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 групи ШІ-13
Басараб Дмитрій Богданович

Львів 2024

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

Мета роботи: *Навчитись працювати з Динамічними структурами (Черга, Стек, Списки, Дерево), вивчити Алгоритми обробки динамічних структур*

Теоретичні відомості:

1) C++
   https://www.youtube.com/watch?v=2UDMGCcRCjo&list=PLiPRE8VmJzOpn6PzYf0higmCEyGzo2A5g

Виконання роботи:

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

Завдання №1 - VNS Lab 10 var 20

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.

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.

*20.Записи в лінійному списку містять ключове поле типу *char (рядок символів). Сформувати двонаправлений список. Знищити елемент із заданим ключем. Додати по К елементів на початок й в кінець списку.*

Завдання №2 - Algotester Lab78v1

Lab 78v1

Limits: 2 sec., 256 MiB

Ваше завдання - власноруч реалізувати структуру даних "Двозв'язний список".
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 списку, який треба вставити.
У третьому рядку N цілих чисел - список, який треба вставити на позицію index

Видалення:
Ідентифікатор - erase
Ви отримуєте 2 цілих числа - index 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 списку.

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 списку на екран:
Ідентифікатор - print
Ви не отримуєте аргументів.
Ви виводите усі елементи списку через пробіл.

Input

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

Output

Відповіді на запити у визначеному в умові форматі.

Завдання №3 - Algotester Lab5v3

Lab 5v3

Limits: 1 sec., 256 MiB

У вас є карта гори розміром N×M.

Також ви знаєте координати {x,y}, у яких знаходиться вершина гори.

Ваше завдання - розмалювати карту таким чином, щоб найнижча точка мала число 0, а пік гори мав найбільше число.

Клітинкі які мають суміжну сторону з вершиною мають висоту на один меншу, суміжні з ними і не розфарбовані мають ще на 1 меншу висоту і так далі.

Input

У першому рядку 2 числа N та M - розміри карти

у другому рядку 2 числа x та y - координати піку гори

Output

N рядків по M елементів в рядку через пробіл - висоти карти.

Завдання №4 - Practice task 1-3 Linked List

Зв’язаний список

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;

Пояснення прикладу

Спочатку ми визначаємо просту структуру ***Node*** для нашого пов’язаного списку.
Потім функція ***reverse*** ітеративно змінює список, маніпулюючи наступними покажчиками кожного вузла.
printList — допоміжна функція для відображення списку.
Основна функція створює зразок списку, демонструє реверсування та друкує вихідний і обернений списки.

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***.

Пояснення прикладу

- Для пов'язаного списку визначено структуру ***Node***.
- Функція ***compare*** ітеративно проходить обидва списки одночасно, порівнюючи дані в кожному вузлі.
- Якщо виявлено невідповідність даних або якщо довжина списків різна (один список закінчується раніше іншого), функція повертає ***false***.
- Основна функція ***main*** створює два списки та демонструє порівняння.

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

Завдання №5 - Practice task 4-5 Binary Tree

Задача №4 - Віддзеркалення дерева

TreeNode *create_mirror_flip(TreeNode *root);
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

Задача №5 - Записати кожному батьківському вузлу суму підвузлів

void tree_sum(TreeNode *root);
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

