Міністерство освіти і науки України

Національний університет «Львівська політехніка»

Кафедра систем штучного інтелекту

Звіт

про виконання лабораторних та практичних робіт блоку № 6

На тему: «Динамічні структури (Черга, Стек, Списки, Дерево). Алгоритми обробки динамічних структур.»

з дисципліни: «Основи програмування»

до:

ВНС Лабораторної Роботи № 10

Алготестер Лабораторних Робіт № 5, 7-8

Практичних Робіт до блоку № 6

Виконав:

Студент групи ШІ-11

Маркевич Владислав

Львів 2024

Тема роботи

Динамічні структури, види динамічних структур, їх використання, алгоритми їх обробки.

Мета роботи

1. Навчитись створювати та використовувати динамічні структури, такі як: Черга, Стек, Списки, Дерево.
2. Навчитись виконувати алгоритми обробки динамічний структур.

Теоретичні відомості

1. Перенавантаження операторі виводу

   https://acode.com.ua/urok-141-perevantazhennya-operatoriv-vvodu-i-vyvodu/#toc-0

2. Класи

   https://acode.com.ua/urok-121-klasy-ob-yekty-i-metody/#toc-1

   https://acode.com.ua/urok-183-shablony-klasiv/

3. Черга

   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/

4. Стек

   https://www.bestprog.net/uk/2019/09/18/c-the-concept-of-stack-operations-on-the-stack-an-example-implementation-of-the-stack-as-a-dynamic-array-ua/

5. Списки

   https://codelessons.dev/ru/spisok-list-v-s-polnyj-material/

6. Дерево

https://www.bing.com/ck/a?!&&p=d92fe6ba3879a4e47f751a99580f5f4fe2193328a2a4e32cef85f6aa2f6603bcJmltdHM9MTczMjc1MjAwMA&ptn=3&ver=2&hsh=4&fclid=1c78f629-a87f-6de6-3f44-e249a96d6cf2&psq=%d0%b4%d0%b5%d1%80%d0%b5%d0%b2%d0%b0+%d1%83+%d1%81%2b%2b&u=a1aHR0cHM6Ly9wdXJlY29kZWNwcC5jb20vdWsvYXJjaGl2ZXMvMjQ4Mw&ntb=1

Виконання роботи

Завдання №1 -VNS lab 10 - варіант 7

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

using namespace std;

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

};

class doubleLinkedList
{
    private:
    Node* head;

    public:
    doubleLinkedList() : head(nullptr){}

    void EnterToTheEnd(int data){
        Node* newNode = new Node();
        (*newNode).data = data;
        (*newNode).next = nullptr;
        if (head == nullptr){
            (*newNode).prev = nullptr;
            head = newNode;
            return;
        }

        Node* temp = head;
        while((*temp).next != nullptr){
            temp = (*temp).next;
        }
        (*temp).next = newNode;
```

```
    void EnterToTheEnd(int data){

        Node* temp = head;
        while((*temp).next != nullptr){
            temp = (*temp).next;
        }
        (*temp).next = newNode;
        (*newNode).prev = temp;
    }

    void deleteBegin(){
        if(head == nullptr){
            cout<<"Лист наразі пустий"<<endl;
            return;
        }
        Node* temp = head;
        head = (*head).next;
        if(head != nullptr){
            (*head).prev = nullptr;
        }
        delete temp;
    }

    void display(){
        Node* temp = head;
        while (temp != nullptr){
            cout<<(*temp).data<<" ";
            temp = (*temp).next;
        }
        cout<<endl;
    }
    Node* getHead() const{
    return head;
}
void delList(){
    Node* temp = head;
```

```
void delList(){
    Node* temp = head;
    while(temp != nullptr){
        Node* next = (*temp).next;
        delete temp;
        temp = next;
    }
    head = nullptr;
}
void restore(const string& filename){
    ifstream file(filename);
    if(!file){
        cout<<"Файл не вдалось відкрити"<<endl;
    }
    int data;
    while(file >> data){
        EnterToTheEnd(data);
    }
    file.close();
}
};

void copytofile(const string& filename, Node* head){
    ofstream file(filename);
    if(!file){
        cout<<"Не вдалось відкрити файл";
        return;
    }
    Node* temp = head;
    while(temp != nullptr){
        file << (*temp).data<<" ";
```

```
void emptyList(const string& filename, Node* head){
    ofstream file(filename);
    if(!file){
        cout<<"Не вдалось відкрити файл";
        return;
    }
    if(head == nullptr){
        cout<<"Список порожній"<<endl;
    }
    else{Node* temp = head;
    while(temp != nullptr){
        file << (*temp).data<<" ";
        temp = (*temp).next;
    }
  }
    file.close();
}

int main(){
    doubleLinkedList list;
    list.EnterToTheEnd(10);
    list.EnterToTheEnd(20);
    list.EnterToTheEnd(20);
    cout<<"Початковий лист: "<<endl;
    list.display();

    list.deleteBegin();
    cout<<"Після видалення першого елементу: "<<endl;
    list.display();

    list.EnterToTheEnd(40);
    cout<<"Після добавлення елементу в кінець: "<<endl::
```

```
    string name;
    cin>>name;

    copytofile(name, list.getHead());

    list.delList();
    cout<<"Список видалено "<<endl;

    string secondname;
    cin>>secondname;
    emptyList(secondname, list.getHead());

    list.restore(name);
    cout<<"Відновленний список: "<<endl;
    list.display();

    list.delList();
    cout<<"Список знову видалено( ";

    list.display();
    return 0;

