

Міністерство освіти і науки України
Національний університет «Львівська політехніка»
Кафедра систем штучного інтелекту



Звіт

про виконання лабораторних та практичних робіт блоку № 4

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

до:

Практичних Робіт до блоку № 1

Виконала:

Студентка групи ІІІ-12
Костак Олеся Михайлівна

Львів – 2024 р.

Тема роботи: Динамічні структури (Черга, Стек, Списки, Дерево).

Алгоритми обробки динамічних структур.

Мета роботи: Освоїти роботу з динамічними структурами даних, такими як черга, стек, списки та дерева. Навчитися реалізовувати базові операції над цими структурами, включаючи створення, додавання, видалення, пошук та обхід елементів. Розвинути навички алгоритмічної обробки динамічних структур для ефективного вирішення прикладних задач.

Теоретичні відомості:

1. Теоретичні відомості з переліком важливих тем:

- Тема №*.1: Основи Динамічних Структур Даних.
- Тема №*.2: Стек.
- Тема №*.3: Черга.
- Тема №*.4: Зв'язні Списки.
- Тема №*.5: Дерева.
- Тема №*.6: Алгоритми Обробки Динамічних Структур.

2. Індивідуальний план опрацювання теорії:

Джерела Інформації:

1. Сайт [geeksforgeeks](https://www.geeksforgeeks.com/)
2. Сайт [programiz.com](https://www.programiz.com/)
3. Сайт [w3schools.com](https://www.w3schools.com/)
4. Ютуб-канал Блоган
5. Лекції Пшеничного
6. Сайт [cppreference.com](https://www.cppreference.com/)

Тема №*.1: Основи Динамічних Структур Даних.

- *Що опрацьовано:*
 - Стаття [Dynamic Data Structures](#)

Тема №*.2: Стек.

- *Що опрацьовано:*
 - Відео [C++ • Теорія • Урок 141 • ADT • Стек](#)

Тема №*.3: Черга.

- *Що опрацьовано*
 - Стаття [Introduction to Queue Data Structure](#)

Тема №*.4: Зв'язні Списки.

- *Що опрацьовано:*
 - Відео [C++ • Теорія • Урок 139 • ADT • Однозв'язний список](#)
 - Стаття [Delete a Linked List node at a given position](#)

Тема №*.5: Дерева.

- *Що опрацьовано:*
 - Відео [C++ • Теорія • Урок 144 • ADT • Бінарне дерево](#)
 - Стаття [Binary Tree](#)

Тема №*.6: Алгоритми Обробки Динамічних Структур.

- Що опрацьовано:
 - [Searching Algorithms](#)

Виконання роботи:

Завдання №1 VNS Lab 10 - Task 1

1. Опрацювання завдання та вимог до програм та середовища:

- Варіант завдання: 5
- Деталі завдання:

Для кожного варіанту розробити такі функції:

1. Створення списку.
2. Додавання елемента в список (у відповідності зі своїм варіантом).
3. Знищення елемента зі списку (у відповідності зі своїм варіантом).
4. Друк списку.
5. Запис списку у файл.
6. Знищення списку.
7. Відновлення списку з файлу.

Відповідно до мого варіанту:

Записи в лінійному списку містять ключове поле типу `int`. Сформувати однонаправлений список. Знищити з нього `K` елементів, починаючи із заданого номера, додати `K` елементів, починаючи із заданого номера;

