

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 ІІІ-12

Смачило Іванна

Львів – 2024

Тема роботи

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

Мета роботи

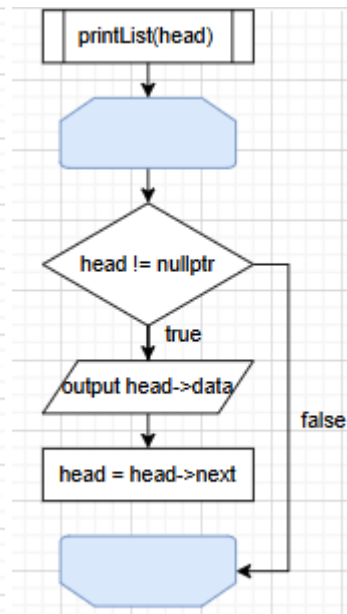
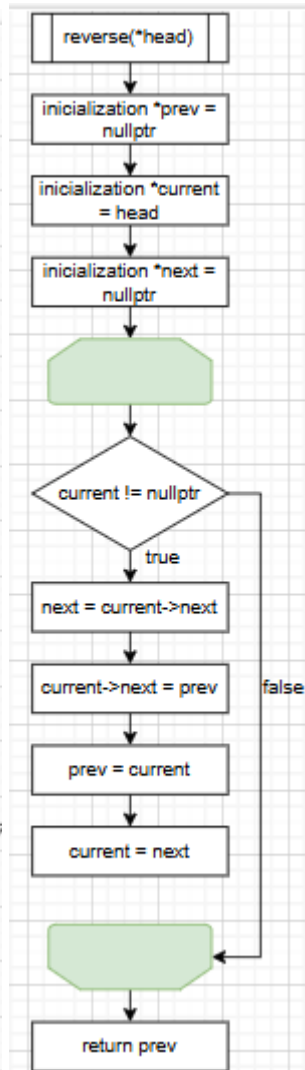
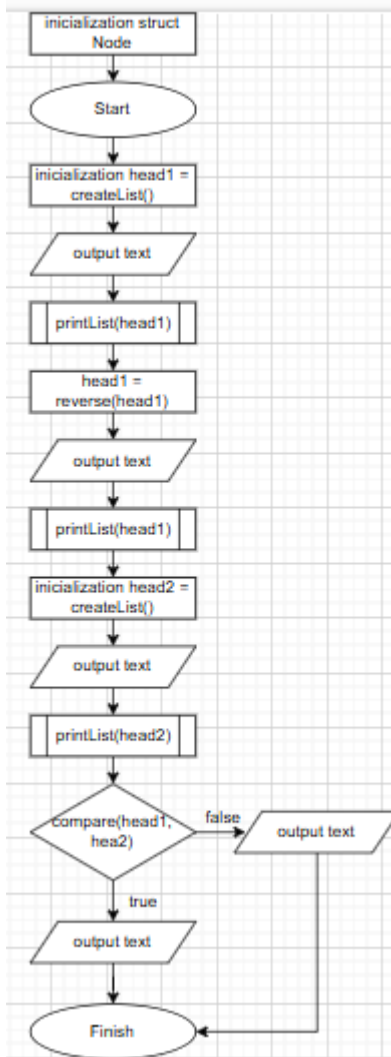
1. Навчитись створювати та використовувати динамічні структури, такі як: Черга, Стек, Списки, Дерево.
2. Навчитись виконувати алгоритми обробки динамічний структур.

