Міністерство освіти і науки України
Національний університет «Львівська політехніка»
Кафедра систем штучного інтелекту


# Звіт

**про виконання лабораторних та практичних робіт блоку № 6**
На тему: «Динамічні структури (Черга, Стек, Списки, Дерево). Алгоритми обробки динамічних структур.»
***з дисципліни:*** «Основи програмування»
до:
ВНС Лабораторної Роботи № 10
Алготестер Лабораторної Роботи № 5
Алготестер Лабораторної Роботи № 7-8
Практичних Робіт до блоку № 6

**Виконав:**
Студент групи ШІ-13
Кобзар Артем Сергійович

**Тема:** Динамічні структури (Черга, Стек, Списки, Дерево). Алгоритми обробки динамічних структур.
**Мета:** Ознайомитися з динамічними структурами та алгоритмами їх обробки. Реалізація у програмі мовою С++.

## Теоретичні відомості:

- лекції, практичні
- вказівки до лабораторних робіт ВНС
- https://www.programiz.com/cpp-programming
- geeksforgeeks.org
- w3schools.com/cpp

# Виконання роботи

**Завдання №1.1 - Реверс списку (Reverse list)**

**Реалізувати метод реверсу списку:** Node* reverse(Node *head);
Умови задачі:

- використовувати цілочисельні значення в списку;
- реалізувати метод реверсу;
- реалізувати допоміжний метод виведення вхідного і обернутого списків

**Завдання №1.2 – Порівняння списків bool compare (Node *h1, Node*h2);**
Умови задачі:

- використовувати цілочисельні значення в списку;
- реалізувати функцію, яка ітеративно проходиться по обох списках і порівнює дані в кожному вузлі;
- якщо виявлено невідповідність даних або якщо довжина списків різна (один список закінчується раніше іншого), функція повертає false.

**Задача №1.3 – Додавання великих чиселNode* add (Node *n1, Node *n2)**
*Умови задачі:*

- використовувати цифри від 0 до 9 для значень у списку;
- реалізувати функцію, яка обчислює суму двох чисел, які збережено в списку; молодший розряд числа записано в голові списка (напр. 379 $\Longrightarrow$ 9→7→3);
- функція повертає новий список, передані в функцію списки не модифікуються.

**Завдання №1.4 - Віддзеркалення дерева**
**TreeNode *create_mirror_flip(TreeNode *root);**
*Умови задачі:*

- використовувати цілі числа для значень у вузлах дерева
- реалізувати функцію, що проходить по всіх вузлах дерева і міняє місцями праву і ліву вітки дерева
- функція повертає нове дерево, передане в функцію дерево не модифікується

**Завдання №1.5 – Записати кожному батьківському вузлу суму підвузлів void tree_sum(TreeNode *root);**
*Умови задачі:*

- використовувати цілочисельні значення у вузлах дерева;
- реалізувати функцію, яка ітеративно проходить по бінарному дереві і записує у батьківський вузол суму значень підвузлів
- вузол-листок не змінює значення
- значення змінюються від листків до кореня дерева

**Завдання №2 VNS Lab #10 (Variant 18)**
Записи в лінійному списку містять ключове поле типу *char (рядок символів). Сформувати двонаправлений список. Знищити елемент з заданим ключем.Додати К елементів у початок списку.
**Завдання №3 Algotester Lab 5v3**
У вас є карта гори розміром N × M.
Також ви знаєте координати {x, y}, у яких знаходиться вершина гори. Ваше завдання - розмалювати карту таким чином, щоб найнижча точка мала число 0, а пік гори мав найбільше число. Клітинкі які мають суміжну сторону з вершиною мають висоту на один меншу, суміжні з ними і не розфарбовані мають ще на 1 меншу висоту і так далі.
**Завдання №4 Algotester Lab 7_8v1**
Ваше завдання - власноруч реалізувати структуру даних "Двозв’язний список".
Ви отримаєте Q запитів, кожен запит буде починатися зі слова-ідентифікатора, після якого

