Міністерство освіти і науки України

Національний університет «Львівська політехніка»

Кафедра систем штучного інтелекту



**Звіт**

про виконання

**Лабораторних та практичних робіт № 10**

***з дисципліни:*** «Мови та парадигми програмування»

***з розділу***: «Динамічні структури (Черга, Стек, Списки, Дерево). Алгоритми обробки динамічних структур»

***Виконала:***

студентка групи ШІ-12

Борщ Анастасія Ігорівна

# **Тема роботи:**

Знання динамічних структур даних на мові програмування C++ і маніпулювати ними. Принцип роботи з бінарними деревами. Ознайомлення з поняттями стеків, черг і пов'язаних списків. Ознайомлення з основними алгоритмами обробки динамічних структур.

# **Мета роботи:**

Дізнатися , як керувати динамічними структурами даних у C++. Вивчити базові алгоритми обробки динамічних структур (черги, стеки).Застосувати отримані знання на практиці.

# **Теоретичні відомості:**

1. Теоретичні відомості з переліком важливих тем:

* Тема №1: Черги.
* Тема №2: Стеки.
* Тема №3: Однозв’язний список в C++.
* Тема №4: Двозв’язний список в C++.
* Тема №5: Бінарні дерева.

1. Індивідуальний план опрацювання теорії:

* Тема №1: Черги.
  + Джерела Інформації
    - Стаття : <https://uk.myservername.com/queue-data-structure-c-with-illustration>
    - Відео: [C++ Теорія ⦁ Урок 142 ⦁ ADT ⦁ Черга](https://www.youtube.com/watch?v=Yhw8NbjrSFA&ab_channel=%D0%91%D0%BB%D0%BE%D0%B3%D0%B0%D0%BD%E2%A6%81%D0%A3%D1%80%D0%BE%D0%BA%D0%B8%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F)
  + Що опрацьовано:
    - Опрацьовано теорію про роботу черги в мові C++
  + Статус: Ознайомлена
  + Початок опрацювання теми: 13.12.2023
  + Звершення опрацювання теми: 13.11.2023
* Тема №2: Стеки.
  + Джерела Інформації
    - Стаття: <https://acode.com.ua/urok-111-stek-i-kupa>
  + Що опрацьовано:
    - Опрацьовано теорію про роботу стеку в мові C++
  + Статус: Ознайомлена
  + Початок опрацювання теми: 13.12.2023
  + Звершення опрацювання теми: 13.12.2023
* Тема №3: Однозв’язний список в C++.
  + Джерела Інформації:
    - Відео: <https://www.youtube.com/watch?v=-25REjF_atI>
  + Що опрацьовано:
    - Опрацьовано поняття однозв’язного списку в мові C++
  + Статус: Ознайомлена
  + Початок опрацювання теми: 12.12.2023
  + Звершення опрацювання теми: 12.12.2023
* Тема №4: Двозв’язний список в C++.
  + Джерела Інформації:
    - Відео: <https://www.youtube.com/watch?v=QLzu2-_QFoE>
  + Що опрацьовано:
    - Опрацьовано основні методи роботи з ввонаправленим списком в мові програмування C++
  + Статус: Ознайомлена
  + Початок опрацювання теми: 12.12.2023
  + Звершення опрацювання теми: 12.12.2023
* Тема №5: Бінарні дерева
  + Джерела Інформації:
    - Відео: <https://www.youtube.com/watch?v=qBFzNW0ALxQ>
  + Що опрацьовано:
    - Опрацьовано основні методи роботи з бінарними деревами в мові програмування C++
  + Статус: Ознайомлена
  + Початок опрацювання теми: 14.12.2023
  + Звершення опрацювання теми: 14.12.2023

# **Виконання роботи:**

## **1. Опрацювання завдання та вимог до програм та середовища:**

Завдання №1 VNS Lab 10

* Варіант завдання: 18
* Деталі завдання :

1. Написати функцію для створення списку. Функція може створювати порожній список, а потім додавати в нього елементи.