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

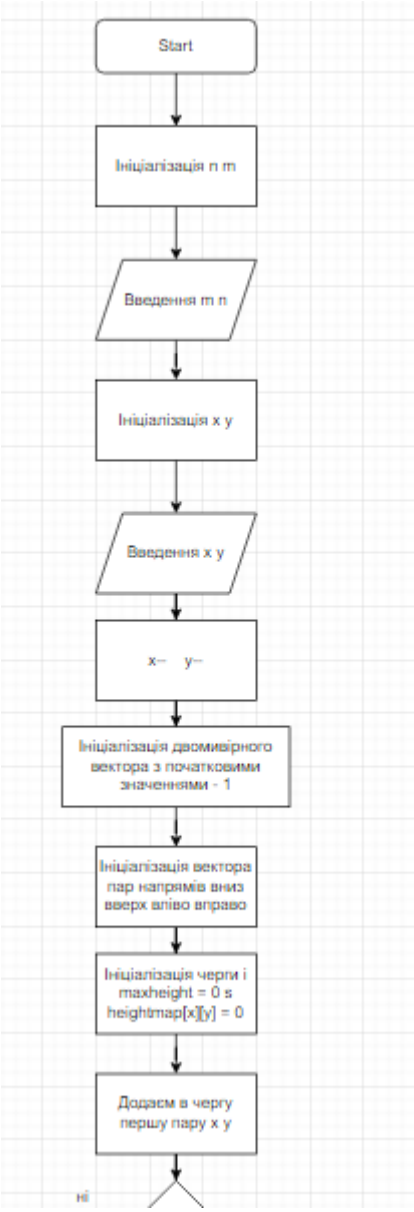
Завдання №1 - VNS Lab 10 var 20

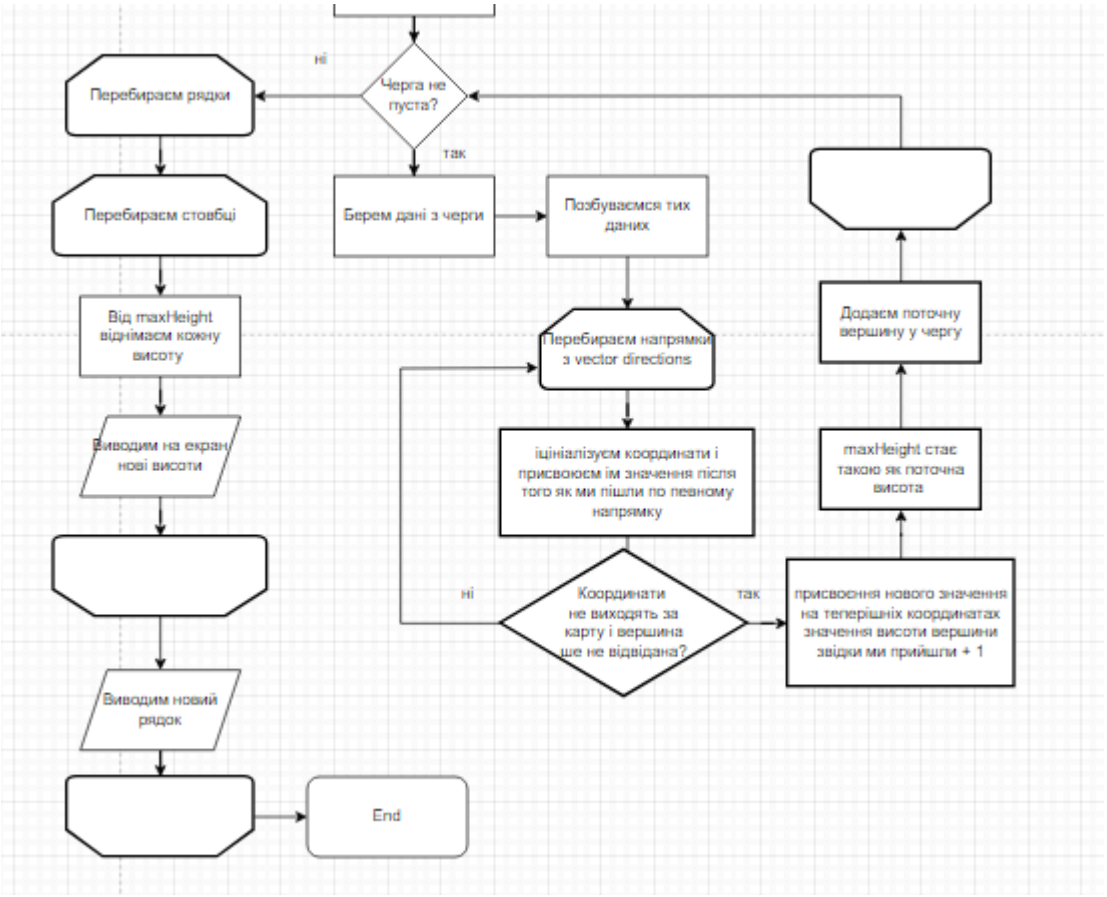
Планований час 6 год

Завдання №2 - Algotester Lab78v1

Планований час 8 год

Завдання №3 - Algotester Lab5v3





Планований час 4 год

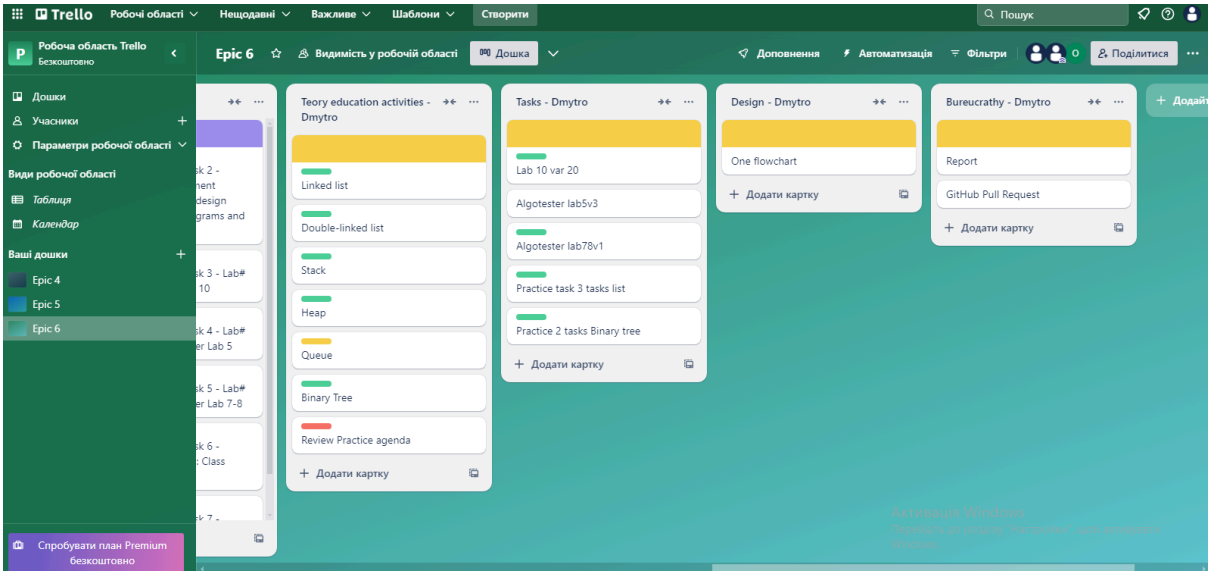
Завдання №4 - Practice task 1-3 Linked List

Планований час 5 год

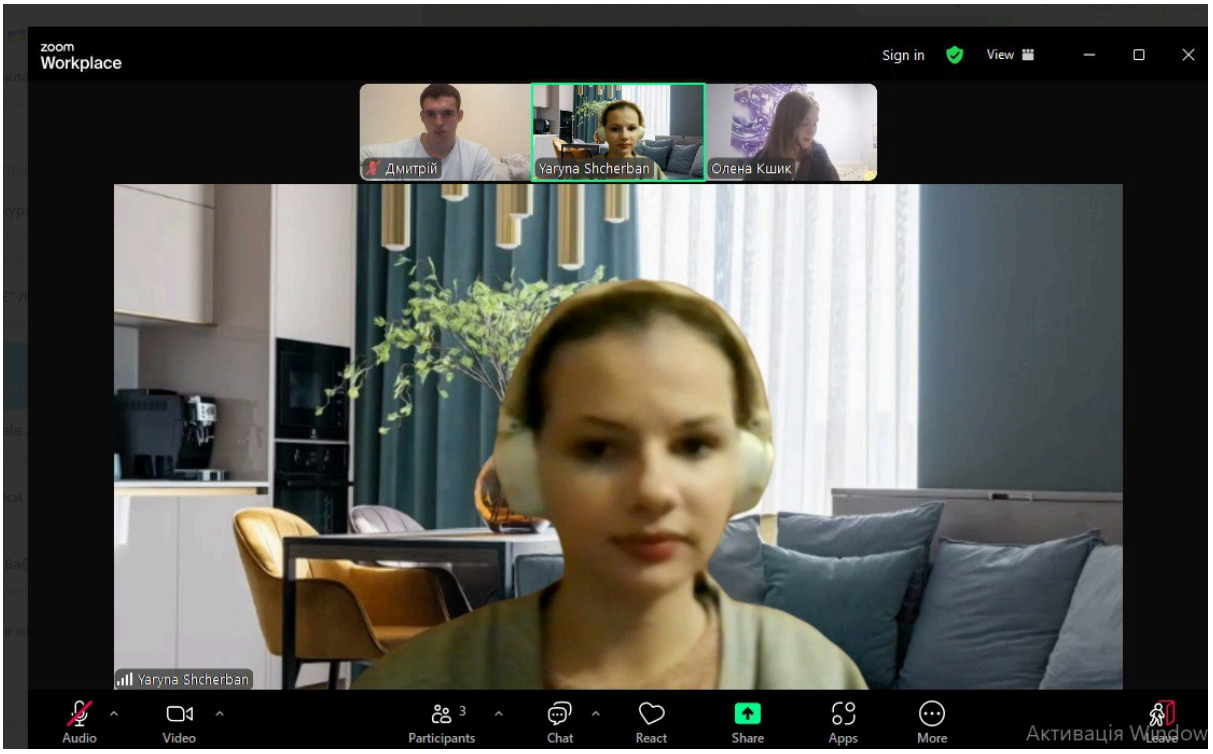
Завдання №5 - Practice task 4-5 Binary Tree

