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 Лабораторної Роботи № 5
Алготестер Лабораторної Роботи № 7-8
Практичних Робіт до блоку № 6

Виконав:
Студент групи ШІ-12
Тимчук Дмитро Сергійович

Львів 2024

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

Мета роботи: навчитись працювати з динамічними структурами, спробувати написати власні алгоритми для таких структур як: черга, стек, список та дерево.

Теоретичні відомості:

1) Структури
2) Класи
3) Список
4) Подвійний список
5) Бінарне дерево

Індивідуальний план опрацювання теорії:

- Тема №1 Структури (50 хв) (https://www.youtube.com/watch?v=999IE-6b7_s)
- Тема №2 Класи (50 хв) (https://www.youtube.com/watchv=ZbsukxxV5_Q&list=PLiPRE8VmJzOpn6PzYf0higmCEyGzo2A5g&index=95)
- Тема №3 Список (70 хв) (https://www.youtube.com/watchv=-25REjF_atI&list=PLiPRE8VmJzOpn6PzYf0higmCEyGzo2A5g&index=141)
- Тема №4 Двозв’язний список (40 хв) (https://www.youtube.com/watch?v=QLzu2_QFoE&list=PLiPRE8VmJzOpn6PzYf0higmCEyGzo2A5g)
- Тема №5 Бінарне дерево (50 хв) (https://www.youtube.com/watch?v=qBFzNW0ALxQ&list=PLiPRE8VmJzOpn6PzYf0higmCEyGzo2A5g)

Виконання роботи

Завдання №1 Epic 6 Task 3 - Lab# programming: VNS Lab 10

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

struct Node
{
    char* data;
    Node* prev;
    Node* next;
};

class double_linked_list
{
private:

    Node* head;
    Node* tail;

public:
    double_linked_list ()
    {
       head = nullptr;
       tail = nullptr;
    }

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

    //Додавання нового елемента в кінець
    void push_back (const char* element)
    {
        Node* new_node = new Node();
        new_node->data = new char [strlen(element) + 1];
        strcpy(new_node->data, element);

        new_node->prev = tail;
        new_node->next = nullptr;

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

    //Виведення списку
    void print_list()
    {
        if (head == nullptr) cout<<"List is empty";
        else
        {
            Node* current = head;
            while (current != nullptr)
            {
               cout<<current->data<<" ";
               current = current->next;
            }
        }

        cout<<endl;
    }

    //Видалення однакових елементів
    void delete_identical_elements()
    {
        Node* i = head;

        while (i != nullptr)
        {
            Node* current = i->next;
            while (current != nullptr)
            {
                if (!strcmp(i->data, current->data))
                {
                    if (current == tail)
                    {
                        Node* temp = current;
                        tail = tail->prev;
                        tail->next = nullptr;
                        current = current->next;
                        delete temp;
                    }
                    else
                    {
                        Node* temp = current;

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

```
                        current = current->next;
                        delete temp;
                    }
                }
                else current = current->next;
            }
            i = i->next;
        }
    }

    //Додавання елемента після заданого
    void add_element (const char* index_element, const char* add_element)
    {
        Node* current = head;
        while (current != nullptr)
        {
            if (!strcmp(current->data, index_element))
            {
                Node* new_node = new Node();
                new_node->data = new char [strlen(add_element) + 1];
                strcpy(new_node->data, add_element);

                if (current == tail)
                {
                    new_node->prev = current;
                    new_node->next = nullptr;

                    current->next = new_node;
                    tail = new_node;

                    return;
                }
                else
                {
                    new_node->prev = current;
                    new_node->next = current->next;

                    current->next = new_node;
                    new_node->next->prev = new_node;

                    return;
                }
            }

            current = current->next;
        }
    }

    //Копіювання списку в файл
    void add_list_to_file(const char* file_name)
    {
        ofstream fout;
        fout.open(file_name);

        Node* current = head;

        while (current != nullptr)
        {
            fout<<current->data<<endl;
            current = current->next;
        }

        fout.close();
    }

    //Знищення списку
    void delete_list()
    {
        while (head != nullptr)
        {
            Node* temp = head;
            head = head->next;
            delete temp;
        }

        tail = nullptr;
    }

    //Копіювання списку з файлу
    void copy_list_from_file(const char* file_name)
    {
        ifstream fin;
        fin.open(file_name);


        char new_element[256];
        while (fin>>new_element)
        {
            push_back(new_element);
        }

        fin.close();
    }
};



int main()
{
    const char file_name[] = "List.txt";
    double_linked_list list;
    list.print_list();
```

```
    list.push_back("Dima");//Додавання елемента в кінець
    list.push_back("Mark");
    list.push_back("Dima");
    list.push_back("Sonia");
    list.push_back("Sonia");

    cout<<"Initial list: ";
    list.print_list();

    list.delete_identical_elements();//Видалення однакових елементів
    cout<<"List after deleting: ";
    list.print_list();

    list.add_element("Mark", "Nazar");//додавання елемента після заданого
    list.add_element("Sonia", "Taras");//додавання елемента після заданого
    cout<<"List after adding: ";
    list.print_list();

    list.add_list_to_file(file_name);  //запис у файл

    list.delete_list();//Знищення списку

    list.print_list();

    list.copy_list_from_file(file_name);//Відновлення списку з файлу
    cout<<"Restored list: ";
    list.print_list();

    list.delete_list();//Знищення списку
    return 0;
}
```

```
List is empty
Initial list: Dima Mark Dima Sonia Sonia
List after deleting: Dima Mark Sonia
List after adding: Dima Mark Nazar Sonia Taras
List is empty
Restored list: Dima Mark Nazar Sonia Taras
PS C:\Users\dimat>
```