йдуть його аргументи.
Вам будуть поступати запити такого типу:
• Вставка:
Ідентифікатор - insert
Ви отримуєте ціле число index елемента, на місце якого робити вставку.
Після цього в наступному рядку рядку написане число N - розмір списку, який треба вставити.
У третьому рядку N цілих чисел - список, який треба вставити на позицію index.
• Видалення:
Ідентифікатор - erase
Ви отримуєте 2 цілих числа - index, індекс елемента, з якого почати видалення та n -кількість елементів, яку треба видалити.
• Визначення розміру:
Ідентифікатор - size
Ви не отримуєте аргументів.
Ви виводите кількість елементів у списку.
• Отримання значення і-го елементу
Ідентифікатор - get
Ви отримуєте ціле число - index, індекс елемента.
Ви виводите значення елемента за індексом.
• Модифікація значення і-го елементу
Ідентифікатор - set
Ви отримуєте 2 цілих числа - індекс елемента, який треба змінити, та його нове значення.
• Вивід списку на екран
Ідентифікатор - print
Ви не отримуєте аргументів.
Ви виводите усі елементи списку через пробіл.

**Завдання №5 Self-practice task (Algotester Lab 5v2)**

В пустелі існує незвичайна печера, яка є двохвимірною.
Її висота це N, ширина - M.
Всередині печери є пустота, пісок та каміння. Пустота позначається буквою , пісок S і каміння X;
Одного дня стався землетрус і весь пісок посипався вниз. Він падає на найнижчу клітинку з пустотою, але він не може пролетіти через каміння.
Ваше завдання сказати, як буде виглядати печера після землетрусу.

# Код, дизайн та оцінка часу

**Завдання №1.1-1.3**

```cpp
#include <iostream>
using namespace std;

class Node {
public:
    int value;
    Node* next;
    Node* prev;

    Node(int val) : value(val), next(nullptr), prev(nullptr) {}
};

class DoubleLinkedList {
    Node* head;
public:
    DoubleLinkedList() : head(nullptr) {}

    void addAtPosition(int value, int position) {
        if (position < 1) {
            cout << "Invalid position." << endl;
            return;
        }

        Node* newNode = new Node(value);

        if (position == 1) {
            newNode->next = head;
            if (head != nullptr) {
                head->prev = newNode;
            }
            head = newNode;
        } else {
            Node* current = head;
            for (int i = 1; i < position - 1 && current != nullptr; ++i) {
                current = current->next;
            }

            if (current == nullptr) {
                cout << "Position out of range." << endl;
                delete newNode;
            } else {
                newNode->next = current->next;
                newNode->prev = current;
```

```
            if (current->next != nullptr) {
                current->next->prev = newNode;
            }
            current->next = newNode;
        }
    }
}

void display() const {
    Node* temp = head;
    while (temp != nullptr) {
        cout << temp->value << " ";
        temp = temp->next;
    }
    cout << endl;
}

DoubleLinkedList reverse() const {
    DoubleLinkedList reversedList;
    Node* current = head;

    while (current != nullptr) {
        reversedList.addAtPosition(current->value, 1);
        current = current->next;
    }

    return reversedList;
}

bool isEqual(const DoubleLinkedList& other) const {
    Node* node1 = head;
    Node* node2 = other.head;

    while (node1 != nullptr && node2 != nullptr) {
        if (node1->value != node2->value) {
            return false;
        }
        node1 = node1->next;
        node2 = node2->next;
```

```
    while (node1 != nullptr && node2 != nullptr) {
        if (node1->value != node2->value) {
            return false;
        }
        node1 = node1->next;
        node2 = node2->next;
    }

    return node1 == nullptr && node2 == nullptr;
}

DoubleLinkedList sumWith(const DoubleLinkedList& other) const {
    DoubleLinkedList result;
    Node* ptr1 = head;
    Node* ptr2 = other.head;
    int carry = 0;

    while (ptr1 != nullptr || ptr2 != nullptr || carry != 0) {
        int total = carry;
        if (ptr1 != nullptr) {
            total += ptr1->value;
            ptr1 = ptr1->next;
        }
        if (ptr2 != nullptr) {
            total += ptr2->value;
            ptr2 = ptr2->next;
        }

        result.addAtPosition(total % 10, 1);
        carry = total / 10;
    }

    return result.reverse();
}
};
```

```
113 int main() {
114     DoubleLinkedList listA, listB;
115 
116     listA.addAtPosition(1, 1);
117     listA.addAtPosition(2, 2);
118     listA.addAtPosition(3, 3);
119     listA.addAtPosition(7, 4);
120 
121     cout << "Original list: ";
122     listA.display();
123 
124     listB = listA.reverse();
125     cout << "Reversed list: ";
126     listB.display();
127 
128     if (listA.isEqual(listB)) {
129         cout << "Lists are identical." << endl;
130     } else {
131         cout << "Lists differ." << endl;
132     }
133 
134     DoubleLinkedList listC = listA.sumWith(listA);
135     cout << "Sum of the list with itself: ";
136     listC.display();
137 
138     return 0;
139 }
```

```
Original list: 1 2 3 7
Reversed list: 7 3 2 1
Lists differ.
Sum of the list with itself: 2 4 6 4 1
```

