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 Лабораторних Робіт № 5, 7-8

Практичних Робіт до блоку № 6

Виконала:

Студентка групи ШІ-12

Смачило Іванна

Львів – 2024

Тема роботи

Динамічні структури, види динамічних структур, їх використання, алгоритми їх обробки.

Мета роботи

1. Навчитись створювати та використовувати динамічні структури, такі як: Черга, Стек, Списки, Дерево.

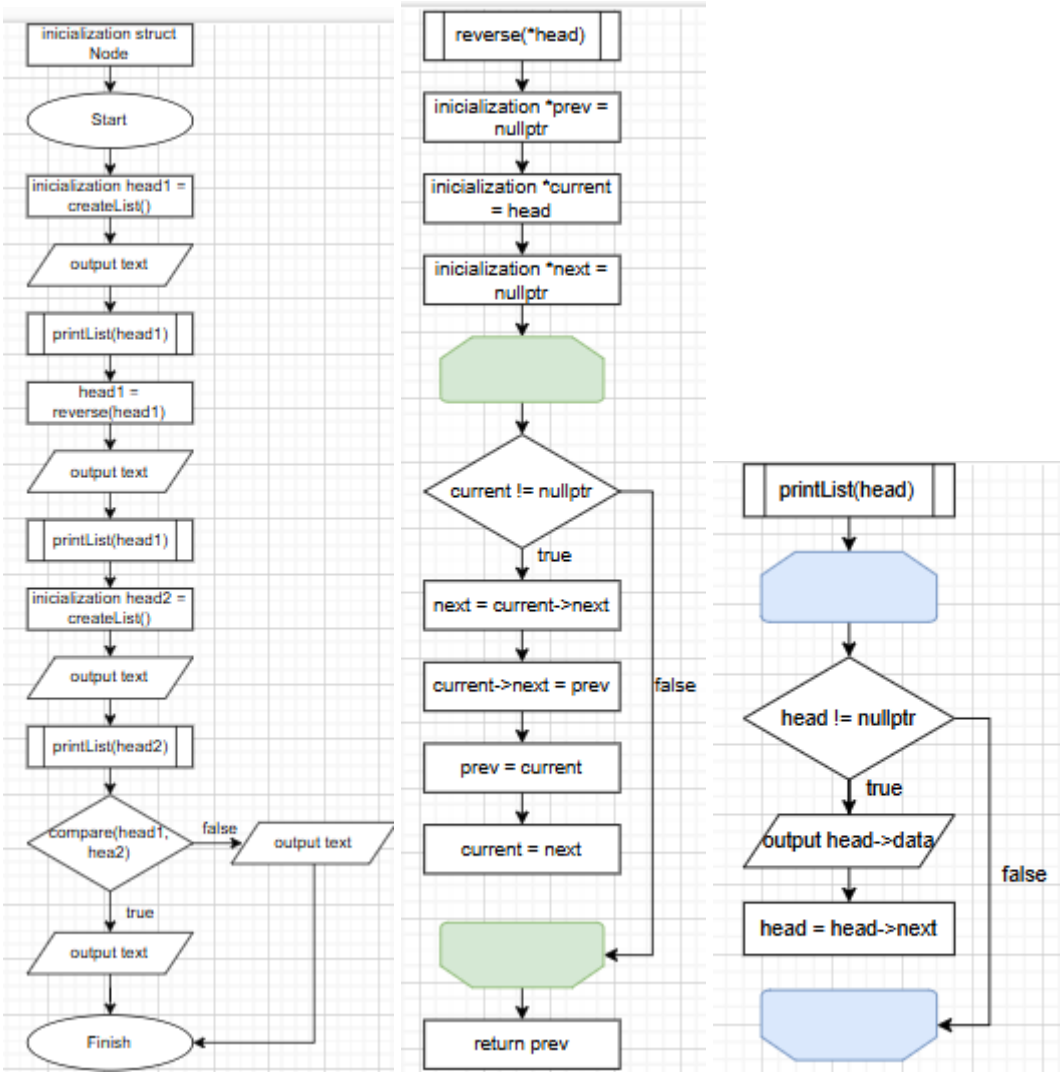
2. Навчитись виконувати алгоритми обробки динамічний структур.

