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

Звіт

про виконання лабораторних та практичних робіт блоку № 4

На тему: «Динамічні структури (Черга, Стек, Списки, Дерево). Алгоритми обробки динамічних структур»

до:

Практичних Робіт до блоку № 1

Виконала:
Студентка групи ШІ-12
Костак Олеся Михайлівна

Львів – 2024 р.

Тема роботи: Динамічні структури (Черга, Стек, Списки, Дерево). Алгоритми обробки динамічних структур.
Мета роботи: Освоїти роботу з динамічними структурами даних, такими як черга, стек, списки та дерева. Навчитися реалізовувати базові операції над цими структурами, включаючи створення, додавання, видалення, пошук та обхід елементів. Розвинути навички алгоритмічної обробки динамічних структур для ефективного вирішення прикладних задач.

Теоретичні відомості:

1. Теоретичні відомості з переліком важливих тем:

- Тема №*.1: Основи Динамічних Структур Даних.
- Тема №*.2: Стек.
- Тема №*.3: Черга.
- Тема №*.4: Зв'язні Списки.
- Тема №*.5: Дерева.
- Тема №*.6: Алгоритми Обробки Динамічних Структур.

2. Індивідуальний план опрацювання теорії:

Джерела Інформації:

1. Сайт geeksforgeeks
2. Сайт programiz.com
3. Сайт w3schools.com
4. Ютуб-канал Блоган
5. Лекції Пшеничного
6. Сайт cppreference.com

Тема №*.1: **Основи Динамічних Структур Даних.**

- *Що опрацьовано:*
  - Стаття Dynamic Data Structures

Тема №*.2: **Стек.**

- *Що опрацьовано:*
  - Відео C++ • Теорія • Урок 141 • ADT • Стек

Тема №*.3: **Черга.**

- *Що опрацьовано*
  - Стаття Introduction to Queue Data Structure

Тема №*.4: **Зв'язні Списки.**

- *Що опрацьовано:*
  - Відео C++ • Теорія • Урок 139 • ADT • Однозв'язний список
  - Стаття Delete a Linked List node at a given position

Тема №*.5: **Дерева.**

- *Що опрацьовано:*
  - Відео C++ • Теорія • Урок 144 • ADT • Бінарне дерево
  - Стаття Binary Tree

Тема №*.6: **Алгоритми Обробки Динамічних Структур.**

- *Що опрацьовано*:
  - Searching Algorithms

Виконання роботи:

Завдання **№1 VNS Lab 10 - Task 1**

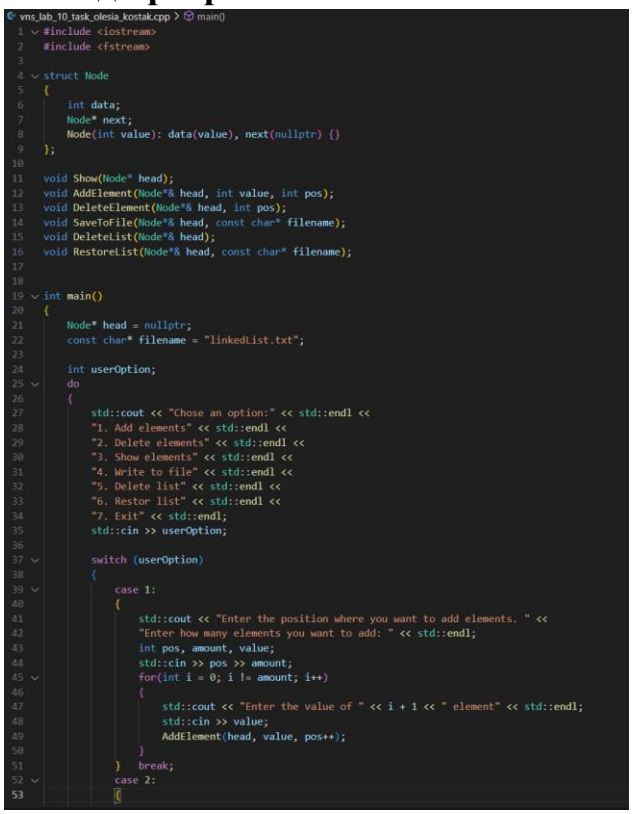
1. **Опрацювання завдання та вимог до програм та середовища:**

- *Варіант завдання*: 5
- *Деталі завдання*:

Для кожного варіанту розробити такі функції:
1. Створення списку.
2. Додавання елемента в список (у відповідності зі своїм варіантом).
3. Знищення елемента зі списку (у відповідності зі своїм варіантом).
4. Друк списку.
5. Запис списку у файл.
6. Знищення списку.
7. Відновлення списку з файлу.