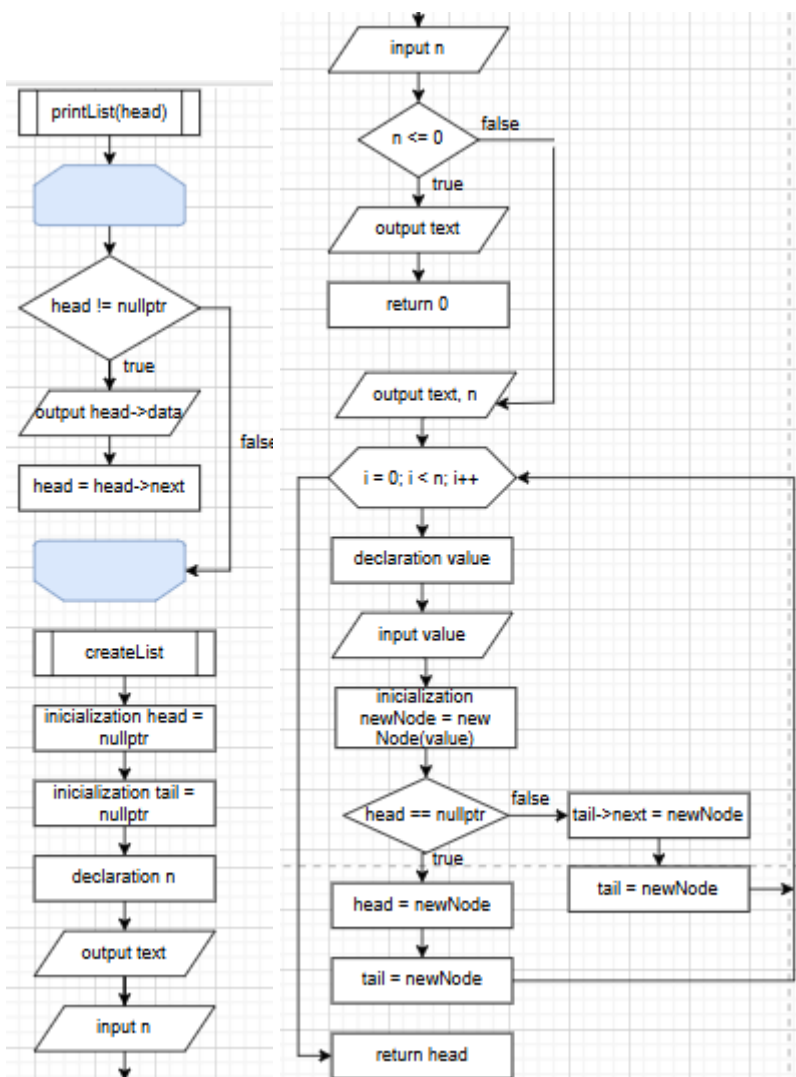
Теоретичні відомості

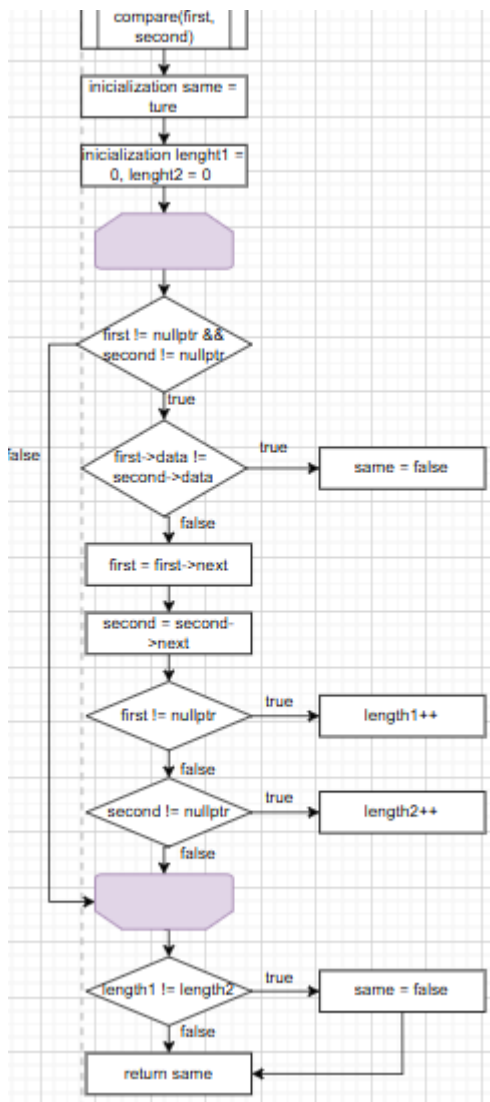
1. Стек і купа:
https://www.bestprog.net/uk/2019/09/18/c-the-concept-of-stack-operations-on-the-stack-an-example-implementation-of-the-stack-as-a-dynamic-array-ua/#google_vignette
<https://acode.com.ua/urok-111-stek-i-kupa/>
<https://dystosvita.org.ua/mod/page/view.php?id=888>
2. Динамічні масиви: <https://acode.com.ua/urok-90-dynamichni-masyvy/>
3. Черга: <https://itproger.com.ua/spravka/cpp/queue>
<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. Зв'язні списки: <https://prometheus.org.ua/cs50/sections/section6.html>
5. Список list: <https://codelessons.dev/ru/spisok-list-v-s-polnyj-material/>
6. Circular linked list: <https://www.geeksforgeeks.org/circular-linked-list-in-cpp/>
7. Бінарні дерева: https://purecodecpp.com/uk/archives/2483#google_vignette
8. Перевантаження операторів вводу і виводу:
<https://acode.com.ua/urok-141-perevantazhennya-operatoriv-vvodu-i-vyvodu/#toc-0>
9. Reverse in C++: <https://www.geeksforgeeks.org/stdreverse-in-c/>
10. Compare in C++: <https://www.geeksforgeeks.org/stdstringcompare-in-c/>
11. C++ Program For Inserting A Node In A Linked List:
<https://www.geeksforgeeks.org/cpp-program-for-inserting-a-node-in-a-linked-list/>

