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

Виконала:
Студентка групи ШІ-12
Ляшко Леся Ігорівна

Львів 2024

Тема роботи: Динамічні структури (Черга, Стек, Списки, Дерево). Алгоритми обробки динамічних структур.
Мета роботи: Ознайомитися з такими динамічними структурами, як: черга, стек, списки, дерево. Спробувати реалізувати дані структури в задачах, зрозуміти дію. Реалізувати двозв’язний список.

Теоретичні відомості:

1) Теоретичні відомості з переліком важливих тем:

- **Тема №1: Practice# programming: Class Practice Task.**
- **Тема №2: Lab# programming:VNS Lab 10 Task .**
- **Тема №3: Lab# programming: Algotester Lab 5 Variant2.**
- **Тема №4: Lab# programming: Algotester Lab7-8 Variant1.**
- **Тема №5: Lab# programming: Algotester Lab7-8 Variant3.**
- **Тема №6: Practice# programming: Self Practice Task.**

2) Індивідуальний план опрацювання теорії:

1. Class Practice Task.

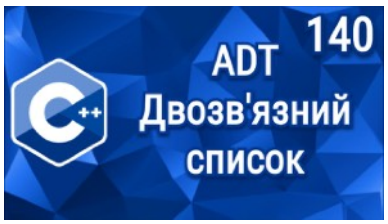
Опрацьовано та ознайомлена. 26.11.24

Linked list: https://www.w3schools.com/dsa/dsa_theory_linkedlists.php
https://www.geeksforgeeks.org/cpp-linked-list/
https://www.geeksforgeeks.org/reverse-a-linked-list/

2. VNS Lab 10 Task .

Опрацьовано та ознайомлена. 26.11.24

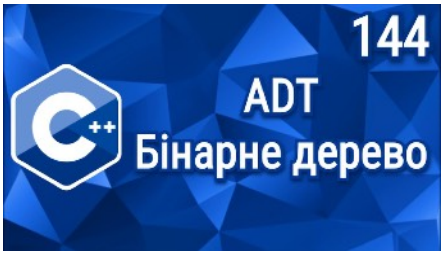
Двозв’язний список:



4. Algotester Lab7-8 .

Опрацьовано та ознайомлена. 27-28.11.24

Бінарне дерево: https://www.geeksforgeeks.org/insertion-in-a-binary-tree-in-level-order/
https://www.geeksforgeeks.org/binary-tree-in-cpp/



5. Self Practice Task.
Опрацьовано та ознайомлена. 25.11.24

Черги: https://www.bestprog.net/uk/2019/09/26/c-queue-general-concepts-ways-to-implement-the-queue-implementing-a-queue-as-a-dynamic-array-ua/
Стек і купа https://acode.com.ua/urok-111-stek-i-kupa/

