Міністерство освіти і науки України

Національний університет «Львівська політехніка»

Кафедра систем штучного інтелекту

Звіт

про виконання лабораторних та практичних робіт блоку № 6

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

з дисципліни: «Основи програмування»

до:

ВНС Лабораторної Роботи № 10

Алготестер Лабораторних Робіт № 5, 7-8

Практичних Робіт до блоку № 6

Виконала:

Студентка групи ШІ-12

Олійник Божена

Львів 2024

Тема роботи

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

Мета роботи

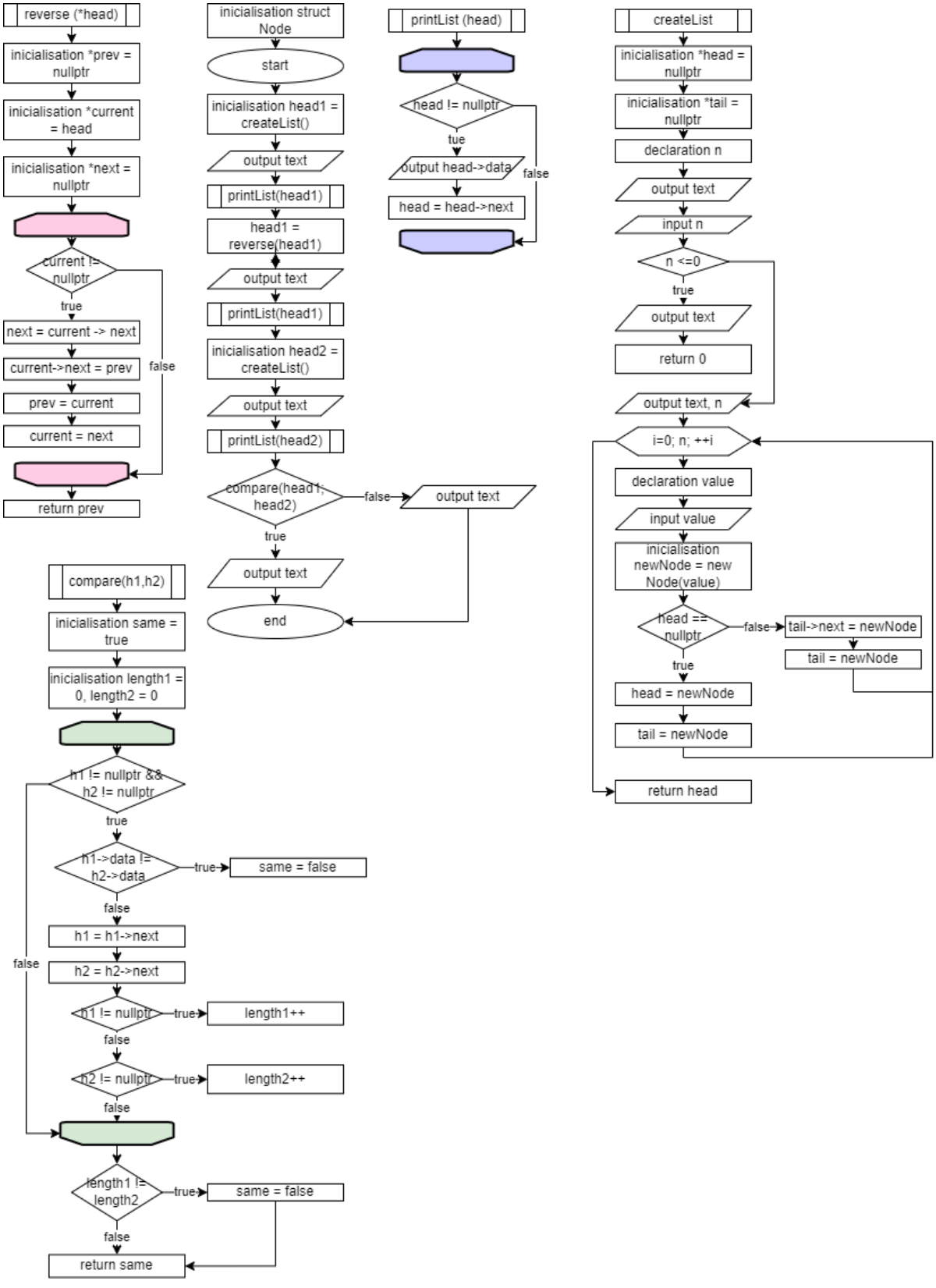
1. Навчитись створювати та використовувати динамічні структури, такі як: Черга, Стек, Списки, Дерево.
2. Навчитись виконувати алгоритми обробки динамічний структур.

