Міністерство освіти і науки України

Національний університет «Львівська політехніка»

Кафедра систем штучного інтелекту



**Звіт**

про виконання

**Лабораторних та практичних робіт № 10, 5, 7, 8**

***з дисципліни:*** «Мови та парадигми програмування»

***з розділу***: «Динамічні структури (Черга, Стек, Списки, Дерево). Алгоритми обробки динамічних структур.»

***Виконав:***

студент групи ШІ-11

Рєвков Володимир

# **Тема роботи:**

Ознайомлення з темою «Динамічні структури (Черга, Стек, Списки, Дерево). Алгоритми обробки динамічних структур» та виконання Лабораторної роботи ВНС №10, Лабораторних робіт з Алготестеру №7, 8, Практичної та Самостійної Практичної роботи.

# **Мета роботи:**

Ознайомитись з темою «Динамічні структури (Черга, Стек, Списки, Дерево). Алгоритми обробки динамічних структур» та виконати Лабораторну роботу ВНС №10, Лабораторні роботи з Алготестеру №7, 8, Практичну та Самостійну Практичну роботи.

# **Теоретичні відомості:**

1. Теоретичні відомості з переліком важливих тем:

* Тема №1: Динамічні структури.
* Тема №2: Алгоритми обробки динамічних структур.

1. Індивідуальний план опрацювання теорії:

* Тема №1: Динамічні структури.
  + Джерела Інформації
    - Відео.

<https://youtu.be/o5wJkJJpKtM?feature=shared>

<https://www.youtube.com/playlist?list=PLTxllHdfUq4d-DE16EDkpeb8Z68DU7Z_Q>

<https://youtu.be/GBST5uQ_yos?feature=shared>

<https://youtu.be/jaK4pn1jXTo?feature=shared>

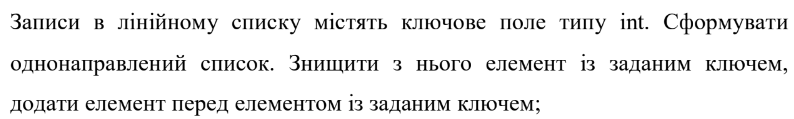
* + Що опрацьовано:
    - Опрацьовано інформацію про динамічні структури.
  + Статус: Ознайомлений частково
  + Початок опрацювання теми: 20.12.2023
  + Звершення опрацювання теми: 20.12.2023

# **Виконання роботи:**

## **1. Опрацювання завдання та вимог до програм та середовища:**

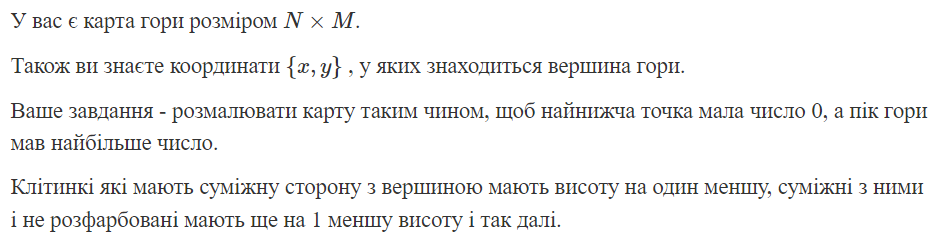
Завдання №1 Лабораторна робота ВНС №10

* Варіант завдання **- 2**
* Деталі завдання



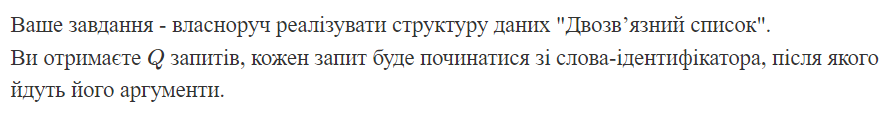
Завдання №2 Лабораторна робота Алготестер №5

* Варіант завдання **- 3**
* Деталі завдання



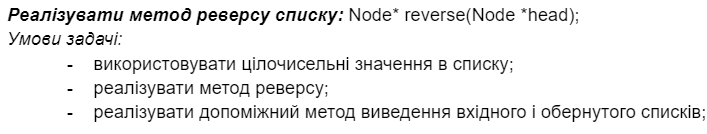
Завдання №3 Лабораторна робота Алготестер №78

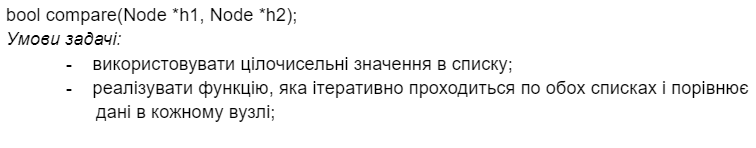
* Варіант завдання **- 1**
* Деталі завдання