2. Написати функцію для друку списку. Функція повинна передбачати вивід повідомлення, якщо список порожній.

3. Написати функції для знищення й додавання елементів списку у відповідності зі своїм варіантом.

4. Виконати зміни в списку й друк списку після кожної зміни.

5. Написати функцію для запису списку у файл.

6. Написати функцію для знищення списку.

7. Записати список у файл, знищити його й виконати друк (при друці повинне бути видане повідомлення "Список порожній").

8. Написати функцію для відновлення списку з файлу.

9. Відновити список і роздрукувати його.

10.Знищити список.

Записати в лінійному списку містять ключове поле типу \*char (рядок символів). Сформувати двонаправлений список. Знищити елемент із заданим ключем. Додати К елементів у початок списку.

Завдання №2 Algotester Lab 5

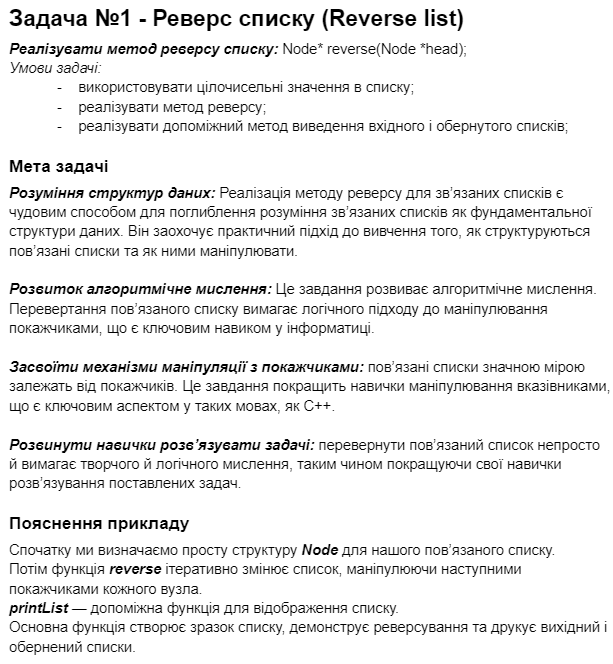
* Варіант завдання: 3
* Деталі завдання: <https://algotester.com/en/ContestProblem/DisplayWithEditor/134644>

Завдання №3 Algotester Lab 7-8

* Варіант завдання: 2
* Деталі завдання: <https://algotester.com/en/ContestProblem/DisplayWithEditor/134648>

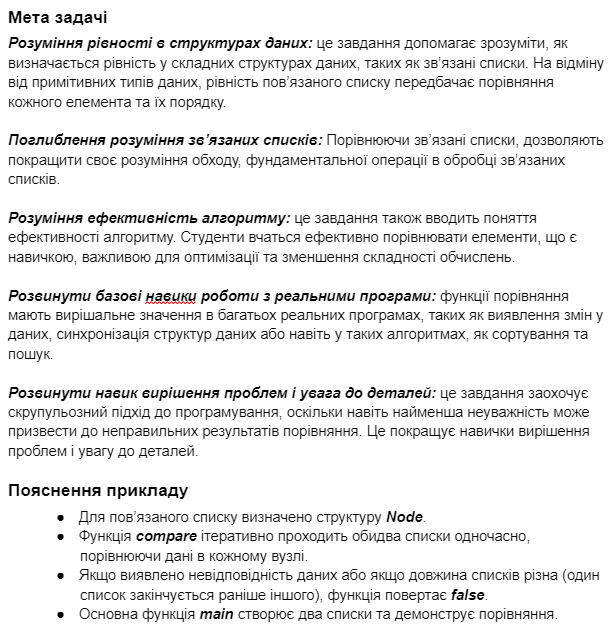
Завдання №4 Class Practice Work Task 1

* Варіант завдання: немає
* Деталі завдання:



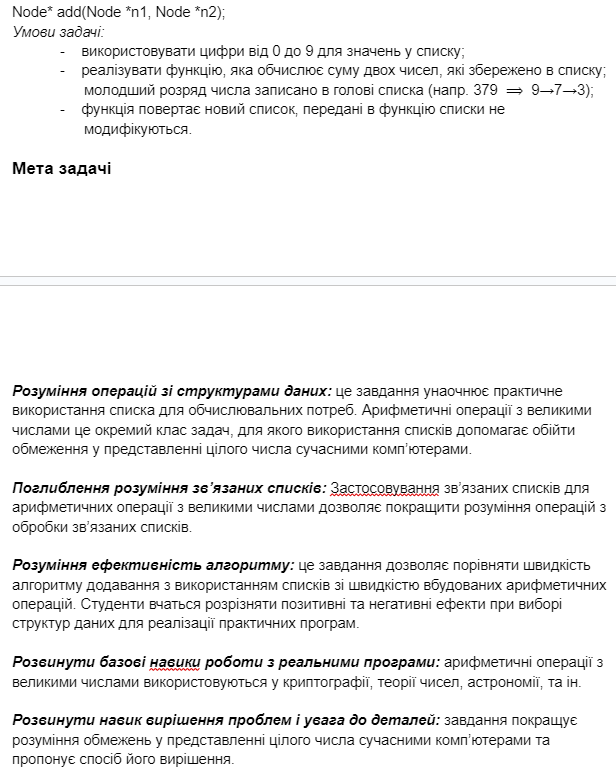
Завдання №5 Class Practice Work Task 2

* Варіант завдання: немає
* Деталі завдання:



Завдання №6 Class Practice Work Task 3

* Варіант завдання: немає
* Деталі завдання:



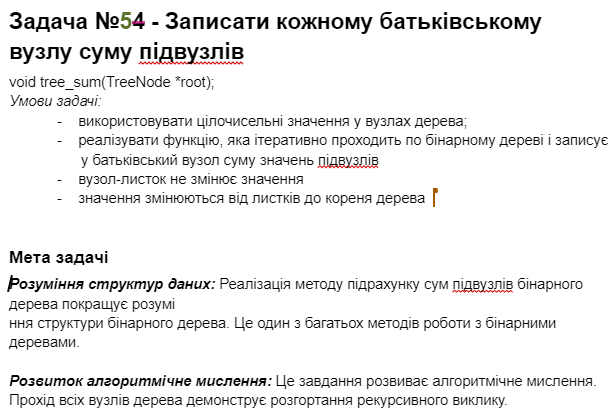
Завдання №7 Class Practice Work Task 4

* Варіант завдання: немає
* Деталі завдання:



Завдання №8 Class Practice Work Task 5

* Варіант завдання: немає
* Деталі завдання:



Завдання №9 Self Practice Work

* Варіант завдання: немає
* Деталі завдання:

<https://algotester.com/uk/ArchiveProblem/DisplayWithEditor/40475#mySolutions>

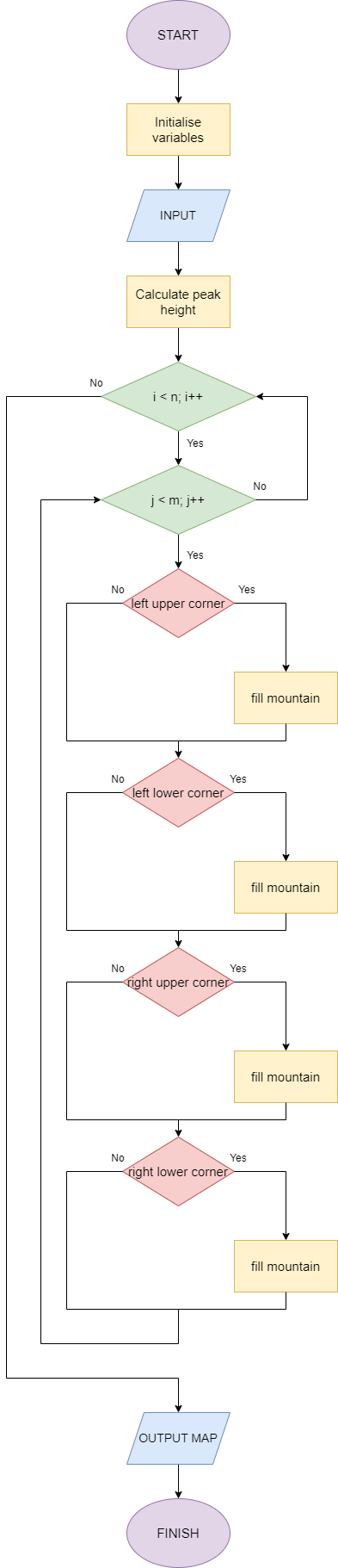
**2. Дизайн та планована оцінка часу виконання завдань:**

