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 Лабораторної Роботи № 5
Алготестер Лабораторної Роботи № 7-8
Практичних Робіт до блоку № 6

Виконав:
Студент(ка) групи ШІ-13
Яцишин Роман Олегович

Львів – 2024

Тема:

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

Мета:

навчитись працювати з динамічними структурами, реалізувати зв’язний список та навчитись працювати з бінарними деревами.

Теоретичні відомості:

1. Основи Динамічних Структур Даних:
    - Вступ до динамічних структур даних: визначення та важливість
    - Виділення пам'яті для структур даних (stack і heap)
2. Стек:
    - Визначення та властивості стеку
    - Операції push, pop, top: реалізація та використання
    - Приклади використання стеку: обернений польський запис, перевірка балансу дужок
    - Переповнення стеку
3. Черга:
    - Визначення та властивості черги
    - Операції enqueue, dequeue, front: реалізація та застосування
4. Зв'язні Списки:
    - Визначення однозв'язного та двозв'язного списку
    - Принципи створення нових вузлів, вставка між існуючими, видалення, створення кільця(circular linked list)
    - Основні операції: обхід списку, пошук, доступ до елементів, об'єднання списків
    - Приклади використання списків: управління пам'яттю, FIFO та LIFO структури
5. Дерева:
    - Бінарні дерева: вставка, пошук, видалення
    - Обхід дерева: в глибину (preorder, inorder, postorder), в ширину
    - Застосування дерев: дерева рішень, хеш-таблиці
    - Складніші приклади дерев: AVL, Червоно-чорне дерево
6. Алгоритми Обробки Динамічних Структур:
    - Основи алгоритмічних патернів: ітеративні, рекурсивні
    - Алгоритми пошуку, сортування даних, додавання та видалення елементів

Виконання роботи:

1. Опрацювання завдання та вимог до програм та середовища:

Програмний код №1

Написати програму, у якій створюються динамічні структури й виконати їхню обробку у відповідності зі своїм варіантом.

Для кожного варіанту розробити такі функції:

1. Створення списку.
2. Додавання елемента в список (у відповідності зі своїм варіантом).
3. Знищення елемента зі списку (у відповідності зі своїм варіантом).
4. Друк списку.
5. Запис списку у файл.
6. Знищення списку.
7. Відновлення списку з файлу.

Порядок виконання роботи

1. Написати функцію для створення списку. Функція може створювати порожній список, а потім додавати в нього елементи.
2. Написати функцію для друку списку. Функція повинна передбачати вивід повідомлення, якщо список порожній.
3. Написати функції для знищення й додавання елементів списку у відповідності зі своїм варіантом.
4. Виконати зміни в списку й друк списку після кожної зміни.
5. Написати функцію для запису списку у файл.
6. Написати функцію для знищення списку.
7. Записати список у файл, знищити його й виконати друк (при друці повинне бути видане повідомлення "Список порожній").
8. Написати функцію для відновлення списку з файлу.
9. Відновити список і роздрукувати його.
10. Знищити список.

Варіант:

4. Записи в лінійному списку містять ключове поле типу int. Сформувати однонаправлений список. Знищити з нього елемент із заданим номером, додати К елементів, починаючи із заданого номера;

Програмний код №2

У світі Атод сестри Ліна і Рілай люблять грати у гру. У них є дошка із 8-ми рядків і 8-ми стовпців. На перетині i-го рядка і j-го стовпця лежить магічна куля, яка може світитись магічним світлом (тобто у них є 64 кулі). На початку гри деякі кулі світяться, а деякі ні... Далі вони обирають N куль і для кожної читають магічне заклинання, після чого всі кулі, які лежать на перетині стовпця і рядка обраної кулі міняють свій стан (ті що світяться - гаснуть, ті, що не світяться - загораються).

Також вони вирішили трохи Вам допомогти і придумали спосіб як записати стан дошки одним числом a із 8-ми байт, а саме (див. Примітки):

- Молодший байт задає перший рядок матриці;
- Молодший біт задає перший стовпець рядку;
- Значення біту каже світиться куля чи ні (0 - ні, 1 - так);

Тепер їх цікавить яким буде стан дошки після виконання N заклинань і вони дуже просять Вас їм допомогти.

Вхідні дані

У першому рядку одне число a - поточний стан дошки.

У другому рядку N - кількість заклинань.

У наступних N рядках по 2 числа R_i, C_i - рядок і стовпець кулі над якою виконується заклинання.

