

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

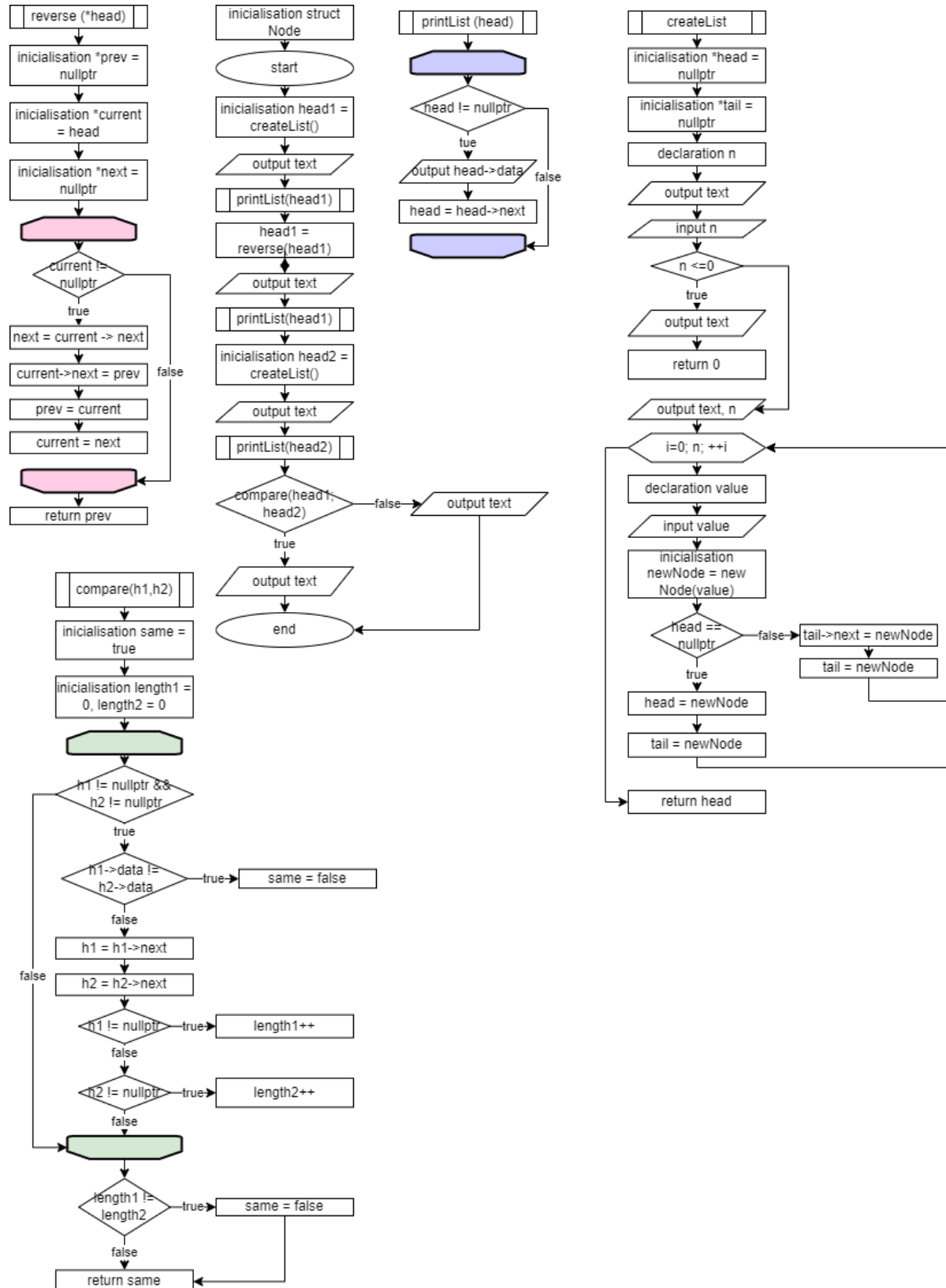
1. Навчитись створювати та використовувати динамічні структури, такі як: Черга, Стек, Списки, Дерево.
2. Навчитись виконувати алгоритми обробки динамічний структур.