Відповідно до мого варіанту:
Записи в лінійному списку містять ключове поле типу int. Сформувати однонаправлений список. Знищити з нього К елементів, починаючи із заданого номера, додати К елементів, починаючи із заданого номера;

2. **Код програми**

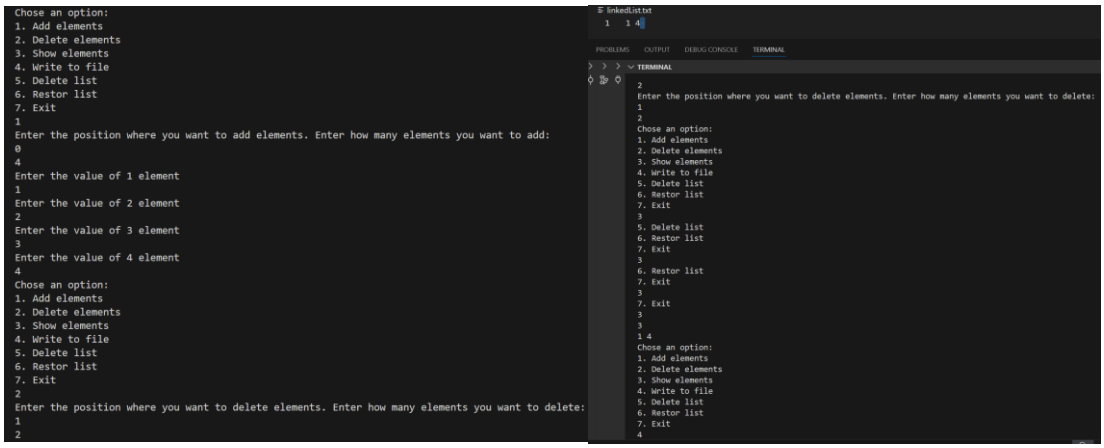


```
vns_lab_10_task_olesia_kostak.cpp > main()
#include <iostream>
#include <fstream>

struct Node
{
    int data;
    Node* next;
    Node(int value): data(value), next(nullptr) {}
};

void Show(Node* head);
void AddElement(Node*& head, int value, int pos);
void DeleteElement(Node*& head, int pos);
void SaveToFile(Node*& head, const char* filename);
void DeleteList(Node*& head);
void RestoreList(Node*& head, const char* filename);


int main()
{
    Node* head = nullptr;
    const char* filename = "linkedList.txt";

    int userOption;
    do
    {
        std::cout << "Chose an option:" << std::endl <<
        "1. Add elements" << std::endl <<
        "2. Delete elements" << std::endl <<
        "3. Show elements" << std::endl <<
        "4. Write to file" << std::endl <<
        "5. Delete list" << std::endl <<
        "6. Restor list" << std::endl <<
        "7. Exit" << std::endl;
        std::cin >> userOption;

        switch (userOption)
        {
            case 1:
            {
                std::cout << "Enter the position where you want to add elements. " <<
                "Enter how many elements you want to add: " << std::endl;
                int pos, amount, value;
                std::cin >> pos >> amount;
                for(int i = 0; i != amount; i++)
                {
                    std::cout << "Enter the value of " << i + 1 << " element" << std::endl;
                    std::cin >> value;
                    AddElement(head, value, pos++);
                }
            }  break;
            case 2:
            {
```

```
                std::cout << "Enter the position where you want to delete elements. " <<
                "Enter how many elements you want to delete: " << std::endl;
                int pos, amount;
                std::cin >> pos >> amount;
                for(int i = 0; i != amount; i++)
                    DeleteElement(head, pos);
            }   break;
            case 3:
                Show(head);
                break;
            case 4:
                SaveToFile(head, filename);
                break;
            case 5:
                DeleteList(head);
                break;
            case 6:
                RestoreList(head, filename);
                break;
            default:
                break;
        }
    }
    while (userOption != 7);

    return 0;
}

void Show(Node* head)
{
    if (head == nullptr)
        std::cout << "List is empty";
    else
    {
        Node* current = head;
        while (current != nullptr)
        {
            std::cout << current->data  << " ";
            current = current->next;
        }
    }
    std::cout << std::endl;
}

void AddElement(Node*& head, int value, int pos)···
```

```
{
    if (pos < 0)
    {
        std::cout << "Position is not correct" << std::endl;
        return;
    }

    Node* newNode = new Node(value);
    if (pos == 0)
    {
        newNode->next = head;
        head = newNode;
        return;
    }

    Node* current = head;
    for(int i = 0; i < pos - 1 && current != nullptr; i++)
        current = current->next;

    newNode->next = current->next;
    current->next = newNode;
}

void DeleteElement(Node*& head, int pos)
{
    if (pos < 0)
    {
        std::cout << "Position is not correct" << std::endl;
        return;
    }
    if (head == nullptr)
        return;
    if (pos == 0)
    {
        Node* temp = head;
        head = temp->next;
        delete temp;
        return;
    }
    Node* previous = head;
    Node* temp = head;
    for(int j = 0; j < pos; j++)
    {
        previous = temp;
        temp = temp->next;
        if (temp == nullptr)
        {
            std::cout << "Position out of range" << std::endl;
            return;
        }
    }
    previous->next = temp->next;
    delete temp;
}

void SaveToFile(Node*& head, const char* filename)
{
    std::ofstream file(filename);
    if (!file)
    {
        perror("Error opening file");
        exit(1);
    }

    Node* current = head;
    while(current != nullptr)
    {
        file << current->data << " ";
        current = current->next;
    }
    file.close();
}

void DeleteList(Node*& head)
{
    while (head != nullptr)
    {
        Node* current = head;
        head = head->next;
        delete current;
    }
}

void RestoreList(Node*& head, const char* filename)
{
    std::ifstream file(filename);
    if(!filename)
    {
        perror("Error opening file");
        exit(1);
    }
    int data;
    int pos = 0;
    while(file >> data)
    {
        AddElement(head, data, pos);
        pos++;
    }
    file.close();
}
```

