
# 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

Лящук Соломія Володимирівна

Львів 2024

**ITERATION 6 / EPIC 6**

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

**Tasks - (Задачі 6 ітерації в Trello або Any Tasks Manager):**

- Epic 6 Task 1 - Theory Education Activities
- Epic 6 Task 2 - Requirements management (understand tasks) and design activities (draw flow diagrams and estimate tasks 3-7)
- Epic 6 Task 3 - Lab# programming: VNS Lab 10
- Epic 6 Task 4 - Lab# programming: Algotester Lab 5
- Epic 6 Task 5 - Lab# programming: Algotester Lab 7-8
- Epic 6 Task 6 - Practice# programming: Class Practice Task
- Epic 6 Task 7 - Practice# programming: Self Practice Task
- Epic 6 Task 8 - Result Documentation Report and Outcomes Placement Activities (Docs and Programs on GitHub)
- Epic 6 Task 9 - Results Evaluation and Release

# Task 1

## Theory Education Activities

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

**Agenda (План Практичної):**

1. Основи Динамічних Структур Даних:
    - Вступ до динамічних структур даних: визначення та важливість
    - Виділення пам'яті для структур даних (stack і heap)
    - Приклади простих динамічних структур: динамічний масив
2. Стек:
    - Визначення та властивості стеку
    - Операції push, pop, top: реалізація та використання
    - Приклади використання стеку: обернений польський запис, перевірка балансу дужок
    - Переповнення стеку
3. Черга:
    - Визначення та властивості черги
    - Операції enqueue, dequeue, front: реалізація та застосування
    - Приклади використання черги: обробка подій, алгоритми планування
    - Розширення функціоналу черги: пріоритетні черги
4. Зв'язні Списки:
    - Визначення однозв'язного та двозв'язного списку
    - Принципи створення нових вузлів, вставка між існуючими, видалення, створення кільця(circular linked list)
    - Основні операції: обхід списку, пошук, доступ до елементів, об'єднання списків
    - Приклади використання списків: управління пам'яттю, FIFO та LIFO структури
5. Дерева:
    - Вступ до структури даних "дерево": визначення, типи
    - Бінарні дерева: вставка, пошук, видалення
    - Обхід дерева: в глибину (preorder, inorder, postorder), в ширину
    - Застосування дерев: дерева рішень, хеш-таблиці
    - Складніші приклади дерев: AVL, Червоно-чорне дерево

6. Алгоритми Обробки Динамічних Структур:
    - Основи алгоритмічних патернів: ітеративні, рекурсивні
    - Алгоритми пошуку, сортування даних, додавання та видалення елементів

Sourse:

https://www.youtube.com/watch?v=-25REjF_atI&list=PLiPRE8VmJzOpn6PzYf0higmCEyGzo2A5g&index=139

https://prometheus.org.ua/cs50/sections/section6.html

https://www.geeksforgeeks.org/list-reverse-function-in-c-stl/

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/#google_vignette

https://itproger.com/ua/spravka/cpp/queue

https://www.geeksforgeeks.org/circular-linked-list-in-cpp/

https://purecodecpp.com/uk/archives/2483#google_vignette

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

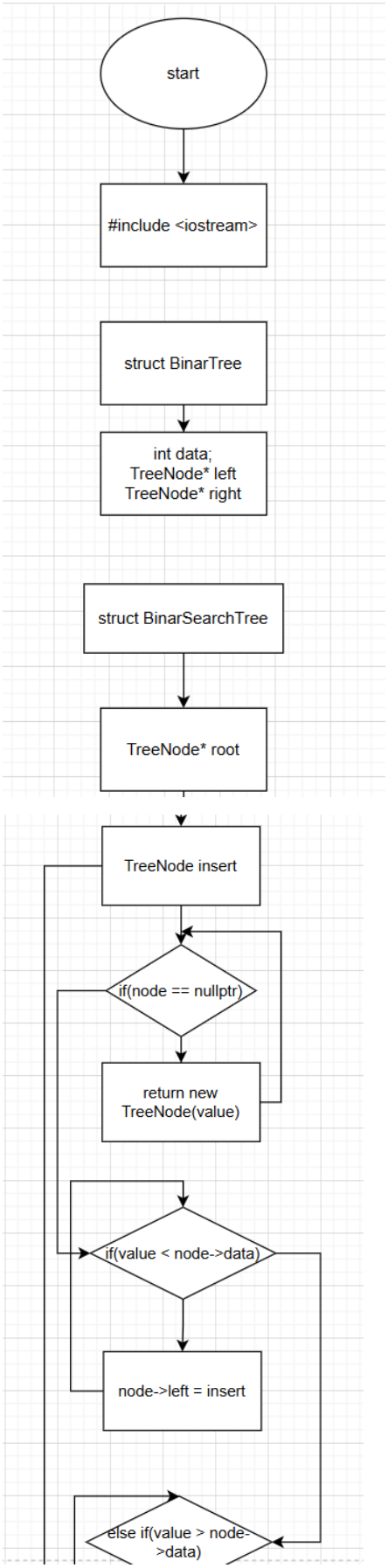
https://www.bestprog.net/uk/2019/09/18/c-the-concept-of-stack-operations-on-the-stack-an-example-implementation-of-the-stack-as-a-dynamic-array-ua/#google_vignette

