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-12
Перхун Максим Віталійович

Львів 2024

Тема роботи: Динамічні структури (Черга, Стек, Списки, Дерево). Алгоритми обробки динамічних структур.

Мета роботи: навчитись працювати з динамічними структурами, спробувати написати власні алгоритми для таких структур як: черга, стек, список та дерево.

Теоретичні відомості:

1) Структури
2) Класи
3) Список
4) Подвійний список
5) Бінарне дерево

Індивідуальний план опрацювання теорії:

- Тема №1 Структури (50 хв) (https://www.youtube.com/watch?v=999IE-6b7_s)
- Тема №2 Класи (50 хв) (https://www.youtube.com/watchv=ZbsukxxV5_Q&list=PLiPRE8VmJzOpn6PzYf0higmCEyGzo2A5g&index=95)
- Тема №3 Список (70 хв) (https://www.youtube.com/watchv=-25REjF_atI&list=PLiPRE8VmJzOpn6PzYf0higmCEyGzo2A5g&index=141)
- Тема №4 Двозв’язний список (40 хв) (https://www.youtube.com/watch?v=QLzu2_QFoE&list=PLiPRE8VmJzOpn6PzYf0higmCEyGzo2A5g)
- Тема №5 Бінарне дерево (50 хв) (https://www.youtube.com/watch?v=qBFzNW0ALxQ&list=PLiPRE8VmJzOpn6PzYf0higmCEyGzo2A5g)