Результати виконання завдань, тестування та фактично затрачений час:

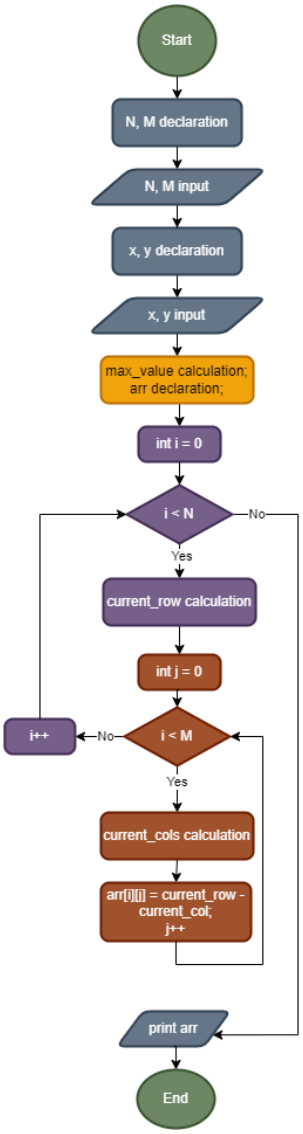


Час затрачений на виконання завдання: 4 год

Завдання **№2 Algotester Lab 5V3**

- *Варіант завдання*: 3
- *Деталі завдання*

1. Діаграма:



2. Код програми:

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

int main()
{
    int N, M;
    std::cin >> N >> M;
    int x, y;
    std::cin >> x >> y;
    int max_value = std::max(N - x, x - 1) + std::max(M - y, y - 1);

    int arr[N][M];
    for(int i = 0; i < N; i++)
        {
            int current_row = max_value - abs(i - x + 1);
            for (int j = 0; j < M; j++)
            {
                int current_col = abs(j - y + 1);
                arr[i][j] = current_row - current_col;
            }
        }

    for(int i = 0; i < N; i++)
    {
        for(int j = 0; j < M; j++)
            std::cout << arr[i][j] << " ";
        std::cout << std::endl;
    }


    return 0;
}
```

3. Результат:

Created	Compiler	Result	Time (sec.)	Memory (MiB)	Actions
17 hours ago	C++ 23	Accepted	0.100	7.383	View

Завдання **№3 Algotester Lab 7-8**

- *Варіант завдання*: 1
- *Деталі завдання*

1. Код програми:

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

template <typename T>
class DoubleLinkedList
{
private:
    struct Node
    {
        T data;
        Node* previous;
        Node* next;
        Node(T value, Node* prev = nullptr, Node* nxt = nullptr): data(value), previous(prev), next(nxt) {}
    };

    Node* head;
    Node* tail;
    int size;

public:
    DoubleLinkedList();
    ~DoubleLinkedList();
    void Insert(int index, const std::vector<T>& lst);
    void Erase(int index, int amount);
    int Size();
    T Get(int index);
    void Set(int index, T value);
    void Print();

    friend std::ostream& operator<<(std::ostream& os, const DoubleLinkedList& list)
        {
            Node* current = list.head;
            while (current != nullptr)
            {
                os << current->data << " ";
                current = current->next;
            }
            return os;
        }
};

template <typename T>
DoubleLinkedList<T>::DoubleLinkedList() : head(nullptr), tail(nullptr), size(0) {}

template <typename T>
DoubleLinkedList<T>::~DoubleLinkedList()
{
    while(head != nullptr)
    {
        Node* current = head;
        head = head->next;
        delete current;
    }
    tail = nullptr;
    size = 0;
}

template <typename T>
void DoubleLinkedList<T>::Insert(int index, const std::vector<T>& lst)
{
    if (index < 0 || index > size)
        return;

    for(int i = 0; i < lst.size(); i++)
    {
        Node* newNode = new Node(lst[i]);

        if (index == 0)
        {
            newNode->next = head;
            if(head != nullptr)
                head->previous = newNode;
            head = newNode;
            if (tail == nullptr)
                tail = newNode;
        }
        else if (index == size)
        {
            newNode->previous = tail;
            if (tail != nullptr)
                tail->next = newNode;
            tail = newNode;
        }
        else
        {
            Node* current = head;
            for(int i = 0; i < index - 1; i++)
                current = current->next;
            newNode->next = current->next;
            newNode->previous = current;
            if (current->next != nullptr)
                current->next->previous = newNode;
            current->next = newNode;
        }
        index++;
        size++;
    }

}

template <typename T>
void DoubleLinkedList<T>::Print()
{
    std::cout << *this << std::endl;
}

template <typename T>
void DoubleLinkedList<T>::Erase(int index, int amount)
{
    if (index < 0 || index > size || amount <= 0 || index + amount > size)
        return;

    Node* current = head;
    for (int i = 0; i < index; i++)
        current = current->next;

    for(int i = 0; i < amount; i++)
    {
        Node* nodeToDelete = current;
        current = current->next;

        if (nodeToDelete->previous != nullptr)
            nodeToDelete->previous->next = nodeToDelete->next;
        else
            head = nodeToDelete->next;
        if (nodeToDelete->next != nullptr)
            nodeToDelete->next->previous = nodeToDelete->previous;
        else
            tail = nodeToDelete->previous;

        delete nodeToDelete;
        size--;
    }
}

template <typename T>
int DoubleLinkedList<T>::Size()
{
    return size;
}

template <typename T>
T DoubleLinkedList<T>::Get(int index)
{
    Node* current = head;
    for(int i = 0; i < index; i++)
        current = current->next;

    return current->data;
}

template <typename T>
void DoubleLinkedList<T>::Set(int index, T value)
{
    Node* current = head;
    for(int i = 0; i < index; i++)
        current = current->next;
    current->data = value;
}

int main()
{
    DoubleLinkedList<int> myList;
    int Q;
    std::cin >> Q;
    std::string question;

    for(int i = 0; i < Q; i++)
    {
        std::cin >> question;

        if(question == "insert")
        {
            int index, N;
            std::cin >> index >> N;
            std::vector<int> lst(N);
            for(int i = 0; i < N; i++)
                std::cin >> lst[i];
            myList.Insert(index, lst);
        }
        if(question == "erase")
        {
            int index, N;
            std::cin >> index >> N;
            myList.Erase(index, N);

        }
        if(question == "size")
        {
            std::cout << myList.Size() << std::endl;
        }
        if(question == "get")
        {
            int index;
            std::cin >> index;
            std::cout << myList.Get(index) << std::endl;
        }
        if(question == "set")
        {
            int index, value;
            std::cin >> index >> value;
            myList.Set(index, value);
        }
        if(question == "print")
        {
            myList.Print();
        }

    }
    return 0;
}
```