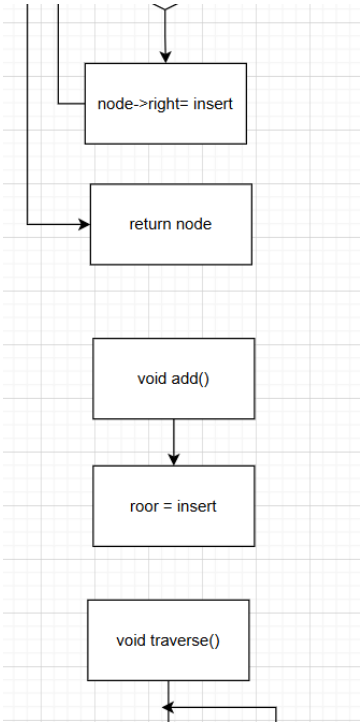
https://www.geeksforgeeks.org/stdstringcompare-in-c/

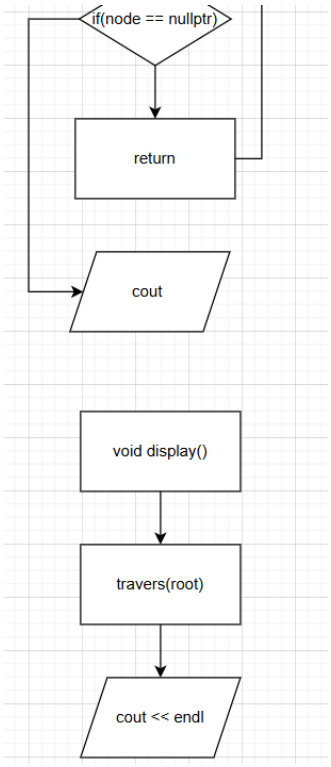
https://www.geeksforgeeks.org/cpp-program-for-inserting-a-node-in-a-linked-list/

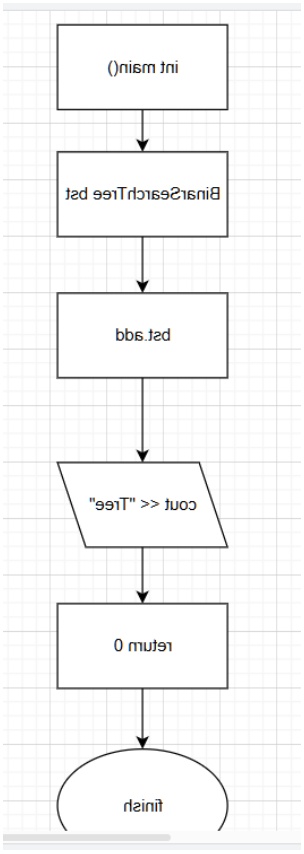
# Task 2

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


start
#include <iostream>
struct BinarTree
int data;
TreeNode* left
TreeNode* right
struct BinarSearchTree
TreeNode* root
TreeNode insert
if(node == nullptr)
return new
TreeNode(value)
if(value < node->data)
node->left = insert
else if(value > node-
>data)


node->right= insert
return node
void add()
roor = insert
void traverse()


if(node == nullptr)
return
cout
void display()
travers(root)
cout << endl




# Task 3

## VNS Lab 10

21.Записи в лінійному списку містять ключове поле типу *char (рядок символів). Сформувати двонаправлений список. Знищити елементи перед і після елемента із заданим ключем. Додати по К елементів у початок й у кінець списку.

