

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 Лабораторної Роботи № 5

Алготестер Лабораторної Роботи № 7-8

Практичних Робіт до блоку № 6

**Виконав:**

Студент групи ШІ-12

Сирватка Олександр Ігорович

Львів 2024

**Тема роботи:** Динамічні структури (Черга, Стек, Списки, Дерево). Алгоритми обробки динамічних структур.

**Мета роботи:** навчитись працювати з динамічними структурами, спробувати написати власні алгоритми для таких структур як: черга, стек, список та дерево.

### **Теоретичні відомості:**

- 1) Структури
- 2) Класи
- 3) Список
- 4) Подвійний список
- 5) Бінарне дерево

### **Індивідуальний план опрацювання теорії:**

- Тема №1 Структури (50 хв)  
([https://www.youtube.com/watch?v=999IE-6b7\\_s](https://www.youtube.com/watch?v=999IE-6b7_s))
- Тема №2 Класи (50 хв)  
([https://www.youtube.com/watch?v=ZbsukxxV5\\_Q&list=PLiPRE8VmJzOpn6PzYf0higmCEyGzo2A5g&index=95](https://www.youtube.com/watch?v=ZbsukxxV5_Q&list=PLiPRE8VmJzOpn6PzYf0higmCEyGzo2A5g&index=95))
- Тема №3 Список (70 хв)  
([https://www.youtube.com/watch?v=-25REjF\\_atI&list=PLiPRE8VmJzOpn6PzYf0higmCEyGzo2A5g&index=141](https://www.youtube.com/watch?v=-25REjF_atI&list=PLiPRE8VmJzOpn6PzYf0higmCEyGzo2A5g&index=141))
- Тема №4 Двобічний список (40 хв)  
([https://www.youtube.com/watch?v=QLzu2\\_QFoE&list=PLiPRE8VmJzOpn6PzYf0higmCEyGzo2A5g](https://www.youtube.com/watch?v=QLzu2_QFoE&list=PLiPRE8VmJzOpn6PzYf0higmCEyGzo2A5g))
- Тема №5 Бінарне дерево (50 хв)  
(<https://www.youtube.com/watch?v=qBFzNW0ALxQ&list=PLiPRE8VmJzOpn6PzYf0higmCEyGzo2A5g>)

