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

Виконав:
Студент групи ШІ-12
Климишин Данило

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: Списки, Дерево.
2. Навчитись виконувати алгоритми обробки динамічний структур.

Джерела:

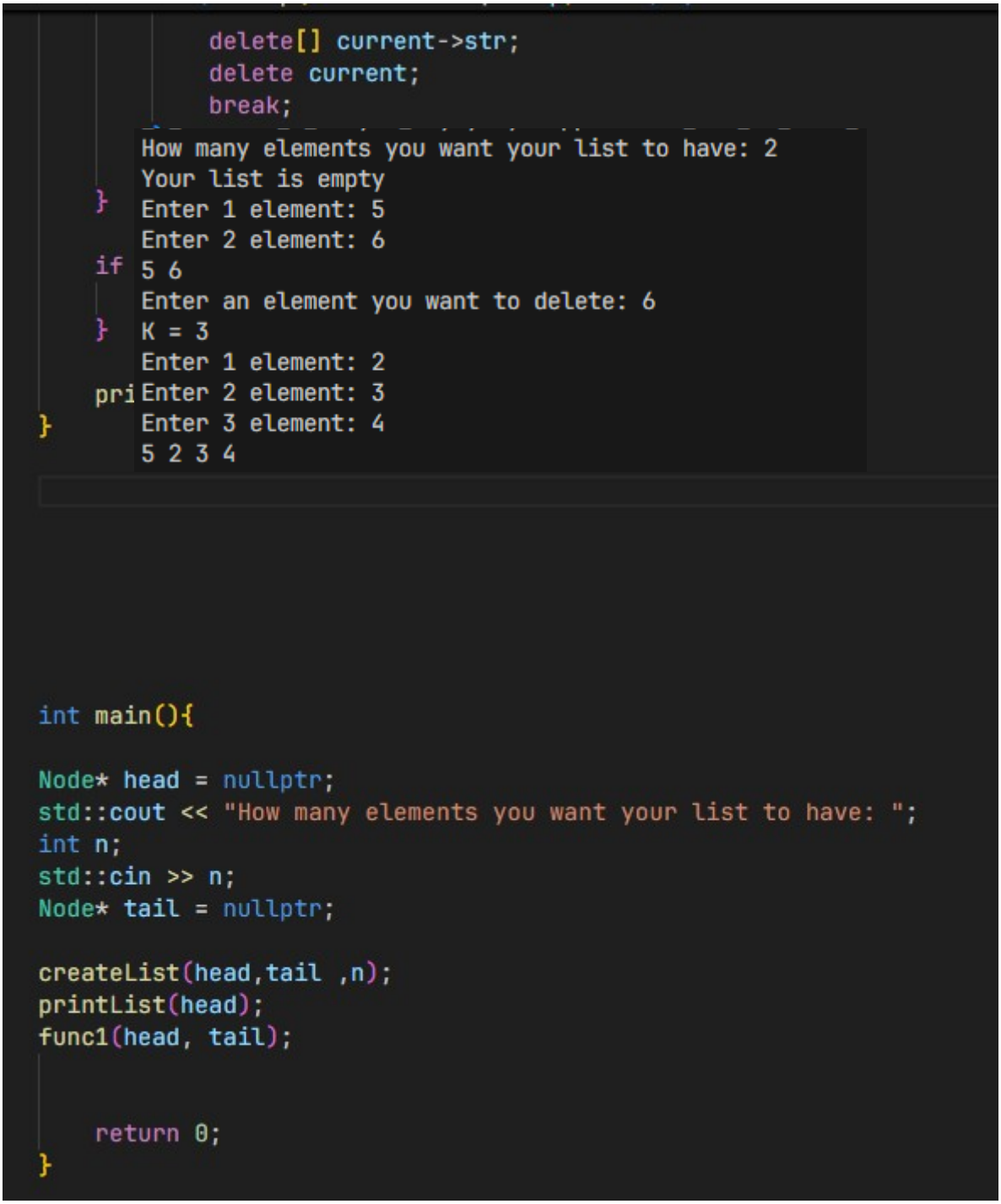
https://www.youtube.com/watch?v=qBFzNW0ALxQ
https://www.youtube.com/watch?v=QLzu2-_QFoE&t=355s
https://www.youtube.com/watch?v=-25REjF_atI&t=376s