Виконання роботи

Task 2 - Requirements management (understand tasks) and design activities (draw flow diagram for Class Practice Task)







Task 3 - Lab# programming: VNS Lab 10

Завдання: 17. Записи в лінійному списку містять ключове поле типу `*char` (рядок символів). Сформувати двонаправлений список. Знищити елемент із заданим номером. Додати `K` елементів у початок списку.

```

1  #include <iostream>
2  #include <fstream>
3  #include <string>
4
5  using namespace std;
6
7  class Node
8  {
9  public:
10     string data;
11     Node* next;
12     Node* prev;
13
14     Node(string data)
15     {
16         this->data = data;
17         this->next = nullptr;
18         this->prev = nullptr;
19     }
20 };
21
22 class DoublyLinkedList
23 {
24 public:
25     Node* head;
26     Node* tail;
27
28     DoublyLinkedList()
29     {
30         head = tail = nullptr;
31     }
32
33     void atBeginning(string data)
34     {
35         Node* newNode = new Node(data);
36         if (head == nullptr)
37         {
38             head = tail = newNode;
39         }
40         else

```

```

40         else
41         {
42             newNode->next = head;
43             head->prev = newNode;
44             head = newNode;
45         }
46     }
47
48     void atEnd(string data)
49     {
50         Node* newNode = new Node(data);
51         if (head == nullptr)
52         {
53             head = tail = newNode;
54         }
55         else
56         {
57             tail->next = newNode;
58             newNode->prev = tail;
59             tail = newNode;
60         }
61     }
62
63     void deleteNode(Node* node)
64     {
65         if (node == nullptr)
66         {
67             return;
68         }
69
70         if (node == head)
71         {
72             head = node->next;
73         }
74         else
75         {
76             node->prev->next = node->next;
77         }

```

```

79         if (node == tail)
80         {
81             tail = node->prev;
82         }
83         else
84         {
85             node->next->prev = node->prev;
86         }
87         delete node;
88     }
89
90     void deleteNode(string data)
91     {
92         Node* current = head;
93         while (current != nullptr)
94         {
95             if (current->data == data)
96             {
97                 deleteNode(current);
98                 return;
99             }
100             current = current->next;
101         }
102         cout << "Node not found!" << "\n";
103     }
104
105     void printList()
106     {
107         Node* current = head;
108         while (current != nullptr)
109         {
110             cout << current->data << " ";
111             current = current->next;
112         }
113         cout << "\n";
114     }
115 }

```

```

117 void writeToFile(const string& lab10)
118 {
119     ofstream file(lab10);
120     Node* current = head;
121
122     while (current != nullptr)
123     {
124         file << current->data << "\n";
125         current = current->next;
126     }
127     file.close();
128 }
129
130 void readFromFile(const string& lab10)
131 {
132     ifstream file(lab10);
133     string data;
134
135     while (getline(file, data))
136     {
137         atEnd(data);
138     }
139     file.close();
140 }
141
142 int main()
143 {
144     DoublyLinkedList list;
145
146     list.atEnd("A");
147     list.atEnd("B");
148     list.atBeginning("X");
149     list.atEnd("C");
150
151     list.printList();
152
153     list.deleteNode("B");
154 }

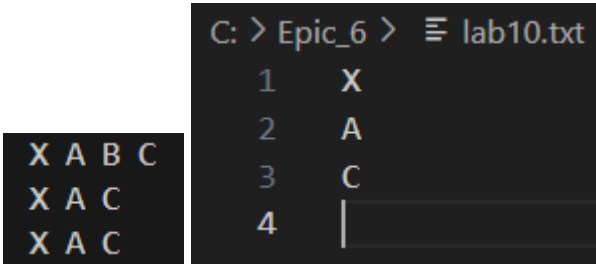
```

```

155
156 list.printList();
157
158 list.writeToFile("lab10.txt");
159
160 Node* current = list.head;
161 while (current != nullptr)
162 {
163     Node* temp = current;
164     current = current->next;
165     delete temp;
166 }
167 list.head = list.tail = nullptr;
168
169 list.readFromFile("lab10.txt");
170
171 list.printList();
172
173 return 0;
174 }

```

Результат:



Task 4 - Lab# programming: Algotester Lab 5

```
1  #include <iostream>
2  #include <cstring>
3
4  using namespace std;
5
6  const int MAX_N = 1000;
7  const int MAX_M = 1000;
8
9  int main()
10 {
11     int N, M;
12     char cave[MAX_N][MAX_M + 1];
13
14     cin >> N >> M;
15
16     for (int i = 0; i < N; i++)
17     {
18         cin >> cave[i];
19     }
20
21     for (int c = 0; c < M; c++)
22     {
23         int empty_row = N - 1;
24
25         for (int r = N - 1; r >= 0; r--)
26         {
27             if (cave[r][c] == 'X')
28             {
29                 empty_row = r - 1;
30             }
31             else if (cave[r][c] == 'S')
32             {
33                 cave[r][c] = '0';
34                 if (empty_row >= 0)
35                 {
36                     cave[empty_row][c] = 'S';
37                     empty_row--;
38                 }
39             }
40         }
41     }
42     cout << "\n";
43     for (int i = 0; i < N; i++)
44     {
45         cout << cave[i] << "\n";
46     }
47
48     return 0;
49 }
```

Результат:

```
5 5
SSOSS
00000
S00XX
0000S
00S00

00000
000SS
000XX
S0000
SSSOS
```

Created	Compiler	Result	Time (sec.)	Memory (MiB)	Actions
34 minutes ago	C++ 23	Accepted	0.021	1.879	View

Task 5 - Lab# programming: Algotester Lab 7-8

Variant 3:

```
1  #include <iostream>
2
3  using namespace std;
4
5  struct Node
6  {
7      int value;
8      Node* left;
9      Node* right;
10
11      Node(int val) : value(val), left(nullptr), right(nullptr) {}
12  };
13
14  class BinarySearchTree
15  {
16  public:
17      Node* root;
18
19      BinarySearchTree() : root(nullptr) {}
20
21      void insert(int value)
22      {
23          root = insertRecursive(root, value);
24      }
25
26      bool contains(int value)
27      {
28          return search(root, value);
29      }
30
31      int size()
32      {
33          return sizeRecursive(root);
34      }
35
36      void print()
37      {
38          printInOrder(root);
39          cout << "\n";
40      }
41
42  private:
43      Node* insertRecursive(Node* node, int value)
44      {
45          if (node == nullptr)
46          {
47              return new Node(value);
48          }
49
50          if (value < node->value)
51          {
52              node->left = insertRecursive(node->left, value);
53          }
54          else if (value > node->value)
55          {
56              node->right = insertRecursive(node->right, value);
57          }
58
59          return node;
60      }
61
62      bool search(Node* node, int value)
63      {
64          if (node == nullptr)
65          {
66              return false;
67          }
68
69          if (value == node->value)
70          {
71              return true;
72          }
73
74          if (value < node->value)
75          {
76              return search(node->left, value);
77          }
78          else
79          {
80              return search(node->right, value);
81          }
82      }
83  }
```

```
81  }
82  }
83
84  int sizeRecursive(Node* node)
85  {
86      if (node == nullptr)
87      {
88          return 0;
89      }
90
91      return 1 + sizeRecursive(node->left) + sizeRecursive(node->right);
92  }
93
94  void printInOrder(Node* node)
95  {
96      if (node == nullptr)
97      {
98          return;
99      }
100
101      printInOrder(node->left);
102      cout << node->value << " ";
103      printInOrder(node->right);
104  }
105 };
106
107 int main() {
108     int Q;
109     cin >> Q;
110
111     BinarySearchTree tree;
112
113     while (Q--)
114     {
115         string command;
116         cin >> command;
117
118         if (command == "insert")
119         {
120             int value;
121             cin >> value;
122             tree.insert(value);
123         }
124         else if (command == "contains")
125         {
126             int value;
127             cin >> value;
128             cout << (tree.contains(value) ? "Yes" : "No") << "\n";
129         }
130         else if (command == "size")
131         {
132             cout << tree.size() << "\n";
133         }
134         else if (command == "print")
135         {
136             tree.print();
137         }
138     }
139
140     return 0;
141 }
```


Результат:

```
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
```

Created	Compiler	Result	Time (sec.)	Memory (MiB)	Actions
2 days ago	C++ 23	Accepted	0.008	1.285	View

