Міністерство освіти і науки України

Національний університет «Львівська політехніка»

Кафедра систем штучного інтелекту



**Звіт**

про виконання

**Звіт**

про виконання

**Лабораторних та практичних робіт № 6**

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

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

***Виконала:***

студентка групи ШІ-13

Поліщук Марина Олександрівна

# **Тема роботи:** Динамічні структури (черга, стек, списки, дерево). Алгоритми обробки динамічних структур.

# **Мета роботи:**

# Навчитися працювати з динамічними структурами (черга, стек, списки, дерево), алгоритмами обробки динамічних структур.

# **Теоретичні відомості:**

1. Теоретичні відомості з переліком важливих тем:

- черга

-стек

-списки

-дерева

1. Індивідуальний план опрацювання теорії:

* Тема №1:черга
  + Джерела Інформації
    - лекції, практичні заняття
    - https://prometheus.org.ua/cs50/sections/section6.html

Черга - це структура даних, що працює за принципом "першим прийшов - першим пішов" (first in - first out, LIFO).

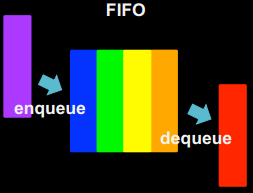


Рисунок 1 FIFO

Для черги визначено дві операції - додавання елемента в кінець черги (enqueue **push()**) та вилучення елемента з початку черги (dequeue **pop()**:).

* + - Статус: Ознайомлена
    - Початок опрацювання теми: 10.10.23
    - Звершення опрацювання теми: 19.12.23
* Тема №2: стек
  + Джерела Інформації:
    - практичні заняття, лекції
    - [https://prometheus.org.ua/cs50/sections/section6.html](https://www.youtube.com/watch?v=V7q9w_s0nns)
  + Що опрацьовано:

Стек - це структура даних, яка працює за принципом "останнім прийшов - першим пішов" (first in - first out, FIFO). Можна уявити собі стіс розносів у їдальні - той рознос, що поклали у стіс останнім, новий клієнт їдальні забере в першу чергу.

LIFO - це абревіатура, яка означає "last in - first out", тобто "останнім прийшов - першим пішов".

Над стеком можна здійснювати дві операції - push (занесення даних) і pop (вилучення даних).

Щоб реалізувати операцію push, необхідно зробити перевірку, чи не перевищує поточний розмір ємність стеку, після чого - вставити елемент на позицію size і збільшити size на одиницю.

Для реалізації операції pop, необхідно перевірити, чи не пустий стек, зменшити поточний розмір на одиницю і повернути елемент.

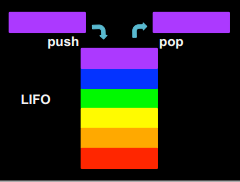


Рисунок 2 LIFO

* + Статус: Ознайомлена
  + Початок опрацювання теми: 10.10.23
  + Звершення опрацювання теми: 19.12.23
* Тема №3: списки
  + Джерела Інформації
    - https://prometheus.org.ua/cs50/sections/section6.html

Що опрацьовано:

Зв'язний список - це структура даних, в якій елементи лінійно впорядковані, але порядок визначається не номерами елементів (як в масивах), а вказівниками, що входять у склад елементів списку та вказують на наступний елемент. Список має "голову" (перший елемент) та "хвіст" (останній елемент).

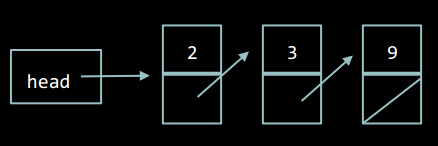


Рисунок 3 зв'язний список

### Переваги і недоліки порівняно з масивами

На відміну від масивів, вставка та вилучення елементів у Зв'язний список потребують сталого часу (О(1)). Також значною перевагою зв'язних списків є можливість легкого розширення: щоб збільшити розмір списку, треба лише додати ще один елемент.

Недоліком зв'язних списків є необхідність проходити весь список, щоб знайти елемент (тобто час доступу до елемента списку - О(n)).

* + Статус: Ознайомлена
  + Початок опрацювання теми: 10.11.23
  + Звершення опрацювання теми: 20.12.23
* Тема №4: дерева
  + Джерела Інформації
    - <https://prometheus.org.ua/cs50/sections/section6.html>
    - <https://www.youtube.com/watch?v=qBFzNW0ALxQ>

Що опрацьовано:

Дерево - це структура даних, де елемент, або вузол, вказує на інші вузли. Приклад дерева:

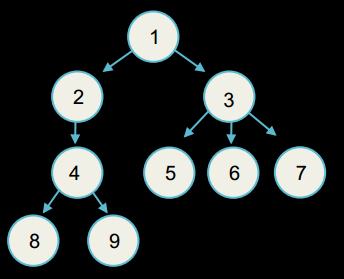


Рисунок 4 дерево

Корінь дерева - верхній вузол у дереві (*1* у прикладі)

Батько - вузол, що вказує на вузол, який ви розглядаєте. Наприклад, батько вузла *3* - вузол *1*.

Нащадок - вузол, на який вказують інші вузли (так, у прикладі вузли *2* та *3* є нащадками вузла *1*).

Листки дерева - вузли, що знаходяться знизу, або вузли, що не мають нащадків.

Брати і сестри - це вузли, що мають спільного батька (у прикладі, вузли *5, 6, 7*).

## **Бінарні дерева**

Бінарне дерево - це дерево, у якому кожен вузол має не більше двох нащадків.

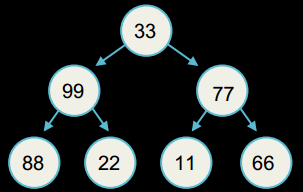


Рисунок 5 бінарне дерево

### Елемент бінарного дерева

Бінарне дерево, як і однозв'язний або двозв'язний список, оголошуються за допомогою свого елемента. У бінарному дереві, кожен елемент має дані (ціле число у прикладі), і посилання на лівого та правого нащадків:

### Бінарне дерево пошуку

У бінарному дереві пошуку все, що справа від батьківського вузла - більше за нього, а все, що зліва - менше.