    }
```

```
Початковий лист:
10 20 20
Після видалення першого елементу:
20 20
Після добавлення елементу в кінець:
20 20 40
Second.txt
Список видалено
First.txt
Список порожній
Відновленний список:
20 20 40
Список знову видалено(
```

Завдання №2 Algotester lab 5 - варіант 3

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

int main() {
    int N, M, x, y;
    cin >> N >> M >> x >> y;
    x--;
    y--;

    vector<vector<int>> mount(N, vector<int>(M, -1));
    queue<pair<int, int>> q;
    q.push({x,y});
    mount[x][y] = 0;

    queue<pair<int, int>> que;
    que.push({x, y});
    mount[x][y] = max(x, N - 1 - x) + max(y, M - 1 - y);

    int directions[4][2] = {{-1, 0}, {1, 0}, {0, -1}, {0, 1}};

    while (!que.empty()) {
        auto [cx, cy] = que.front();
        que.pop();

        for (auto& d : directions) {
            int nx = cx + d[0], ny = cy + d[1];
            if (nx >= 0 && nx < N && ny >= 0 && ny < M && mount[nx][ny] == -1) {
                mount[nx][ny] = mount[cx][cy] - 1;
                que.push({nx, ny});
            }
        }
    }

    for (int i = 0; i < N; i++) {
        for (int j = 0; j < M; j++) {
            cout << mount[i][j] << " ";
```

```cpp
        for (int i = 0; i < N; i++) {
            for (int j = 0; j < M; j++) {
                cout << mount[i][j] << " ";
            }
            cout << endl;
        }

        return 0;
    }
```

```
3 4
2 2
1 2 1 0
2 3 2 1
1 2 1 0
```

Завдання №3 **A**lgotester lab 7-8 - варіант 2-3

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

template <class T>
class dynamicArray {
private:
    T *data;

public:
    int size;
    int capacity;



    dynamicArray() {
        this->size = 0;
        this->capacity = 1;
        this->data = new T[1];
    }

    void insert(int index, int chys, T *elements) {
        while (size + chys >= capacity)
            capacity *= 2;
        T *temp = new T[capacity];

        for (int i = 0; i < index; i++)
            temp[i] = data[i];

        for (int i = 0; i < chys; i++)
            temp[index + i] = elements[i];

        for (int i = index; i < size; i++)
            temp[i + chys] = data[i];

        this->size += chys;
        delete[] data;
        data = temp;
```

```
        T *temp = new T[capacity];
        int newSize = 0;
        for (int i = 0; i < this->size; i++)
        {
            if (i < index || i >= index + chys)
            {
                temp[newSize] = data[i];
                newSize++;
            }
        }
        this->size -= chys;
        delete[] data;
        data = temp;
    }

    T get(int index) {
        return this->data[index];
    }

    void set(int index, T value) {
        this->data[index] = value;
    }

    void print(const string &separator) {
        for (int i = 0; i < this->size; i++) {
            cout << data[i];
            if (i < size - 1) cout << separator;
        }
        cout << endl;
    }
};
```

```cpp
int main() {
    dynamicArray<int> array;

    int queryCount;
    cin >> queryCount;

    while (queryCount--) {


        string command;
        cin >> command;
        if (command == "insert") {
            int index, chys;
            cin >> index >> chys;
            int *elements = new int[chys];

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

            array.insert(index, chys, elements);
            delete[] elements;
        } else if (command == "erase") {
            int index, chys;
            cin >> index >> chys;
            array.erase(index, chys);
        } else if (command == "size") {
            cout << array.size << endl;
        } else if (command == "capacity") {
            cout << array.capacity << endl;
        } else if (command == "get") {
            int index;
            cin >> index;
            cout << array.get(index) << endl;
        } else if (command == "set") {
            int index, chys;
```

```
            cin >> index;
            cout << array.get(index) << endl;
        } else if (command == "set") {
            int index, chys;
            cin >> index >> chys;
            array.set(index, chys);
        } else if (command == "print") {
            array.print(" ");
        }
    }
    return 0;
}
```

```
12

size
0
capacity
1

insert 0 2
100 100

size
2
capacity
4

insert 0 2
102 102

size
4
capacity
8

insert 0 2
103 103

size
6
capacity
8

print
103 103 102 102 100 100
```

```cpp
#include <iostream>

using namespace std;

template<typename T>

class BinarySearchTree
{
private:
    struct Node
    {
        T data;
        Node* left;
        Node* right;
        Node(T value) : data(value), left(nullptr), right(nullptr) {}
    };

    Node* root;
    int treesize;