Завдання №2 Epic 6 Task 4 - Lab# programming: Algotester Lab 5

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

//Розкладання початкового числа в матрицю
void make_board (uint64_t starting_board, bool* matrix)
{
    int count = 0;

    while (starting_board > 0)
    {
        while (starting_board  % 2 == 0)
        {
            starting_board /= 2;
            count++;
        }

        *(matrix + count) = true;
        starting_board--;
    }

}

//формування числа з матриці
uint64_t making_final_board(bool* matrix)
{
    int count = 0;
    uint64_t final_board = 0;
    while (count < 64)
    {
        if (*(matrix + count) == true)
        {
            uint64_t a = pow(2, count);
            final_board += a;
        }

        count++;
    }

    return final_board;
}

int main()
{
    uint64_t  starting_board;

    cin>>starting_board;
    bool board[8][8]{};
    make_board(starting_board, &board[0][0]);

    int rows[8]{}, columns[8]{};
    int n, row, column;

    cin>>n;
    for (int i = 0; i < n; i++)
    {
        cin>>row>>column;

        board[row - 1][column - 1] = !board[row - 1][column - 1];

        for (int j = 0; j < 8; j++)
        {
            board[row - 1][j] = !board[row - 1][j];
            board[j][column - 1] = !board[j][column - 1];
        }

    }

    uint64_t final_board;
    final_board = making_final_board(&board[0][0]);
    cout<<final_board<<endl;

    return 0;
}
```

```
0
4
1 1
1 2
2 2
2 1
771
PS C:\Users\dimat>
```

Завдання №3 Epic 6 Task 5 - Lab# programming: Algotester Lab 7-8

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

template <typename T>
class binary_tree
{
private:
    struct tree_node
    {
        T data;
        tree_node* left;
        tree_node* right;

        tree_node (T value): data(value), left(nullptr), right(nullptr){};
    };

    tree_node* root;
    int size = 0;

    //Виведення дерева
    void print_tree(tree_node* node, ostream& os) const
    {
        if (node != nullptr)
        {
            print_tree(node->left, os);

            os << node->data << " ";

            print_tree(node->right, os);
        }
    }

    //Вставити число
    tree_node* insert(tree_node* node, T value)
    {
        if (node == nullptr)
        {
            size++;
            return new tree_node(value);
        }

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

        if (value < node->data)
        {
            node->left = insert(node->left, value);
        }
        return node;
    }
    //знайти число
    bool contains (tree_node* node, T value)
    {
        if (node == nullptr)
        {
            return false;
        }

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

        return true;
    }

public:
    binary_tree(): root(nullptr), size(0){};

    void insert(T value)
    {
        root = insert(root, value);
    }

    bool contains(T value)
    {
        return contains(root, value);
    }

    int size_of_tree()
    {
        return size;
    }

    friend ostream& operator<<(ostream& os, const  binary_tree& tree)
    {
        tree.print_tree(tree.root, os);
        return os;
    }


};
```

```cpp
int main()
{
    binary_tree<int> my_tree;

    int n;
    string s;
    cin>>n;

    for (int i = 0; i < n; i++)
    {
        cin>>s;
        if (s == "size")
        {
            cout<<my_tree.size_of_tree()<<endl;
        }
        else if (s == "print")
        {
            cout<<my_tree<<endl;
        }
        else if (s =="insert")
        {
            int value;
            cin>>value;
            my_tree.insert(value);
        }
        else if (s == "contains")
        {
            int value;
            cin>>value;
            if (my_tree.contains(value)) cout<<"Yes"<<endl;
            else cout<<"No"<<endl;
        }
    }

    return 0;
}
```

```
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
PS C:\Users\dimat>
```