Планований час 4 год

3) *Конфігурація середовища до виконання завдань:*



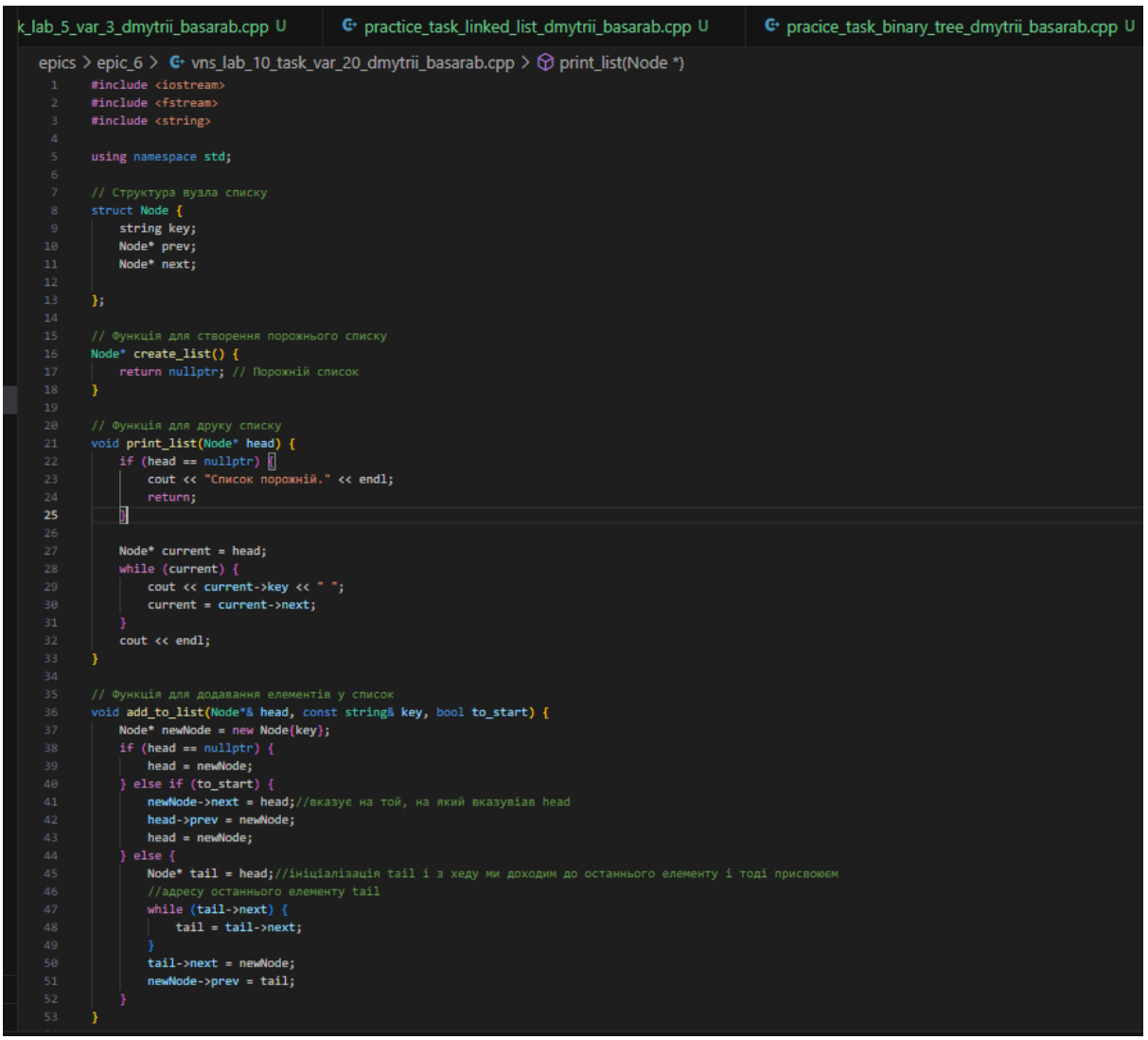
Trello



Робота в команді

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

Завдання №1 - VNS Lab 10 var 20

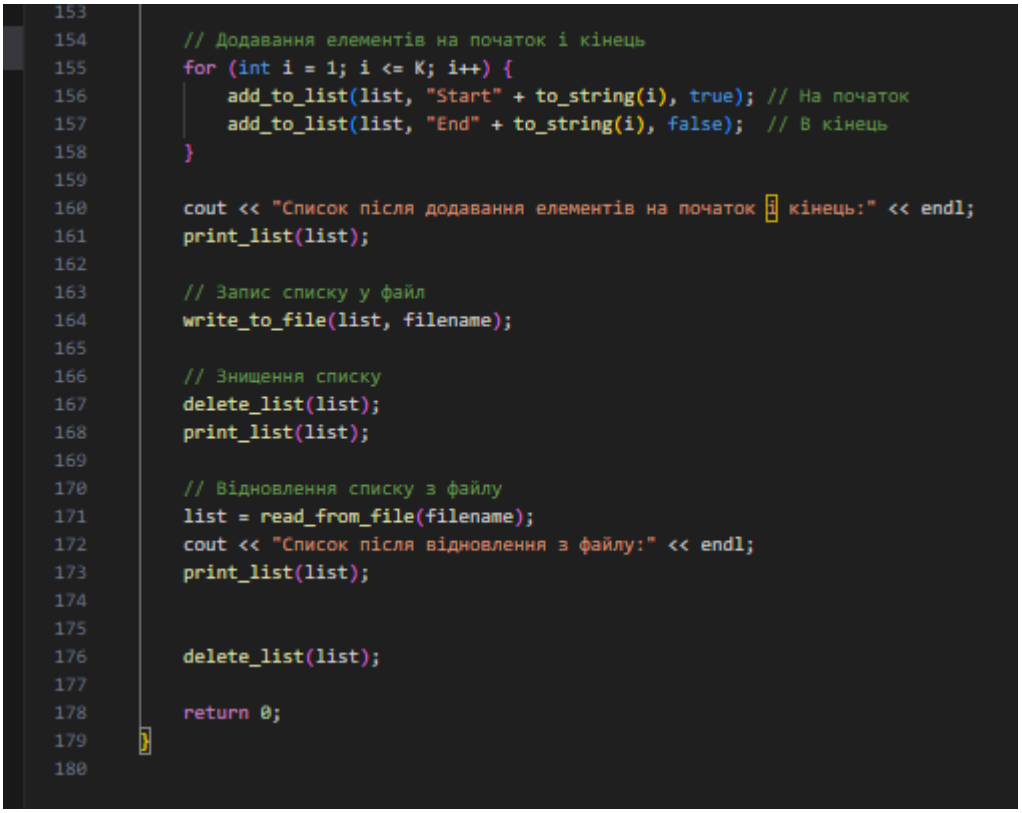

k_lab_5_var_3_dmytrii_basarab.cpp U
practice_task_linked_list_dmytrii_basarab.cpp U
pracice_task_binary_tree_dmytrii_basarab.cpp U
epics > epic_6 > vns_lab_10_task_var_20_dmytrii_basarab.cpp > print_list(Node *)
#include <iostream>
#include <fstream>
#include <string>
using namespace std;
// Структура вузла списку
struct Node {
string key;
Node* prev;
Node* next;
};
// Функція для створення порожнього списку
Node* create_list() {
return nullptr; // Порожній список
}
// Функція для друку списку
void print_list(Node* head) {
if (head == nullptr) {
cout << "Список порожній." << endl;
return;
}
Node* current = head;
while (current) {
cout << current->key << " ";
current = current->next;
}
cout << endl;
}
// Функція для додавання елементів у список
void add_to_list(Node*& head, const string& key, bool to_start) {
Node* newNode = new Node{key};
if (head == nullptr) {
head = newNode;
} else if (to_start) {
newNode->next = head;//вказує на той, на який вказував head
head->prev = newNode;
head = newNode;
} else {
Node* tail = head;//ініціалізація tail і з хеду ми доходим до останнього елементу і тоді присвоюєм
//адресу останнього елементу tail
while (tail->next) {
tail = tail->next;
}
tail->next = newNode;
newNode->prev = tail;
}
}

k_lab_5_var_3_dmytrii_basarab.cpp U | practice_task_linked_list_dmytrii_basarab.cpp U | pracice_task_binary_tree_dmy

epics > epic_6 > vns_lab_10_task_var_20_dmytrii_basarab.cpp > add_to_list(Node *&, const string &, bool)

```
36  void add_to_list(Node*& head, const string& key, bool to_start) {
44      } else {
53  }
54
55  // Функція для знищення елемента за ключем
56  void delete_by_key(Node*& head, const string& key) {
57      if (head == nullptr) {
58          cout << "Список порожній." << endl;
59          return;
60      }
61
62      Node* current = head;
63      while (current && current->key != key) {
64          current = current->next;
65      }
66
67      if (current == nullptr) {
68          cout << "Елемент із ключем '" << key << "' не знайдено." << endl;
69          return;
70      }
71
72      if (current->prev) {
73          current->prev->next = current->next;
74      } else {//ми на першому елементі
75          head = current->next;
76      }
77
78      if (current->next) {
79          current->next->prev = current->prev;
80      }
81
82      delete current;
83      cout << "Елемент із ключем '" << key << "' видалено." << endl;
84  }
85
86  // Функція для запису списку у файл
87  void write_to_file(Node* head, const string& filename) {
88      ofstream file(filename);
89      if (!file) {
90          cout << "Помилка відкриття файлу!" << endl;
91          return;
92      }
93
94      Node* current = head;
95      while (current) {
96          file << current->key << endl;
97          current = current->next;
98      }
99
100     file.close();
101     cout << "Список записано у файл '" << filename << "'." << endl;
102 }
103
```

```cpp
// Функція для відновлення списку з файлу
Node* read_from_file(const string& filename) {
    ifstream file(filename);
    if (!file) {
        cout << "Помилка відкриття файлу!" << endl;
        return nullptr;
    }

    Node* head = nullptr;