Вихідні дані

Одне число b - стан дошки після виконання N заклинань.

Програмний код №3

Ваше завдання - власноруч реалізувати структуру даних "Двійкове дерево пошуку".
Ви отримаєте Q запитів, кожен запит буде починатися зі слова-ідентифікатора, після якого йдуть його параметри.

Вам будуть поступати запити такого типу:

- **Вставка**:
  Ідентифікатор - $insert$
  Ви отримуєте ціле число $value$ - число, яке треба вставити в дерево.
- **Пошук**:
  Ідентифікатор - $contains$
  Ви отримуєте ціле число $value$ - число, наявність якого у дереві необхідно перевірити.
  Якщо $value$ наявне в дереві - ви виводите Yes, у іншому випадку No.
- **Визначення розміру**:
  Ідентифікатор - $size$
  Ви не отримуєте аргументів.
  Ви виводите кількість елементів у дереві.
- **Вивід дерева на екран**
  Ідентифікатор - $print$
  Ви не отримуєте аргументів.
  Ви виводите усі елементи дерева через пробіл.
  Реалізувати використовуючи перегрузку оператора $<<$

Вхідні дані

Ціле число Q - кількість запитів.
У наступних рядках Q запитів у зазначеному в умові форматі.

Вихідні дані

Відповіді на запити у зазначеному в умові форматі.

Програмний код №3

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

Вам будуть поступати запити такого типу:

- **Вставка**:
  Ідентифікатор - $insert$
  Ви отримуєте ціле число $index$ елемента, на місце якого робити вставку.
  Після цього в наступному рядку рядку написане число N - розмір масиву, який треба вставити.
  У третьому рядку N цілих чисел - масив, який треба вставити на позицію $index$.
- **Видалення**:
  Ідентифікатор - $erase$
  Ви отримуєте 2 цілих числа - $index$, індекс елемента, з якого почати видалення та n - кількість елементів, яку треба видалити.
- **Визначення розміру**:
  Ідентифікатор - $size$
  Ви не отримуєте аргументів.
  Ви виводите кількість елементів у динамічному масиві.
- **Визначення кількості зарезервованої пам'яті**:
  Ідентифікатор - $capacity$
  Ви не отримуєте аргументів.
  Ви виводите кількість зарезервованої пам'яті у динамічному масиві.
  Ваша реалізація динамічного масиву має мати фактор росту (Growth factor) рівний 2.
- **Отримання значення** i-го елементу
  Ідентифікатор - get
  Ви отримуєте ціле число - $index$, індекс елемента.
  Ви виводите значення елемента за індексом. Реалізувати використовуючи перегрузку оператора []
- **Модифікація значення** i-го елементу
  Ідентифікатор - set
  Ви отримуєте 2 цілих числа - індекс елемента, який треба змінити, та його нове значення. Реалізувати використовуючи перегрузку оператора []
- **Вивід динамічного масиву на екран**
  Ідентифікатор - $print$
  Ви не отримуєте аргументів.
  Ви виводите усі елементи динамічного масиву через пробіл.
  Реалізувати використовуючи перегрузку оператора $<<$

Вхідні дані

Ціле число Q - кількість запитів.
У наступних рядках Q запитів у зазначеному в умові форматі.

Вихідні дані

Відповіді на запити у зазначеному в умові форматі.

Програмний код №4

Задача №1 - Реверс списку (Reverse list)

Реалізувати метод реверсу списку: Node* reverse(Node *head);
Умови задачі:

- використовувати цілочисельні значення в списку;
- реалізувати метод реверсу;
- реалізувати допоміжний метод виведення вхідного і обернутого списків;

Задача №2 - Порівняння списків

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

Умови задачі:

- використовувати цілочисельні значення в списку;
- реалізувати функцію, яка ітеративно проходиться по обох списках і порівнює дані в кожному вузлі;
- якщо виявлено невідповідність даних або якщо довжина списків різна (один список закінчується раніше іншого), функція повертає ***false***.

Задача №3 – Додавання великих чисел

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

Умови задачі:

- використовувати цифри від 0 до 9 для значень у списку;
- реалізувати функцію, яка обчислює суму двох чисел, які збережено в списку; молодший розряд числа записано в голові списка (напр. 379 $\Longrightarrow$ 9→7→3);
- функція повертає новий список, передані в функцію списки не модифікуються.