Variant 1:

```
1  #include <iostream>
2
3  using namespace std;
4
5  template <typename T>
6  class DoublyLinkedList
7  {
8  private:
9      struct Node
10     {
11         T value;
12         Node* next;
13         Node* prev;
14         Node(T val) : value(val), next(nullptr), prev(nullptr) {}
15     };
16
17     Node* head;
18     Node* tail;
19     int size;
20
21 public:
22     DoublyLinkedList() : head(nullptr), tail(nullptr), size(0) {}
23     ~DoublyLinkedList() { clear(); }
24
25     void insert(int index, int n, T* values)
26     {
27         if (index < 0 || index > size) return;
28
29         Node* current = head;
30         Node* prevNode = nullptr;
31
32         for (int i = 0; i < index; i++)
33         {
34             prevNode = current;
35             current = current->next;
36         }
37
38         for (int i = 0; i < n; i++)
39         {
40             Node* newNode = new Node(values[i]);
41
42             if (!head)
43             {
44                 head = tail = newNode;
45             }
46             else if (!prevNode)
47             {
48                 newNode->next = head;
49                 head->prev = newNode;
50                 head = newNode;
51             }
52             else
53             {
54                 newNode->next = current;
55                 newNode->prev = prevNode;
56                 if (prevNode) prevNode->next = newNode;
57                 if (current) current->prev = newNode;
58             }
59
60             prevNode = newNode;
61             if (!current) tail = newNode;
62         }
63
64         size += n;
65     }
66
67     void erase(int index, int n)
68     {
69         if (index < 0 || index >= size || n <= 0) return;
70
71         Node* current = head;
72
73         for (int i = 0; i < index; i++)
74         {
75             current = current->next;
76         }
77     }
```

```

79     for (int i = 0; i < n && current; i++)
80     {
81         Node* toDelete = current;
82         if (toDelete->prev) toDelete->prev->next = toDelete->next;
83         if (toDelete->next) toDelete->next->prev = toDelete->prev;
84
85         if (toDelete == head) head = toDelete->next;
86         if (toDelete == tail) tail = toDelete->prev;
87
88         current = current->next;
89         delete toDelete;
90         size--;
91     }
92 }
93
94 int getSize() const
95 {
96     return size;
97 }
98
99 T get(int index) const
100 {
101     if (index < 0 || index >= size) throw out_of_range("Index out of range");
102
103     Node* current = head;
104     for (int i = 0; i < index; i++)
105     {
106         current = current->next;
107     }
108
109     return current->value;
110 }
111
112 void set(int index, T value)
113 {
114     if (index < 0 || index >= size) throw out_of_range("Index out of range");
115
116     Node* current = head;

```

```

117     for (int i = 0; i < index; i++)
118     {
119         current = current->next;
120     }
121
122     current->value = value;
123 }
124
125 void print() const
126 {
127     Node* current = head;
128     while (current)
129     {
130         cout << current->value << " ";
131         current = current->next;
132     }
133     cout << "\n";
134 }
135
136 void clear()
137 {
138     Node* current = head;
139     while (current)
140     {
141         Node* toDelete = current;
142         current = current->next;
143         delete toDelete;
144     }
145
146     head = tail = nullptr;
147     size = 0;
148 }
149
150 friend ostream& operator<<(ostream& os, const DoublyLinkedList<T>& list)
151 {
152     Node* current = list.head;
153     while (current)
154     {
155         os << current->value << " ";
156         current = current->next;
157     }
158     os << "\n";
159     return os;
160 }

```

```

155         os << current->value << " ";
156         current = current->next;
157     }
158     return os;
159 }
160 };
161
162 int main()
163 {
164     DoublyLinkedList<int> list;
165     int Q;
166     cin >> Q;
167
168     while (Q--) {
169         string command;
170         cin >> command;
171
172         if (command == "insert")
173         {
174             int index, n;
175             cin >> index >> n;
176
177             int* values = new int[n];
178             for (int i = 0; i < n; i++)
179             {
180                 cin >> values[i];
181             }
182
183             list.insert(index, n, values);
184             delete[] values;
185         }
186         else if (command == "erase")
187         {
188             int index, n;
189             cin >> index >> n;
190             list.erase(index, n);
191         }

```

```

192         else if (command == "size")
193         {
194             cout << list.getSize() << "\n";
195         }
196         else if (command == "get")
197         {
198             int index;
199             cin >> index;
200             try
201             {
202                 cout << list.get(index) << "\n";
203             }
204             catch (const out_of_range& e)
205             {
206                 cout << "Error: " << e.what() << "\n";
207             }
208         }
209         else if (command == "set")
210         {
211             int index, value;
212             cin >> index >> value;
213             try
214             {
215                 list.set(index, value);
216             }
217             catch (const out_of_range& e)
218             {
219                 cout << "Error: " << e.what() << "\n";
220             }
221         }
222         else if (command == "print")
223         {
224             cout << list << "\n";
225         }
226     }
227     return 0;
228 }

```

Результат:

```

5

insert
0 3
1 2 3

erase
0 2

set
0 10

size
1
print
10

```

Created	Compiler	Result	Time (sec.)	Memory (MiB)	Actions
a day ago	C++ 23	Accepted	0.008	1.223	View