    void insert(Node*& node, T value){
        if (node == nullptr){
            node = new Node(value);
            treesize ++;
        }else if(value< node->data){
            insert(node->left,value);
        }else if(value > node->data){
            insert(node->right,value);
        }
    }
    bool contains(Node* node, T value) const {
        if (node == nullptr) {
            return false;
        } else if (value == node->data) {
            return true;
        } else if (value < node->data) {
            return contains(node->left, value);
```

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

    void print(Node* node) const {
        if (node != nullptr) {
            print(node->left);
            cout << node->data << " ";
            print(node->right);
        }
    }

    void clear(Node* node) {
        if (node != nullptr) {
            clear(node->left);
            clear(node->right);
            delete node;
        }
    }
public:
    BinarySearchTree() : root(nullptr), treesize(0) {}
    ~BinarySearchTree(){
        clear(root);
    }
void insert(T value){
```

```
51      void clear(Node* node) {
58  public:
59      BinarySearchTree() : root(nullptr), treesize(0) {}
60      ~BinarySearchTree(){
61          clear(root);
62      }
63  void insert(T value){
64      insert(root,value);
65  }
66  bool contains(T value) const{
67      return contains(root, value);
68  }
69
70  int size() const {
71      return treesize;
72  }
73  void print() const {
74      print(root);
75      cout<<endl;
76  }
77  };
78
79  int main(){
80      BinarySearchTree<int> bst;
81      int Q;
82      cin>>Q;
83      for(int i = 0; i<Q;i++){
84          string command;
85          cin>>command;
86          if(command == "insert"){
87              int value;
88              cin>>value;
89              bst.insert(value);
90          }else if(command == "contains"){
91              int value;
92              cin>>value;
```

```
        }else if(command == "contains"){
            int value;
            cin>>value;
            if(bst.contains(value)){
                cout<<"Yes"<<endl;
            }else{
                cout<<"No"<<endl;
            }
        }else if(command == "size"){
            cout<<bst.size()<<endl;
        }else if(command == "print"){
            bst.print();
        }
    }
    return 0;
}
```

```
11

size
0
insert 5
insert 4
print
4 5
insert 5
print
4 5
insert 1
print
1 4 5
contains 5
Yes
contains 0
No
size
3
```

Завдання №4 Class Practice Work

```cpp
#include <iostream>

using namespace std;

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

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

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

void insert(Node*& head, int value){
    Node* newNode = new Node(value);
    if(head == nullptr){
```

```
    if(head == nullptr){
        head = newNode;
    }else{
        Node* temp = head;
        while(temp->next != nullptr){
            temp = temp->next;
        }
        temp->next = newNode;
    }
}

bool compare(Node* h1,Node* h2){
    while(h1 != nullptr && h2 != nullptr){
        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 != nullptr || n2 != nullptr){
        int sum = carry;
        if(n1 != nullptr){
            sum += n1->data;
            n1 = n1->next;
        }
        if(n2 != nullptr){
            sum += n2->data;
```

```
            if(n2 != nullptr){
                n2 = n2->next;
            }
            carry = sum / 10;
            sum = sum % 10;

            Node* newNode = new Node(sum);
            if(result == nullptr){
                result = newNode;
                tail = result;
        }else{
            tail->next = newNode;
            tail = tail->next;
        }
        }
        if (carry > 0) {
            tail->next = new Node(carry);
        }
        return result;
    }

    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* newRoot = new TreeNode(root->data);
        newRoot->left = create_mirror_flip(root->right);
```

```
    }
    TreeNode* newRoot = new TreeNode(root->data);
    newRoot->left = create_mirror_flip(root->right);
    newRoot->right = create_mirror_flip(root->left);
    return newRoot;
}

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


TreeNode* insert(TreeNode* root, int value) {
    if (root == nullptr) {
        return new TreeNode(value);
    }
    if (value < root->data) {
        root->left = insert(root->left, value);
        return root;
    }else{
        root->right = insert(root->right, value);
    }
    return root;
}

int tree_sum(TreeNode* root) {
    if (root == nullptr) {
        return 0;
    }
    if (root->left == nullptr && root->right == nullptr) {
        return root->data;
```

```cpp
int tree_sum(TreeNode* root) {
    if (root == nullptr) {
        return 0;
    }
    if (root->left == nullptr && root->right == nullptr) {
        return root->data;
    }
    int leftSum = tree_sum(root->left);
    int rightSum = tree_sum(root->right);
    root->data = leftSum + rightSum;
    return root->data;
}

int main(){
    Node* head1 = nullptr;
    Node* head2 = nullptr;

    insert(head1,1);
    insert(head1,2);
    insert(head1,3);
    insert(head1,4);
    insert(head1,5);

    insert(head2,1);
    insert(head2,2);
    insert(head2,3);
    insert(head2,4);
    insert(head2,5);

    cout<<"Список 1:"<<endl;
    printList(head1);

    cout<<"Список 2:"<<endl;
    printList(head2);
```

```
    printList(head2);

    if(compare(head1,head2)){
        cout<<"Списки ідентичні"<<endl;
    }else{
        cout<<"Списки різні"<<endl;
    }
    head1 = reverse(head1);

    cout<<"Обернений перший список: "<<endl;
    printList(head1);

    if(compare(head1,head2)){
        cout<<"Списки однакові"<<endl;
    }else{
        cout<<"Списки різні"<<endl;
    }

    Node* result = add(head1,head2);

    cout<<"Сума додавання: ";
    printList(result);

    TreeNode* root = nullptr;

    root = insert(root, 4);
    root = insert(root, 2);
    root = insert(root, 6);
    root = insert(root, 1);
    root = insert(root, 3);
    root = insert(root, 5);
    root = insert(root, 7);
```

```
    root = insert(root, 7);

    cout << "Оригінальне дерево: ";
    printTree(root);
    cout << endl;


    tree_sum(root);

    cout << "Дерево після обчислення суми: ";
    printTree(root);
    cout << endl;


    return 0;
}
```