Завдання №4 Epic 6 Task 6 - Practice# programming: Class Practice Task

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

//Linked list
struct Node
{
    int data;
    Node* next;
};

//Додавання елемента в кінець
Node* push_back(Node* head, const int& element)
{
    Node* new_node = new Node;
    new_node->data = element;
    new_node->next = nullptr;

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

        current->next = new_node;
    }

    return head;
}
```

```cpp
//Виведення списку
void print_list(Node* head)
{
    if (head == nullptr)
    {
        cout<<"List is empty"<<endl;
    }
    else
    {
        Node* current = head;
        while (current != nullptr)
        {
            cout<<current->data<<" ";
            current = current->next;
        }
        cout<<endl;
    }
}

//Реверс списку
Node* reverse_list(Node* head)
{
    if (head == nullptr)
    {
        cout<<"List is empty"<<endl;
        return 0;
    }
    else
    {
        Node* prev = nullptr;
        Node* current = head;
        Node* next = nullptr;

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

        return prev;
    }
}
```

```cpp
//Порівняння на рівність двох списків
bool compare(Node* head_1, Node* head_2)
{
    Node* current_1 = head_1;
    Node* current_2 = head_2;

    while ((current_1 != nullptr) && (current_2 != nullptr))
    {
        if (current_1->data != current_2->data)
        {
            return false;
        }
        current_1 = current_1->next;
        current_2 = current_2->next;
    }

    if ((current_1 != nullptr) || (current_2 != nullptr))
    {
        return false;
    }
    else return true;
}

//Додавання двох великих чисел
Node* add_two_numbers(Node* n1, Node* n2)
{
    Node* current_1 = n1;
    Node* current_2 = n2;
    Node* sum = nullptr;
    int r = 0;
    int s = 0;
    while (current_2 != nullptr)
    {
        s = current_1->data + current_2->data + r;

        if (s > 9)
        {
            sum = push_back(sum, s % 10);
            r = s / 10;
        }
        else
        {
            sum = push_back(sum, s);
            r = 0;
        }

        current_1 = current_1->next;
        current_2 = current_2->next;
    }

    if (current_1 != nullptr)
    {
        while (current_1 != nullptr)
        {
            s = current_1->data + r;

            if (s > 9)
            {
                sum = push_back(sum, s % 10);
                r = s / 10;
            }
            else
            {
                sum = push_back(sum, s);
                r = 0;
            }

            current_1 = current_1->next;
        }
    }
    else if (r != 0)
    {
        sum = push_back(sum, r);
    }

    return sum;
}

//Виведення числа
void print_number(Node* head)
{
    if (head == nullptr)
    {
        cout<<"List is empty"<<endl;
    }
    else
    {
        Node* current = head;
        while (current != nullptr)
        {
            cout<<current->data;
            current = current->next;
        }
        cout<<endl;
    }
}
//Linked list

```

```cpp
//Binary tree
struct tree_node
{
    int data;
    tree_node* left;
    tree_node* right;

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



//Функція для вставлення елемента в дерево
tree_node* insert(tree_node* node, int value)
{
    if (node == nullptr)
    {
        return new tree_node(value);
    }
    else
    {
        if (value < node->data)
        {
            node->left = insert(node->left, value);
        }
        else if (value > node->data)
        {
            node->right = insert(node->right, value);
        }

        return node;
    }

}

//Віддзеркалення дерева
tree_node* create_mirror_flip(tree_node* node)
{

    if (node == nullptr)
    {
        return nullptr;
    }

    tree_node* new_node = new tree_node(node->data);

    new_node->right = create_mirror_flip(node->left);
    new_node->left = create_mirror_flip(node->right);

    return new_node;
}
```

```cpp
//Сума підвузлів
tree_node* tree_sum (tree_node* node)
{

    if ((node == nullptr) || ((node->left == nullptr) && (node->right == nullptr))) return nullptr;

    tree_sum(node->left);
    tree_sum(node->right);

    node->data = 0;
    if (node->right != nullptr)
    {
       node->data += node->right->data;
    }
    if (node->left != nullptr)
    {
       node->data += node->left->data;
    }

    return node;
}

//Виведення дерева
void print_tree(tree_node* node)
{
    if (node == nullptr)
    {
        return;
    }

    cout<<node->data<<" ";
    print_tree(node->left);
    print_tree(node->right);

