

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

Виконала:

Студент(ка) групи ШІ-12

Бугай Софія Володимирівна

Тема роботи:

Основи динамічних структур даних: стек, черга, зв'язний список, дерево.

Мета роботи:

Освоєння основних принципів роботи з динамічною пам'яттю, отримання навичок реалізації та використання стеку, черги, зв'язних списків та дерев.

Теоретичні відомості:

- 1) Теоретичні відомості з переліком важливих тем:
 - Тема №*.1: Динамічні структури даних.
 - Тема №*.2: Алгоритми обробки.
- 2) Індивідуальний план опрацювання теорії:
 - Тема №*.1: Динамічні структури даних.
 - Джерела Інформації
 - Лекції О. Пшеничного.
 - Практичні заняття М. Фаріон.
 - Відео [Neso Academy : Basics of Dynamic Memory Allocation.](#)
 - Сайт [GeeksforGeeks : Linked List in C++.](#)
 - Статус: Ознайомлена в більшості
 - Початок опрацювання теми: 18.11
 - Звершення опрацювання теми: 28.11
 - Тема №*.2: Алгоритми обробки.
 - Джерела Інформації:
 - Лекції О. Пшеничного.
 - Практичні заняття М. Фаріон.
 - Стаття.
 - Курс.
 - Статус: Ознайомлена частково
 - Початок опрацювання теми: 18.11
 - Звершення опрацювання теми: 28.11

Виконання роботи:

1. Опрацювання завдання та вимог до програм та середовища:

Завдання №1 Зв'язний список та Бінарні дерева

- Деталі завдання : Реалізувати :
 - Метод реверсу списку
 - Порівняння списків
 - Додавання великих чисел
 - Віддзеркалювання дерева
 - Запис кожному батьківському вузлу суму підвузлів
- Час на реалізацію : 2 год

Завдання №2 VNS Labs 10

- Варіант завдання : 13
- Деталі завдання : Написати програму, у якій створюються динамічні структури й виконати їхню обробку у відповідності зі своїм варіантом. Розробити такі функції:
 1. Створення списку.

2. Додавання елемента в список .
3. Знищення елемента зі списку.
4. Друк списку.
5. Запис списку у файл.
6. Знищення списку.
7. Відновлення списку з файлу.

Записи в лінійному списку містять ключове поле типу `*char` (рядок символів).

Сформуванати двонаправлений список. Знищити з нього K перших елементів. Додати елемент після елемента, що починається із зазначеного символу.

- Час на реалізацію : 4 год

Завдання №3 Algotester Lab 5

- Варіант завдання : 1
- Деталі завдання : У світі Атод сестри Ліна і Рілай люблять грати у гру. У них є дошка із 8-ми рядків і 8-ми стовпців. На перетині ii-го рядка і j-го стовпця лежить магічна куля, яка може світитись магічним світлом (тобто у них є 64 кулі). На початку гри деякі кулі світяться, а деякі ні... Далі вони обирають N кулі і для кожної читають магічне заклиння, після чого всі кулі, які лежать на перетині стовпця і рядка обраної кулі міняють свій стан (ті що світяться - гаснуть, ті, що не світяться - загораються). Також вони вирішили трохи Вам допомогти і придумали спосіб як записати стан дошки одним числом а із 8-ми байт, а саме (див. Примітки):
 - Молодший байт задає перший рядок матриці;
 - Молодший біт задає перший стовпець рядку;
 - Значення біту каже світиться куля чи ні (0 - ні, 1 - так);