Витрачено приблизно 3.5 години

# Завдання №1.4-1.5

```
    void update_to_sum()
    {
        update_to_sum(root);
    }

    private:
    BinarySearchTreeNode* insert(BinarySearchTreeNode* node, int value)
    {
        if (node == nullptr)
        {
            return new BinarySearchTreeNode(value);
        }
        if (value < node->val)
        {
            node->left = insert(node->left, value);
        }
        else if (value > node->val)
        {
            node->right = insert(node->right, value);
        }
        return node;
    }

    void print_in_order(BinarySearchTreeNode* node)
    {
        if (node != nullptr)
        {
            print_in_order(node->left);
            cout << node->val << " ";
            print_in_order(node->right);
        }
    }

    BinarySearchTreeNode* create_mirror(BinarySearchTreeNode* node)
    {
        if (node == nullptr)
        {
            return nullptr;
        }
        BinarySearchTreeNode* mirror_node = new BinarySearchTreeNode(node->val);
        mirror_node->left = create_mirror(node->right);
        mirror_node->right = create_mirror(node->left);
```

```
#include <iostream>
using namespace std;

class BinarySearchTreeNode
{
    public:
    int val;
    BinarySearchTreeNode* left;
    BinarySearchTreeNode* right;
    BinarySearchTreeNode(int value)
    {
        val = value;
        left = nullptr;
        right = nullptr;
    }
};

class BinarySearchTree
{
    BinarySearchTreeNode* root;
    public:
    BinarySearchTree() { root = nullptr; }

    void insert(int value)
    {
        root = insert(root, value);
    }

    void print_in_order()
    {
        print_in_order(root);
        cout << endl;
    }

    BinarySearchTree create_mirror()
    {
        BinarySearchTree mirrored_tree;
        mirrored_tree.root = create_mirror(root);
        return mirrored_tree;
    }

    void update_to_sum()
    {
```

```
        return mirror_node;
    }

    int update_to_sum(BinarySearchTreeNode* node)
    {
        if (node == nullptr)
        {
            return 0;
        }
        int left_sum = update_to_sum(node->left);
        int right_sum = update_to_sum(node->right);
        int old_value = node->val;
        node->val = left_sum + right_sum;
        return node->val + old_value;
    }
};

int main()
{
    BinarySearchTree bst;
    bst.insert(10);
    bst.insert(4);
    bst.insert(15);
    bst.insert(2);
    bst.insert(6);
    bst.insert(12);
    bst.insert(18);

    cout << "Original Tree (In-order): ";
    bst.print_in_order();

    BinarySearchTree mirrored_tree = bst.create_mirror();
    cout << "Mirrored Tree (In-order): ";
    mirrored_tree.print_in_order();

    bst.update_to_sum();
    cout << "Updated Tree to Sum (In-order): ";
    bst.print_in_order();

    return 0;
}
```

```
Original Tree (In-order): 2 4 6 10 12 15 18
Mirrored Tree (In-order): 18 15 12 10 6 4 2
Updated Tree to Sum (In-order): 0 8 0 57 0 30 0
```

