Міністерство освіти і науки України

Національний університет «Львівська політехніка»

Кафедра систем штучного інтелекту



*Figure 1.*

**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.»

***Виконав:***

студент групи ШІ-13

***Булишин Віктор***

**Тема роботи:**

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

**Мета роботи:**

Навчитись працювати з динамічними структурами (Чергами, Стеками, Списками, Деревом). Алгоритмами обробки динамічних структур.

**Теоретичні відомості:**

1. Теоретичні відомості з переліком важливих тем:

* Тема №1: Динамічні структури(Черга, Стеки, Списки, Дерево).
* Тема №2: Алгоритми обробки динамічних структур.

1. Індивідуальний план опрацювання теорії:

* Тема №1: Динамічні структури(Черга, Стеки, Списки, Дерево).
  + Джерела Інформації
    - <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/>
    - <https://acode.com.ua/urok-111-stek-i-kupa/>
    - <https://www.bestprog.net/uk/2022/02/11/c-linear-singly-linked-list-general-information-ua/>
    - <https://purecodecpp.com/archives/2483>
  + Що опрацьовано:
    - Дізнався що таке черга та навчився з нею працювати.
    - Дізнався що таке стеки та навчився з ними працювати.
    - Дізнався що таке списки та навчився з ними працювати.
    - Дізнався що таке дерево та навчився з ним працювати.
  + Статус: Ознайомлений
  + Початок опрацювання теми: 18.12.2023
  + Звершення опрацювання теми: 20.12.2023
* Тема №2: Алгоритми обробки динамічних структур.
  + Джерела Інформації:
    - <http://cpp.dp.ua/vykorystannya-struktur/>
  + Що опрацьовано:
    - Навчився обробляти динамічні масиви та структури.
  + Статус: Ознайомлений
  + Початок опрацювання теми: 18.12.2023
  + Звершення опрацювання теми: 20.12.2023

**Виконання роботи:**

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

Завдання №1 Lab# programming: VNS Lab 10

* Варіант завдання №14
* У програмі створюються динамічні структури та обробляються згідно поставленого завдання
* У кінці список знищується та відновлюється.

Завдання №2 Lab# programming: Algotester Lab 5

* Варіант завдання №3
* Написати праграмний код в середовищі С++ який імітує землетрус в печері.
* Пісок буде падати поки його щось не зупинить.

Завдання №3 Lab# programming: Algotester Lab 7-8

* Варіант завдання №1
* У цій програмі зображена структура дерева двійкового пошуку.
* Щоб зробити запити необхідно вказати слово індетифікатор та його параметри