## 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=a1aHR0cHM6Ly9wdXJlY29kZW50C5jb20vdWsvYXJjaGl2ZXNvMjQ4Mw&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 (10xв)

```
1  #include <iostream>
2  #include <fstream>
3  using namespace std;
4
5  struct Node
6  {
7      int data;
8      Node *next;
9  };
10
11 class SinglyLinkedList
12 {
13 private:
14     Node *head;
15
16 public:
17     SinglyLinkedList() : head(nullptr) {}
18
19     void createlist()
20     {
21         head = nullptr;
22     }
23
24     void addElements(int startPosition, int k, int value)
25     {
26         for (int i = 0; i < k; ++i)
27         {
28             addElement(startPosition + i, value + i);
29         }
30     }
31
32     void addElement(int position, int value)
33     {
34         Node *newNode = new Node{value, nullptr};
35
36         if (position <= 0 || !head)
37         {
38             newNode->next = head;
39             head = newNode;
40         }
41         else
42         {
43             Node *temp = head;
44             int index = 0;
45             while (temp->next && index < position - 1)
46             {
47                 temp = temp->next;
48                 index++;
49             }
50             newNode->next = temp->next;
51             temp->next = newNode;
52         }
53     }
54
55     void deleteElement(int element)
56     {
57         int position = element - 1;
58         if (!head)
59         {
60             cout << "Список порожній, видалення неможливе" << endl;
61             return;
62         }
63
64         if (position <= 0)
65         {
66             Node *toDelete = head;
67             head = head->next;
68             delete toDelete;
69         }
70         else
71         {
72             Node *temp = head;
73             int index = 0;
74             while (temp->next && index < position - 1)
75             {
76                 temp = temp->next;
77                 index++;
78             }
79             if (!temp->next)
80             {
81                 cout << "Некоректна позиція для видалення!" << endl;
82                 return;
83             }
84             Node *toDelete = temp->next;
85             temp->next = toDelete->next;
86             delete toDelete;
87         }
88     }
89
90     void printList() const
91     {
92         if (!head)
93         {
94             cout << "Список порожній" << endl;
95             return;
96         }
97         Node *temp = head;
98         while (temp)
99         {
100             cout << temp->data << " ";
101             temp = temp->next;
102         }
103         cout << endl;
104     }
105 }
```

```

106 void writeToFile(const string &filename) const
107 {
108     ofstream outFile(filename);
109     if (outFile.is_open())
110     {
111         Node *temp = head;
112         while (temp)
113         {
114             outFile << temp->data << " ";
115             temp = temp->next;
116         }
117         outFile.close();
118     }
119     else
120     {
121         cerr << "Не вдалося відкрити файл для запису!" << endl;
122         return;
123     }
124 }
125
126 void restoreFromFile(const string &filename)
127 {
128     deleteList();
129     ifstream inFile(filename);
130     if (inFile.is_open())
131     {
132         int value;
133         while (inFile >> value)
134         {
135             addElement(INT_MAX, value);
136         }
137         inFile.close();
138     }
139     else
140     {
141         cerr << "Не вдалося відкрити файл для читання!" << endl;
142         return;
143     }
144 }
145
146 void deleteList()
147 {
148     while (head)
149     {
150         Node *toDelete = head;
151         head = head->next;
152         delete toDelete;
153     }
154 }
155
156 ~SinglyLinkedList()
157 {
158     deleteList();
159 }
160
161 int main()
162 {
163     SinglyLinkedList list;
164     list.createList();
165
166     int el;
167     cout << "Введіть кількість елементів списку: ";
168     cin >> el;
169
170     for (int i = 0; i < el; i++)
171     {
172         int pos, val;
173         cout << "Введіть позицію елемента та його значення: ";
174         cin >> pos >> val;
175         list.addElement(pos, val);
176     }
177     list.printList();
178
179     int delpos;
180     cout << "Введіть позицію елемента який потрібно видалити: ";
181     cin >> delpos;
182
183     list.deleteElement(delpos);
184     list.printList();
185
186     int addpos, num, addval;
187     cout << "Введіть позицію елемента після якого потрібно додати нові, ";
188     cout << "скільки елементів додати та значення першого доданого: ";
189     cin >> addpos >> num >> addval;
190
191     list.addElements(addpos, num, addval);
192     list.printList();
193
194     list.writeToFile("list.txt");
195
196     list.deleteList();
197     list.printList();
198
199     list.restoreFromFile("list.txt");
200     list.printList();
201
202     list.deleteList();
203
204     remove("list.txt");
205
206     return 0;
207 }
208
209 }
210

```

```

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

```

1  #include <iostream>
2  #include <cstring>
3  using namespace std;
4
5  const int MAX_N = 1000;
6  const int MAX_M = 1000;
7
8  int main() {
9      int N, M;
10     char cave[MAX_N][MAX_M + 1];
11
12     cin >> N >> M;
13
14     for (int i = 0; i < N; ++i) {
15         cin >> cave[i];
16     }
17
18     for (int col = 0; col < M; ++col) {
19         int empty_row = N - 1;
20
21         for (int row = N - 1; row >= 0; --row) {
22             if (cave[row][col] == 'X') {
23                 empty_row = row - 1;
24             } else if (cave[row][col] == 'S') {
25                 cave[row][col] = '0';
26                 if (empty_row >= 0) {
27                     cave[empty_row][col] = 'S';
28                     --empty_row;
29                 }
30             }
31         }
32     }
33     cout << endl;
34     for (int i = 0; i < N; ++i) {
35         cout << cave[i] << endl;
36     }
37
38     return 0;
39 }

```