Витрачено приблизно 3.5 години

# Завдання №2 VNS Lab #10 (Variant 18)

```cpp
#include <iostream>
#include <cstring>
#include <cstdarg>

using namespace std;

struct Node {
    char* key;
    Node* next;
    Node* prev;
};

void createList(Node*& head) {
    cout << "Creating list..." << endl;
    const char* elements[] = {"First", "Second", "Third", "Fourth", "Fifth", "Sixth"};

    for (int i = 0; i < 6; i++) {
        Node* newNode = new Node;
        newNode->key = strdup(elements[i]);
        newNode->next = nullptr;

        if (head == nullptr) {
            newNode->prev = nullptr;
            head = newNode;
        } else {
            Node* temp = head;
            while (temp->next != nullptr) {
                temp = temp->next;
            }
            temp->next = newNode;
            newNode->prev = temp;
        }
    }
}

void deleteByKey(Node*& head, const char* key) {
    cout << "Deleting node with key: " << key << "..." << endl;
    if (head == nullptr) return;

    Node* current = head;
    while (current != nullptr) {
        if (strcmp(current->key, key) == 0) {
            if (current->prev != nullptr) {
                if (current->prev != nullptr) {
                    current->prev->next = current->next;
                } else {
                    head = current->next;
                }
                if (current->next != nullptr) {
                    current->next->prev = current->prev;
                }
                free(current->key);
                delete current;
                return;
            }
            current = current->next;
        }
    }

    void addNodesToStart(Node*& head, int count, ...) {
        cout << "Adding " << count << " nodes to the start..." << endl;

        va_list args;
        va_start(args, count);

        for (int i = 0; i < count; i++) {
            const char* key = va_arg(args, const char*);
            Node* newNode = new Node;
            newNode->key = strdup(key);
            newNode->next = head;
            newNode->prev = nullptr;

            if (head != nullptr) {
                head->prev = newNode;
            }
            head = newNode;
        }

        va_end(args);
    }

    void printList(Node* head) {
        cout << "List contents:" << endl;
```

```
81  void printList(Node* head) {
82      cout << "List contents:" << endl;
83      if (head == nullptr) {
84          cout << "The list is empty." << endl;
85          return;
86      }
87
88      Node* current = head;
89      while (current != nullptr) {
90          cout << current->key << " ";
91          current = current->next;
92      }
93      cout << endl;
94  }
95
96  void destroyList(Node*& head) {
97      cout << "Destroying the list..." << endl;
98      while (head != nullptr) {
99          Node* temp = head->next;
100         free(head->key);
101         delete head;
102         head = temp;
103     }
104 }
105
106 int main() {
107     Node* head = nullptr;
108     const char* keyToDelete = "Fourth";
109     createList(head);
110     printList(head);
111     deleteByKey(head, keyToDelete);
112     printList(head);
113     addNodesToStart(head, 3, "New1", "New2", "New3");
114     printList(head);
115     destroyList(head);
116     printList(head);
117
118     return 0;
119 }
```

```
Creating list...
List contents:
First Second Third Fourth Fifth Sixth
Deleting node with key: Fourth...
List contents:
First Second Third Fifth Sixth
Adding 3 nodes to the start...
List contents:
New3 New2 New1 First Second Third Fifth Sixth
Destroying the list...
List contents:
The list is empty.
```

Витрачено приблизно 3 години

# Завдання №3 Algotester Lab 5v3

```cpp
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;

int main()
{
    int N, M;
    cin >> N >> M;

    int peakX, peakY;
    cin >> peakX >> peakY;


    peakX -= 1;
    peakY -= 1;

    vector<vector<int>> heightMap(N, vector<int>(M, 0));

    int maxHeight = max(peakX, N - 1 - peakX) + max(peakY, M - 1 - peakY);

    for (int row = 0; row < N; row++)
    {
        for (int col = 0; col < M; ++col)
        {
            int verticalDist = abs(peakX - row);
            int horizontalDist = abs(peakY - col);

            heightMap[row][col] = maxHeight - (verticalDist + horizontalDist);
        }
    }

    for (const auto& row : heightMap)
    {
        for (const auto& cell : row)
        {
            cout << cell << " ";
        }
        cout << endl;
    }

    return 0;
}
```

```
3 9
1 2
8 9 8 7 6 5 4 3 2
7 8 7 6 5 4 3 2 1
6 7 6 5 4 3 2 1 0
```