Реалізація завдань

1. Class Practice Task.

Очікуваний час виконання завдання: 1 год.
Реальність: 2 год

```
#include <iostream>
struct Node {
    int data;
    Node* next;
    Node(int val) : data(val), next(nullptr) {}
};
Node* reverse(Node* head) {
    Node* prev = nullptr;
    Node* current = head;
    Node* next = nullptr;
    while (current != nullptr) {
        next = current->next;
        current->next = prev;
        prev = current;
        current = next;
    }
    return prev;
}
void printList(Node* head) {
    while (head != nullptr) {
        std::cout << head->data << " ";
        head = head->next;
    }
    std::cout << std::endl;
}
bool compare(Node* h1, Node* h2) {
    while (h1 != nullptr && h2 != nullptr) {
```

```
        if (h1->data != h2->data) {
            return false;
        }
        h1 = h1->next;
        h2 = h2->next;
    }
    return h1 == nullptr && h2 == nullptr;
}

Node* add(Node* n1, Node* n2) {
    Node dummy(0);
    Node* current = &dummy;
    int carry = 0;
    while (n1 != nullptr || n2 != nullptr || carry) {
        int sum = carry;
        if (n1 != nullptr) {
            sum += n1->data;
            n1 = n1->next;
        }
        if (n2 != nullptr) {
            sum += n2->data;
            n2 = n2->next;
        }
        carry = sum / 10;
        current->next = new Node(sum % 10);
        current = current->next;
    }
    return dummy.next;
}
struct TreeNode {
    int value;
    TreeNode* left;
    TreeNode* right;
    TreeNode(int val) : value(val), left(nullptr), right(nullptr)
{}
};
TreeNode* create_mirror_flip(TreeNode* root) {
   if (root == nullptr) return nullptr;
    TreeNode* new_root = new TreeNode(root->value);
    new_root->left = create_mirror_flip(root->right);
    new_root->right = create_mirror_flip(root->left);
    return new_root;
}
void printTree(TreeNode* root) {
    if (root != nullptr) {
        printTree(root->left);
        std::cout << root->value << " ";
        printTree(root->right);
    }
}
void tree_sum(TreeNode* root) {
    if (root == nullptr) return;
```

```
    int left_sum = 0;
    int right_sum = 0;
    if (root->left != nullptr) {
        left_sum += root->left->value;
    }
    if (root->right != nullptr) {
        right_sum += root->right->value;
    }
    tree_sum(root->left);
    tree_sum(root->right);
    if(!root->left && !root->right) return;
    root->value = left_sum + right_sum;
}
int main() {
    Node* head = new Node(1);
    head->next = new Node(2);
    head->next->next = new Node(3);
    head->next->next->next = new Node(4);
    std::cout << "Original Linked List: ";
    printList(head);
    head = reverse(head);
    std::cout << "Reversed Linked List: ";
    printList(head);
    Node* list1 = new Node(1);
    list1->next = new Node(2);
    list1->next->next = new Node(3);
    Node* list2 = new Node(1);
    list2->next = new Node(2);
    list2->next->next = new Node(3);
    std::cout << "Are lists equal? " << (compare(list1, list2) ?
"Yes" : "No") << std::endl;
    list2->next->next->data = 4;
    std::cout << "Are lists equal after modification? " <<
(compare(list1, list2) ? "Yes" : "No") << std::endl;
    Node* num1 = new Node(9);
    num1->next = new Node(9);
    num1->next->next = new Node(9);
    Node* num2 = new Node(1);
    num2->next = new Node(0);
    num2->next->next = new Node(0);
    Node* sum = add(num1, num2);
    std::cout << "Sum of numbers: ";
    printList(sum);

    TreeNode* root = new TreeNode(1);
    root->left = new TreeNode(2);
    root->right = new TreeNode(3);
    root->left->left = new TreeNode(4);
    root->left->right = new TreeNode(5);
    std::cout << "Original Tree: ";
    printTree(root);
```

```
    std::cout << std::endl;
    TreeNode* mirroredRoot = create_mirror_flip(root);
    std::cout << "Mirrored Tree: ";
    printTree(mirroredRoot);
    std::cout << std::endl;
    TreeNode* sumTreeRoot = new TreeNode(1);
    sumTreeRoot->left = new TreeNode(2);
    sumTreeRoot->right = new TreeNode(3);
    sumTreeRoot->left->left = new TreeNode(4);
    sumTreeRoot->left->right = new TreeNode(5);
    std::cout << "Tree before summing: ";
    printTree(sumTreeRoot);
    std::cout << std::endl;
    tree_sum(sumTreeRoot);
    std::cout << "Tree after summing subtrees: ";
    printTree(sumTreeRoot);
    std::cout << std::endl;
    return 0;
}
```

```
Original Linked List: 1 2 3 4
Reversed Linked List: 4 3 2 1
Are lists equal? Yes
Are lists equal after modification? No
Sum of numbers: 0 0 0 1
Original Tree: 4 2 5 1 3
Mirrored Tree: 3 1 5 2 4
Tree before summing: 4 2 5 1 3
Tree after summing subtrees: 4 9 5 5 3
llesya@MacBook-Air-liashko enik6 %
```