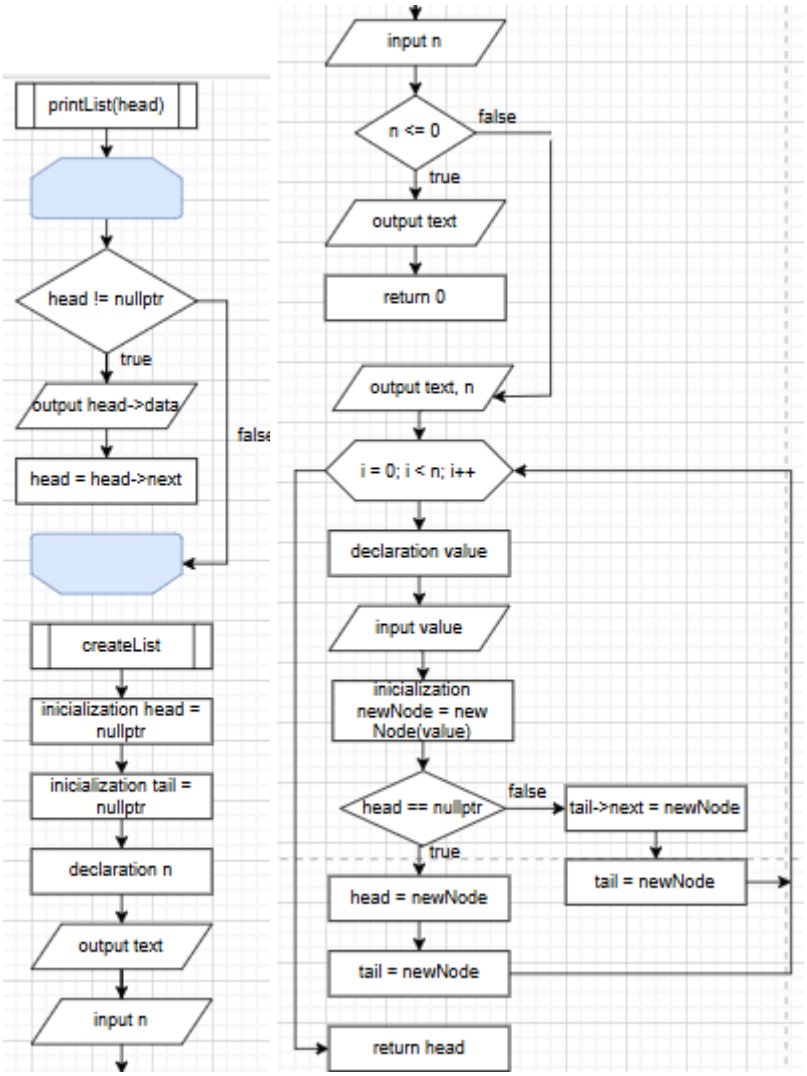
Теоретичні відомості

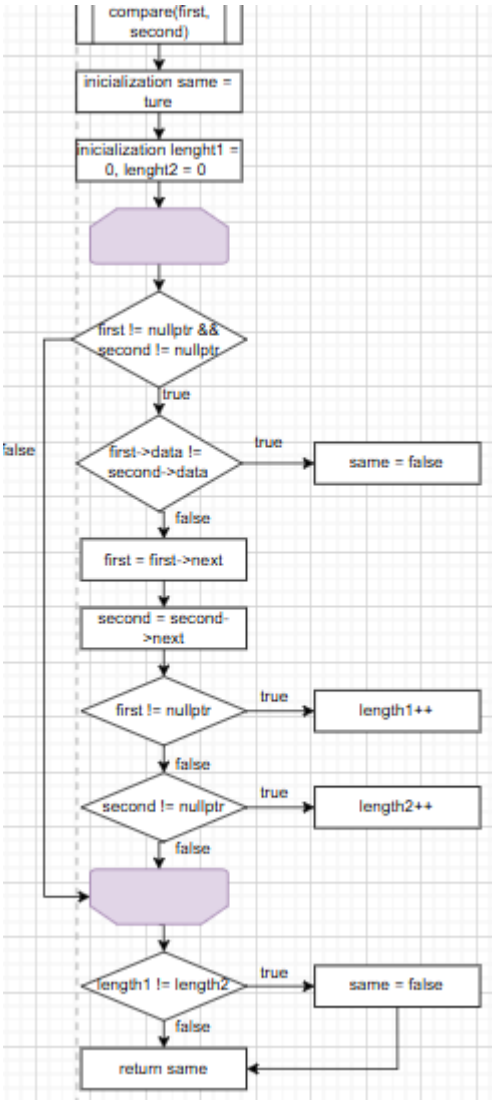
1. Стек і купа:
https://www.bestprog.net/uk/2019/09/18/c-the-concept-of-stack-operations-on-the-stack-an-example-implementation-of-the-stack-as-a-dynamic-array-ua/#google_vignette
https://acode.com.ua/urok-111-stek-i-kupa/
https://dystosvita.org.ua/mod/page/view.php?id=888
2. Динамічні масиви: https://acode.com.ua/urok-90-dynamichni-masyvy/
3. Черга: https://itproger.com/ua/spravka/cpp/queue
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/
4. Зв’язні списки: https://prometheus.org.ua/cs50/sections/section6.html
5. Список list: https://codelessons.dev/ru/spisok-list-v-s-polnyj-material/
6. Circular linked list: https://www.geeksforgeeks.org/circular-linked-list-in-cpp/
7. Бінарні дерева: https://purecodecpp.com/uk/archives/2483#google_vignette
8. Перевантаження операторів вводу і виводу:
https://acode.com.ua/urok-141-perevantazhennya-operatoriv-vvodu-i-vyvodu/#toc-0
9. Reverse in C++: https://www.geeksforgeeks.org/stdreverse-in-c/
10. Compare in C++: https://www.geeksforgeeks.org/stdstringcompare-in-c/
11. C++ Program For Inserting A Node In A Linked List:
https://www.geeksforgeeks.org/cpp-program-for-inserting-a-node-in-a-linked-list/

Виконання роботи

Task 2 - Requirements management (understand tasks) and design activities (draw flow diagram for Class Practice Task)


inicialization struct Node
Start
inicialization head1 = createList()
output text
printList(head1)
