Міністерство освіти і науки України
Національний університет "Львівська Політехніка"

Кафедра систем штучного інтелекту

Епік №6

з дисципліни
«Основи програмування»

Виконав:
студент групи ШІ-11
Гнатюк Ярослав

Львів – 2024 р.

Епік №6

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

Мета роботи: Дослідити основи динамічних структур даних (черга, стек, списки, дерево) та розробити алгоритми їхньої ефективної обробки для розв'язання задач із зберігання, пошуку, сортування та маніпулювання даними.

Теоретичні відомості:

- Однозв’язний список: https://www.youtube.com/watch?v=-25REjF_atI&t=53s&pp=ygUm0LHQu9C-0LPQsNC9INC30LLRj9C30L3RliDRgdC_0LjRgdC60Lg%3D
- Бінарні дерева: https://www.youtube.com/watch?v=qBFzNW0ALxQ&pp=ygUs0LHRltC90LDRgNC90ZYg0LTQtdGA0LXQstCwIGMrKyDQsdC70L7Qs9Cw0L0%3D
- Стек (LIFO): https://www.w3schools.com/cpp/cpp_stacks.asp
- Черга (FIFO): https://www.w3schools.com/cpp/cpp_queues.asp

Виконання роботи

Частина 1

Завдання №1

Назва: VNS Lab 10 Variant 5

Опис: 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.

Записи в лінійному списку містять ключове поле типу int. Сформувати однонаправлений список. Знищити з нього K елементів, починаючи із заданого номера, додати K елементів, починаючи із заданого номера.

Завдання №2

Назва: Algotester Lab 5 Variant 1

Опис: 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 змінюють 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.

Вимоги:

$$0 \leq N \leq 10^3$$

$$1 \leq R_i, C_i \leq 8$$

$$0 \leq a, b < 2^{64}$$

Завдання №3

Назва: Algotester Lab 78 Variant 2

Опис: 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 рівний 2.
- Отримання значення і-го елементу: Ідентифікатор - get
Ви отримуєте ціле число - index, індекс елемента. Ви виводите значення елемента за індексом.
- Модифікація значення і-го елементу: Ідентифікатор - set. Ви отримуєте 2 цілих числа - індекс елемента, який треба змінити, та його нове значення.
- Вивід динамічного масиву на екран: Ідентифікатор - print. Ви не отримуєте аргументів. Ви виводите усі елементи динамічного масиву через пробіл.

Вимоги:

$$0 \leq Q \leq 10^5$$
$$0 \leq l_i \leq 10^5$$
$$\|l\| \leq 10^5$$

Завдання №4

Назва: Algotester Lab 78 Variant 3

Опис: 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 <<

Вимоги:

$$0 \leq Q \leq 10^3$$
$$0 \leq t_i \leq 10^3$$

Завдання №5

Назва: Practice Work Task 1

Опис:

1. ***Реалізувати метод реверсу списку:***

   Умови задачі:

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

2. ***Порівняння списків***

   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***.

3. Додавання великих чисел

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. $379 \implies 9 \to 7 \to 3$);
- функція повертає новий список, передані в функцію списки не модифікуються.

Завдання №6

Назва: Practice Work Task 1

Опис:

1. Віддзеркалення дерева

Умови задачі:

- використовувати цілі числа для значень у вузлах дерева
- реалізувати функцію, що проходить по всіх вузлах дерева і міняє місцями праву і ліву вітки дерева
- функція повертає нове дерево, передане в функцію дерево не модифікується

2. Записати кожному батьківському вузлу суму підвузлів

Умови задачі:

- використовувати цілочисельні значення у вузлах дерева;
- реалізувати функцію, яка ітеративно проходить по бінарному дереві і записує у батьківський вузол суму значень підвузлів
- вузол-листок не змінює значення

- значення змінюються від листків до кореня дерева

Завдання №7

Назва: Self Practice Work (Зрада)

Опис:

Коли сонце заходить за обрій і настає ніч, пластуни розпалюють багаття, сідають навколо нього та починають займатися своїми улюбленими справами: співають пісні, розповідають один одному цікаві та страшні історії, діляться досвідом, вивчають сузір'я на небі. Коли всі пісні вже заспівано та всі історії розказано, пластуни, втомившись, розходяться по своїх наметах, щоб відпочити та підготувати свої юні організми до наступного, насиченого на події дня.

Та цього разу не все так просто, як пише книжка. Один намет кудись пропав. Помітивши пропажу, всі пластуни в паніці кинулися врізнобіч. «Де моя гітара??!!», — кричав один. «Де ми будемо спати??!!» — кричали інші. Пропажа намету явно підірвала бойовий дух пластунського загону. До таких ударів долі вони були не готові.

Але знайшлися в загоні двоє відчайдухів, які не змогли змиритися з жорстокою долею і вирішили взяти справу під свій контроль. Звали їх Зеник і Марічка. Найперше вони знайшли всіх своїх колег-пластунів, які в паніці порозбігалися по лісу, і зібрали їх навколо багаття. Наступне завдання, яке стояло перед Зеником і Марічкою — визначити, хто де буде спати цієї ночі. Підрахувавши кількість спальних місць з урахуванням пропажі, Зеник і Марічка дійшли висновку, що місць на всіх не вистачить, і декому доведеться провести цю ніч просто неба.