    string key;
    while (getline(file, key)) {
        add_to_list(head, key, false);
    }

    file.close();
    cout << "Список відновлено з файлу '" << filename << "'." << endl;
    return head;
}

// Функція для знищення списку
void delete_list(Node*& head) {
    while (head) {
        Node* temp = head;
        head = head->next;
        delete temp;//ми зберегли елемент, темп прирівняли до
        //того на який вказує хед і переставили хед на наступний
        //так ми не втратили отой елемент який тре видалити
    }
    cout << "Список знищено." << endl;
}

// Основна програма
int main() {
    Node* list = create_list();
    int K = 2; // Кількість елементів для додавання на початок і кінець
    string filename = "list.txt";

    // Додавання елементів
    add_to_list(list, "Alpha", false);
    add_to_list(list, "Beta", false);
    add_to_list(list, "Gamma", false);

    cout << "Список після додавання елементів:" << endl;
    print_list(list);

    // Видалення елемента
    delete_by_key(list, "Beta");
    cout << "Список після видалення елемента 'Beta':" << endl;
    print_list(list);
```



```
153
154     // Додавання елементів на початок і кінець
155     for (int i = 1; i <= K; i++) {
156         add_to_list(list, "Start" + to_string(i), true); // На початок
157         add_to_list(list, "End" + to_string(i), false);  // В кінець
158     }
159
160     cout << "Список після додавання елементів на початок і кінець:" << endl;
161     print_list(list);
162
163     // Запис списку у файл
164     write_to_file(list, filename);
165
166     // Знищення списку
167     delete_list(list);
168     print_list(list);
169
170     // Відновлення списку з файлу
171     list = read_from_file(filename);
172     cout << "Список після відновлення з файлу:" << endl;
173     print_list(list);
174
175
176     delete_list(list);
177
178     return 0;
179 }
180
```

Завдання №2 - Algotester Lab78v1

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

using namespace std;

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

// Глобальні змінні для списку
Node* head = nullptr;
Node* tail = nullptr;
size_t listSize = 0;

// Додавання елементів на задану позицію
void insert(int index, const vector<int>& values) {
    if (index < 0 || index > listSize) {
        cerr << "Неправильне число" << endl;
    }

    for (int val : values) {
        Node* newNode = new Node{val};

        if (index == 0) { // Вставка на початок
            if (head == nullptr) {// Коли елементів нема
                tail = newNode;
                head = tail;
            } else {// Коли 1 елемент то перед тим вставояєм
                newNode->next = head;
                head->prev = newNode;
                head = newNode;
            }
        } else if (index == listSize) { // Вставка в кінець
            newNode->prev = tail;
            tail->next = newNode;
            tail = newNode;
        } else { // Вставка в середину
            Node* current = head;
            for (int i = 0; i < index; ++i) {// Доходим до потібного індексу
                current = current->next;
            }
            newNode->next = current;// Тре його вставити перед елеметом ло якого ми дойшли
            newNode->prev = current->prev;
            if (current->prev) {// То коли ми в 0 індексі
                current->prev->next = newNode;
            }
            current->prev = newNode;
        }
```