Завдання №4 Practice# programming: Class Practice Task

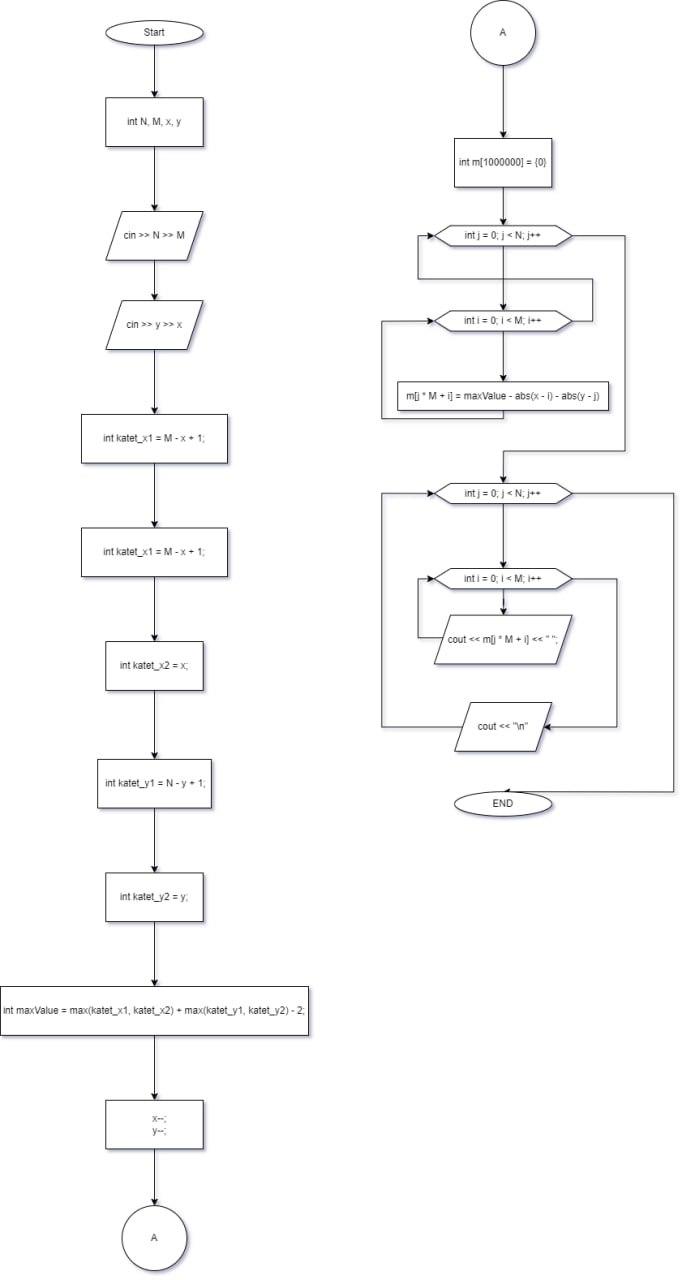
* Варіант завдання без варіанту
* У цьому завданні виконується 5 задач.
* Усі задачі у різних файлах.

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

Програма №1 Lab# programming: VNS Lab 10

* Планований час на реалізацію 2 години
* У кінці список знищується та відновлюється.

Програма №2 Lab# programming: Algotester Lab 5 + Блок-схема



*Figure 2.*

* Планований час на реалізацію 1,5 години
* Пісок буде падати поки його щось не зупинить.

Програма №3 Lab# programming: Algotester Lab 7-8

* Планований час на реалізацію 2 години
* Щоб зробити запити необхідно вказати слово індетифікатор та його параметри

Програма №4 Practice# programming: Class Practice Task

* Планований час на реалізацію 3 години
* Усі задачі у різних файлах.

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

Завдання №1,2,3,4 Деталі по конфігурації середовища + скріншоти з підписами до скріншотів.

Завдання №1,2,3,4,5,6 Деталі по конфігурації середовища + скріншоти з підписами до

Пакет розширень C/C++ включає набір розширень для розробки C++ у Visual Studio Code: C/C++, C/C++ Extensions Pack.

Розширення C/C++ додає підтримку мови C/C++ до Visual Studio Code, включаючи функції редагування (IntelliSense) і налагодження(debugging).

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

Завдання VNS 10 :

#include <iostream>

#include <cstring>

#include <fstream>

using namespace std;

struct Node {

char key[10];

Node\* next;

Node\* prev;

};

struct dblLinkedList {

Node\* head;

Node\* tail;

int size;

};

dblLinkedList\* createList() {

dblLinkedList\* List = new dblLinkedList;

List->head = nullptr;

List->tail = nullptr;

List->size = 0;

return List;

}

void destroyList(dblLinkedList\* List) {

Node\* data = List->head;

while (data != nullptr) {

Node\* next\_data = data->next;

delete data;

data = next\_data;

List->size--;

}

List->head = nullptr;

List->tail = nullptr;

}

void printList(dblLinkedList\* List) {

Node\* data = List->head;

if (data == nullptr) {

cout << "List is empty" << endl;

return;

} else {

while (data != nullptr) {

cout << data->key << " ";

data = data->next;

}

cout << endl;

}

}

void addEl(dblLinkedList\* List, int index, const char\* str) {

index--;

if (index > List->size || index < 0) {

cout << "wrong index:(" << endl;

return;

}

Node\* newNode = new Node;

strcpy(newNode->key, str);

if (List->head == nullptr) {

List->head = newNode;

newNode->next = nullptr;

newNode->prev = nullptr;