Програма №1 VNS Lab 10

* Планований час на реалізацію: 2 години

Програма №2 Algotester Lab 5

* Блок-схема



*Рисунок 1. Блок-схема до програми №2*

* Планований час на реалізацію: 1,5 години

Програма №3 Algotester Lab 7-8

* Планований час на реалізацію: 2 години

Програма №4 Class Practice Work Task 1

* Планований час на реалізацію: 40 хвилин

Програма №5 Class Practice Work Task 2

* Планований час на реалізацію: 40 хвилин

Програма №6 Class Practice Work Task 3

* Планований час на реалізацію: 40 хвилин

Програма №7 Class Practice Work Task 4

* Планований час на реалізацію: 40 хвилин

Програма №8 Class Practice Work Task 5

* Планований час на реалізацію: 40 хвилин

Програма №9 Self Practice Work

* Планований час на реалізацію: 40 хвилин

## **3. Конфігурація середовища до виконання завдань:**

Для виконання поставлених завдань додаткова конфігурація середовища не є необхідною

## **4. Код програм з посиланням на зовнішні ресурси:**

Завдання №1 VNS Lab 10

#include <iostream>

#include <cstring>

struct Node {

    char\* data;

    Node\* prev;

    Node\* next;

};

void add(Node\*& head, Node\*& tail, const char\* value);

void deleteNode(Node\*& head, Node\*& tail, const char\* key);

void print(const Node\* head);

Node\* createList(const char\* values[], int size);

void destroyList(Node\*& head);

int main() {

    Node\* head = nullptr;

    Node\* tail = nullptr;

    const char\* initialValues[] = {"sun", "sky", "sea"};

    int initialSize = sizeof(initialValues) / sizeof(initialValues[0]);

    head = createList(initialValues, initialSize);

    std::cout << "Initial list of elements: ";

    print(head);

    std::cout << "Enter the number of elements to add: ";

    int k;

    std::cin >> k;

    for (int i = 0; i < k; ++i) {

        char value[50];

        std::cout << "Enter the value for element " << i + 1 << ": ";

        std::cin >> value;

        add(head, tail, value);

    }

    std::cout << "List of elements: ";

    print(head);

    char key[50];

    std::cout << "Enter the key to delete an element: ";

    std::cin >> key;

    deleteNode(head, tail, key);

    std::cout << "Updated list of elements after deletion: ";

    print(head);

    destroyList(head);

    return 0;

}

void add(Node\*& head, Node\*& tail, const char\* value) {

    char\* newValue = new char[strlen(value) + 1];

    strcpy(newValue, value);

    Node\* newNode = new Node{newValue, nullptr, nullptr};

    if (!head) {

        head = tail = newNode;

    } else {

        head->prev = newNode;

        newNode->next = head;

        head = newNode;

    }

}

void deleteNode(Node\*& head, Node\*& tail, const char\* key) {

    Node\* current = head;

    while (current) {

        if (strcmp(current->data, key) == 0) {

            if (current->prev)

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

            else

                head = current->next;

            if (current->next)

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

            else

                tail = current->prev;

            delete[] current->data;

            delete current;

            return;