```
53          ++index;
54          ++listSize;
55      }
56  }
57
58  // Видалення елементів, починаючи з індексу
59  void erase(int index, int n) {
60      if (index < 0 || index >= listSize || n < 0 || index + n > listSize) {
61          cerr << "Неправильне число" << endl;
62      }
63
64      Node* current = head;
65      for (int i = 0; i < index; ++i) {
66          current = current->next;
67      }
68  // Доходим до індекса
69      for (int i = 0; i < n; ++i) {//повторюєм скільки тре
70          Node* toDelete = current;
71          if (current->prev) {// З'єднуєм вперед
72              current->prev->next = current->next;
73          } else {// Якшо то перший елемент
74              head = current->next;
75          }
76          if (current->next) { // З'єднуєм назад
77              current->next->prev = current->prev;
78          } else { // Якшо то останній елемент
79              tail = current->prev;
80          }
81          current = current->next; // Ідем далі
82          delete toDelete;
83          --listSize;
84      }
85  }
86
87  // Отримання значення елемента за індексом
88  int get(int index) {
89      if (index < 0 || index >= listSize) {
90          cerr << "Неправильне число" << endl;
91      }
92
93      Node* current = head;
94      for (int i = 0; i < index; ++i) {
95          current = current->next;
96      } // Просто доходим як в минулих циклах до елемента і видаєм дані
97      return current->value;
98  }
99
100 // Модифікація значення елемента за індексом
```

```
100     // Модифікація значення елемента за індексом
101     void set(int index, int value) {
102         if (index < 0 || index >= listSize) {
103             cerr << "Неправильне число" << endl;
104         }
105 
106         Node* current = head;
107         for (int i = 0; i < index; ++i) {
108             current = current->next;
109         }
110         current->value = value;
111     }
112 
113     // Друк списку
114     void printList() {
115         Node* current = head;
116         while (current) {
117             cout << current->value << " ";
118             current = current->next;
119         }
120         cout << endl;
121     }
122 
123     // Обробка запитів
124     void processQueries() {
125         int Q;
126         cin >> Q;
127 
128         while (--Q >= 0) {
129             string command;
130             cin >> command;
131 
132             if (command == "insert") {
133                 int index, N;
134                 cin >> index >> N;
135                 vector<int> values(N);
136                 for (int i = 0; i < N; ++i) {
137                     cin >> values[i];
138                 }
139                 insert(index, values);
140 
141             } else if (command == "erase") {
142                 int index, n;
143                 cin >> index >> n;
144                 erase(index, n);
145 
146             } else if (command == "size") {
147                 cout << listSize << endl;
148 
149             } else if (command == "get") {
150                 int index;
```

```
        } else if (command == "get") {
            int index;
            cin >> index;
            cout << get(index) << endl;

        } else if (command == "set") {
            int index, value;
            cin >> index >> value;
            set(index, value);

        } else if (command == "print") {
            printList();

        } else {
            cerr << "Невідома команда: " << command << endl;
        }
    }
}

int main() {
    processQueries();
//коли ми ту роботу свою закінчили то виходим з функції і очищаєм память з х
    // Очищення пам'яті
    while (head != nullptr) {
        Node* toDelete = head;
        head = head->next;
        delete toDelete;
    }

    return 0;
}
```

Завдання №3 - Algotester Lab5v3

epics > epic_6 > algotester_task_lab_3_var_3_dmytro_basarab.cpp > ...

```cpp
#include <iostream>
#include <queue>
#include <utility>//для pair
#include <vector>

using namespace std;
int main()
{
    int n, m;
    cin >> n >> m;
    int x, y;
    cin >> x >> y;
    x--;
    y--;

    vector<vector<int>> heightMap(n, vector<int>(m, -1));
    vector<pair<int, int>> directions = {{-1, 0}, {1, 0}, {0, -1}, {0, 1}};

    queue<pair<int, int>> q;
    heightMap[x][y] = 0;
    int maxHeight = 0;

    q.push({x, y});


    while (!q.empty())
    {
        auto [cx, cy] = q.front();//взяти перший елемент з черги куди ми підем
        q.pop();
        for (auto [dx, dy] : directions) //перебирає directions
        {
            int nx = cx + dx;
            int ny = cy + dy;
            if (nx >= 0 && nx < n && ny >= 0 && ny < m && heightMap[nx][ny] == -1)
            {// знаходе шляхти шоб не вийти за межі карти і іти на ше не відвідану вершину
                heightMap[nx][ny] = heightMap[cx][cy] + 1;//бере значення з тої точки звідки прийшов
                maxHeight = heightMap[nx][ny];
                q.push({nx, ny});
            }
        }
    }
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < m; j++)
        {
            cout << maxHeight - heightMap[i][j] << ' ';
        }//таке як реверс шоб в нашій старій карті найбільше значення стало 0 а найвище maxheight
        cout << endl;
    }

