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(ла):

Студент(ка) групи ШІ-12

Соснило Богдан Тарасович

Львів 2024

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

Мета роботи: Ознайомитися з основними динамічними структурами даних (черга, стек, списки, дерево), вивчити їхні особливості та застосування в алгоритмах. Розробити і реалізувати алгоритми обробки цих структур для ефективного виконання різних задач.

Теоретичні відомості

1. Перенавантаження операторі виводу

https://acode.com.ua/urok-141-perevantazhennya-operatoriv-vvodu-i-vyvodu/#toc-0

2. Класи

https://acode.com.ua/urok-121-klasy-ob-yekty-i-metody/#toc-1
https://acode.com.ua/urok-183-shablony-klasiv/

3. Черга

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. Стек

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/

5. Списки

https://codelessons.dev/ru/spisok-list-v-s-polnyj-material/

https://www.youtube.com/watch?v=-25REjF_atI

6. Дерева

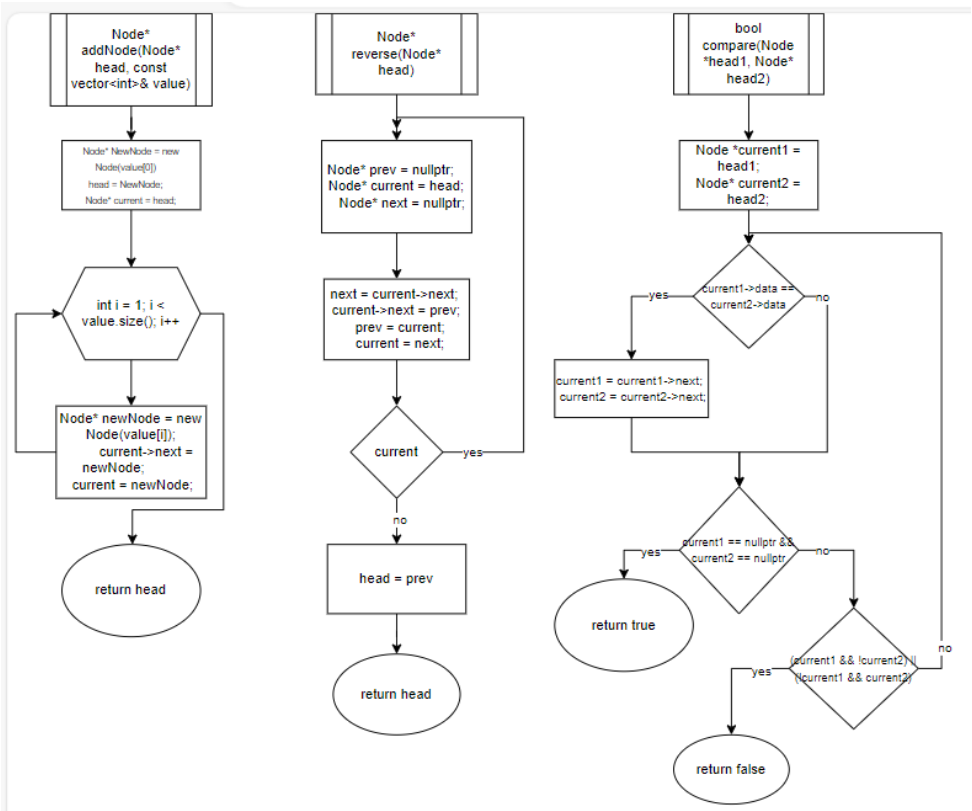
https://www.youtube.com/watch?v=qBFzNW0ALxQ