2. VNS Lab 10 Task .

Очікуваний час виконання завдання: 1 год.
Реальність: 2 год

```
#include <iostream>
#include <fstream>
#include <string>
using namespace std;
struct Node {
    string key;
    Node* prev;
    Node* next;
    Node(const string& value) : key(value), prev(nullptr),
next(nullptr) {}
};
class DoublyLinkedList {
```

```
private:
    Node* head;
    Node* tail;
public:
    DoublyLinkedList() : head(nullptr), tail(nullptr) {}
    void createEmptyList() {
        head = tail = nullptr;
    }
    void append(const string& key) {
        Node* newNode = new Node(key);
        if (!head) {
            head = tail = newNode;
        } else {
            tail->next = newNode;
            newNode->prev = tail;
            tail = newNode;
        }
    }
    void prepend(const string& key) {
        Node* newNode = new Node(key);
        if (!head) {
            head = tail = newNode;
        } else {
            newNode->next = head;
            head->prev = newNode;
            head = newNode;
        }
    }
    void printList() const {
        if (!head) {
            cout << "Список порожній!" << endl;
            return;
        }
        Node* current = head;
        while (current) {
            cout << current->key << " <-> ";
            current = current->next;
        }
        cout << "NULL" << endl;
    }
    void deleteByKey(const string& key) {
        if (!head) {
            cout << "Список порожній! Видалення неможливе." <<
endl;
            return;
        }
        Node* current = head;
        while (current && current->key != key) {
            current = current->next;
        }
        if (!current) {
```

```
            cout << "Елемент із ключем '" << key << "' не
знайдено." << endl;
            return;
        }
        if (current == head) {
            head = head->next;
            if (head) head->prev = nullptr;
        } else if (current == tail) {
            tail = tail->prev;
            if (tail) tail->next = nullptr;
        } else {
            current->prev->next = current->next;
            current->next->prev = current->prev;
        }
        delete current;
        cout << "Елемент із ключем '" << key << "' видалено." <<
endl;
    }
    void saveToFile(const string& filename) const {
        ofstream file(filename);
        if (!file.is_open()) {
            cout << "Помилка відкриття файлу для запису." << endl;
            return;
        }
        Node* current = head;
        while (current) {
            file << current->key << endl;
            current = current->next;
        }
        file.close();
        cout << "Список збережено у файл '" << filename << "'." <<
endl;
    }
    void restoreFromFile(const string& filename) {
        ifstream file(filename);
        if (!file.is_open()) {
            cout << "Помилка відкриття файлу для читання." <<
endl;
            return;
        }
        destroyList();
        string key;
        while (getline(file, key)) {
            append(key);
        }
        file.close();
        cout << "Список відновлено з файлу '" << filename << "'."
<< endl;
    }
    void destroyList() {
        while (head) {
            Node* temp = head;
```