Виконання роботи

Завдання №1 Epic 6 Task 3 - Lab# programming: VNS Lab 10

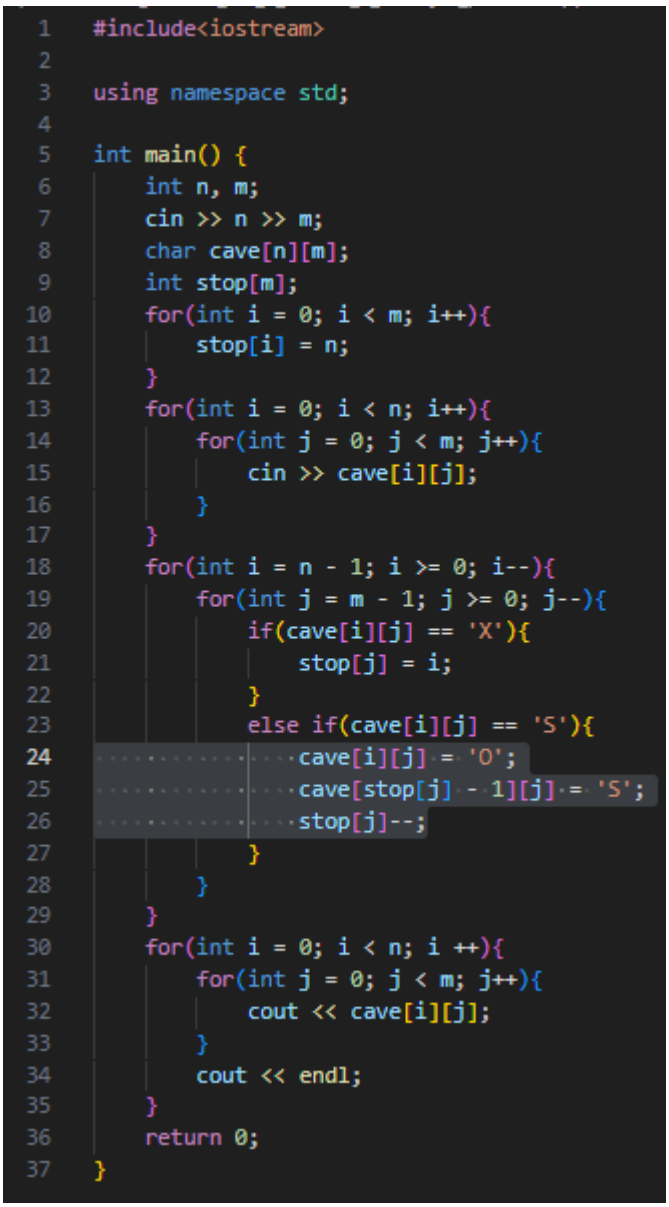
```cpp
#include<iostream>
#include <cstring>
#include <cstdio>

using namespace std;

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

Node* createNode(const char* value) {
    Node* newNode = new Node();
    newNode->text = new char[strlen(value) + 1];
    strcpy(newNode->text, value);
    newNode->next = nullptr;
    newNode->prev = nullptr;
    return newNode;
}

void deleteNode(Node* node) {
    delete[] node->text;
    delete node;
}


void append(Node*& head, Node*& tail, const char* value) {
    Node* newNode = createNode(value);
    if (!head) {
        head = tail = newNode;
    }
    else {
        tail->next = newNode;
        newNode->prev = tail;
        tail = newNode;
    }
}

void printList(Node* head){
    if (!head){
        cout << "Список порожній" << endl;
        return;
    }
    Node* current = head;
    while(current){
        cout << current -> text << " ";
        current = current -> next;
    }
    cout << endl;

}
```

```cpp
void deleteFirstK(Node*& head, Node*& tail, int k) {
    while (head && k--) {
        Node* temp = head;
        head = head->next;
        if (head) {
            head->prev = nullptr;
        }
        else {
            tail = nullptr;
        }
        deleteNode(temp);
    }
}

void addAfter(Node*& head, Node*& tail, char symbol, const char* value) {
    Node* current = head;
    while (current) {
        if (current->text[0] == symbol) {
            Node* newNode = createNode(value);
            newNode->next = current->next;
            newNode->prev = current;

            if (current->next) {
                current->next->prev = newNode;
            }
            else {
                tail = newNode;
            }
            current->next = newNode;
            return;
        }
        current = current->next;
    }
}

void saveToFile(Node* head, const char* filename) {
    FILE* fileStream = fopen(filename, "w");
    if (!fileStream) {
        cerr << "Помилка відкриття файлу для запису!" << endl;
        return;
    }
    Node* current = head;
    while (current) {
        fprintf(fileStream, "%s\n", current->text);
        current = current->next;
    }
    fclose(fileStream);
}
```

```
102
103 void restoreFromFile(Node*& head, Node*& tail, const char* filename) {
104     FILE* fileStream = fopen(filename, "r");
105     if (!fileStream) {
106         cerr << "Помилка відкриття файлу для читання!" << endl;
107         return;
108     }
109     char buffer[256];
110     while (fscanf(fileStream, "%255s", buffer) != EOF) {
111         append(head, tail, buffer);
112     }
113     fclose(fileStream);
114 }
115
116 void deleteList(Node*& head, Node*& tail) {
117     while (head) {
118         Node* temp = head;
119         head = head->next;
120         deleteNode(temp);
121     }
122     tail = nullptr;
123 }
124
```

```
126 int main() {
127     Node* head = nullptr;
128     Node* tail = nullptr;
129
130     append(head, tail, "apple");
131     append(head, tail, "banana");
132     append(head, tail, "cherry");
133     append(head, tail, "orange");
134     cout << "Початковий список: ";
135     printList(head);
136
137     int k;
138     cout << "Введіть кількість перших елементів, які треба видалити - ";
139     cin >> k;
140     deleteFirstK(head, tail, k);
141     cout << "Список після видалення перших 2 елементів: ";
142     printList(head);
143
144     char c;
145     cout << "Введіть символ з якого починається елемент після якого треба додати елемент - ";
146     cin >> c;
147     addAfter(head, tail, c, "peach");
148     cout << "Список після вставлення елементу: ";
149     printList(head);
150
151     saveToFile(head, "list.txt");
152     cout << "Список записано у файл." << endl;
153
154     deleteList(head, tail);
155     cout << "Після знищення списку: ";
156     printList(head);
157
158     restoreFromFile(head, tail, "list.txt");
159     cout << "Список після відновлення з файлу: ";
160     printList(head);
161
162     deleteList(head, tail);
163     cout << "Після остаточного знищення списку: ";
164     printList(head);
165     return 0;
166 }
```

Завдання №2 Epic 6 Task 4 - Lab# programming: Algotester Lab 5



```cpp
#include<iostream>

using namespace std;

int main() {
    int n, m;
    cin >> n >> m;
    char cave[n][m];
    int stop[m];
    for(int i = 0; i < m; i++){
        stop[i] = n;
    }
    for(int i = 0; i < n; i++){
        for(int j = 0; j < m; j++){
            cin >> cave[i][j];
        }
    }
    for(int i = n - 1; i >= 0; i--){
        for(int j = m - 1; j >= 0; j--){
            if(cave[i][j] == 'X'){
                stop[j] = i;
            }
            else if(cave[i][j] == 'S'){
                cave[i][j] = 'O';
                cave[stop[j] - 1][j] = 'S';
                stop[j]--;
            }
        }
    }
    for(int i = 0; i < n; i ++){
        for(int j = 0; j < m; j++){
            cout << cave[i][j];
        }
        cout << endl;
    }
    return 0;
}
```