4 дні тому | C++ 23 | Зараховано | 0.101 | 7.500 | Перегляд

Витрачено приблизно 1 годину

# Завдання №4 Algotester Lab 7_8v1

```cpp
#include <iostream>
#include <vector>
using namespace std;

template <class T>
class Node {
public:
    T data;
    Node* next;

    Node(T value) : data(value), next(nullptr) {}
};

template <class T>
class LinkedList {
    Node<T>* head;
public:
    LinkedList() : head(nullptr) {}

    void insert(int index, const vector<T>& arr) {
        if (index == 0) {
            for (int i = arr.size() - 1; i >= 0; --i) {
                Node<T>* newNode = new Node<T>(arr[i]);
                newNode->next = head;
                head = newNode;
            }
        } else {
            Node<T>* current = head;
            for (int i = 0; i < index - 1 && current; ++i) {
                current = current->next;
            }

            for (const T& value : arr) {
                Node<T>* newNode = new Node<T>(value);
                newNode->next = current->next;
                current->next = newNode;
                current = newNode;
            }
        }
    }

    void erase(int index, int n) {
        if (index == 0) {
            for (int i = 0; i < n && head; ++i) {
                Node<T>* temp = head;
                head = head->next;
                delete temp;
            }
        } else {
            Node<T>* current = head;
            for (int i = 0; i < index - 1 && current; ++i) {
                current = current->next;
            }

            for (int i = 0; i < n && current && current->next; ++i) {
                Node<T>* temp = current->next;
                current->next = temp->next;
                delete temp;
            }
        }
    }

    int size() const {
        int count = 0;
        Node<T>* current = head;
        while (current) {
            ++count;
            current = current->next;
        }
        return count;
    }

    T get(int index) const {
        Node<T>* current = head;
        for (int i = 0; i < index && current; ++i) {
            current = current->next;
        }
        return current ? current->data : T();
    }

```

```
    void set(int index, T value) {
        Node<T>* current = head;
        for (int i = 0; i < index && current; ++i) {
            current = current->next;
        }
        if (current) current->data = value;
    }

    void print() const {
        Node<T>* current = head;
        while (current) {
            cout << current->data << " ";
            current = current->next;
        }
        cout << endl;
    }
};

int main() {
    LinkedList<int> list;
    string opt;
    int Q;
    cin >> Q;

    for (int i = 0; i < Q; ++i) {
        cin >> opt;
        if (opt == "insert") {
            int index, size;
            cin >> index >> size;
            vector<int> arr(size);
            for (int j = 0; j < size; ++j) {
                cin >> arr[j];
            }
            list.insert(index, arr);
        } else if (opt == "erase") {
            int index, n;
            cin >> index >> n;
            list.erase(index, n);
        } else if (opt == "size") {
            cout << list.size() << endl;
        } else if (opt == "get") {
            int index;
            cin >> index;
            cout << list.get(index) << endl;
        } else if (opt == "set") {
            int index, a;
            cin >> index >> a;
            list.set(index, a);
        } else if (opt == "print") {
            list.print();
        }
    }

    return 0;
}
```

```
9
insert
0
5
1 2 3 4 5

insert
2
3
7 7 7

print
1 2 7 7 7 3 4 5

erase
1 2

print
1 7 7 3 4 5

size
6

get
3
3

set
3 13

print
1 7 7 13 4 5
```