Теоретичні відомості

1. Перенавантаження операторі виводу https://acode.com.ua/urok-141-perevantazhennya-operatoriv-vvodu-i-vyvodu/#toc-0
2. Класи https://acode.com.ua/urok-121-klasy-ob-yekty-i-metody/#toc-1 https://acode.com.ua/urok-183-shablony-klasiv/
3. Черга https://www.bestprog.net/uk/2019/09/26/c-queue-general-concepts-ways-to-implement-the-queue-implementing-a-queue-as-a-dynamic-array-ua/
4. Стек https://www.bestprog.net/uk/2019/09/18/c-the-concept-of-stack-operations-on-the-stack-an-example-implementation-of-the-stack-as-a-dynamic-array-ua/
5. Списки https://codelessons.dev/ru/spisok-list-v-s-polnyj-material/
6. Дерево https://www.bing.com/ck/a?!&&p=d92fe6ba3879a4e47f751a99580f5f4fe2193328a2a4e32cef85f6aa2f6603bcJmltdHM9MTczMjc1MjAwMA&ptn=3&ver=2&hsh=4&fclid=1c78f629-a87f-6de6-3f44-e249a96d6cf2&psq=%d0%b4%d0%b5%d1%80%d0%b5%d0%b2%d0%b0+%d1%83+%d1%81%2b%2b&u=a1aHR0cHM6Ly9wdXJlY29kZWNwcC5jb20vdWsvYXJjaGl2ZXMvMjQ4Mw&ntb=1

Виконання роботи

Task 2 - Requirements management (understand tasks) and design activities (draw flow diagrams and estimate tasks 3-7) (1 год 40хв)



Task 3 - Lab# programming: VNS Lab 10 (10хв)

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

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

class SinglyLinkedList
{
private:
    Node *head;

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

    void createList()
    {
        head = nullptr;
    }

    void addElements(int startPosition, int k, int value)
    {
        for (int i = 0; i < k; ++i)
        {
            addElement(startPosition + i, value + i);
        }
    }

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

        if (position <= 0 || !head)
        {
            newNode->next = head;
            head = newNode;
        }
        else
        {
            Node *temp = head;
            int index = 0;
            while (temp->next && index < position - 1)
            {
                temp = temp->next;
                index++;
            }
            newNode->next = temp->next;
            temp->next = newNode;
        }
    }

    void deleteElement(int element)
    {
        int position = element - 1;
        if (!head)
        {
            cout << "Список порожній, видалення неможливе" << endl;
            return;
        }

        if (position <= 0)
        {
            Node *toDelete = head;
            head = head->next;
            delete toDelete;
        }
        else
        {
            Node *temp = head;
            int index = 0;
            while (temp->next && index < position - 1)
            {
                temp = temp->next;
                index++;
            }
            if (!temp->next)
            {
                cout << "Некоректна позиція для видалення!" << endl;
                return;
            }
            Node *toDelete = temp->next;
            temp->next = toDelete->next;
            delete toDelete;
        }
    }

    void printList() const
    {
        if (!head)
        {
            cout << "Список порожній" << endl;
            return;
        }
        Node *temp = head;
        while (temp)
        {
            cout << temp->data << " ";
            temp = temp->next;
        }
        cout << endl;
    }
```