2. Код програми

```
1  #include <iostream>
2  #include <fstream>
3
4  struct Node
5  {
6      int data;
7      Node* next;
8      Node(int value): data(value), next(nullptr) {}
9  };
10
11 void Show(Node* head);
12 void AddElement(Node*& head, int value, int pos);
13 void DeleteElement(Node*& head, int pos);
14 void SaveToFile(Node*& head, const char* filename);
15 void DeleteList(Node*& head);
16 void RestoreList(Node*& head, const char* filename);
17
18
19 int main()
20 {
21     Node* head = nullptr;
22     const char* filename = "linkedList.txt";
23
24     int userOption;
25     do
26     {
27         std::cout << "Chose an option:" << std::endl <<
28         "1. Add elements" << std::endl <<
29         "2. Delete elements" << std::endl <<
30         "3. Show elements" << std::endl <<
31         "4. Write to file" << std::endl <<
32         "5. Delete list" << std::endl <<
33         "6. Restore list" << std::endl <<
34         "7. Exit" << std::endl;
35         std::cin >> userOption;
36
37         switch (userOption)
38         {
39             case 1:
40             {
41                 std::cout << "Enter the position where you want to add elements. " <<
42                 "Enter how many elements you want to add: " << std::endl;
43                 int pos, amount, value;
44                 std::cin >> pos >> amount;
45                 for(int i = 0; i != amount; i++)
46                 {
47                     std::cout << "Enter the value of " << i + 1 << " element" << std::endl;
48                     std::cin >> value;
49                     AddElement(head, value, pos++);
50                 }
51                 break;
52             case 2:
53                 DeleteElement(head, pos);
```

```

54         std::cout << "Enter the position where you want to delete elements. " <<
55         "Enter how many elements you want to delete: " << std::endl;
56         int pos, amount;
57         std::cin >> pos >> amount;
58         for(int i = 0; i != amount; i++)
59             DeleteElement(head, pos);
60     }
61     case 3:
62         Show(head);
63         break;
64     case 4:
65         SaveToFile(head, filename);
66         break;
67     case 5:
68         Deletelist(head);
69         break;
70     case 6:
71         RestoreList(head, filename);
72         break;
73     default:
74         break;
75     }
76 }
77 while (userOption != 7);
78
79 return 0;
80 }
81
82 void Show(Node* head)
83 {
84     if (head == nullptr)
85         std::cout << "List is empty";
86     else
87     {
88         Node* current = head;
89         while (current != nullptr)
90         {
91             std::cout << current->data << " ";
92             current = current->next;
93         }
94     }
95     std::cout << std::endl;
96 }
97
98 > void AddElement(Node*& head, int value, int pos) ...

```

```

99 {
100     if (pos < 0)
101     {
102         std::cout << "Position is not correct" << std::endl;
103         return;
104     }
105
106     Node* newNode = new Node(value);
107     if (pos == 0)
108     {
109         newNode->next = head;
110         head = newNode;
111         return;
112     }
113
114     Node* current = head;
115     for(int i = 0; i < pos - 1 && current != nullptr; i++)
116         current = current->next;
117
118     newNode->next = current->next;
119     current->next = newNode;
120 }
121
122 void DeleteElement(Node*& head, int pos)
123 {
124     if (pos < 0)
125     {
126         std::cout << "Position is not correct" << std::endl;
127         return;
128     }
129     if (head == nullptr)
130         return;
131     if (pos == 0)
132     {
133         Node* temp = head;
134         head = temp->next;
135         delete temp;
136         return;
137     }
138     Node* previous = head;
139     Node* temp = head;
140     for(int j = 0; j < pos; j++)
141     {
142         previous = temp;
143         temp = temp->next;
144         if (temp == nullptr)
145         {
146             std::cout << "Position out of range" << std::endl;
147             return;
148         }
149     }
150     previous->next = temp->next;
151     delete temp;
152 }
153
154 void SaveToFile(Node*& head, const char* filename)
155 {
156     std::ofstream file(filename);
157     if (!file)
158     {
159         perror("Error opening file");
160         exit(1);
161     }
162
163     Node* current = head;
164     while(current != nullptr)
165     {
166         file << current->data << " ";
167         current = current->next;
168     }
169     file.close();
170 }
171
172 void Deletelist(Node*& head)
173 {
174     while (head != nullptr)
175     {
176         Node* current = head;
177         head = head->next;
178         delete current;
179     }
180 }
181
182 void RestoreList(Node*& head, const char* filename)
183 {
184     std::ifstream file(filename);
185     if(!file)
186     {
187         perror("Error opening file");
188         exit(1);
189     }
190     int data;
191     int pos = 0;
192     while(file >> data)
193     {
194         AddElement(head, data, pos);
195         pos++;
196     }
197     file.close();
198 }