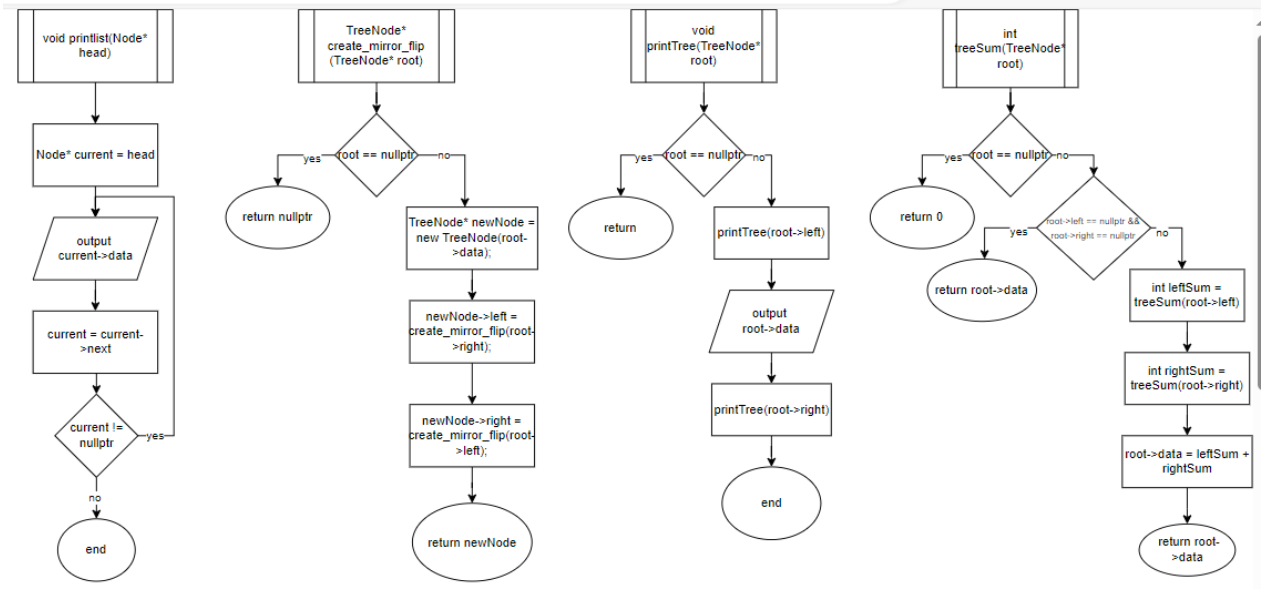
https://www.geeksforgeeks.org/binary-tree-data-structure/

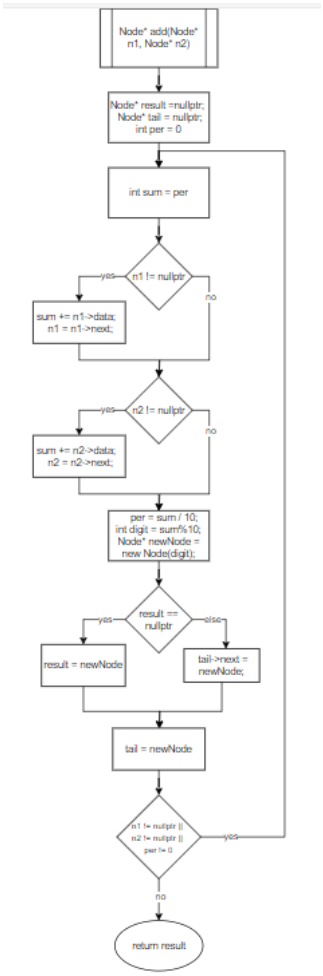
Виконання роботи

Task 2 - Requirements management (understand tasks) and design activities (draw flow diagrams and estimate tasks 3-7) (2 години)

Функції:

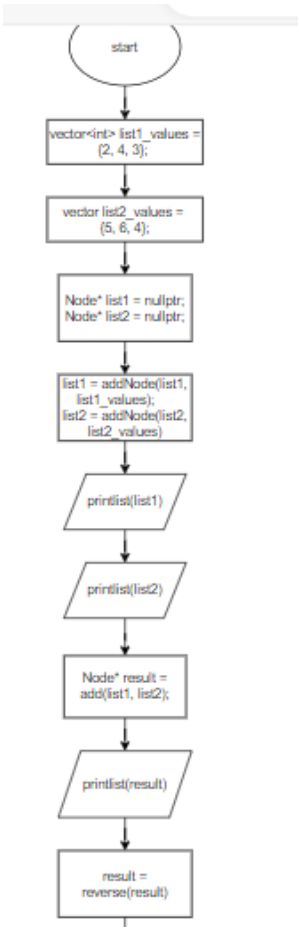




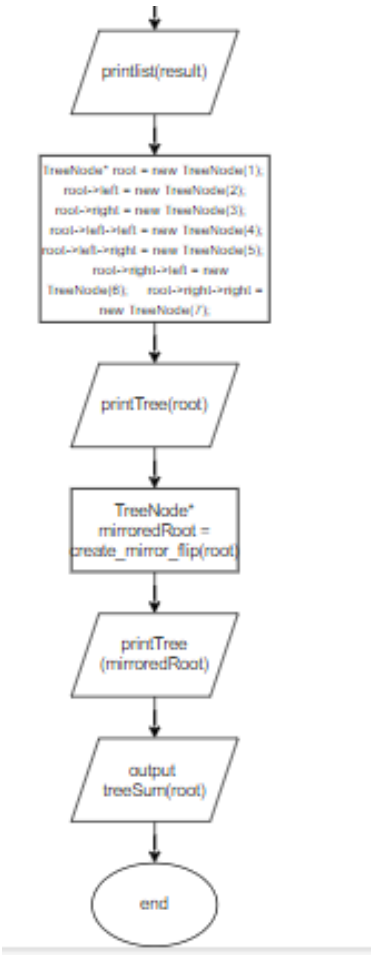


Основна програма:

1 частина



2 частина



Task 3 - Lab# programming: VNS Lab 10 (1 година)

4. Записи в лінійному списку містять ключове поле типу int. Сформувати однонаправлений список. Знищити з нього елемент із заданим номером, додати К елементів, починаючи із заданого номера;

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

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

Node* addNode(Node* head, const vector<int>& value){
    Node* NewNode = new Node(value[0]);
    head = NewNode;
    Node* current = head;
    for (int i = 1; i < value.size(); i++) {
        Node* newNode = new Node(value[i]);
        current->next = newNode;
        current = newNode;
    }
    return head;

    }

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

void printFile(string file) {
    ifstream inFile(file);
    if (!inFile) {
        cerr << "Error: Unable to open file " << file << endl;
        return;
    }

    string content;
    getline(inFile, content);
    inFile.close();

    if (content.empty()) {
        cout << "Список порожній" << endl;
    } else {
        cout << "Вміст файлу: " << content << endl;
    }
}

void addtoFile(string file, vector<int> node){
    ofstream outFile(file);

if (!outFile) {
        cerr << "Error: Unable to open file " << file << endl;
        return;
    }

    for (int n : node) {
        outFile << n << " ";
    }
    outFile.close();

}

void clearFile(string file) {
    ofstream outFile(file, ofstream::trunc);
    if (!outFile) {
        cerr << "Error: Unable to open file " << file << endl;
        return;
    }
    outFile.close();
}

Node* restoreFromFile(string file) {
    ifstream inFile(file);
    if (!inFile) {
        cerr << "Error: Unable to open file " << file << endl;
        return {};
    }
    Node* head = nullptr;
    vector<int> node;
    int value;
    while (inFile >> value) {
        node.push_back(value);
    }

    addNode(head, node);
    inFile.close();

    return head;
}

void clearList(Node*& head) {
    while (head) {
        Node* temp = head;
        head = head->next;
        delete temp;
    }
}

void deleteAtPosition(Node*& head, int position) {
    if (!head || position < 1) {
        cout << "Некоректний номер елемента" << endl;
        return;
    }
    if (position == 1) {
        Node* temp = head;
        head = head->next;
        delete temp;
        return;
    }
    Node* current = head;
    for (int i = 1; current && i < position - 1; ++i) {
        current = current->next;
    }
    if (!current || !current->next) {
        cout << "Елемент із заданим номером не існує" << endl;
        return;
    }
    Node* temp = current->next;
    current->next = temp->next;
    delete temp;
}

void addKElements(Node*& head, int position, int K) {
    if (position < 1 || K < 1) {
        cout << "Некоректний номер або кількість елементів" << endl;
        return;
    }
    Node* current = head;

    for (int i = 1; current && i < position; ++i) {
```

```
    for (int i = 1; current && i < position; ++i) {
        current = current->next;
    }

    if (!current) {
        cout << "Позиція поза списком" << endl;
        return;
    }

    for (int i = 0; i < K; ++i) {
        int value;
        cout << "Введіть значення для нового елемента: ";
        cin >> value;