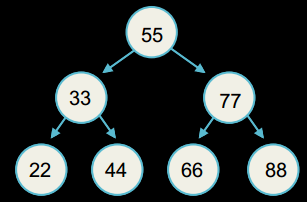


Рисунок 6 бінарне дерево пошуку

Таким чином, пошук елемента в бінарному дереві пошуку дуже схожий на пошук елемента в масиві за алгоритмом бінарного пошуку. Єдина різниця - при пошуку в бінарному дереві, замість лівої або правої половини масиву ми щоразу розглядаємо ліве або праве піддерево.

* + Статус: Ознайомлена
  + Початок опрацювання теми: 10.11.23
  + Звершення опрацювання теми: 20.12.23

# **Виконання роботи:**

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

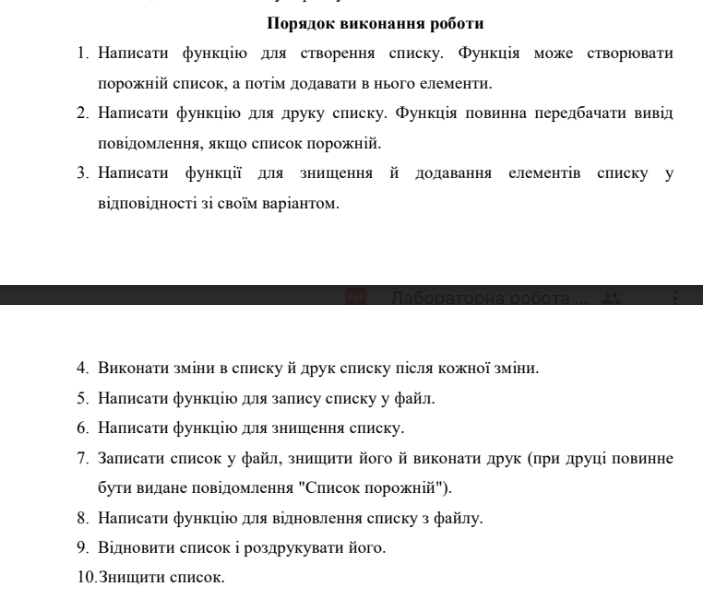
Завдання №1 Опрацювання теорії

* Деталі завдання: опрацювання різноманітних матеріалів, аналіз та дослідження відео та статей на задані теми. Вивчення найважливіших моментів.

Завдання № 2 VNS Lab 10 - Task 1

* Варіант завдання: 21

- Деталі завдання:



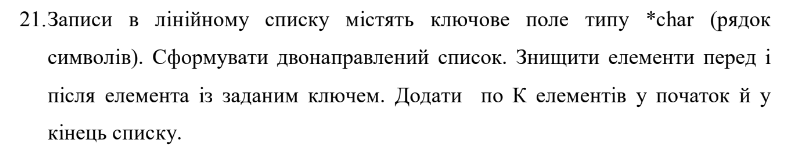


Рисунок 7 деталі завдання

* Важливі деталі для врахування в імплементації програми: вміння працювати з двозвязними списками.

Завдання № 3 Algotester Lab 5v2

* Варіант завдання: 2
* Деталі завдання:

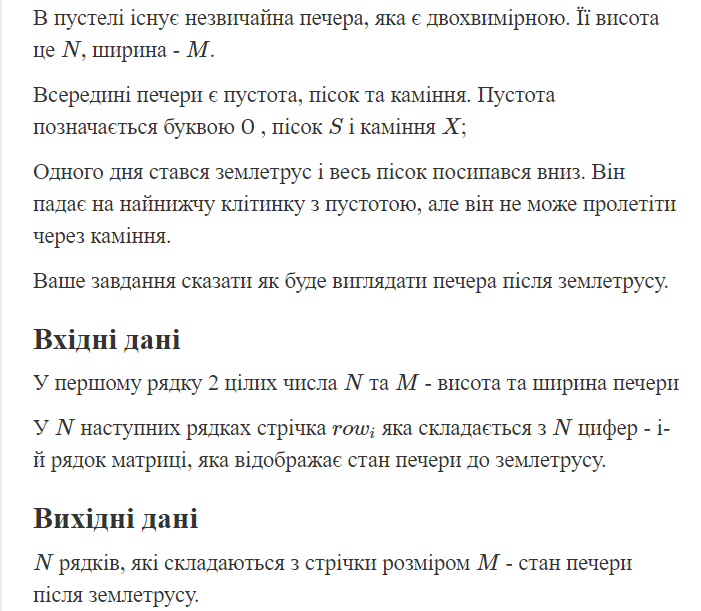
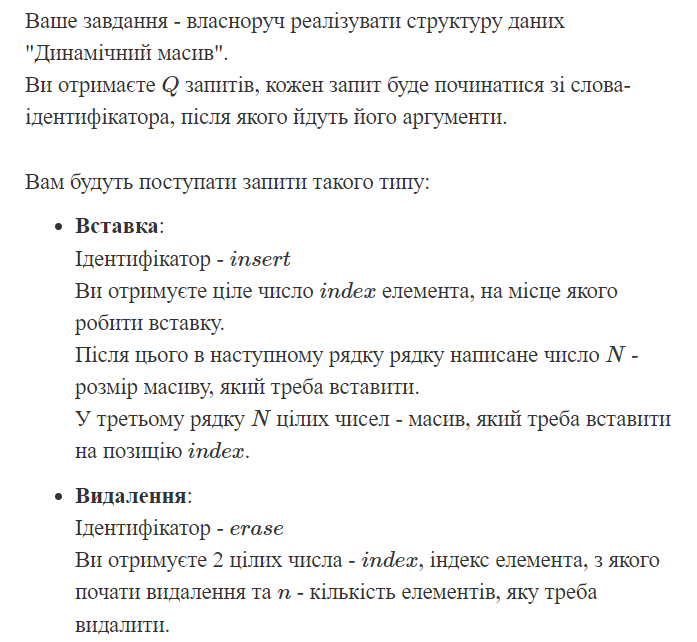
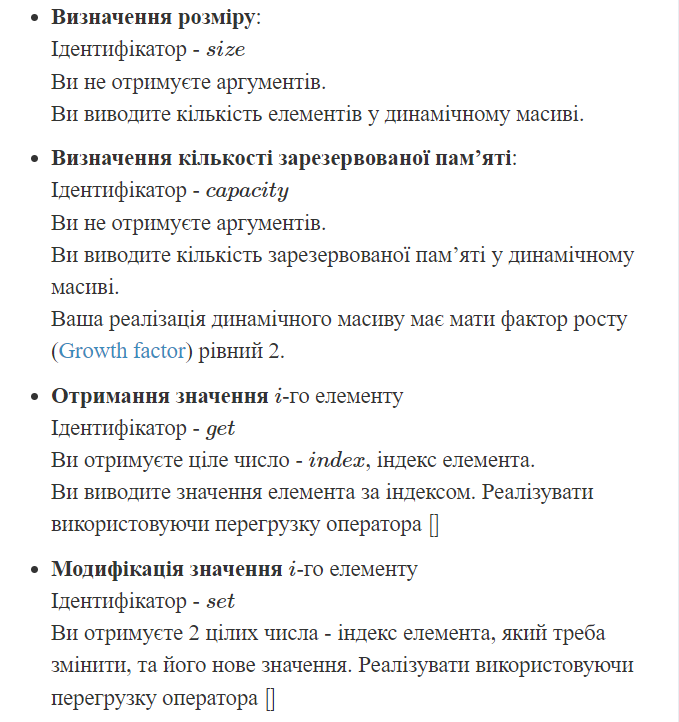


