Міністерство освіти і науки України
Національний університет «Львівська політехніка»
Кафедра систем штучного інтелекту

Звіт
про виконання лабораторних та практичних робіт блоку № 6

з дисципліни: «Основи програмування»

Виконав:

Студент групи ШІ-11

Лопатін Володимир Дмитрович

Львів 2024

Тема:

Динамічні структури (Черга, Стек, Списки, Дерево). Алгоритми обробки динамічних структур. Поняття класу та його застосування

Мета роботи:

Ознайомитися з принципами реалізації та використання динамічних структур даних (черга, стек, списки, дерево) у програмуванні. Розвинути навички проектування алгоритмів для обробки динамічних структур, а також засвоїти поняття класу, принципи його реалізації та застосування для створення об'єктно-орієнтованих рішень.

Теоретичні відомості:

- Динамічні структури даних
- Робота з динамічними структурами
- Поняття класу
- Практичне застосування класів в ООП

Динамічні структури даних:

Не був знайомий, на парах отримав базове поняття, дорозібрався через різні ресурси.
Витрачено 40 хв.

Робота з динамічними структурами:

Уперше зіткнувся в цьому епіку.
Витратив 1 годину.

Поняття класу:

Був знайомий, проте потрібно було нагадати.

Витратив 45 хвилин.

Практичне застосування класів в ООП:

Мав певне уявленняЮ але на практиці кл=раще зрозумів.

Для повного розуміння потрібно було 45 хвилин.

Виконання роботи:

1) Опрацювання завдання та вимог до програм та серидовища:

Завдання №1 з ВНС

«Лабораторна № 10 варіант 19»

Потрібно створити клас двозв'зний список та додати методи додавання на початок, додавання в кінець, видалення за індексом, виводу елементів, збереження в файл та читання з файлу.

Завдання №2 з Algotester

«Лабораторна №5 варіант 1»

Потрібно створити програму, яка приймає стан дошки в байтах, виконує закляття над однією клітинкою n разів і потім виводить стан дошки в байтах.

Завдання №3

«Практичне завдання»

Однозв'язний список:

Реалізувати метод реверсу однозв’язного списку: Node* reverse(Node *head);
bool compare(Node *h1, Node *h2) – метод порівняння двох списків
Node* add(Node *n1, Node *n2) – додавання чисел через список

Бінарне дерево:

TreeNode *create_mirror_flip(TreeNode *root) – метод віддзеркалення дерева
void tree_sum(TreeNode *root) – метод знаходження кожному батьківському вузлу суми підвузлів.

Завдання №4

«Лабораторна 7-8 з Algotester»