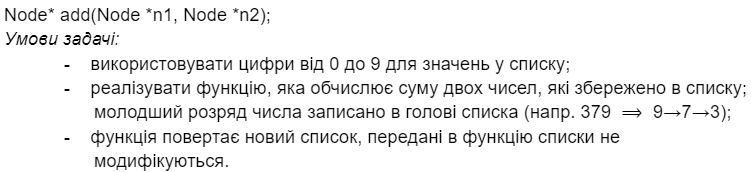


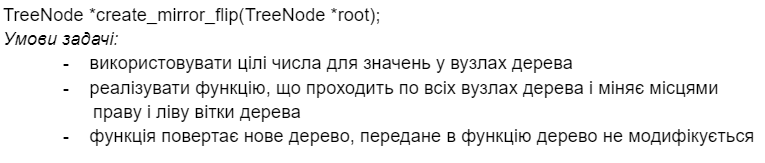
Завдання №4 Практична робота №1

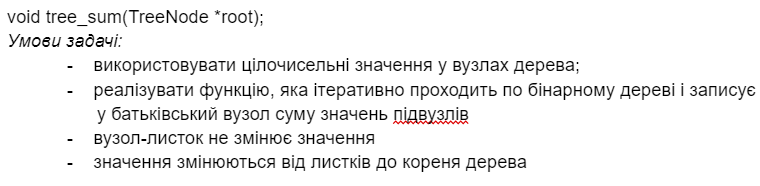
* Деталі завдання











Завдання №5 Самостійна практична робота №1

* Деталі завдання: Керування зв’язним списком(сортування, вивід, заповнення, запис в файл, очищення, відновлення з файлу).

## **2. Дизайн та планована оцінка часу виконання завдань:**

Завдання №2 Лабораторна робота Алготестер №5

* Блок-схема

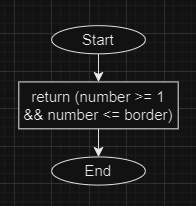


Рисунок 1: check\_number()

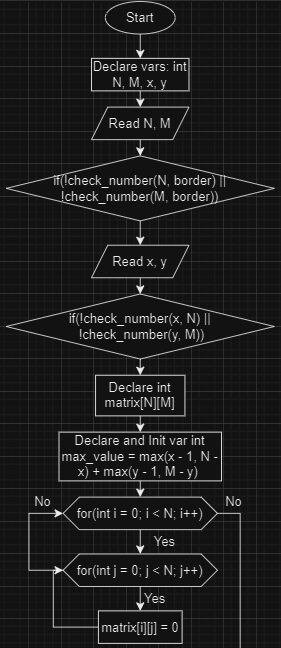
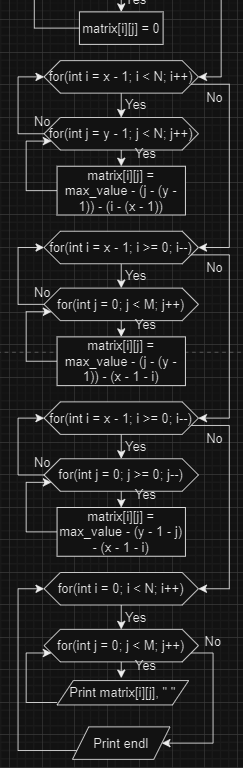
 

Рисунок 2: main(), part 1 Рисунок 3: main(), part 2

* Планований час на реалізацію – **1 година**

## **3. Код програм з посиланням на зовнішні ресурси:**

Завдання №1 Лабораторна робота ВНС №10

#include <iostream>

using namespace std;

const char\* data\_file\_name = "data\_file.txt";

const char\* key\_file\_name = "key\_file.txt";

int node\_count = 0;

struct Node

{

    int data;

    int key;

    Node\* next;

};

Node\* create\_empty\_linked\_list()

{

    return nullptr;

}

void add\_to\_file(int data, int key)

{

    FILE\* d\_file = fopen(data\_file\_name, "ab");

    FILE\* k\_file = fopen(key\_file\_name, "ab");

    if(d\_file != nullptr && k\_file != nullptr)

    {

        if(fwrite(&data, sizeof(int), 1, d\_file) != 1 || fwrite(&key, sizeof(int), 1, k\_file) != 1)

        {

            cout << "Error: Files writing is failed" << endl << endl;

        }

        if(fclose(d\_file) != 0 || fclose(k\_file) != 0)

        {

            cout << "Error: Files closing is failed" << endl << endl;

        }

    }

    else

    {

        cout << "Error: Files opening is failed" << endl << endl;

    }

}

void remove\_from\_file(int data, int key)

{

    const char\* temp\_d\_file\_name = "temp\_data\_file.txt";

    const char\* temp\_k\_file\_name = "temp\_key\_file.txt";

    FILE\* d\_file = fopen(data\_file\_name, "rb");

    FILE\* k\_file = fopen(key\_file\_name, "rb");