Виконання роботи

Завдання №1 -VNS lab 10 - варіант 22

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

struct Node{
    char *str;
    Node *next_el;
    Node *prev;
};



void createList(Node*& head, Node*& tail, int n){
    if(head == nullptr){
        std::cout << "Your list is empty"<< std::endl;
}

tail = nullptr;
std::cout << "Enter 1 element: ";
char *text = new char[50];
std::cin.ignore();

fgets(text, 50, stdin);
text[strcspn(text, "\n")] = '\0';

head = new Node {text, nullptr};
Node* current = head;
char *loop;
for(int i = 0;i < n-1; ++i){
    std::cout << "Enter " << i + 2 << " element: ";
    loop = new char[50];
    fgets(loop, 50, stdin);
    loop[strcspn(loop, "\n")] = '\0';

    current->next_el = new Node {new char[strlen(loop) + 1], nullptr};
    strcpy(current->next_el->str, loop);
    current->next_el->prev = current;
    current = current->next_el;
}
```

```
tail = current;
while (current != nullptr) {
    if (strcmp(current->str, temp) == 0) {
        found = true;

        Node* nextNode = current->next_el;
        Node* previousNode = current->prev;
        for (int i = 0; i < k; ++i) {
            std::cout << "Enter " << i + 1 << " element: ";
            loop = new char[50];
            fgets(loop, 50, stdin);
            loop[strcspn(loop, "\n")] = '\0';

            Node* newNode = new Node {new char[strlen(loop) + 1], nullptr, previousNode};
            strcpy(newNode->str, loop);


            if (previousNode != nullptr) {
                previousNode->next_el = newNode;
            } else {
                head = newNode;
            }
            previousNode = newNode;
        }

        if (previousNode != nullptr) {
            previousNode->next_el = current->next_el;
        }
        if (nextNode != nullptr) {
            nextNode->prev = previousNode;
        }

        if (current == head) {
            head = current->next_el;
        }
        if (current == tail) {
            tail = previousNode;
        }

        --

    char *loop;
    bool found = false;

    while (current != nullptr) {
        if (strcmp(current->str, temp) == 0) {
            found = true;
```



```
            delete[] current->str;
            delete current;
            break;
        }
    }
    if (!found) {
        cout << "Element not found in the list." << endl;
    }
}
void printList(Node* head) {
    Node* current = head;
    while (current != nullptr) {
        cout << current->str << " ";
        current = current->next;
    }
    cout << endl;
}
int main(){

Node* head = nullptr;
std::cout << "How many elements you want your list to have: ";
int n;
std::cin >> n;
Node* tail = nullptr;

createList(head,tail ,n);
printList(head);
func1(head, tail);


    return 0;
}
```

```
How many elements you want your list to have: 2
Your list is empty
Enter 1 element: 5
Enter 2 element: 6
5 6
Enter an element you want to delete: 6
K = 3
Enter 1 element: 2
Enter 2 element: 3
Enter 3 element: 4
5 2 3 4
```

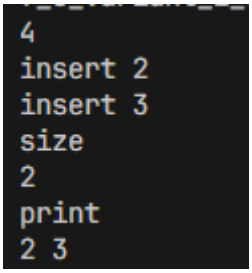
Завдання №2 **A**lgotester lab 5 - варіант 3

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

using namespace std;

int main() {
    int N, M, x, y;
    cin >> N >> M >> x >> y;
    x--; y--;

    vector<vector<int>> mount(N, vector<int>(M, -1));
    queue<pair<int, int>> q;
    q.push({x, y});

    int max_height = max(x, N - 1 - x) + max(y, M - 1 - y);
    mount[x][y] = max_height;

    int dx[] = {-1, 1, 0, 0};
    int dy[] = {0, 0, -1, 1};

    while (!q.empty()) {
        int cx = q.front().first;
        int cy = q.front().second;
        q.pop();
        for (int i = 0; i < 4; i++) {
            int nx = cx + dx[i];
            int ny = cy + dy[i];
            if (nx >= 0 && nx < N && ny >= 0 && ny < M && mount[nx][ny] == -1) {
                mount[nx][ny] = mount[cx][cy] - 1;
                q.push({nx, ny});
            }
        }
    }

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

    return 0;
}
```