vns_lab_10_task_solomia_liashchuk.cpp | shsu.cpp

C: > cpp > vns_lab_10_task_solomia_liashchuk.cpp > addEnd(Node *&, const string &)

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

using namespace std;

struct Node
{
    Node* next;
    Node* prev;
    string key;

    Node(const string& value)
    {
        key = value;
        next = nullptr;
        prev = nullptr;
    }
};

void addEnd(Node*& head, const string& value)
{
    Node* newNode = new Node(value);

    if(head == nullptr)
    {
        head = newNode;
        return;
    }

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

```

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

void deleteElements(Node*& head, const string& targetKey)
{
    Node* current = head;

    while(current != nullptr && current->key != targetKey)
    {
        current = current->next;
    }

    if(current == nullptr)
    {
        cout << "Error: Key not found" << endl;
        return;
    }

    if(current->prev != nullptr)
    {
        Node* toDelete = current->prev;

        if(toDelete->prev != nullptr)
        {
            toDelete->prev->next = current;
        }
        else
        {
            head = current;
        }

        current->prev = toDelete->prev;
        delete toDelete;
    }

    if(current->next != nullptr)
    {
        Node* toDelete = current->next;

        if(toDelete->next != nullptr)
        {
            toDelete->next->prev = current;
        }

        current->next = toDelete->next;
        delete toDelete;
    }
}

void printList(Node* head)
{
    Node* current = head;

    while(current != nullptr)
    {
        cout << current->key << " -> ";
        current = current->next;
    }
    cout << "nullptr" << endl;
}
```

```
97
98  void addToStart(Node*& head, const string& value)
99  {
100     Node* newNode = new Node(value);
101     newNode->next = head;
102     if(head != nullptr)
103     {
104         head->prev = newNode;
105     }
106
107     head = newNode;
108 }
109
110 int main()
111 {
112     Node* head = nullptr;
113     addEnd(head, "Блакитний");
114     addEnd(head, "Червоний");
115     addEnd(head, "Чорний");
116     addEnd(head, "Рожевий");
117
118     cout << "Початковий список:" << endl;
119     printList(head);
120
121     deleteElements(head, "Beta");
122     cout << "Список після видалення елементів перед і після 'Beta':" << endl;
123     printList(head);
124
125     addToStart(head, "Start");
126     cout << "Список після додавання елемента на початок:" << endl;
127     printList(head);
128
129     return 0;
130 }
```

# Task 4

## Algotester Lab 5

### Lab 5v2

*Limits: 1 sec., 256 MiB*

В пустелі існує незвичайна печера, яка є двохвимірною. Її висота це $N$, ширина - $M$.

Всередині печери є пустота, пісок та каміння. Пустота позначається буквою O , пісок $S$ і каміння $X$;

Одного дня стався землетрус і весь пісок посипався вниз. Він падає на найнижчу клітинку з пустотою, але він не може пролетіти через каміння.

Ваше завдання сказати як буде виглядати печера після землетрусу.

```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 col = 0; col < M; col++)
    {
        int empty_row = N - 1;
        for(int row = N - 1; row >= 0; row--)
        {
            if(cave[row][col] == 'X')
            {
                empty_row = row - 1;
            }
            else if(cave[row][col] == 'S')
            {
                cave[row][col] = 'O';
                if(empty_row >= 0)
                {
                    cave[empty_row][col] = 'S';
                    empty_row--;
                }
            }
        }
    }

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

    return 0;
}
```

## Task 5

### Algotester Lab 7-8

### V – 2

Ваше завдання - власноруч реалізувати структуру даних "Динамічний масив". Ви отримаєте QQ запитів, кожен запит буде починатися зі слова-ідентифікатора, після якого йдуть його аргументи.

C: > cpp > algotester_lab_7_8_variant_2_solomia_liashchuk.cpp > DynamicArray<S> > createSize(int)