```

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

```

Chose an option:
1. Add elements
2. Delete elements
3. Show elements
4. Write to file
5. Delete list
6. Restor list
7. Exit
1
Enter the position where you want to add elements. Enter how many elements you want to add:
0
4
Enter the value of 1 element
1
Enter the value of 2 element
2
Enter the value of 3 element
3
Enter the value of 4 element
4
Chose an option:
1. Add elements
2. Delete elements
3. Show elements
4. Write to file
5. Delete list
6. Restor list
7. Exit
2
Enter the position where you want to delete elements. Enter how many elements you want to delete:
1
2

InkedList.txt
1 1 4
2
Enter the position where you want to delete elements. Enter how many elements you want to delete:
1
2
Chose an option:
1. Add elements
2. Delete elements
3. Show elements
4. Write to file
5. Delete list
6. Restor list
7. Exit
3
5. Delete list
6. Restor list
7. Exit
3
6. Restor list
7. Exit
3
3
1 4
Chose an option:
1. Add elements
2. Delete elements
3. Show elements
4. Write to file
5. Delete list
6. Restor list
7. Exit
4

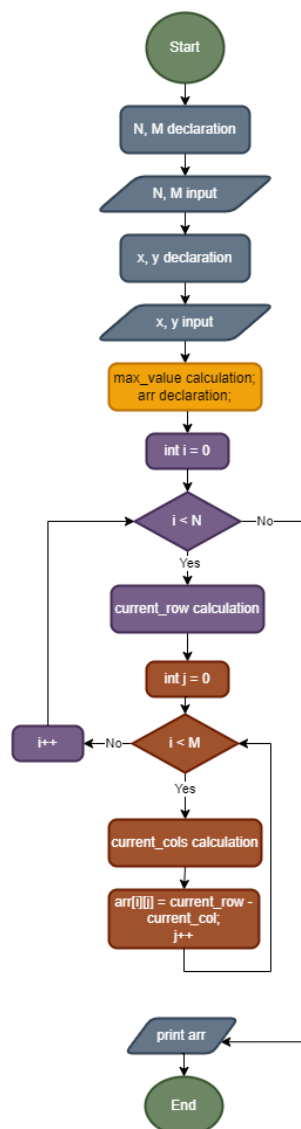
```

Час затрачений на виконання завдання: 4 год

Завдання №2 Algotester Lab 5V3

- *Варіант завдання: 3*
- [Деталі завдання](#)

1. Діаграма:



2. Код програми:

```
1  #include <iostream>
2  #include <algorithm>
3
4  int main()
5  {
6      int N, M;
7      std::cin >> N >> M;
8      int x, y;
9      std::cin >> x >> y;
10     int max_value = std::max(N - x, x - 1) + std::max(M - y, y - 1);
11
12     int arr[N][M];
13     for(int i = 0; i < N; i++)
14     {
15         int current_row = max_value - abs(i - x + 1);
16         for (int j = 0; j < M; j++)
17         {
18             int current_col = abs(j - y + 1);
19             arr[i][j] = current_row - current_col;
20         }
21     }
22
23     for(int i = 0; i < N; i++)
24     {
25         for(int j = 0; j < M; j++)
26             std::cout << arr[i][j] << " ";
27         std::cout << std::endl;
28     }
29
30     return 0;
31 }
32
33
```

3. Результат:

Created	Compiler	Result	Time (sec.)	Memory (MiB)	Actions
17 hours ago	C++ 23	Accepted	0.100	7.383	View

Завдання №3 Algotester Lab 7-8

- *Варіант завдання: 1*
- [Деталі завдання](#)