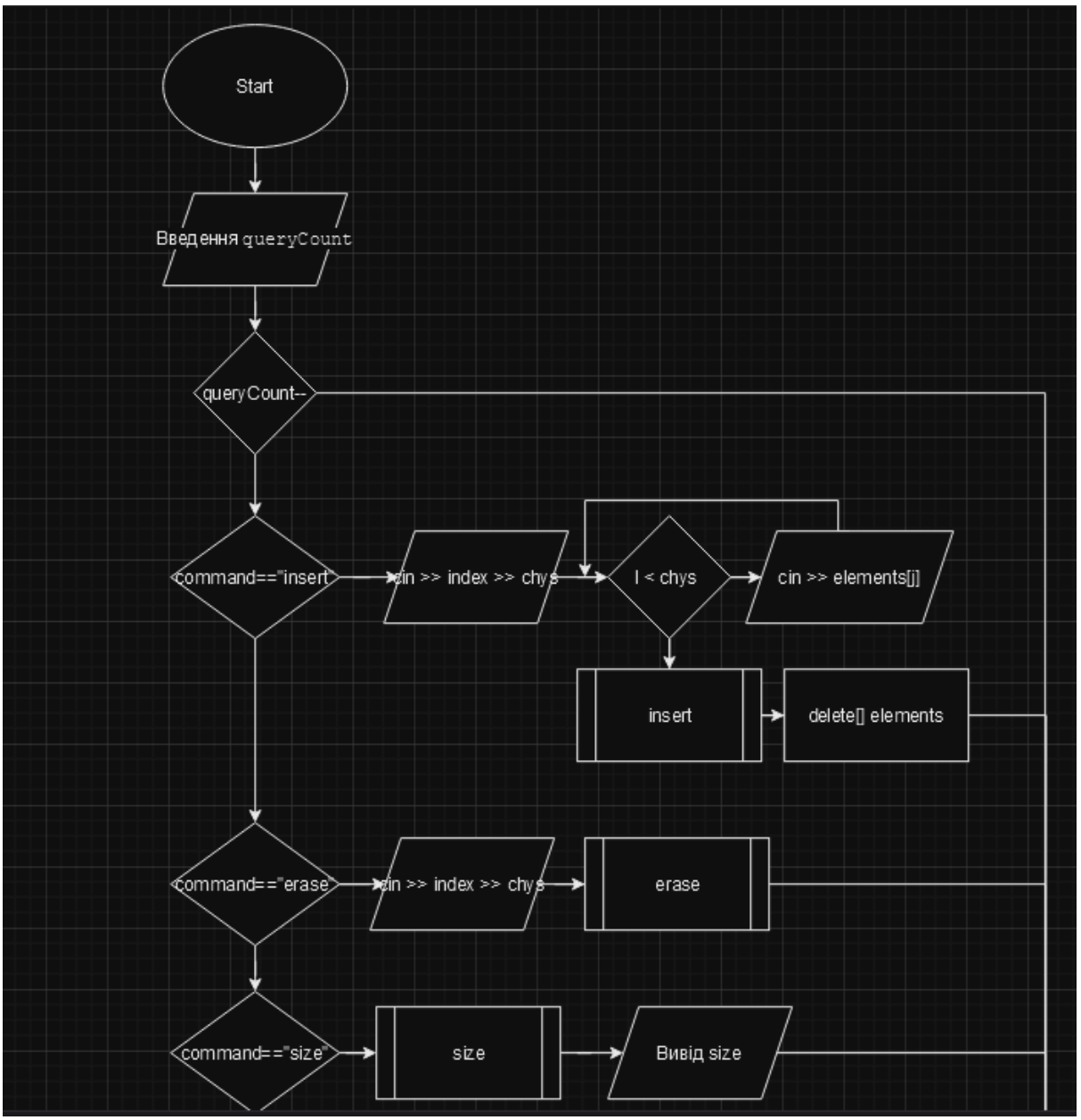
```
Список 1:
1 2 3 4 5
Список 2:
1 2 3 4 5
Списки ідентичні
Обернений перший список:
5 4 3 2 1
Списки різні
Сума додавання: 6 6 6 6 6
Оригінальне дерево: 1 2 3 4 5 6 7
Дерево після обчислення суми: 1 4 3 16 5 12 7
```