```cpp
void writeToFile(const string &filename) const
{
    ofstream outFile(filename);
    if (outFile.is_open())
    {
        Node *temp = head;
        while (temp)
        {
            outFile << temp->data << " ";
            temp = temp->next;
        }
        outFile.close();
    }
    else
    {
        cerr << "Не вдалося відкрити файл для запису!" << endl;
        return;
    }
}

void restoreFromFile(const string &filename)
{
    deleteList();
    ifstream inFile(filename);
    if (inFile.is_open())
    {
        int value;
        while (inFile >> value)
        {
            addElement(INT_MAX, value);
        }
        inFile.close();
    }
    else
    {
        cerr << "Не вдалося відкрити файл для читання!" << endl;
        return;
    }
}

void deleteList()
{
    while (head)
    {
        Node *toDelete = head;
        head = head->next;
        delete toDelete;
    }
}

~SinglyLinkedList()
{
    deleteList();
}
```

```cpp

int main()
{
    SinglyLinkedList list;

    list.createList();

    int el;
    cout << "Введіть кількість елементів списку: ";
    cin >> el;

    for (int i = 0; i < el; i++)
    {
        int pos, val;
        cout << "Введіть позицію елемента та його значення: ";
        cin >> pos >> val;
        list.addElement(pos, val);
    }
    list.printList();

    int delpos;
    cout << "Введіть позицію елемента який потрібно видалити: ";
    cin >> delpos;

    list.deleteElement(delpos);
    list.printList();

    int addpos, num, addval;
    cout << "Введіть позицію елемента після якого потрібно додати нові, ";
    cout << "скільки елементів додати та значення першого доданого: ";
    cin >> addpos >> num >> addval;

    list.addElements(addpos, num, addval);
    list.printList();

    list.writeToFile("list.txt");

    list.deleteList();
    list.printList();

    list.restoreFromFile("list.txt");
    list.printList();

    list.deleteList();

    remove("list.txt");

    return 0;
}

```

```
Введіть кількість елементів списку: 5
Введіть позицію елемента та його значення: 1 1
Введіть позицію елемента та його значення: 2 2
Введіть позицію елемента та його значення: 3 3
Введіть позицію елемента та його значення: 4 4
Введіть позицію елемента та його значення: 5 5
1 2 3 4 5
Введіть позицію елемента який потрібно видалити: 3
1 2 4 5
Введіть позицію елемента після якого потрібно додати нові, скільки елементів додати та значення першого доданого: 2 4 10
1 2 10 11 12 13 4 5
Список порожній
1 2 10 11 12 13 4 5
```

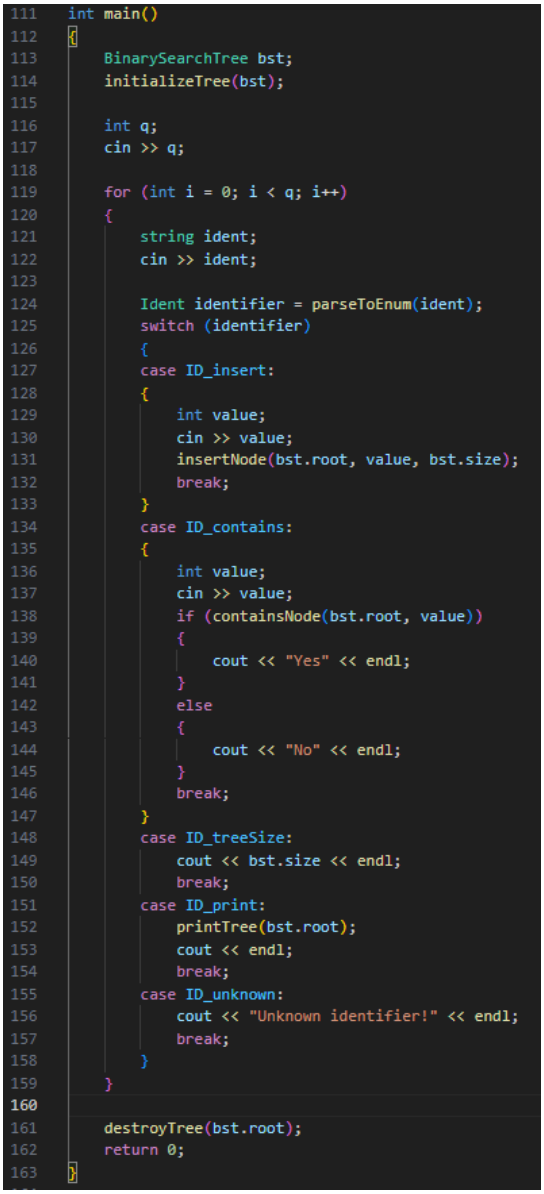
Task 4 - Lab# programming: Algotester Lab 5

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

const int MAX_N = 1000;
const int MAX_M = 1000;

int main() {
    int N, M;
    char cave[MAX_N][MAX_M + 1];

    cin >> N >> M;

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

    for (int col = 0; col < M; ++col) {
        int empty_row = N - 1;

        for (int row = N - 1; row >= 0; --row) {
            if (cave[row][col] == 'X') {
                empty_row = row - 1;
            } else if (cave[row][col] == 'S') {
                cave[row][col] = 'O';
                if (empty_row >= 0) {
                    cave[empty_row][col] = 'S';
                    --empty_row;
                }
            }
        }
    }
    cout << endl;
    for (int i = 0; i < N; ++i) {
        cout << cave[i] << endl;
    }

    return 0;
}
```

```
5 5
SSOSS
OOOOO
SOOXX
OOOOS
OOSOO

OOOOO
OOOSS
OOOXX
SOOOO
SSSOS
```