1. Код програми:

```

1  #include <iostream>
2  #include <vector>
3
4  template <typename T>
5  class DoubleLinkedList
6  {
7  private:
8      struct Node
9      {
10         T data;
11         Node* previous;
12         Node* next;
13         Node(T value, Node* prev = nullptr, Node* nxt = nullptr): data(value), previous(prev), next(nxt) {}
14     };
15
16     Node* head;
17     Node* tail;
18     int size;
19
20 public:
21     DoubleLinkedList();
22     ~DoubleLinkedList();
23     void Insert(int index, const std::vector<T>& lst);
24     void Erase(int index, int amount);
25     int Size();
26     T Get(int index);
27     void Set(int index, T value);
28     void Print();
29
30     friend std::ostream& operator<<(std::ostream& os, const DoubleLinkedList& list)
31     {
32         Node* current = list.head;
33         while (current != nullptr)
34         {
35             os << current->data << " ";
36             current = current->next;
37         }
38         return os;
39     }
40 };
41
42 template <typename T>
43 DoubleLinkedList<T>::~DoubleLinkedList(): head(nullptr), tail(nullptr), size(0) {}
44
45 template <typename T>
46 DoubleLinkedList<T>::~DoubleLinkedList()
47 {
48     while(head != nullptr)
49     {
50         Node* current = head;
51         head = head->next;
52         delete current;
53     }
54
55     tail = nullptr;
56     size = 0;
57 }
58
59 template <typename T>
60 void DoubleLinkedList<T>::Insert(int index, const std::vector<T>& lst)
61 {
62     if (index < 0 || index > size)
63         return;
64
65     for(int i = 0; i < lst.size(); i++)
66     {
67         Node* newNode = new Node(lst[i]);
68
69         if (index == 0)
70         {
71             newNode->next = head;
72             if(head != nullptr)
73                 head->previous = newNode;
74             head = newNode;
75             if (tail == nullptr)
76                 tail = newNode;
77         }
78         else if (index == size)
79         {
80             newNode->previous = tail;
81             if (tail != nullptr)
82                 tail->next = newNode;
83             tail = newNode;
84         }
85         else
86         {
87             Node* current = head;
88             for(int i = 0; i < index - 1; i++)
89                 current = current->next;
90             newNode->next = current->next;
91             newNode->previous = current;
92             if (current->next != nullptr)
93                 current->next->previous = newNode;
94             current->next = newNode;
95
96             index++;
97             size++;
98         }
99     }
100 }
101
102 template <typename T>
103 void DoubleLinkedList<T>::Print()
104 {
105     std::cout << *this << std::endl;
106 }
107
108 template <typename T>
109 void DoubleLinkedList<T>::Erase(int index, int amount)
110 {
111     if (index < 0 || index > size || amount <= 0 || index + amount > size)
112         return;
113
114     Node* current = head;
115     for (int i = 0; i < index; i++)
116         current = current->next;
117
118     for(int i = 0; i < amount; i++)
119     {
120         Node* nodeToDelete = current;
121         current = current->next;
122
123         if (nodeToDelete->previous != nullptr)
124             nodeToDelete->previous->next = nodeToDelete->next;
125         else
126             head = nodeToDelete->next;
127         if (nodeToDelete->next != nullptr)
128             nodeToDelete->next->previous = nodeToDelete->previous;
129         else
130             tail = nodeToDelete->previous;
131
132         delete nodeToDelete;
133         size--;
134     }
135 }
136
137 template <typename T>
138 int DoubleLinkedList<T>::Size()
139 {
140     return size;
141 }
142
143 template <typename T>
144 T DoubleLinkedList<T>::Get(int index)
145 {
146     Node* current = head;
147     for(int i = 0; i < index; i++)
148         current = current->next;
149
150     return current->data;
151 }
152
153 template <typename T>
154 void DoubleLinkedList<T>::Set(int index, T value)
155 {
156     std::cin >> index >> value;
157     myList.Set(index, value);
158 }
159
160 if(question == "print")
161 {
162     myList.Print();
163 }
164
165 return 0;
166 }
167
168 int main()
169 {
170     DoubleLinkedList<int> myList;
171     int Q;
172     std::cin >> Q;
173     std::string question;
174
175     for(int i = 0; i < Q; i++)
176     {
177         std::cin >> question;
178
179         if(question == "insert")
180         {
181             int index, N;
182             std::cin >> index >> N;
183             std::vector<int> lst(N);
184             for(int i = 0; i < N; i++)
185                 std::cin >> lst[i];
186             myList.Insert(index, lst);
187         }
188         if(question == "erase")
189         {
190             int index, N;
191             std::cin >> index >> N;
192             myList.Erase(index, N);
193         }
194         if(question == "size")
195         {
196             std::cout << myList.Size() << std::endl;
197         }
198         if(question == "get")
199         {
200             int index;
201             std::cin >> index;
202             std::cout << myList.Get(index) << std::endl;
203         }
204         if(question == "set")
205         {
206             int index, value;
207             std::cin >> index >> value;
208             myList.Set(index, value);
209         }
210     }
211 }

```

2. Результат:

4 days ago	C++ 23	Accepted	0.008	1.344	View
------------	--------	----------	-------	-------	----------------------