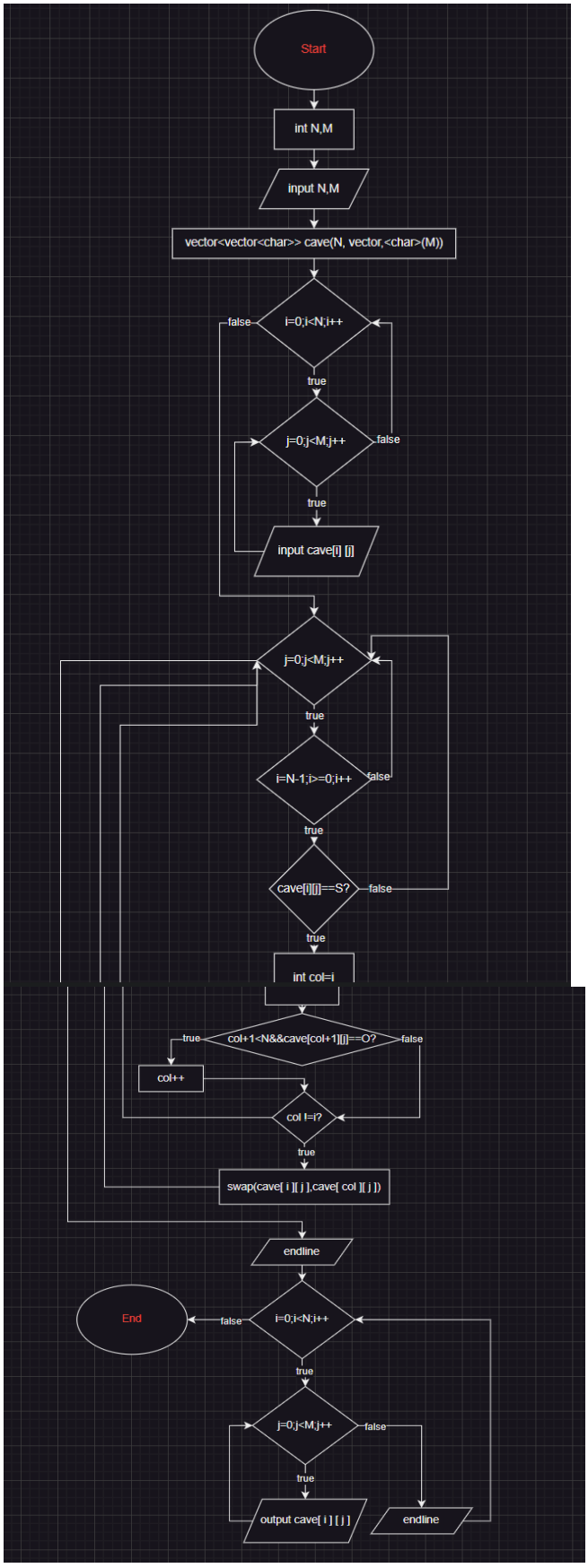
4 дні тому | C++ 23 | Зараховано | 0.008 | 1.309 | Перегляд

Витрачено приблизно 3.5 години

# Завдання №5 Self-practice task (Algotester Lab 5v2)

```cpp
#include <iostream>
#include <vector>
using namespace std;
int main()
{
    int N, M;
    cin >> N >> M;
    vector<vector<char>> cave(N, vector<char>(M));
    for (int i = 0; i < N; i++)
    {
        for (int j = 0; j < M; j++)
        {
            cin >> cave[i][j];
        }
    }
    for (int j = 0; j < M; j++)
    {
        for (int i = N - 1; i >= 0; i--)
        {
            if (cave[i][j] == 'S')
            {
                int col = i;
                while (col + 1 < N && cave[col + 1][j] == 'O')
                {
                    col++;
                }
                if (col != i)
                {
                    swap(cave[i][j], cave[col][j]);
                }
            }
        }
    }
    cout << endl;
    for (int i = 0; i < N; i++)
    {
        for (int j = 0; j < M; j++)
        {
            cout << cave[i][j];
        }
        cout << endl;
    }
}
```