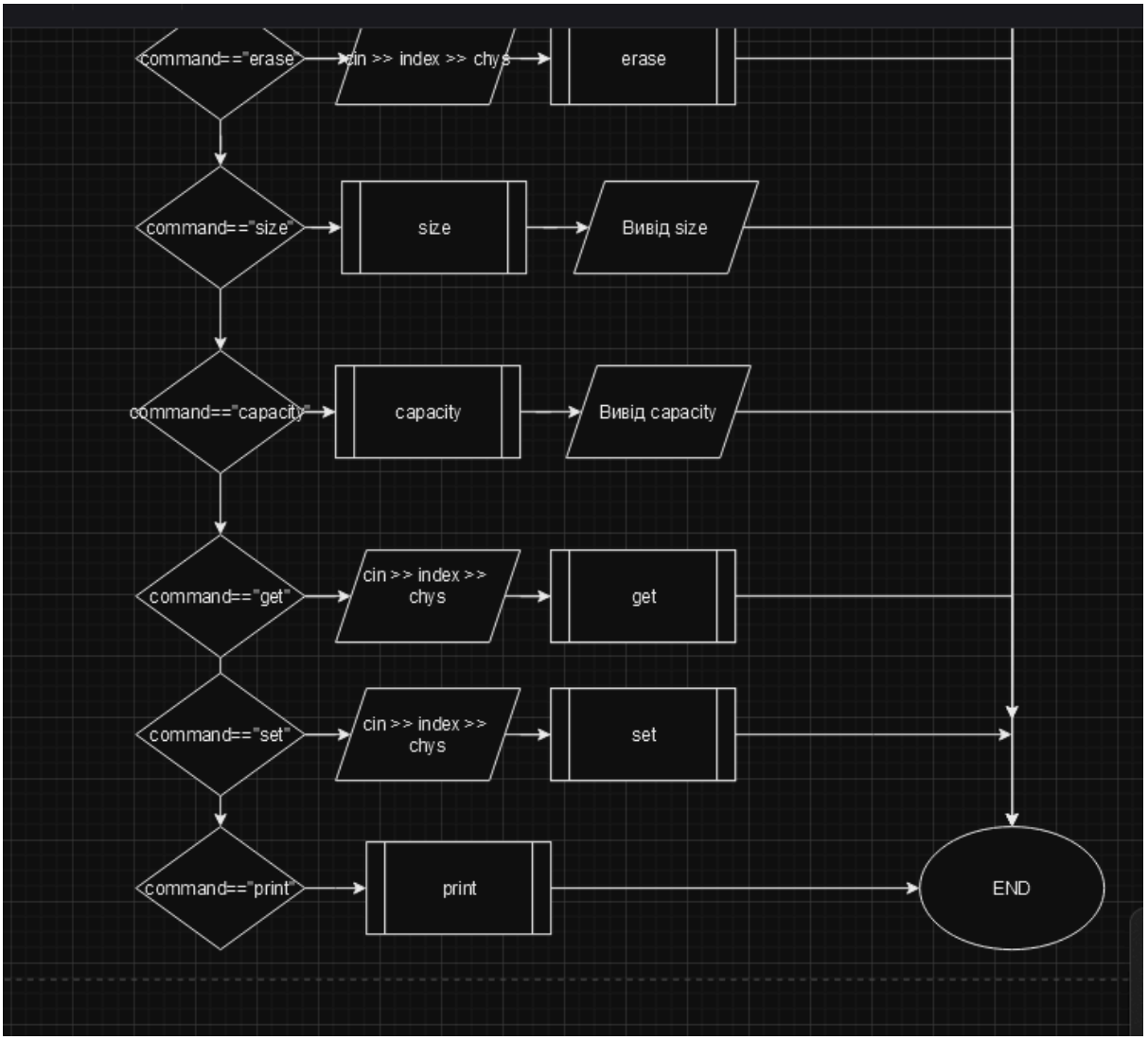
Завдання №5 Self Practice Work

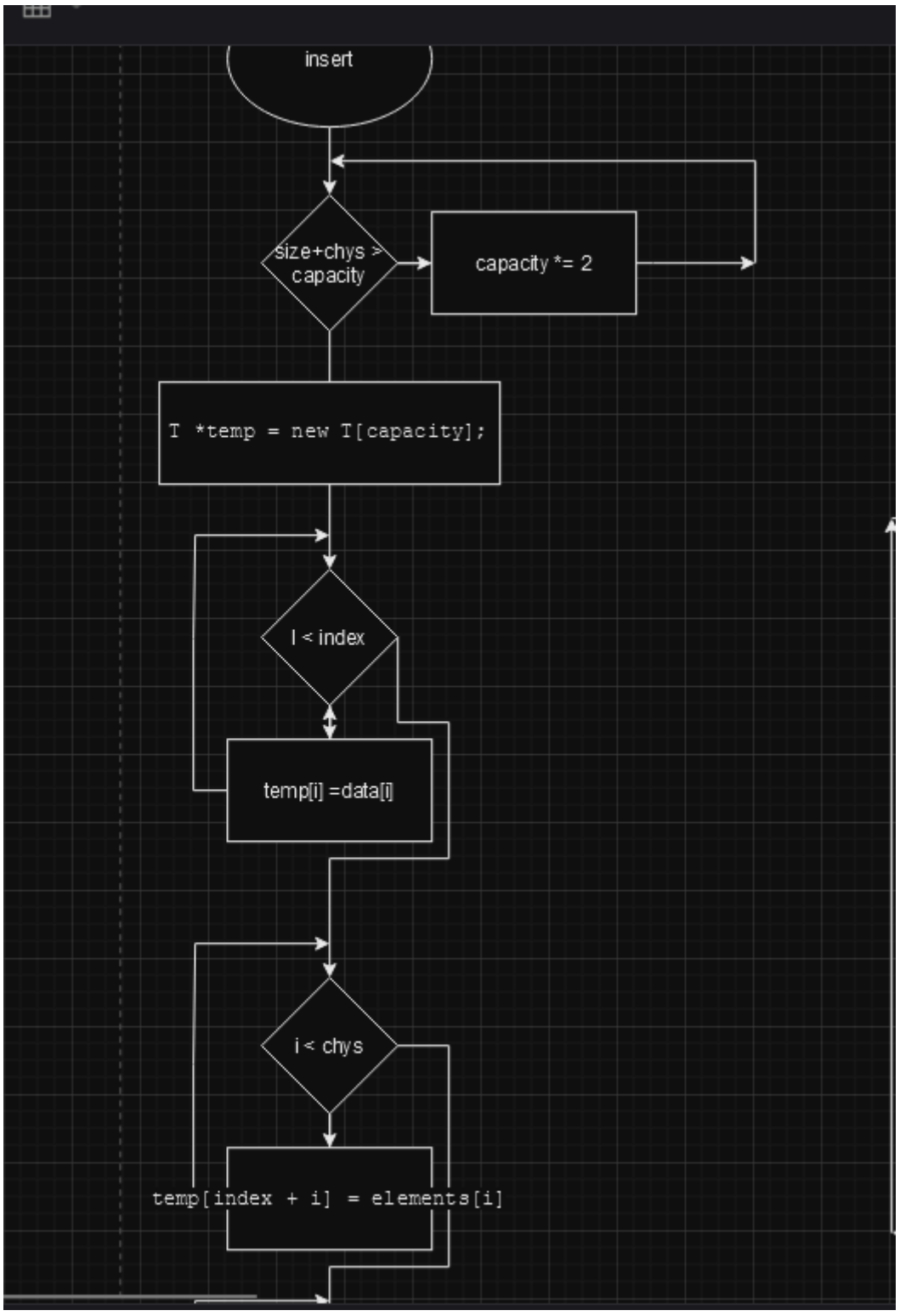
```cpp
#include <iostream>
#include <algorithm>

using namespace std;

int main(){
    int a,b;

    cin>>a >>b;

    int maxim, minimum;
    maxim = max(a,b);
    minimum = min(a,b);

    if(maxim - minimum > 1){
        cout<<minimum + 1<<endl;
    }
    else{
        cout<<-1<<endl;
    }
    return 0;
}
```