        Node* newNode = new Node(value);
        newNode->next = current->next;
        current->next = newNode;
        current = newNode;
    }
}


int main() {

    string fileName = "list.txt";
    Node* head = nullptr;
    int k;
    vector<int> vec;
    cout << "Введіть к-сть елементів у списку: ";
    cin>>k;
    for(int i = 0; i<k;i++){
        cin>>vec[i];
    }
    head = addNode(head, vec);
    cout << "Початковий список: ";
    printlist(head);
    cout<<endl;
    deleteAtPosition(head, 2);
    cout << "Після видалення елемента з позиції 2: ";
    printlist(head);
    cout<<endl;
    addKElements(head, 3, 3);
    cout << "Після додавання 3 елементів із значенням від 50: ";
    printlist(head);
    cout<<endl;
    addtoFile(fileName, vec);

    clearList(head);
    cout << "Після знищення списку: ";
    printlist(head);
    cout<<endl;
    head = restoreFromFile(fileName);
    cout << "Після відновлення списку з файлу: ";
    printFile(fileName);
    cout<<endl;
    clearList(head);
    cout << "Після остаточного знищення списку: ";
    printlist(head);
    cout<<endl;
    return 0;
}
```

```
Введіть к-сть елементів у списку: 5
1 2 3 4 5
Початковий список: 12345
Після видалення елемента з позиції 2: 1345
Введіть значення для нового елемента: 7
Введіть значення для нового елемента: 8
Введіть значення для нового елемента: 9
Після додавання 3 елементів із значенням від 50: 1347895
Після знищення списку:
Після відновлення списку з файлу: Вміст файлу: 1 2 3 4 5

Після остаточного знищення списку:
```

Task 4 - Lab# programming: Algotester Lab 5(30 хв)

У вас є карта гори розміром $N \times M$.

Також ви знаєте координати $\{x, y\}$, у яких знаходиться вершина гори.

Ваше завдання - розмалювати карту таким чином, щоб найнижча точка мала число 0, а пік гори мав найбільше число.

Клітинкі які мають суміжну сторону з вершиною мають висоту на один меншу, суміжні з ними і не розфарбовані мають ще на 1 меншу висоту і так далі.

Вхідні дані

У першому рядку 2 числа N та M - розміри карти

у другому рядку 2 числа x та y - координати піку гори

Вихідні дані

N рядків по M елементів в рядку через пробіл - висоти карти.

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


int main(){

int N, M;
cin >> N >> M;
int matrix[N][M];
int x,y;
cin >>x >> y;
x--;
y--;
for(int i = 0; i<N; i++){
    for(int j = 0; j<M;j++){
        int distance = abs(i-x)+abs(j-y);
        matrix[i][j] = max(x,N-x-1) + max(y, M-y-1) - distance;

    }
}

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


    return 0;
}
```

```
3 4
3 3
0 1 2 1
1 2 3 2
2 3 4 3
```

Task 5 - Lab# programming: Algotester Lab 7-8(1) (1.5 години)

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

struct DynamicMas {
private:
    int cap;
    int size;
    int* mas;

    void resize(int newcap) {
        int* mas1 = new int[newcap];
        for (int i = 0; i < size; i++) {
            mas1[i] = mas[i];
        }
        delete[] mas;
        mas = mas1;
        cap = newcap;
    }

public:
    DynamicMas() {
        size = 0;
        cap = 1;
        mas = new int[cap];
    }

    ~DynamicMas() {
        delete[] mas;
    }

    void getsize() {
        cout << size << endl;
    }

    void getcapacity() {
        cout << cap << endl;
    }

    void insert(int a, int N, int* numb) {
        if (size + N >= cap) {
            while (cap <= size + N) {
                cap *= 2;
            }
            resize(cap);
        }
        for (int i = size - 1; i >= a; i--) {
            mas[i + N] = mas[i];
        }

        for (int i = 0; i < N; i++) {
            mas[a + i] = numb[i];
        }
        size += N;