Task 5 - Lab# programming: Algotester Lab 7-8 (1 variant)

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

struct Node
{
    int value;
    Node *left;
    Node *right;
};

struct BinarySearchTree
{
    Node *root;
    int size;
};

enum Ident
{
    ID_insert,
    ID_contains,
    ID_treeSize,
    ID_print,
    ID_unknown
};

Ident parseToEnum(const string &id)
{
    if (id == "insert")
        return ID_insert;
    if (id == "contains")
        return ID_contains;
    if (id == "size")
        return ID_treeSize;
    if (id == "print")
        return ID_print;
    return ID_unknown;
}

void initializeTree(BinarySearchTree &tree)
{
    tree.root = nullptr;
    tree.size = 0;
}

Node *createNode(int value)
{
    Node *newNode = new Node;
    newNode->value = value;
    newNode->left = nullptr;
    newNode->right = nullptr;
    return newNode;
}

void insertNode(Node *&node, int value, int &treeSize)
{
    if (node == nullptr)
    {
        node = createNode(value);
        ++treeSize;
    }
    else if (value < node->value)
    {
        insertNode(node->left, value, treeSize);
    }
    else if (value > node->value)
    {
        insertNode(node->right, value, treeSize);
    }
}

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

void printTree(Node *node)
{
    if (node == nullptr)
        return;
    printTree(node->left);
    cout << node->value << " ";
    printTree(node->right);
}

void destroyTree(Node *node)
{
    if (node != nullptr)
    {
        destroyTree(node->left);
        destroyTree(node->right);
        delete node;
    }
}
```



```
int main()
{
    BinarySearchTree bst;
    initializeTree(bst);

    int q;
    cin >> q;

    for (int i = 0; i < q; i++)
    {
        string ident;
        cin >> ident;

        Ident identifier = parseToEnum(ident);
        switch (identifier)
        {
        case ID_insert:
        {
            int value;
            cin >> value;
            insertNode(bst.root, value, bst.size);
            break;
        }
        case ID_contains:
        {
            int value;
            cin >> value;
            if (containsNode(bst.root, value))
            {
                cout << "Yes" << endl;
            }
            else
            {
                cout << "No" << endl;
            }
            break;
        }
        case ID_treeSize:
            cout << bst.size << endl;
            break;
        case ID_print:
            printTree(bst.root);
            cout << endl;
            break;
        case ID_unknown:
            cout << "Unknown identifier!" << endl;
            break;
        }
    }

    destroyTree(bst.root);
    return 0;
}
```

```
11

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

Task 6 - Lab# programming: Algotester Lab 7-8 (2 variant)