    FILE\* temp\_d\_file = fopen(temp\_d\_file\_name, "wb");

    FILE\* temp\_k\_file = fopen(temp\_k\_file\_name, "wb");

    int f\_data;

    int f\_key;

    if(d\_file != nullptr && k\_file != nullptr && temp\_d\_file != nullptr && temp\_k\_file != nullptr)

    {

        while(fread(&f\_data, sizeof(int), 1, d\_file) == 1 && fread(&f\_key, sizeof(int), 1, k\_file) == 1)

        {

            if(f\_data != data && f\_key != key)

            {

                if(!fwrite(&f\_data, sizeof(int), 1, temp\_d\_file) || !fwrite(&f\_key, sizeof(int), 1, temp\_k\_file))

                {

                    cout << "Error: File writing is failed" << endl << endl;

                }

            }

        }

        if(fclose(d\_file) != 0 || fclose(k\_file) != 0 || fclose(temp\_d\_file) != 0 || fclose(temp\_k\_file) != 0)

        {

            cout << "Error: File closing is failed" << endl << endl;

        }

    }

    else

    {

        cout << "Error: Files opening is failed" << endl << endl;

    }

    d\_file = fopen(data\_file\_name, "wb");

    k\_file = fopen(key\_file\_name, "wb");

    temp\_d\_file = fopen(temp\_d\_file\_name, "rb");

    temp\_k\_file = fopen(temp\_k\_file\_name, "rb");

    if(d\_file != nullptr && k\_file != nullptr && temp\_d\_file != nullptr && temp\_k\_file != nullptr)

    {

        while(fread(&f\_data, sizeof(int), 1, temp\_d\_file) == 1 && fread(&f\_key, sizeof(int), 1, temp\_k\_file) == 1)

        {

            if(!fwrite(&f\_data, sizeof(int), 1, d\_file) || !fwrite(&f\_key, sizeof(int), 1, k\_file))

            {

                cout << "Error: File writing is failed" << endl << endl;

            }

        }

        if(fclose(d\_file) != 0 || fclose(k\_file) != 0 || fclose(temp\_d\_file) != 0 || fclose(temp\_k\_file) != 0)

        {

            cout << "Error: File closing is failed" << endl << endl;

        }

    }

    else

    {

        cout << "Error: Files opening is failed" << endl << endl;

    }

    remove(temp\_d\_file\_name);

    remove(temp\_k\_file\_name);

}

void print\_list(const Node\* head)

{

    const Node\* current = head;

    if(current == nullptr)

    {

        cout << "List is empty";

    }

    else

    {

        cout << "List: ";

        while (current != nullptr)

        {

            cout << current->data << "{" << current->key << "} ";

            current = current->next;

        }

    }

    cout << endl << endl;

}

int get\_data()

{

    int data;

    cout << "Enter data: ";

    cin >> data;

    cout << endl;

    return data;

}

int get\_key()

{

    int key;

    cout << "Enter key: ";

    cin >> key;

    cout << endl;

    return key;

}

void add(Node\*& head, int data, int key)

{

    Node\* new\_node = new Node;

    new\_node->data = data;

    new\_node->key = key;

    new\_node->next = nullptr;

    if (head == nullptr)

    {

        head = new\_node;

    }

    else

    {

        Node\* last = head;

        while (last->next != nullptr)

        {

            last = last->next;

        }

        last->next = new\_node;

    }

    cout << "Element with data '" << data << "' and key '" << key << "' was successfully added" << endl << endl;

}

void add\_before\_key(Node\*& head, int key, int new\_data, int new\_key)

{

    Node\* current = head;

    Node\* prev = nullptr;

    while (current != nullptr && current->key != key)

    {

        prev = current;

        current = current->next;

    }

    if (current != nullptr)

    {

        Node\* new\_node = new Node;

        new\_node->data = new\_data;

        new\_node->key = new\_key;

        new\_node->next = current;

        if (prev == nullptr)

        {

            head = new\_node;

        }

        else

        {

            prev->next = new\_node;

        }

        add\_to\_file(new\_data, new\_key);

        cout << "Element with data '" << new\_data << "' and key '" << new\_key << "' was successfully added" << endl << endl;

    }

    else

    {

        cout << "Error: Element with key '" << key << "' wasn't found" << endl << endl;

    }

}

void remove\_by\_key(Node\*& head, int key)

{

    Node\* current = head;

    Node\* prev = nullptr;

    while (current != nullptr && current->key != key)

    {

        prev = current;

        current = current->next;

    }

    if (current != nullptr)

    {

        if (prev == nullptr)

        {

            head = current->next;

        }

        else

        {

            prev->next = current->next;

        }

        remove\_from\_file(current->data, current->key);

        cout << "Element '" << current->data << " with key '" << current->key << "' was successfully removed" << endl << endl;

        delete current;