}
//Binary tree
```

```cpp
int main()
{
    //task_1
    Node* head = nullptr;

    for (int i = 0; i < 10; i++)
    {
        head = push_back(head, i);
    }

    cout<<"Starting list: ";
    print_list(head);

    Node* new_head = reverse_list(head);
    cout<<"Reversed list: ";
    print_list(new_head);

    //task 2
    Node* head_1 = nullptr;
    Node* head_2 = nullptr;

    head_1 = push_back(head_1, 5);
    head_1 = push_back(head_1, 6);
    head_1 = push_back(head_1, 5);
    head_1 = push_back(head_1, 7);
    head_1 = push_back(head_1, 8);

    head_2 = push_back(head_2, 5);
    head_2 = push_back(head_2, 6);
    head_2 = push_back(head_2, 4);

    if (compare(head_1, head_2))
    {
        cout<<"Lists are equal"<<endl;
    }
    else cout<<"Lists aren`t equal"<<endl;

    //task 3
    string num_1, num_2, box;
    cout<<"Enter first number: ";
    cin>>num_1;
    cout<<"Enter second number: ";
    cin>>num_2;

    if (num_2.length() > num_1.length())
    {
        box = num_1;
        num_1 = num_2;
        num_2 = box;
    }
```

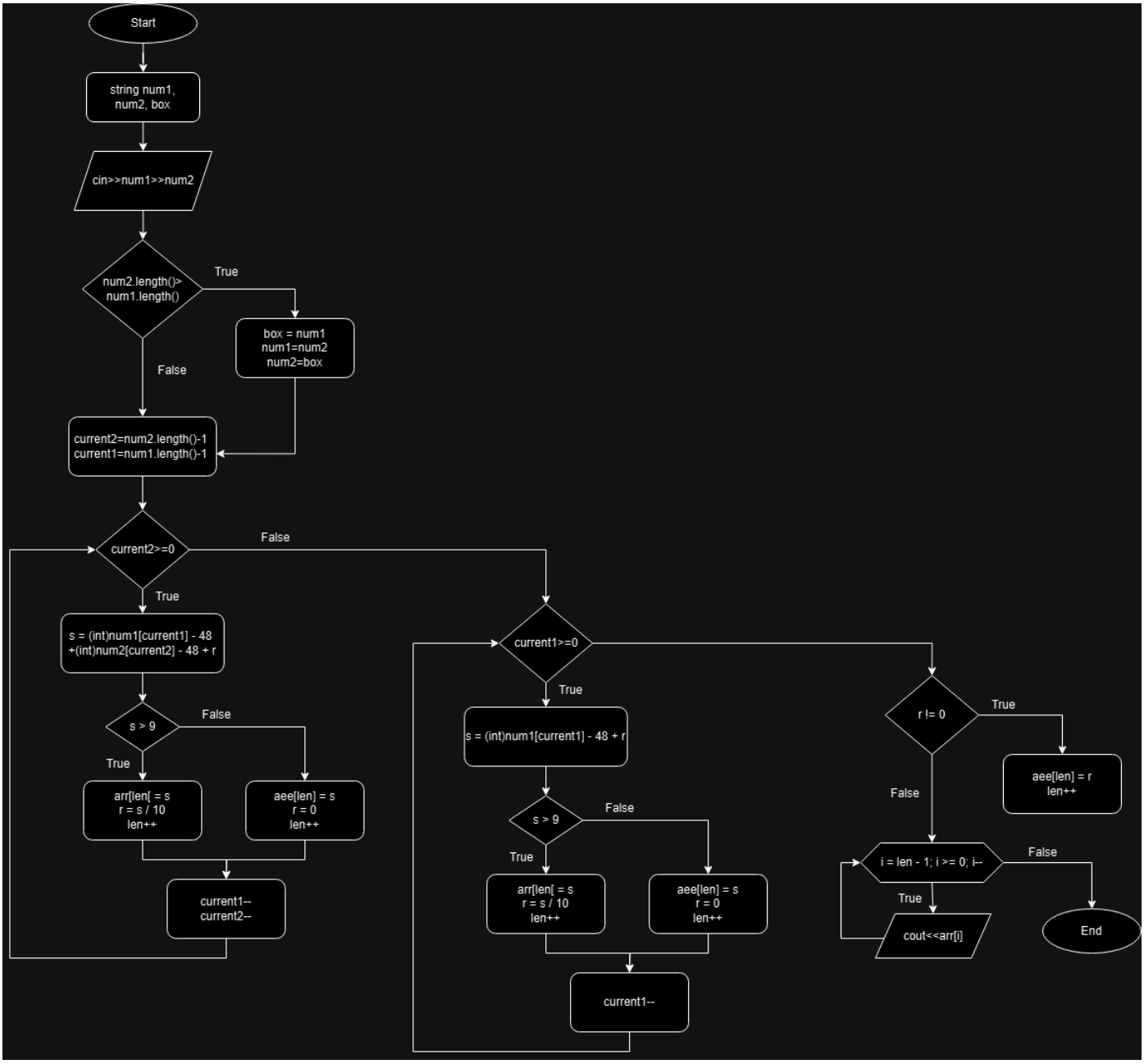
```cpp
    Node* n1 = nullptr;
    Node* n2 = nullptr;