# Виконання роботи

## Завдання №1 Epic 6 Task 3 - Lab# programming: VNS Lab 10

```
1 #include <iostream>
2 #include <fstream>
3
4 using namespace std;
5
6 struct Node {
7     int key;
8     Node* next;
9 };
10
11 // Функція для створення порожнього списку
12 Node* createlist() {
13     return nullptr; // Створемо порожній список
14 }
15
16 // Функція для друку списку
17 void printList(Node* head) {
18     if (!head) {
19         cout << "Список порожній" << endl;
20         return;
21     }
22     Node* temp = head;
23     while (temp) {
24         cout << temp->key << " ";
25         temp = temp->next;
26     }
27     cout << endl;
28 }
29
30 // Функція для додавання елемента в список
31 void addElement(Node*& head, int value, int position) {
32     Node* newNode = new Node(value, nullptr); // Створемо новий вузол
33     if (position == 0 || !head) { // Додавання на початок списку
34         newNode->next = head; // Новий вузол вказує на поточну голову
35         head = newNode; // Оновлюємо голову списку
36         return;
37     }
38
39     Node* temp = head; // Починаємо з голови
40     for (int i = 0; i < position - 1 && temp->next; ++i) {
41         temp = temp->next; // Переходимо до вузла перед позицією вставки
42     }
43     newNode->next = temp->next; // Новий вузол вказує на наступний вузол
44     temp->next = newNode; // Поточний вузол вказує на новий вузол
45 }
46
47 // Функція для видалення елемента зі списку
48 void deleteElement(Node*& head, int position) {
49     if (!head) return;
50
51     Node* temp = head;
52     if (position == 0) {
53         head = head->next;
54         delete temp;
55         return;
56     }
57
58     for (int i = 0; temp && i < position - 1; ++i) {
59         temp = temp->next;
60     }
61     if (!temp || !temp->next) return;
62
63     Node* next = temp->next->next;
```

```
61     if (!temp || !temp->next) return;
62
63     Node* next = temp->next->next;
64     delete temp->next;
65     temp->next = next;
66 }
67
68 // Функція для запису списку у файл
69 void writeToFile(Node* head, const string& filename) {
70     ofstream file(filename);
71
72     // Перевірка, чи вдалося відкрити файл
73     if (!file.is_open()) {
74         cerr << "Не вдалося відкрити файл для запису: " << filename << endl;
75         return;
76     }
77
78     Node* temp = head;
79     while (temp) {
80         file << temp->key << endl;
81         temp = temp->next;
82     }
83     file.close(); // Закриваємо файл після запису
84 }
85
86 // Функція для відновлення списку з файлу
87 Node* readFromFile(const string& filename) {
88     ifstream file(filename);
89
90     // Перевірка, чи вдалося відкрити файл
91     if (!file.is_open()) {
92         cerr << "Не вдалося відкрити файл: " << filename << endl;
93         return nullptr;
94     }
95
96     Node* head = nullptr;
97     Node* tail = nullptr;
98     int value;
99
100     while (file >> value) {
101         Node* newNode = new Node(value, nullptr);
102
103         if (!head) {
104             head = tail = newNode; // Якщо список порожній, новий вузол стає головою
105         } else {
106             tail->next = newNode; // Додаємо новий вузол в кінець списку
107             tail = newNode; // Оновлюємо tail
108         }
109     }
110
111     file.close(); // Закриваємо файл після зчитування
112     return head; // Повертаємо голову списку
113 }
114
115 // Функція для знищення списку
116 void deleteList(Node*& head) {
117     while (head) {
118         Node* temp = head;
119         head = head->next;
120         delete temp;
121     }
122 }
```

```
124 int main() {
125     Node* list = createlist();
126
127     addElement(list, 10, 0); // Додавання елементів
128     addElement(list, 20, 1);
129     addElement(list, 30, 2);
130     printList(list);
131
132     deleteElement(list, 1); // Видалення елемента
133     printList(list);
134
135     writeToFile(list, "list.txt"); // Запис у файл
136     deleteList(list);
137     printList(list); // Список порожній
138
139     list = readFromFile("list.txt"); // Відновлення з файлу
140     printList(list);
141
142     deleteList(list); // Очищення пам'яті
143     return 0;
144 }
145
```

## Завдання №2 Epic 6 Task 4 - Lab# programming: Algotester Lab 5

```
1  #include <iostream>
2  #include <vector>
3  #include <string>
4
5  using namespace std;
6
7  int main() {
8      int N, M;
9      cin >> N >> M;
10
11     vector<string> cave(N);
12
13
14     for (int i = 0; i < N; ++i) {
15         cin >> cave[i];
16     }
17
18
19     for (int j = 0; j < M; ++j) {
20         int bottom = N;
21
22         // Обробляємо стовпець зверху вниз
23         for (int i = N - 1; i >= 0; --i) {
24             if (cave[i][j] == 'X') {
25                 bottom = i;
26             } else if (cave[i][j] == 'S') {
27
28                 cave[i][j] = '0';
29                 cave[--bottom][j] = 'S';
30             }
31         }
32     }
33
34
35     for (int i = 0; i < N; ++i) {
36         cout << cave[i] << endl;
37     }
38
39     return 0;
40 }
41
```