    }

    else

    {

        cout << "Error: Element with key '" << key << "' wasn't found" << endl << endl;

    }

}

void clear\_list(Node\*& head)

{

    while (head != nullptr)

    {

        Node\* temp = head;

        head = head->next;

        delete temp;

    }

    cout << "List was successfully cleared" << endl << endl;

}

Node\* reverse\_list(Node\* head)

{

    Node\* current = head;

    Node\* prev = nullptr;

    Node\* next = nullptr;

    while (current != nullptr)

    {

        next = current->next;

        current->next = prev;

        prev = current;

        current = next;

    }

    head = prev;

    return head;

}

void restore\_list(Node\*& head)

{

    if(head != nullptr)

    {

        clear\_list(head);

    }

    FILE\* d\_file = fopen(data\_file\_name, "rb");

    FILE\* k\_file = fopen(key\_file\_name, "rb");

    if(d\_file != nullptr && k\_file != nullptr)

    {

        int f\_data;

        int f\_key;

        while(fread(&f\_data, sizeof(int), 1, d\_file) == 1 && fread(&f\_key, sizeof(int), 1, k\_file) == 1)

        {

            add(head, f\_data, f\_key);

        }

        cout << "List has been restored" << endl << endl;

        head = reverse\_list(head);

        if(fclose(d\_file) != 0 || fclose(k\_file) != 0)

        {

            cout << "Error: File closing is failed" << endl << endl;

        }

    }

    else

    {

        cout << "Error: File opening is failed" << endl << endl;

    }

}

int main()

{

    int data;

    int key;

    Node\* head = create\_empty\_linked\_list();

    cout << "Enter element" << endl << endl;

    data = get\_data();

    key = get\_key();

    add(head, data, key);

    add\_to\_file(data, key);

    cout << "Enter element" << endl << endl;

    data = get\_data();

    key = get\_key();

    add(head, data, key);

    add\_to\_file(data, key);

    print\_list(head);

    cout << "Enter key to remove" << endl << endl;

    key = get\_key();

    remove\_by\_key(head, key);

    print\_list(head);

    cout << "Enter key to add element before" << endl << endl;

    key = get\_key();

    cout << "New" << endl << endl;

    int new\_data = get\_data();

    cout << "New" << endl << endl;

    int new\_key = get\_key();

    add\_before\_key(head, key, new\_data, new\_key);

    print\_list(head);

    clear\_list(head);

    print\_list(head);

    restore\_list(head);

    print\_list(head);

    clear\_list(head);

    remove(data\_file\_name);

    remove(key\_file\_name);

}

<https://github.com/volodymyr-rievkov/VolodymyrR_Reprository/blob/main/Epic%206/Lab10T2/vns_lab_10_task_1_variant_2_volodymyr_rievkov.cpp>

Завдання №2 Лабораторна робота Алготестер №5

#include <iostream>

#include <algorithm>

using namespace std;

const int border = 1000;

bool check\_number(int number, int border)

{

    return(number >= 1 && number <= border);

}

int main(){

    int N, M, x, y;

    cin >> N >> M;

    if(!check\_number(N, border) || !check\_number(M, border))

    {

        return 0;

    }

    cin >> x >> y;

    if(!check\_number(x, N) || !check\_number(y, M))

    {

        return 0;

    }

    int matrix[N][M];

    int max\_value = max(x - 1, N - x) + max(y - 1, M - y);

    for (int i = 0; i < N; i++)

    {

        for (int j = 0; j < M; j++)

        {

            matrix[i][j] = 0;

        }

    }

    for (int i = x - 1; i < N; i++)

    {

        for (int j = y - 1; j < M; j++)

        {

            matrix[i][j] = max\_value - (j - (y - 1)) - (i - (x - 1));

        }

    }

    for (int i = x - 1; i >= 0; i--)

    {

        for (int j = y - 1; j < M; j++)

        {

            matrix[i][j] = max\_value - (j - (y - 1)) - (x - 1 - i);

        }

    }

    for (int i = x - 1; i >= 0; i--)

    {

        for (int j = y - 1; j >= 0; j--)

        {

            matrix[i][j] = max\_value - (y - 1 - j) - (x - 1 - i);

        }

    }

    for (int i =  x - 1; i < N; i++)

    {

        for (int j = y - 1; j >= 0; j--)

        {

            matrix[i][j] = max\_value - (y - 1 - j) - (i - (x - 1));

        }

    }

    for (int i = 0; i < N; i++)

    {

        for (int j = 0; j < M; j++)

        {

            cout << matrix[i][j] << " ";

        }

        cout << endl;

    }

}

<https://github.com/volodymyr-rievkov/VolodymyrR_Reprository/blob/main/Epic%206/AlgtL5T3/algt_lab_5_variant_3_volodymyr_rievkov.cpp>