Програмний код №5

В пустелі існує незвичайна печера, яка є двохвимірною. Її висота це N, ширина - M.

Всередині печери є пустота, пісок та каміння. Пустота позначається буквою O , пісок S і каміння X;

Одного дня стався землетрус і весь пісок посипався вниз. Він падає на найнижчу клітинку з пустотою, але він не може пролетіти через каміння.

Ваше завдання сказати як буде виглядати печера після землетрусу.

Вхідні дані

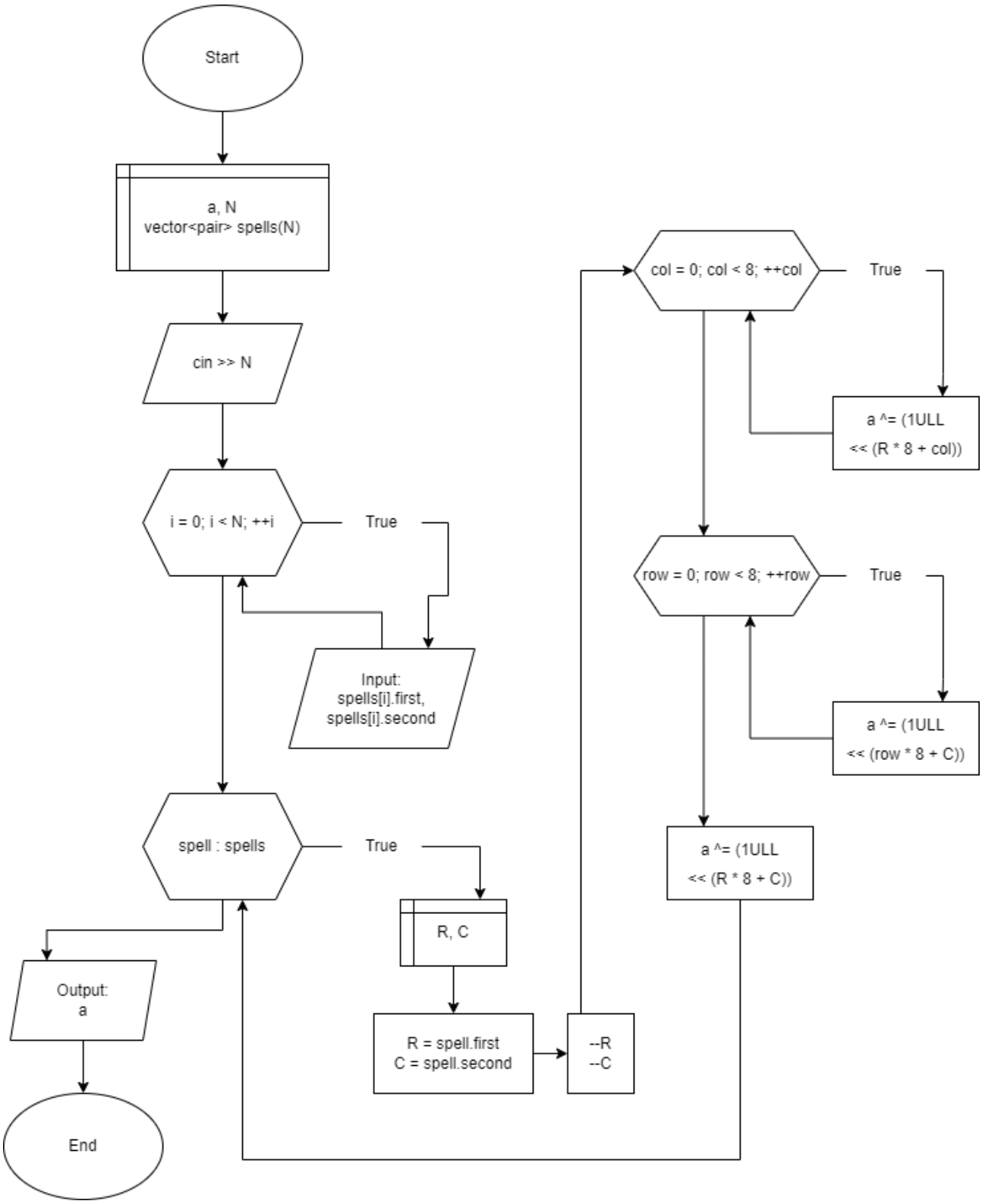
У першому рядку 2 цілих числа N та M - висота та ширина печери

У N наступних рядках стрічка row_i яка складається з N цифер - і-й рядок матриці, яка відображає стан печери до землетрусу.