List->tail = newNode;

List->size++;

} else {

Node\* current = List->head;

int i = 0;

while (current != nullptr && i < index) {

current = current->next;

i++;

}

if (current == List->head) {

newNode->next = List->head;

List->head->prev = newNode;

List->head = newNode;

newNode->prev = nullptr;

} else if (current == nullptr) {

newNode->next = nullptr;

newNode->prev = List->tail;

List->tail->next = newNode;

List->tail = newNode;

} else {

newNode->next = current;

newNode->prev = current->prev;

current->prev->next = newNode;

current->prev = newNode;

}

List->size++;

}

}

void deleteEl(dblLinkedList\* List, int index) {

index--;

if (List->size == 0) {

cout << "it is nothing to delete" << endl;

return;

}

if (index >= List->size || index < 0) {

cout << "wrong index" << endl;

return;

}

Node\* current = List->head;

int i = 0;

while (current != nullptr && i < index) {

current = current->next;

i++;

}

if (current->prev != nullptr) {

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

}

if (current->next != nullptr) {

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

}

if (current->prev == nullptr) {

List->head = current->next;

}

if (current->next == nullptr) {

List->tail = current->prev;

}

List->size--;

delete current;

}

void write\_to\_file(dblLinkedList\* List, const char\* filename) {

ofstream oFile(filename);

if (oFile.is\_open()) {

Node\* data = List->head;

while (data != nullptr) {

oFile << data->key << endl;

data = data->next;

}

oFile.close();

} else {

cout << "error opening file" << endl;

}

}

dblLinkedList\* restore(dblLinkedList\* List, const char\* filename) {

ifstream iFile(filename);

if (iFile.is\_open()) {

char str[20];

while (iFile.getline(str, sizeof(str))) {

Node\* elem = new Node;

strcpy(elem->key, str);

if (List->head == nullptr) {

List->head = elem;

List->tail = elem;

elem->next = nullptr;

elem->prev = nullptr;

List->size++;

} else {

elem->prev = List->tail;

elem->next = nullptr;

elem->prev->next = elem;

List->tail = elem;

List->size++;

}

}

iFile.close();

} else {

cout << "error opening file" << endl;

}

return List;

}

int main() {

dblLinkedList\* list = createList();

printList(list);

addEl(list, 1, "a");

addEl(list, 2, "b");

addEl(list, 3, "c");

addEl(list, 4, "d");

addEl(list, 5, "e");

cout << "formed list: ";

printList(list);

int K;

cout << endl << "enter the number of elements: " << endl;

cin >> K;

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

char word[10];

int index;

cout << "enter the short word: ";

cin >> word;

cout << "enter the index of new element: ";

cin >> index;

addEl(list, index, word);

}

cout << "updated list: ";

printList(list);

cout << endl << "enter the number of elements to delete: ";

cin >> K;

if (K > list->size) {

cout << "there are not many elements in the list" << endl;

} else {

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

int index;

cout << endl << "enter the index of element to delete: ";

cin >> index;

deleteEl(list, index);

cout << "after deleting: ";

printList(list);

}

}

write\_to\_file(list, "dbllinkedlist.txt");

destroyList(list);

cout << endl << "after destroying: ";

printList(list);

list = restore(createList(), "dbllinkedlist.txt");

cout << "after restoring: ";

printList(list);

destroyList(list);

return 0;

}

Завдання Algo 5 :

#include <iostream>

using namespace std;

int main() {

int N, M, x, y;

cin >> N >> M;

cin >> y >> x;

int katet\_x1 = M - x + 1;

int katet\_x2 = x;

int katet\_y1 = N - y + 1;

int katet\_y2 = y;

int maxValue = max(katet\_x1, katet\_x2) + max(katet\_y1, katet\_y2) - 2;

x--;

y--;

int m[1000000] = {0};

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

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

m[j \* M + i] = maxValue - abs(x - i) - abs(y - j);

}

}

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

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

cout << m[j \* M + i] << " ";

}

cout << "\n";

}

return 0;