2. Результат:

4 days ago	C++ 23	Accepted	0.008	1.344	View

Завдання **№5 Class Practice Work (1-3)**

- *Варіант завдання*: -
- *Деталі завдання*:
  1. Реалізувати метод реверсу списку: Node* reverse(Node *head);
  2. Порівняти два списки
  3. Додавання великих чисел (додати два списки)

1. Код програми:

practice_work_task_1-3_olesia_kostak.cpp > main()

```cpp
#include <iostream>

class LinkedList
{
private:
    struct Node
    {
        int data;
        Node* next;

        Node(): data(0), next(nullptr) {}
        Node(int val, Node* node = nullptr) : data(val), next(node) {}
    };
    Node* head;
    int size;

public:
    LinkedList();
    ~LinkedList();
    void PushBack(int data);
    void ShowList();
    void ReverseList();
    bool CompareTwoLists(LinkedList& list2);
    void AddElementsOfTwoLists(LinkedList& list2);
};

LinkedList::LinkedList(): head(nullptr), size(0) {}
LinkedList::~LinkedList()
{
    while(head != nullptr)
    {
        Node* current = head;
        head = head->next;
        delete current;
    }
}
void LinkedList::ShowList()
{
    Node* current = head;
    while(current != nullptr)
    {
        std::cout << current->data << " ";
        current = current->next;
    }
    std::cout << std::endl;
}
void LinkedList::PushBack(int data)
{
    Node* newNode = new Node(data);
    if(head == nullptr)
    {
        head = newNode;
        return;
    }
    Node* current = head;
    while(current->next != nullptr)
        current = current->next;
    current->next = newNode;

    size++;
}
void LinkedList::ReverseList()
{
    Node* previous = nullptr;
    Node* current = head;
    Node* nextNode;

    while(current != nullptr)
    {
        nextNode = current->next;
        current->next = previous;
        previous = current;
        current = nextNode;
    }
    head = previous;
}
bool LinkedList::CompareTwoLists(LinkedList& list2)
{
    Node* current1 = head;
    Node* current2 = list2.head;

    while(current1 != nullptr && current2 != nullptr)
    {
        if(current1->data > current2->data)
            return 0;
```

```
            return 0;
        if(current2->data > current1->data)
            return 0;
        current1 = current1->next;
        current2 = current2->next;
    }
    if((!current1) && (!current2))
        return 1;
    else
        return 0;
}
void LinkedList::AddElementsOfTwoLists(LinkedList& list2)
{
    this->ReverseList();
    list2.ReverseList();
    Node* current1 = head;
    Node* current2 = list2.head;
    long sum = 0;
    int r = 0;

    LinkedList res;
    while (current1 != nullptr || current2 != nullptr || r != 0)
    {
        sum = r;
        if(current1 != nullptr)
        {
            sum += current1->data;
            current1 = current1->next;
        }
        if(current2 != nullptr)
        {
            sum += current2->data;
            current2 = current2->next;
        }
        res.PushBack(sum % 10);
        r = sum / 10;
    }
    res.ReverseList();
    res.ShowList();
    this->ReverseList();
    list2.ReverseList();
}
int main()
{
    LinkedList list1;
    list1.PushBack(8);
    list1.PushBack(9);
    list1.PushBack(7);
    list1.PushBack(1);

    std::cout << "A list before rersing: "; list1.ShowList();
    list1.ReverseList();
    std::cout << "A list after rersing: "; list1.ShowList(); std::cout << std::endl;
    list1.ReverseList();

    LinkedList list2;
    list2.PushBack(8);
    list2.PushBack(9);
    list2.PushBack(7);

    std::cout << "Two lists to compare and add: " << std::endl;
    list1.ShowList(); list2.ShowList();
    std::cout << "Are these lists equal: ";
    std::cout << (list1.CompareTwoLists(list2)? "Yes" : "No") << std::endl << std::endl;
    std::cout << "Sum of two lists: ";
    list1.AddElementsOfTwoLists(list2);

    return 0;
}
```