## Завдання №3 Epic 6 Task 5 - Lab# programming: Algotester Lab 7-8 v1

```
1 #include <iostream>
2
3 using namespace std;
4
5 class Node {
6 public:
7     int data;
8     Node *next;
9     Node *prev;
10
11     Node(int value) {
12         data = value;
13         next = nullptr;
14         prev = nullptr;
15     }
16 };
17
18 class LinkedList {
19 public:
20     LinkedList() {
21         size = 0;
22         head = nullptr;
23         tail = nullptr;
24     }
25
26     ~LinkedList() {
27         Node *current = head;
28         while (current) {
29             Node *nextNode = current->next;
30             delete current;
31             current = nextNode;
32         }
33     }
34
35     void insert(int index, int listSize, int values[]);
36
37     void erase(int index, int count);
38
39     int get(int index);
40
41     void set(int index, int value);
42
43     int getSize();
44
45     void print();
46
47 private:
48     Node *head;
49     Node *tail;
50     int size;
51 };
52
53 void LinkedList::insert(int index, int listSize, int values[]) {
54     if (index < 0 || index > size || listSize <= 0) return;
55
56     Node *current = head;
57     Node *prev = nullptr;
58
59     for (int i = 0; i < index; i++) {
60         prev = current;
61         current = current->next;
62     }
63
120 }
121
122 Node *current = head;
123
124 // find node at index
125 for (int i = 0; i < index; i++) {
126     current = current->next;
127 }
128
129 return current->data;
130 }
131
132 void LinkedList::set(int index, int value) {
133     if (index < 0 || index >= size) {
134         return;
135     }
136
137     Node *current = head;
138
139     // find node at index
140     for (int i = 0; i < index; i++) {
141         current = current->next;
142     }
143
144     current->data = value;
145 }
146
147 int LinkedList::getSize() {
148     return size;
149 }
150
151 void LinkedList::print() {
152     Node *current = head;
153     while (current) {
154         cout << current->data << " ";
155         current = current->next;
156     }
157     cout << endl;
158 }
159
160 void Choice(LinkedList &list, const string &choice) {
161     if (choice == "insert") {
162         int index, n;
163         cin >> index >> n;
164         int elements[n];
165         for (int i = 0; i < n; i++) {
166             cin >> elements[i];
167         }
168         list.insert(index, n, elements);
169     } else if (choice == "erase") {
170         int index, count;
171         cin >> index >> count;
172         list.erase(index, count);
173     } else if (choice == "get") {
174         int index;
175         cin >> index;
176         cout << list.get(index) << endl;
177     } else if (choice == "set") {
178         int index;
179         int value;
180         cin >> index >> value;
181         list.set(index, value);
182     } else if (choice == "size") {
```

```

61     current = current->next;
62 }
63
64 for (int j = listSize - 1; j >= 0; j--) {
65     Node *newNode = new Node(values[j]);
66     newNode->next = current;
67     if (current) {
68         current->prev = newNode;
69     } else {
70         tail = newNode;
71     }
72     current = newNode;
73     size++;
74 }
75
76 if (prev) {
77     prev->next = current;
78     current->prev = prev;
79 } else {
80     head = current;
81 }
82 if (!current->next) {
83     tail = current;
84 }
85 }
86
87 void LinkedList::erase(int index, int count) {
88     if (index < 0 || index >= size || count <= 0) return;
89
90     Node *current = head;
91     Node *prev = nullptr;
92
93     for (int i = 0; i < index; i++) {
94         prev = current;
95         current = current->next;
96     }
97
98     for (int j = 0; j < count && current; j++) {
99         Node *nextNode = current->next;
100         if (nextNode) {
101             nextNode->prev = current->prev;
102         } else {
103             tail = prev;
104         }
105         delete current;
106         current = nextNode;
107         size--;
108     }
109
110     if (prev) {
111         prev->next = current;
112     } else {
113         head = current;
114     }
115 }
116
117 int LinkedList::get(int index) {
118     if (index < 0 || index >= size) {
119         return 0;
120     }
121
122     Node *current = head;
123
124     while (current && index > 0) {
125         current = current->next;
126         index--;
127     }
128     return current->value;
129 }
130
131 void LinkedList::print() const {
132     if (!head) {
133         cout << "Empty list" << endl;
134         return;
135     }
136     Node *current = head;
137     while (current) {
138         cout << current->value << " ";
139         current = current->next;
140     }
141     cout << endl;
142 }
143
144 int main() {
145     LinkedList list;
146
147     int q;
148     cin >> q;
149     string choice;
150     for (int i = 0; i < q; i++) {
151         cin >> choice;
152         Choice(list, choice);
153     }
154
155     list.print();
156
157     return 0;
158 }