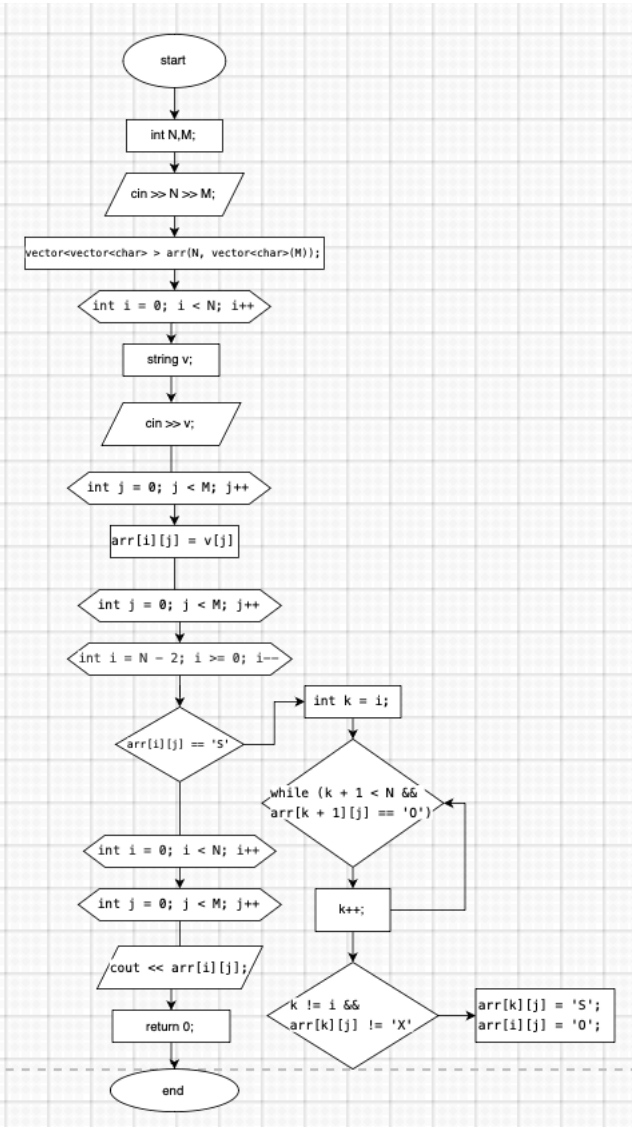
```
            head = head->next;
            delete temp;
        }
        tail = nullptr;
        cout << "Список знищено." << endl;
    }
    ~DoublyLinkedList() {
        destroyList();
    }
};
int main() {
    DoublyLinkedList list;
    list.createEmptyList();
    list.append("Елемент1");
    list.append("Елемент2");
    list.append("Елемент3");
    list.printList();
    list.prepend("НовийЕлемент");
    list.printList();
    string keyToDelete;
    cout << "Введіть ключ для видалення: ";
    cin >> keyToDelete;
    list.deleteByKey(keyToDelete);
    list.printList();
    list.saveToFile("list.txt");
    list.destroyList();
    list.printList();
    list.restoreFromFile("list.txt");
    list.printList();
    list.destroyList();
    list.printList();
    return 0;
}
```

```
Елемент1 <-> Елемент2 <-> Елемент3 <-> NULL
НовийЕлемент <-> Елемент1 <-> Елемент2 <-> Елемент3 <-> NULL
Введіть ключ для видалення: Елемент1
Елемент із ключем 'Елемент1' видалено.
НовийЕлемент <-> Елемент2 <-> Елемент3 <-> NULL
Список збережено у файл 'list.txt'.
Список знищено.
Список порожній!
Список знищено.
Список відновлено з файлу 'list.txt'.
НовийЕлемент <-> Елемент2 <-> Елемент3 <-> NULL
Список знищено.
Список порожній!
Список знищено.
llesya@MacBook-Air-liashko епік6 %
```

3. Algotester Lab 5 Variant2.

Очікуваний час виконання завдання: 1 год.
Реальність: 1.5 год

```cpp
#include <iostream>
#include <vector>
#include <string>
using namespace std;
int main() {
    int N, M;
    cin >> N >> M;
    vector<vector<char> > arr(N, vector<char>(M));
    for (int i = 0; i < N; i++) {
        string v;
        cin >> v;
        for (int j = 0; j < M; j++) {
            arr[i][j] = v[j];
        }
    }
    for (int j = 0; j < M; j++) {
        for (int i = N - 2; i >= 0;
i--) {
            if (arr[i][j] == 'S') {
                int k = i;
                while (k + 1 < N &&
arr[k + 1][j] == 'O') {
                    k++;
                }
                if (k != i &&
arr[k][j] != 'X') {
                    arr[k][j] =
'S';
                    arr[i][j] =
'O';
                }
            }
        }
    }
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < M; j++)
{
            cout << arr[i][j];
        }
        cout << endl;
    }
    return 0;
}
```



4. Algotester Lab7-8 .

Очікуваний час виконання завдання: 1 год.