}

Завдання Algo 78

#include <iostream>

#include <vector>

template <typename T>

class DoublyLinkedList {

private:

class Node {

public:

T data;

Node\* prev;

Node\* next;

Node(const T& value) : data(value), prev(nullptr), next(nullptr) {}

};

Node\* head;

Node\* tail;

int size;

public:

DoublyLinkedList() : head(nullptr), tail(nullptr), size(0) {}

void insert(int index, int N, const T\* values) {

if (index < 0 || index > size) {

return;

}

Node\* newNode = nullptr;

Node\* current = nullptr;

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

newNode = new Node(values[i]);

if (head == nullptr) {

head = newNode;

tail = head;

} else if (index == 0) {

newNode->next = head;

head->prev = newNode;

head = newNode;

} else if (index == size) {

tail->next = newNode;

newNode->prev = tail;

tail = newNode;

} else {

current = head;

for (int j = 0; j < index - 1; ++j) {

current = current->next;

}

newNode->next = current->next;

newNode->prev = current;

current->next->prev = newNode;

current->next = newNode;

}

++size;

++index;

}

}

void erase(int index, int n) {

if (index < 0 || index >= size || n <= 0) {

return;

}

Node\* current = head;

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

current = current->next;

}

while (n > 0 && current != nullptr) {

Node\* nextNode = current->next;

if (current->prev != nullptr) {

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

} else {

head = current->next;

}

if (current->next != nullptr) {

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

} else {

tail = current->prev;

}

delete current;

--size;

--n;

current = nextNode;

}

}

int getSize() const {

return size;

}

T get(int index) const {

if (index < 0 || index >= size) {

return T();

}

Node\* current = head;

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

current = current->next;

}

return current->data;

}

void set(int index, const T& value) {

if (index < 0 || index >= size) {

return;

}

Node\* current = head;

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

current = current->next;

}

current->data = value;

}

void print() const {

Node\* current = head;

while (current != nullptr) {

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

current = current->next;

}

std::cout << std::endl;

}

~DoublyLinkedList() {

while (head != nullptr) {

Node\* temp = head;

head = head->next;

delete temp;

}

}

};

int main() {

int Q;

std::cin >> Q;

DoublyLinkedList<int> list;

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

std::string identifier;

std::cin >> identifier;

if (identifier == "insert") {

int index, N;

std::cin >> index >> N;

std::vector<int> values(N);

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

std::cin >> values[j];

}

list.insert(index, N, values.data());

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

int index, n;

std::cin >> index >> n;

list.erase(index, n);

} else if (identifier == "size") {

std::cout << list.getSize() << std::endl;

} else if (identifier == "get") {

int index;

std::cin >> index;

std::cout << list.get(index) << std::endl;

} else if (identifier == "set") {

int index, value;

std::cin >> index >> value;

list.set(index, value);

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

list.print();

}

}

return 0;

}

Завдання Class practice

Завдання №1

#include <iostream>

// Визначення структури Node

struct Node {

int data;

Node\* next;

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

};

// Метод реверсу для зв'язаного списку

Node\* reverse(Node\* head) {

Node\* prev = nullptr;

Node\* current = head;

Node\* next = nullptr;

while (current != nullptr) {

next = current->next; // Зберігаємо наступний вузол

current->next = prev; // Робимо зворотній зв'язок

// Пересуваємо покажчики для наступної ітерації

prev = current;

current = next;

}

// Новий початок списку - останній вузол

head = prev;

return head;

}

// Допоміжна функція для виведення списку

void printList(Node\* head) {

while (head != nullptr) {

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

head = head->next;

}

std::cout << std::endl;

}

int main() {

// Створення зразка списку: 1 -> 2 -> 3 -> 4 -> 5

Node\* head = new Node(1);

head->next = new Node(2);

head->next->next = new Node(3);

head->next->next->next = new Node(4);

head->next->next->next->next = new Node(5);

std::cout << "Original list: ";

printList(head);

// Виклик методу реверсу

head = reverse(head);

std::cout << "Reversed list: ";

printList(head);

return 0;