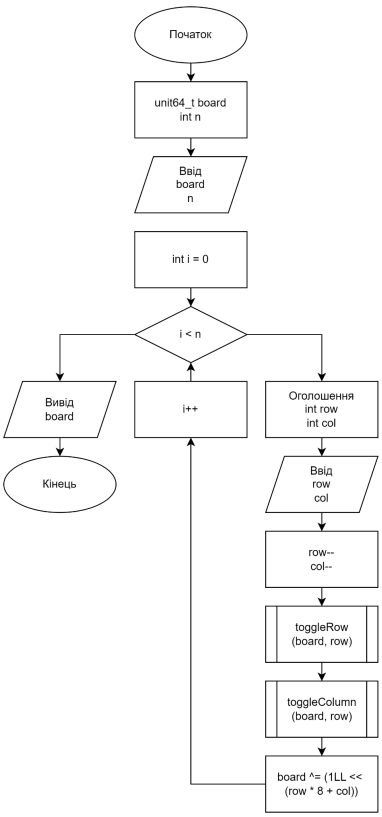
Для того, щоб підняти бойовий дух своїх побратимів і зробити процес вибору щасливців, що будуть спати в наметах, якомога більш чесним, Зеник із Марічкою придумали таку гру. Грають троє пластунів: двоє ведучих (Зеник і Марічка) та гравець. Починається гра з того, що гравець записує на аркуші паперу будь-яке натуральне число. Після цього кожен із ведучих називає по одному натуральному числу. Зеник називає число на проміжку від a до b, а Марічка називає число на проміжку від c до d. Потім гравець показує число, яке він записав на карточці. Якщо добуток чисел, названих ведучими, дорівнює числу, написаному на карточці, то гравець переміг. Інакше, він програв.

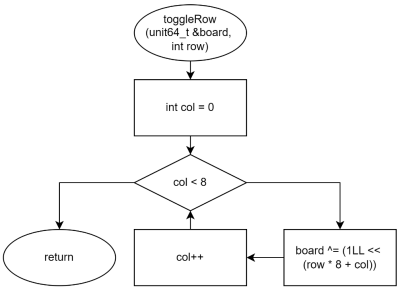
У цю гру Зеник із Марічкою збираються зіграти з кожним зі своїх побратимів. Пластунята, що виграють, підуть спати до наметів, які залишилися. Решта ж проведуть ніч просто неба.

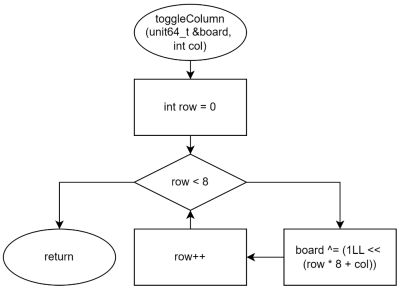
Та перед початком гри Зеник із Марічкою засумнівалися — а чи вистачить місця в наметах на всіх переможців? Для кожного свого товариша Зеник і Марічка точно знають, яке число вона чи він запише на карточці під час гри. Завдання для вас — знайти ймовірність виграшу для кожного пластуна.

Частина 2

Завдання №2







Частина 3

Завдання №1

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

using namespace std;

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

class LinkedList {
    private:
        Node *head;

        void deleteNode(Node*& current) {
            Node* temp = current;
            current = current->next;
            delete temp;
        }

        void destroyTree() {
            while (head) {
                deleteNode(head);
            }
        }

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

        ~LinkedList() {
            destroyTree();
        }

        void addElement(int position, int diapazone) {
            srand(time(nullptr));

            for (int i = 0; i < diapazone; i++) {
                int value = (rand() % 90) + 10;

                Node *newNode = new Node{value, nullptr};

                if (position == 0 || head == nullptr) {
                    newNode->next = head;
                    head = newNode;
                } else {
                    Node* current = head;
                    for (int j = 0; j < position - 1 && current->next != nullptr;
                        current = current->next;
                    }
                    newNode->next = current->next;
                    current->next = newNode;
                }
                position++;
            }
        }
```

```cpp
    }

    void removeElements(int position, int diapazone) {
        if (!head || diapazone <= 0 || position < 0) {
            cout << "Invalid range or empty list. No elements removed." << endl;
            return;
        }

        Node* current = head;
        Node* prev = nullptr;

        for (int i = 0; i < position && current != nullptr; ++i) {
            prev = current;
            current = current->next;
        }

        for (int i = 0; i < diapazone && current != nullptr; ++i) {
            Node* temp = current;
            current = current->next;
            if (prev) {
                prev->next = current;
            } else {
                head = current;
            }
            delete temp;
        }
    }

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

    void saveToFile(const string& filename) const {
        ofstream file(filename);
        if (!file.is_open()) {
            cerr << "Unable to open file for writing!" << endl;
            return;
        }