Task 6 - Practice# programming: Class Practice Task

```

1  #include <iostream>
2
3  using namespace std;
4
5  struct Node
6  {
7      int data;
8      Node *next;
9
10     Node(int value) : data(value), next(nullptr) {}
11 };
12
13 Node *reverse(Node *head)
14 {
15     Node *prev = nullptr;
16     Node *current = head;
17     Node *next = nullptr;
18
19     while (current != nullptr)
20     {
21         next = current->next;
22         current->next = prev;
23         prev = current;
24         current = next;
25     }
26
27     return prev;
28 }
29
30 void printList(Node *head)
31 {
32     while (head != nullptr)
33     {
34         cout << head->data << " ";
35         head = head->next;
36     }
37     cout << "\n";
38 }
39

```

```

39
40 Node *createList()
41 {
42     Node *head = nullptr;
43     Node *tail = nullptr;
44     int n;
45
46     cout << "Enter number of elements in list: ";
47     cin >> n;
48
49     if (n <= 0)
50     {
51         cout << "List is empty!" << "\n";
52         return nullptr;
53     }
54
55     cout << "Enter " << n << " integer numbers:" << "\n";
56     for (int i = 0; i < n; i++)
57     {
58         int value;
59         cin >> value;
60
61         Node *newNode = new Node(value);
62
63         if (head == nullptr)
64         {
65             head = newNode;
66             tail = newNode;
67         }
68         else
69         {
70             tail->next = newNode;
71             tail = newNode;
72         }
73     }
74
75     return head;
76 }
77

```

```

78 bool compare(Node *first, Node *second)
79 {
80     bool same = true;
81     int length1 = 0, length2 = 0;
82     while (first != nullptr && second != nullptr)
83     {
84         if (first->data != second->data)
85         {
86             same = false;
87         }
88         first = first->next;
89         second = second->next;
90         if (first != nullptr)
91         {
92             length1++;
93         }
94         if (second != nullptr)
95         {
96             length2++;
97         }
98     }
99     if (length1 != length2)
100     {
101         same = false;
102     }
103     return same;
104 }
105
106 int main()
107 {
108     Node *head1 = createList();
109
110     cout << "First list: ";
111     printList(head1);
112
113     head1 = reverse(head1);
114
115     cout << "Reversed list: ";
116     printList(head1);
117 }

```

```

118
119 Node *head2 = createList();
120
121 cout << "Second list: ";
122 printList(head2);
123
124 if (compare(head1, head2))
125 {
126     cout << "Second list is the same as reversed list number one!" << "\n";
127 }
128 else
129 {
130     cout << "Second list isn't the same as reversed list number one!" << "\n";
131 }
132
133 return 0;
134 }

```

Результат:

```

Enter number of elements in list: 5
Enter 5 integer numbers:
3
6
7
4
2
First list: 3 6 7 4 2
Reversed list: 2 4 7 6 3
Enter number of elements in list: 5
Enter 5 integer numbers:
2
4
7
6
3
Second list: 2 4 7 6 3
Second list is the same as reversed list number one!

```

```

1  #include <iostream>
2
3  using namespace std;
4
5  struct Node
6  {
7      int data;
8      Node* left;
9      Node* right;
10     Node(int value): data(value), left(nullptr), right(nullptr) {}
11 };
12
13 void Insert(Node*& root, int value);
14 void Show(Node* root);
15 void TreeFree(Node*& root);
16 Node* TreeMirror(Node*& root);
17 void TreeSum(Node*& root);
18
19
20 int main()
21 {
22     Node* myTree = nullptr;
23     Insert(myTree, 50);
24     Insert(myTree, 40);
25     Insert(myTree, 60);
26     Insert(myTree, 30);
27     Insert(myTree, 70);
28     Insert(myTree, 80);
29     Insert(myTree, 20);
30
31     cout << "Tree: ";
32     Show(myTree);
33     cout << "\n";
34     cout << "Mirrored Tree: ";
35     TreeMirror(myTree);
36     Show(myTree);
37     TreeMirror(myTree);
38     cout << "\n" << "Tree of sum: ";
39     TreeSum(myTree);
40     Show(myTree);

```

```

41     TreeFree(myTree);
42
43     return 0;
44 }
45
46 void Insert(Node*& root, int value)
47 {
48     if (root == nullptr)
49     {
50         root = new Node(value);
51         return;
52     }
53     if (root->data == value)
54         return;
55
56     if (root->data > value)
57     {
58         if (root->left == nullptr)
59         {
60             root->left = new Node(value);
61             return;
62         }
63         Insert(root->left, value);
64     }
65     if (root->data < value)
66     {
67         if (root->right == nullptr)
68         {
69             root->right = new Node(value);
70             return;
71         }
72         Insert(root->right, value);
73     }
74 }
75
76 void Show(Node* root)
77 {
78     if (root != nullptr)
79     {
80         Show(root->left);
81         cout << root->data << " ";
82         Show(root->right);
83     }
84 }
85
86 void TreeFree(Node*& root)
87 {
88     if (root != nullptr)
89     {
90         TreeFree(root->left);
91         TreeFree(root->right);
92         delete root;
93     }
94 }
95
96 Node* TreeMirror(Node*& root)
97 {
98     if (root == nullptr)
99         return nullptr;
100    Node* left = TreeMirror(root->left);
101    Node* right = TreeMirror(root->right);
102
103    root->right = left;
104    root->left = right;
105    return root;
106 }
107
108 void TreeSum(Node*& root)
109 {
110     if (root == nullptr)
111         return;
112
113     TreeSum(root->left);
114     TreeSum(root->right);
115
116     if (root->left != nullptr)
117         root->data += root->left->data;