```

Завдання №3 Epic 6 Task 5 - Lab# programming: Algotester Lab 7-8 v2

```

1  #include <iostream>
2
3  using namespace std;
4
5  template<typename T>
6  class Node {
7  public:
8      T data;
9      Node<T> *next;
10     Node<T> *prev;
11
12     Node(T value) {
13         data = value;
14         next = nullptr;
15         prev = nullptr;
16     }
17 };
18
19 template<typename T>
20 class LinkedList {
21 public:
22     LinkedList() {
23         size = 0;
24         head = nullptr;
25         tail = nullptr;
26     }
27     ~LinkedList() {
28         Node<T> *current = head;
29         while (current) {
30             Node<T> *nextNode = current->next;
31             delete current;
32             current = nextNode;
33         }
34     }
35
36     void insert(int index, int listSize, T values[]);
37     void erase(int index, int count);
38
39     int get(int index);
40     void set(int index, T value);
41
42     int getSize();
43     void print();
44
45 private:
46     Node<T> *head;
47     Node<T> *tail;
48     int size;
49 };
50
51 template<class T>
52 void LinkedList<T>::insert(int index, int listSize, T values[]) {
53     if (index < 0 || index > size || listSize <= 0) return;
54
55     Node<T> *current = head;
56     Node<T> *prev = nullptr;
57
58     for (int i = 0; i < index; i++) {
59         prev = current;
60         current = current->next;
61     }
62
63     for (int i = 0; i < index; i++) {
64         prev = current;
65         current = current->next;
66     }
67
68     Node<T> *newNode = new Node<T>(values[j]);
69     newNode->next = current;
70     if (current) {
71         current->prev = newNode;
72     } else {
73         tail = newNode;
74     }
75     current = newNode;
76     size++;
77 }
78
79 if (prev) {
80     prev->next = current;
81     current->prev = prev;
82 } else {
83     head = current;
84 }
85 if (!current->next) {
86     tail = current;
87 }
88 }
89
90 template<class T>
91 void LinkedList<T>::erase(int index, int count) {
92     if (index < 0 || index >= size || count <= 0) return;
93
94     Node<T> *current = head;
95     Node<T> *prev = nullptr;
96
97     for (int i = 0; i < index; i++) {
98         prev = current;
99         current = current->next;
100     }
101
102     for (int j = 0; j < count && current; j++) {
103         Node<T> *nextNode = current->next;
104         if (nextNode) {
105             nextNode->prev = current->prev;
106         } else {
107             tail = prev;
108         }
109         delete current;
110         current = nextNode;
111         size--;
112     }
113
114     if (prev) {
115         prev->next = current;
116     } else {
117         head = current;
118     }
119 }
120

```



```

124 template<class T>
125 int LinkedList<T>::get(int index) {
126     if (index < 0 || index >= size) {
127         return 0;
128     }
129
130     Node<T> *current = head;
131
132     for (int i = 0; i < index; i++) {
133         current = current->next;
134     }
135
136     return current->data;
137 }
138
139 template<class T>
140 void LinkedList<T>::set(int index, T value) {
141     if (index < 0 || index >= size) {
142         return;
143     }
144
145     Node<T> *current = head;
146
147     for (int i = 0; i < index; i++) {
148         current = current->next;
149     }
150     current->data = value;
151 }
152
153 template<class T>
154 int LinkedList<T>::getSize() {
155     return size;
156 }
157
158 template<class T>
159 void LinkedList<T>::print() {
160     Node<T> *current = head;
161     while (current) {
162         cout << current->data << " ";
163         current = current->next;
164     }
165     cout << endl;
166 }
167
168 template<typename T>
169 void Choice(LinkedList<T> &list, const string &choice) {
170     if (choice == "insert") {
171         int index, n;
172         cin >> index >> n;
173         T elements[n];
174         for (int i = 0; i < n; i++) {
175             cin >> elements[i];
176         }
177         list.insert(index, n, elements);
178     } else if (choice == "erase") {
179         int index, count;
180         cin >> index >> count;
181         list.erase(index, count);
182     } else if (choice == "get") {
183         int index;
184         cin >> index;
185         cout << list.get(index) << endl;
186     } else if (choice == "set") {
187         int index;
188         T value;
189         cin >> index >> value;
190         list.set(index, value);
191     } else if (choice == "size") {
192         cout << list.getSize() << endl;
193     } else {
194         list.print();
195     }
196 }
197
198 int main() {
199     LinkedList<int> list;
200
201     int q;
202     cin >> q;
203
204     string choice;
205     for (int i = 0; i < q; i++) {
206         cin >> choice;
207         Choice(list, choice);
208     }
209
210     return 0;
211 }