    return 0;
}
```

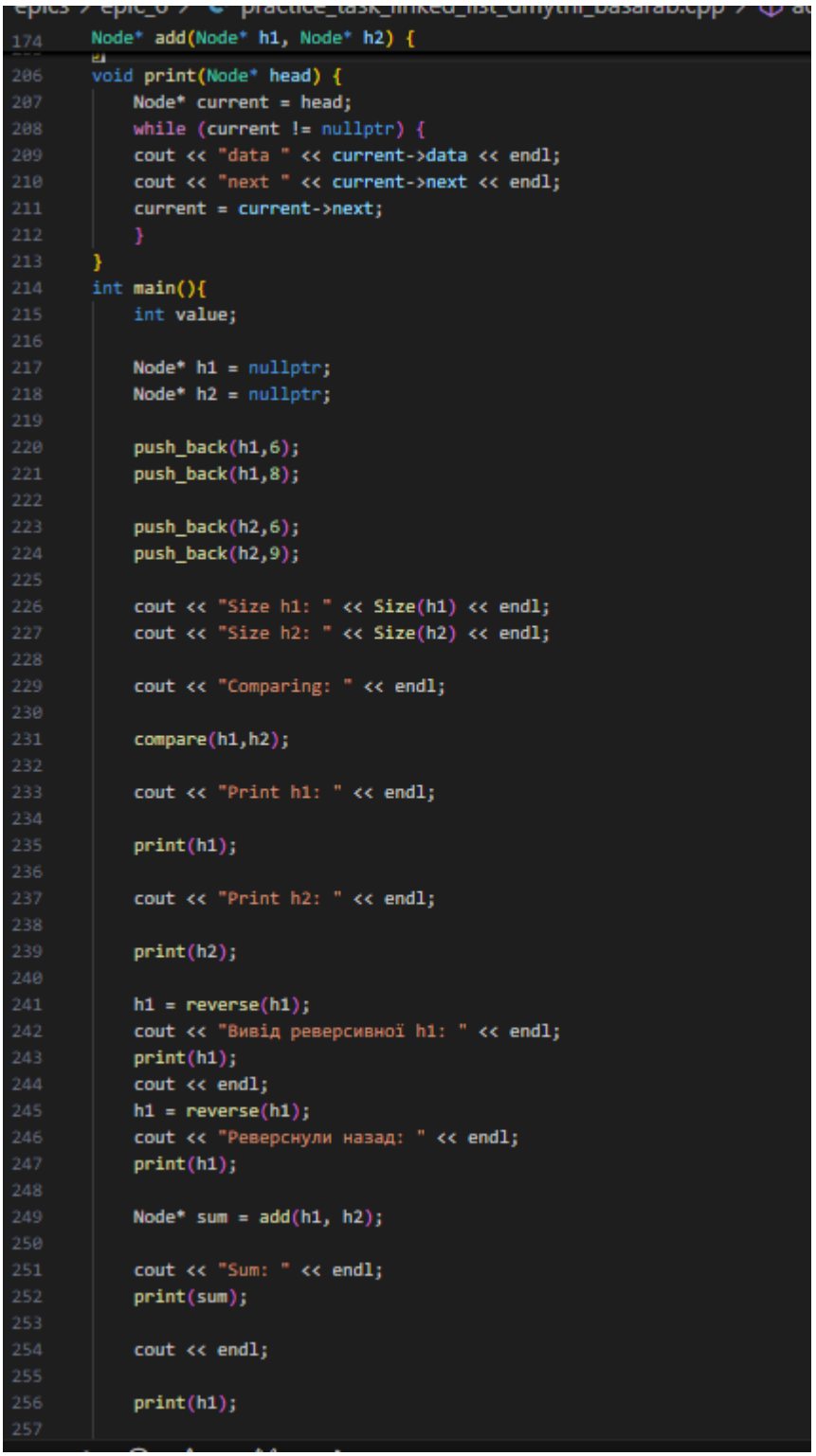
Завдання №4 - Practice task 1-3 Linked List

```cpp
#include <iostream>

using namespace std;

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



void push_back(Node*& head,int value) {

    if (head == nullptr) {
        head = new Node{value, nullptr};

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

        }
        current->next = new Node {value, nullptr};

    }
}

void push_front(Node*& head,int value) {

    if (head == nullptr) {
        head = new Node{value, nullptr};
    } else {
        Node* newNode = new Node {value, head};
        head = newNode;
    }
}

void pop_back(Node*& head) {

    if (head == nullptr) {
        cout << "Empty!" << endl;
        return;

    } else if (head->next == nullptr) {
        delete head;
        head = nullptr;
    } else {
        Node* previous = head;
        Node* current = head->next;
        while (current->next != nullptr) {
            previous = current;
            current = current->next;
        }
```

```
            delete current;
            previous->next = nullptr;
        }
    }

    void pop_front(Node*& head) {

        if (head == nullptr) {
            cout << "Empty!" << endl;
            return;

        } else if (head->next == nullptr) {
            delete head;
            head = nullptr;
        } else {
            Node* newHead = head->next;
            delete head;
            head = newHead;
        }
    }


    void remove(Node*& head,int value) {

            Node* previous = nullptr;
            Node* current = head;//0 елементів нічо не спрацює

            while (current != nullptr) {
                if (current->data == value) {
                    if (previous == nullptr) {
                        head = head->next;//перевіряєм чи ми на голові
                        //тому пшлях = h next
                    } else {
                        previous->next = current->next;
                    }
                delete current;
                return;
                }
            previous = current;
            current = current->next;
            }

        }

    bool find(Node*& head,int value) {
        if (head == nullptr) {
            return false;
        }

        Node* current = head;
```

```
105         while(current != nullptr) {
106             if(current->data == value) {
107                 cout << "Found!" << endl;
108                 return true;
109             }
110             current = current->next;
111         }
112         return false;
113     }
114 
115     int Size(Node* head) {
116         int size = 0;
117         Node* current = head;
118 
119         while(current != nullptr) {
120             size++;
121             current = current->next;
122         }
123 
124 
125         return size;
126 
127     }
128 
129     bool compare(Node* h1,Node* h2) {
130 
131         Node* current1 = h1;
132         Node* current2 = h2;
133 
134         if(Size(h1) != Size(h2)) {
135             cout << "Not the same(" << endl;
136             return false;
137         } else {
138             if (h1 == nullptr && h2 == nullptr) {//нема елементів
139                     cout << "Not the same(";
140                     return false;
141         }
142 
143         while(current1 != nullptr && current2 != nullptr) {
144             if(current1->data != current2->data) {
145                 cout <<"Not the same(" << endl;
146                 return false;
147             }
148             current1 = current1->next;
149             current2 = current2->next;
150         }
151         cout << "Yeeeess" << endl;
152         return true;
153         }
```

```
156
157
158     Node* reverse(Node* head) {
159         Node* prev = nullptr;   // Вказівник на попередній вузол
160         Node* current = head;   // Вказівник на поточний вузол
161         Node* next = nullptr;   // Вказівник на наступний вузол
162
163         while (current != nullptr) {
164             next = current->next; // Зберігаємо наступний вузол
165             current->next = prev; // Міняємо напрямок зв'язку
166             prev = current;       // Зсуваємо prev на поточний вузол
167             current = next;       // Переходимо до наступного вузла
168         }
169
170         return prev;
171     }
172
173
174     Node* add(Node* h1, Node* h2) {
175         Node* result_head = nullptr;  // Результуючий список
176         Node* tail = nullptr;    // Вказівник на останній вузол результатуючого списку
177         int carry = 0;           // Перенос у старший розряд
178
179         while (h1 != nullptr || h2 != nullptr || carry > 0) {
180             int sum = carry;  // Почати з переносу
181
182             if (h1 != nullptr) {
183                 sum += h1->data;// carry + h1
184                 h1 = h1->next;
185             }
186
187             if (h2 != nullptr) {
188                 sum += h2->data; // carry + h1 + h2
189                 h2 = h2->next;
190             }
191
192             carry = sum / 10;  // Оновити перенос і залишити на наступну ітерацію
193             Node* newNode = new Node{sum % 10, nullptr};  // Створюємо новий вузол
194
195             if (result_head == nullptr) {  // Якщо список порожній, новий вузол стає головою
196                 result_head = newNode;
197                 tail = newNode;
198             } else {  // Інакше додаємо новий вузол до кінця
199                 tail->next = newNode;
200                 tail = newNode;  // Оновлюємо хвіст
201             }
202         }
203
204         return result_head;
205     }
206
```