    for (int i = num_1.length() - 1; i >= 0; i--)
    {
        n1 = push_back(n1, (int)num_1[i] - 48);
    }
    for (int i = num_2.length() - 1; i >= 0; i--)
    {
        n2 = push_back(n2, (int)num_2[i] - 48);
    }

    Node* sum;
    sum = add_two_numbers(n1, n2);

    Node* new_sum = reverse_list(sum);
    cout<<num_1<<" + "<<num_2<<" = ";
    print_number(new_sum);

    //task 4
    tree_node* root = nullptr;

    root = insert(root, 5);
    root = insert(root, 3);
    root = insert(root, 6);
    root = insert(root, 2);
    root = insert(root, 10);


    tree_node* new_root;
    new_root = create_mirror_flip(root);

    cout<<"First tree: ";
    print_tree(root);

    cout<<"\nMirrored tree: ";
    print_tree(new_root);

    root = tree_sum(root);
    cout<<"\nSum tree: ";
    print_tree(root);

    return 0;
}
```

```
Starting list: 0 1 2 3 4 5 6 7 8 9
Reversed list: 9 8 7 6 5 4 3 2 1 0
Lists aren`t equal
Enter first number: 11231331301303133013
Enter second number: 13021301301321030131313313
13021301301321030131313313 + 11231331301303133013 = 13021301413634343144344446326
First tree: 5 3 2 6 10
Mirrored tree: 5 6 10 3 2
Sum tree: 12 2 2 10 10
PS C:\Users\dimat>
```

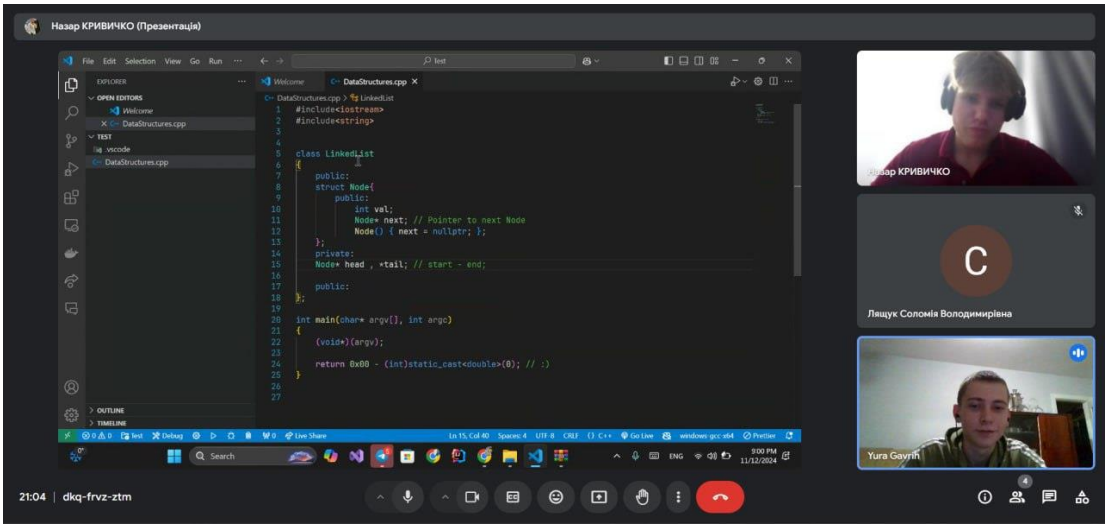
Завдання №5 Epic 6 Task 7 - Practice# programming: Self Practice Task

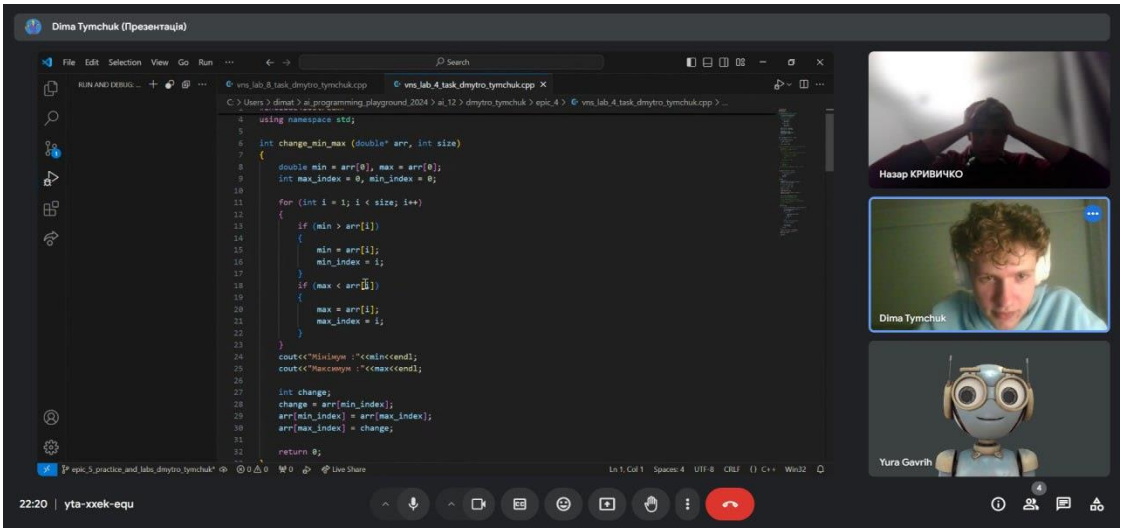
```cpp
#include<iostream>
#include<string>
using namespace std;