Вихідні дані

N рядків, які складаються з стрічки розміром M - стан печери після землетрусу.

2. Дизайн:



3. Код програми:

1) VNS Lab 10

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

// Структура вузла списку
struct Node {
    int data;
    Node* next;
};

// Функція створення порожнього списку
Node* createList() {
    return nullptr;
}

void addToTail(Node*& head, int value) {
    Node* newNode = new Node;
    newNode->data = value;
    newNode->next = nullptr;

    if (!head) {
        head = newNode;
        return;
    }

    Node* current = head;
    while (current->next) {
        current = current->next;
    }
    current->next = newNode;
}

void fillList(Node*& head) {
    for (int i = 1; i <= 10; ++i) {
        addToTail(head, i);
    }
}
```

```cpp
// Функція друку списку
void printList(Node* head) {
    if (!head) {
        cout << "Список порожній." << endl;
        return;
    }
    Node* current = head;
    while (current) {
        cout << current->data << " ";
        current = current->next;
    }
    cout << endl;
}

// Функція додавання елемента в список з заданої позиції
void addElements(Node*& head, int position, int count) {
    Node* current = head;
    int currentIndex = 1;

    while (current && currentIndex < position) {
        current = current->next;
        currentIndex++;
    }

    for (int i = 0; i < count; i++) {
        Node* newNode = new Node;
        cout << "Введіть значення для нового елемента: ";
        cin >> newNode->data;
        newNode->next = current ? current->next : nullptr;
        if (current) {
            current->next = newNode;
        } else {
            head = newNode;
        }
        current = newNode;

```

```
        printList(head);
    }
}

// Функція видалення елемента зі списку за номером
void deleteElement(Node*& head, int position) {
    if (!head) {
        cout << "Список порожній." << endl;
        return;
    }

    if (position == 1) {
        Node* temp = head;
        head = head->next;
        delete temp;
        return;
    }

    Node* current = head;
    int currentIndex = 1;

    while (current->next && currentIndex < position - 1) {
        current = current->next;
        currentIndex++;
    }

    if (!current->next) {
        cout << "Неможливо видалити: позиція виходить за межі списку." << endl;
        return;
    }

    Node* temp = current->next;
    current->next = temp->next;
    delete temp;
```

```
    printList(head);
}

// Функція запису списку у файл
void saveToFile(Node* head, const string& filename) {
    ofstream file(filename);
    if (!file) {
        cout << "Помилка відкриття файлу." << endl;
        return;
    }

    Node* current = head;
    while (current) {
        file << current->data << " ";
        current = current->next;
    }
    file.close();
}

// Функція знищення списку
void deleteList(Node*& head) {
    while (head) {
        Node* temp = head;
        head = head->next;
        delete temp;
    }
    cout << "Список знищено." << endl;
}

// Функція відновлення списку з файлу
void restoreFromFile(Node*& head, const string& filename) {
    ifstream file(filename);
    if (!file) {
        cout << "Помилка відкриття файлу." << endl;
        return;
    }
```

```
147     deleteList(head);
148
149     int value;
150     Node* tail = nullptr;
151     while (file >> value) {
152         Node* newNode = new Node{value, nullptr};
153         if (!head) {
154             head = newNode;
155         } else {
156             tail->next = newNode;
157         }
158         tail = newNode;
159     }
160     file.close();
161
162     printList(head);
163 }
164
165
166
167 int main() {
168     Node* list = createList();
169     fillList(list);
170
171     int choice, position, count;
172
173     printList(list);
174
175     cout << "Введіть номер елемента для видалення: ";
176     cin >> position;
177     deleteElement(list, position);
178
```

```
179     cout << "Введіть позицію для додавання: ";
180     cin >> position;
181     cout << "Введіть кількість елементів для додавання: ";
182     cin >> count;
183     addElements(list, position, count);
184
185     saveToFile(list, "list.txt");
186
187     deleteList(list);
188
189     printList(list);
190
191     restoreFromFile(list, "list.txt");
192
193     deleteList(list);
194 }
195
```

Node *list

2) Algotester Lab 5