head1 = reverse(head1)
output text
printList(head1)
inicialization head2 = createList()
output text
printList(head2)
compare(head1, hea2)
false
output text
true
output text
Finish
reverse(*head)
inicialization *prev = nullptr
inicialization *current = head
inicialization *next = nullptr
current != nullptr
true
false
next = current->next
current->next = prev
prev = current
current = next
return prev
printList(head)
head != nullptr
true
false
output head->data
head = head->next


printList(head)
head != nullptr
true
output head->data
head = head->next
createList
inicialization head = nullptr
inicialization tail = nullptr
declaration n
output text
input n
input n
n <= 0
false
true
output text
return 0
output text, n
i = 0; i < n; i++
declaration value
input value
inicialization newNode = new Node(value)
head == nullptr
false
tail->next = newNode
true
head = newNode
tail = newNode
tail = newNode
return head



Task 3 - Lab# programming: VNS Lab 10

Завдання: 17.Записи в лінійному списку містять ключове поле типу *char (рядок символів). Сформувати двонаправлений список. Знищити елемент із заданим номером. Додати К елементів у початок списку.

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

using namespace std;

class Node
{
public:
    string data;
    Node* next;
    Node* prev;

    Node(string data)
    {
        this->data = data;
        this->next = nullptr;
        this->prev = nullptr;
    }
};

class DoublyLinkedList
{
public:
    Node* head;
    Node* tail;