Рисунок 8 деталі завдання

Важливі деталі для врахування в імплементації програми: знання двовимірних масивів, умовними операторами, вміння з ними працювати.

* Завдання №4 Algotester Lab 78v2
* Варіант завдання: 2
* Деталі завдання:





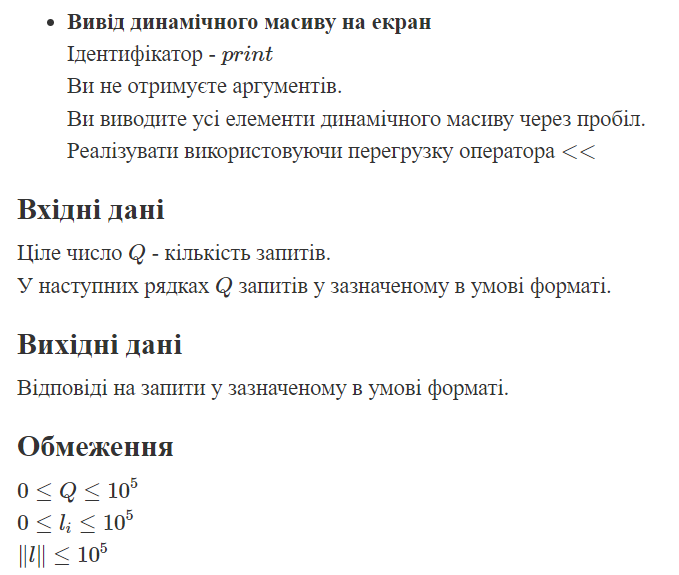


Рисунок 9 деталі завдання

* Важливі деталі для врахування в імплементації програми: врахування обмежень на алготестері
* Завдання №5 Class Practice Task 1
* Варіант завдання:
* Деталі завдання:

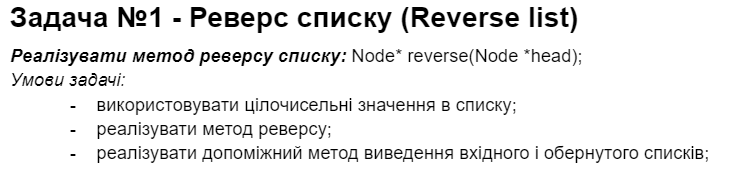


Рисунок 10 деталі завдання

* Важливі деталі для врахування в імплементації програми: ми створюємо ще один двозвязний список, куди запушуємо всі значення з першого, але в порядку навпаки і присвоюємо значення з другого списку першому.

Завдання №6 Class Practice Task 2

* Деталі завдання: Реалізувати програму, яка перевіряє, чи дане слово чи число є паліндромом за допомогою рекурсії.

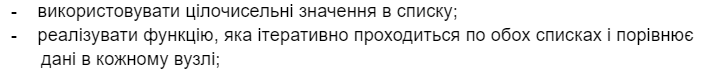




Рисунок 11 деталі завдання

* Важливі деталі для врахування в імплементації програми: вміння працювати з функціями
* Завдання №7 Class Practice Task 3
* Деталі завдання:

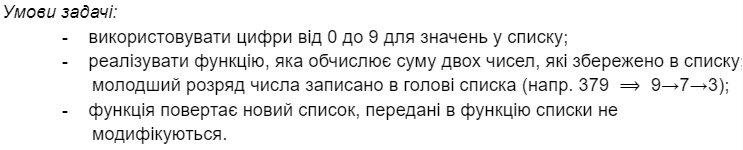
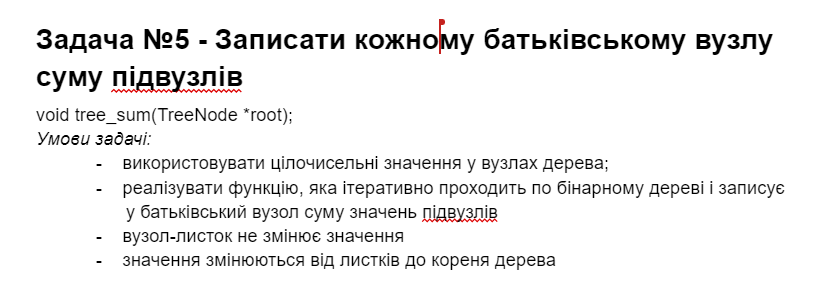


Рисунок 12 деталі завдання

* Важливі деталі для врахування в імплементації програми: використано джерело: https://www.geeksforgeeks.org/sum-of-two-linked-lists/
* Завдання №8 Class Practice Task 4
* Деталі завдання:



* Важливі деталі для врахування в імплементації програми: вміння працювати з деревами
* Завдання №9 Class Practice Task 5
* Деталі завдання:

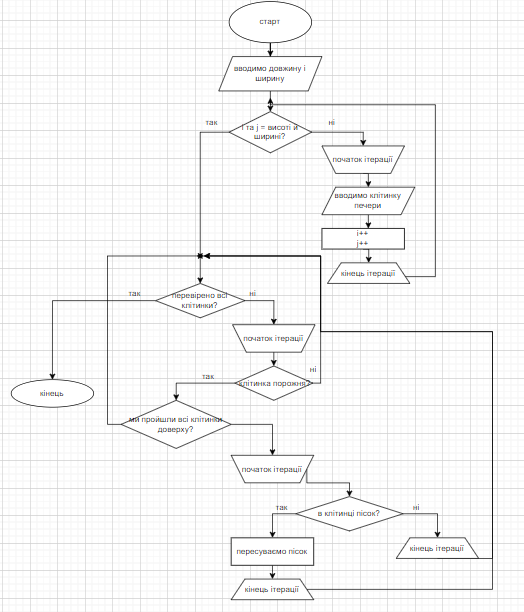


* Важливі деталі для врахування в імплементації програми: вміння працювати з деревами

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

Програма №5 Self Practice Task

* Блок-схема



* Планований час на реалізацію: 2 години
* Важливі деталі для врахування в імплементації: використання умовних операторів, умов алго ((

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

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

Завдання № 2 Деталі по програмі + Вставка з кодом з підписами до вставки. Посилання на файл програми у пул-запиті GitHub

#include <iostream>

#include <fstream>

#include <string>

using namespace std;

struct Node

{

    char data;

    Node\* prev;

    Node\* next;

};

class dll

{

private:

    Node head;

    Node tail;

public:

        //1. Написати функцію для створення списку.

    dll(){ // функція, яка нічого не повертає, але створює елемент класу

        head.prev = nullptr;

        head.next = &tail;

        tail.prev = &head;

        tail.next = nullptr;

    }

    Node\* start() {

        return head.next; //Node0

    }

    //head = Node

    //head.next = Node\*

    Node\* end() {

        return tail.prev;

    }

    //вставка елементів

    void push(char data) {

        insert\_after(head, data);

    }

    void push\_back(char data) {

        insert\_after(\*tail.prev, data);

    }

    void insert\_after(Node &after, char value){

        Node\* newNode = new Node(); //"чиста" нода

        newNode->data = value; //а тут вже щось записуємо

        // cout << "ins: " << newNode->data << endl;

        newNode->next = after.next;

        newNode->prev = &after;

        after.next->prev = newNode;

        after.next = newNode;

    }

        // 2. Написати функцію для друку списку.

    void print(){

        Node\* node = start(); //ми створюємо, який буде посиланням на початок списку

        if(node->next == nullptr)

        {

            cout << "list is empty...\n";

            return;

        }

        while(node->next != nullptr)

        {

            cout << node->data << " ";

            node = node->next;

        }

        cout << endl;

    }

        //3. видалення згідно зі своїм варіантом

    void deleteBeforeAndAfter(int key){ // key >= 2

        if(key < 2)

        {

            cout << "wrong key\n";

            return;

        }

        key--;

        deleteElement(key-1);

        deleteElement(key);

    }

        //3. Написати функції для знищення й додавання елементів списку

    void deleteElement(int key){

        Node\* node = start();

        for(int i = 1; i < key; i++) // проходимось до потрібного вузла в dll

            node = node->next;

        node->next = node->next->next;

        delete node->next->prev;

        node->next->prev = node;

    }

        //5. Написати функцію для запису списку у файл

    void writeToFile(string fileName)

    {

        ofstream fout(fileName);

        Node\* node = start();

        while(node->next != nullptr)

        {

            fout << node->data << endl;

            node = node->next;

        }

    }

        //8. Написати функцію для відновлення списку з файлу

    void restoreFromFile(string fileName)

    {

        Node\* node = start();

        ifstream fin ("newFileWithData.dat");

        if (!fin.is\_open())

            return;

        string str;

        while(getline(fin, str))

            push\_back(\*str.c\_str());

        fin.close();

    }

        //7. знищення списку

    void destroyList()

    {

        Node\* current = start()->next;

            //видалення кожного вузла

        while (current->next != nullptr)

        {

            delete current->prev;

            current = current->next;

        }

        delete current->prev;

            //обнуляємо список

        head.next = &tail;

        tail.prev = &head;

        head.prev = nullptr;

        tail.next = nullptr;

    }

};

int main()

{

    dll list = dll(); //запускаємо функцію в класі dll для ініціалізації об'єкту класу

    // char chars[20];

        // 4. Виконати зміни в списку

        cout << "origin list: \n";

        list.push\_back('8');

        //4.й друк після зміни

    list.print();

    //Додати по К елементів у початок й у кінець списку.

    int k = 4;

    for (int i = 0; i < k; i++)

    {

        list.push(97 + i); //97 -> ASCII код літери a

        list.push\_back(48 + i); //48 -> 0

    }

    cout << "list after adding k elements to start and end: \n";

    list.print();

        //Знищити елементи перед і після елемента із заданим ключем.

    int key = 3;

    list.deleteBeforeAndAfter(key);

    cout << "list after deleting elements behind and after key: \n";

    list.print();

        //запис та зчитування з файлу

    list.writeToFile("newFileWithData.dat");

    dll list2 = dll();

    list2.restoreFromFile("newFileWithData.dat");

    // list2.deleteElement(2);

    // cout << list2.end()->data;

    cout << "restored list: \n";

    list2.print();

    list2.destroyList();

    cout << "there we delete list...\n";

    list2.print();

    return 0;

}

## Завдання № 3 Деталі по програмі + Вставка з кодом з підписами до вставки. Посилання на файл програми у пул-запиті GitHub

# include <iostream>

using namespace std;

int main (){

    int N, M;

    cin >> N >> M;

    char arr[N][M];

    for(int i = 0; i < N; i++)

        {

            for(int j = 0; j < M; j++)

            {

                cin >> arr[i][j];

            }

        }

    for(int i = N-1; i >= 0; i--)

        {

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

                if(arr[i][j]=='O'){

                    for (int k=i; k>= 0; k--){

                        if (arr[k][j]=='S'){

                            arr [i][j]='S';

                            arr [k][j]='O';

                            break;

                        }

                        else if (arr[k][j]=='X'){

                            break;

                        }

                    }

                }

            }

        }

    for(int i = 0; i < N; i++){

        for(int j = 0; j < M; j++){

            cout << arr[i][j];

        }

       cout <<  endl;

    }

    return 0;

}

Завдання № 4 Деталі по програмі + Вставка з кодом з підписами до вставки. Посилання на файл програми у пул-запиті GitHub

#include <iostream>

#include <string>

using namespace std;