```cpp
#include <iostream>
#include <vector>
#include <cstdint>

using namespace std;

int main() {
    uint64_t a;
    int N;
    cin >> a >> N;

    vector<pair<int, int>> spells(N);
    for (int i = 0; i < N; ++i) {
        cin >> spells[i].first >> spells[i].second;
    }

    for (const auto& spell : spells) {
        int R = spell.first;
        int C = spell.second;
        --R; --C;

        for (int col = 0; col < 8; ++col) {
            a ^= (1ULL << (R * 8 + col));
        }

        for (int row = 0; row < 8; ++row) {
            a ^= (1ULL << (row * 8 + C));
        }

        a ^= (1ULL << (R * 8 + C));
    }

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

```

3) Algotester Lab 7-8 Variant 1

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

struct Node {
    int value;
    Node* left;
    Node* right;
    Node(int val) : value(val), left(nullptr), right(nullptr) {}
};

class BinarySearchTree {

private:
    Node* root;
    Node* insert(Node* node, int value) {
    if (node == nullptr) {
        return new Node(value);
    }

    if (value < node->value) {
        node->left = insert(node->left, value);
    } else if (value > node->value) {
        node->right = insert(node->right, value);
    } else {}

    return node;
}


    bool contains(Node* node, int value) {
        if (node == nullptr) {
            return false;
        }
        if (node->value == value) {
            return true;
        }
        if (value < node->value) {
            return contains(node->left, value);
```

```
        }
        return contains(node->right, value);
    }

    int size(Node* node) {
        if (node == nullptr) {
            return 0;
        }
        return 1 + size(node->left) + size(node->right);
    }

    void printInOrder(Node* node, ostream& os) const {
        if (node != nullptr) {
            printInOrder(node->left, os);
            os << node->value << " ";
            printInOrder(node->right, os);
        }
    }

public:
    BinarySearchTree() : root(nullptr) {}

    void insert(int value) {
        root = insert(root, value);
    }

    bool contains(int value) {
        return contains(root, value);
    }

    int size() {
        return size(root);
    }

```

```
    friend ostream& operator<<(ostream& os, const BinarySearchTree& tree) {
        tree.printInOrder(tree.root, os);
        return os;
    }
};

int main() {
    int Q;
    cin >> Q;

    BinarySearchTree tree;
    vector<string> results;

    for (int i = 0; i < Q; ++i) {
        string query;
        cin >> query;

        if (query == "insert") {
            int value;
            cin >> value;
            tree.insert(value);
        } else if (query == "contains") {
            int value;
            cin >> value;
            if (tree.contains(value)) {
                results.push_back("Yes");
            } else {
                results.push_back("No");
            }
        } else if (query == "size") {
            results.push_back(to_string(tree.size()));
        } else if (query == "print") {
            ostringstream os;
            os << tree;
            results.push_back(os.str());
        }
    }
```

```
    for (const string& result : results) {
        cout << result << endl;
    }

    return 0;
}
```

4) Algotester Lab 7-8 Variant 2

```cpp
#include <iostream>
#include <stdexcept>
#include <vector>
#include <string>
#include <sstream>

using namespace std;

template <type
class DynamicArray {
private:
    T* data;
    size_t capacity;
    size_t size;

    void growCapacity(size_t minCapacity) {
        while (capacity <= minCapacity) {
            capacity *= 2;
        }
        T* newData = new T[capacity];
        for (size_t i = 0; i < size; ++i) {
            newData[i] = data[i];
        }
        delete[] data;
        data = newData;
    }

public:
    DynamicArray() : data(new T[1]), capacity(1), size(0) {}

    ~DynamicArray() {
        delete[] data;
    }

    void insert(size_t index, size_t n, const T* elements) {
        if (index > size) {
            throw out_of_range("Index out of bounds");
        }
        if (size + n >= capacity) {
            growCapacity(size + n);
```

```
41          }
42          for (size_t i = size; i > index; --i) {
43              data[i + n - 1] = data[i - 1];
44          }
45          for (size_t i = 0; i < n; ++i) {
46              data[index + i] = elements[i];
47          }
48          size += n;
49      }
50
51      void erase(size_t index, size_t n) {
52          if (index >= size || index + n > size) {
53              throw out_of_range("Index out of bounds");
54          }
55          for (size_t i = index; i < size - n; ++i) {
56              data[i] = data[i + n];
57          }
58          size -= n;
59      }
60
61      size_t getSize() const {
62          return size;
63      }
64
65      size_t getCapacity() const {
66          return capacity;
67      }
68
69      T& operator[](size_t index) {
70          if (index >= size) {
71              throw out_of_range("Index out of bounds");
72          }
73          return data[index];
74      }
75
```