Завдання №5 Class Practice Work (1-3)

- *Варіант завдання:* -
- *Деталі завдання:*
 1. Реалізувати метод реверсу списку: `Node* reverse(Node *head);`
 2. Порівняти два списки
 3. Додавання великих чисел (додати два списки)

1. Код програми:

```
practice_work_task_1-3_olesia_kostak.cpp > main()
1 #include <iostream>
2
3 class LinkedList
4 {
5 private:
6     struct Node
7     {
8         int data;
9         Node* next;
10
11         Node(): data(0), next(nullptr) {}
12         Node(int val, Node* node = nullptr) : data(val), next(node) {}
13     };
14     Node* head;
15     int size;
16
17 public:
18     LinkedList();
19     ~LinkedList();
20     void PushBack(int data);
21     void ShowList();
22     void ReverseList();
23     bool CompareTwoLists(LinkedList& list2);
24     void AddElementsOfTwoLists(LinkedList& list2);
25 };
26
27 LinkedList::LinkedList(): head(nullptr), size(0) {}
28 LinkedList::~LinkedList()
29 {
30     while(head != nullptr)
31     {
32         Node* current = head;
33         head = head->next;
34         delete current;
35     }
36 }
37 void LinkedList::ShowList()
38 {
39     Node* current = head;
40     while(current != nullptr)
41     {
42         std::cout << current->data << " ";
43         current = current->next;
44     }
45     std::cout << std::endl;
46 }
47 void LinkedList::PushBack(int data)
48 {
49     Node* newNode = new Node(data);
50     if(head == nullptr)
51     {
52         head = newNode;
53         return;
54     }
55     Node* current = head;
56     while(current->next != nullptr)
57         current = current->next;
58     current->next = newNode;
59     size++;
60 }
61
62 void LinkedList::ReverseList()
63 {
64     Node* previous = nullptr;
65     Node* current = head;
66     Node* nextNode;
67
68     while(current != nullptr)
69     {
70         nextNode = current->next;
71         current->next = previous;
72         previous = current;
73         current = nextNode;
74     }
75     head = previous;
76 }
77 bool LinkedList::CompareTwoLists(LinkedList& list2)
78 {
79     Node* current1 = head;
80     Node* current2 = list2.head;
81
82     while(current1 != nullptr && current2 != nullptr)
83     {
84         if(current1->data > current2->data)
85             return 0;
```



```

85         return 0;
86         if(current2->data > current1->data)
87             return 0;
88         current1 = current1->next;
89         current2 = current2->next;
90     }
91     if((!current1) && (!current2))
92         return 1;
93     else
94         return 0;
95 }
96 void LinkedList::AddElementsOfTwoLists(LinkedList& list2)
97 {
98     this->Reverselist();
99     list2.Reverselist();
100     Node* current1 = head;
101     Node* current2 = list2.head;
102     long sum = 0;
103     int r = 0;
104
105     LinkedList res;
106     while (current1 != nullptr || current2 != nullptr || r != 0)
107     {
108         sum = r;
109         if(current1 != nullptr)
110         {
111             sum += current1->data;
112             current1 = current1->next;
113         }
114         if(current2 != nullptr)
115         {
116             sum += current2->data;
117             current2 = current2->next;
118         }
119         res.PushBack(sum % 10);
120         r = sum / 10;
121     }
122     res.Reverselist();
123     res.ShowList();
124
125     this->Reverselist();
126     list2.Reverselist();
127 }
128 int main()
129 {
130     LinkedList list1;
131     list1.PushBack(8);
132     list1.PushBack(9);
133     list1.PushBack(7);
134     list1.PushBack(1);
135
136     std::cout << "A list before reversing: "; list1.ShowList();
137     list1.Reverselist();
138     std::cout << "A list after reversing: "; list1.ShowList(); std::cout << std::endl;
139     list1.Reverselist();
140
141     LinkedList list2;
142     list2.PushBack(8);
143     list2.PushBack(9);
144     list2.PushBack(7);
145
146     std::cout << "Two lists to compare and add: " << std::endl;
147     list1.ShowList(); list2.ShowList();
148     std::cout << "Are these lists equal: ";
149     std::cout << (list1.CompareTwoLists(list2)? "Yes" : "No") << std::endl << std::endl;
150     std::cout << "Sum of two lists: ";
151     list1.AddElementsOfTwoLists(list2);
152     return 0;
153 }
154