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

int main()
{
    string num1, num2, box;
    cin>>num1>>num2;
    int arr[1000]{};

    if (num1.length() < num2.length())
    {
        box = num1;
        num1 = num2;
        num2= box;
    }

    int current2 = num2.length() - 1;
    int current1 = num1.length() - 1;
    int s = 0, r = 0;
    int len = 0;
    while (current2 >= 0)
    {
        s = (int)num1[current1] - 48 + (int)num2[current2] - 48 + r;

        if (s > 9)
        {
            arr[len] = s % 10;
            r = s / 10;
            len++;
        }
        else
        {
            arr[len] = s;
            r = 0;
            len++;
        }
        current2--;
        current1--;
    }

    if (current1 >= 0)
    {
        while (current1 >= 0 )
        {
            s = (int)num1[current1] - 48 + r;

            if (s > 9)
            {
                arr[len] = s % 10;
                r = s / 10;
                len++;
            }
            else
            {
                arr[len] = s;
                r = 0;
                len++;
            }

            current1--;
        }
    }
    else if (r != 0)
    {
        arr[len] = r;
        len++;
    }

    for (int i = len - 1; i >= 0; i--)
    {
        cout<<arr[i];
    }

    return 0;
}
```

```
terpreter=m1
700000000000001 600000000000001
1300000000000002
PS C:\Users\dimat>
```


Start
string num1, num2, box
cin>>num1>>num2
num2.length()> num1.length()
True
box = num1
num1=num2
num2=box
False
current2=num2.length()-1
current1=num1.length()-1
current2>=0
False
True
s = (int)num1[current1] - 48 +(int)num2[current2] - 48 + r
s > 9
False
True
arr[len[= s
r = s / 10
len++
aee[len] = s
r = 0
len++
current1--
current2--
current1>=0
True
s = (int)num1[current1] - 48 + r
s > 9
False
True
arr[len[= s
r = s / 10
len++
aee[len] = s
r = 0
len++
current1--
r != 0
True
aee[len] = r
len++
False
i = len - 1; i >= 0; i--
False
True
cout<<arr[i]
End

Робота в команді





Висновок: під час виконання лабораторної роботи я навчився краще працювати і розуміти динамічні структури. Навчився писати власні алгоритми для роботи з такими структурами.