template<class T> size_t DynamicArray<T>::size

```
    const T& operator[](size_t index) const {
        if (index >= size) {
            throw out_of_range("Index out of bounds");
        }
        return data[index];
    }

    friend ostream& operator<<(ostream& os, const DynamicArray& arr) {
        for (size_t i = 0; i < arr.size; ++i) {
            os << arr.data[i] << " ";
        }
        return os;
    }
};

int main() {
    size_t Q;
    cin >> Q;

    DynamicArray<int> arr;
    vector<string> output;

    for (size_t q = 0; q < Q; ++q) {
        string command;
        cin >> command;

        if (command == "insert") {
            size_t index, N;
            cin >> index >> N;
            int* elements = new int[N];
            for (size_t i = 0; i < N; ++i) {
                cin >> elements[i];
            }
            arr.insert(index, N, elements);
            delete[] elements;
        }

```

```
        else if (command == "erase") {
            size_t index, n;
            cin >> index >> n;
            arr.erase(index, n);
        }

        else if (command == "size") {
            output.push_back(to_string(arr.getSize()));
        }

        else if (command == "capacity") {
            output.push_back(to_string(arr.getCapacity()));

        }

        else if (command == "get") {
            size_t index;
            cin >> index;
            output.push_back(to_string(arr[index]));
        }

        else if (command == "set") {
            size_t index;
            int value;
            cin >> index >> value;
            arr[index] = value;
        }

        else if (command == "print") {
            ostringstream oss;
            oss << arr;
            output.push_back(oss.str());
        }

        else {
            cerr << "Unknown command" << endl;
        }
    }

    for (const auto& line : output) {
        cout << line << endl;
    }

    return 0;
}
```

5) Class Practice task

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

// Structures
struct Node {
    int data;
    Node* next;
    Node(int val) : data(val), next(nullptr) {}
};

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


// Functions
Node* reverse(Node* head) {
    Node* prev = nullptr;
    Node* current = head;
    Node* next = nullptr;

    while (current) {
        next = current->next;
        current->next = prev;
        prev = current;
        current = next;
    }
    return prev;
}

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

```cpp
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;
        sum = sum % 10;

        Node* newNode = new Node(sum);
        if (!result) {
            result = tail = newNode;
        } else {
            tail->next = newNode;
            tail = tail->next;
        }
    }
```

```
    return result;
}

TreeNode* create_mirror_flip(TreeNode* root) {
    if (!root) return nullptr;

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

    return newRoot;
}

int tree_sum(TreeNode* root) {
    if (!root) return 0;

    int leftSum = tree_sum(root->left);
    int rightSum = tree_sum(root->right);

    if (root->left || root->right) {
        root->data = leftSum + rightSum;
    }
    return root->data + (root->left ? root->left->data : 0) + (root->right ? root->right->data : 0);
}

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

int main() {
```

```cpp
int main() {

    // Задача 1
    Node* head = new Node(1);
    head->next = new Node(2);
    head->next->next = new Node(3);
    head->next->next->next = new Node(4);

    cout << "Original list: ";
    printList(head);

    head = reverse(head);

    cout << "Reversed list: ";
    printList(head);

    // Задача 2
    Node* h1 = new Node(1);
    h1->next = new Node(2);
    h1->next->next = new Node(3);

    Node* h2 = new Node(1);
    h2->next = new Node(2);
    h2->next->next = new Node(3);

    if (compare(h1, h2)) {
        cout << "Lists are equal." << endl;
    } else {
        cout << "Lists are not equal." << endl;
    }

    // Задача 3
    Node* n1 = new Node(9);
    n1->next = new Node(7);
    n1->next->next = new Node(3);

```

```
    Node* n2 = new Node(6);
    n2->next = new Node(8);
    n2->next->next = new Node(5);

    Node* result = add(n1, n2);

    cout << "Sum: ";
    printList(result);

    // Задача 4
    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);

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

    TreeNode* mirror = create_mirror_flip(root);

    cout << "Mirror tree: ";
    printTree(mirror);
    cout << endl;

    // Задача 5
    cout << "Original tree: ";
    printTree(root);
    cout << endl;

    tree_sum(root);

    cout << "Tree after sum computation: ";
    printTree(root);
    cout << endl;

    return 0;
}
```

6) Self Practice Task

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

int main() {
    int N, M;
    cin >> N >> M;