Завдання №3 Лабораторна робота Алготестер №78

-

Завдання №4 Практична робота №1

#include <iostream>

using namespace std;

const char\* data\_file\_name = "data\_file.txt";

const char\* key\_file\_name = "key\_file.txt";

int node\_count = 0;

struct Node

{

    int data;

    int key;

    Node\* next;

};

Node\* create\_empty\_linked\_list()

{

    return nullptr;

}

void add\_to\_file(int data, int key)

{

    FILE\* d\_file = fopen(data\_file\_name, "ab");

    FILE\* k\_file = fopen(key\_file\_name, "ab");

    if(d\_file != nullptr && k\_file != nullptr)

    {

        if(fwrite(&data, sizeof(int), 1, d\_file) != 1 || fwrite(&key, sizeof(int), 1, k\_file) != 1)

        {

            cout << "Error: Files writing is failed" << endl << endl;

        }

        if(fclose(d\_file) != 0 || fclose(k\_file) != 0)

        {

            cout << "Error: Files closing is failed" << endl << endl;

        }

    }

    else

    {

        cout << "Error: Files opening is failed" << endl << endl;

    }

}

void remove\_from\_file(int data, int key)

{

    const char\* temp\_d\_file\_name = "temp\_data\_file.txt";

    const char\* temp\_k\_file\_name = "temp\_key\_file.txt";

    FILE\* d\_file = fopen(data\_file\_name, "rb");

    FILE\* k\_file = fopen(key\_file\_name, "rb");

    FILE\* temp\_d\_file = fopen(temp\_d\_file\_name, "wb");

    FILE\* temp\_k\_file = fopen(temp\_k\_file\_name, "wb");

    int f\_data;

    int f\_key;

    if(d\_file != nullptr && k\_file != nullptr && temp\_d\_file != nullptr && temp\_k\_file != nullptr)

    {

        while(fread(&f\_data, sizeof(int), 1, d\_file) == 1 && fread(&f\_key, sizeof(int), 1, k\_file) == 1)

        {

            if(f\_data != data && f\_key != key)

            {

                if(!fwrite(&f\_data, sizeof(int), 1, temp\_d\_file) || !fwrite(&f\_key, sizeof(int), 1, temp\_k\_file))

                {

                    cout << "Error: File writing is failed" << endl << endl;

                }

            }

        }

        if(fclose(d\_file) != 0 || fclose(k\_file) != 0 || fclose(temp\_d\_file) != 0 || fclose(temp\_k\_file) != 0)

        {

            cout << "Error: File closing is failed" << endl << endl;

        }

    }

    else

    {

        cout << "Error: Files opening is failed" << endl << endl;

    }

    d\_file = fopen(data\_file\_name, "wb");

    k\_file = fopen(key\_file\_name, "wb");

    temp\_d\_file = fopen(temp\_d\_file\_name, "rb");

    temp\_k\_file = fopen(temp\_k\_file\_name, "rb");

    if(d\_file != nullptr && k\_file != nullptr && temp\_d\_file != nullptr && temp\_k\_file != nullptr)

    {

        while(fread(&f\_data, sizeof(int), 1, temp\_d\_file) == 1 && fread(&f\_key, sizeof(int), 1, temp\_k\_file) == 1)

        {

            if(!fwrite(&f\_data, sizeof(int), 1, d\_file) || !fwrite(&f\_key, sizeof(int), 1, k\_file))

            {

                cout << "Error: File writing is failed" << endl << endl;

            }

        }

        if(fclose(d\_file) != 0 || fclose(k\_file) != 0 || fclose(temp\_d\_file) != 0 || fclose(temp\_k\_file) != 0)

        {

            cout << "Error: File closing is failed" << endl << endl;

        }

    }

    else

    {

        cout << "Error: Files opening is failed" << endl << endl;

    }

    remove(temp\_d\_file\_name);

    remove(temp\_k\_file\_name);

}

void print\_list(const Node\* head)

{

    const Node\* current = head;

    if(current == nullptr)

    {

        cout << "List is empty";

    }

    else

    {

        cout << "List: ";

        while (current != nullptr)

        {

            cout << current->data << "{" << current->key << "} ";

            current = current->next;

        }

    }

    cout << endl << endl;

}

int get\_data()

{

    int data;

    cout << "Enter data: ";

    cin >> data;

    cout << endl;

    return data;

}

int get\_key()

{

    int key;

    cout << "Enter key: ";

    cin >> key;

    cout << endl;

    return key;

}

void add(Node\*& head, int data, int key)

{

    Node\* new\_node = new Node;

    new\_node->data = data;

    new\_node->key = key;

    new\_node->next = nullptr;

    if (head == nullptr)

    {

        head = new\_node;

    }

    else

    {

        Node\* last = head;

        while (last->next != nullptr)

        {

            last = last->next;