Завдання №3 Epic 6 Task 5 - Lab# programming: Algotester Lab 7-8

```cpp
#include <iostream>
#include <string>
#include <algorithm>

using namespace std;

enum Operation {
    INSERT,
    SIZE,
    PRINT,
    CONTAINS,
    UNKNOWN
};

Operation getOperation(const string& command) {
    if (command == "insert") return INSERT;
    if (command == "size") return SIZE;
    if (command == "print") return PRINT;
    if (command == "contains") return CONTAINS;
    return UNKNOWN;
}

template<typename T = int>
class binaryTree {
private:
    struct Node {
        T value;
        Node* left;
        Node* right;
        Node(T val) : value(val), left(nullptr), right(nullptr) {}
    };

    Node* root;
    int treeSize;

    void destroyTree(Node* root) {
        if (root != nullptr) {
            destroyTree(root->left);
            destroyTree(root->right);
            delete root;
        }
    }

    void insertRecursively(Node* node, T value) {
        if (value < node->value) {
            if (node->left == nullptr) {
                node->left = new Node(value);
                treeSize++;
            }
            else {
                insertRecursively(node->left, value);
            }
        }
        else if (value > node->value) {
            if (node->right == nullptr) {
                node->right = new Node(value);
                treeSize++;
            } else {
                insertRecursively(node->right, value);
            }
        }
    }

    bool containsRecursively(Node* node, T value) {
        if (node == nullptr) return false;
        if (value == node->value) return true;
        if (value < node->value) return containsRecursively(node->left, value);
        return containsRecursively(node->right, value);
    }

    void printTree(Node* node, ostream& os) const {
        if (node != nullptr) {
            printTree(node->left, os);
            os << node->value << " ";
            printTree(node->right, os);
        }
    }

public:
    binaryTree() : root(nullptr), treeSize(0) {}

    ~binaryTree() {
        destroyTree(root);
    }

    void insert(T value) {
        if (root == nullptr) {
            root = new Node(value);
            treeSize++;
        } else {
            insertRecursively(root, value);
        }
    }
```

```
95      bool contains(T value) {
96          return containsRecursively(root, value);
97      }
98
99      int getSize() const {
100         return treeSize;
101     }
102
103
104     friend ostream& operator<<(ostream& os, const binaryTree& tree) {
105         tree.printTree(tree.root, os);
106         return os;
107     }
108 };
109
110 int main() {
111     int Q;
112     cin >> Q;
113     binaryTree<int> tree;
114
115     for (int i = 0; i < Q; i++) {
116         string option;
117         cin >> option;
118         Operation operation = getOperation(option);
119
120         switch (operation) {
121             case INSERT: {
122                 int value;
123                 cin >> value;
124                 tree.insert(value);
125                 break;
126             }
127             case SIZE: {
128                 cout << tree.getSize() << endl;
129                 break;
130             }
131             case CONTAINS: {
132                 int value;
133                 cin >> value;
134                 cout << (tree.contains(value) ? "Yes" : "No") << endl;
135                 break;
136             }
137             case PRINT: {
138                 cout << tree << endl;
139                 break;
140             }
141             default:
142                 break;
143         }
144     }
145 }
```

Завдання №4 Epic 6 Task 6 - Practice# programming: Class Practice Task

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

//Linked list
struct Node
{
    int data;
    Node* next;
};

//Додавання елемента в кінець
Node* push_back(Node* head, const int& element)
{
    Node* new_node = new Node;
    new_node->data = element;
    new_node->next = nullptr;

    if (head == nullptr)
    {
        head = new_node;
    }
    else
    {
        Node* current = head;
        while (current->next != nullptr)
        {
            current = current->next;
        }

        current->next = new_node;
    }

    return head;
}
```

```cpp
//Виведення списку
void print_list(Node* head)
{
    if (head == nullptr)
    {
        cout<<"List is empty"<<endl;
    }
    else
    {
        Node* current = head;
        while (current != nullptr)
        {
            cout<<current->data<<" ";
            current = current->next;
        }
        cout<<endl;
    }
}