}

Завдання №2

#include <iostream>

// Визначення структури Node

struct Node {

int data;

Node\* next;

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

};

// Функція порівняння списків

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

while (h1 != nullptr && h2 != nullptr) {

// Порівняння даних у вузлах

if (h1->data != h2->data) {

return false;

}

// Перехід до наступних вузлів

h1 = h1->next;

h2 = h2->next;

}

// Порівняння довжин списків

return (h1 == nullptr && h2 == nullptr);

}

int main() {

// Створення першого списку: 1 -> 2 -> 3

Node\* head1 = new Node(1);

head1->next = new Node(2);

head1->next->next = new Node(3);

// Створення другого списку: 1 -> 2 -> 3

Node\* head2 = new Node(1);

head2->next = new Node(2);

head2->next->next = new Node(3);

// Порівняння списків

if (compare(head1, head2)) {

std::cout << "Lists are equal." << std::endl;

} else {

std::cout << "Lists are not equal." << std::endl;

}

// Додавання додаткового елементу до другого списку

head2->next->next->next = new Node(4);

// Повторне порівняння списків

if (compare(head1, head2)) {

std::cout << "Lists are equal." << std::endl;

} else {

std::cout << "Lists are not equal." << std::endl;

}

return 0;

}

Завдання №3

#include <iostream>

// Визначення структури Node

struct Node {

int data;

Node\* next;

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

};

// Функція додавання двох чисел, представлених у вигляді списків

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

Node\* result = nullptr; // Результат додавання

Node\* current = nullptr; // Поточний вузол в результуючому списку

int carry = 0; // Перенос для наступного розряду

// Поки є числа для додавання або є перенос

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

// Обчислення суми та переносу

int sum = (n1 ? n1->data : 0) + (n2 ? n2->data : 0) + carry;

carry = sum / 10;

// Створення нового вузла для результуючого списку

Node\* newNode = new Node(sum % 10);

// Додавання вузла до результуючого списку

if (result == nullptr) {

result = newNode;

current = result;

} else {

current->next = newNode;

current = newNode;

}

// Перехід до наступних вузлів

if (n1) n1 = n1->next;

if (n2) n2 = n2->next;

}

return result;

}

// Допоміжна функція для виведення списку

void printList(Node\* head) {

while (head != nullptr) {

std::cout << head->data;

head = head->next;

if (head != nullptr) {

std::cout << " -> ";

}

}

std::cout << std::endl;

}

int main() {

// Перше число: 1234

Node\* num1 = new Node(4);

num1->next = new Node(3);

num1->next->next = new Node(2);

num1->next->next->next = new Node(1);

// Друге число: 5678

Node\* num2 = new Node(8);

num2->next = new Node(7);

num2->next->next = new Node(6);

num2->next->next->next = new Node(5);

// Додавання двох чисел

Node\* sum = add(num1, num2);

// Виведення результату

std::cout << "Sum: ";

printList(sum);

return 0;

}

Завдання №4

#include <iostream>

// Визначення структури TreeNode

struct TreeNode {

int data;

TreeNode\* left;

TreeNode\* right;

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

};

// Функція для віддзеркалення бінарного дерева

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

if (root == nullptr) {

return nullptr;

}

// Створення нового вузла з оберненими лівою і правою вітками

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

mirroredRoot->left = create\_mirror\_flip(root->right);

mirroredRoot->right = create\_mirror\_flip(root->left);

return mirroredRoot;

}

// Функція для виведення бінарного дерева (в порядку обходу)

void print\_tree(TreeNode\* root) {

if (root == nullptr) {

return;

}

print\_tree(root->left);

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

print\_tree(root->right);

}

int main() {

// Створення бінарного дерева: 1

// / \

// 2 3

// / \

// 4 5

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);

// Віддзеркалення бінарного дерева

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

// Виведення вихідного та віддзеркаленого дерева

std::cout << "Original Tree: ";

print\_tree(root);

std::cout << std::endl;

std::cout << "Mirrored Tree: ";

print\_tree(mirroredRoot);

std::cout << std::endl;

return 0;