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

struct Node {
    int value;
    Node* left;
    Node* right;

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

enum Ident {
    Ident_insert,
    Ident_contains,
    Ident_treeSize,
    Ident_print,
    Ident_unknown
};

Ident parseToEnum(const string& id) {
    if (id == "insert") return Ident_insert;
    if (id == "contains") return Ident_contains;
    if (id == "size") return Ident_treeSize;
    if (id == "print") return Ident_print;
    return Ident_unknown;
}

class BinarySearchTree {
private:
    Node* root;
    int tree_size;

    void insert(Node*& node, int value) {
        if (node == nullptr) {
            node = new Node(value);
            ++tree_size;
        } else if (value < node->value) {
            insert(node->left, value);
        } else if (value > node->value) {
            insert(node->right, value);
        }
    }

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

```
57      void print(Node* node, ostream& os) const {
58          if (node == nullptr) return;
59          print(node->left, os);
60          os << node->value << " ";
61          print(node->right, os);
62      }
63
64      void destroyTree(Node* node) {
65          if (node != nullptr) {
66              destroyTree(node->left);
67              destroyTree(node->right);
68              delete node;
69          }
70      }
71
72  public:
73      BinarySearchTree() : root(nullptr), tree_size(0) {}
74
75      ~BinarySearchTree() {
76          destroyTree(root);
77      }
78
79      void insert(int value) {
80          insert(root, value);
81      }
82
83      bool contains(int value) const {
84          return contains(root, value);
85      }
86
87      int size() const {
88          return tree_size;
89      }
90
91      friend ostream& operator<<(ostream& os, const BinarySearchTree& tree) {
92          tree.print(tree.root, os);
93          return os;
94      }
95  };
96
97  int main() {
98      BinarySearchTree bst;
99      int q;
100
101     cin >> q;
102     for (int i = 0; i < q; i++) {
103         string ident;
104         cin >> ident;
105
106         Ident identifier = parseToEnum(ident);
107         switch (identifier) {
108         case Ident_insert: {
109             int value;
110             cin >> value;
```

```cpp
int main() {
    BinarySearchTree bst;
    int q;

    cin >> q;
    for (int i = 0; i < q; i++) {
        string ident;
        cin >> ident;

        Ident identifier = parseToEnum(ident);
        switch (identifier) {
        case Ident_insert: {
            int value;
            cin >> value;
            bst.insert(value);
            break;
        }
        case Ident_contains: {
            int value;
            cin >> value;
            if (bst.contains(value))
            {
                cout << "Yes" << endl;
            }
            else
            {
                cout << "No" << endl;
            }
            break;
        }
        case Ident_treeSize:
            cout << bst.size() << endl;
            break;
        case Ident_print:
            cout << bst << endl;
            break;
        case Ident_unknown:
            cout << "Unknown identifier!" << endl;
            break;
        }
    }

    return 0;
}
```

```
11

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

Task 7 - Practice# programming: Class Practice Task (1год)

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

struct Node
{
    int data;
    Node *next;

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

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

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

    return prev;
}

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

Node *createList()
{
    Node *head = nullptr;
    Node *tail = nullptr;
    int n;

    cout << "Введіть кількість елементів у списку: ";
    cin >> n;

    if (n <= 0)
    {
        cout << "Список порожній!" << endl;
        return nullptr;
    }
    cout << "Введіть " << n << " цілих чисел:" << endl;
    for (int i = 0; i < n; ++i)
    {
        int value;
        cin >> value;

        Node *newNode = new Node(value);
        if (head == nullptr)
        {
            head = newNode;
            tail = newNode;
        }
        else
        {
            tail->next = newNode;
            tail = newNode;
        }
    }

    return head;
}

bool compare(Node *h1, Node *h2)
{
    bool same = true;
    int length1 = 0, length2 = 0;
    while (h1 != nullptr && h2 != nullptr)
    {
        if (h1->data != h2->data)
        {
            same = false;
        }
        h1 = h1->next;
        h2 = h2->next;
        if (h1 != nullptr){
            length1++;
        }
        if (h2 != nullptr){
            length2++;
        }
    }
    if ( length1 != length2){
        same = false;
    }

    return same;
}
```

```cpp
int main()
{
    Node *head1 = createList();

    cout << "Початковий список: ";
    printList(head1);

    head1 = reverse(head1);

    cout << "Реверсований список: ";
    printList(head1);

    Node *head2 = createList();

    cout << "Другий список: ";
    printList(head2);

    if (compare(head1, head2))
    {
        cout << "Другий список дорівнює реверсованому першому списку." << endl;
    }
    else
    {
        cout << "Другий список не дорівнює реверсованому першому списку." << endl;
    }

    return 0;
}
```

```
Введіть кількість елементів у списку: 5
Введіть 5 цілих чисел:
1 2 3 4 5
Початковий список: 1 2 3 4 5
Реверсований список: 5 4 3 2 1
Введіть кількість елементів у списку: 5
Введіть 5 цілих чисел:
5 4 3 2 6
Другий список: 5 4 3 2 6
Другий список не дорівнює реверсованому першому списку.
```