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

template<typename S>
class DynamicArray
{
    private:
    S* data;
    int size;
    int capacity;

    void createSize(int newCapcity)
    {
        S* newData = new S[newCapcity];
        for(int i = 0; i < size; i++)
        {
            newData[i] = data[i];
        }
        delete[] data;
        data = newData;
        capacity = newCapcity;
    }

    public:
    DynamicArray() : size(0), capacity(1)
    {
        data = new S[capacity];
    }
    ~DynamicArray()
    {
        delete[] data;
    }

    void insert(int index, int n, S* values)
    {
        if(size + n > capacity)
        {
            int new_capacity = capacity;
            while(size + n >= new_capacity)
            {
                new_capacity *= 2;
            }
            createSize(new_capacity);
        }
        for(int i = size - 1; i >= index; i--)
        {
            data[i + n] = data[i];
        }
        for(int i = 0; i < n; i++)
        {
            data[index + i] = values[i];
        }
        size += n;
        if(size == capacity)
        {
            capacity *= 2;
            createSize(capacity);
        }
    }

```

```
61      void erase(int index, int n)
62      {
63          for (int i = index + n; i < size; i++)
64          {
65              data[i - n] = data[i];
66          }
67          size -= n;
68
69      }
70
71      int get_size() const
72      {
73          return size;
74      }
75      int get_capacity() const
76      {
77          return capacity;
78      }
79      S& operator[](int index)
80      {
81          if(index < 0 || index >= size)
82          {
83              throw out_of_range("ERROR");
84          }
85          return data[index];
86      }
87      friend ostream& operator<<(ostream& os, const DynamicArray& arr)
88      {
89          for (int i = 0; i < arr.size; i++)
90          {
91              os << arr.data[i] << (i < arr.size - 1 ? " " : "");
92          }
93          return os;
94      }
```

```cpp
int main()
{
    DynamicArray<int> arr;
    int Q;
    cin >> Q;

    for(int i = 0; i < Q; i++)
    {
        string c;
        cin >> c;

        if(c == "insert")
        {
            int index, n;
            cin >> index >> n;
            int* values = new int[n];
            for(int j = 0; j < n; j++)
            {
                cin >> values[j];
            }
            arr.insert(index, n, values);
            delete[] values;
        }
        else if(c == "erase")
        {
            int index, n;
            cin >> index >> n;
            arr.erase(index, n);

        }
        else if (c == "size")
        {
            cout << arr.get_size() << endl;
        }

        else if(c == "get")
        {
            int index;
            cin >> index;
            cout << arr[index] << endl;

        }
        else if(c == "set")
        {
            int index, value;
            cin >> index >> value;
            arr[index] = value;

        }
        else if (c == "print")
        {
            cout << arr << endl;
        }
    }
    return 0;
}
```

## Реалізація через структуру:

C: > cpp > algotester_lab_7_8_variant_3_solomia_liashchuk.cpp > main()

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

template<typename S>
struct DynamicArray
{
    S* data;
    int size;
    int capacity;

    void createSize(int newCapacity)
    {
        S* newData = new S[newCapacity];
        for(int i = 0; i < size; i++)
        {
            newData[i] = data[i];
        }
        delete[] data;
        data = newData;
        capacity = newCapacity;
    }

    DynamicArray() : size(0), capacity(1)
    {
        data = new S[capacity];
    }

    ~DynamicArray()
    {
        delete[] data;
    }

    void insert(int index, int n, S* values)
    {
        if(size + n > capacity)
        {
            int new_capacity = capacity;
            while(size + n > new_capacity)
            {
                new_capacity *= 2;
            }
            createSize(new_capacity);
        }

        for(int i = size - 1; i >= index; i--)
        {
            data[i + n] = data[i];
        }

        for (int i = 0; i < n; i++)
        {
            data[index + i] = values[i];
        }

        size += n;
    }

    void erase(int index, int n)
    {
        for (int i = index + n; i < size; i++)
        {
            data[i - n] = data[i];
        }
        size -= n;
    }
```

```
    int get_size() const
    {
        return size;
    }

    int get_capacity() const
    {
        return capacity;
    }

    S& operator[](int index)
    {
        if (index < 0 || index >= size)
        {
            throw out_of_range("ERROR");
        }
        return data[index];
    }

    friend ostream& operator<<(ostream& os, const DynamicArray& arr)
    {
        for (int i = 0; i < arr.size; i++)
        {
            os << arr.data[i] << (i < arr.size - 1 ? " " : "");
        }
        return os;
    }
};
```

```
int main()
{
    DynamicArray<int> arr;
    int Q;
    cin >> Q;

    for(int i = 0; i < Q; i++)
    {
        string c;
        cin >> c;

        if(c == "insert")
        {
            int index, n;
            cin >> index >> n;
            int* values = new int[n];
            for(int j = 0; j < n; j++)
            {
                cin >> values[j];
            }
            arr.insert(index, n, values);
            delete[] values;
        }
        else if(c == "erase")
        {
            int index, n;
            cin >> index >> n;
            arr.erase(index, n);
        }
        else if(c == "size")
        {
            cout << arr.get_size() << endl;
        }
        else if(c == "capacity")
        {
```

```
        else if(c == "capacity")
        {
            cout << arr.get_capacity() << endl;
        }
        else if(c == "get")
        {
            int index;
            cin >> index;
            cout << arr[index] << endl;
        }
        else if(c == "set")
        {
            int index, value;
            cin >> index >> value;
            arr[index] = value;
        }
        else if(c == "print")
        {
            cout << arr << endl;
        }
    }

    return 0;
}