        }

        last->next = new\_node;

    }

    cout << "Element with data '" << data << "' and key '" << key << "' was successfully added" << endl << endl;

}

void add\_before\_key(Node\*& head, int key, int new\_data, int new\_key)

{

    Node\* current = head;

    Node\* prev = nullptr;

    while (current != nullptr && current->key != key)

    {

        prev = current;

        current = current->next;

    }

    if (current != nullptr)

    {

        Node\* new\_node = new Node;

        new\_node->data = new\_data;

        new\_node->key = new\_key;

        new\_node->next = current;

        if (prev == nullptr)

        {

            head = new\_node;

        }

        else

        {

            prev->next = new\_node;

        }

        add\_to\_file(new\_data, new\_key);

        cout << "Element with data '" << new\_data << "' and key '" << new\_key << "' was successfully added" << endl << endl;

    }

    else

    {

        cout << "Error: Element with key '" << key << "' wasn't found" << endl << endl;

    }

}

void remove\_by\_key(Node\*& head, int key)

{

    Node\* current = head;

    Node\* prev = nullptr;

    while (current != nullptr && current->key != key)

    {

        prev = current;

        current = current->next;

    }

    if (current != nullptr)

    {

        if (prev == nullptr)

        {

            head = current->next;

        }

        else

        {

            prev->next = current->next;

        }

        remove\_from\_file(current->data, current->key);

        cout << "Element '" << current->data << " with key '" << current->key << "' was successfully removed" << endl << endl;

        delete current;

    }

    else

    {

        cout << "Error: Element with key '" << key << "' wasn't found" << endl << endl;

    }

}

void clear\_list(Node\*& head)

{

    while (head != nullptr)

    {

        Node\* temp = head;

        head = head->next;

        delete temp;

    }

    cout << "List was successfully cleared" << endl << endl;

}

Node\* reverse\_list(Node\* head)

{

    Node\* current = head;

    Node\* prev = nullptr;

    Node\* next = nullptr;

    while (current != nullptr)

    {

        next = current->next;

        current->next = prev;

        prev = current;

        current = next;

    }

    head = prev;

    return head;

}

void restore\_list(Node\*& head)

{

    if(head != nullptr)

    {

        clear\_list(head);

    }

    FILE\* d\_file = fopen(data\_file\_name, "rb");

    FILE\* k\_file = fopen(key\_file\_name, "rb");

    if(d\_file != nullptr && k\_file != nullptr)

    {

        int f\_data;

        int f\_key;

        while(fread(&f\_data, sizeof(int), 1, d\_file) == 1 && fread(&f\_key, sizeof(int), 1, k\_file) == 1)

        {

            add(head, f\_data, f\_key);

        }

        cout << "List has been restored" << endl << endl;

        head = reverse\_list(head);

        if(fclose(d\_file) != 0 || fclose(k\_file) != 0)

        {

            cout << "Error: File closing is failed" << endl << endl;

        }

    }

    else

    {

        cout << "Error: File opening is failed" << endl << endl;

    }

}

int main()

{

    int data;

    int key;

    Node\* head = create\_empty\_linked\_list();

    cout << "Enter element" << endl << endl;

    data = get\_data();

    key = get\_key();

    add(head, data, key);

    add\_to\_file(data, key);

    cout << "Enter element" << endl << endl;

    data = get\_data();

    key = get\_key();

    add(head, data, key);

    add\_to\_file(data, key);

    print\_list(head);

    cout << "Enter key to remove" << endl << endl;

    key = get\_key();

    remove\_by\_key(head, key);

    print\_list(head);

    cout << "Enter key to add element before" << endl << endl;

    key = get\_key();

    cout << "New" << endl << endl;

    int new\_data = get\_data();

    cout << "New" << endl << endl;

    int new\_key = get\_key();

    add\_before\_key(head, key, new\_data, new\_key);

    print\_list(head);

    clear\_list(head);

    print\_list(head);

    restore\_list(head);

    print\_list(head);

    clear\_list(head);

    remove(data\_file\_name);

    remove(key\_file\_name);

}

<https://github.com/volodymyr-rievkov/VolodymyrR_Reprository/blob/main/Epic%206/PractiseTask/practise_task_1_volodymyr_rievkov.cpp>

Завдання №5 Самостійна практична робота №1

#include<iostream>

using namespace std;

const char\* file\_name = "list\_file.txt";

int elements\_count;

bool app\_exit = false;

struct Node

{

    int data;

    Node\* next;

};

void add(Node\*& head, int number)

{

    Node\* new\_node = new Node;

    new\_node->data = number;

    new\_node->next = nullptr;

    if (head == nullptr)

    {

        head = new\_node;

    }

    else

    {

        Node\* last = head;

        while (last->next != nullptr)

        {

            last = last->next;

        }

        last->next = new\_node;