}

Завдання №5

#include <iostream>

// Визначення структури TreeNode

struct TreeNode {

int data;

TreeNode\* left;

TreeNode\* right;

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

};

// Функція для запису суми підвузлів кожному батьківському вузлу

void tree\_sum(TreeNode\* root) {

if (root == nullptr) {

return;

}

// Рекурсивний виклик для лівого та правого піддерева

tree\_sum(root->left);

tree\_sum(root->right);

// Якщо вузол не є листком, обчислити та записати суму підвузлів

if (root->left != nullptr) {

root->data += root->left->data;

}

if (root->right != nullptr) {

root->data += root->right->data;

}

}

// Функція для виведення бінарного дерева (в порядку обходу)

void print\_tree(TreeNode\* root) {

if (root == nullptr) {

return;

}

print\_tree(root->left);

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

print\_tree(root->right);

}

int main() {

// Створення бінарного дерева: 1

// / \

// 2 3

// / \

// 4 5

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);

// Запис суми підвузлів кожному батьківському вузлу

tree\_sum(root);

// Виведення результату

std::cout << "Tree with sums: ";

print\_tree(root);

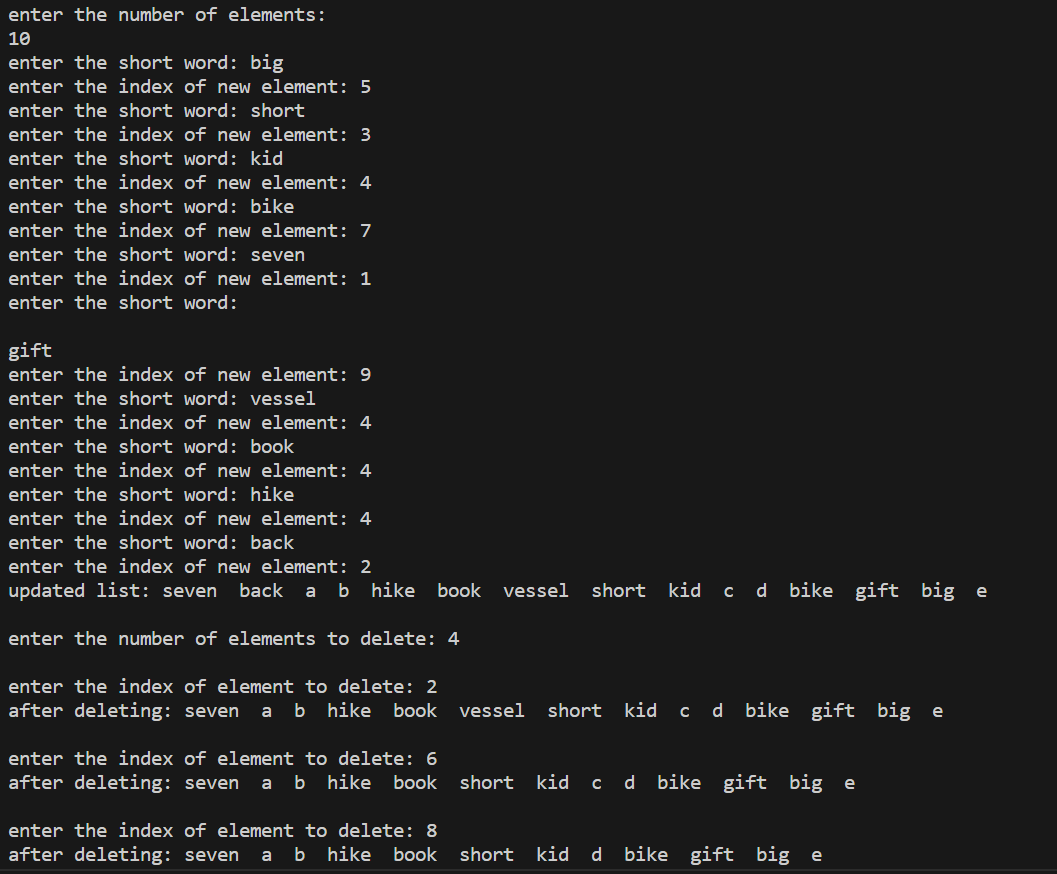
std::cout << std::endl;

return 0;