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

Завдання №5

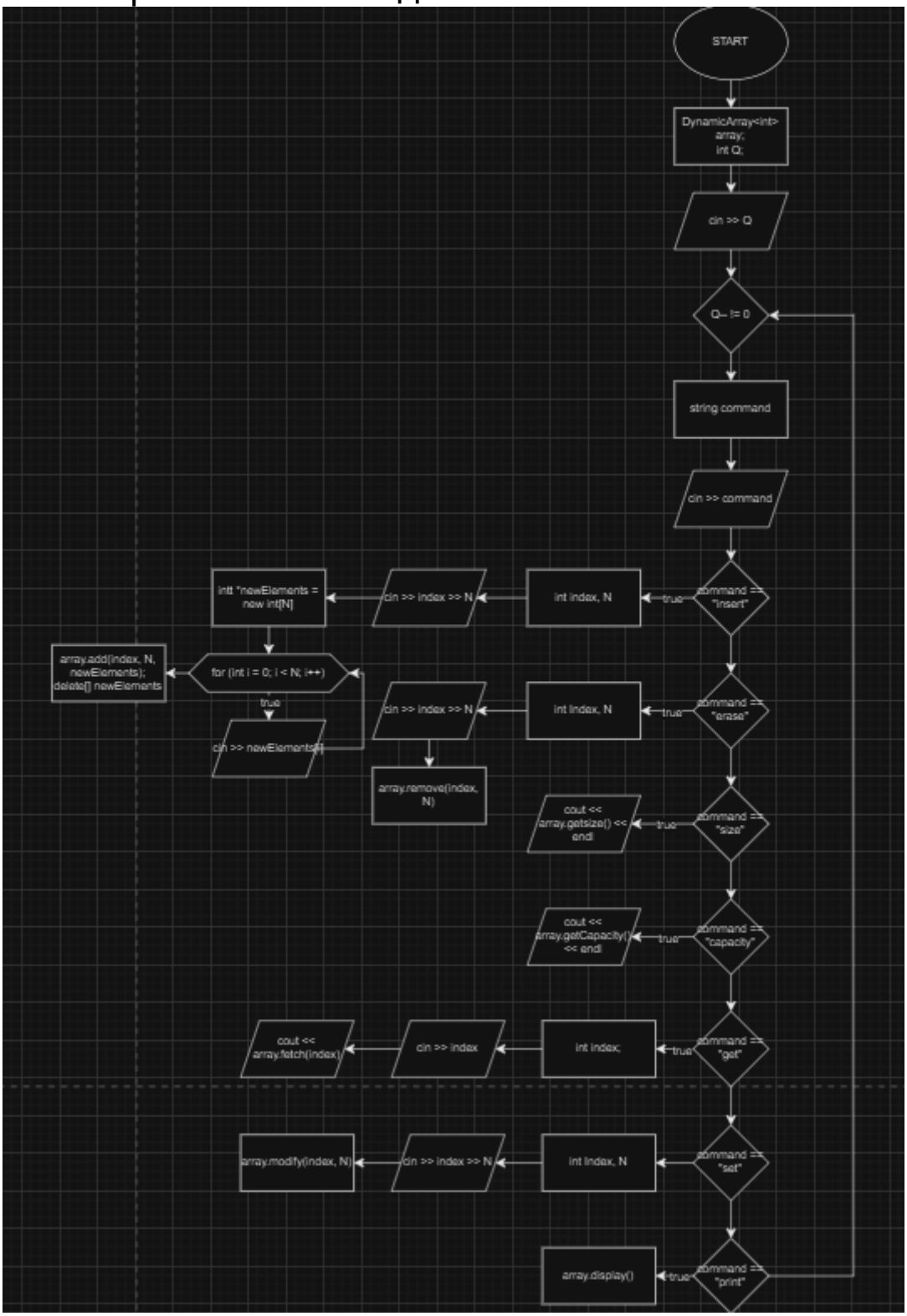
«Self practice з Algotester»

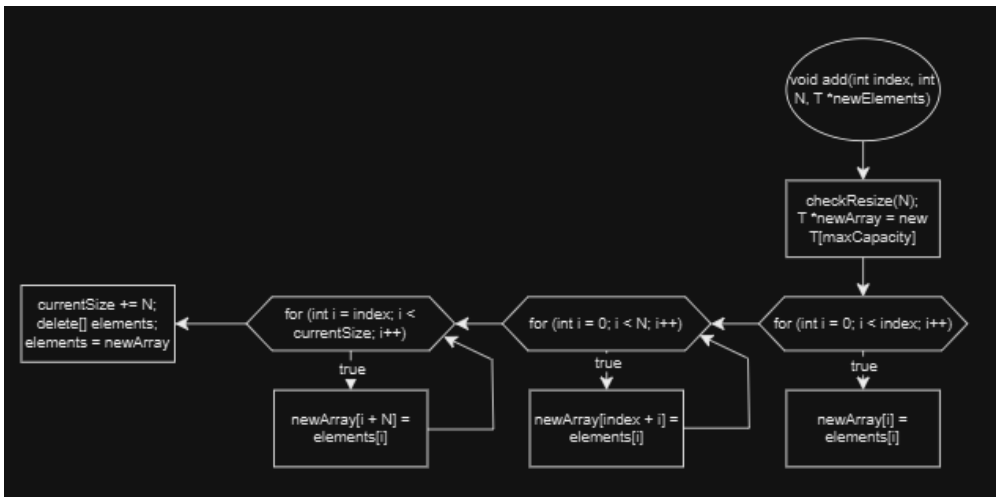
Задача полягала в тому, щоб закодувати рядок використовуючи букву і кількість разів її повторень.