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

    vector<string> result(N, string(M, 'O'));

    for (int col = 0; col < M; ++col) {
        int emptyRow = N - 1;
        for (int row = N - 1; row >= 0; --row) {
            if (cave[row][col] == 'X') {
                result[row][col] = 'X';
                emptyRow = row - 1;
            } else if (cave[row][col] == 'S') {
                result[emptyRow][col] = 'S';
                --emptyRow;
            }
        }
    }

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

    return 0;
}
```

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

1) VNS Lab 10

```
PS D:\Epic 6>  & 'c:\Users\1\.vscode\extensions\ms
soft-MIEngine-In-mmtjt2cv.cpl' '--stdout=Microsoft
-MIEngine-Pid-dlcqafgs.gfi' '--dbgExe=C:\msys64\uc
1 2 3 4 5 6 7 8 9 10
Введіть номер елемента для видалення: 2
1 3 4 5 6 7 8 9 10
Введіть позицію для додавання: 6
Введіть кількість елементів для додавання: 2
Введіть значення для нового елемента: 69
1 3 4 5 6 7 69 8 9 10
Введіть значення для нового елемента: 96
1 3 4 5 6 7 69 96 8 9 10
Список знищено.
Список порожній.
Список знищено.
1 3 4 5 6 7 69 96 8 9 10
Список знищено.
```

Фактично затрачений час: 2 год

3) Algotester Lab 7-8 Variant 1

```
PS D:\Epic 6>  & 'c:\Users\1\.vs
soft-MIEngine-In-pn4exozs.p0n' '
-MIEngine-Pid-wh50v5qz.vpp' '--d
11
size
insert 4
insert 5
print
insert 5
print
insert 1
print
contains 5
contains 0
size
0
4 5
4 5
1 4 5
Yes
No
3
PS D:\Epic 6>
```

Фактично затрачений час: 2 год

4) Algotester Lab 7-8 Variant 2

```
PS D:\Epic 6>  & 'c:\Users\1\.vscod
soft-MIEngine-In-im32riov.gss' '--s
-MIEngine-Pid-ldzthxjg.m50' '--dbgE
12
size
capacity
insert 0 2
100 100
size
capacity
insert 0 2
102 102
size
capacity
insert 0 2
103 103
size
capacity
print
0
1
2
4
4
8
6
8
103 103 102 102 100 100
PS D:\Epic 6>
```

Фактично затрачений час: 2 год

5) Class practice task