```

5 5
SS0SS
00000
S00XX
0000S
00S00

00000
000SS
000XX
S0000
SSS0S

```

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

```

1  #include <iostream>
2  #include <string>
3  using namespace std;
4
5  struct Node
6  {
7      int value;
8      Node *left;
9      Node *right;
10 };
11
12 struct BinarySearchTree
13 {
14     Node *root;
15     int size;
16 };
17
18 enum Ident
19 {
20     ID_insert,
21     ID_contains,
22     ID_treeSize,
23     ID_print,
24     ID_unknown
25 };
26
27 Ident parseToEnum(const string &id)
28 {
29     if (id == "insert")
30         return ID_insert;
31     if (id == "contains")
32         return ID_contains;
33     if (id == "size")
34         return ID_treeSize;
35     if (id == "print")
36         return ID_print;
37     return ID_unknown;
38 }
39
40 void initializeTree(BinarySearchTree &tree)
41 {
42     tree.root = nullptr;
43     tree.size = 0;
44 }
45
46 Node *createNode(int value)
47 {
48     Node *newNode = new Node;
49     newNode->value = value;
50     newNode->left = nullptr;
51     newNode->right = nullptr;
52     return newNode;
53 }
54
55 void insertNode(Node *&node, int value, int &treeSize)
56 {
57     if (node == nullptr)
58     {
59         node = createNode(value);
60         ++treeSize;
61     }
62     else if (value < node->value)
63     {
64         insertNode(node->left, value, treeSize);
65     }
66     else if (value > node->value)
67     {
68         insertNode(node->right, value, treeSize);
69     }
70 }
71
72 bool containsNode(Node *node, int value)
73 {
74     if (node == nullptr)
75     {
76         return false;
77     }
78     else if (value == node->value)
79     {
80         return true;
81     }
82     else if (value < node->value)
83     {
84         return containsNode(node->left, value);
85     }
86     else
87     {
88         return containsNode(node->right, value);
89     }
90 }
91
92 void printTree(Node *node)
93 {
94     if (node == nullptr)
95         return;
96     printTree(node->left);
97     cout << node->value << " ";
98     printTree(node->right);
99 }
100
101 void destroyTree(Node *node)
102 {
103     if (node != nullptr)
104     {
105         destroyTree(node->left);
106         destroyTree(node->right);
107         delete node;
108     }
109 }

```

```

111 int main()
112 {
113     BinarySearchTree bst;
114     initializeTree(bst);
115
116     int q;
117     cin >> q;
118
119     for (int i = 0; i < q; i++)
120     {
121         string ident;
122         cin >> ident;
123
124         Ident identifier = parseToEnum(ident);
125         switch (identifier)
126         {
127             case ID_insert:
128             {
129                 int value;
130                 cin >> value;
131                 insertNode(bst.root, value, bst.size);
132                 break;
133             }
134             case ID_contains:
135             {
136                 int value;
137                 cin >> value;
138                 if (containsNode(bst.root, value))
139                 {
140                     cout << "Yes" << endl;
141                 }
142                 else
143                 {
144                     cout << "No" << endl;
145                 }
146                 break;
147             }
148             case ID_treeSize:
149                 cout << bst.size << endl;
150                 break;
151             case ID_print:
152                 printTree(bst.root);
153                 cout << endl;
154                 break;
155             case ID_unknown:
156                 cout << "Unknown identifier!" << endl;
157                 break;
158         }
159     }
160
161     destroyTree(bst.root);
162     return 0;
163 }

```

```

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)



```

1  #include <iostream>
2  #include <string>
3  using namespace std;
4
5  struct Node {
6      int value;
7      Node* left;
8      Node* right;
9
10     Node(int val) : value(val), left(nullptr), right(nullptr) {}
11 };
12
13 enum Ident {
14     Ident_insert,
15     Ident_contains,
16     Ident_treeSize,
17     Ident_print,
18     Ident_unknown
19 };
20
21 Ident parseToEnum(const string& id) {
22     if (id == "insert") return Ident_insert;
23     if (id == "contains") return Ident_contains;
24     if (id == "size") return Ident_treeSize;
25     if (id == "print") return Ident_print;
26     return Ident_unknown;
27 }
28
29 class BinarySearchTree {
30 private:
31     Node* root;
32     int tree_size;
33
34     void insert(Node*& node, int value) {
35         if (node == nullptr) {
36             node = new Node(value);
37             ++tree_size;
38         } else if (value < node->value) {
39             insert(node->left, value);
40         } else if (value > node->value) {
41             insert(node->right, value);
42         }
43     }
44
45     bool contains(Node* node, int value) const {
46         if (node == nullptr) {
47             return false;
48         } else if (value == node->value) {
49             return true;
50         } else if (value < node->value) {
51             return contains(node->left, value);
52         } else {
53             return contains(node->right, value);
54         }
55     }