    }
    void erase(int a, int n) {
        for (int i = a; i < size - n; i++) {
            mas[i] = mas[i + n];
        }
        size -= n;
    }

    int& operator[](int a) {
        return mas[a];
    }

    friend ostream& operator<<(ostream& out, const DynamicMas& arr) {
        for (int i = 0; i < arr.size; i++) {
            out << arr.mas[i] << " ";
        }
        return out;
    }
};

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

    for (int i = 0; i < Q; i++) {
        string com;
        cin >> com;
        if (com == "size") {
            vec.getsize();
        } else if (com == "capacity") {
            vec.getcapacity();
        } else if (com == "insert") {
            int a, b;
            cin >> a >> b;
            int* elem = new int[b];
            for (int j = 0; j < b; j++) {
                cin >> elem[j];
            }
            vec.insert(a, b, elem);
            delete[] elem;
        } else if (com == "erase") {
            int a, n;
            cin >> a >> n;
            vec.erase(a, n);
        } else if (com == "get") {
            int a;
            cin >> a;
            cout << vec[a] << endl;
        } else if (com == "set") {
            int index, value;
            cin >> index >> value;
            vec[index] = value;
        } else if (com == "print") {
            cout << vec << endl;
        }
    }

    return 0;
}
```

```
12
size
0
capacity
1
insert 0 2
100 100
size
2
capacity
4
insert 0 2
102 102
size
4
capacity
8
insert 0 2
103 103
size
6
capacity
8
print
103 103 102 102 100 100
```

З використанням шаблону класу:

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

template <typename shablon>
class DynamicMas {
private:
    int cap;
    int size;
    shablon* mas;

    void resize(int newcap) {
        shablon* newArr = new shablon[newcap];
        for (int i = 0; i < size; i++) {
            newArr[i] = mas[i];
        }
        delete[] mas;
        mas = newArr;
        cap = newcap;
    }

public:
    DynamicMas() {
        size = 0;
        cap = 1;
        mas = new shablon[cap];
    }

    ~DynamicMas() {
        delete[] mas;
    }

    void getSize() const {
        cout << size << endl;
    }

    void getCapacity() const {
        cout << cap << endl;
    }

    void insert(int pos, int count, shablon* elements) {
        if (size + count >= cap) {
            while (cap <= size + count) {
                cap *= 2;
            }
            resize(cap);
        }
        for (int i = size - 1; i >= pos; i--) {
            mas[i + count] = mas[i];
        }

        for (int i = 0; i < count; i++) {
            mas[pos + i] = elements[i];
        }
        size += count;
    }

    void erase(int pos, int count) {
        for (int i = pos; i < size - count; i++) {
            mas[i] = mas[i + count];
        }
        size -= count;
    }

    shablon& operator[](int index) {
        return mas[index];
    }

    friend ostream& operator<<(ostream& out, const DynamicMas<shablon>& dynamicMas) {
        for (int i = 0; i < dynamicMas.size; i++) {
            out << dynamicMas.mas[i] << " ";
        }
        return out;
    }
};

int main() {
    int Q;
    cin >> Q;
    DynamicMas<int> vec;

    for (int i = 0; i < Q; i++) {
        string command;
        cin >> command;
        if (command == "size") {
            vec.getSize();
        } else if (command == "capacity") {
            vec.getCapacity();
        } else if (command == "insert") {
            int pos, count;
            cin >> pos >> count;
            int* elements = new int[count];
            for (int j = 0; j < count; j++) {
                cin >> elements[j];
            }
            vec.insert(pos, count, elements);
            delete[] elements;
        } else if (command == "erase") {
            int pos, count;
            cin >> pos >> count;
            vec.erase(pos, count);
        } else if (command == "get") {
            int index;
            cin >> index;
            cout << vec[index] << endl;
        } else if (command == "set") {
            int index, value;
            cin >> index >> value;
            vec[index] = value;
        } else if (command == "print") {
            cout << vec << endl;
        }
    }

    return 0;
}
```

```
12
size
0
capacity
1
insert 0 2
100 100
size
2
capacity
4
insert 0 2
102 102
size
4
capacity
8
insert 0 2
103 103
size
6
capacity
8
print
103 103 102 102 100 100
```