```

2. Результат:

```

A list before reversing: 8 9 7 1
A list after reversing: 1 7 9 8

```

```

Two lists to compare and add:
8 9 7 1
8 9 7
Are these lists equal: No

Sum of two lists: 9 8 6 8

```

Завдання №5 Class Practice Work (4-5)

- *Варіант завдання:* -
- *Деталі завдання:*
 1. Віддзеркалити дерево.
 2. Записати кожному батьківському вузлу суму підвузлів.

1. Код програми:

```

G practice_work_task_4-5_olesia_kostak.cpp > ...
1  #include <iostream>
2
3  struct Node
4  {
5      int data;
6      Node* left;
7      Node* right;
8      Node(int value): data(value), left(nullptr), right(nullptr) {}
9  };
10
11 void Insert(Node*& root, int value);
12 void Show(Node* root);
13 void FreeTree(Node*& root);
14 Node* MirrorTree(Node*& root);
15 void TreeSum(Node*& root);
16
17
18 int main()
19 {
20     Node* myTree = nullptr;
21     Insert(myTree, 50);
22     Insert(myTree, 40);
23     Insert(myTree, 60);
24     Insert(myTree, 30);
25     Insert(myTree, 70);
26     Insert(myTree, 80);
27     Insert(myTree, 20);
28
29     std::cout << "Tree: ";
30     Show(myTree);
31     std::cout << std::endl;
32     std::cout << "Mirrored Tree: ";
33     MirrorTree(myTree);
34     Show(myTree);
35     MirrorTree(myTree);
36     std::cout << std::endl << "Tree of sum: ";
37     TreeSum(myTree);
38     Show(myTree);
39     FreeTree(myTree);
40
41     return 0;
42 }
43
44 void Insert(Node*& root, int value)
45 {
46     if (root == nullptr)
47     {
48         root = new Node(value);
49         return;
50     }
51     if (root->data == value)
52         return;
53
54     if (root->data > value)
55     {
56         if (root->left == nullptr)
57         {
58             root->left = new Node(value);
59             return;
60         }
61         Insert(root->left, value);
62     }
63     if (root->data < value)
64     {
65         if (root->right == nullptr)
66         {
67             root->right = new Node(value);
68             return;
69         }
70         Insert(root->right, value);
71     }
72 }
73
74 void Show(Node* root)
75 {
76     if (root != nullptr)
77     {
78         Show(root->left);
79         std::cout << root->data << " ";
80         Show(root->right);
81     }
82 }
83
84 void FreeTree(Node*& root)
85 {
86     if (root != nullptr)
87     {
88         FreeTree(root->left);
89         FreeTree(root->right);
90         delete root;
91     }
92 }
93
94 Node* MirrorTree(Node*& root)
95 {
96     if (root == nullptr)
97         return nullptr;
98     Node* left = MirrorTree(root->left);
99     Node* right = MirrorTree(root->right);
100
101     root->right = left;
102     root->left = right;
103     return root;
104
105
106 void TreeSum(Node*& root)
107 {
108     if (root == nullptr)
109         return;
110
111     TreeSum(root->left);
112     TreeSum(root->right);
113
114     if (root->left != nullptr)
115         root->data += root->left->data;
116
117     if (root->right != nullptr)
118         root->data += root->right->data;
119 }