// <int>

template <typename T> //створюємо клас для будь якого типу данних

class DynamicArray

{

private:

    T\* arr;

    int size;

    int capacity;

    const int growthFactor = 2;

    void resize()

    {

        do {

            capacity \*= growthFactor; //збільшуємо фактичний розмір масиву

        } while (capacity <= size);

        T\* newArr = new T[capacity]; //new виділяє пам'ять та повертає адресу цієї памяті

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

            newArr[i] = arr[i];

        delete[] arr; //arr більше не посиляється на старі данні

        arr = newArr; //переписавши їй у новий збільшений масив в циклі for, ми перепосилаємо старий масив на новий

    }

public:

    DynamicArray() : size(0), capacity(1) {

        arr = new T[capacity];

    }

    ~DynamicArray() { //для автоматичного видалення данних, якщо їх життевий цикл закінчився (використання, напр у функціях)

        delete[] arr;

    }

    void insert(int index, int length, T\* values)

    {

        if(index > size)

            return;

            //збільшуємо фактичний розмір масиву

        while (size + length >= capacity)

            resize();

            //тут ми копіюємо данні зі старого в новий, то того індексу, після якого нам треба вставити потрібно значення

        T\* newArr = new T[capacity];

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

            newArr[i] = arr[i];

        }

        for (int i = 0; i < length; i++) //тут ми додаємо ці потрібні значення values[i]

            newArr[index + i] = values[i];

        for (int i = index + length; i < size + length; i++) //додаємо значення які залилшилися

            newArr[i] = arr[i - length];

        delete[] arr;

        arr = newArr;

        size += length;

    }

    void erase(int index, int n)

    {

        T\* newArr = new T[capacity];

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

            newArr[i] = arr[i];

        for (int i = index + n; i < size; i++)

            newArr[i - n] = arr[i];

        delete[] arr;

        arr = newArr;

        size -= n;

    }

    int getSize() {

        return size;

    }

    int getCapacity() {

        return capacity;

    }

    int get(int index) {

        return arr[index];

    }

    void set(int index, T value) {

        arr[index] = value;

    }

    void print()

    {

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

            cout << arr[i] << " ";

        cout << endl;

    }

};

int main()

{

    int q;

    cin >> q;

    DynamicArray<int> myArr;

    for (int i = 0; i < q; i++)

    {

        string input;

        cin >> input;

        if (input == "size")

            cout << myArr.getSize() << endl;

        else if (input == "capacity")

            cout << myArr.getCapacity() << endl;

        else if (input == "insert")

        {

            int index, count;

            cin >> index >> count;

            int arr[count];

            for (int j = 0; j < count; j++)

                cin >> arr[j];

            myArr.insert(index, count, arr);

        }

        else if (input == "get")

        {

            int index;

            cin >> index;

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

        }

        else if (input == "set")

        {

            int index, value;

            cin >> index >> value;

            myArr.set(index, value);

        }

        else if (input == "erase")

        {

            int index, count;

            cin >> index >> count;

            myArr.erase(index, count);

        }

        else if (input == "print")

            myArr.print();

    }

    return 0;

}

## Завдання № 5 Деталі по програмі + Вставка з кодом з підписами до вставки. Посилання на файл програми у пул-запиті GitHub

#include <iostream>

#include <string>

using namespace std;

struct Node

{

    int data;

    Node\* prev;

    Node\* next;

};

class dll

{

private:

    Node head;

    Node tail;

public:

    dll()

    {

        head.prev = nullptr;

        head.next = &tail;

        tail.prev = &head;

        tail.next = nullptr;

    }

    Node\* start() {

        return head.next;

    }

    Node\* end() {

        return tail.prev;

    }

    void push(int data) {

        insert\_after(head, data);

    }

    void push\_back(int data) {

        insert\_after(\*tail.prev, data);

    }

    void insert\_after(Node &after, int value)

    {

        Node\* newNode = new Node();

        newNode->data = value;

        newNode->next = after.next;

        newNode->prev = &after;

        after.next->prev = newNode;

        after.next = newNode;

    }

    void print()

    {

        Node\* node = start();

        if(node->next == nullptr)

        {

            cout << "list is empty...\n";

            return;

        }

        while(node->next != nullptr)

        {

            cout << node->data << " ";

            node = node->next;

        }

        cout << endl;

    }

    dll reverse()

    {

        dll copy = dll();

        for (Node\* node = start(); node->next != nullptr; node = node->next)

            copy.push(node->data);

        return copy;

    }

};

int main()

{

    dll list = dll();

        cout << "origin list: \n";

    int k = 4;

    for (int i = 0; i < k; i++)

    {

        list.push(i);

        list.push\_back(-i - 5);

    }

    list.print();

    list = list.reverse();

    cout << "reverse list: \n";

    list.print();

    return 0;

}

Завдання № 6 Деталі по програмі + Вставка з кодом з підписами до вставки. Посилання на файл програми у пул-запиті GitHub

#include <iostream>

#include <fstream>

#include <string>

using namespace std;

struct Node

{

    int data;

    Node\* next;

};

class LinkedList

{

private:

    Node head;

    Node tail;

public:

    LinkedList() // нічого не повертає, але створює елемент класу

    {

        head.next = &tail;

        tail.next = nullptr;

    }

    Node\* start() {

        return head.next;

    }

    //вставка елементів

    void push(char data) {

        insert\_after(head, data);

    }

    void insert\_after(Node &after, char value)

    {

        Node\* newNode = new Node();

        newNode->data = value; // записуємо

        newNode->next = after.next;

        after.next = newNode;

        length++;

    }

        // 2. Написати функцію для друку списку.

    void print()

    {

        Node\* node = start(); //ми створюємо, який буде посиланням на початок списку

        if(node->next == nullptr)

        {

            cout << "list is empty...\n";

            return;

        }

        while(node->next != nullptr)

        {

            cout << node->data << " ";

            node = node->next;

        }

        cout << endl;

    }

    int length = 0;

    bool compareWith(LinkedList &secondList)

    {

        if(length != secondList.length)

            return false;

        Node\* n1 = start();

        Node\* n2 = secondList.start();

        while (n1->next != nullptr)

        {

            if (n1->data != n2->data)

                return false;

            n1 = n1->next;

            n2 = n2->next;

        }

        return true;

    }

};

int main()