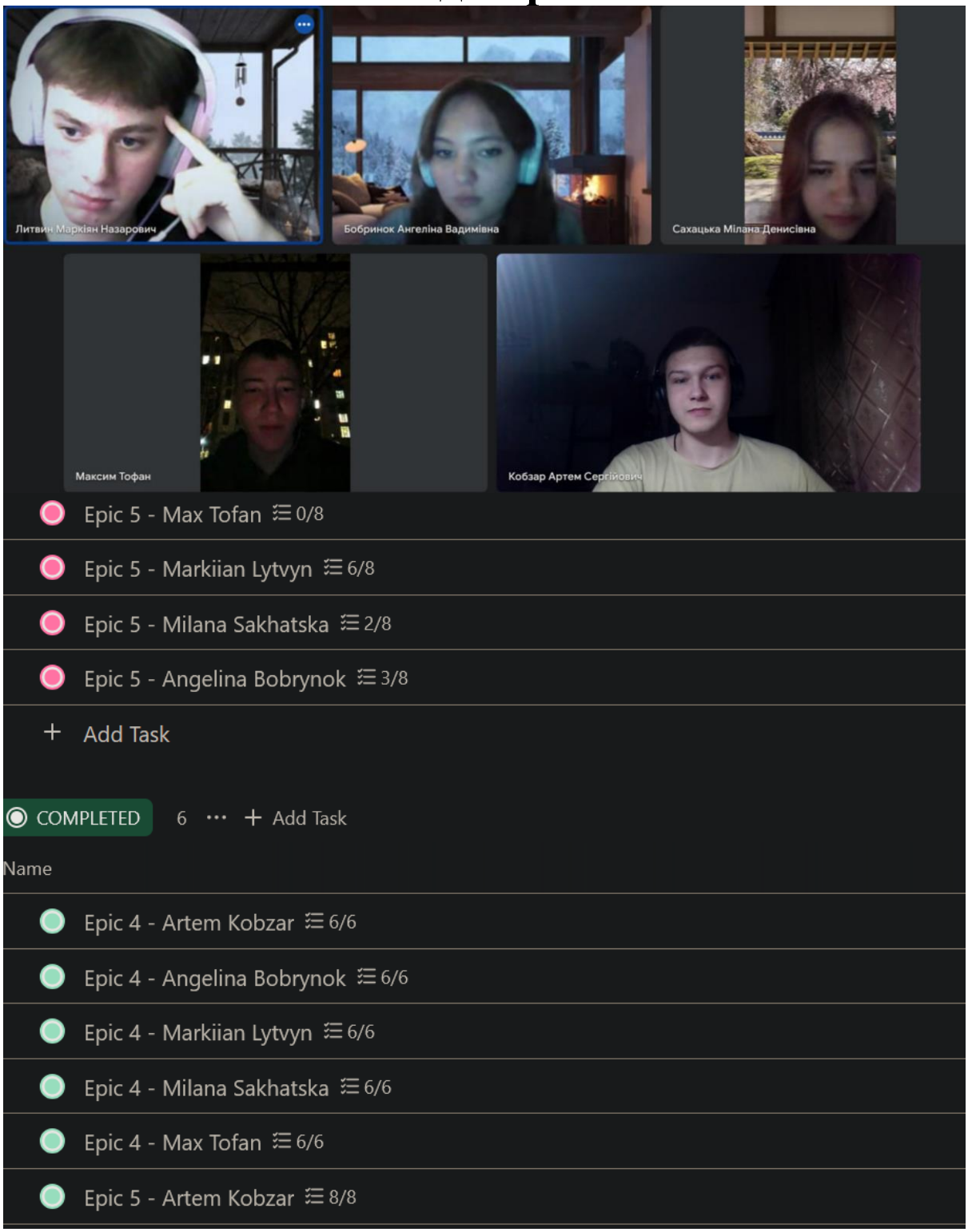
```
5 5
SSOSS
OOOOO
SOOXX
OOOOS
OOSOO

OOOOO
OOOSS
OOOXX
SOOOO
SSSOS
```





Витрачено приблизно 1 годину

# Командна робота




**Висновок:** в цьому епіку я ознайомився з динамічними структурами, а також навчився їх реалізовувати за допомогою класів у C++.