        }

        current = current->next;

    }

    std::cout << "Element with key '" << key << "' not found." << std::endl;

}

void print(const Node\* head) {

    if (!head) {

        std::cout << "The list is empty" << std::endl;

        return;

    }

    const Node\* current = head;

    while (current) {

        std::cout << current->data << " ";

        current = current->next;

    }

    std::cout << std::endl;

}

Node\* createList(const char\* values[], int size) {

    Node\* head = nullptr;

    Node\* tail = nullptr;

    for (int i = 0; i < size; ++i) {

        add(head, tail, values[i]);

    }

    return head;

}

void destroyList(Node\*& head) {

    while (head) {

        Node\* next = head->next;

        delete[] head->data;

        delete head;

        head = next;

    }

}

Завдання №2 Algotester Lab 5

#include <iostream>

#include <cmath>

using namespace std;

int main()

{

    int N, M, x, y;

    cin >> N >> M >> x >> y;

    int map[N][M];

    x -= 1;

    y -= 1;

    int peak\_height = 0;

    for (int i = 0; i < N; ++i)

    {

        for (int j = 0; j < M; ++j)

        {

            int distance = abs(i - x) + abs(j - y);

            map[i][j] = distance;

            if (distance > peak\_height)

            {

                peak\_height = distance;

            }

        }

    }

    for (int i = 0; i < N; ++i)

    {

        for (int j = 0; j < M; ++j)

        {

            cout << peak\_height - map[i][j] << " ";

        }

        cout << endl;

    }

    return 0;

}

Завдання №3 Algotester Lab 7-8

#include <iostream>

using namespace std;

class DynamicArray {

private:

    int \*array;

    int currentSize;

    int currentCapacity;

    void resize(int newCapacity) {

        int \*tempArray = new int[newCapacity];

        for (int i = 0; i < currentSize; ++i) {

            tempArray[i] = array[i];

        }

        delete[] array;

        array = tempArray;

        currentCapacity = newCapacity;

    }

public:

    DynamicArray() : array(nullptr), currentSize(0), currentCapacity(1) {

        array = new int[currentCapacity];

    }

    ~DynamicArray() {

        delete[] array;

    }

    void insert(int index, int N, int \*elements) {

        if (currentSize + N >= currentCapacity) {

            while (currentSize + N >= currentCapacity) {

                currentCapacity \*= 2;

            }

            resize(currentCapacity);

        }

        for (int i = currentSize - 1; i >= index; --i) {

            array[i + N] = array[i];

        }

        for (int i = 0; i < N; ++i) {

            array[index + i] = elements[i];

        }

        currentSize += N;

    }void erase(int index, int n) {

        for (int i = index; i < currentSize - n; ++i) {

            array[i] = array[i + n];

        }

        currentSize -= n;

    }

    int get(int index) {

        return array[index];

    }

    void set(int index, int value) {

        array[index] = value;

    }

    int size() {

        return currentSize;

    }

    int capacity() {

        return currentCapacity;

    }

    // Перегрузка оператора []

    int &operator[](int index) {

        return array[index];

    }

    // Перегрузка оператора <<

    friend ostream &operator<<(ostream &os, const DynamicArray &dArray) {

        for (int i = 0; i < dArray.currentSize; ++i) {

            os << dArray.array[i] << " ";

        }

        return os;

    }

};

int main() {

    int Q;

    cin >> Q;

    DynamicArray dynamicArray;

    for (int i = 0; i < Q; ++i) {

        string query;

        cin >> query;

                if (query == "insert") {

            int index, N;

            cin >> index >> N;

            int \*elements = new int[N];

            for (int j = 0; j < N; ++j) {

                cin >> elements[j];

            }

            dynamicArray.insert(index, N, elements);

            delete[] elements;

        } else if (query == "erase") {

            int index, n;

            cin >> index >> n;

            dynamicArray.erase(index, n);

        } else if (query == "size") {

            cout << dynamicArray.size() << endl;