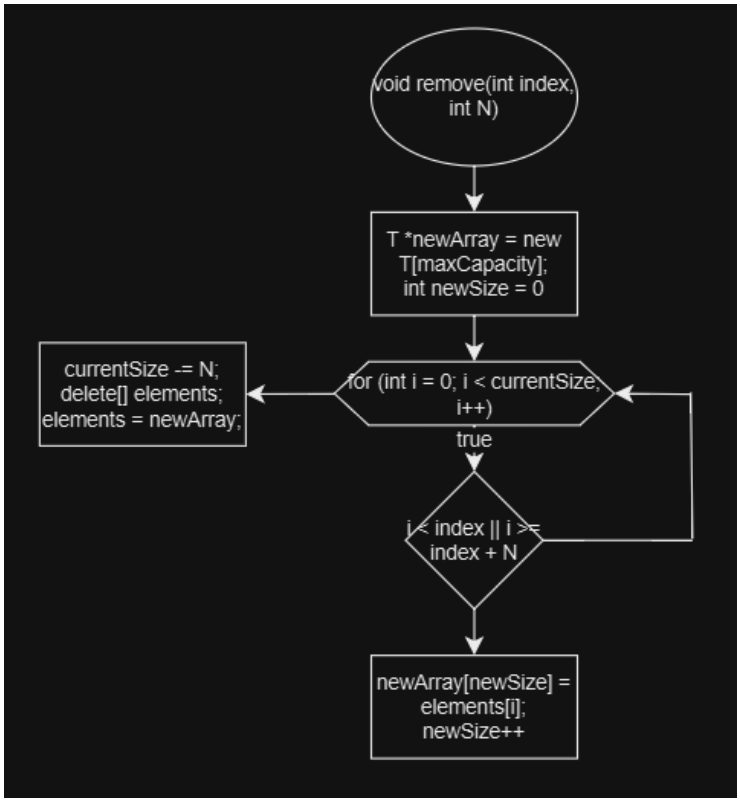
2) Дизайн та планова оцінка часу виконання завдань:

Завдання №4

Хотів зробити за 2 години.







1) Код програм з посиланням на зовнішні ресурси:

Завдання №1

```cpp
#include <iostream>
#include <fstream>
#include <cstring>

using namespace std;

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

    Node(const char* value) : prev(nullptr), next(nullptr) {
        data = new char[strlen(value) + 1];
        strcpy(data, value);
    }

    ~Node() {
        delete[] data;
    }
};

class DoublyLinkedList {
private:
    Node* head;
    Node* tail;

public:
    DoublyLinkedList() : head(nullptr), tail(nullptr) {}

    ~DoublyLinkedList() {
        clear();
    }

    void addToHead(const char* value) {
        Node* newNode = new Node(value);
        if (!head) {
            head = tail = newNode;
        } else {
            newNode->next = head;
            head->prev = newNode;
            head = newNode;
        }
    }
```

```
    void deleteByIndices(int index, int k) {
        Node* temp = head;
        for (int i = 0; temp != nullptr && i < index; i++) {
            temp = temp->next;
        }
        for (int i = 0; temp != nullptr && i < k; i++) {
            if (temp->prev) {
                temp->prev->next = temp->next;
            }
            if (temp->next) {
                temp->next->prev = temp->prev;
            }
            Node* toDelete = temp;
            temp = temp->next;
            delete toDelete;
        }
    }

    void printList() {
        Node* temp = head;
        while (temp) {
            cout << temp->data << " ";
            temp = temp->next;
        }
        cout << endl;
    }

    void clear() {
        while (head) {
            Node* temp = head;
            head = head->next;
            delete temp;
        }
        tail = nullptr;
    }

    void saveToFile(const char* filename) {
        ofstream outFile(filename);
        Node* temp = head;
        while (temp) {
            outFile << temp->data << endl;
            temp = temp->next;
        }
    }
```

```
    void loadFromFile(const char* filename) {
        ifstream inFile(filename);
        string line;
        while (getline(inFile, line)) {
            addToHead(line.c_str());
        }
    }
};

int main() {
    DoublyLinkedList list;
    list.addToHead("First");
    list.addToHead("Second");
    list.addToHead("Third");

    list.printList();

    list.deleteByIndices(1, 2);
    list.printList();

    list.saveToFile("list.txt");
    list.clear();

    list.loadFromFile("list.txt");
    list.printList();

    return 0;
}
```