2. Результат:

```
A list before rersing: 8 9 7 1
A list after rersing: 1 7 9 8

Two lists to compare and add:
8 9 7 1
8 9 7
Are these lists equal: No

Sum of two lists: 9 8 6 8
```

Завдання **№5 Class Practice Work (4-5)**

- *Варіант завдання*: -
- *Деталі завдання*:
  1. Віддзеркалити дерево.
  2. Записати кожному батьківському вузлу суму підвузлів.

1. Код програми:

practice_work_task_4-5_olesia_kostak.cpp > ...

```cpp
#include <iostream>

struct Node
{
    int data;
    Node* left;
    Node* right;
    Node(int value): data(value), left(nullptr), right(nullptr) {}
};

void Insert(Node*& root, int value);
void Show(Node* root);
void FreeTree(Node*& root);
Node* MirrorTree(Node*& root);
void TreeSum(Node*& root);


int main()
{
    Node* myTree = nullptr;
    Insert(myTree, 50);
    Insert(myTree, 40);
    Insert(myTree, 60);
    Insert(myTree, 30);
    Insert(myTree, 70);
    Insert(myTree, 80);
    Insert(myTree, 20);

    std::cout << "Tree: ";
    Show(myTree);
    std::cout << std::endl;
    std::cout << "Mirrored Tree: ";
    MirrorTree(myTree);
    Show(myTree);
    MirrorTree(myTree);
    std::cout << std::endl << "Tree of sum: ";
    TreeSum(myTree);
    Show(myTree);
    FreeTree(myTree);

    return 0;
}

void Insert(Node*& root, int value)
{
    if (root == nullptr)
    {
        root = new Node(value);
        return;
    }
    if (root->data == value)
        return;
    if (root->data > value)
    {
        if (root->left == nullptr)
        {
            root->left = new Node(value);
            return;
        }
        Insert(root->left, value);
    }
    if(root->data < value)
    {
        if (root->right == nullptr)
        {
            root->right = new Node(value);
            return;
        }
        Insert(root->right, value);
    }
}

void Show(Node* root)
{
    if(root != nullptr)
    {
        Show(root->left);
        std::cout << root->data << " ";
        Show(root->right);
    }
}

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

Node* MirrorTree(Node*& root)
{
    if (root == nullptr)
        return nullptr;
    Node* left = MirrorTree(root->left);
    Node* right = MirrorTree(root->right);

    root->right = left;
    root->left = right;
    return root;
}

void TreeSum(Node*& root)
{
    if (root == nullptr)
        return;

    TreeSum(root->left);
    TreeSum(root->right);

    if (root->left != nullptr)
        root->data += root->left->data;

    if (root->right != nullptr)
        root->data += root->right->data;
}
```

2. Результат:

```
Tree: 20 30 40 50 60 70 80
Mirrored Tree: 80 70 60 50 40 30 20
Tree of sum: 20 50 90 350 210 150 80
```

Завдання **№6 Self Practice Work (**[Lab 78v3](#)**)**

1. Код програми:

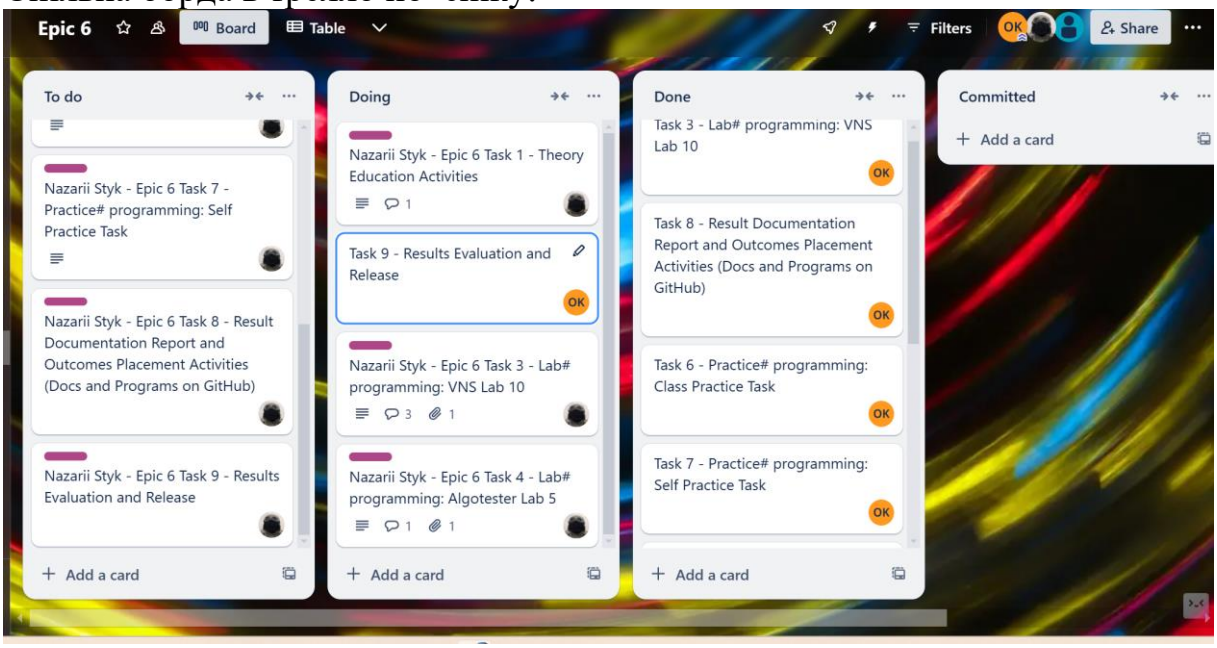
```cpp
#include <iostream>

template <typename T>
class BinaryTree
{
private:
    struct TreeNode
    {
        T data;
        TreeNode* left;
        TreeNode* right;
        TreeNode(T value): data(value), left(nullptr), right(nullptr) {};
    };
    TreeNode* root;
    int size;
    void Insert(TreeNode*& root, T value);
    void Print(const TreeNode* root) const;

public:
    BinaryTree();
    ~BinaryTree();
    void Insert(T value);
    bool Contains(T value) const;
    int Size() const;
    void Print() const;
    void FreeTree(TreeNode* root);
};

template <typename T>
BinaryTree<T>::BinaryTree(): root(nullptr), size(0) {}

template <typename T>
BinaryTree<T>::~BinaryTree()
{
    FreeTree(root);
    size = 0;
}

template <typename T>
void BinaryTree<T>::FreeTree(TreeNode* root)
{
    if (root != nullptr)
    {
        FreeTree(root->left);
        FreeTree(root->right);
        delete root;
        size = 0;
    }
}

template <typename T>
void BinaryTree<T>::Insert(TreeNode*& root, T value)
{
    if (root == nullptr)
    {
        root = new TreeNode(value);
        size++;
        return;
    }
    if (root->data == value)
        return;
    if (value < root->data)
    {
        if (root->left == nullptr)
        {
            root->left = new TreeNode(value);
            size++;
            return;
        }
        Insert(root->left, value);
    }
    if (value > root->data)
    {
        if (root->right == nullptr)
        {
            root->right = new TreeNode(value);
            size++;
            return;
        }
        Insert(root->right, value);
    }
}
template <typename T>
void BinaryTree<T>::Insert(T value)
{
    Insert(root, value);
}

template <typename T>
void BinaryTree<T>::Print(const TreeNode* root) const
{
    if (root != nullptr)
    {
        Print(root->left);
        std::cout << root->data << " ";
        Print(root->right);
    }
}
template <typename T>
void BinaryTree<T>::Print() const
{
    Print(root);
}

template <typename T>
bool BinaryTree<T>::Contains(T target) const
{
    TreeNode* current = root;
    while (current != nullptr)
    {
        if (current->data == target)
            return true;
        if (target > current->data)
            current = current->right;
        else
            current = current->left;
    }
    return false;
}

template <typename T>
int BinaryTree<T>::Size() const
{
    return size;
}

int main()
{
    int Q;
    std::cin >> Q;
    std::string question;
    BinaryTree<int> myTree;

    for(int i = 0; i < Q; i++)
    {
        std::cin >> question;
        if (question == "insert")
        {
            int value;
            std::cin >> value;
            myTree.Insert(value);
        }
        if (question == "contains")
        {
            int target;
            std::cin >> target;
            std::cout << (myTree.Contains(target) ? "Yes" : "No") << std::endl;
        }
        if (question == "size")
        {
            std::cout << myTree.Size() << std::endl;
        }
        if (question == "print")
        {
            myTree.Print();
            std::cout << std::endl;
        }
    }
    return 0;
}
```