```

## Task 6

### Class Practice Task(робота з списками1-3)

C: > cpp > practice_work_team_tasks_solomia_liashchuk.cpp > main()

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

struct Node
{
    int value;
    Node* next;
};

Node* head = nullptr;
Node* head2 = nullptr;
void Show(Node* head)
{
    if(head == nullptr)
    {
        cout << "list is emty" << endl;
    }

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

};

```

```
27
28  Node* reverse(Node* head)
29  {
30      Node* next = nullptr;
31      Node* current = head;
32      Node* prev = nullptr;
33      while (current != nullptr)
34      {
35          next = current->next;
36          current->next = prev;
37          prev = current;
38          current = next;
39      }
40
41      return prev;
42  }
43
44  bool compare(Node *h1, Node *h2)
45  {
46      while(h1 != nullptr && h2 != nullptr)
47      {
48          if(h1->value != h2->value)
49          {
50              cout << "Not compare" << endl;
51              return false;
52          }
53
54          h1 = h1->next;
55          h2 = h2->next;
56      }
57
58      return h1 == nullptr && h2 == nullptr;
59  }
60
```

```
61  Node* add(Node *n1, Node *n2)
62  {
63      Node* resultHead = nullptr;
64      Node* resultTail = nullptr;
65      int carry = 0;
66
67      while(n1 != nullptr || n2 != nullptr || carry != 0)
68      {
69          int digit1 = (n1 != nullptr) ? n1->value : 0;
70          int digit2 = (n2 != nullptr) ? n2->value : 0;
71
72          int sum = digit1 + digit2 + carry;
73          carry = sum / 10;
74          int currentDigit = sum % 10;
75
76          Node* newNode = new Node{currentDigit, nullptr};
77          if(resultHead == nullptr)
78          {
79              resultHead = newNode;
80              resultTail = resultHead;
81          }
82          else
83          {
84              resultTail->next = newNode;
85              resultTail = newNode;
86          }
87
88          if(n1) n1 = n1->next;
89          if(n2) n2 = n2->next;
90      }
91      return resultHead;
92  }
```

```cpp
int main()
{
    Node* head = new Node{3, nullptr};
    head->next = new Node{8, nullptr};
    head->next->next = new Node{1, nullptr};

    cout << "List one: " << endl;
    Show(head);


    Node* head2 = new Node{4, nullptr};
    head2->next = new Node{6, nullptr};
    head2->next->next = new Node{9, nullptr};

    cout << "List two: " << endl;
    Show(head2);

    head = reverse(head);
    cout << "Reversede list" << endl;
    Show(head);

    if(compare(head, head2))
    {
        cout << "Lists are equal" << endl;
    }
    else
    {
        cout << "Lists are not equal" << endl;
    }

    Node* result = add(head, head2);

    cout << "Result of addition: ";
    Show(result);
```

```cpp
    while(head != nullptr)
    {
        Node* current = head;
        head = head->next;
        delete current;
    }

    while(head2 != nullptr)
    {
        Node* temp2 = head2;
        head2 = head2->next;
        delete temp2;
    }

    while(result != nullptr)
    {
        Node* temp = result;
        result = result->next;
        delete temp;
    }

    return 0;

}
```

## Class Practice Task(робота з деревами4-5)

C: > cpp > practice_work_team_tasks_2_solomia_liashchuk.cpp > main()

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

struct TreeNode
{
    int val;
    TreeNode *left;
    TreeNode *right;

    TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
};

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

    TreeNode* mirroredRoot = new TreeNode(root->val);

    mirroredRoot->left = create_mirror_flip(root->right);
    mirroredRoot->right = create_mirror_flip(root->left);

    return mirroredRoot;
}