}

**5. Результати виконання завдань:**

Завдання Vns 10 :



*Figure 3.*

Завдання №2

<https://algotester.com/en/ProblemSolution/Display/1527273>

Завдання №3

<https://algotester.com/en/ProblemSolution/Display/1527680>

Завдання Class practice :

prac1-



*Figure 4.*

prac2-



*Figure 5.*

prac3-



*Figure 6.*

prac4-



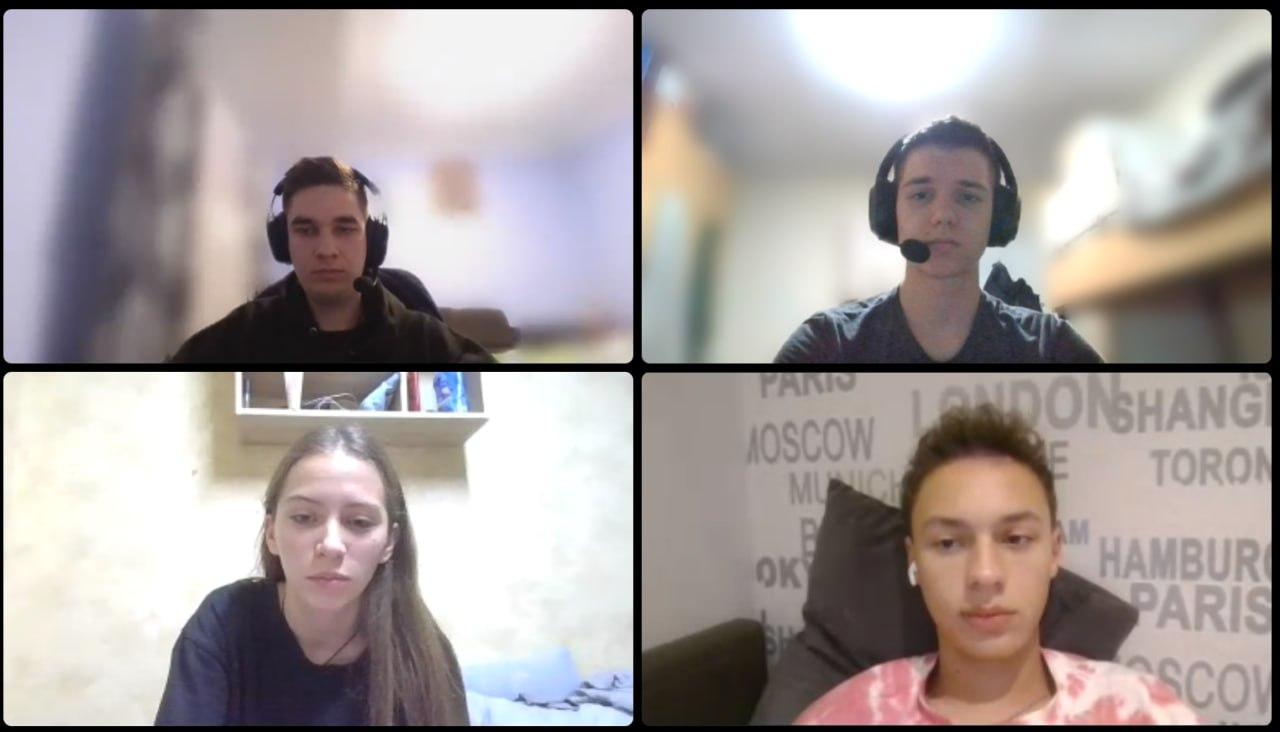
*Figure 7.*

prac5-



*Figure 8.*

**6. Кооперація з командою:**



*Figure 9.*

**Висновки:**

Дізнався та навчився працювати з динамічними структурами (Чергами, Стеками, Списками, Деревом), та алгоритмами обробки динамічних структур.