```

2. 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

```

Завдання №6 Self Practice Work (Lab 78v3)

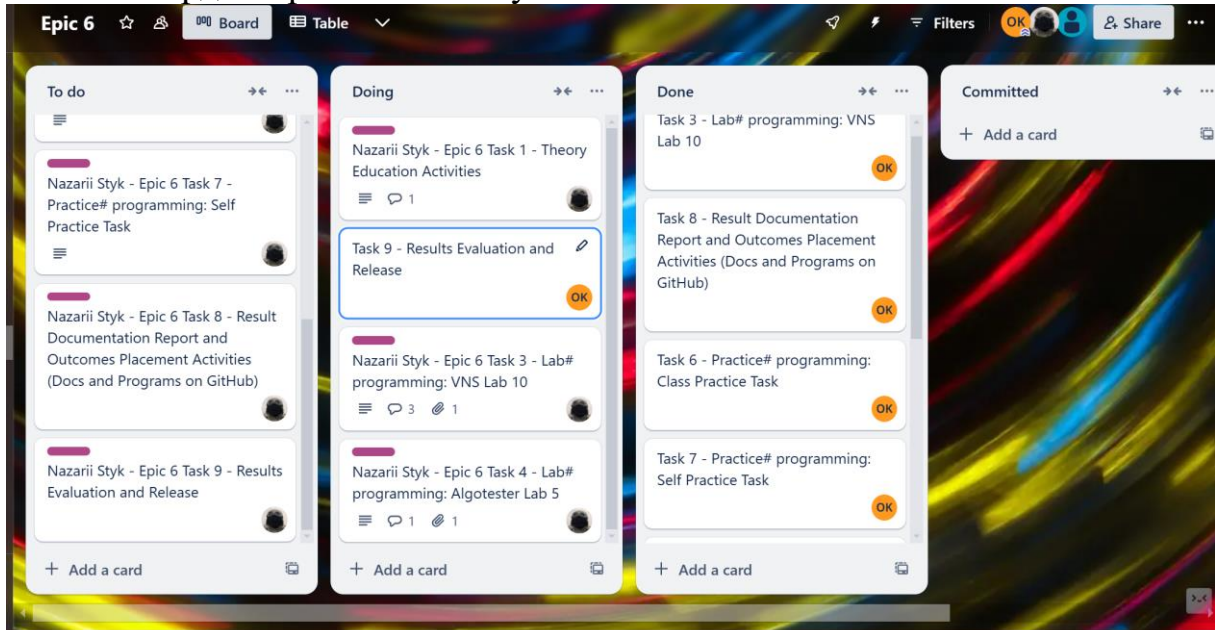
1. Код програми:

```
1 #include <iostream>
2
3 template <typename T>
4 class BinaryTree
5 {
6 private:
7     struct TreeNode
8     {
9         T data;
10        TreeNode* left;
11        TreeNode* right;
12        TreeNode(T value): data(value), left(nullptr), right(nullptr) {};
13    };
14    TreeNode* root;
15    int size;
16    void Insert(TreeNode*& root, T value);
17    void Print(const TreeNode* root) const;
18
19 public:
20    BinaryTree();
21    ~BinaryTree();
22    void Insert(T value);
23    bool Contains(T value) const;
24    int Size() const;
25    void Print() const;
26    void FreeTree(TreeNode* root);
27 };
28
29 template <typename T>
30 BinaryTree<T>::BinaryTree(): root(nullptr), size(0) {}
31
32 template <typename T>
33 BinaryTree<T>::~BinaryTree()
34 {
35     FreeTree(root);
36     size = 0;
37 }
38
39 template <typename T>
40 void BinaryTree<T>::FreeTree(TreeNode* root)
41 {
42     if (root != nullptr)
43     {
44         FreeTree(root->left);
45         FreeTree(root->right);
46         delete root;
47         size = 0;
48     }
49 }
50
51 template <typename T>
52 void BinaryTree<T>::Insert(TreeNode*& root, T value)
53 {
54
55 }
56
57 template <typename T>
58 bool BinaryTree<T>::Contains(T target) const
59 {
60     TreeNode* current = root;
61     while (current != nullptr)
62     {
63         if (current->data == target)
64             return true;
65         if (target > current->data)
66             current = current->right;
67         else
68             current = current->left;
69     }
70     return false;
71 }
72
73 template <typename T>
74 int BinaryTree<T>::Size() const
75 {
76     return size;
77 }
78
79 int main()
80 {
81     int Q;
82     std::cin >> Q;
83     std::string question;
84     BinaryTree<int> myTree;
85
86     for(int i = 0; i < Q; i++)
87     {
88         std::cin >> question;
89         if (question == "insert")
90         {
91             int value;
92             std::cin >> value;
93             myTree.Insert(value);
94         }
95         if (question == "contains")
96         {
97             int target;
98             std::cin >> target;
99             std::cout << (myTree.Contains(target) ? "Yes" : "No") << std::endl;
100         }
101         if (question == "size")
102         {
103             std::cout << myTree.Size() << std::endl;
104         }
105         if (question == "print")
106         {
107
108         }
109     }
110 }
111
112 if (root == nullptr)
113 {
114     root = new TreeNode(value);
115     size++;
116     return;
117 }
118 if (root->data == value)
119     return;
120 if (value < root->data)
121 {
122     if (root->left == nullptr)
123     {
124         root->left = new TreeNode(value);
125         size++;
126         return;
127     }
128     Insert(root->left, value);
129 }
130 if (value > root->data)
131 {
132     if (root->right == nullptr)
133     {
134         root->right = new TreeNode(value);
135         size++;
136         return;
137     }
138     Insert(root->right, value);
139 }
140
141 template <typename T>
142 void BinaryTree<T>::Insert(T value)
143 {
144     Insert(root, value);
145 }
146
147 template <typename T>
148 void BinaryTree<T>::Print(const TreeNode* root) const
149 {
150     if (root != nullptr)
151     {
152         Print(root->left);
153         std::cout << root->data << " ";
154         Print(root->right);
155     }
156 }
157
158 template <typename T>
159 void BinaryTree<T>::Print() const
160 {
161     Print(root);
162 }
163
164 myTree.Print();
165 std::cout << std::endl;
166
167 }
168
169 return 0;
170
171 }
```

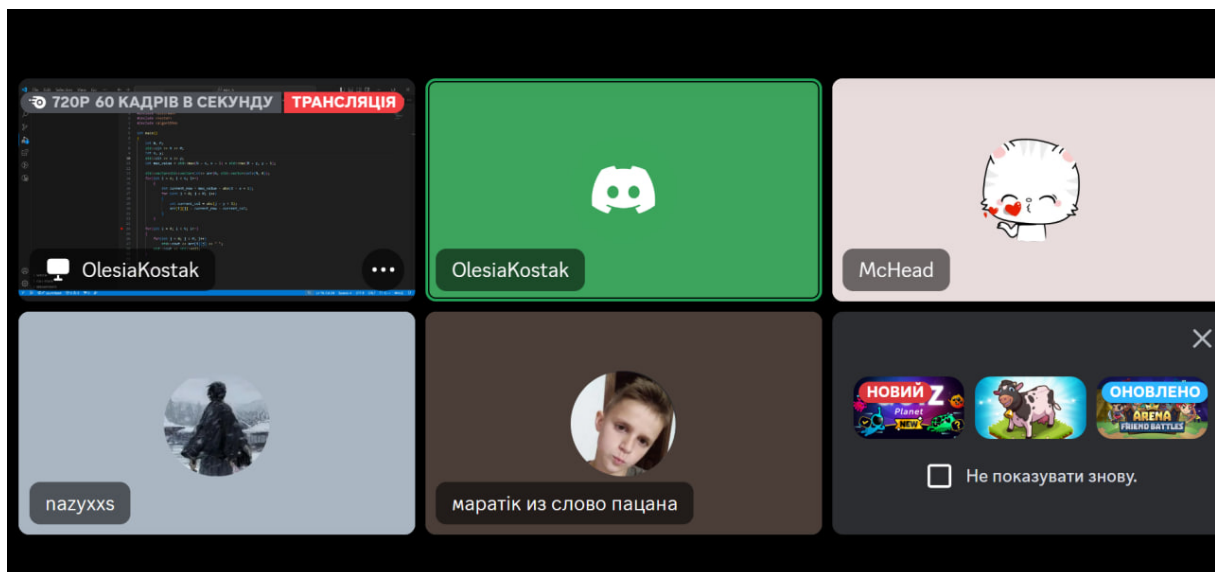
2. Результат:

3 days ago	C++ 23	Accepted	0.008	1.422	View
------------	--------	----------	-------	-------	----------------------

Кооперація з командою: Спільна борда в трелло по епіку:



Балакаємо про епіки:



Висновки:

По завершенню завдань Епіка №6, я освоїла роботу з динамічними структурами даних, такими як черга, стек, списки та дерева. Навчилася реалізовувати базові операції над цими структурами, включаючи створення, додавання, видалення, пошук та обхід елементів. Розвинула навички алгоритмічної обробки динамічних структур для ефективного вирішення прикладних задач.