Завдання №3 **A**lgotester lab 7-8 - варіант 3

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

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

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

void insert(Node*& root, int value) {
    if (root == nullptr) {
        root = new Node;
        root->value = value;
        root->left = nullptr;
        root->right = nullptr;

   void print(Node* root) {
       if (root == nullptr)
           return;
       print(root->left);
       cout << root->value << " ";
}      print(root->right);
   }
bo
   int main() {
       int Q;
       cin >> Q;
       Node* root = nullptr;

       while (Q--) {
           string command;
           cin >> command;
}
           if (command == "insert") {
in             int value;
               cin >> value;
               insert(root, value);
           } else if (command == "contains") {
}              int value;
               cin >> value;
               if (search(root, value)) {
                   cout << "Yes\n";
               } else {
                   cout << "No\n";
               }
           } else if (command == "size") {
               cout << size(root) << endl;
           } else if (command == "print") {
               print(root);
               cout << endl;
           }
       }

       return 0;
   }
```



Завдання №4 Class Practice Work

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

struct Node{
    int value;
    Node *next_el;
    Node *prev;
};
struct TreeNode {
    int value;
    TreeNode* left;
    TreeNode* right;
};

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

 void insert(TreeNode*& root, int value) {
     if (root == nullptr) {
         root = new TreeNode;
         root->value = value;
         root->left = nullptr;
         root->right = nullptr;
}    } else {
         if (value < root->value) {
v            insert(root->left, value);
         } else {
             insert(root->right, value);
         }
     }
 }


}
```

```cpp
void createList(Node*& head, Node*& tail, int n){

tail = nullptr;
std::cout << "Enter 1 element: ";
int el;
std::cin >> el;


head = new Node {el, nullptr};
Node* current = head;
for(int i = 0;i < n-1; ++i){

    std::cout << "Enter " << i + 2 << " element: ";
    std::cin >> el;
    current->next_el = new Node {el, nullptr};
    current->next_el->prev = current;
    current = current->next_el;
}
```