    }

}

void fill\_list(Node\*& head)

{

    FILE\* file = fopen(file\_name, "wb");

    if(file != nullptr)

    {

        int elements\_number;

        int element;

        int count = 0;

        cout << "Enter number of elements: ";

        cin >> elements\_number;

        for(int i = 0; i < elements\_number; i++)

        {

            cout << ": ";

            cin >> element;

            add(head, element);

            if(fwrite(&element, sizeof(int), 1, file) != 1)

            {

                cout << "Error: File writing is failed" << endl;

            }

            count++;

        }

        if(fclose(file) != 0)

        {

            cout << "Error: File closing is failed" << endl;

        }

        elements\_count = count;

    }

    else

    {

        cout << "Error: File opening is failed" << endl;

    }

    cout << endl;

}

void print\_list(const Node\* head)

{

    const Node\* current = head;

    if(current == nullptr)

    {

        cout << "List is empty" << endl;

    }

    else

    {

        while (current != nullptr)

        {

            cout << current->data << " ";

            current = current->next;

        }

    }

    cout << endl << endl;

}

void clear\_list(Node\*& head)

{

    while (head != nullptr)

    {

        Node\* temp = head;

        head = head->next;

        delete temp;

    }

}

void restore\_list(Node\*& head)

{

    if(head != nullptr)

    {

        clear\_list(head);

    }

    FILE\* file = fopen(file\_name, "rb");

    if(file != nullptr)

    {

        int element;

        for(int i = 0; i < elements\_count; i++)

        {

            if(fread(&element, sizeof(int), 1, file) == 1)

            {

                add(head, element);

            }

            else

            {

                cout << "Error: File reading is failed" << endl << endl;

                break;

            }

        }

        if(fclose(file) != 0)

        {

            cout << "Error: File closing is failed" << endl << endl;

        }

    }

    else

    {

        cout << "Error: File reading is failed" << endl << endl;

    }

}

void swap\_nodes\_values(Node\*& node\_1, Node\*& node\_2) {

    int temp = node\_1->data;

    node\_1->data = node\_2->data;

    node\_2->data = temp;

}

void sort\_list\_by\_asc(Node\*& head)

{

    int swapped;

    Node\* current;

    Node\* last = nullptr;

    if (head == nullptr)

    {

        return;

    }

    do {

        swapped = 0;

        current = head;

        while (current->next != last)

        {

            if (current->data > current->next->data)

            {

                swap\_nodes\_values(current, current->next);

                swapped = 1;

            }

            current = current->next;

        }

        last = current;

    } while (swapped);

}

void sort\_list\_by\_desc(Node\*& head)

{

    int swapped;

    Node\* current;

    Node\* last = nullptr;

    if (head == nullptr)

    {

        return;

    }

    do {

        swapped = 0;

        current = head;

        while (current->next != last)

        {

            if (current->data < current->next->data)

            {

                swap\_nodes\_values(current, current->next);

                swapped = 1;

            }

            current = current->next;

        }

        last = current;

    } while (swapped);

}

int get\_action()

{

    int action;

    do

    {

        cout << " - Print list(1)" << endl

            << " - Sort list asc(2)" << endl

            << " - Sort list desc(3)" << endl

            << " - Clear list(4)" << endl

            << " - Restore list(5)" << endl

            << " - Exit(6)" << endl

            << "Enter action: ";

        cin >> action;

    } while (action < 1 && action > 6);

    return action;

}

void list\_managment()

{

    Node\* head = nullptr;

    fill\_list(head);

    while(!app\_exit)

    {

        switch(get\_action())

        {

            case 1:

                print\_list(head);

                break;

            case 2:

                sort\_list\_by\_asc(head);

                cout << "List is sorted by asc" << endl << endl;

                break;

            case 3:

                sort\_list\_by\_desc(head);

                cout << "List is sorted by desc" << endl << endl;

                break;

            case 4:

                clear\_list(head);

                break;

            case 5:

                restore\_list(head);

                break;

            case 6:

                app\_exit = true;

                break;

        }

    }

}

int main()

{

    list\_managment();

}

<https://github.com/volodymyr-rievkov/VolodymyrR_Reprository/blob/main/Epic%206/SelfPractiseTask/self_practise_task_volodymyr_rievkov.cpp>