```

```cpp

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

int main()
{
    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);

    cout << "Оригінальне дерево: ";
    print_tree(root);
    cout << std::endl;

    TreeNode* mirroredRoot = create_mirror_flip(root);

    cout << "Віддзеркалене дерево: ";
    print_tree(mirroredRoot);
    cout << std::endl;

    return 0;
}

```

# Task 7

## Self Practice Task

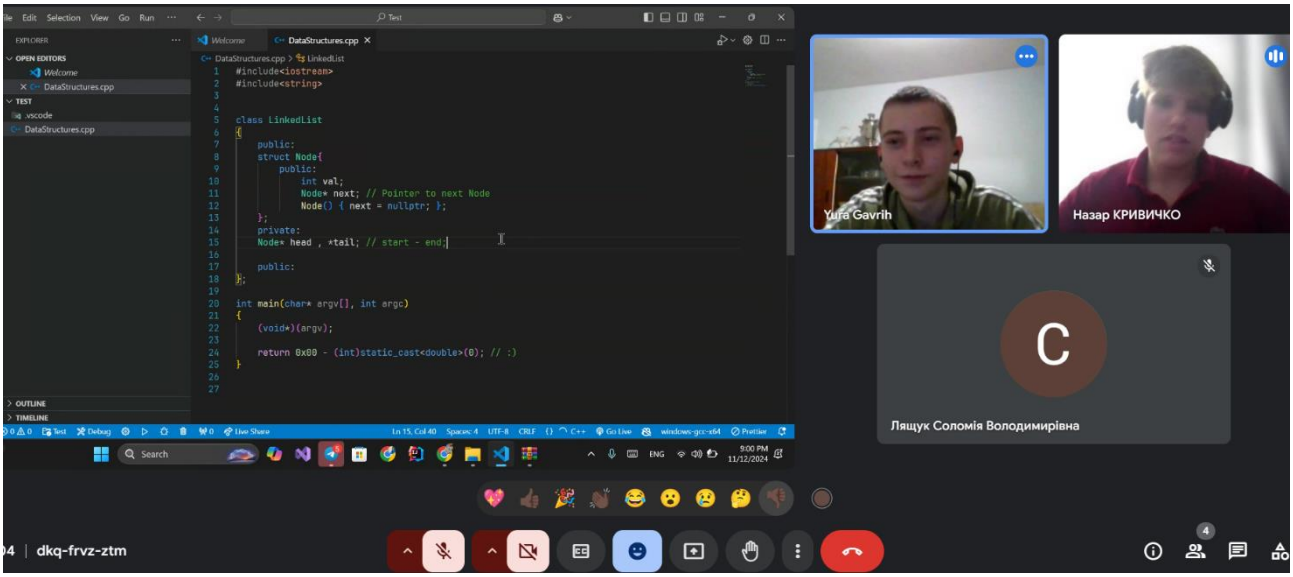
C: > cpp > practice_work_self_algotester_tasks_solomia_liashchuk.cpp.cpp > ...

```cpp
#include <iostream>

struct TreeNode
{
    int data;
    TreeNode* left;
    TreeNode* right;

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

struct BinarySearchTree
{
    TreeNode* root;
    BinarySearchTree() : root(nullptr) {}

    TreeNode* insert(TreeNode* node, int value)
    {
        if(node == nullptr)
        {
            return new TreeNode(value);
        }

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

```

```
31
32          return node;
33      }
34
35      void add(int value) {
36          root = insert(root, value);
37      }
38
39      void traverse(TreeNode* node)
40      {
41          if(node == nullptr)
42          {
43              return;
44          }
45          std::cout << node->data << " ";
46          traverse(node->left);
47          traverse(node->right);
48      }
49
50      void display()
51      {
52          traverse(root);
53          std::cout << std::endl;
54      }
55  };
56
```

```
56
57  int main()
58  {
59      BinarySearchTree bst;
60      bst.add(10);
61      bst.add(5);
62      bst.add(15);
63      bst.add(3);
64      bst.add(7);
65      std::cout << "Дерево: ";
66      bst.display();
67
68      return 0;
69  }
70
```

**Робота в команді:**




**Висновок:** цей епік був найскладніший, тому що великий об’єм нового матеріалу. Я дізналася багато нового й одразу попрактикувалася в цих темах, відкрила для себе нову нішу в програмуванні. Дуже цікаво було працювати з списками та деревами.