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

    void loadFromFile(const string& filename) {
        destroyTree();
```

```cpp
            ifstream file(filename);
            if (!file.is_open()) {
                cerr << "Unable to open file for reading!" << endl;
                return;
            }

            int value;
            while (file >> value) {
                addElement(value, getSize());
            }
            file.close();
        }

        int getSize() const {
            int size = 0;
            Node* current = head;
            while (current) {
                ++size;
                current = current->next;
            }
            return size;
        }
};

int main() {
    LinkedList list;
    string filename = "list_data.txt";

    while (true) {
        cout << "\n1. Add elements to the list" << endl;
        cout << "2. Remove elements from the list" << endl;
        cout << "3. Print the list" << endl;
        cout << "4. Save the list to a file" << endl;
        cout << "5. Load the list from a file" << endl;
        cout << "\nChoose an action: ";

        int choice;
        cin >> choice;

        if (cin.fail()) {
            cout << "Invalid input. Exiting program." << endl;
            break;
        }

        switch (choice) {
        case 1:
            int position, diapazone;
            cout << "Enter the position of the first element: ";
            cin >> position;
            cout << "Enter the number of new items: ";
            cin >> diapazone;

            if (cin.fail() || position < 0 || diapazone <= 0) {
                cout << "Invalid input. Exiting program." << endl;
                return 1;
            }
```

```cpp
            cout << "Enter the position of the first element: ";
            cin >> position;
            cout << "Enter the number of new items: ";
            cin >> diapazone;

            if (cin.fail() || position < 0 || diapazone <= 0) {
                cout << "Invalid input. Exiting program." << endl;
                return 1;
            }

            list.addElement(position, diapazone);
            cout << "Elements added successfully." << endl;
            break;

        case 2:
            int startPos, count;
            cout << "Enter the starting position: ";
            cin >> startPos;
            cout << "Enter the number of elements to remove: ";
            cin >> count;

            if (cin.fail() || startPos < 0 || count <= 0) {
                cout << "Invalid input. Exiting program." << endl;
                return 1;
            }

            list.removeElements(startPos, count);
            cout << "Elements removed successfully." << endl;
            break;

        case 3:
            cout << "The list is: ";
            list.printList();
            break;

        case 4:
            list.saveToFile(filename);
            cout << "List saved to file successfully." << endl;
            break;

        case 5:
            list.loadFromFile(filename);
            cout << "List loaded from file successfully." << endl;
            break;

        default:
            cout << "Invalid input. Exiting program." << endl;
            return 1;
        }
    }

    return 0;
}
```

Орієнтовний час виконання: 2 год

Реальний час виконання: 2.5 год

Завдання №2

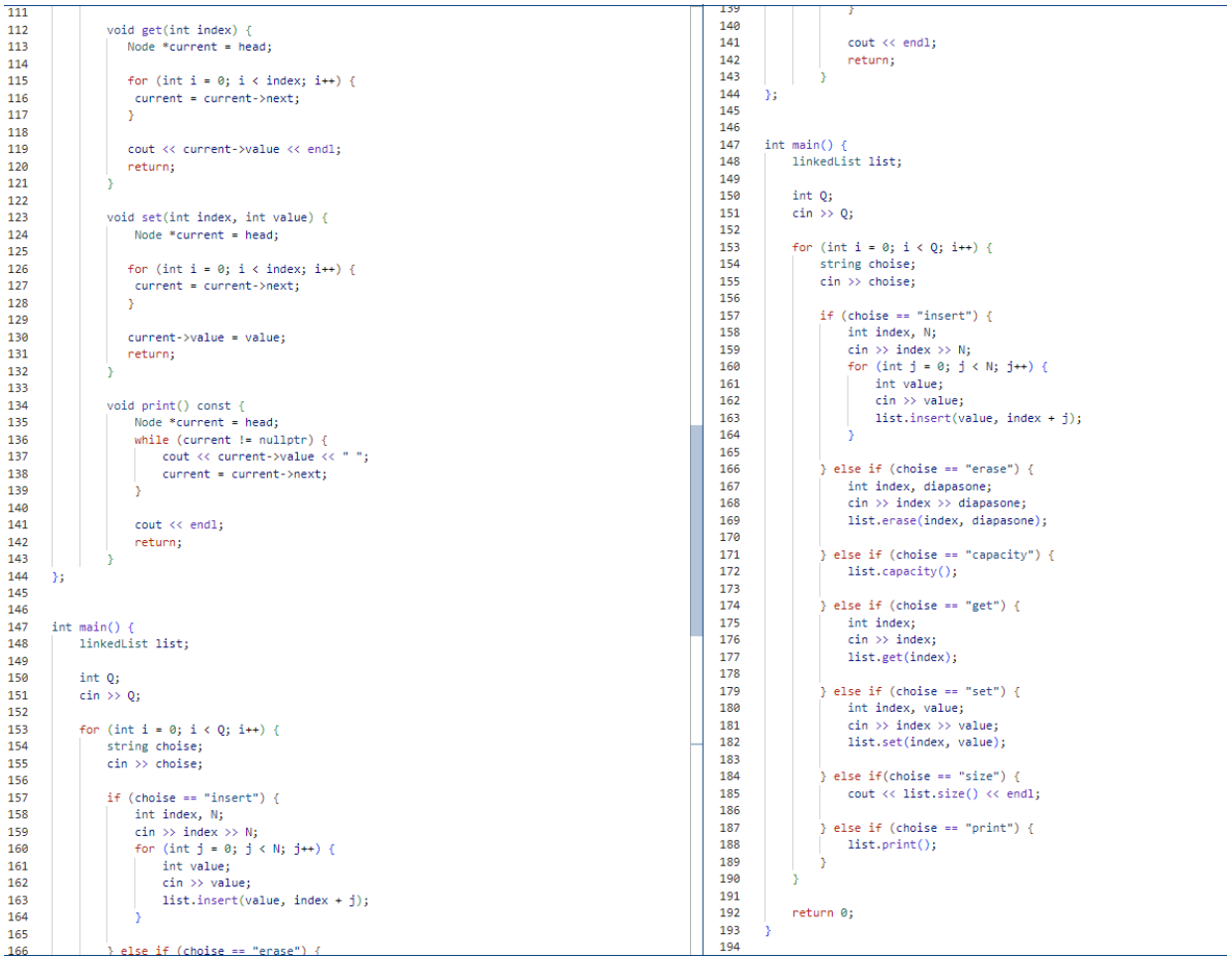
```cpp
#include <iostream>
#include <cstdint>