```
PS D:\Epic 6>  & 'c:\Users\1\.vscode\exte
soft-MIEngine-In-lhj1zek0.xod' '--stdout=
-MIEngine-Pid-1gotzmih.t2u' '--dbgExe=C:\
Original list: 1 2 3 4
Reversed list: 4 3 2 1
Lists are equal.
Sum: 5 6 9
Original tree: 1 2 4 5 3
Mirror tree: 1 3 2 5 4
Original tree: 1 2 4 5 3
Tree after sum computation: 21 9 4 5 3
PS D:\Epic 6>
```

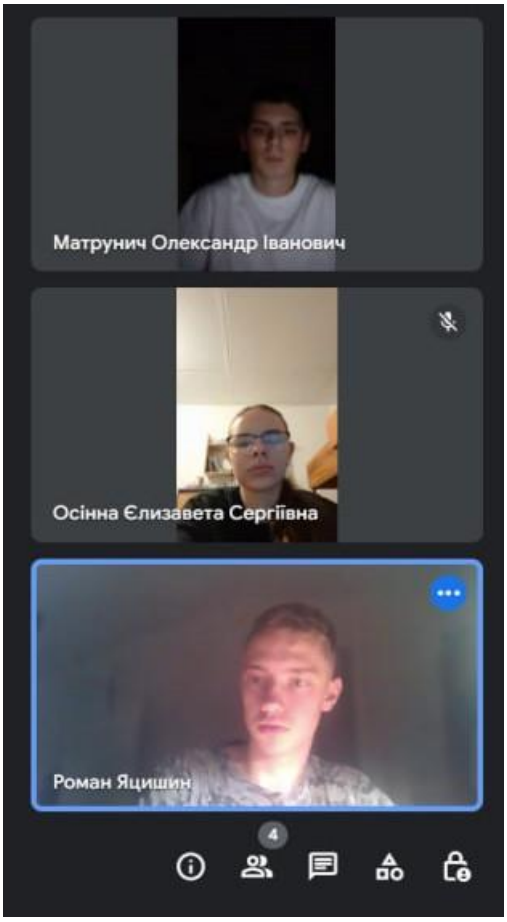
Фактично затрачений час: 2 год

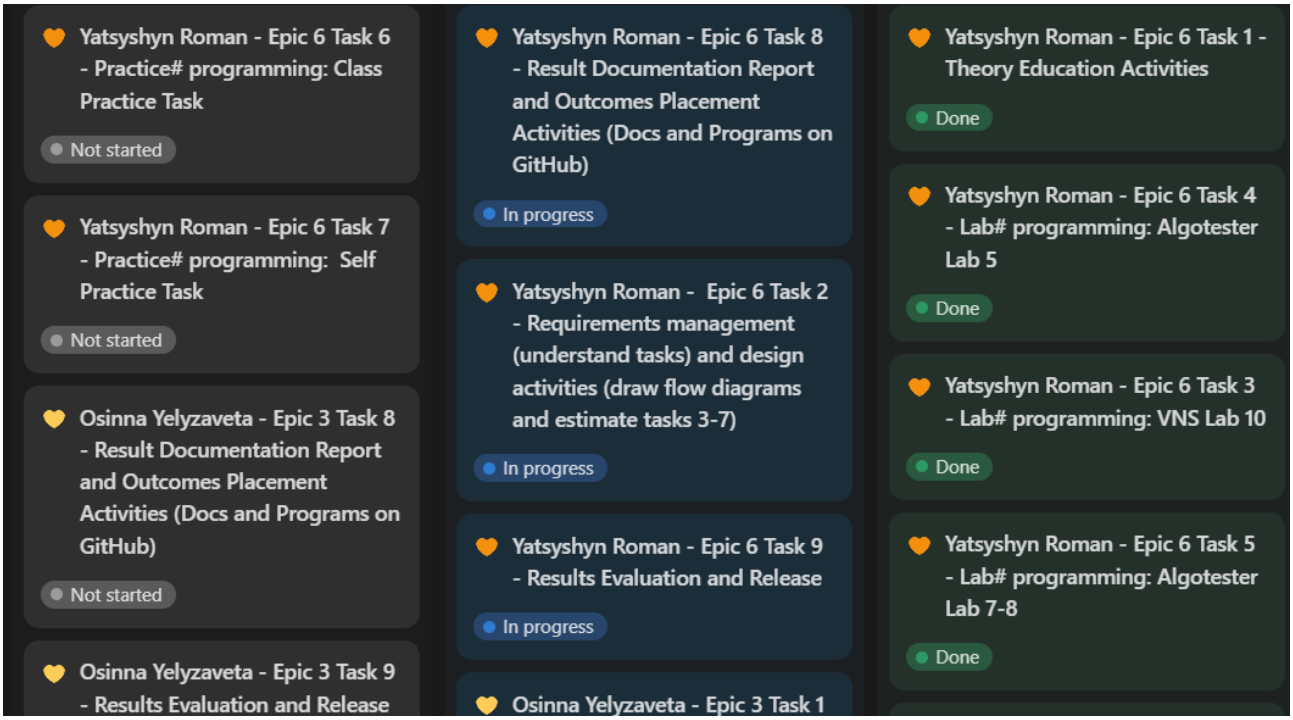
6) Self Practice Task

```
PS D:\Epic 6>  & 'c:\Use
soft-MIEngine-In-emzuys4
-MIEngine-Pid-1hoxffgm.z
10 10
SSSSSSSSSS
OXOOOSOOOO
OOOOOOOOOO
XOXOXOXOXO
SSSSSOOOOO
OOOOSXSXXO
OOOOOOOOOO
XOOOOOOOOO
OOOOOOOOXO
SSSXOOOOOO
OSOOOOOOOO
OXOOOOOOOO
SOSOSOSOSO
XOXOXSXOXO
OOOOOSOSOO
OOOOOXOXXO
SOOOOOOOOO
XOOSOOOOOO
OSSSSOOOXO
SSSXSOSOOS
PS D:\Epic 6>
```

Фактично затрачений час: 1 год.

6. Кооперація з командою:





Висновок:

У ході виконання роботи було розглянуто основні динамічні структури даних: черги, стеки, зв’язні списки та бінарні дерева. Було реалізовано алгоритми для додавання, видалення та обробки елементів у зв’язному списку, а також

операції з бінарним деревом, такі як вставка елементів, дзеркальне відображення та обчислення сум піддерев.