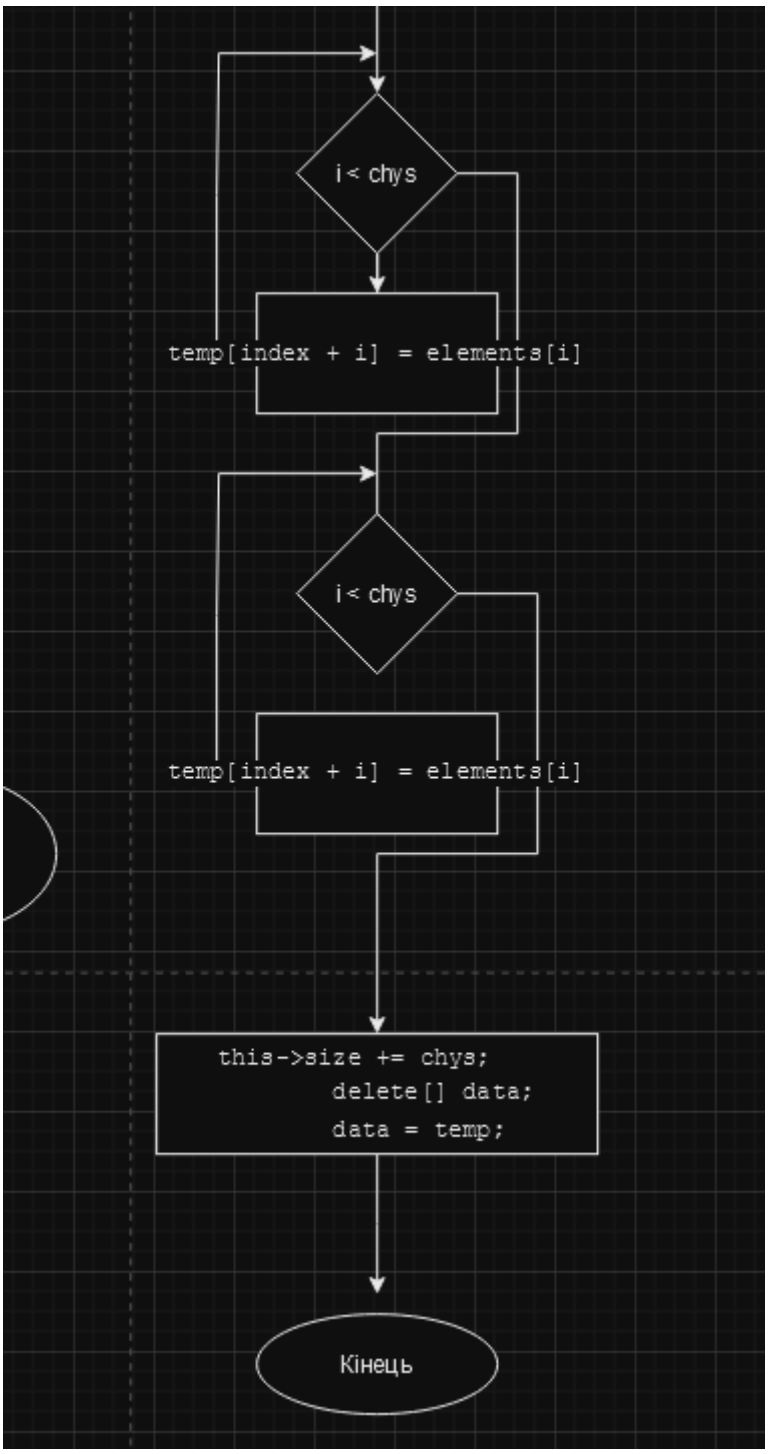
```
4 5
-1
```

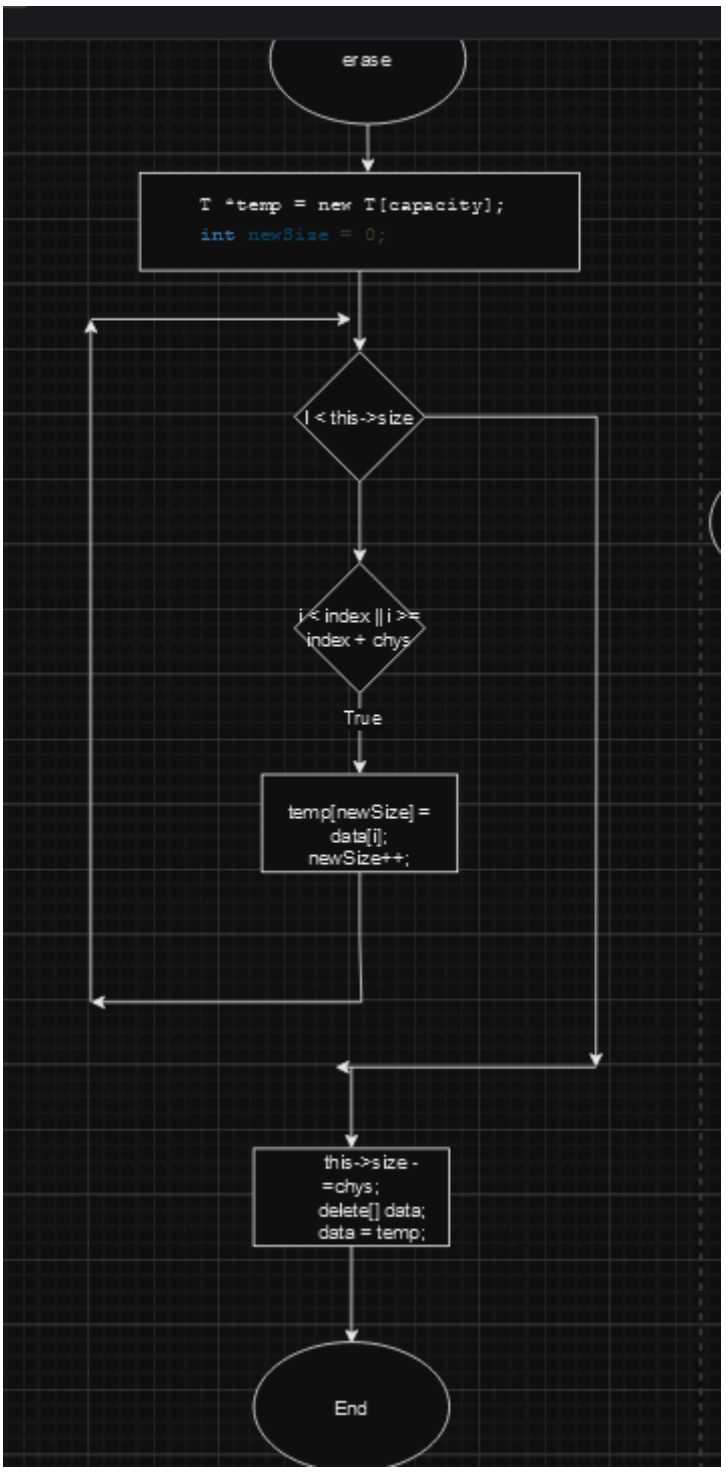
Блок-схема:

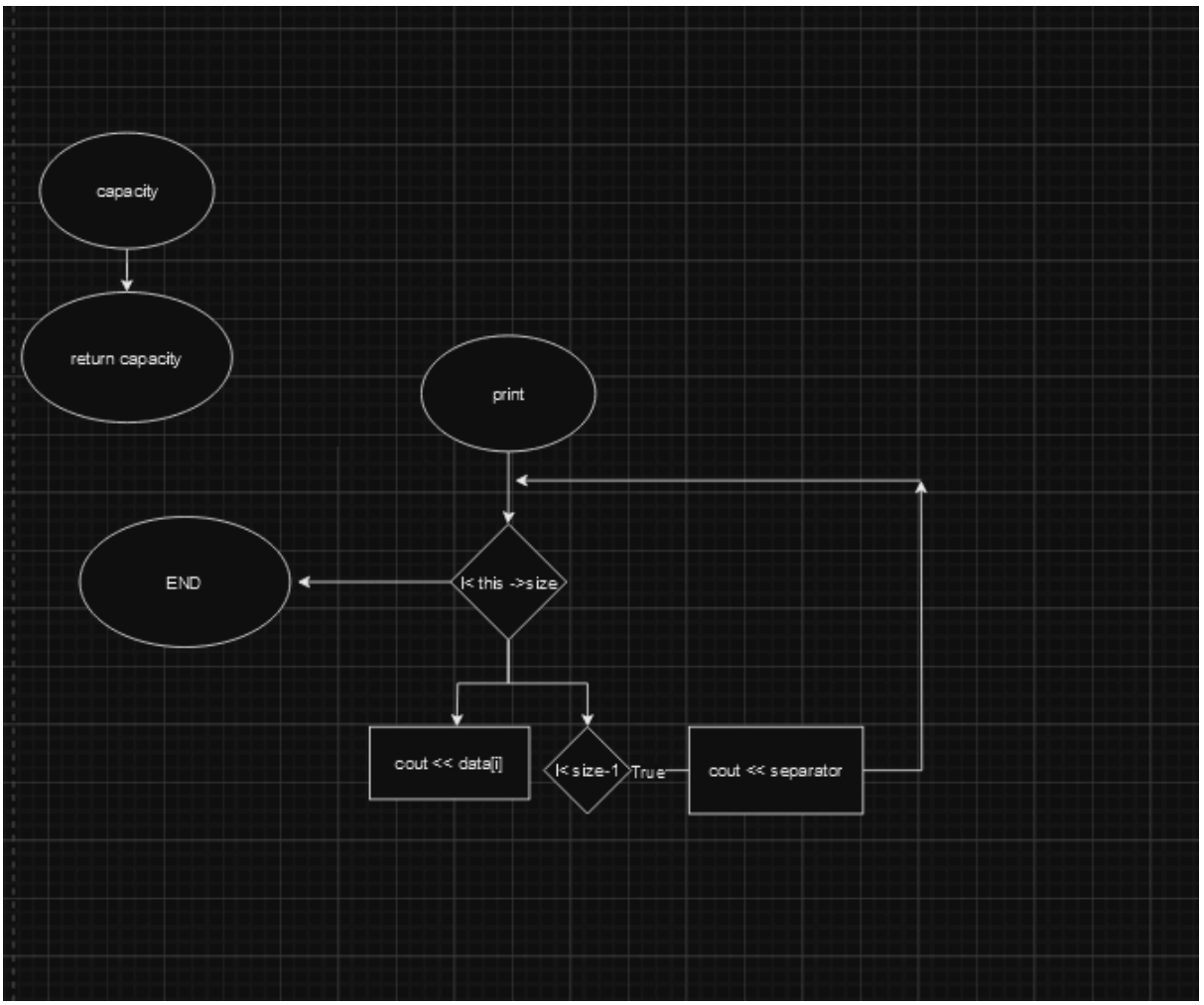

Start
Введення queryCount
queryCount--
command=="insert"
cin >> index >> chys
l < chys
cin >> elements[j]
insert
delete[] elements
command=="erase"
cin >> index >> chys
erase
command=="size"
size
Вивід size


command=="erase"
cin >> index >> chys
erase
command=="size"
size
Вивід size
command=="capacity"
capacity
Вивід capacity
command=="get"
cin >> index >> chys
get
command=="set"
cin >> index >> chys
set
command=="print"
print
END

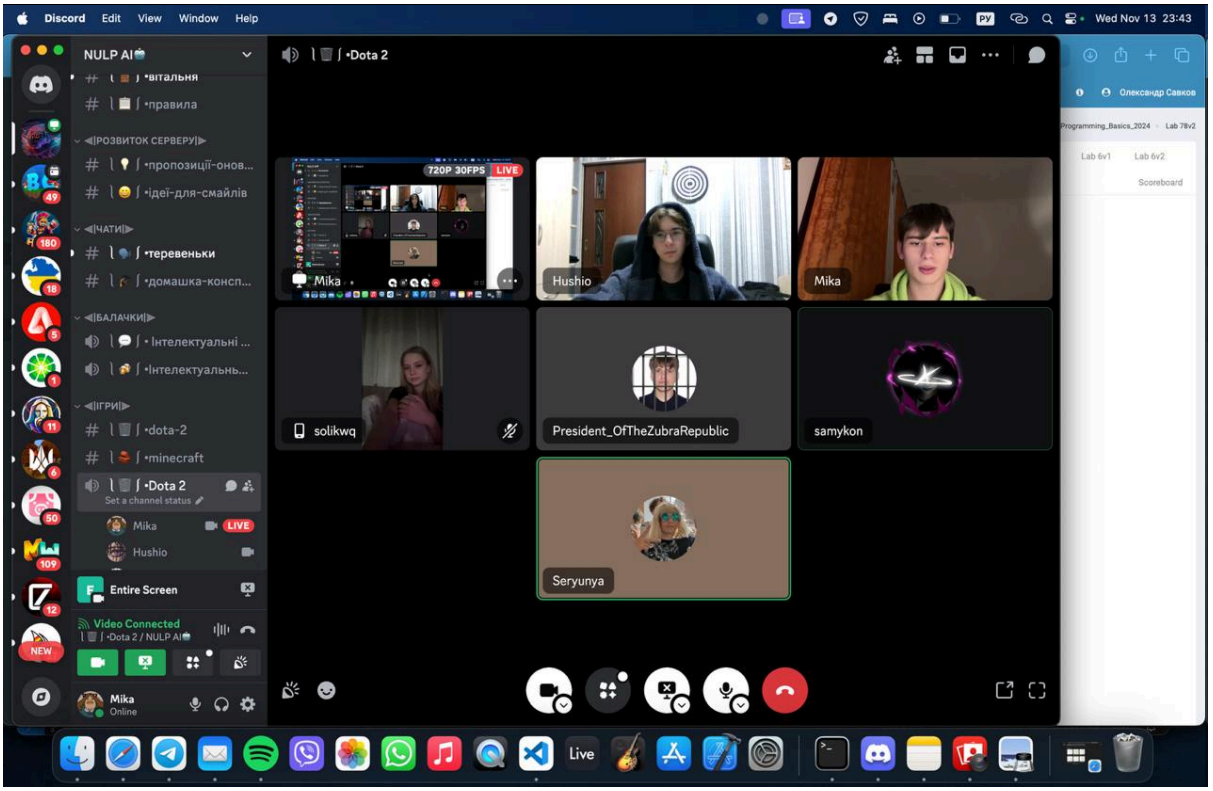

insert
size+chys > capacity
capacity *= 2
T *temp = new T[capacity];
I < index
temp[i] =data[i]
i < chys
temp[index + i] = elements[i]


i < chys
temp[index + i] = elements[i]
i < chys
temp[index + i] = elements[i]
this->size += chys;
delete[] data;
data = temp;
Кінець


erase
T *temp = new T[capacity];
int newSize = 0;
i < this->size
j < index || i >= index + chys
True
temp[newSize] = data[i];
newSize++;
this->size -= chys;
delete[] data;
data = temp;
End



Зустрічі з командою



Висновок:

Завдяки цій роботі я зрозумів принципи динамічних структур даних, їх ключові операції та відмінності між статичним і динамічним виділенням пам’яті. Практичні вправи зі стеком, чергою, зв’язним списком і деревом допомогли мені опанувати алгоритми обробки даних у пам'яті та ефективне управління ресурсами. Цей досвід дозволив мені створювати адаптивні рішення для роботи з великими та змінними обсягами даних, підвищуючи продуктивність програм.