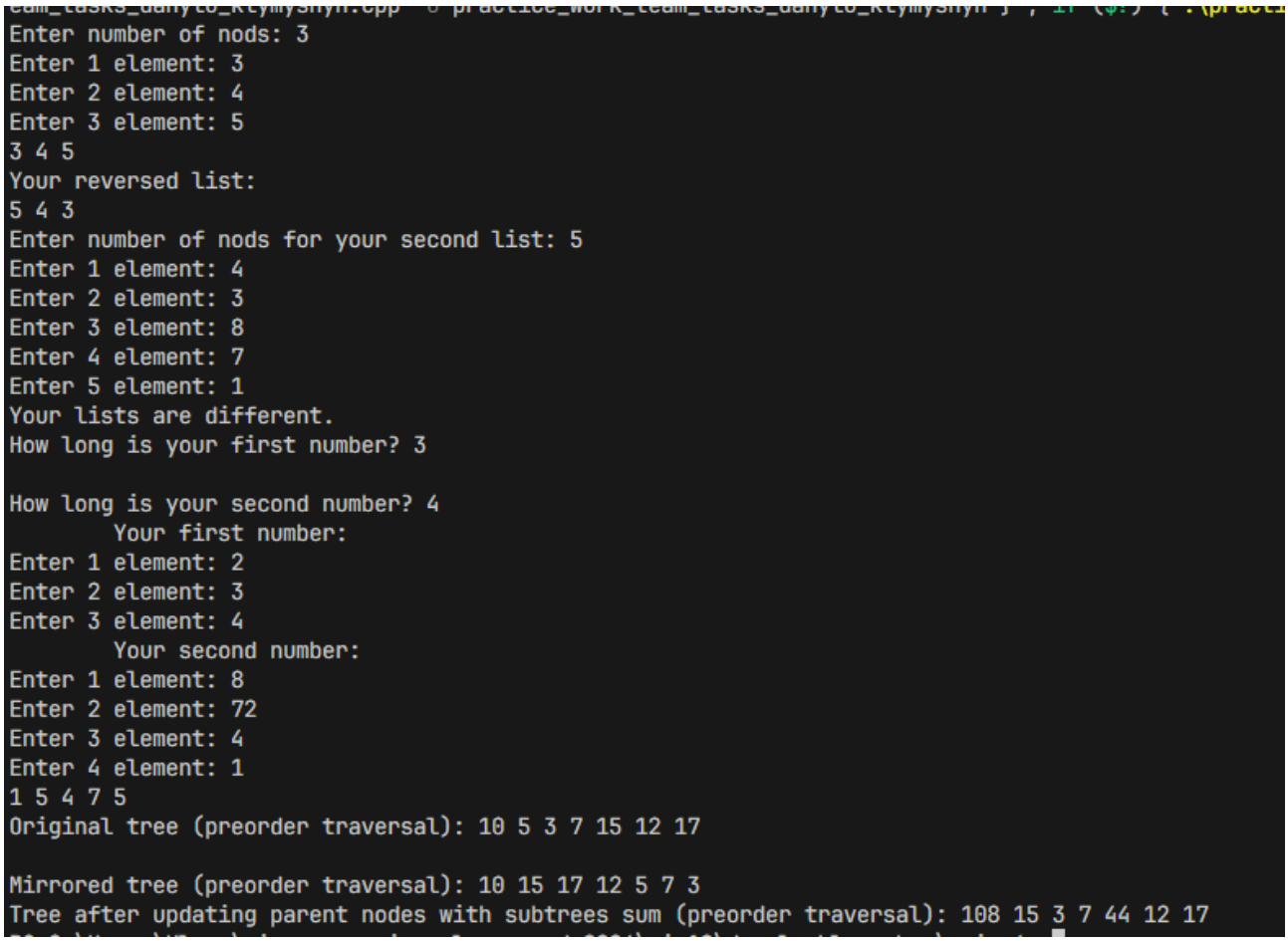
```
tail = current;
}


void printList(Node *head){

Node *current = head;


while(current != nullptr){
    std::cout << current->value << ' ';
    current = current->next_el;
}
    return;
}


Node* reverse(Node*& head) {
    if (head == nullptr || head->next_el == nullptr) {
        return head;
    }

    Node* left = head;
    Node* right = head;

    while (right->next_el != nullptr) {
        right = right->next_el;
    }

    while (left != right && left->prev != right) {
        std::swap(left->value, right->value);
        left = left->next_el;
        right = right->prev;
    }

    return head;
}
```

```
bool check = true;
int main(){


Node* head = nullptr;
Node* tail = nullptr;
Node* head2 = nullptr;
Node* tail2 = nullptr;



Node* h1 = nullptr;
Node* t1 = nullptr;

Node* h2 = nullptr;
Node* t2 = nullptr;

int n;
std::cout << "Enter number of nods: ";
std::cin >> n;

}createList(head, tail, n);

printList(head);
iNode* x = reverse(head);
std::cout << "\nYour reversed list: " << std::endl;
printList(head);



int k;
std::cout << "\nEnter number of nods for your second list: ";
std::cin >> k;
createList(head2, tail2, k);

}bool check_result = compare(head, head2);
if(check_result)
    std::cout << "Your lists are similiar.";
else
```