{

    LinkedList list = LinkedList();

    LinkedList list2 = LinkedList();

    for (int i = 0; i < 6; i++)

    {

        list.push(i);

        list2.push(i);

    }

    list.print();

    list2.print();

    list.compareWith(list2) ? cout << "equal" : cout << "different";

    cout << endl;

    list.push(5);

    list2.push(-5);

    list.print();

    list2.print();

    list2.compareWith(list) ? cout << "equal" : cout << "different";

    return 0;

}

Завдання № 7 Деталі по програмі + Вставка з кодом з підписами до вставки. Посилання на файл програми у пул-запиті GitHub

#include <bits/stdc++.h>

using namespace std;

class Node {

public:

    int data;

    Node\* next;

};

//для додавання вузолу з початок списку

void push(Node\*\* head\_ref, //передаємо адресу вказівника head, а не сам вказівник head

                          int new\_data) {

    Node\* new\_node = new Node();

    new\_node->data = new\_data;

    new\_node->next = (\*head\_ref);

    (\*head\_ref) = new\_node;

    //(\*head\_ref) = new\_node; - ця операція має значення за адресою head\_ref, тобто ми

    //змінюємо сам вказівник head на новий вузол, що створений у цій функції.

}

//виведення на екран

void printList(Node\* node) {

    while (node != NULL) {

        cout << node->data << " ";

        node = node->next;

    }

    cout << endl;

}

//функція для обміну двома вказівниками на вузли.

void swapPointer(Node\*\* a, Node\*\* b) {

    Node\* t = \*a;

    \*a = \*b;

    \*b = t;

}

int getSize(Node\* node) {

    int size = 0;

    while (node != NULL) {

        node = node->next;

        size++;

    }

    return size;

}

//додає два списки того ж розміру разом

Node\* addSameSize(Node\* head1, Node\* head2, int\* carry) {

    if (head1 == NULL)

        return NULL;

    int sum;

    // Створюємо новий вузол для результату

    Node\* result = new Node();

    result->next = addSameSize(head1->next, head2->next, carry);

    sum = head1->data + head2->data + \*carry;

    \*carry = sum / 10; //обчислює "перенос" для наступного розряду

    sum = sum % 10; //обмежує значення sum до одиниць

    result->data = sum; //присвоюємо вузлу result (який представляє результат додавання вузлів) значення sum

    return result;

}

//викликається після додавання меншого списку до підсписку більшого списку того ж розміру.

void addCarryToRemaining(Node\* head1, Node\* cur, int\* carry, Node\*\* result) {

    int sum;

    if (head1 != cur) { //cur - це вказівник на останній вузол у потрібній частині списку, яку ми додаємо до результату.

        addCarryToRemaining(head1->next, cur, carry, result);

        //просуваємося вперед у списку до останнього вузла, який потрібно додати до результату.

        sum = head1->data + \*carry; //\*carry. Ми додаємо значення поточного вузла до значення "переносу"

        \*carry = sum / 10;

        sum %= 10;

        push(result, sum);

    }

}

//визначає, який список довший, а потім викликає addSameSize для додавання списків

//того ж розміру та addCarryToRemaining для обробки залишкових елементів.

void addList(Node\* head1, Node\* head2, Node\*\* result) {

    Node\* cur;

    //Перевірка, чи один зі списків є порожнім:

    if (head1 == NULL) {

        \*result = head2;

        return;

    } else if (head2 == NULL) {

        \*result = head1;

        return;

    }

    //функція getSize підраховує кількість вузлів у списку.

    int size1 = getSize(head1);

    int size2 = getSize(head2);

    int carry = 0;

    if (size1 == size2)

        \*result = addSameSize(head1, head2, &carry);

    else {

        //обчислення кількості вузлів, що залишилося в більшому списку після додавання рівних розміром частин.

        int diff = abs(size1 - size2);

        //Якщо один список коротший за інший, вони обмінюються.

        if (size1 < size2)

            swapPointer(&head1, &head2);

        for (cur = head1; diff--; cur = cur->next);

        \*result = addSameSize(cur, head2, &carry);

        addCarryToRemaining(head1, cur, &carry, result);

    }

    //Якщо carry не дорівнює нулю, це означає, що в останньому додаванні відбувся перенос. Тут він додається до результату.

    if (carry)

        push(result, carry);

}

int main() {

    Node \*head1 = NULL, \*head2 = NULL, \*result = NULL;

    int arr1[] = { 9, 9, 9 };

    int arr2[] = { 1, 8 };

    int size1 = sizeof(arr1) / sizeof(arr1[0]);

    int size2 = sizeof(arr2) / sizeof(arr2[0]);

    for (int i = size1 - 1; i >= 0; --i)

        push(&head1, arr1[i]);

    for (int i = size2 - 1; i >= 0; --i)

        push(&head2, arr2[i]);

    addList(head1, head2, &result);

    printList(result);

    return 0;

}

Завдання № 8 Деталі по програмі + Вставка з кодом з підписами до вставки. Посилання на файл програми у пул-запиті GitHub

#include <iostream>

using namespace std;

// Визначення структури вузла дерева

struct TreeNode {

    int val;

    TreeNode \*left;

    TreeNode \*right;

    // Конструктор для створення вузла зі значенням x

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

    //отримує значення для вузла і встановлює вказівники на ліве та праве піддерево як nullptr

};

// Функція для створення віддзеркаленого дерева

TreeNode \*create\_mirror\_flip(TreeNode \*root) {

    if (root == nullptr) {

        return nullptr;

    }

    // Створення нового вузла зі значенням вузла root, але з обміняними лівим і правим піддеревами

    TreeNode \*mirror = new TreeNode(root->val); // Створення нового вузла зі значенням вузла root,

    //minor - вказівник на наш елемент

    mirror->right = create\_mirror\_flip(root->left); // Рекурсивно викликає саму себе для лівого піддерева root, присвоюючи його правому піддереву новий вузол

    mirror->left = create\_mirror\_flip(root->right); // Рекурсивно викликає саму себе для правого піддерева root, присвоюючи його лівому піддереву новий вузол

    // Повертає віддзеркалене піддерево

    return mirror;