Реальність: 2 год

```cpp
#include <iostream>
```

```cpp
#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>
class Tree {
private:
    struct Node {
        T value;
        Node* left;
        Node* right;
        Node(T val) : value(val), left(nullptr), right(nullptr) {}
    };
    Node* root;
    int size;
    void clear(Node* root) {
        if (root != nullptr) {
            clear(root->left);
            clear(root->right);
            delete root;
        }
    }
    void insertP(Node* node, T value) {
        if (value < node->value) {
            if (node->left == nullptr) {
                node->left = new Node(value);
                size++;
            }
            else {
                insertP(node->left, value);
            }
        }
        else if (value > node->value) {
            if (node->right == nullptr) {
                node->right = new Node(value);
                size++;
            }
            else {
                insertP(node->right, value);
            }
        }
    }
    bool containsP(Node* node, T value) {
        if (node == nullptr) return false;
        if (value == node->value) return true;
        if (value < node->value) return containsP(node->left, value);
        return containsP(node->right, value);
    }
    void printP(Node* node, ostream& os) const {
        if (node != nullptr) {
            printP(node->left, os);
            os << node->value << " ";
            printP(node->right, os);
```

```
        }
    }
public:
    Tree() : root(nullptr), size(0) {}
    ~Tree() {
        clear(root);
    }
    void insert(T value) {
        if (root == nullptr) {
            root = new Node(value);
            size++;
        }
        else {
            insertP(root, value);
        }
    }
    bool contains(T value) {
        return containsP(root, value);
    }
    int getSize() const {
        return size;
    }
    friend ostream& operator<<(ostream& os, const Tree&
tree) {
        tree.printP(tree.root, os);
        return os;
    }
};
int main() {
    int Q;
    cin >> Q;
    Tree<int> tree;
    for (int i = 0; i < Q; i++) {
        string option;
        cin >> option;
        Operation operation = getOperation(option);
        switch (operation) {
        case INSERT: {
            int value;
            cin >> value;
            tree.insert(value);
            break;
        }
        case SIZE: {
            cout << tree.getSize() << endl;
            break;
        }
        case CONTAINS: {
            int value;
            cin >> value;
            cout << (tree.contains(value) ? "Yes" : "No")
<< endl;
            break;
        }
        case PRINT: {
            cout << tree;
            break;
        }
        default:
            break;
        }
    }
}
```

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

insert
2
3
7 7 7

print

e1 2 7 7 7 3 4 5
rase
1 2

print

size

get
3

set
3 13

print
1 7 7 3 4 5
6
3
1 7 7 13 4 5
```

Очікуваний час виконання завдання: 2 год.

Реальність: 1.5 год

```
#include <iostream>
using namespace std;
template <typename T>
class DoublyLinkedList {
private:
    struct Node {
        T value;
        Node* next;
        Node* prev;
        Node(T val) : value(val), next(nullptr), prev(nullptr) {}
    };
    Node* head;
    Node* tail;
    int size;
public:
    DoublyLinkedList() : head(nullptr), tail(nullptr), size(0) {}
    ~DoublyLinkedList() { clear(); }
    void insert(int index, int n, T* values) {
        if (index < 0 || index > size) {
            cout << "Error: Index out of range" << endl;
            return;
        }
        Node* current = head;
        Node* prevNode = nullptr;
        for (int i = 0; i < index; ++i) {
            prevNode = current;
            current = current->next;
        }
        for (int i = 0; i < n; ++i) {
            Node* newNode = new Node(values[i]);
            if (!head) {
                head = tail = newNode;
            } else if (!prevNode) {
                newNode->next = head;
                head->prev = newNode;
                head = newNode;
            } else {
                newNode->next = current;
                newNode->prev = prevNode;
                if (prevNode) prevNode->next = newNode;
                if (current) current->prev = newNode;
            }
            prevNode = newNode;
            if (!current) tail = newNode;
        }
        size += n;
    }
    void erase(int index, int n) {
        if (index < 0 || index >= size || n <= 0) {
```

```
            cout << "Error: Invalid index or number of elements"
<< endl;
            return;
        }
        Node* current = head;
       for (int i = 0; i < index; ++i) {
            current = current->next;
        }
        for (int i = 0; i < n && current; ++i) {
            Node* toDelete = current;
            if (toDelete->prev) toDelete->prev->next = toDelete-
>next;
            if (toDelete->next) toDelete->next->prev = toDelete-
>prev;
            if (toDelete == head) head = toDelete->next;
            if (toDelete == tail) tail = toDelete->prev;
            current = current->next;
            delete toDelete;
            size--;
        }
    }
    int getSize() const {
        return size;
    }
    T get(int index) const {
        if (index < 0 || index >= size) {
            cout << "Error: Index out of range" << endl;
            return T();
        }
        Node* current = head;
        for (int i = 0; i < index; ++i) {
            current = current->next;
        }
        return current->value;
    }
    void set(int index, T value) {
        if (index < 0 || index >= size) {
            cout << "Error: Index out of range" << endl;
            return;
        }
        Node* current = head;
        for (int i = 0; i < index; ++i) {
            current = current->next;
        }
        current->value = value;
    }
    void print() const {
        Node* current = head;
        while (current) {
            cout << current->value << " ";
            current = current->next;
        }
```