    DoublyLinkedList()
    {
        head = tail = nullptr;
    }

    void atBeginning(string data)
    {
        Node* newNode = new Node(data);
        if (head == nullptr)
        {
            head = tail = newNode;
        }
        else
        {
            newNode->next = head;
            head->prev = newNode;
            head = newNode;
        }
    }

    void atEnd(string data)
    {
        Node* newNode = new Node(data);
        if (head == nullptr)
        {
            head = tail = newNode;
        }
        else
        {
            tail->next = newNode;
            newNode->prev = tail;
            tail = newNode;
        }
    }

    void deleteNode(Node* node)
    {
        if (node == nullptr)
        {
            return;
        }

        if (node == head)
        {
            head = node->next;
        }
        else
        {
            node->prev->next = node->next;
        }

        if (node == tail)
        {
            tail = node->prev;
        }
        else
        {
            node->next->prev = node->prev;
        }

        delete node;
    }

    void deleteNode(string data)
    {
        Node* current = head;
        while (current != nullptr)
        {
            if (current->data == data)
            {
                deleteNode(current);
                return;
            }
            current = current->next;
        }
        cout << "Node not found!" << "\n";
    }

    void printList()
    {
        Node* current = head;
        while (current != nullptr)
        {
            cout << current->data << " ";
            current = current->next;
        }
        cout << "\n";
    }

    void writeToFile(const string& lab10)
    {
        ofstream file(lab10);
        Node* current = head;

        while (current != nullptr)
        {
            file << current->data << "\n";
            current = current->next;
        }
        file.close();
    }

    void readFromFile(const string& lab10)
    {
        ifstream file(lab10);
        string data;

        while (getline(file, data))
        {
            atEnd(data);
        }
        file.close();
    }
};

int main()
{
    DoublyLinkedList list;

    list.atEnd("A");
    list.atEnd("B");
    list.atBeginning("X");
    list.atEnd("C");

    list.printList();

    list.deleteNode("B");

    list.printList();

    list.writeToFile("lab10.txt");

    Node* current = list.head;
    while (current != nullptr)
    {
        Node* temp = current;
        current = current->next;
        delete temp;
    }
    list.head = list.tail = nullptr;

    list.readFromFile("lab10.txt");

    list.printList();

    return 0;
}
```

Результат:

```
X A B C
X A C
X A C
```

C: > Epic_6 > lab10.txt

```
1  X
2  A
3  C
4
```

Task 4 - Lab# programming: Algotester Lab 5

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

using namespace std;

const int MAX_N = 1000;
const int MAX_M = 1000;

int main()
{
    int N, M;
    char cave[MAX_N][MAX_M + 1];

    cin >> N >> M;

    for (int i = 0; i < N; i++)
    {
        cin >> cave[i];
    }

    for (int c = 0; c < M; c++)
    {
        int empty_row = N - 1;

        for (int r = N - 1; r >= 0; r--)
        {
            if (cave[r][c] == 'X')
            {
                empty_row = r - 1;
            }
            else if (cave[r][c] == 'S')
            {
                else if (cave[r][c] == 'S')
                {
                    cave[r][c] = '0';
                    if (empty_row >= 0)
                    {
                        cave[empty_row][c] = 'S';
                        empty_row--;
                    }
                }
            }
        }
        cout << "\n";
        for (int i = 0; i < N; i++)
        {
            cout << cave[i] << "\n";
        }

        return 0;
    }
```

Результат:

```
5 5
SS0SS
00000
S0XX
0000S
00S00

00000
000SS
00XX
S0000
SSS0S
```

Created	Compiler	Result	Time (sec.)	Memory (MiB)	Actions
34 minutes ago	C++ 23	Accepted	0.021	1.879	View