}

// Функція для обходу дерева в порядку "впорядкованого виводу"

void inorderTraversal(TreeNode \*root) {

    if (root != nullptr) {

        inorderTraversal(root->left); // Рекурсивний виклик для лівого піддерева

        cout << root->val << " "; // Вивід значення поточного вузла

        inorderTraversal(root->right); // Рекурсивний виклик для правого піддерева

    }

}

int main() {

    // Приклад вхідного дерева

    TreeNode \*root = new TreeNode(3);

    root->left = new TreeNode(1);

    root->right = new TreeNode(4);

    root->right->right = new TreeNode(8);

    root->left->right = new TreeNode(2);

    cout << "Original tree: ";

    inorderTraversal(root); // Вивід вхідного дерева у порядку "впорядкованого виводу"

    cout << endl;

    // Створення віддзеркаленого дерева без модифікації вхідного дерева

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

    cout << "Changed tree: ";

    inorderTraversal(mirrorTree); // Вивід віддзеркаленого дерева у порядку "впорядкованого виводу"

    cout << endl;

    return 0;

}

Завдання № 9 Деталі по програмі + Вставка з кодом з підписами до вставки. Посилання на файл програми у пул-запиті GitHub

#include <iostream>

#include <stack>

using namespace std;

struct TreeNode {

    int data;

    TreeNode\* left;

    TreeNode\* right;

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

};

void print\_tree(TreeNode\* root) {

    if (root) {

        print\_tree(root->left);

        cout << root->data << " ";

        print\_tree(root->right);

    }

}

void calculate\_tree\_sum(TreeNode\* root) {

    stack<TreeNode\*> nodes; //Створення стеку nodes

    TreeNode\* prev = nullptr;//для зберігання попереднього вузла під час обходу.

    //Якщо вхідний вузол root не є nullptr, то додаємо його до стеку.

    if (root)

        nodes.push(root);

    while (!nodes.empty()) {

        TreeNode\* current = nodes.top();

    /\*Умова перевіряє, чи поточний вузол - перший вузол у своєму піддереві або чи він є лівим або правим нащадком попереднього вузла.

    Якщо так, то переходимо до наступного вузла у піддереві (лівого, якщо існує, інакше правого).\*/

        if (!prev || prev->left == current || prev->right == current) {

            if (current->left)

                nodes.push(current->left);

            else if (current->right)

                nodes.push(current->right);

    //Якщо поточний вузол був лівим нащадком попереднього вузла, перевіряємо наявність правого нащадка поточного вузла.

        } else if (current->left == prev) {

            if (current->right)

                nodes.push(current->right);

                /\*Якщо поточний вузол не є першим у своєму піддереві та має обидва нащадки або вже обробив їх, тоді обчислюємо суму його значення

                та значень лівого та правого нащадків (якщо вони існують) і записуємо цю суму у поле data поточного вузла. Після цього вузол

                видаляється зі стеку, оскільки ми завершили обробку його нащадків.\*/

        } else {

            current->data = current->data +

                            (current->left ? current->left->data : 0) +

                            (current->right ? current->right->data : 0);

            nodes.pop();

        }

        //prev оновлюється до поточного вузла для використання у наступній ітерації.

        prev = current;

    }

}

int main() {

    TreeNode\* root = new TreeNode(4);

    root->left = new TreeNode(2);

    root->right = new TreeNode(6);

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

    root->left->right = new TreeNode(3);

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

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

    cout << "Tree before changes: ";

    print\_tree(root);

    cout << endl;

    calculate\_tree\_sum(root);

    cout << "Tree after adding: ";

    print\_tree(root);

    cout << endl;

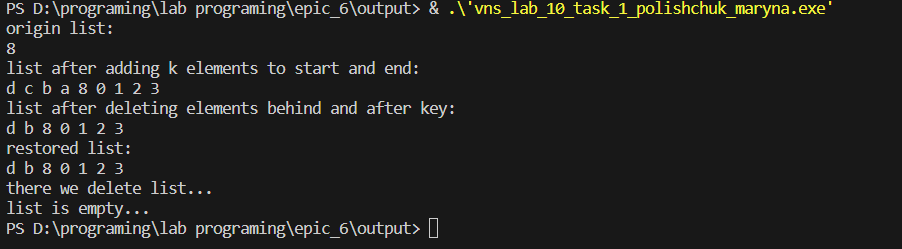
    return 0;

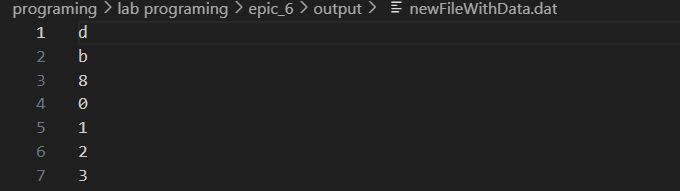
}

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

Завдання № 2 VNS Lab 10 - Task 1

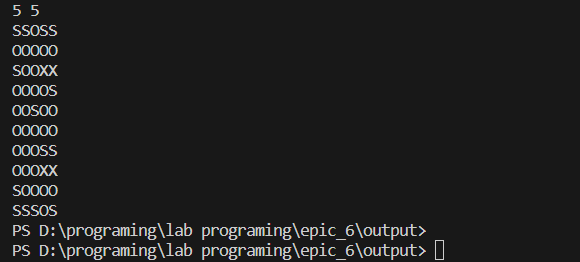
Деталі по виконанню і тестуванню програми