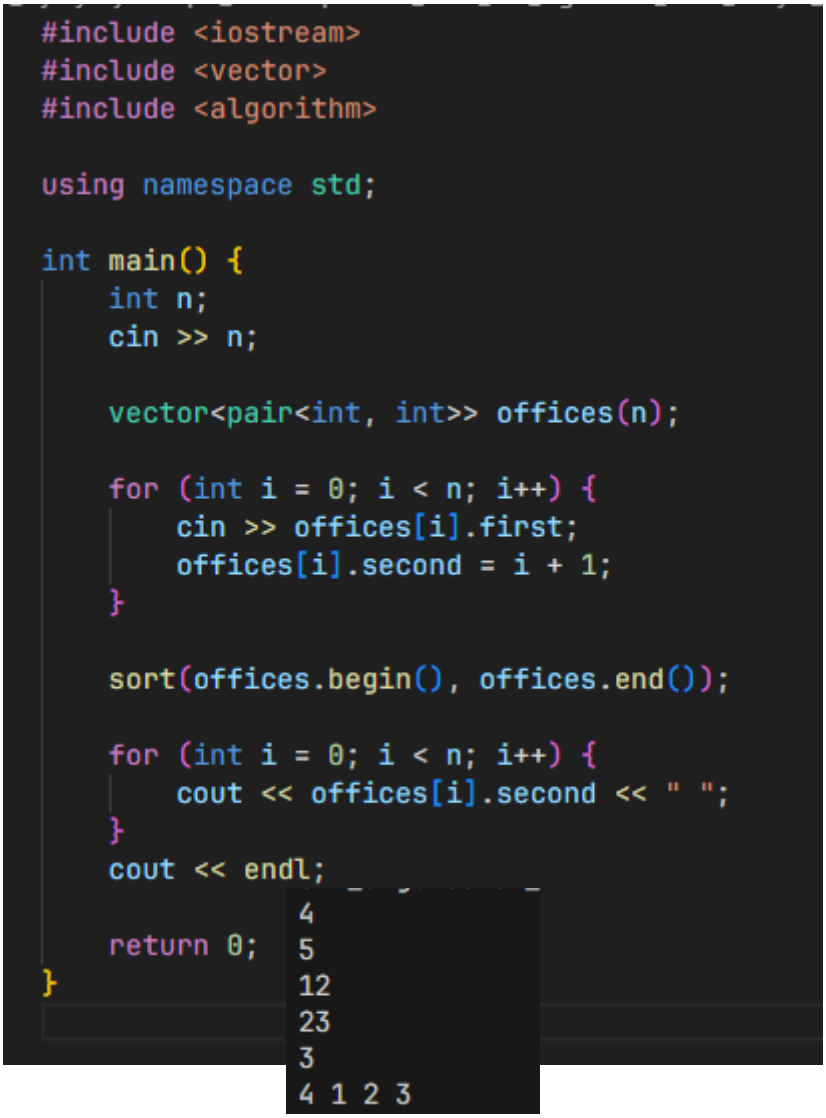
```
else
    std::cout << "Your lists are different.";

std::cout << "\nHow long is your first number? ";
int n1;
std::cin >> n1;
std::cout << "\nHow long is your second number? ";
int n2;
std::cin >> n2;

std::cout << "\tYour first number: " << std::endl;
```

```
    std::cout << "\nOriginal tree (preorder traversal): ";
    printTree(root);
    std::cout << std::endl;

    TreeNode* mirrorTree = create_mirror_flip(root);

    std::cout << "\nMirrored tree (preorder traversal): ";
    printTree(mirrorTree);
    std::cout << std::endl;



    tree_sum(root);

    std::cout << "Tree after updating parent nodes with subtrees sum (preorder traversal): ";
    printTree(root);
    std::cout << std::endl;




    return 0;
}
```

```
    std::cout << "\nOriginal tree (preorder traversal): ";
    printTree(root);
    std::cout << std::endl;

    TreeNode* mirrorTree = create_mirror_flip(root);

    std::cout << "\nMirrored tree (preorder traversal): ";
```