Task 5 - Lab# programming: Algotester Lab 7-8

Variant 3:

```cpp
#include <iostream>

using namespace std;

struct Node
{
    int value;
    Node* left;
    Node* right;

    Node(int val) : value(val), left(nullptr), right(nullptr) {}
};

class BinarySearchTree
{
public:
    Node* root;

    BinarySearchTree() : root(nullptr) {}

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

    bool contains(int value)
    {
        return search(root, value);
    }

    int size()
    {
        return sizeRecursive(root);
    }

    void print()
    {
        printInOrder(root);
        cout << "\n";
    }
```

```cpp
private:
    Node* insertRecursive(Node* node, int value)
    {
        if (node == nullptr)
        {
            return new Node(value);
        }

        if (value < node->value)
        {
            node->left = insertRecursive(node->left, value);
        }
        else if (value > node->value)
        {
            node->right = insertRecursive(node->right, value);
        }

        return node;
    }

    bool search(Node* node, int value)
    {
        if (node == nullptr)
        {
            return false;
        }

        if (value == node->value)
        {
            return true;
        }

        if (value < node->value)
        {
            return search(node->left, value);
        }
        else
        {
            return search(node->right, value);
```

```cpp
        }
    }

    int sizeRecursive(Node* node)
    {
        if (node == nullptr)
        {
            return 0;
        }

        return 1 + sizeRecursive(node->left) + sizeRecursive(node->right);
    }

    void printInOrder(Node* node)
    {
        if (node == nullptr)
        {
            return;
        }

        printInOrder(node->left);
        cout << node->value << " ";
        printInOrder(node->right);
    }
};

int main() {
    int Q;
    cin >> Q;

    BinarySearchTree tree;

    while (Q--)
    {
        string command;
        cin >> command;
```

```cpp

        if (command == "insert")
        {
            int value;
            cin >> value;
            tree.insert(value);
        }
        else if (command == "contains")
        {
            int value;
            cin >> value;
            cout << (tree.contains(value) ? "Yes" : "No") << "\n";
        }
        else if (command == "size")
        {
            cout << tree.size() << "\n";
        }
        else if (command == "print")
        {
            tree.print();
        }
    }

    return 0;
}
```

Результат:

```
11

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

Created	Compiler	Result	Time (sec.)	Memory (MiB)	Actions
2 days ago	C++ 23	Accepted	0.008	1.285	View

Variant 1:

```cpp
#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) 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) 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) throw out_of_range("Index out of range");

        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) throw out_of_range("Index out of range");

        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 << "\n";
    }

    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;

    while (Q--) {
        string command;
        cin >> command;

        if (command == "insert")
        {
            int index, n;
            cin >> index >> n;

            int* values = new int[n];
            for (int i = 0; i < n; i++)
            {
                cin >> values[i];
            }

            list.insert(index, n, values);
            delete[] values;
        }
        else if (command == "erase")
        {
            int index, n;
            cin >> index >> n;
            list.erase(index, n);
        }
```

```
        else if (command == "size")
        {
            cout << list.getSize() << "\n";
        }
        else if (command == "get")
        {
            int index;
            cin >> index;
            try
            {
                cout << list.get(index) << "\n";
            }
            catch (const out_of_range& e)
            {
                cout << "Error: " << e.what() << "\n";
            }
        }
        else if (command == "set")
        {
            int index, value;
            cin >> index >> value;
            try
            {
                list.set(index, value);
            } catch (const out_of_range& e)
            {
                cout << "Error: " << e.what() << "\n";
            }
        }
        else if (command == "print")
        {
            cout << list << "\n";
        }
    }

    return 0;
}
```

Результат:

```
5

insert
0 3
1 2 3

erase
0 2

set
0 10

size
1
print
10
```

Created	Compiler	Result	Time (sec.)	Memory (MiB)	Actions
a day ago	C++ 23	Accepted	0.008	1.223	View

Task 6 - Practice# programming: Class Practice Task

```cpp
#include <iostream>

using namespace std;

struct Node
{
    int data;
    Node *next;

    Node(int value) : data(value), 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)
    {
        cout << head->data << " ";
        head = head->next;
    }
    cout << "\n";
}
```

```cpp

Node *createList()
{
    Node *head = nullptr;
    Node *tail = nullptr;
    int n;

    cout << "Enter number of elements in list: ";
    cin >> n;

    if (n <= 0)
    {
        cout << "List is empty!" << "\n";
        return nullptr;
    }

    cout << "Enter " << n << " integer numbers:" << "\n";
    for (int i = 0; i < n; i++)
    {
        int value;
        cin >> value;