//Реверс списку
Node* reverse_list(Node* head)
{
    if (head == nullptr)
    {
        cout<<"List is empty"<<endl;
        return 0;
    }
    else
    {
        Node* prev = nullptr;
        Node* current = head;
        Node* next = nullptr;

        while (current != nullptr)
        {
            next = current->next;
            current->next = prev;
            prev = current;
            current = next;
        }

        return prev;
    }
}
```

```cpp
//Порівняння на рівність двох списків
bool compare(Node* head_1, Node* head_2)
{
    Node* current_1 = head_1;
    Node* current_2 = head_2;

    while ((current_1 != nullptr) && (current_2 != nullptr))
    {
        if (current_1->data != current_2->data)
        {
            return false;
        }
        current_1 = current_1->next;
        current_2 = current_2->next;
    }

    if ((current_1 != nullptr) || (current_2 != nullptr))
    {
        return false;
    }
    else return true;
}

//Додавання двох великих чисел
Node* add_two_numbers(Node* n1, Node* n2)
{
    Node* current_1 = n1;
    Node* current_2 = n2;
    Node* sum = nullptr;
    int r = 0;
    int s = 0;
    while (current_2 != nullptr)
    {
        s = current_1->data + current_2->data + r;

        if (s > 9)
        {
            sum = push_back(sum, s % 10);
            r = s / 10;
        }
        else
        {
            sum = push_back(sum, s);
            r = 0;
        }

        current_1 = current_1->next;
        current_2 = current_2->next;
    }
```

```cpp
    if (current_1 != nullptr)
    {
        while (current_1 != nullptr)
        {
            s = current_1->data + r;

            if (s > 9)
            {
                sum = push_back(sum, s % 10);
                r = s / 10;
            }
            else
            {
                sum = push_back(sum, s);
                r = 0;
            }

            current_1 = current_1->next;
        }
    }
    else if (r != 0)
    {
        sum = push_back(sum, r);
    }

    return sum;
}

//Виведення числа
void print_number(Node* head)
{
    if (head == nullptr)
    {
        cout<<"List is empty"<<endl;
    }
    else
    {
        Node* current = head;
        while (current != nullptr)
        {
            cout<<current->data;
            current = current->next;
        }
        cout<<endl;
    }
}
//Linked list
```

```cpp
//Binary tree
struct tree_node
{
    int data;
    tree_node* left;
    tree_node* right;

    tree_node(int value) : data(value), left(nullptr), right(nullptr){}
};



//Функція для вставлення елемента в дерево
tree_node* insert(tree_node* node, int value)
{
    if (node == nullptr)
    {
        return new tree_node(value);
    }
    else
    {
        if (value < node->data)
        {
            node->left = insert(node->left, value);
        }
        else if (value > node->data)
        {
            node->right = insert(node->right, value);
        }

        return node;
    }

}

//Віддзеркалення дерева
tree_node* create_mirror_flip(tree_node* node)
{

    if (node == nullptr)
    {
        return nullptr;
    }

    tree_node* new_node = new tree_node(node->data);

    new_node->right = create_mirror_flip(node->left);
    new_node->left = create_mirror_flip(node->right);

    return new_node;
}
```

```cpp
//Сума підвузлів
tree_node* tree_sum (tree_node* node)
{

    if ((node == nullptr) || ((node->left == nullptr) && (node->right == nullptr))) return nullptr;

    tree_sum(node->left);
    tree_sum(node->right);

    node->data = 0;
    if (node->right != nullptr)
    {
       node->data += node->right->data;
    }
    if (node->left != nullptr)
    {
       node->data += node->left->data;
    }

    return node;
}

//Виведення дерева
void print_tree(tree_node* node)
{
    if (node == nullptr)
    {
        return;
    }

    cout<<node->data<<" ";
    print_tree(node->left);
    print_tree(node->right);