using namespace std;

void toggleRow(uint64_t &board, int row) {
    for (int col = 0; col < 8; col++) {
        board ^= (1ULL << (row * 8 + col));
    }
}

void toggleColumn(uint64_t &board, int col) {
    for (int row = 0; row < 8; row++) {
        board ^= (1ULL << (row * 8 + col));
    }
}

int main() {
    uint64_t board;
    int n;

    cin >> board >> n;

    for (int i = 0; i < n; i++) {
        int r, c;
        cin >> r >> c;

        r--;
        c--;

        toggleRow(board, r);

        toggleColumn(board, c);

        board ^= (1ULL << (r * 8 + c));
    }

    cout << board << endl;

    return 0;
}

```

Орієнтовний час виконання: 3 год

Реальний час виконання: 3.5 год

Завдання №3

```cpp
#include <iostream>

using namespace std;


struct Node {
    int value;
    Node *next;
};


class linkedList {
    private:
        Node* head;

        void deleteNode(Node*& current) {
            Node* temp = current;
            current = current->next;
            delete temp;
        }

        void destroyTree() {
            while (head != nullptr) {
                deleteNode(head);
            }
        }

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

        ~linkedList() {
            destroyTree();
        }

        void insert(int value, int position) {
            Node *newNode = new Node{value, nullptr};

            if (position == 0) {
                newNode->next = head;
                head = newNode;
                return;
            }

            Node *current = head;
            int currentIndex = 0;

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

            newNode->next = current->next;
            current->next = newNode;
            return;
        }

        void erase(int index, int diapasone) {
            if (index == 0) {
                for (int i = 0; i < diapasone && head != nullptr; i++) {
                    deleteNode(head);
                }
                return;
            }

            Node *current = head;
            int currentIndex = 0;

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

            Node *toDelete = current->next;

            for (int i = 0; i < diapasone && toDelete != nullptr; i++) {
                deleteNode(toDelete);
            }

            current->next = toDelete;
            return;
        }

        int size() {
            Node *current = head;

            int listSize = 0;
            while (current != nullptr) {
                current = current->next;
                listSize++;
            }

            return listSize;
        }

        void capacity() {
            Node *current = head;
            if (current == nullptr) {
                cout << 1 << endl;
                return;
            }

            int listSize = size();

            int num = 2;
            while (listSize >= num) {
                num *= 2;
            }
            cout << num << endl;
            return;
        }

        void get(int index) {
```



```
111
112         void get(int index) {
113            Node *current = head;
114
115            for (int i = 0; i < index; i++) {
116             current = current->next;
117            }
118
119            cout << current->value << endl;
120            return;
121         }
122
123         void set(int index, int value) {
124             Node *current = head;
125
126            for (int i = 0; i < index; i++) {
127             current = current->next;
128            }
129
130            current->value = value;
131            return;
132         }
133
134         void print() const {
135             Node *current = head;
136             while (current != nullptr) {
137                 cout << current->value << " ";
138                 current = current->next;
139             }
140
141             cout << endl;
142             return;
143         }
144 };
145
146
147 int main() {
148     linkedList list;
149
150     int Q;
151     cin >> Q;
152
153     for (int i = 0; i < Q; i++) {
154         string choise;
155         cin >> choise;
156
157         if (choise == "insert") {
158             int index, N;
159             cin >> index >> N;
160             for (int j = 0; j < N; j++) {
161                 int value;
162                 cin >> value;
163                 list.insert(value, index + j);
164             }
165
166         } else if (choise == "erase") {
```

