Ministerul Educației și Cercetării al Republicii Moldova
CENTRUL DE EXCELENȚĂ ÎN INFORMATICĂ ȘI TEHNOLOGII INFORMAȚIONALE

Lucrare individuală N-1

Tema: Elaborarea programelor destinate prelucrării structurilor dinamice de date

Disciplina: Programarea calculatoarelor
Varianta: 26

Grupa: P-2423
Elev: Pricop Maxim

Profesor: Cojocaru Liuba

Chișinău, 2025

Sarcina

De compus programul de creare a unei liste circulare cu elemente de tip INTEGER, apoi de contorizat elementele acestei liste ce au "vecini" egali.

Rezultat

```
Meniu:
1. Adauga element
2. Sterge element
3. Afiseaza lista
4. Numarul de elemente cu vecini egali
5. Sterge lista
6. Iesire
Optiunea aleasa: 1
Valoarea de adaugat: 3
Pozitia la care se adauga (0 - inceput, 0 - sfarsit): 0
Element adaugat cu succes!



Meniu:
1. Adauga element
2. Sterge element
3. Afiseaza lista
4. Numarul de elemente cu vecini egali
5. Sterge lista
6. Iesire
Optiunea aleasa: 1
Valoarea de adaugat: 6
Pozitia la care se adauga (0 - inceput, 1 - sfarsit): 1
Element adaugat cu succes!



Meniu:
1. Adauga element
2. Sterge element
3. Afiseaza lista
4. Numarul de elemente cu vecini egali
5. Sterge lista
6. Iesire
Optiunea aleasa: 1
Valoarea de adaugat: 3
Pozitia la care se adauga (0 - inceput, 2 - sfarsit): 2
Element adaugat cu succes!



Meniu:
1. Adauga element
2. Sterge element
3. Afiseaza lista
4. Numarul de elemente cu vecini egali
5. Sterge lista
6. Iesire
Optiunea aleasa: 3
Lista: 3, 6, 3.



Meniu:
1. Adauga element
2. Sterge element
3. Afiseaza lista
4. Numarul de elemente cu vecini egali
5. Sterge lista
6. Iesire
Optiunea aleasa: 4
Numarul de elemente cu vecini egali: 1
```

```
Meniu:
1. Adauga element
2. Sterge element
3. Afiseaza lista
4. Numarul de elemente cu vecini egali
5. Sterge lista
6. Iesire
Optiunea aleasa: 1
Valoarea de adaugat: 5
Pozitia la care se adauga (0 - inceput, 5 - sfarsit): 0
Element adaugat cu succes!



Meniu:
1. Adauga element
2. Sterge element
3. Afiseaza lista
4. Numarul de elemente cu vecini egali
5. Sterge lista
6. Iesire
Optiunea aleasa: 3
Lista: 5, 8, 3, 5, 8, 5.



Meniu:
1. Adauga element
2. Sterge element
3. Afiseaza lista
4. Numarul de elemente cu vecini egali
5. Sterge lista
6. Iesire
Optiunea aleasa: 4
Numarul de elemente cu vecini egali: 1
```

```
Meniu:
1. Adauga element
2. Sterge element
3. Afiseaza lista
4. Numarul de elemente cu vecini egali
5. Sterge lista
6. Iesire
Optiunea aleasa: 1
Valoarea de adaugat: 9
Pozitia la care se adauga (0 - inceput, 2 - sfarsit): 2
Element adaugat cu succes!



Meniu:
1. Adauga element
2. Sterge element
3. Afiseaza lista
4. Numarul de elemente cu vecini egali
5. Sterge lista
6. Iesire
Optiunea aleasa: 3
Lista: 9, 9, 9.



Meniu:
1. Adauga element
2. Sterge element
3. Afiseaza lista
4. Numarul de elemente cu vecini egali
5. Sterge lista
6. Iesire
Optiunea aleasa: 4
Numarul de elemente cu vecini egali: 3
```

Rezolvare

Programul este modular (alcătuit din 3 fișiere: 1 header și 2 .cpp). Am decis ca să implementez lista lănțuită printr-o clasă pentru simplitate.