Завдання №2

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

int main() {
    unsigned long long a;
    int N;
    cin >> a;
    cin >> N;
    if (N < 0 || N > 1000) {
        return 0;
    }

    for (int i = 0; i < N; ++i) {
        int Ri, Ci;
        cin >> Ri >> Ci;
        if (Ri < 1 || Ri > 8 || Ci < 1 || Ci > 8) {
            return 0;
        }

        unsigned long long row_mask = 0xFFULL << (8 * (Ri - 1));
        unsigned long long col_mask = 0;
        for (int j = 0; j < 8; ++j) {
            col_mask |= (1ULL << ((Ci - 1) + j * 8));
        }

        a ^= (row_mask | col_mask);
    }

    cout << a << endl;
    return 0;
}
```

Завдання №3

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

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

void printList(Node* head) {
    while (head) {
        cout << head->data;
        if (head->next) cout << " -> ";
        head = head->next;
    }
    cout << endl;
}

Node* reverse(Node* head) {
    Node* prev = nullptr;
    Node* current = head;
    while (current) {
        Node* nextNode = current->next;
        current->next = prev;
        prev = current;
        current = nextNode;
    }
    return prev;
}

bool compare(Node* h1, Node* h2) {
    while (h1 && h2) {
        if (h1->data != h2->data) return false;
        h1 = h1->next;
        h2 = h2->next;
    }
    return h1 == nullptr && h2 == nullptr;
}

Node* add(Node* n1, Node* n2) {
    Node* result = nullptr;
    Node* tail = nullptr;
    int carry = 0;
    while (n1 || n2 || carry) {
        int sum = carry;
```

```
        if (n1) {
            sum += n1->data;
            n1 = n1->next;
        }
        if (n2) {
            sum += n2->data;
            n2 = n2->next;
        }
        carry = sum / 10;
        int digit = sum % 10;
        Node* newNode = new Node(digit);
        if (!result) {
            result = tail = newNode;
        } else {
            tail->next = newNode;
            tail = newNode;
        }
    }
    return result;
}

Node* createListFromArray(int arr[], int size) {
    Node* head = nullptr;
    Node* tail = nullptr;
    for (int i = 0; i < size; ++i) {
        Node* newNode = new Node(arr[i]);
        if (!head) {
            head = tail = newNode;
        } else {
            tail->next = newNode;
            tail = newNode;
        }
    }
    return head;
}

int main() {
    int arr1[] = {1, 2, 3, 4, 5};
    Node* list1 = createListFromArray(arr1, 5);
    cout << "Original List: ";
    printList(list1);
    list1 = reverse(list1);
    cout << "Reversed List: ";
```

```
int main() {
    int arr1[] = {1, 2, 3, 4, 5};
    Node* list1 = createListFromArray(arr1, 5);
    cout << "Original List: ";
    printList(list1);
    list1 = reverse(list1);
    cout << "Reversed List: ";
    printList(list1);

    int arr2[] = {1, 2, 3, 4, 5};
    Node* list2 = createListFromArray(arr2, 5);
    cout << "Lists are " << (compare(list1, list2) ? "equal" : "not equal") << "." << endl;

    int num1[] = {9, 9, 9};
    int num2[] = {1};
    Node* number1 = createListFromArray(num1, 3);
    Node* number2 = createListFromArray(num2, 1);
    cout << "Number 1: ";
    printList(number1);
    cout << "Number 2: ";
    printList(number2);
    Node* sum = add(number1, number2);
    cout << "Sum: ";
    printList(sum);
    return 0;
}
```

```cpp
#include <iostream>
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) return nullptr;

    TreeNode* left = create_mirror_flip(root->left);
    TreeNode* right = create_mirror_flip(root->right);

    root->left = right;
    root->right = left;

    return root;
}

void tree_sum(TreeNode* root) {
    if (!root) return;

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

    if (root->left) root->data += root->left->data;
    if (root->right) root->data += root->right->data;
}

void printTree(TreeNode* root) {
    if (!root) return;
    cout << root->data << " ";
    printTree(root->left);
    printTree(root->right);
}

TreeNode* createTreeFromArray(int arr[], int size) {
    if (size == 0) return nullptr;
    TreeNode* root = new TreeNode(arr[0]);
    TreeNode* nodes[size];
    nodes[0] = root;
```

```
    for (int i = 1; i < size; ++i) {
        TreeNode* newNode = new TreeNode(arr[i]);
        if (i % 2 == 1) {
            nodes[(i - 1) / 2]->left = newNode;
        } else {
            nodes[(i - 2) / 2]->right = newNode;
        }
        nodes[i] = newNode;
    }

    return root;


int main() {
    int arr[] = {1, 2, 3, 4, 5, 6, 7};
    TreeNode* root = createTreeFromArray(arr, 7);

    cout << "Original Tree: ";
    printTree(root);
    cout << endl;

    TreeNode* mirroredTree = create_mirror_flip(root);
    cout << "Mirrored Tree: ";
    printTree(mirroredTree);
    cout << endl;

    tree_sum(mirroredTree);
    cout << "Tree after sum of subtrees: ";
    printTree(mirroredTree);
    cout << endl;

    return 0;
```