```
139             }
140
141             cout << endl;
142             return;
143         }
144 };
145
146
147 int main() {
148     linkedList list;
149
150     int Q;
151     cin >> Q;
152
153     for (int i = 0; i < Q; i++) {
154         string choise;
155         cin >> choise;
156
157         if (choise == "insert") {
158             int index, N;
159             cin >> index >> N;
160             for (int j = 0; j < N; j++) {
161                 int value;
162                 cin >> value;
163                 list.insert(value, index + j);
164             }
165
166         } else if (choise == "erase") {
167             int index, diapasone;
168             cin >> index >> diapasone;
169             list.erase(index, diapasone);
170
171         } else if (choise == "capacity") {
172             list.capacity();
173
174         } else if (choise == "get") {
175             int index;
176             cin >> index;
177             list.get(index);
178
179         } else if (choise == "set") {
180             int index, value;
181             cin >> index >> value;
182             list.set(index, value);
183
184         } else if(choise == "size") {
185             cout << list.size() << endl;
186
187         } else if (choise == "print") {
188             list.print();
189         }
190     }
191
192     return 0;
193 }
194
```

Орієнтовний час виконання: 3 год

Реальний час виконання: 3 год

Завдання №4

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

struct TreeNode {
    int key;
    TreeNode* left;
    TreeNode* right;

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

class BinaryTree {
private:
    TreeNode* root;
    long long treeSize;

    void clear(TreeNode* node) {
        if (!node) return;
        clear(node->left);
        clear(node->right);
        delete node;
    }

public:
    BinaryTree() : root(nullptr), treeSize(0) {}

    ~BinaryTree() {
        clear(root);
    }

    void insert(int value) {
        if (contains(value, root) == false) {
            if (root == nullptr) {
                root = new TreeNode{value};
                treeSize++;
                return;
            }

            TreeNode *current = root;
            while(true) {
                if (value > current->key) {
                    if (current->right != nullptr) current = current->right;
                    else {
                        current->right = new TreeNode{value};
                        treeSize++;
                        return;
                    }

                } else if (value < current->key) {
                    if (current->left != nullptr) current = current->left;
                    else {
                        current->left = new TreeNode{value};
                        treeSize++;
                        return;
                    }
                }
            }
        }

    void containsCall(int value) {
        if (contains(value, root) == true) cout << "Yes" << endl;
        else cout << "No" << endl;
    }

    bool contains(int value, TreeNode *current) {
        if (current == nullptr) {
            return false;
        }
        if (value == current->key) {
            return true;
        }
        if (value < current->key) {
            return contains(value, current->left);
        } else {
            return contains(value, current->right);
        }
    }

    void size() {
        cout << treeSize << endl;
    }

    void printCall() {
        print(root);
    }

    void print(TreeNode* node) {
        if (node == nullptr) return;
        print(node->left);
        cout << node->key << " ";
        print(node->right);
    }

};

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

    while (Q--) {
        string command;
        cin >> command;

        if (command == "insert") {
            int value;
            cin >> value;
            tree.insert(value);
```

```cpp
int main() {
    int Q;
    cin >> Q;
    BinaryTree tree;

    while (Q--) {
        string command;
        cin >> command;

        if (command == "insert") {
            int value;
            cin >> value;
            tree.insert(value);

        } else if (command == "contains") {
            int value;
            cin >> value;
            tree.containsCall(value);

        } else if (command == "size"){
            tree.size();

        } else if (command == "print") {
           tree.printCall();
           cout << endl;
        }
    }

    return 0;
}