```
174  Node* add(Node* h1, Node* h2) {
206  void print(Node* head) {
207      Node* current = head;
208      while (current != nullptr) {
209      cout << "data " << current->data << endl;
210      cout << "next " << current->next << endl;
211      current = current->next;
212      }
213  }
214  int main(){
215      int value;
216
217      Node* h1 = nullptr;
218      Node* h2 = nullptr;
219
220      push_back(h1,6);
221      push_back(h1,8);
222
223      push_back(h2,6);
224      push_back(h2,9);
225
226      cout << "Size h1: " << Size(h1) << endl;
227      cout << "Size h2: " << Size(h2) << endl;
228
229      cout << "Comparing: " << endl;
230
231      compare(h1,h2);
232
233      cout << "Print h1: " << endl;
234
235      print(h1);
236
237      cout << "Print h2: " << endl;
238
239      print(h2);
240
241      h1 = reverse(h1);
242      cout << "Вивід реверсивної h1: " << endl;
243      print(h1);
244      cout << endl;
245      h1 = reverse(h1);
246      cout << "Реверснули назад: " << endl;
247      print(h1);
248
249      Node* sum = add(h1, h2);
250
251      cout << "Sum: " << endl;
252      print(sum);
253
254      cout << endl;
255
256      print(h1);
257
```

Завдання №5 - Practice task 4-5 Binary Tree

```cpp
#include <iostream>

using namespace std;

// Структура дерева
struct TreeNode {
    int data;
    TreeNode* left;
    TreeNode* right;

};

// Функція створення дзеркального віддзеркалення дерева
TreeNode* create_mirror_flip(TreeNode* root) {
    // Якщо дерево порожнє, повертаємо null
    if (root == nullptr) {
        return nullptr;
    }

    // Створюємо новий вузол для поточного кореня
    TreeNode* newRoot = new TreeNode{root->data};

    //  віддзеркалення для лівої і правої гілок
    //для нового дерева іде вліво а операції виконує для нашого дерева справа
    newRoot->left = create_mirror_flip(root->right);
    newRoot->right = create_mirror_flip(root->left);

    return newRoot;
}

// Функція для виведення дерева (це обхід в глибину)
void print_inorder(TreeNode* root) {
    if (root == nullptr) return;

    print_inorder(root->left);
    cout << root->data << " ";
    print_inorder(root->right);
}

// Функція для видалення дерева (для очищення пам'яті)
void delete_tree(TreeNode* root) {
    if (root == nullptr) return;

    delete_tree(root->left);
    delete_tree(root->right);
    delete root;
}


//-  реалізувати функцію, яка ітеративно проходить по бінарному дереві і записує
// у батьківський вузол суму значень підвузлів
int tree_sum(TreeNode* root) {
    // Якщо вузол порожній, повертаємо 0
```

```
    // Якщо вузол порожній, повертаємо 0
    if (root == nullptr) {
        return 0;
    }

    // Якщо це тільки корінь, його значення не змінюється, повертаємо його
    if (root->left == nullptr && root->right == nullptr) {
        return root->data;
    }

    // Рекурсивно обчислюємо суми для лівого і правого підвузлів
    int leftSum = tree_sum(root->left);
    int rightSum = tree_sum(root->right);

    // Оновлюємо значення поточного вузла сумою підвузлів
    root->data = leftSum + rightSum;

    // Повертаємо нове значення вузла
    return root->data;
}

int main() {
    //дерево:
    //        4
    //       / \
    //      2   5
    //     / \
    //    1   3
    TreeNode* root = new TreeNode{4};
    root->left = new TreeNode{2};
    root->right = new TreeNode{5};
    root->left->left = new TreeNode{1};
    root->left->right = new TreeNode{3};

    // Виведемо дерево до оновлення
    cout << "Наше дерево: ";
    print_inorder(root);
    cout << endl;

    root = create_mirror_flip(root);
    cout << "Наше дерево після дзеркала: ";
    print_inorder(root);
    cout << endl;

    root = create_mirror_flip(root);
    // Оновлюємо дерево, записуючи суми підвузлів у кожний батьківський вузол
    tree_sum(root);

    // Виведемо дерево після оновлення
    cout << "Дерево коли ми сумуємо його два дочірні елементи: ";
    // ідемо в кінець потім вверх оновляєм елементи з кінця і обходим так само знизу вверх
```

```
    // Виведемо дерево після оновлення
    cout << "Дерево коли ми сумуємо його два дочірні елементи: ";
    // ідемо в кінець потім вверх оновляєм елементи з кінця і обходим так само знизу вверх ";
    print_inorder(root);
    cout << endl;

    // Очищаємо пам'ять
    delete_tree(root);

    return 0;
}
```

5) *Результати виконання завдань, тестування та фактично затрачений час:*

Завдання №1 - VNS Lab 10 var 20

```
PS C:\projects>  & 'c:\Users\Дмитр?й\.vscode\extensions\ms-vscode
ngine-In-sscnu21r.b0u' '--stdout=Microsoft-MIEngine-Out-rcjam5ia.
rc' '--dbgExe=C:\msys64\mingw64\bin\gdb.exe' '--interpreter=mi'
Список після додавання елементів:
Alpha Beta Gamma
Елемент із ключем 'Beta' видалено.
Список після видалення елемента 'Beta':
Alpha Gamma
Список після додавання елементів на початок і кінець:
Start2 Start1 Alpha Gamma End1 End2
Список записано у файл 'list.txt'.
Список знищено.
Список порожній.
Список відновлено з файлу 'list.txt'.
Список після відновлення з файлу:
Start2 Start1 Alpha Gamma End1 End2
Список знищено.
PS C:\projects>
```

Фактично затрачений час 6 год

Завдання №2 - Algotester Lab78v1