Header

```
#pragma once
#include <cstddef>

class CircularDoublyLinkedList {
    private:
        struct Node {
            public:
                int data;
                Node *next;
                Node *prev;

                Node(int data);
        };

        Node *tail = nullptr;
        std::size_t listSize = 0;

    public:
        CircularDoublyLinkedList();
        ~CircularDoublyLinkedList();

        Node *getHead() const noexcept { return tail ? tail->next : nullptr; }
        Node *getTail() const noexcept { return tail; }

        bool push(int value, std::size_t position);
        bool pop(std::size_t position);

        std::size_t countElementsWithEqualNeighbors() const;

        void clear() noexcept;
        bool isEmpty() const noexcept { return listSize == 0; }
        std::size_t size() const noexcept { return listSize; }
};
```

Cod Listă

```cpp
#include "../include/linkedLists/CircularLinkedList.hpp"

CircularDoublyLinkedList::Node::Node(int data) : data(data), next(this), prev(this) {}

CircularDoublyLinkedList::CircularDoublyLinkedList() : tail(nullptr), listSize(0) {}
CircularDoublyLinkedList::~CircularDoublyLinkedList() { clear(); }

bool CircularDoublyLinkedList::push(int value, std::size_t position) {
    if (position > listSize) return false;
    Node *newNode = new Node(value);

    if (listSize == 0) {
        tail = newNode;
        listSize++;

        return true;
    }

    Node *head = tail->next;
    Node *oldNode = nullptr;
    if (position == 0) {
        oldNode = tail;
    } else {
        std::size_t insertPosition = position - 1;

        if (insertPosition <= listSize / 2) {
            oldNode = head;

            for (std::size_t i = 0; i < insertPosition; i++) {
                oldNode = oldNode->next;
            }
        } else {
            oldNode = tail;

            for (std::size_t i = listSize - 1; i > insertPosition; i--) {
                oldNode = oldNode->prev;
            }
        }
    }

    newNode->next = oldNode->next;
    newNode->prev = oldNode;

    oldNode->next->prev = newNode;
    oldNode->next = newNode;
```

```
    if (position == listSize) tail = newNode;

    listSize++;
    return true;
}

bool CircularDoublyLinkedList::pop(std::size_t position) {
    if (listSize == 0 || position >= listSize) return false;
    if (listSize == 1) {
        delete tail;

        tail = nullptr;
        listSize = 0;

        return true;
    }

    Node *head = tail->next;

    if (position == 0) {
        Node *newHead = head->next;

        tail->next = newHead;
        newHead->prev = tail;

        head->next = nullptr;
        head->prev = nullptr;
        delete head;

        listSize--;
        return true;
    } else if (position == listSize - 1) {
        Node *newTail = tail->prev;

        newTail->next = head;
        head->prev = newTail;

        tail->next = nullptr;
        tail->prev = nullptr;
        delete tail;

        tail = newTail;
        listSize--;

        return true;
    }
```

```
    Node *target = nullptr;

    if (position <= listSize / 2) {
        target = head;
        for (std::size_t i = 0; i < position; i++) {
            target = target->next;
        }
    } else {
        target = tail;
        for (std::size_t i = listSize - 1; i > position; i--) {
            target = target->prev;
        }
    }

    target->prev->next = target->next;
    target->next->prev = target->prev;

    delete target;
    listSize--;

    return true;
}

std::size_t CircularDoublyLinkedList::countElementsWithEqualNeighbors() const {
    if (listSize < 3 || tail == nullptr) return 0;

    std::size_t counter = 0;
    Node *currentNode = tail->next;
    for (std::size_t i = 0; i < listSize; ++i) {
        if (currentNode->prev->data == currentNode->next->data) ++counter;

        currentNode = currentNode->next;
    }
    return counter;
}

void CircularDoublyLinkedList::clear() noexcept {
    if (listSize == 0 || tail == nullptr) return;

    if (listSize == 1) {
        delete tail;

        tail = nullptr;
        listSize = 0;

        return;
    }
```

```
    Node *currentNode = tail->next;
    Node *endNode = tail;

    while (currentNode != endNode) {
        Node *next = currentNode->next;
        delete currentNode;

        currentNode = next;
    }

    endNode->next = nullptr;
    endNode->prev = nullptr;
    delete endNode;

    tail = nullptr;
    listSize = 0;
}
```