```

Орієнтовний час виконання: 1 год

Реальний час виконання: 1.5 год

Завдання №5

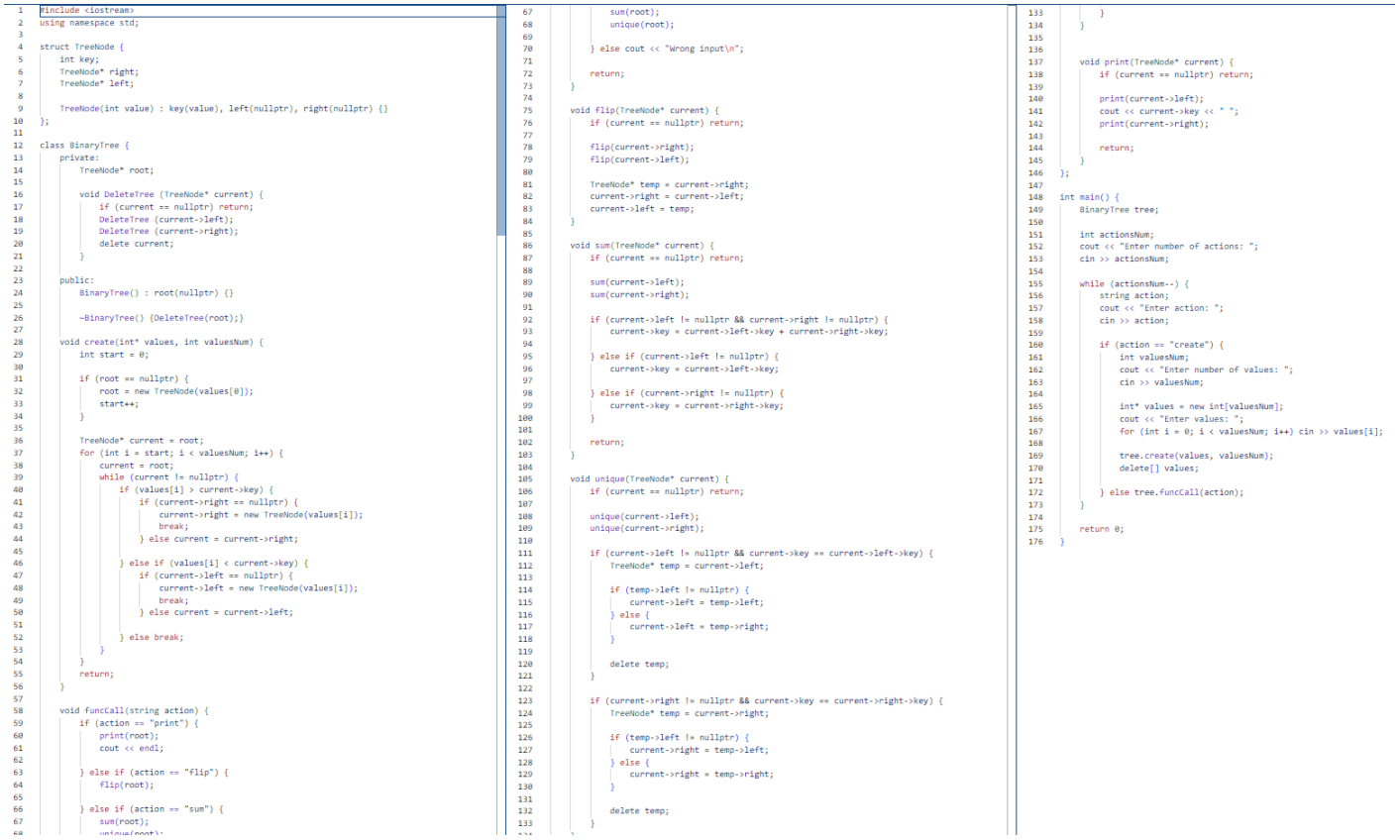
```cpp
#include <iostream>
using namespace std;

struct Node {
    int value;
    Node* next;

    Node(int key) : value(key), next(nullptr) {}
};

class LinkedList {
private:
    Node* head1;
    Node* head2;
    Node* head3;

    void deleteNode(Node*& current) {
        Node* temp = current;
        current = current->next;
        delete temp;
    }

    void destroyLists() {
        while (head1) {
            deleteNode(head1);
        }
        while (head2) {
            deleteNode(head2);
        }
        while (head3) {
            deleteNode(head3);
        }
    }

public:
    LinkedList() : head1(nullptr), head2(nullptr), head3(nullptr) {}

    ~LinkedList() {
        destroyLists();
    }

    void createCaller(int headNum, int* values, int valuesNum) {
        switch (headNum) {
        case 1:
            create(head1, values, valuesNum);
            break;
        case 2:
            create(head2, values, valuesNum);
            break;
        case 3:
            create(head3, values, valuesNum);
            break;
        default:
            cout << "Invalid list number" << endl;
        }
    }

    void create(Node*& head, int* values, int valuesNum) {
        Node* current = head;
        int start = 0;

        if (current == nullptr) {
            head = new Node(values[0]);
            current = head;
            start++;

        } else {
            while (current->next != nullptr) {
                current = current->next;
            }
        }

        for (int i = start; i < valuesNum; i++) {
            current->next = new Node(values[i]);
            current = current->next;
        }
    }

    void funcCaller(string action, int listNum) {
        if (action == "reverse") {
            switch (listNum) {
            case 1: reverse(head1); break;
            case 2: reverse(head2); break;
            case 3: reverse(head3); break;
            default: cout << "Invalid list number" << endl;
            }
        } else if (action == "print") {
            switch (listNum) {
            case 1: print(head1); break;
            case 2: print(head2); break;
            case 3: print(head3); break;
            default: cout << "Invalid list number" << endl;
            }
        } else {
            cout << "Invalid action" << endl;
        }
    }

    void reverse(Node*& head) {
        Node* curNode = head;
        Node* nextNode = nullptr;
        Node* prevNode = nullptr;

        while (curNode != nullptr) {
            nextNode = curNode->next;
            curNode->next = prevNode;
            prevNode = curNode;
            curNode = nextNode;
        }
        head = prevNode;
    }

    }

    void add() {
        Node* current1 = head1;
        Node* current2 = head2;
        Node* current3 = head3;

        while (current1 != nullptr || current2 != nullptr) {
            int value = 0;

            if (current1 != nullptr) {
                value += current1->value;
                current1 = current1->next;
            }

            if (current2 != nullptr) {
                value += current2->value;
                current2 = current2->next;
            }

            Node* newNode = new Node(value);

            if (head3 == nullptr) {
                head3 = newNode;
                current3 = head3;
            } else {
                current3->next = newNode;
                current3 = newNode;
            }
        }
    }

    bool compare() {
        Node* current1 = head1;
        Node* current2 = head2;

        while (current1 != nullptr && current2 != nullptr) {
            if (current1->value != current2->value)
                return false;
            current1 = current1->next;
            current2 = current2->next;
        }

        return current1 == nullptr && current2 == nullptr;
    }


    void print(Node* head) {
        Node* current = head;
        while (current != nullptr) {
            cout << current->value << " ";
            current = current->next;
        }
        cout << endl;
    }
};
```

```cpp
    void print(Node* head) {
        Node* current = head;
        while (current != nullptr) {
            cout << current->value << " ";
            current = current->next;
        }
        cout << endl;
    }
};

int main() {
    LinkedList list;

    int actionsNum;
    cout << "Enter number of actions: ";
    cin >> actionsNum;

    while (actionsNum--) {
        string action;
        cout << "Enter action: ";
        cin >> action;

        if (action == "add") {
            list.add();
            continue;
        } else if (action == "compare") {
            if (bool answer = list.compare()) cout << "true\n";
            else cout << "false\n";

            continue;
        }

        int headNum;
        cout << "Enter list number: ";
        cin >> headNum;

        if (action == "create") {
            int valuesNum;
            cout << "Enter number of values: ";
            cin >> valuesNum;

            int* values = new int[valuesNum];
            cout << "Enter values: ";
            for (int i = 0; i < valuesNum; i++) cin >> values[i];

            list.createCaller(headNum, values, valuesNum);
            delete[] values;

        } else {
            list.funcCaller(action, headNum);
        }
    }

    return 0;
}
```

Орієнтовний час виконання: 1 год

Реальний час виконання: 1.5 год

Завдання №6



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

struct TreeNode {
    int key;
    TreeNode* right;
    TreeNode* left;