Завдання №4

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

template <class T>
class DynamicArray {
private:
    T *elements;

    void checkResize(int N) {
        while (currentSize + N >= maxCapacity) {
            maxCapacity *= 2;
        }
    }

public:
    int currentSize;
    int maxCapacity;

    DynamicArray() {
        currentSize = 0;
        maxCapacity = 1;
        elements = new T[1];
    }

    void add(int index, int N, T *newElements) {
        checkResize(N);

        T *newArray = new T[maxCapacity];

        for (int i = 0; i < index; i++) {
            newArray[i] = elements[i];
        }

        for (int i = 0; i < N; i++) {
            newArray[index + i] = newElements[i];
        }

        for (int i = index; i < currentSize; i++) {
            newArray[i + N] = elements[i];
        }

        currentSize += N;
        delete[] elements;
        elements = newArray;
    }
```

```
    void remove(int index, int N) {
        T *newArray = new T[maxCapacity];
        int newSize = 0;

        for (int i = 0; i < currentSize; i++) {
            if (i < index || i >= index + N) {
                newArray[newSize] = elements[i];
                newSize++;
            }
        }

        currentSize -= N;
        delete[] elements;
        elements = newArray;
    }

    int getSize() const {
        return currentSize;
    }

    int getCapacity() const {
        return maxCapacity;
    }

    T fetch(int index) const {
        return elements[index];
    }

    void modify(int index, T value) {
        elements[index] = value;
    }

    void display(const string &separator = " ") const {
        for (int i = 0; i < currentSize; i++) {
            cout << elements[i];
            if (i < currentSize - 1) {
                cout << separator;
            }
        }
        cout << endl;
    }
};
```

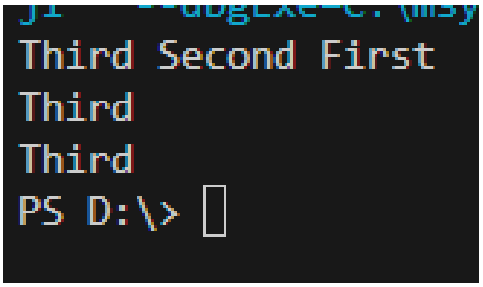
```cpp
int main() {
    DynamicArray<int> array;
    int Q;
    cin >> Q;
    while (Q--) {
        string command;
        cin >> command;
        if (command == "insert") {
            int index, N;
            cin >> index >> N;
            int *newElements = new int[N];