Task 8 - Practice# programming: Self Practice Task (15хв)

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

// Прямокутна ковзанка №1835

const int MAX_N = 1500;

int n, m, k;
int d[MAX_N][MAX_N];
int maxInRow[MAX_N][MAX_N];

void rowMax() {
    for (int i = 0; i < n; ++i) {
        int queue[MAX_N], head = 0, tail = 0;
        for (int j = 0; j < m; ++j) {
            if (head < tail && queue[head] <= j - k) {
                ++head;
            }

            while (head < tail && d[i][queue[tail - 1]] <= d[i][j]) {
                --tail;
            }

            queue[tail++] = j;

            if (j >= k - 1) {
                maxInRow[i][j - k + 1] = d[i][queue[head]];
            }
        }
    }
}

int minMax() {
    int result = INT_MAX;

    for (int j = 0; j <= m - k; ++j) {
        int queue[MAX_N], head = 0, tail = 0;

        for (int i = 0; i < n; ++i) {
            if (head < tail && queue[head] <= i - k) {
                ++head;
            }

            while (head < tail && maxInRow[queue[tail - 1]][j] <= maxInRow[i][j]) {
                --tail;
            }
```

```
33
34  int minMax() {
35      int result = INT_MAX;
36
37      for (int j = 0; j <= m - k; ++j) {
38          int queue[MAX_N], head = 0, tail = 0;
39
40          for (int i = 0; i < n; ++i) {
41              if (head < tail && queue[head] <= i - k) {
42                  ++head;
43              }
44
45              while (head < tail && maxInRow[queue[tail - 1]][j] <= maxInRow[i][j]) {
46                  --tail;
47              }
48
49              queue[tail++] = i;
50
51              if (i >= k - 1) {
52                  result = min(result, maxInRow[queue[head]][j]);
53              }
54          }
55      }
56
57      return result;
58  }
59
60  int main() {
61      cin >> n >> m >> k;
62
63      for (int i = 0; i < n; ++i) {
64          for (int j = 0; j < m; ++j) {
65              cin >> d[i][j];
66          }
67      }
68
69      rowMax();
70
71      int result = minMax();
72
73      cout << result << endl;
74
75      return 0;
76  }
77
```

```
3 3 2
2 5 4
4 1 3
3 2 1
3
```

Зустрічі з командою

З командою зустрічалися двічі, на зустрічах обговорювали питання та прогрес по епіку.

Висновок

В ході даного епіку я навчилась використовувати різні види класів, такі як: черга, список, стек, дерево, а також використовувати алгоритми обробки динамічних структур.