```
code.cpptools-1.22.11-win32-x64\deb
wsDebugLauncher.exe' '--stdin=Micro
ebvxo.j03' '--stdout=Microsoft-MIEn
z' '--stderr=Microsoft-MIEngine-Err
pid=Microsoft-MIEngine-Pid-obi0u2jc
sys64\mingw64\bin\gdb.exe' '--inter
9
insert
0
5
1 2 3 4 5
insert
2
3
7 7 7
print
1 2 7 7 7 3 4 5
erase
1 2
print
1 7 7 3 4 5
size
6
get
3
3
set
3 13
print
1 7 7 13 4 5
```

(gdb) Launch (projects) Ln 172, Col 14

Created	Compiler	Result	Time (sec.)	Memory (MiB)	Actions
13 hours ago	C++ 23	Accepted	0.008	1.320	View
a day ago	C++ 23	Accepted	0.008	1.348	View
a day ago	C++ 23	Accepted	0.008	1.414	View

Фактично затрачений час 5 год

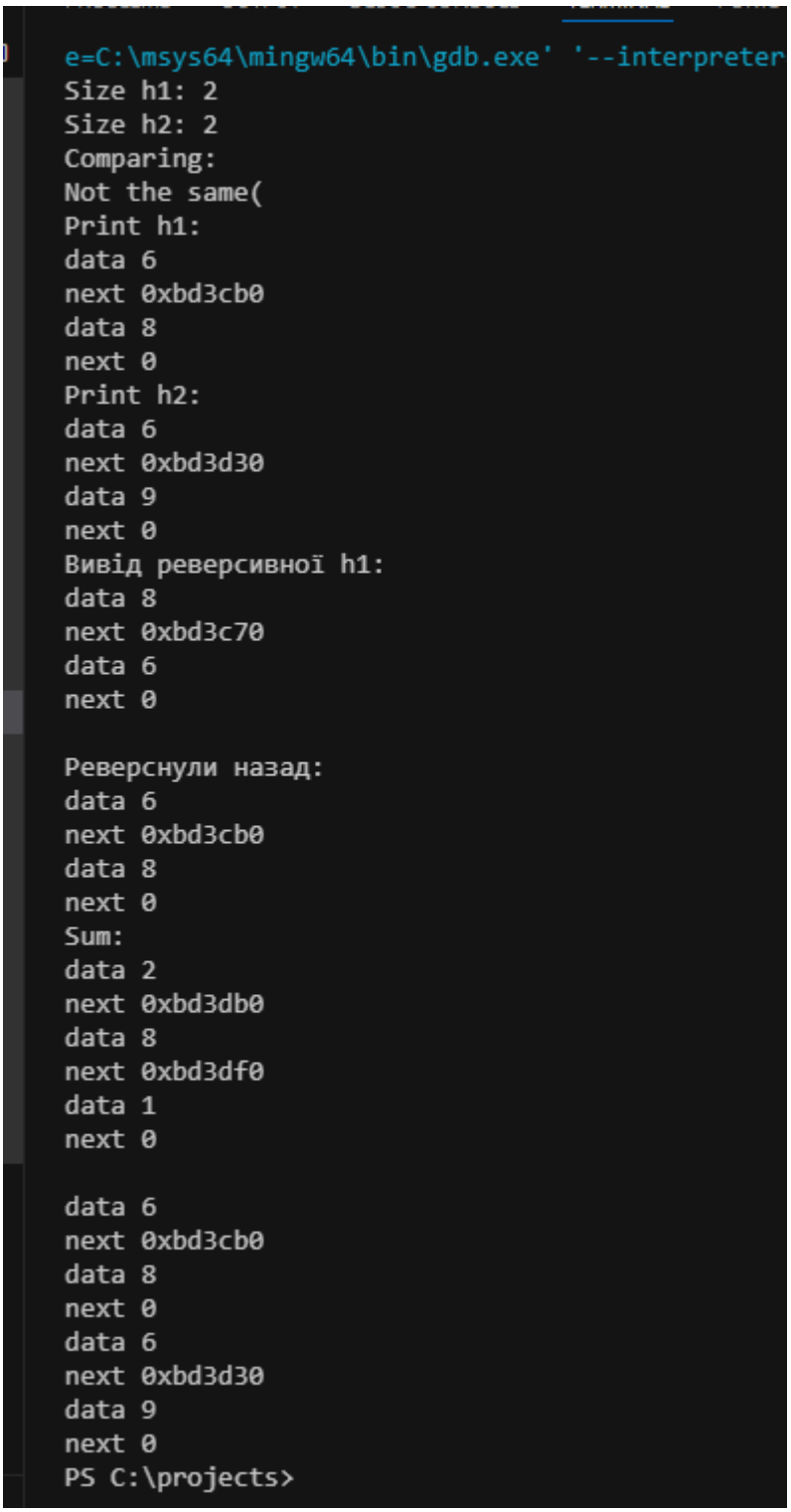
Завдання №3 - Algotester Lab5v3

```
PS C:\projects>  & 'c:\Users\Дмитр?й\.vsc
\ms-vscode.cpptools-1.22.11-win32-x64\debu
\WindowsDebugLauncher.exe' '--stdin=Micros
In-scrruxpr.b21' '--stdout=Microsoft-MIEng
nb4.ske' '--stderr=Microsoft-MIEngine-Erro
g' '--pid=Microsoft-MIEngine-Pid-5of10nrr
e=C:\msys64\mingw64\bin\gdb.exe' '--interp
4 4 2 2
2 3 2 1
3 4 3 2
2 3 2 1
1 2 1 0
PS C:\projects>
```

Created	Compiler	Result	Time (sec.)	Memory (MiB)	Actions
a minute ago	C++ 23	Accepted	0.094	7.523	View
a day ago	C++ 23	Accepted	0.092	7.805	View
2 days ago	C++ 23	Wrong Answer 1	0.003	0.918	View

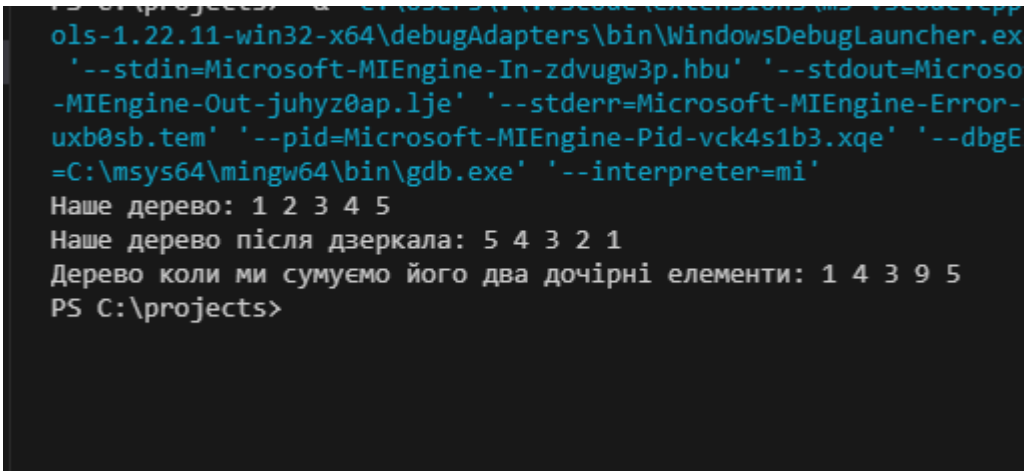
Фактично затрачений час 4 год

Завдання №4 - Practice task 1-3 Linked List



Фактично затрачений час 7 год

Завдання №5 - Practice task 4-5 Binary Tree



Фактично затрачений час 8 год

Висновки: *На цій лабораторній роботі я навчився працювати з Динамічними структурами (Черга, Стек, Списки, Дерево), вивчив Алгоритми обробки динамічних структур*