```



**Завдання №4** Epic 6 Task 6 - Practice# programming: Class Practice Task

```

1  #include<iostream>
2  #include<string>
3  using namespace std;
4
5  //Linked list
6  struct Node
7  {
8      int data;
9      Node* next;
10 };
11
12 //Додавання елемента в кінець
13 Node* push_back(Node* head, const int& element)
14 {
15     Node* new_node = new Node;
16     new_node->data = element;
17     new_node->next = nullptr;
18
19     if (head == nullptr)
20     {
21         head = new_node;
22     }
23     else
24     {
25         Node* current = head;
26         while (current->next != nullptr)
27         {
28             current = current->next;
29         }
30
31         current->next = new_node;
32     }
33
34     return head;
35 }
36

```

```

37 //Виведення списку
38 void print_list(Node* head)
39 {
40     if (head == nullptr)
41     {
42         cout<<"List is empty"<<endl;
43     }
44     else
45     {
46         Node* current = head;
47         while (current != nullptr)
48         {
49             cout<<current->data<<" ";
50             current = current->next;
51         }
52         cout<<endl;
53     }
54 }
55
56 //Реверс списку
57 Node* reverse_list(Node* head)
58 {
59     if (head == nullptr)
60     {
61         cout<<"List is empty"<<endl;
62         return 0;
63     }
64     else
65     {
66         Node* prev = nullptr;
67         Node* current = head;
68         Node* next = nullptr;
69
70         while (current != nullptr)
71         {
72             next = current->next;
73             current->next = prev;
74             prev = current;
75             current = next;
76         }
77
78         return prev;
79     }
80 }
81

```

```

82 //Порівняння на рівність двох списків
83 bool compare(Node* head_1, Node* head_2)
84 {
85     Node* current_1 = head_1;
86     Node* current_2 = head_2;
87
88     while ((current_1 != nullptr) && (current_2 != nullptr))
89     {
90         if (current_1->data != current_2->data)
91         {
92             return false;
93         }
94         current_1 = current_1->next;
95         current_2 = current_2->next;
96     }
97
98     if ((current_1 != nullptr) || (current_2 != nullptr))
99     {
100         return false;
101     }
102     else return true;
103 }
104
105 //Додавання двох великих чисел
106 Node* add_two_numbers(Node* n1, Node* n2)
107 {
108     Node* current_1 = n1;
109     Node* current_2 = n2;
110     Node* sum = nullptr;
111     int r = 0;
112     int s = 0;
113     while (current_2 != nullptr)
114     {
115         s = current_1->data + current_2->data + r;
116
117         if (s > 9)
118         {
119             sum = push_back(sum, s % 10);
120             r = s / 10;
121         }
122         else
123         {
124             sum = push_back(sum, s);
125             r = 0;
126         }
127
128         current_1 = current_1->next;
129         current_2 = current_2->next;
130     }
131 }

```

```

132 if (current_1 != nullptr)
133 {
134     while (current_1 != nullptr)
135     {
136         s = current_1->data + r;
137
138         if (s > 9)
139         {
140             sum = push_back(sum, s % 10);
141             r = s / 10;
142         }
143         else
144         {
145             sum = push_back(sum, s);
146             r = 0;
147         }
148
149         current_1 = current_1->next;
150     }
151 }
152 else if (r != 0)
153 {
154     sum = push_back(sum, r);
155 }
156
157 return sum;
158 }
159
160 //Виведення числа
161 void print_number(Node* head)
162 {
163     if (head == nullptr)
164     {
165         cout<<"List is empty"<<endl;
166     }
167     else
168     {
169         Node* current = head;
170         while (current != nullptr)
171         {
172             cout<<current->data;
173             current = current->next;
174         }
175         cout<<endl;
176     }
177 }
178 //Linked list
179

