n";

        cout << "8. Sortare lista.\n";

        cout << "9. Eliberare memorie lista.\n";

        cout << "0. Iesire din program.\n";

        cout << "Alegeti o optiune: ";

        cin >> optiune;

        switch (optiune) {

            case 1: {

                char denumire[100], capitala[50];

                float suprafata, pib;

                int populatie;

                cout << "Introduceti detalii pentru statul de inserat la inceputul listei:\n";

                cout << "Denumire: ";

                cin >> denumire;

                cout << "Capitala: ";

                cin >> capitala;

                cout << "Suprafata: ";

                cin >> suprafata;

                cout << "Populatie: ";

                cin >> populatie;

                cout << "PIB: ";

                cin >> pib;

                lista.insertAtBeginning(denumire, capitala, suprafata, populatie, pib);

                break;

            }

            case 2: {

                char denumire[100], capitala[50];

                float suprafata, pib;

                int populatie;

                cout << "Introduceti detalii pentru statul de inserat la sfarsitul listei:\n";

                cout << "Denumire: ";

                cin >> denumire;

                cout << "Capitala: ";

                cin >> capitala;

                cout << "Suprafata: ";

                cin >> suprafata;

                cout << "Populatie: ";

                cin >> populatie;

                cout << "PIB: ";

                cin >> pib;

                lista.insertAtEnd(denumire, capitala, suprafata, populatie, pib);

                break;

            }

            case 3: {

                char denumireCautata[100];

                cout << "Introduceti denumirea statului cautat: ";

                cin >> denumireCautata;

                if (lista.search(denumireCautata)) {

                    cout << "Statul a fost gasit in lista.\n";

                } else {

                    cout << "Statul nu a fost gasit in lista.\n";

                }

                break;

            }

            case 4: {

                char oldDenumire[100], newDenumire[100];

                float newSuprafata, newPib;

                int newPopulatie;

                cout << "Introduceti denumirea statului de modificat: ";

                cin >> oldDenumire;

                cout << "Introduceti noile detalii pentru statul " << oldDenumire << ":\n";

                cout << "Denumire: ";

                cin >> newDenumire;

                cout << "Suprafata: ";

                cin >> newSuprafata;

                cout << "Populatie: ";

                cin >> newPopulatie;

                cout << "PIB: ";

                cin >> newPib;

                lista.modify(oldDenumire, newDenumire, newSuprafata, newPopulatie, newPib);

                break;

            }

            case 5: {

                char denumire1[100], denumire2[100];

                cout << "Introduceti denumirea primului stat: ";

                cin >> denumire1;

                cout << "Introduceti denumirea celui de-al doilea stat: ";

                cin >> denumire2;

                lista.swapNodes(denumire1, denumire2);

                break;

            }

            case 6:

                cout << "Lista de stat este:\n";

                lista.displayList();

                break;

            case 7:

                cout << "Lungimea listei este: " << lista.length() << "\n";

                break;

            case 8:

                lista.bubbleSort();

                cout << "Lista a fost sortata.\n";

                break;

            case 9:

                lista.freeList();

                cout << "Memoria a fost eliberata.\n";

                break;

            case 0:

                cout << "Iesire din program.\n";

                break;

            default:

                cout << "Optiune invalida. Alegeti din nou.\n";

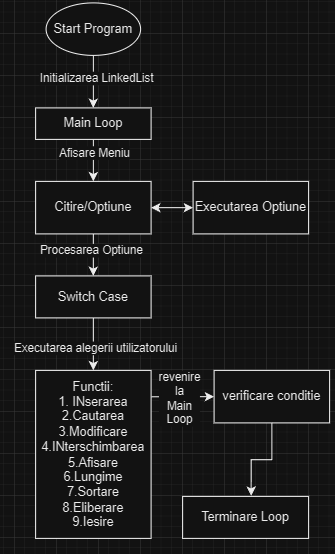
                break;

        }

    } while (optiune != 0);

    return 0;