7 годин тому	C++ 23	**Зараховано**	0.006	1.383	Перегляд
7 годин тому	C++ 23	**Зараховано**	0.006	1.285	Перегляд

Task 6 – Practice# programming: Class Practice Task(2 години)

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

using namespace std;

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

Node* addNode(Node* head, const vector<int>& value) {
    if (value.empty()) return nullptr;
    Node* newNode = new Node(value[0]);
    head = newNode;
    Node* current = head;
    for (int i = 1; i < value.size(); i++) {
        Node* temp = new Node(value[i]);
        current->next = temp;
        current = temp;
    }
    return head;
}

Node* reverse(Node* head) {
    Node* prev = nullptr;
    Node* current = head;
    Node* next = nullptr;
    while (current) {
        next = current->next;
        current->next = prev;
        prev = current;
        current = next;
    }
    head = prev;
    return head;
}

bool compare(Node* head1, Node* head2) {
    Node* current1 = head1;
    Node* current2 = head2;
    while (current1 && current2) {
        if (current1->data != current2->data) {
            return false;
        }
        current1 = current1->next;
        current2 = current2->next;
    }
    return current1 == nullptr && current2 == nullptr;
}

Node* add(Node* n1, Node* n2) {
    Node* result = nullptr;
    Node* tail = nullptr;
    int per = 0;

    while (n1 != nullptr || n2 != nullptr || per != 0) {
        int sum = per;

        if (n1 != nullptr) {
            sum += n1->data;
            n1 = n1->next;
        }

        if (n2 != nullptr) {
            sum += n2->data;
            n2 = n2->next;
        }

        per = sum / 10;
        int digit = sum % 10;

        Node* newNode = new Node(digit);

        if (result == nullptr) {
            result = newNode;
        } else {
            tail->next = newNode;
        }
        tail = newNode; // Необхідно оновити tail
    }

    return result;
}

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

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

TreeNode* create_mirror_flip(TreeNode* root) {
    if (root == nullptr) {
        return nullptr;
    }

    TreeNode* newNode = new TreeNode(root->data);
    newNode->left = create_mirror_flip(root->right);
    newNode->right = create_mirror_flip(root->left);

    return newNode;
}

void printTree(TreeNode* root) {
    if (root == nullptr) {
        return;
    }
    printTree(root->left);
    cout << root->data << " ";
    printTree(root->right);
}

int treeSum(TreeNode* root) {
    if (root == nullptr) {
        return 0;
    }


    int leftSum = treeSum(root->left);
    int rightSum = treeSum(root->right);

```

```
    int rightSum = treeSum(root->right);

    return root->data + leftSum + rightSum;
}

int main() {
    vector<int> list1_values = {2, 4, 3};
    vector<int> list2_values = {5, 6, 4};

    Node* list1 = nullptr;
    Node* list2 = nullptr;

    list1 = addNode(list1, list1_values);
    list2 = addNode(list2, list2_values);

    cout << "List 1: ";
    printlist(list1);

    cout << "List 2: ";
    printlist(list2);

    Node* result = add(list1, list2);

    cout << "Sum of List 1 and List 2: ";
    printlist(result);

    result = reverse(result);

    cout << "Reversed sum: ";
    printlist(result);

    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);
    root->right->left = new TreeNode(6);
    root->right->right = new TreeNode(7);

    cout << "Original Tree: ";
    printTree(root);
    cout << endl;