```
        cout << endl;
    }
    void clear() {
        Node* current = head;
        while (current) {
            Node* toDelete = current;
            current = current->next;
            delete toDelete;
        }
        head = tail = nullptr;
        size = 0;
    }
    friend ostream& operator<<(ostream& os, const
DoublyLinkedList<T>& list) {
        Node* current = list.head;
        while (current) {
            os << current->value << " ";
            current = current->next;
        }
        return os;
    }
};
int main() {
    DoublyLinkedList<int> list;
    int Q;
    cin >> Q;
    for (int i = 0; i < Q; ++i) {
        string command;
        cin >> command;
        if (command == "insert") {
            int index, n;
            cin >> index >> n;
            int* elements = new int[n];
            for (int j = 0; j < n; ++j) cin >>
elements[j];
            list.insert(index, n, elements);
            delete[] elements;
        } else if (command == "erase") {
            int index, n;
            cin >> index >> n;
            list.erase(index, n);
        } else if (command == "size") {
            cout << list.getSize() << endl;
        } else if (command == "get") {
            int index;
            cin >> index;
            int value = list.get(index);
            if (index >= 0 && index < list.getSize())
{
                cout << value << endl;
            }
        } else if (command == "set") {
```

```
11

size
insert 5
insert 4
print
insert 5
print
in0
4 5
4 5
sert 1
print
contains 5
contains 0
size
1 4 5
Yes
No
3
```