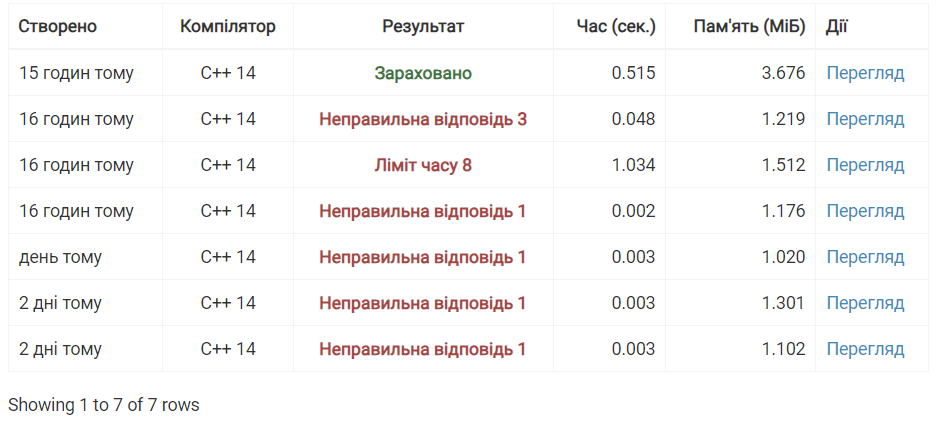




Час затрачений на виконання завдання: 3 дні

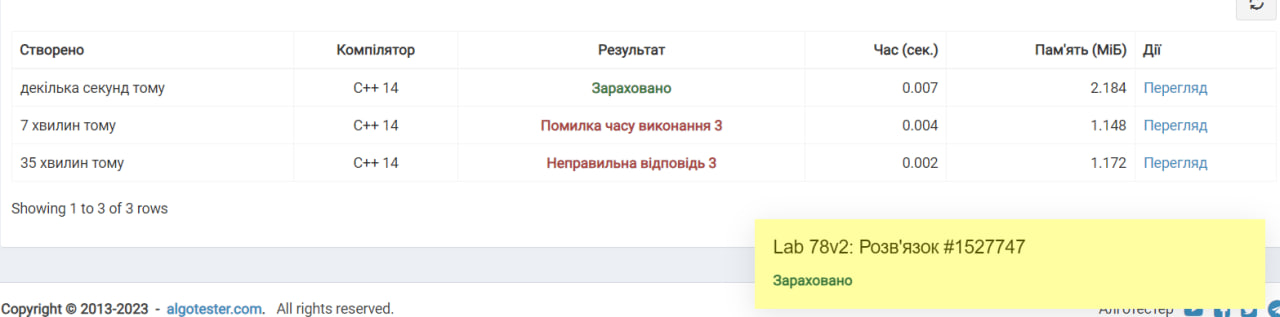
Завдання № 3 Algotester Lab 5v2

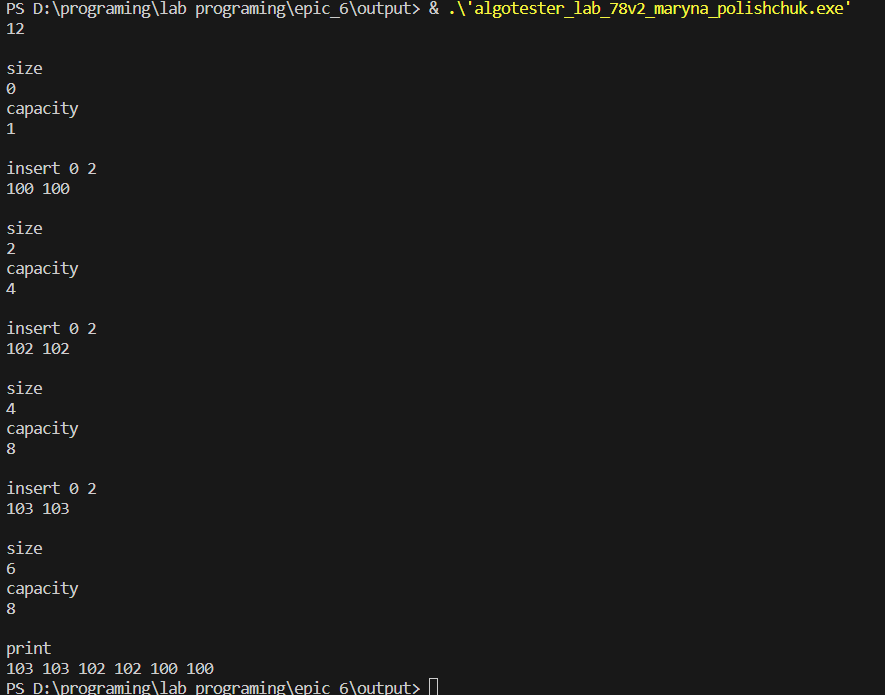




Час затрачений на виконання завдання: 10 годин

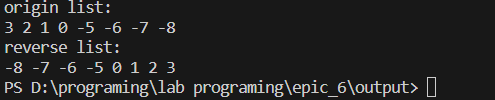
Завдання № 4 Algotester Lab 78v2

Деталі по виконанню і тестуванню програми 

 Час затрачений на виконання завдання: 10 годин

Завдання № 5 Class Practice task 1

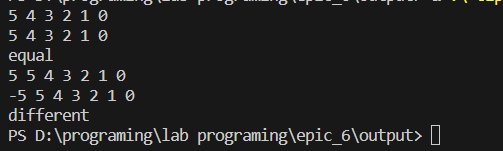
Деталі по виконанню і тестуванню програми



Час затрачений на виконання завдання: 6 годин

Завдання № 6 Class Practice Task 2

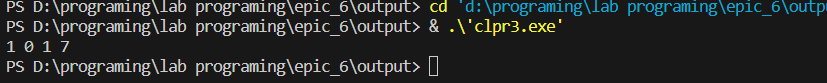
Деталі по виконанню і тестуванню програми



Час затрачений на виконання завдання: 5 годин

Завдання №7 Class Practice Task 3

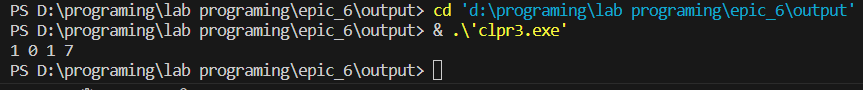
Деталі по виконанню і тестуванню програми



Час затрачений на виконання завдання: 5 годин

Завдання №7 Class Practice Task 3

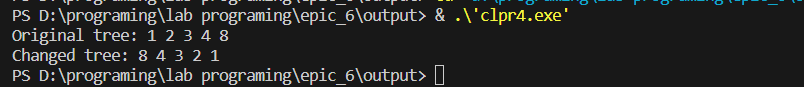
Деталі по виконанню і тестуванню програми



Час затрачений на виконання завдання: 3 години

Завдання №8 Class Practice Task 4

Деталі по виконанню і тестуванню програми



Час затрачений на виконання завдання: 4 години

Завдання №9  Class Practice Task 5

 Деталі по виконанню і тестуванню програми



Час затрачений на виконання завдання: 3 години

# **Висновки:**

# Ми навчилися працювати з динамічними структурами (черга, стек, списки, дерево), алгоритмами обробки динамічних структур.