        } else if (query == "capacity") {

            cout << dynamicArray.capacity() << endl;

        } else if (query == "get") {

            int index;

            cin >> index;

            cout << dynamicArray.get(index) << endl;

        } else if (query == "set") {

            int index, value;

            cin >> index >> value;

            dynamicArray.set(index, value);

        } else if (query == "print") {

            cout << dynamicArray << endl;

        }

    }

    return 0;

}

Завдання №4 Class Practice Work Task 1

#include <iostream>

using namespace std;

struct Node

{

    int data;

    Node\* next;

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

};

Node\* reverse(Node\* head)

{

    Node\* previous = nullptr;

    Node\* next = nullptr;

    Node\* current = head;

    while (current != nullptr)

    {

        next = current->next;

        current->next = previous;

        previous = current;

        current = next;

    }

    return previous;

}

void printList(Node\* head)

{

    while (head != nullptr)

    {

        cout << head->data << " ";

        head = head->next;

    }

    cout << endl;

}

int main()

{

    Node\* head = new Node(6);

    head->next = new Node(7);

    head->next->next = new Node(8);

    head->next->next->next = new Node(9);

    cout << "The list before reversing: ";

    printList(head);

    head = reverse(head);

    cout << "The list after reversing: ";

    printList(head);

    return 0;

}

Завдання №5 Class Practice Work Task 2

#include <iostream>

using namespace std;

struct Node

{

    int data;

    Node\* next;

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

};

bool compare(Node\* h1, Node\* h2)

{

    while (h1 && h2)

    {

        if (h1->data != h2->data)

        {

            return false;

        }

        h1 = h1->next;

        h2 = h2->next;

    }

    return !h1 && !h2;

}

int main()

{

    Node\* head1 = new Node(1);

    head1->next = new Node(2);

    head1->next->next = new Node(3);

    Node\* head2 = new Node(2);

    head2->next = new Node(2);

    head2->next->next = new Node(1);

    if (compare(head1, head2))

    {

        cout << "The lists are identical";

    }

    else

    {

        cout << "The lists are different";

    }

    return 0;

}

Завдання №6 Class Practice Work Task 3

#include <iostream>

using namespace std;

struct Node

{

    int data;

    Node\* next;

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

};

Node\* add(Node\* n1, Node\* n2)

{

    Node\* resultHead = nullptr;

    Node\* resultTail = nullptr;

    int carry = 0;

    while (n1 != nullptr || n2 != nullptr || carry != 0)

    {

        int sum = carry;

        if (n1 != nullptr)

        {

            sum += n1->data;

            n1 = n1->next;

        }

        if (n2 != nullptr)

        {

            sum += n2->data;

            n2 = n2->next;

        }

        carry = sum / 10;

        sum %= 10;

        Node\* newNode = new Node(sum);

        if (resultHead == nullptr)

        {

            resultHead = resultTail = newNode;

        }

        else

        {

            resultTail->next = newNode;

            resultTail = newNode;

        }

    }

    return resultHead;

}

void printNum(Node\* num)

{

    while (num != nullptr)

    {

        cout << num->data;

        num = num->next;

    }

}

int main()

{

    Node\* num1 = new Node(9);

    num1->next = new Node(8);

    num1->next->next = new Node(1);

    Node\* num2 = new Node(5);

    num2->next = new Node(1);

    num2->next->next = new Node(6);

    Node\* result = add(num1, num2);

    cout << "Number 1: ";

    printNum(num1);

    cout << endl;

    cout << "Number 2: ";

    printNum(num2);

    cout << endl;

    cout << "The sum is: ";

    printNum(result);

    cout << endl;

    return 0;

}

Завдання №7 Class Practice Work Task 4

#include <iostream>

#include <queue>

using namespace std;

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\* mirroredNode = new TreeNode(root->data);

    queue<TreeNode\*> originalQueue, mirroredQueue;

    originalQueue.push(root);

    mirroredQueue.push(mirroredNode);

    while (!originalQueue.empty())