}

****

**Fig.1 schema logica pentru main.cpp**

**Header.h**

#ifndef HEADER\_H

#define HEADER\_H

struct StateNode {

    char denumire[100];

    char capitala[50];

    float suprafata;

    int populatie;

    float pib;

    StateNode\* next;

};

class LinkedList {

private:

    StateNode\* head;

public:

    LinkedList();

    ~LinkedList();

    void insertAtBeginning(char denumire[], char capitala[], float suprafata, int populatie, float pib);

    void insertAtEnd(char denumire[], char capitala[], float suprafata, int populatie, float pib);

    bool search(char denumire[]);

    void modify(char oldDenumire[], char newDenumire[], float newSuprafata, int newPopulatie, float newPib);

    void swapNodes(char denumire1[], char denumire2[]);

    void displayList();

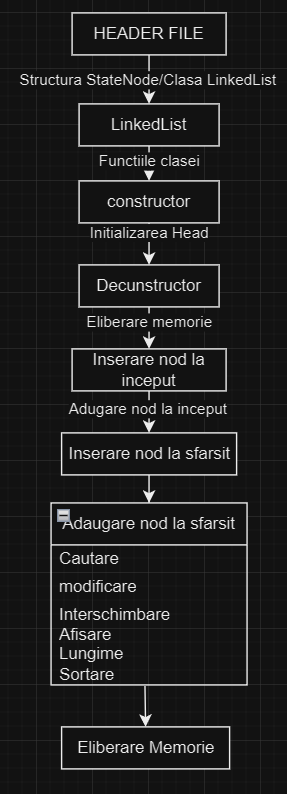
    int length();

    void bubbleSort();

    void freeList();

};

#endif /\* HEADER\_H \*/

****

**Fig.2 schema logica pentru Header.h**

**Functii.cpp**

#include "Header.h"

#include <iostream>

#include <cstring>

using namespace std;

LinkedList::LinkedList() {

    head = nullptr;

}

LinkedList::~LinkedList() {

    freeList();

}

void LinkedList::insertAtBeginning(char denumire[], char capitala[], float suprafata, int populatie, float pib) {

    StateNode\* newNode = new StateNode;

    strcpy(newNode->denumire, denumire);

    strcpy(newNode->capitala, capitala);

    newNode->suprafata = suprafata;

    newNode->populatie = populatie;

    newNode->pib = pib;

    newNode->next = head;

    head = newNode;

}

void LinkedList::insertAtEnd(char denumire[], char capitala[], float suprafata, int populatie, float pib) {

    StateNode\* newNode = new StateNode;

    strcpy(newNode->denumire, denumire);

    strcpy(newNode->capitala, capitala);

    newNode->suprafata = suprafata;

    newNode->populatie = populatie;

    newNode->pib = pib;

    newNode->next = nullptr;

    if (head == nullptr) {

        head = newNode;

        return;

    }

    StateNode\* temp = head;

    while (temp->next != nullptr) {

        temp = temp->next;

    }

    temp->next = newNode;

}

bool LinkedList::search(char denumire[]) {

    StateNode\* temp = head;

    while (temp != nullptr) {

        if (strcmp(temp->denumire, denumire) == 0) {

            return true;

        }

        temp = temp->next;

    }

    return false;

}

void LinkedList::modify(char oldDenumire[], char newDenumire[], float newSuprafata, int newPopulatie, float newPib) {

    StateNode\* temp = head;

    while (temp != nullptr) {

        if (strcmp(temp->denumire, oldDenumire) == 0) {

            strcpy(temp->denumire, newDenumire);

            temp->suprafata = newSuprafata;

            temp->populatie = newPopulatie;

            temp->pib = newPib;

            return;

        }

        temp = temp->next;

    }

    cout << "Statul " << oldDenumire << " nu a fost gasit in lista.\n";

}

void LinkedList::swapNodes(char denumire1[], char denumire2[]) {

    StateNode\* temp1 = head;

    StateNode\* temp2 = head;