```

```

180 //Binary tree
181 struct tree_node
182 {
183     int data;
184     tree_node* left;
185     tree_node* right;
186
187     tree_node(int value) : data(value), left(nullptr), right(nullptr){}
188 };
189
190
191
192 //Функція для вставлення елемента в дерево
193 tree_node* insert(tree_node* node, int value)
194 {
195     if (node == nullptr)
196     {
197         return new tree_node(value);
198     }
199     else
200     {
201         if (value < node->data)
202         {
203             node->left = insert(node->left, value);
204         }
205         else if (value > node->data)
206         {
207             node->right = insert(node->right, value);
208         }
209         return node;
210     }
211 }
212
213
214
215 //Віддзеркалення дерева
216 tree_node* create_mirror_flip(tree_node* node)
217 {
218
219     if (node == nullptr)
220     {
221         return nullptr;
222     }
223
224     tree_node* new_node = new tree_node(node->data);
225
226     new_node->right = create_mirror_flip(node->left);
227     new_node->left = create_mirror_flip(node->right);
228
229     return new_node;
230 }
231

```

```

232
233 //Сума підвисів
234 tree_node* tree_sum (tree_node* node)
235 {
236     if ((node == nullptr) || ((node->left == nullptr) && (node->right == nullptr))) return nullptr;
237
238     tree_sum(node->left);
239     tree_sum(node->right);
240
241     node->data = 0;
242     if (node->right != nullptr)
243     {
244         node->data += node->right->data;
245     }
246     if (node->left != nullptr)
247     {
248         node->data += node->left->data;
249     }
250
251     return node;
252 }
253
254 //Виведення дерева
255 void print_tree(tree_node* node)
256 {
257     if (node == nullptr)
258     {
259         return;
260     }
261
262     cout<<node->data<<" ";
263     print_tree(node->left);
264     print_tree(node->right);
265
266 }
267 //Binary tree
268

```

```

int main()
{
    //task_1
    Node* head = nullptr;

    for (int i = 0; i < 10; i++)
    {
        head = push_back(head, i);
    }

    cout<<"Starting list: ";
    print_list(head);

    Node* new_head = reverse_list(head);
    cout<<"Reversed list: ";
    print_list(new_head);

    //task 2
    Node* head_1 = nullptr;
    Node* head_2 = nullptr;

    head_1 = push_back(head_1, 5);
    head_1 = push_back(head_1, 6);
    head_1 = push_back(head_1, 5);
    head_1 = push_back(head_1, 7);
    head_1 = push_back(head_1, 8);

    head_2 = push_back(head_2, 5);
    head_2 = push_back(head_2, 6);
    head_2 = push_back(head_2, 4);

    if (compare(head_1, head_2))
    {
        cout<<"Lists are equal"<<endl;
    }
    else cout<<"Lists aren't equal"<<endl;

    //task 3
    string num_1, num_2, box;
    cout<<"Enter first number: ";
    cin>>num_1;
    cout<<"Enter second number: ";
    cin>>num_2;

    if (num_2.length() > num_1.length())
    {
        box = num_1;
        num_1 = num_2;
        num_2 = box;
    }
}

```

```

320 Node* n1 = nullptr;
321 Node* n2 = nullptr;
322
323 for (int i = num_1.length() - 1; i >= 0; i--)
324 {
325     n1 = push_back(n1, (int)num_1[i] - 48);
326 }
327 for (int i = num_2.length() - 1; i >= 0; i--)
328 {
329     n2 = push_back(n2, (int)num_2[i] - 48);
330 }
331
332 Node* sum;
333 sum = add_two_numbers(n1, n2);
334
335 Node* new_sum = reverse_list(sum);
336 cout<<num_1<<" + "<<num_2<<" = ";
337 print_number(new_sum);
338
339 //task 4
340 tree_node* root = nullptr;
341
342 root = insert(root, 5);
343 root = insert(root, 3);
344 root = insert(root, 6);
345 root = insert(root, 2);
346 root = insert(root, 10);
347
348
349 tree_node* new_root;
350 new_root = create_mirror_flip(root);
351
352 cout<<"First tree: ";
353 print_tree(root);
354
355 cout<<"\nMirrored tree: ";
356 print_tree(new_root);
357
358 root = tree_sum(root);
359 cout<<"\nSum tree: ";
360 print_tree(root);
361
362 return 0;
363 }