    {

        TreeNode\* originalFront = originalQueue.front();

        TreeNode\* mirroredFront = mirroredQueue.front();

        originalQueue.pop();

        mirroredQueue.pop();

        if (originalFront->right)

        {

            mirroredFront->left = new TreeNode(originalFront->right->data);

            originalQueue.push(originalFront->right);

            mirroredQueue.push(mirroredFront->left);

        }

        if (originalFront->left)

        {

            mirroredFront->right = new TreeNode(originalFront->left->data);

            originalQueue.push(originalFront->left);

            mirroredQueue.push(mirroredFront->right);

        }

    }

    return mirroredNode;

}

void correctOutput(TreeNode\* root)

{

    if (root != nullptr)

    {

        correctOutput(root->left);

        cout << root->data << " ";

        correctOutput(root->right);

    }

}

int main()

{

    TreeNode\* root = new TreeNode(1);

    root->left = new TreeNode(11);

    root->left->right = new TreeNode(26);

    root->left->right->left = new TreeNode(32);

    root->right = new TreeNode(82);

    root->right->left = new TreeNode(95);

    cout << "Original binary tree: ";

    correctOutput(root);

    cout << endl;

    TreeNode\* mirroredTree = create\_mirror\_flip(root);

    cout << "Mirrored binary tree: ";

    correctOutput(mirroredTree);

    cout << endl;

    return 0;

}

Завдання №8 Class Practice Work Task 5

#include <iostream>

#include <stack>

#include <queue>

using namespace std;

struct TreeNode

{

    int data;

    TreeNode\* left;

    TreeNode\* right;

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

};

void tree\_sum(TreeNode\* root)

{

    if (root == nullptr)

    {

        return;

    }

    stack<TreeNode\*> stackOfNode;

    TreeNode\* current = root;

    TreeNode\* previous = nullptr;

    while (current != nullptr || !stackOfNode.empty())

    {

        while (current != nullptr)

        {

            stackOfNode.push(current);

            current = current->left;

        }

        current = stackOfNode.top();

        if ((current->left == nullptr && current->right == nullptr) || (previous != nullptr && (previous == current->right || current->right == nullptr)))

        {

            stackOfNode.pop();

            int sum = current->data;

            if (current->left != nullptr)

            {

                sum += current->left->data;

            }

            if (current->right != nullptr)

            {

                sum += current->right->data;

            }

            current->data = sum;

            previous = current;

            current = nullptr;

        }

        else

        {

            current = current->right;

        }

    }

}

void printBinaryTree(TreeNode\* root)

{

    if (root != nullptr)

    {

        queue<TreeNode\*> nodesQueue;

        nodesQueue.push(root);

        while (!nodesQueue.empty())

        {

            int levelSize = nodesQueue.size();

            for (int i = 0; i < levelSize; ++i)

            {

                TreeNode\* current = nodesQueue.front();

                nodesQueue.pop();

                cout << current->data << " ";

                if (current->left != nullptr)

                {

                    nodesQueue.push(current->left);

                }

                if (current->right != nullptr)

                {

                    nodesQueue.push(current->right);

                }

            }

            cout << endl;

        }

    }

}

int main()

{

    TreeNode\* root = new TreeNode(6);

    root->left = new TreeNode(2);

    root->left->left = new TreeNode(1);

    root->left->right = new TreeNode(5);

    root->right = new TreeNode(7);

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

    root->right->right = new TreeNode(1);

    cout << "Original binary tree:\n";

    printBinaryTree(root);

    tree\_sum(root);

    cout << "\nTree after sums:\n";

    printBinaryTree(root);

    return 0;

}

Завдання №9 Self Practice Work

#include<iostream>

using namespace std;

int main()

{

    int n, m, i;

    cin >> n >> m;

    int a[n];

    int b[m];

    for (i = 0; i < n; i++)

    {

        cin >> a[i];

    }

    for (i = 0; i < m; i++)

    {

        cin >> b[i];