    StateNode\* prev1 = nullptr;

    StateNode\* prev2 = nullptr;

    if (!search(denumire1) || !search(denumire2)) {

        cout << "Cel putin unul dintre state nu exista in lista.\n";

        return;

    }

    if (strcmp(denumire1, denumire2) == 0) {

        cout << "Nu se poate interschimba acelasi nod.\n";

        return;

    }

    while (temp1 != nullptr && strcmp(temp1->denumire, denumire1) != 0) {

        prev1 = temp1;

        temp1 = temp1->next;

    }

    while (temp2 != nullptr && strcmp(temp2->denumire, denumire2) != 0) {

        prev2 = temp2;

        temp2 = temp2->next;

    }

    if (temp1 == nullptr || temp2 == nullptr) {

        cout << "Eroare la cautarea nodurilor.\n";

        return;

    }

    if (prev1 != nullptr) {

        prev1->next = temp2;

    } else {

        head = temp2;

    }

    if (prev2 != nullptr) {

        prev2->next = temp1;

    } else {

        head = temp1;

    }

    StateNode\* temp = temp1->next;

    temp1->next = temp2->next;

    temp2->next = temp;

}

void LinkedList::displayList() {

    StateNode\* temp = head;

    int count = 1;

    while (temp != nullptr) {

        cout << "Statul " << count << ":\n";

        cout << "Denumire: " << temp->denumire << "\n";

        cout << "Capitala: " << temp->capitala << "\n";

        cout << "Suprafata: " << temp->suprafata << "\n";

        cout << "Populatie: " << temp->populatie << "\n";

        cout << "PIB: " << temp->pib << "\n";

        cout << "\n";

        temp = temp->next;

        count++;

    }

}

int LinkedList::length() {

    StateNode\* temp = head;

    int count = 0;

    while (temp != nullptr) {

        count++;

        temp = temp->next;

    }

    return count;

}

void LinkedList::bubbleSort() {

    int n = length();

    if (n <= 1) return;

    for (int i = 0; i < n - 1; i++) {

        StateNode\* current = head;

        StateNode\* nextNode = head->next;

        for (int j = 0; j < n - i - 1; j++) {

            if (strcmp(current->denumire, nextNode->denumire) > 0) {

                swap(current->denumire, nextNode->denumire);

                swap(current->capitala, nextNode->capitala);

                swap(current->suprafata, nextNode->suprafata);

                swap(current->populatie, nextNode->populatie);

                swap(current->pib, nextNode->pib);

            }

            current = nextNode;

            nextNode = nextNode->next;

        }

    }

}

void LinkedList::freeList() {

    StateNode\* temp = head;

    while (temp != nullptr) {

        StateNode\* prev = temp;

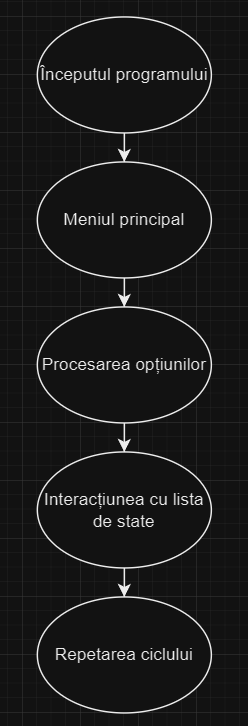
        temp = temp->next;

        delete prev;

    }

    head = nullptr;

}

****

**Fig.3 schema logica pentru Functii.cpp**

**Concluzie:**

În urma acestui laborator, s-a constatat că organizarea codului în clase și metode facilitează dezvoltarea și menținerea proiectelor individuale. Implementarea corectă a funcționalităților și gestionarea memoriei sunt aspecte esențiale pentru prevenirea erorilor și asigurarea performanței optime a aplicațiilor. Este important să se acorde atenție comunicării cu alți dezvoltatori și să se utilizeze resursele disponibile pentru a depăși obstacolele întâlnite în timpul dezvoltării individuale.