```

```

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;

```

```

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;
111                 bst.insert(value);
112                 break;
113             }
114             case Ident_contains: {
115                 int value;
116                 cin >> value;
117                 if (bst.contains(value))
118                 {
119                     cout << "Yes" << endl;
120                 }
121                 else
122                 {
123                     cout << "No" << endl;
124                 }
125                 break;
126             }
127             case Ident_treeSize:
128                 cout << bst.size() << endl;
129                 break;
130             case Ident_print:
131                 cout << bst << endl;
132                 break;
133             case Ident_unknown:
134                 cout << "Unknown identifier!" << endl;
135                 break;
136         }
137     }
138
139     return 0;
140 }

```

```

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)

```
1  #include <iostream>
2  using namespace std;
3
4  struct Node
5  {
6      int data;
7      Node *next;
8
9      Node(int value) : data(value), next(nullptr) {}
10 };
11
12 Node *reverse(Node *head)
13 {
14     Node *prev = nullptr;
15     Node *current = head;
16     Node *next = nullptr;
17
18     while (current != nullptr)
19     {
20         next = current->next;
21         current->next = prev;
22         prev = current;
23         current = next;
24     }
25
26     return prev;
27 }
28
29 void printList(Node *head)
30 {
31     while (head != nullptr)
32     {
33         cout << head->data << " ";
34         head = head->next;
35     }
36     cout << endl;
37 }
38
39 Node *createList()
40 {
41     Node *head = nullptr;
42     Node *tail = nullptr;
43     int n;
44
45     cout << "Введіть кількість елементів списку: ";
46     cin >> n;
47
48     if (n <= 0)
49     {
50         cout << "Список порожній!" << endl;
51         return nullptr;
52     }
53
54     cout << "Введіть " << n << " цілих чисел:" << endl;
55     for (int i = 0; i < n; ++i)
56     {
57         int value;
58         cin >> value;
59
60         Node *newNode = new Node(value);
61         if (head == nullptr)
62         {
63             head = newNode;
64             tail = newNode;
65         }
66         else
67         {
68             tail->next = newNode;
69             tail = newNode;
70         }
71     }
72
73     return head;
74 }
75
76 bool compare(Node *h1, Node *h2)
77 {
78     bool same = true;
79     int length1 = 0, length2 = 0;
80     while (h1 != nullptr && h2 != nullptr)
81     {
82         if (h1->data != h2->data)
83         {
84             same = false;
85         }
86         h1 = h1->next;
87         h2 = h2->next;
88         if (h1 != nullptr){
89             length1++;
90         }
91         if (h2 != nullptr){
92             length2++;
93         }
94     }
95     if (length1 != length2){
96         same = false;
97     }
98
99     return same;
100 }
```

```

102 int main()
103 {
104     Node *head1 = createList();
105
106     cout << "Початковий список: ";
107     printList(head1);
108
109     head1 = reverse(head1);
110
111     cout << "Реверсований список: ";
112     printList(head1);
113
114     Node *head2 = createList();
115
116     cout << "Другий список: ";
117     printList(head2);
118
119     if (compare(head1, head2))
120     {
121         cout << "Другий список дорівнює реверсованому першому списку." << endl;
122     }
123     else
124     {
125         cout << "Другий список не дорівнює реверсованому першому списку." << endl;
126     }
127
128     return 0;
129 }

```

```

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)

```

1  #include <iostream>
2  #include <limits>
3  using namespace std;
4
5  // Прямокутна ковзанка №1835
6
7  const int MAX_N = 1500;
8
9  int n, m, k;
10 int d[MAX_N][MAX_N];
11 int maxInRow[MAX_N][MAX_N];
12
13 void rowMax() {
14     for (int i = 0; i < n; ++i) {
15         int queue[MAX_N], head = 0, tail = 0;
16         for (int j = 0; j < m; ++j) {
17             if (head < tail && queue[head] <= j - k) {
18                 ++head;
19             }
20
21             while (head < tail && d[i][queue[tail - 1]] <= d[i][j]) {
22                 --tail;
23             }
24
25             queue[tail++] = j;
26
27             if (j >= k - 1) {
28                 maxInRow[i][j - k + 1] = d[i][queue[head]];
29             }
30         }
31     }
32 }
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             }

```

```

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.