    }

    int minA = \*min\_element(a, a + n);

    int minB = \*min\_element(b, b + m);

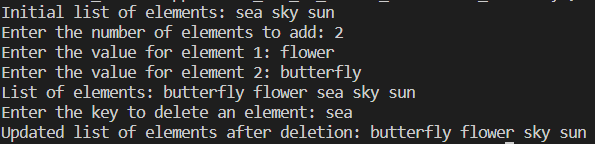
    cout << minA + minB;

    return 0;

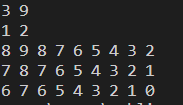
}

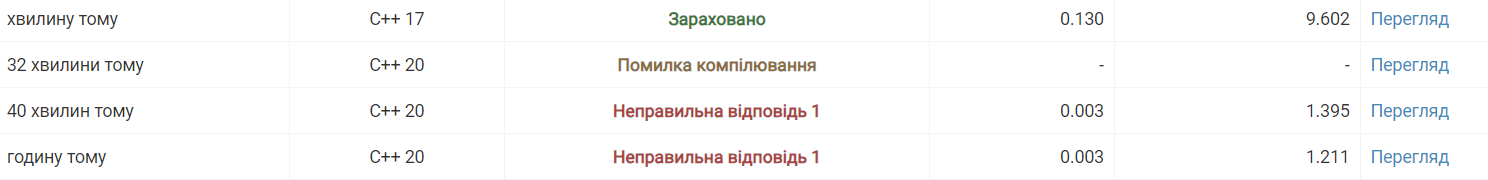
## **5. Результати виконання завдань, тестування та фактично затрачений час:**

Завдання №1 VNS Lab 10



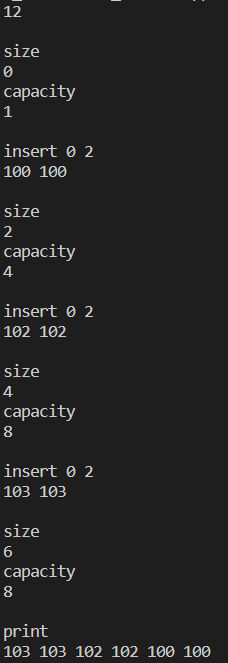
Завдання №2 Algotester Lab 5

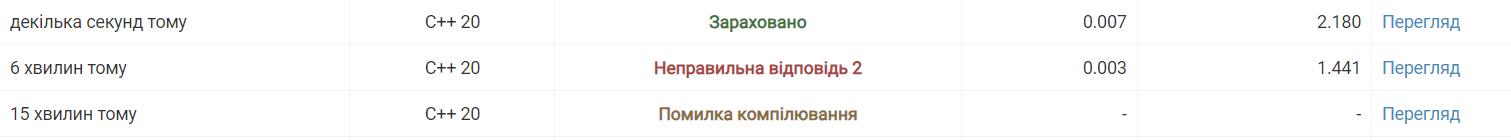




*Рисунок 2. Результат зарахування програми №2 на Алготестері*

Завдання №3 Algotester Lab 7-8





*Рисунок 3. Результат зарахування програми №3 на Алготестері*

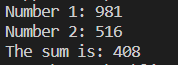
Завдання №4 Class Practice Work Task 1



Завдання №5 Class Practice Work Task 2



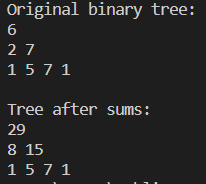
Завдання №6 Class Practice Work Task 3



Завдання №7 Class Practice Work Task 4



Завдання №8 Class Practice Work Task 5



Завдання №9 Self Practice Work

INPUT:

aabbbc

OUTPUT:

3

# **Висновки:**

Працюючи над теоретичним матеріалом та виконуючи завдання розділу, я навчилась працювати з динамічними структурами даних на C++. Під час вивчення матеріалу труднощів не виникло. Всі розглянуті матеріали практично закріплені завдяки їх виконанню в лабораторних і практичних завданнях, а також при самостійній практиці.