Main

```
#include "../include/linkedLists/CircularLinkedList.hpp"
#include <cstddef>
#include <iostream>

int main() {
    CircularDoublyLinkedList list;

    while (true) {
        std::cout << "Meniu:\n";
        std::cout << "1. Adauga element\n";
        std::cout << "2. Sterge element\n";
        std::cout << "3. Afiseaza lista\n";
        std::cout << "4. Numarul de elemente cu vecini egali\n";
        std::cout << "5. Sterge lista\n";
        std::cout << "6. Iesire\n";

        short int userChoice;
        std::cout << "Optiunea aleasa: ";
        if (!(std::cin >> userChoice)) {
            std::cout << "Input invalid!\n\n\n";
            std::cin.clear();
            std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
            continue;
        }

        switch (userChoice) {
            case 1: {
                int value;
                std::size_t position;

                std::cout << "Valoarea de adaugat: ";
                std::cin >> value;

                std::cout << "Pozitia la care se adauga (0 - inceput, " <<
list.size() << " - sfarsit): ";
                std::cin >> position;

                if (list.push(value, position)) std::cout << "Element adaugat cu
succes!\n";
                else std::cout << "Elementul nu a fost putut fi adaugat!\n";
```

```
                break;
            }
            case 2: {
                std::size_t position;

                if (list.size() == 0) {
                    std::cout << "Lista este goala!\n";
                    break;
                }

                std::cout << "Pozitia de sters (0 - inceput, " << list.size() - 1
<< " - sfarsit): ";
                std::cin >> position;

                if (list.pop(position)) std::cout << "Element sters cu succes!\n";
                else std::cout << "Elementul nu a fost putut fi sters!\n";

                break;
            }
            case 3: {
                if (list.size() == 0) {
                    std::cout << "Lista este goala!\n";
                    break;
                }

                auto curentElement = list.getHead();

                std::cout << "Lista: ";
                for (size_t i = 0; i < list.size(); i++) {
                    std::cout << curentElement->data;

                    if (curentElement->next == list.getHead()) std::cout << '.';
                    else std::cout << ", ";

                    curentElement = curentElement->next;
                }
                std::cout << '\n';

                break;
            }
            case 4: {
                std::cout << "Numarul de elemente cu vecini egali: " <<
list.countElementsWithEqualNeighbors() << '\n';
                break;
            }
```

```
            case 5: {
                list.clear();
                std::cout << "Lista a fost stearsa!\n";
                break;
            }
            case 6: {
                list.clear();
                std::cout << "Program inchis!\n";
                return 0;
            }
            default: {
                std::cout << "Optiune invalida!\n";
                break;
            }
        }

        std::cout << "\n\n\n";
    }

    return 0;
}
```

Concluzie

Studiul Individual a permis aprofundarea practică a conceptelor legate de liste înlănțuite și consolidarea competențelor de proiectare a structurilor de date în C++. Implementarea a acoperit operații fundamentale (inserare la poziție arbitrară, ștergere la poziție arbitrară, golirea listei, etc.).

Testarea manuală și scenariile arătate au demonstrat stabilitatea soluției: exemple cu secvențe repetitive, liste mici (1–3 elemente) și liste mai mari au confirmat funcționarea corectă a operațiilor și recuperarea rapidă din situații de margine. Analiza codului a evidențiat că inserțiile/ștergerile la poziție arbitrară rămân O(n), în timp ce operațiile la cap/coadă pot fi O(1) — observații importante pentru dimensionarea performanței în aplicații reale.

În concluzie, lucrarea atinge obiectivele Studiului Individual: oferă o implementare funcțională și bine argumentată a unei structuri de date, evidențiază abilități de depanare și proiectare algoritmică și oferă un fundament solid pentru dezvoltări ulterioare, toate acestea, având o relevanță practică crescută în evaluarea pe termen lung a competențelor dobândite.