}
//Binary tree
```

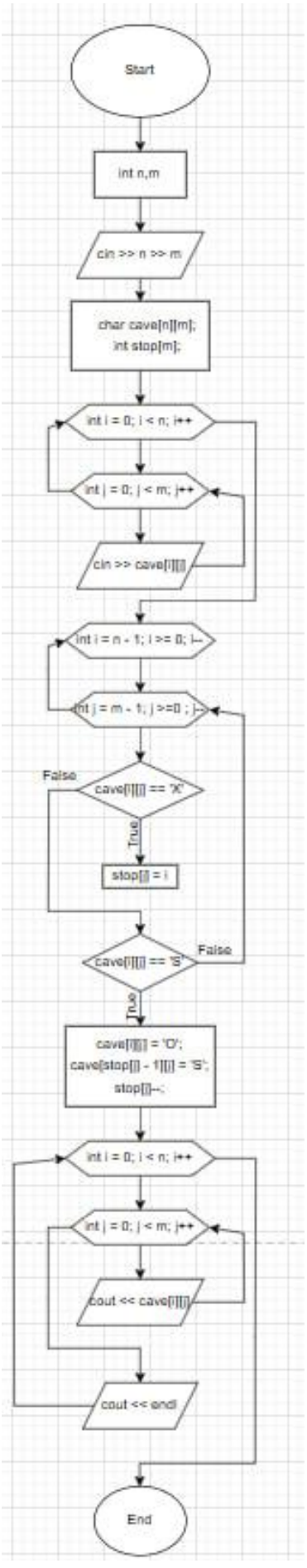
```cpp
int main()
{
    //task_1
    Node* head = nullptr;

    for (int i = 0; i < 10; i++)
    {
        head = push_back(head, i);
    }

    cout<<"Starting list: ";
    print_list(head);

    Node* new_head = reverse_list(head);
    cout<<"Reversed list: ";
    print_list(new_head);

    //task 2
    Node* head_1 = nullptr;
    Node* head_2 = nullptr;

    head_1 = push_back(head_1, 5);
    head_1 = push_back(head_1, 6);
    head_1 = push_back(head_1, 5);
    head_1 = push_back(head_1, 7);
    head_1 = push_back(head_1, 8);

    head_2 = push_back(head_2, 5);
    head_2 = push_back(head_2, 6);
    head_2 = push_back(head_2, 4);

    if (compare(head_1, head_2))
    {
        cout<<"Lists are equal"<<endl;
    }
    else cout<<"Lists aren`t equal"<<endl;

    //task 3
    string num_1, num_2, box;
    cout<<"Enter first number: ";
    cin>>num_1;
    cout<<"Enter second number: ";
    cin>>num_2;

    if (num_2.length() > num_1.length())
    {
        box = num_1;
        num_1 = num_2;
        num_2 = box;
    }
```

```cpp
    Node* n1 = nullptr;
    Node* n2 = nullptr;

    for (int i = num_1.length() - 1; i >= 0; i--)
    {
        n1 = push_back(n1, (int)num_1[i] - 48);
    }
    for (int i = num_2.length() - 1; i >= 0; i--)
    {
        n2 = push_back(n2, (int)num_2[i] - 48);
    }

    Node* sum;
    sum = add_two_numbers(n1, n2);

    Node* new_sum = reverse_list(sum);
    cout<<num_1<<" + "<<num_2<<" = ";
    print_number(new_sum);

    //task 4
    tree_node* root = nullptr;

    root = insert(root, 5);
    root = insert(root, 3);
    root = insert(root, 6);
    root = insert(root, 2);
    root = insert(root, 10);


    tree_node* new_root;
    new_root = create_mirror_flip(root);

    cout<<"First tree: ";
    print_tree(root);

    cout<<"\nMirrored tree: ";
    print_tree(new_root);

    root = tree_sum(root);
    cout<<"\nSum tree: ";
    print_tree(root);