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

class BinaryTree {
    private:
        TreeNode* root;

        void DeleteTree (TreeNode* current) {
            if (current == nullptr) return;
            DeleteTree (current->left);
            DeleteTree (current->right);
            delete current;
        }

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

        ~BinaryTree() {DeleteTree(root);}

    void create(int* values, int valuesNum) {
        int start = 0;

        if (root == nullptr) {
            root = new TreeNode(values[0]);
            start++;
        }

        TreeNode* current = root;
        for (int i = start; i < valuesNum; i++) {
            current = root;
            while (current != nullptr) {
                if (values[i] > current->key) {
                    if (current->right == nullptr) {
                        current->right = new TreeNode(values[i]);
                        break;
                    } else current = current->right;

                } else if (values[i] < current->key) {
                    if (current->left == nullptr) {
                        current->left = new TreeNode(values[i]);
                        break;
                    } else current = current->left;

                } else break;
            }
        }
        return;
    }

    void funcCall(string action) {
        if (action == "print") {
            print(root);
            cout << endl;

        } else if (action == "flip") {
            flip(root);

        } else if (action == "sum") {
            sum(root);
            unique(root);

        } else cout << "Wrong input\n";

        return;
    }

    void flip(TreeNode* current) {
        if (current == nullptr) return;

        flip(current->right);
        flip(current->left);

        TreeNode* temp = current->right;
        current->right = current->left;
        current->left = temp;
    }

    void sum(TreeNode* current) {
        if (current == nullptr) return;

        sum(current->left);
        sum(current->right);

        if (current->left != nullptr && current->right != nullptr) {
            current->key = current->left->key + current->right->key;

        } else if (current->left != nullptr) {
            current->key = current->left->key;

        } else if (current->right != nullptr) {
            current->key = current->right->key;
        }

        return;
    }

    void unique(TreeNode* current) {
        if (current == nullptr) return;

        unique(current->left);
        unique(current->right);

        if (current->left != nullptr && current->key == current->left->key) {
            TreeNode* temp = current->left;

            if (temp->left != nullptr) {
                current->left = temp->left;
            } else {
                current->left = temp->right;
            }

            delete temp;
        }

        if (current->right != nullptr && current->key == current->right->key) {
            TreeNode* temp = current->right;

            if (temp->left != nullptr) {
                current->right = temp->left;
            } else {
                current->right = temp->right;
            }

            delete temp;
        }
    }

    void print(TreeNode* current) {
        if (current == nullptr) return;

        print(current->left);
        cout << current->key << " ";
        print(current->right);

        return;
    }
};

int main() {
    BinaryTree tree;

    int actionsNum;
    cout << "Enter number of actions: ";
    cin >> actionsNum;

    while (actionsNum--) {
        string action;
        cout << "Enter action: ";
        cin >> action;

        if (action == "create") {
            int valuesNum;
            cout << "Enter number of values: ";
            cin >> valuesNum;

            int* values = new int[valuesNum];
            cout << "Enter values: ";
            for (int i = 0; i < valuesNum; i++) cin >> values[i];

            tree.create(values, valuesNum);
            delete[] values;

        } else tree.funcCall(action);
    }

    return 0;
}
```

Орієнтовний час виконання: 1 год

Реальний час виконання: 1 год

Завдання №7

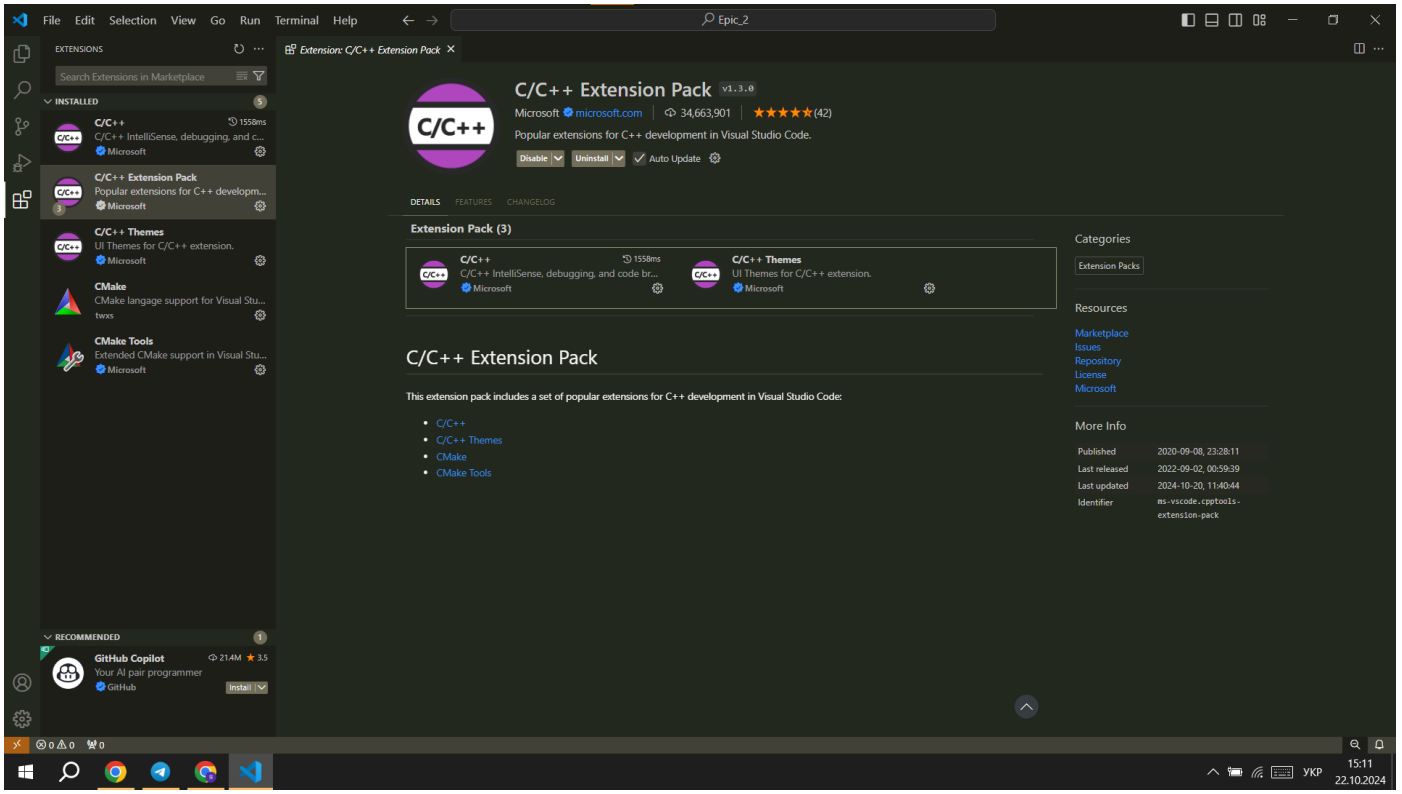
```cpp
#include <iostream>
#include <vector>
#include <cmath>
using namespace std;