## **4. Результати виконання завдань, тестування та фактично затрачений час:**

Завдання №1 Лабораторна робота ВНС №10

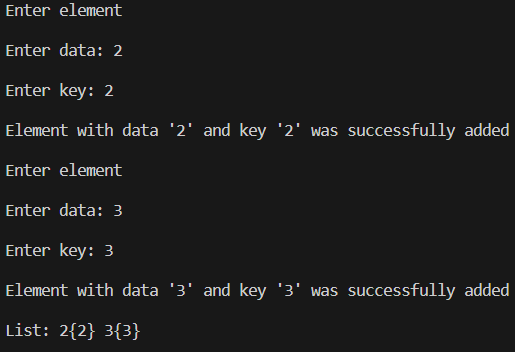
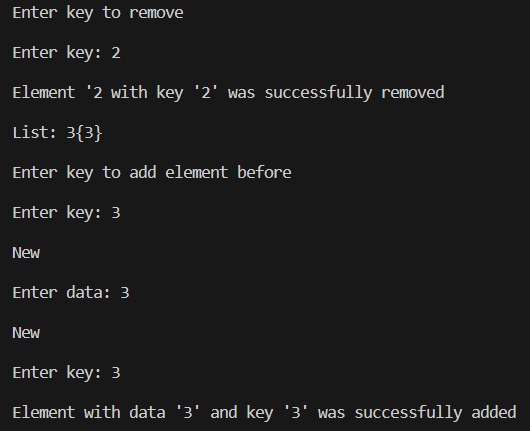
 

Рисунок 4: Результат ЛР№5 Рисунок 5: Результати ЛР№10

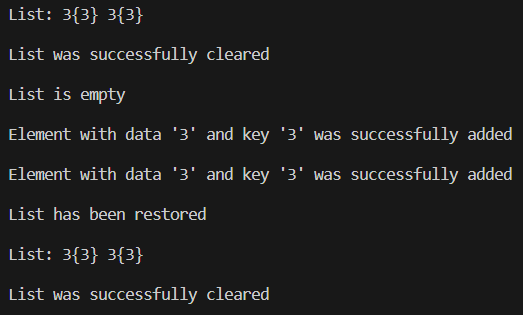


Рисунок 6: Результати ЛР№10

Час затрачений на виконання завдання – **3 години**

Завдання №2 Лабораторна робота Алготестер №5

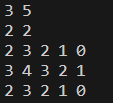


Рисунок 6: Результат Алгт ЛР№5

Час затрачений на виконання завдання – **3 години**

Завдання №3 Лабораторна робота Алготестер №78

-

Час затрачений на виконання завдання - 0

Завдання №4 Практична робота №1

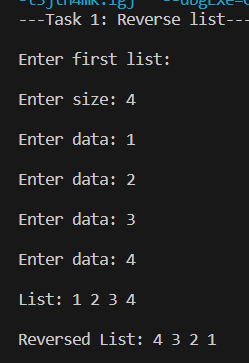
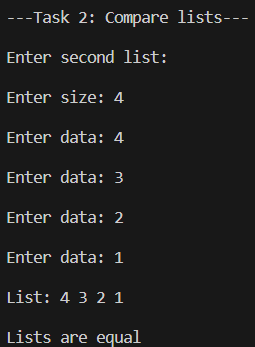
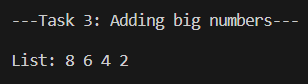
  

Рисунок 7: Результати ПР№1 Рисунок 8: Результати ПР№1 Рисунок 9: Результати ПР№1

Час затрачений на виконання завдання – **3 години**

Завдання №5 Самостійна практична робота №1

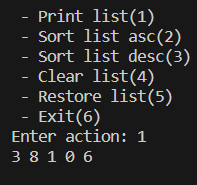
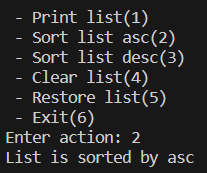
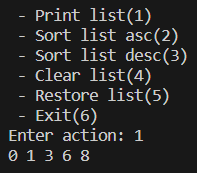
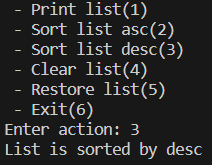
   

Рисунок 10: Self\_P result Рисунок 11: Self\_P result Рисунок 12: Self\_P result Рисунок 13: Self\_P result

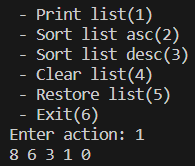
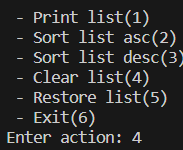
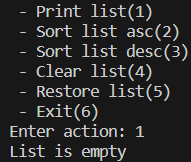
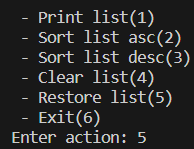
   

Рисунок 14: Self\_P result Рисунок 15: Self\_P result Рисунок 16: Self\_P result Рисунок 17: Self\_P result

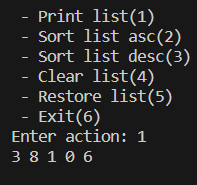


Рисунок 18: Self\_P result

Час затрачений на виконання завдання – **4 години**

# **Висновки:**

Виконуючи завдання з Епіку 6, я дізнався багато нового з теми динамічні структури та методи їх обробки. На жаль не всі завдання зміг зробити, але отримав дуже цінний так корисний досвід.