    return 0;
}
```

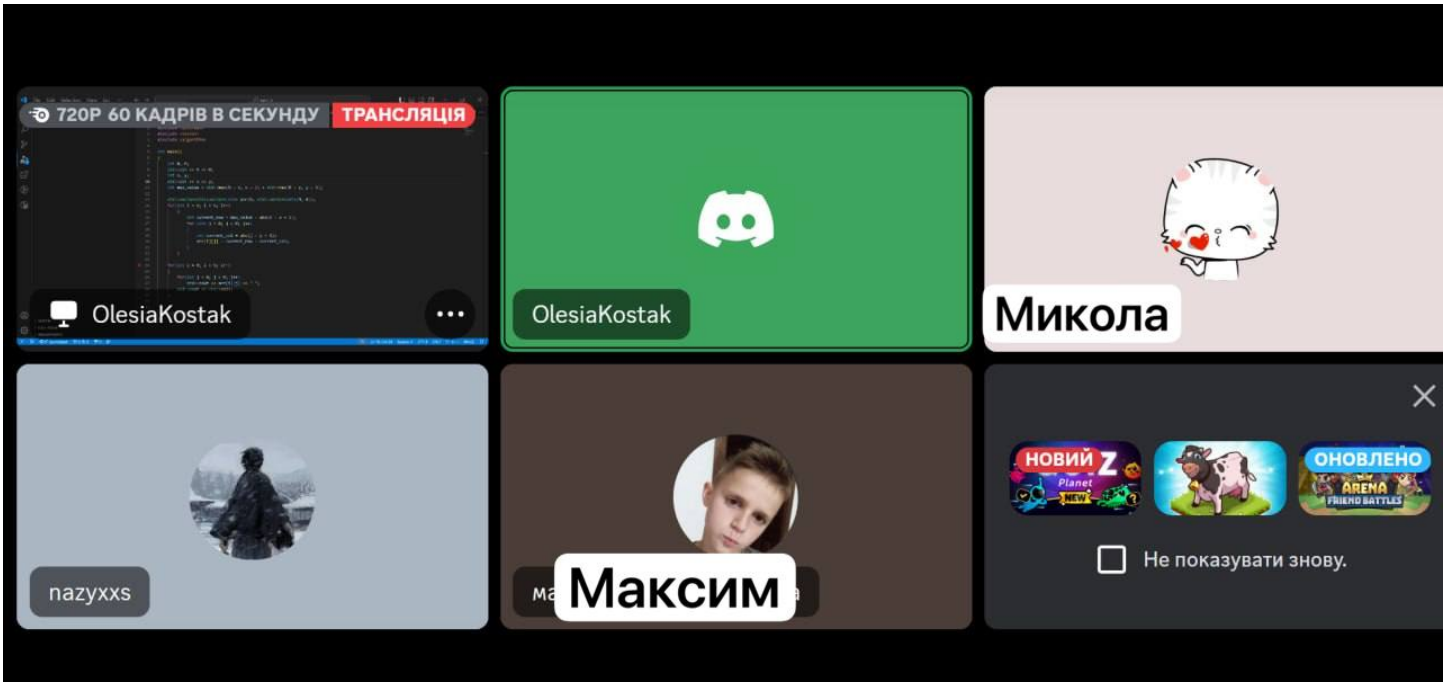
```
Starting list: 0 1 2 3 4 5 6 7 8 9
Reversed list: 9 8 7 6 5 4 3 2 1 0
Lists aren`t equal
Enter first number: 11231331301303133013
Enter second number: 13021301301321030131313313
13021301301321030131313313 + 11231331301303133013 = 13021301413634343144344446326
First tree: 5 3 2 6 10
Mirrored tree: 5 6 10 3 2
Sum tree: 12 2 2 10 10
PS C:\Users\dimat>
```

Завдання №5 Epic 6 Task 7 - Practice# programming: Self Practice Task

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


// Зрада

int main() {
    long long a, b, c, d;
    cin >> a >> b >> c >> d;

    int n;
    cin >> n;

    vector<long long> participants(n);
    for (int i = 0; i < n; ++i) {
        cin >> participants[i];
    }

    long long total_combinations = (b - a + 1) * (d - c + 1);

    for (int i = 0; i < n; ++i) {
        long long number = participants[i];
        long long winning_combinations = 0;

        for (long long x = 1; x * x <= number; ++x) {
            if (number % x == 0) {
                long long y1 = x;
                long long y2 = number / x;

                if (y1 >= a && y1 <= b && y2 >= c && y2 <= d) {
                    ++winning_combinations;
                }

                if (y1 != y2 && y2 >= a && y2 <= b && y1 >= c && y1 <= d) {
                    ++winning_combinations;
                }
            }
        }

        cout << (winning_combinations) << "/" << (total_combinations) << endl;
    }

    return 0;
}
```


Start
int n,m
cin >> n >> m
char cave[n][m];
int stop[m];
int i = 0; i < n; i++
int j = 0; j < m; j++
cin >> cave[i][j]
int i = n - 1; i >= 0; i--
int j = m - 1; j >=0 ; j--
False
cave[i][j] == 'K'
True
stop[j] = i
cave[i][j] == 'S'
False
True
cave[i][j] = 'O';
cave[stop[j] - 1][j] = 'S';
stop[j]--;
int i = 0; i < n; i++
int j = 0; j < m; j++
cout << cave[i][j]
cout << endl
End

Робота в команді



Висновок: під час виконання лабораторної роботи я навчився краще працювати і розуміти динамічні структури. Навчився писати власні алгоритми для роботи з такими структурами.