```

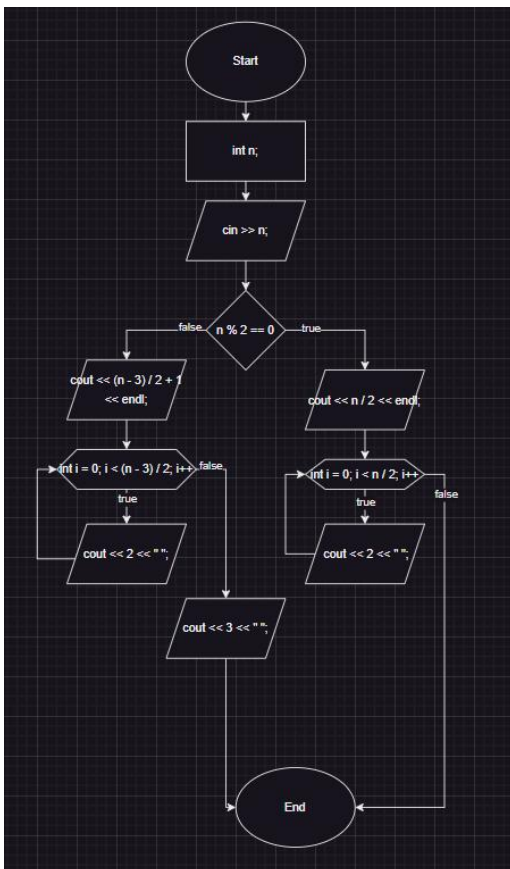
Завдання №5 Epic 6 Task 7 - Practice# programming: Self Practice Task

```

1 #include <iostream>
2 using namespace std;
3
4 int main() {
5     int n;
6     cin >> n;
7
8     if (n % 2 == 0) {
9         // n парне: максимум груп розміром 2
10        cout << n / 2 << endl;
11        for (int i = 0; i < n / 2; i++) {
12            cout << 2 << " ";
13        }
14    } else {
15        // n непарне: одна група з 3, решта — з 2
16        cout << (n - 3) / 2 + 1 << endl; // Одна група з 3 і решта з 2
17        for (int i = 0; i < (n - 3) / 2; i++) {
18            cout << 2 << " ";
19        }
20        cout << 3 << " ";
21    }
22
23    return 0;
24 }
25
26

```

Flowchart:





## Робота в команді

Zoom Workplace Meeting Khrytyna Ivan's screen

docs.google.com/spreadsheets/d/1CYtYh-dfUlgme7BpuT8wZdL\_2zndk7d51552872159gds1552872159

Epic 6 - Team Individual Tasks

АТА6

	C	D	E	F	G	H
1						
2		Программи Код №1	Программи Код №2	Программи Код №3	Программи Код №3	Программи Код №4
3	Студент	Файл 1	Файл 2	Файл 3.1	Файл 3.2	Файл 4
4		VNS Lab 10 - Task 1-N	Algotester Lab 5	Algotester Lab 7-8	Algotester Lab 7-8	Class Practice Work
5		Вариант №1	Вариант №1	Вариант №1	Вариант №1	Код в практичних по темі на заняттях
6						
7	квестві	epic_6_pactice_and_labs_john_black				
8	квестві та в глці на ПК	vns_lab_10_task_john_black.c	algotester_lab_5_task_john_black.c	algotester_lab_7_8_variant_1_1	algotester_lab_7_8_variant_2_1	practice_work_team_tasks_john_black.c
9		YES	YES	YES	YESNO	YES
10		NO	YES	YES	YESNO	NO
11	квестві та в глці на ПК	epic_6_pactice_and_labs_report_john_black.docx				
12	квестві у звіті	YES	YES	YES	YESNO	YES
13	квестві у звіті	YES	YES	YES	YESNO	YES
14	квестві у звіті	YES	YES	YES	YESNO	YES
15	квестві у звіті	YES	YES	YES	YESNO	YES
16						
17	квестві	4	2	3	3	YES
18	квестві	24		1	1	YES
19	квестві	23	2	1	1	YES
20	квестві	22	3	3	3	YES
21	квестві	21	2	2	2	YES
22	квестві	20	3	1	1	YES

WSI-11 WSI-12 WSI-13

Pull request: [https://github.com/artificial-intelligence-department/ai\\_programming\\_playground\\_2024/pull/619](https://github.com/artificial-intelligence-department/ai_programming_playground_2024/pull/619)

**Висновок:** під час виконання лабораторної роботи я навчився краще працювати і розуміти динамічні структури. Навчився писати власні алгоритми для роботи з такими структурами.