```

```

80         Show(root->left);
81         cout << root->data << " ";
82         Show(root->right);
83     }
84 }
85
86 void TreeFree(Node*& root)
87 {
88     if (root != nullptr)
89     {
90         TreeFree(root->left);
91         TreeFree(root->right);
92         delete root;
93     }
94 }
95
96 Node* TreeMirror(Node*& root)
97 {
98     if (root == nullptr)
99         return nullptr;
100    Node* left = TreeMirror(root->left);
101    Node* right = TreeMirror(root->right);
102
103    root->right = left;
104    root->left = right;
105    return root;
106 }
107
108 void TreeSum(Node*& root)
109 {
110     if (root == nullptr)
111         return;
112
113     TreeSum(root->left);
114     TreeSum(root->right);
115
116     if (root->left != nullptr)
117         root->data += root->left->data;

```

```

96 Node* TreeMirror(Node*& root)
97 {
98     if (root == nullptr)
99         return nullptr;
100    Node* left = TreeMirror(root->left);
101    Node* right = TreeMirror(root->right);
102
103    root->right = left;
104    root->left = right;
105    return root;
106 }
107
108 void TreeSum(Node*& root)
109 {
110     if (root == nullptr)
111         return;
112
113     TreeSum(root->left);
114     TreeSum(root->right);
115
116     if (root->left != nullptr)
117         root->data += root->left->data;
118
119     if (root->right != nullptr)
120         root->data += root->right->data;
121 }

```

Результат:

```
Tree: 20 30 40 50 60 70 80
Mirrored Tree: 80 70 60 50 40 30 20
Tree of sum: 20 50 90 350 210 150 80
```

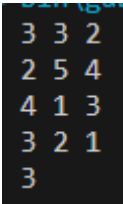
Task 7 - Practice# programming: Self Practice Task

```
1  #include <iostream>
2  #include <climits>
3
4  using namespace std;
5
6  const int MAX_N = 1500;
7
8  int n, m, k;
9  int f[MAX_N][MAX_N];
10 int InRowMax[MAX_N][MAX_N];
11
12 void MaxRow()
13 {
14     for (int i = 0; i < n; i++)
15     {
16         int queue[MAX_N], head = 0, tail = 0;
17         for (int j = 0; j < m; j++)
18         {
19             if (head < tail && queue[head] <= j - k)
20             {
21                 head++;
22             }
23
24             while (head < tail && f[i][queue[tail - 1]] <= f[i][j])
25             {
26                 tail--;
27             }
28
29             queue[tail++] = j;
30
31             if (j >= k - 1)
32             {
33                 InRowMax[i][j - k + 1] = f[i][queue[head]];
34             }
35         }
36     }
37 }
38
```

```
39 int minMax()
40 {
41     int result = INT_MAX;
42
43     for (int j = 0; j <= m - k; j++)
44     {
45         int queue[MAX_N], head = 0, tail = 0;
46
47         for (int i = 0; i < n; i++)
48         {
49             if (head < tail && queue[head] <= i - k)
50             {
51                 head++;
52             }
53
54             while (head < tail && InRowMax[queue[tail - 1]][j] <= InRowMax[i][j])
55             {
56                 tail--;
57             }
58
59             queue[tail++] = i;
60
61             if (i >= k - 1) {
62                 result = min(result, InRowMax[queue[head]][j]);
63             }
64         }
65     }
66
67     return result;
68 }
```

```
70 int main()
71 {
72     cin >> n >> m >> k;
73
74     for (int i = 0; i < n; i++)
75     {
76         for (int j = 0; j < m; j++)
77         {
78             cin >> f[i][j];
79         }
80     }
81
82     MaxRow();
83
84     int result = minMax();
85
86     cout << result << "\n";
87
88     return 0;
89 }
```

Результат:

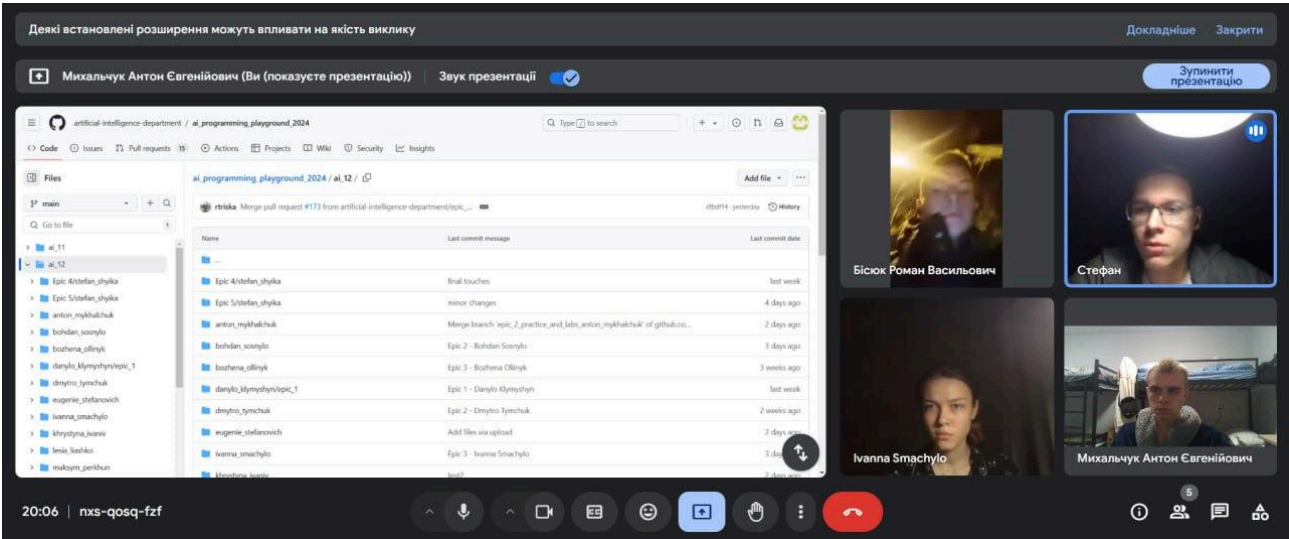
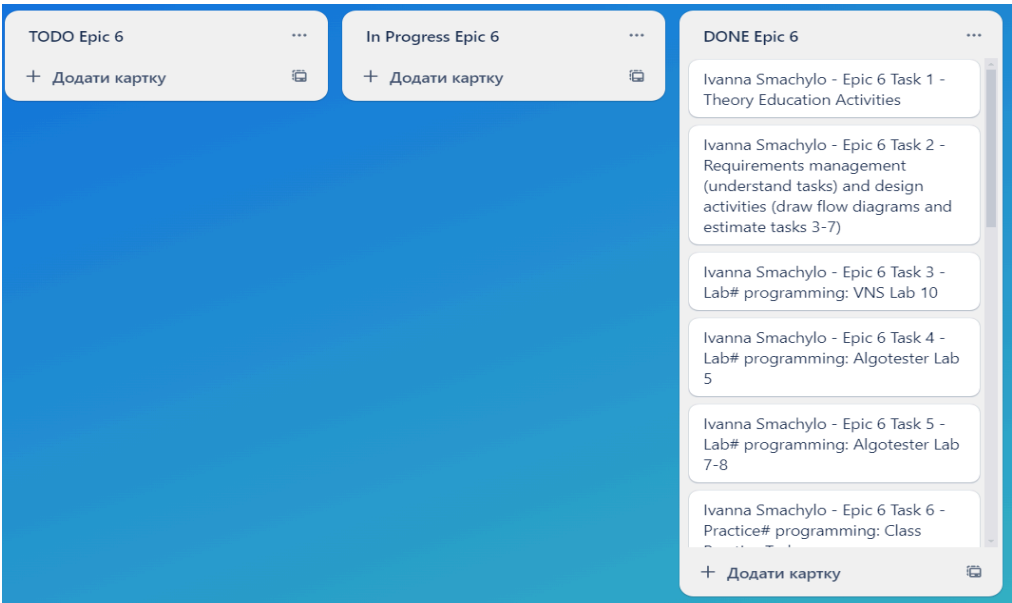


Created	Compiler	Result	Time (sec.)	Memory (MiB)	Actions
a few seconds ago	C++ 23	Accepted	0.971	18.516	View

Зустрічі з командою:

Зустрічались двічі для обговорення задач, поставлених в шостому епіку.

Створили нову дошку в Trello й бачили прогрес одне одного:



Висновок: в ході роботи над даним епіком я навчилась використовувати на практиці нові знання, такі як стек і купа, динамічні масиви, черга, зв'язні списки, список list, бінарні дерева, перевантаження операторів вводу і виводу, reverse, compare in C++.