2. Результат:

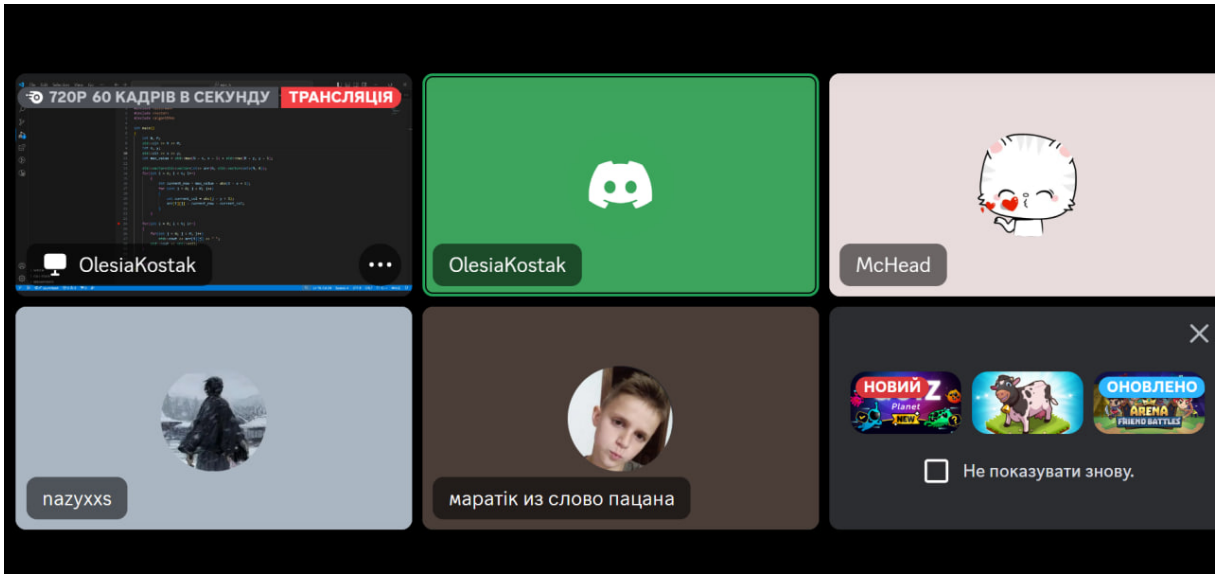
3 days ago	C++ 23	Accepted	0.008	1.422	View

Кооперація з командою:

Спільна борда в трелло по епіку:



Балакаємо про епіки:



Висновки:

По завершенню завдань Епіка №6, я освоїла роботу з динамічними структурами даних, такими як черга, стек, списки та дерева. Навчилася реалізовувати базові операції над цими структурами, включаючи створення, додавання, видалення, пошук та обхід елементів. Розвинула навички алгоритмічної обробки динамічних структур для ефективного вирішення прикладних задач.