    TreeNode* mirroredRoot = create_mirror_flip(root);

    cout << "Mirrored Tree: ";
    printTree(mirroredRoot);
    cout << endl;

    cout << "Sum of Tree Nodes: " << treeSum(root) << endl;

    return 0;
}
```

```
List 1: 2 4 3
List 2: 5 6 4
Sum of List 1 and List 2: 7 0 8
Reversed sum: 8 0 7
Original Tree: 4 2 5 1 6 3 7
Mirrored Tree: 7 3 6 1 5 2 4
Sum of Tree Nodes: 28
```

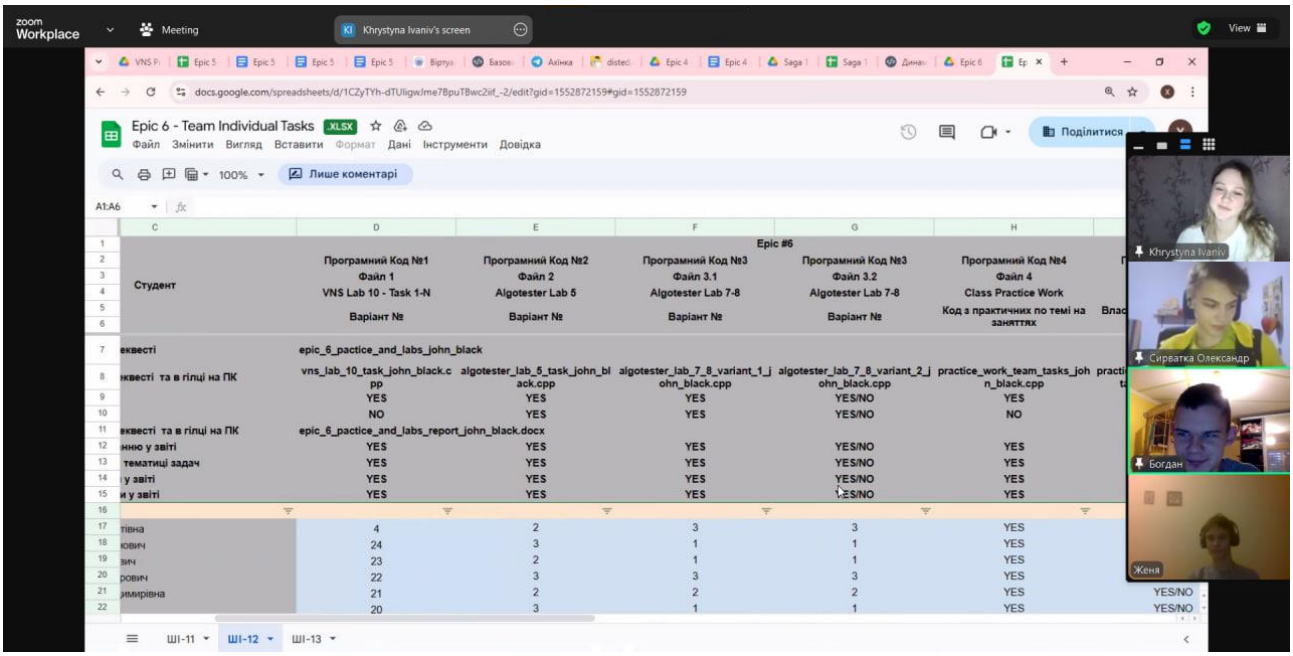
Task 7 - Practice# programming: Self Practice Task

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

using namespace std;

int main() {
    string s;
    cin >> s;

    int n = s.length();

    int changes1 = 0, changes2 = 0;

    for (int i = 0; i < n; ++i) {
        if (i % 2 == 0) {
            if (s[i] != 'B') {
                changes1++;
            }
        } else {
            if (s[i] != 'Y') {
                changes1++;
            }
        }
    }

    for (int i = 0; i < n; ++i) {
        if (i % 2 == 0) {
            if (s[i] != 'Y') {
                changes2++;
            }
        } else {
            if (s[i] != 'B') {
                changes2++;
            }
        }
    }

    cout << min(changes1, changes2) << endl;

    return 0;
}
```

```
YYBBB
2
```

Зустрічі з командою

З командою зустрічалися двічі, на зустрічах обговорювали питання та прогрес по епіку.



Висновок: в ході цього етапу я здобув навички роботи з різними структурами даних, такими як черга, список, стек та дерево, а також оволодів алгоритмами для обробки динамічних структур.