long long calculation(long long a, long long b) {
    if (b == 0) return a;
    else return calculation(b, a % b);
}

int main() {
    long long a, b, c, d;
    cin >> a >> b >> c >> d;

    long long totalCombo = (b - a + 1) * (d - c + 1);

    int n;
    cin >> n;

    vector<long long> participants(n);
    for (int i = 0; i < n; i++) cin >> participants[i];

    for (int i = 0; i < n; i++) {
        long long number = participants[i];
        long long winCombo = 0;

        for (long long j = 1; j * j <= number; j++) {
            if (number % j == 0) {
                long long x1 = j;
                long long x2 = number / j;

                if (x1 >= a && x1 <= b && x2 >= c && x2 <= d) winCombo++;
                if (x1 != x2 && x2 >= a && x2 <= b && x1 >= c && x1 <= d) winCombo++;
            }
        }

        long long divisor = calculation(winCombo, totalCombo);
        cout << (winCombo / divisor) << "/" << (totalCombo / divisor) << endl;
    }

    return 0;
}
```

Орієнтовний час виконання: 1 год

Реальний час виконання: 1 год

Частина 3

Для виконання роботи використовується середовище **Visual Studio Code** зі встановленим розширенням **C/C++ Extension Pack.**



Використані бібліотеки:

- **iostream**
- **vector**
- **cmath**
- **fstream**
- **random**
- **csdint**

Частина 4

Завдання №

Частина 5

Завдання №1

```
1. Add elements to the list
2. Remove elements from the list
3. Print the list
4. Save the list to a file
5. Load the list from a file

Choose an action: 1
Enter the position of the first element: 0
Enter the number of new items: 5
Elements added successfully.

1. Add elements to the list
2. Remove elements from the list
3. Print the list
4. Save the list to a file
5. Load the list from a file

Choose an action: 3
The list is: 23 -> 97 -> 93 -> 23 -> 30 -> null

1. Add elements to the list
2. Remove elements from the list
3. Print the list
4. Save the list to a file
5. Load the list from a file

Choose an action: 2
Enter the starting position: 4
Enter the number of elements to remove: 2
Elements removed successfully.

1. Add elements to the list
2. Remove elements from the list
3. Print the list
4. Save the list to a file
5. Load the list from a file

Choose an action: 3
The list is: 23 -> 97 -> 93 -> 23 -> null

1. Add elements to the list
2. Remove elements from the list
3. Print the list
4. Save the list to a file
5. Load the list from a file

Choose an action: dsfcadsf
Invalid input. Exiting program.
```

Завдання №2

```
0
4
1 1
1 2
2 2
2 1
771
```

Завдання №3

```
12

size
0

insert 0 5
251 252 253 254 255

size
5
capacity
8
print
251 252 253 254 255

get 1
252
set 1 777
get 1
777

erase 1 3

get 1
255

size
2
print
251 255
```

Завдання №4

```
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 №5

```
Enter number of actions: 3
Enter action: create
Enter list number: 1
Enter number of values: 5
Enter values: 1 2 3 4 5
Enter action: reverse
Enter list number: 1
Enter action: print
Enter list number: 1
5 4 3 2 1
```

Завдання №6

```
Enter number of actions: 4
Enter action: create
Enter number of values: 9
Enter values: 5 3 8 6 9 2 7 1 4
Enter action: flip
Enter action: sum
Enter action: print
9 16 7 21 4 5 1
```

Завдання №7

```
1 3 4 7
3
12
4
47
1/6
1/12
0/1
```

Висновок

У цій роботі я здобув практичні навички роботи з лінкованими списками та бінарними деревами в C++. Я навчився створювати, модифікувати, видаляти елементи структур даних, а також реалізовувати алгоритми пошуку, вставки та обходу. Це заклало міцну основу для подальшого вивчення складніших алгоритмів і структур даних.