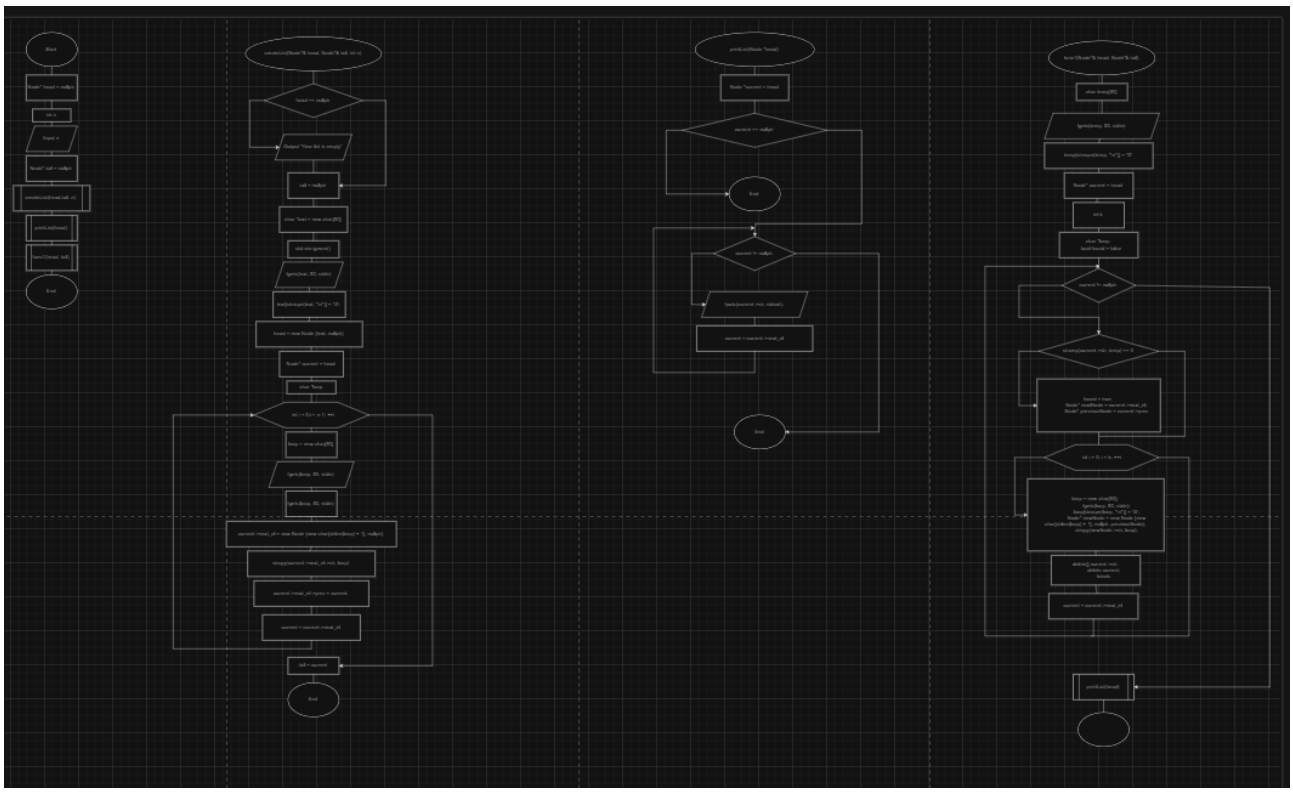
Завдання №5 Self Practice Work



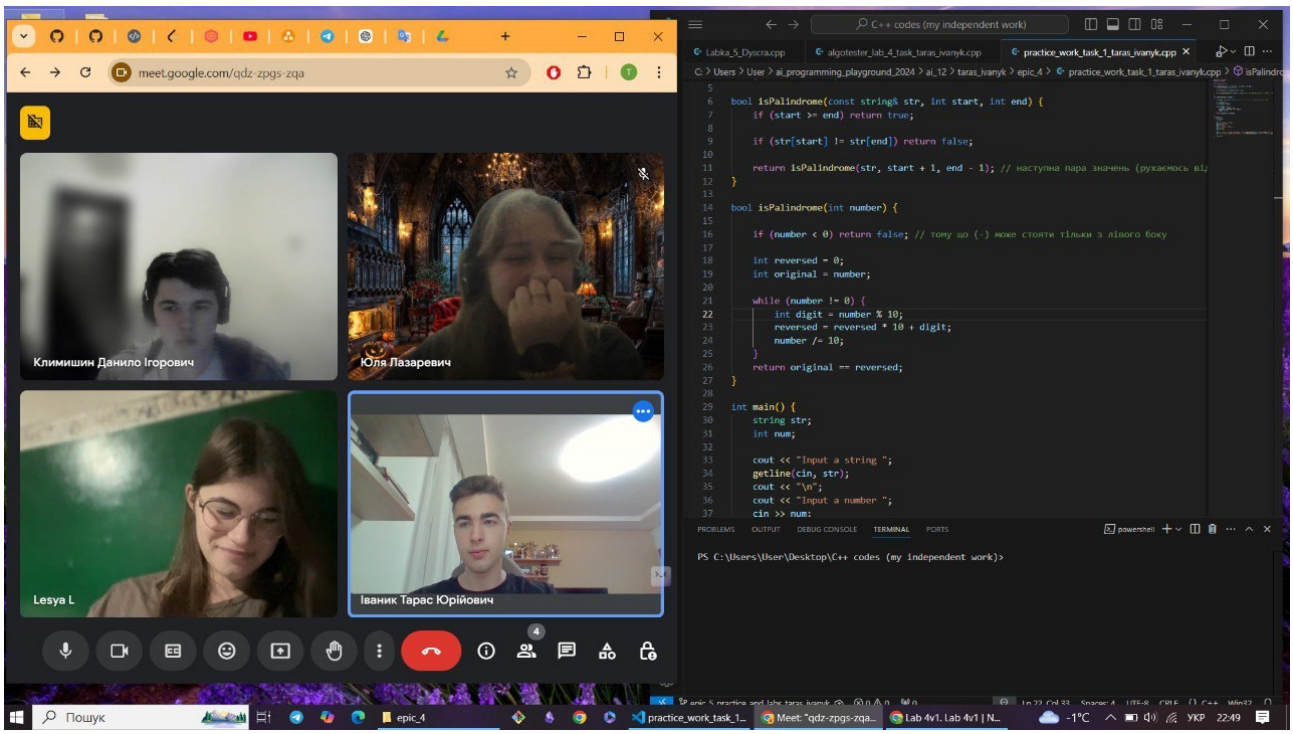
```cpp
#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

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

    vector<pair<int, int>> offices(n);

    for (int i = 0; i < n; i++) {
        cin >> offices[i].first;
        offices[i].second = i + 1;
    }

    sort(offices.begin(), offices.end());

    for (int i = 0; i < n; i++) {
        cout << offices[i].second << " ";
    }
    cout << endl;

    return 0;
}
```

```
4
5
12
23
3
4 1 2 3
```

Діаграма:



Зустрічі з командою



Висновок:

Завдяки цій роботі я зрозумів принципи динамічних структур даних, їх ключові операції та відмінності між статичним і динамічним виділенням пам’яті. Практичні вправи зі зв’язним списком і деревом допомогли мені опанувати алгоритми обробки даних у пам'яті та ефективне управління ресурсами. Цей досвід дозволив мені створювати адаптивні рішення для роботи з великими та змінними обсягами даних, підвищуючи продуктивність програм.