```
            int index, value;
            cin >> index >> value;
            list.set(index, value);
        } else if (command == "print") {
            list.print();
        }
    }
    return 0;
}
```

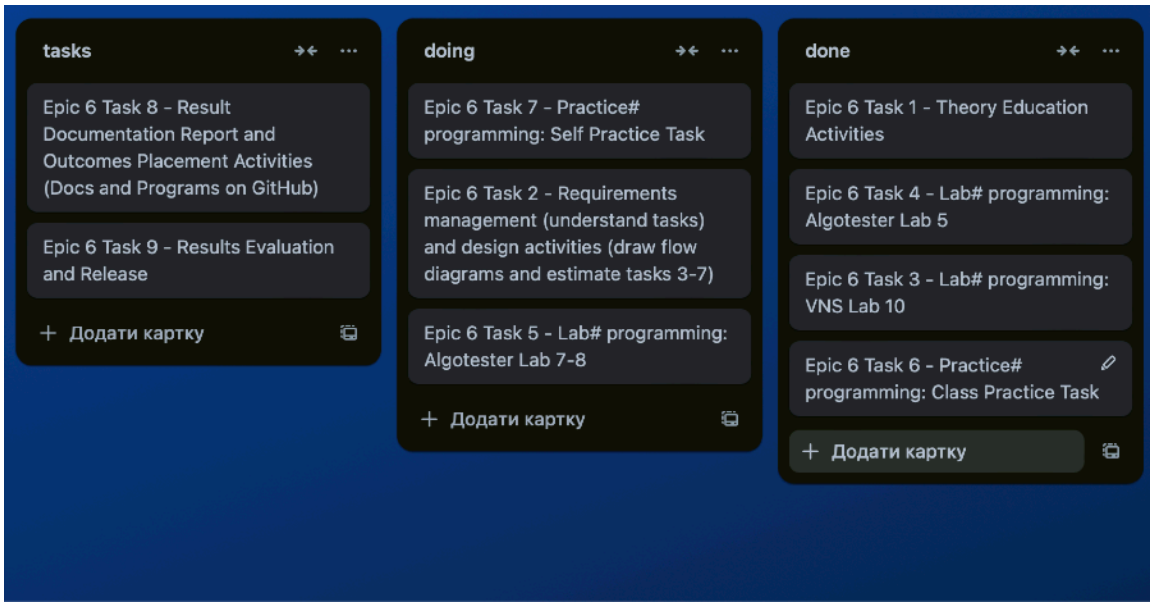
5. Self Practice Task.

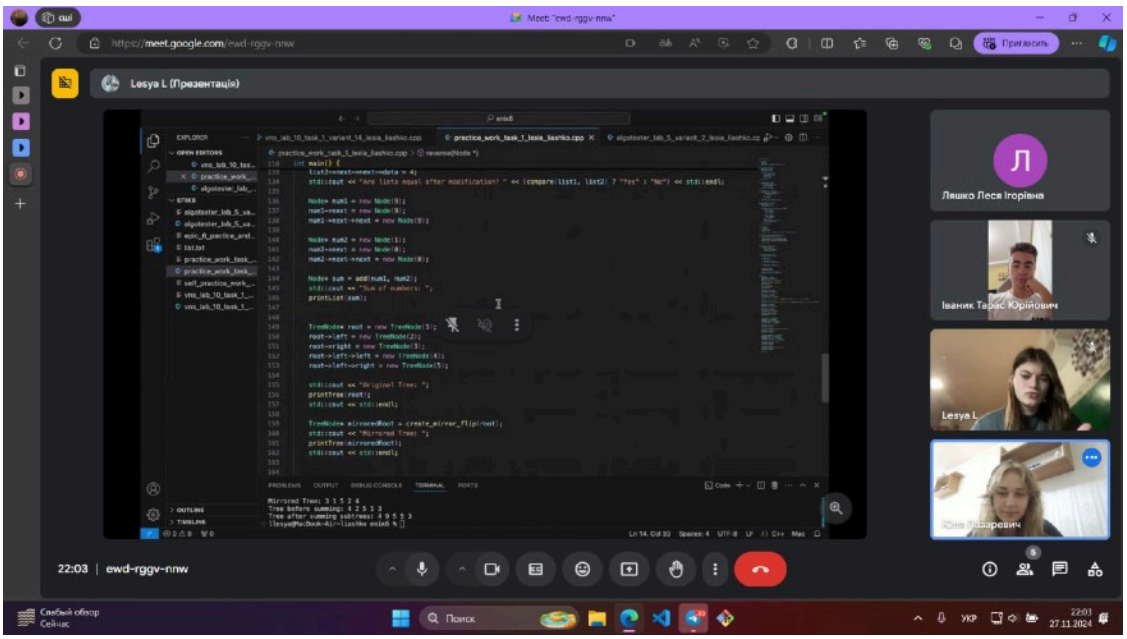
```
#include <iostream>
using namespace std;
class Stack {
private:
    struct Node {
        int value;
        Node* next;
        Node(int val) : value(val), next(nullptr) {}
    };
    Node* top;
public:
    Stack() : top(nullptr) {}
    ~Stack() {
        while (!isEmpty()) {
            pop();
        }
    }
    void push(int value) {
        Node* newNode = new Node(value);
        newNode->next = top;
        top = newNode;
    }
    int pop() {
        if (isEmpty()) {
            cout << "Stack Underflow\n";
            return -1;
        }
        Node* temp = top;
        int value = temp->value;
        top = top->next;
        delete temp;
        return value;
    }
    int peek() {
        if (isEmpty()) {
            cout << "Stack is Empty\n";
            return -1;
```

```
        }
        return top->value;
    }
    bool isEmpty() {
        return top == nullptr;
    }
    void print() {
        if (isEmpty()) {
            cout << "Stack is Empty\n";
            return;
        }
        Node* current = top;
        while (current) {
            cout << current->value << " ";
            current = current->next;
        }
        cout << endl;
    }
};
int main() {
    Stack stack;
    stack.push(10);
    stack.push(20);
    stack.push(30);
    stack.print();
    cout << "Top Element: " << stack.peek() <<
endl;
    stack.pop();
    stack.print();
    return 0;
}
```

```
30 20 10
Top Element: 30
20 10
```

Зустріч з командою.





Висновки: мені вдалося реалізувати такі структури, як списки, дерево та стек. Протягом виконання покращила свої навички в кодингу.