            for (int i = 0; i < N; i++) {
                cin >> newElements[i];
            }

            array.add(index, N, newElements);
            delete[] newElements;
        }
        else if (command == "erase") {
            int index, N;
            cin >> index >> N;
            array.remove(index, N);
        }
        else if (command == "size") {
            cout << array.getSize() << endl;
        }
        else if (command == "capacity") {
            cout << array.getCapacity() << endl;
        }
        else if (command == "get") {
            int index;
            cin >> index;
            cout << array.fetch(index) << endl;
        }
        else if (command == "set") {
            int index, N;
            cin >> index >> N;
            array.modify(index, N);
        }
        else if (command == "print") {
            array.display();
        }
    }
    return 0;
```

Завдання №5

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

using namespace std;

int main() {
    string s;
    cin >> s;

    if (s.length() < 1 || s.length() > 100000) {
        return 0;
    }

    for (char c : s) {
        if (!isupper(c)) {
            return 0;
        }
    }

    deque<string> result;

    int n = s.size();
    int count = 1;

    for (int i = 1; i < n; ++i) {
        if (s[i] == s[i - 1]) {
            ++count;
        } else {
            result.push_back(to_string(count) + s[i - 1]);
            count = 1;
        }
    }
    result.push_back(to_string(count) + s[n - 1]);

    for (const auto& segment : result) {
        cout << segment;
    }

    return 0;
}
```

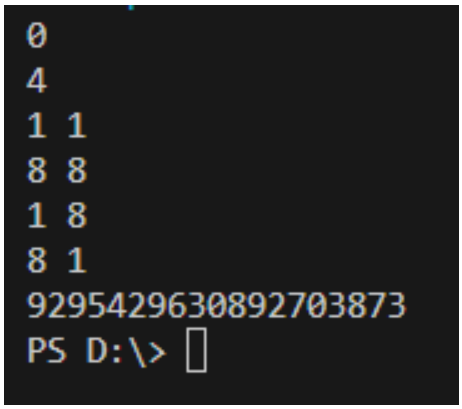
2) Результати виконання завдань, тестування та фактично затрачений час

Завдання №1



Витратив 40 хвилин.

Завдання №2



Витратив на завдання близько 1 години.

Завдання №3

```
Original List: 1 -> 2 -> 3 -> 4 -> 5
Reversed List: 5 -> 4 -> 3 -> 2 -> 1
Lists are not equal.
Number 1: 9 -> 9 -> 9
Number 2: 1
Sum: 0 -> 0 -> 0 -> 1
PS D:\>
```

```
Original Tree: 1 2 4 5 3 6 7
Mirrored Tree: 1 3 7 6 2 5 4
Tree after sum of subtrees: 28 16 7 6 11 5 4
PS D:\>
```

На це завдання пішло 45 хвилин.

Завдання №4

```
2
set 1 4
get 1
4
PS D:\>
```

Витратив на завдання приблизно 60 хвилин.

Завдання №5

```
SSSSSHHGGGFFFVVVYYYY
5S2H3G3F3V4Y
PS D:\>
```

На завдання пішло пів години.

Висновки:

У ході виконання лабораторної роботи було досліджено та освоєно основи роботи з динамічними структурами даних, такими як черга, стек, списки та дерево. Вивчені алгоритми обробки цих структур, а також важливість вибору відповідної структури для конкретної задачі. Завдяки застосуванню об’єктно-орієнтованого підходу та створенню класів для реалізації зазначених структур, було поглиблено розуміння принципів інкапсуляції, абстракції та модульності в програмуванні. Отримані знання дозволяють ефективно використовувати ці структури даних для вирішення широкого спектра практичних задач, а також сприяють розвитку навичок розробки оптимізованих алгоритмів.