        Node *newNode = new Node(value);

        if (head == nullptr)
        {
            head = newNode;
            tail = newNode;
        }
        else
        {
            tail->next = newNode;
            tail = newNode;
        }
    }

    return head;
}

```

```cpp
bool compare(Node *first, Node *second)
{
    bool same = true;
    int length1 = 0, length2 = 0;
    while (first != nullptr && second != nullptr)
    {
        if (first->data != second->data)
        {
            same = false;
        }
        first = first->next;
        second = second->next;
        if (first != nullptr)
        {
            length1++;
        }
        if (second != nullptr)
        {
            length2++;
        }
    }
    if ( length1 != length2)
    {
        same = false;
    }

    return same;
}

int main()
{
    Node *head1 = createList();

    cout << "First list: ";
    printList(head1);

    head1 = reverse(head1);

    cout << "Reversed list: ";
    printList(head1);
```

```cpp

    Node *head2 = createList();

    cout << "Second list: ";
    printList(head2);

    if (compare(head1, head2))
    {
        cout << "Second list is the same as reversed list number one!" << "\n";
    }
    else
    {
        cout << "Second list isn't the same as reversed list number one!" << "\n";
    }

    return 0;
}
```

Результат:

```
Enter number of elements in list: 5
Enter 5 integer numbers:
3
6
7
4
2
First list: 3 6 7 4 2
Reversed list: 2 4 7 6 3
Enter number of elements in list: 5
Enter 5 integer numbers:
2
4
7
6
3
Second list: 2 4 7 6 3
Second list is the same as reversed list number one!
```

```cpp
#include <iostream>

using namespace std;

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 TreeFree(Node*& root);
Node* TreeMirror(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);

    cout << "Tree: ";
    Show(myTree);
    cout << "\n";
    cout << "Mirrored Tree: ";
    TreeMirror(myTree);
    Show(myTree);
    TreeMirror(myTree);
    cout << "\n" << "Tree of sum: ";
    TreeSum(myTree);
    Show(myTree);
    TreeFree(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);
        cout << root->data << " ";
        Show(root->right);
    }
}

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

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

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
```

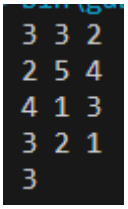
Task 7 - Practice# programming: Self Practice Task

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

using namespace std;

const int MAX_N = 1500;

int n, m, k;
int f[MAX_N][MAX_N];
int InRowMax[MAX_N][MAX_N];

void MaxRow()
{
    for (int i = 0; i < n; i++)
    {
        int queue[MAX_N], head = 0, tail = 0;
        for (int j = 0; j < m; j++)
        {
            if (head < tail && queue[head] <= j - k)
            {
                head++;
            }

            while (head < tail && f[i][queue[tail - 1]] <= f[i][j])
            {
                tail--;
            }

            queue[tail++] = j;

            if (j >= k - 1)
            {
                InRowMax[i][j - k + 1] = f[i][queue[head]];
            }
        }
    }
}
```

```cpp
int minMax()
{
    int result = INT_MAX;

    for (int j = 0; j <= m - k; j++)
    {
        int queue[MAX_N], head = 0, tail = 0;

        for (int i = 0; i < n; i++)
        {
            if (head < tail && queue[head] <= i - k)
            {
                head++;
            }

            while (head < tail && InRowMax[queue[tail - 1]][j] <= InRowMax[i][j])
            {
                tail--;
            }

            queue[tail++] = i;

            if (i >= k - 1) {
                result = min(result, InRowMax[queue[head]][j]);
            }
        }
    }

    return result;
}
```

```cpp
int main()
{
    cin >> n >> m >> k;

    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < m; j++)
        {
            cin >> f[i][j];
        }
    }

    MaxRow();

    int result = minMax();

    cout << result << "\n";

    return 0;
}
```

Результат:

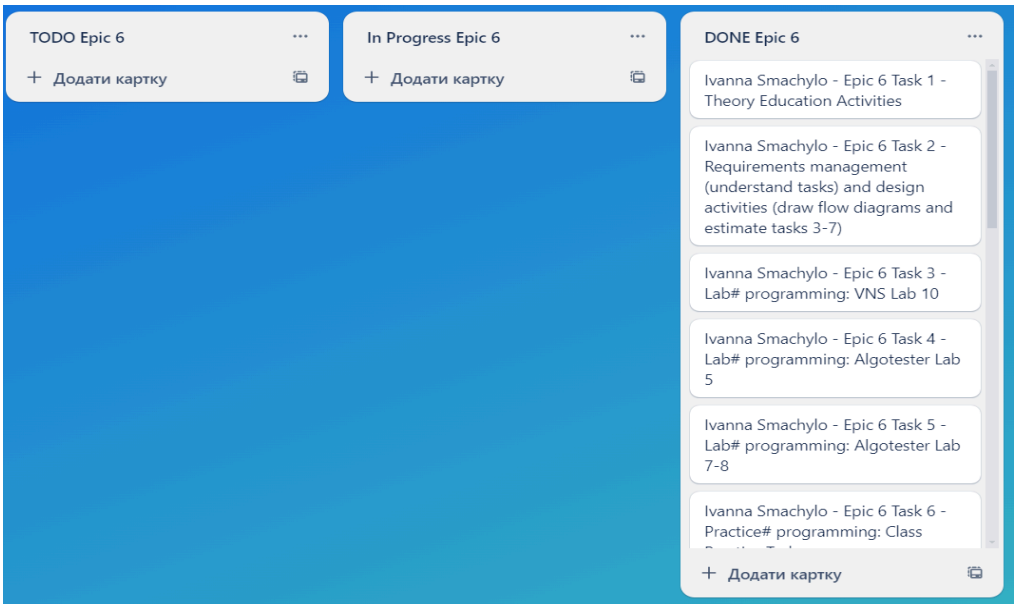


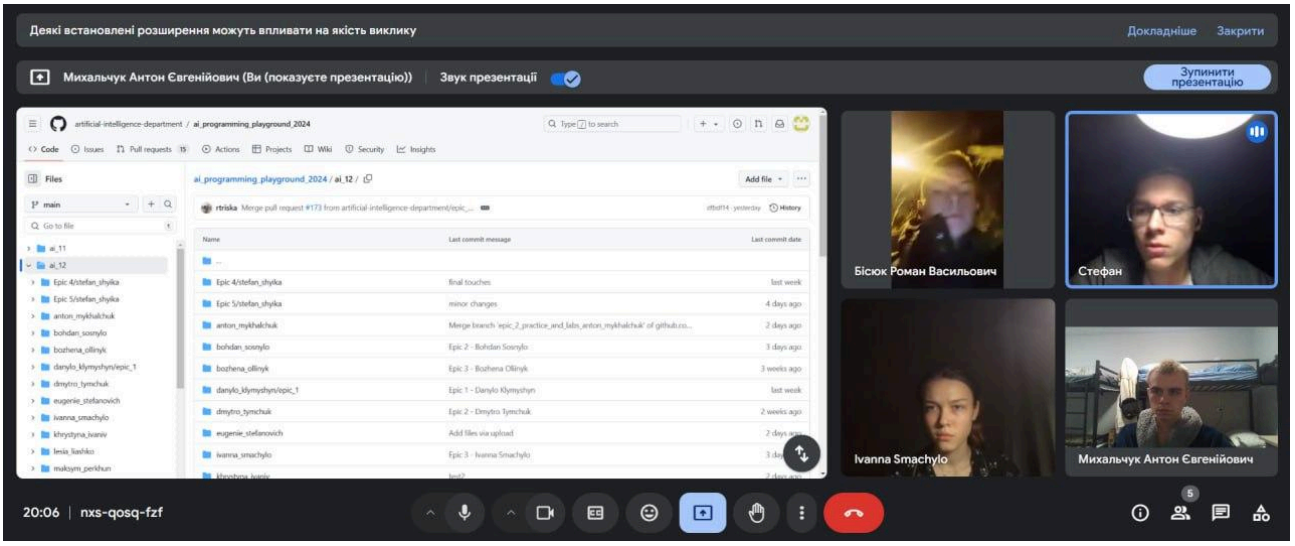
Created	Compiler	Result	Time (sec.)	Memory (MiB)	Actions
a few seconds ago	C++ 23	Accepted	0.971	18.516	View

Зустрічі з командою:

Зустрічались двічі для обговорення задач, поставлених в шостому епіку.

Створили нову дошку в Trello й бачили прогрес одне одного:





Висновок: в ході роботи над даним епіком я навчилась використовувати на практиці нові знання, такі як стек і купа, динамічні масиви, черга, зв’язні списки, список list, бінарні дерева, перевантаження операторів вводу і виводу, reverse, compare in C++.