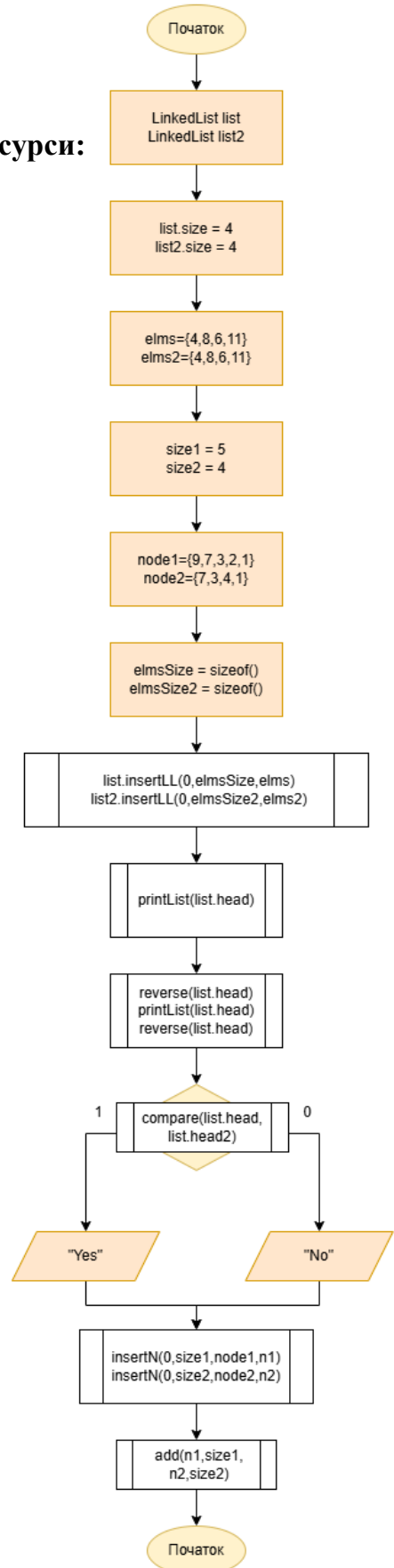
Тепер їх цікавить яким буде стан дошки після виконання N заклинань

- Час на реалізацію : 20 хв

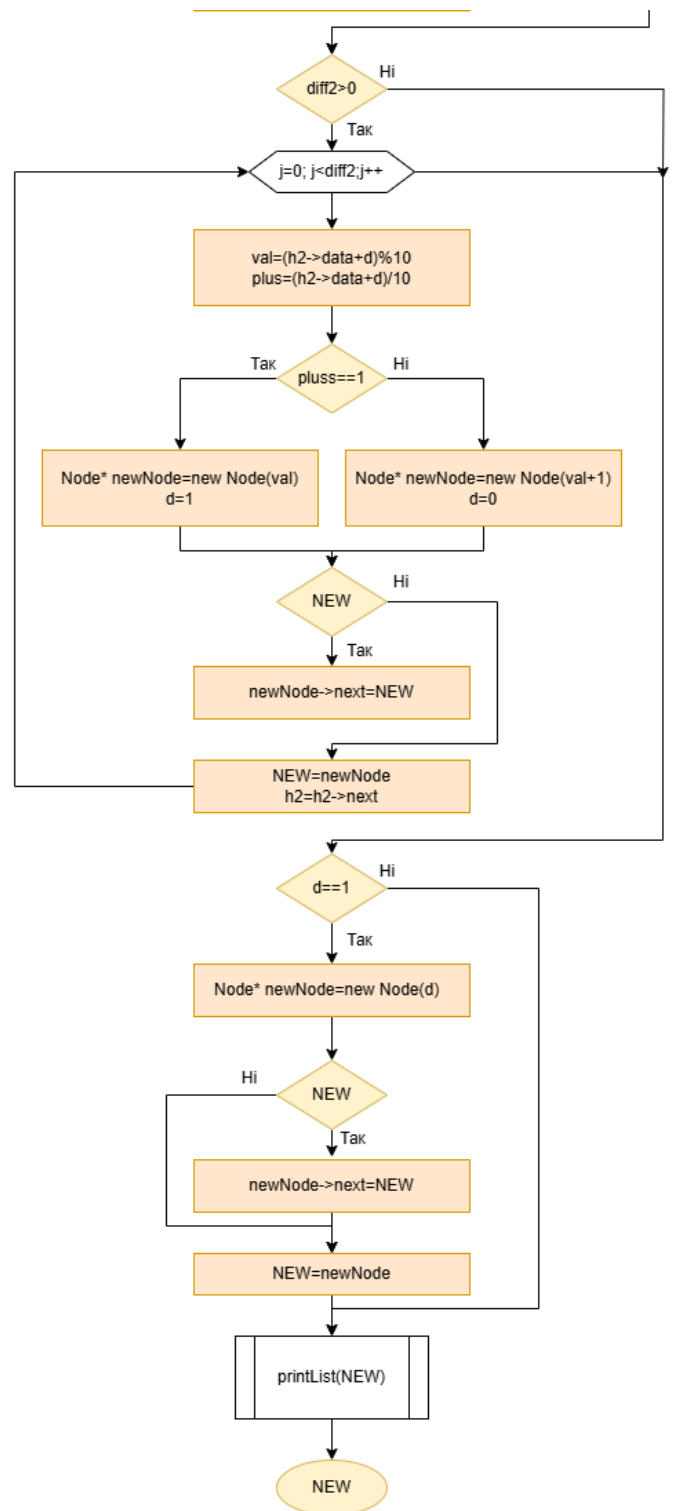
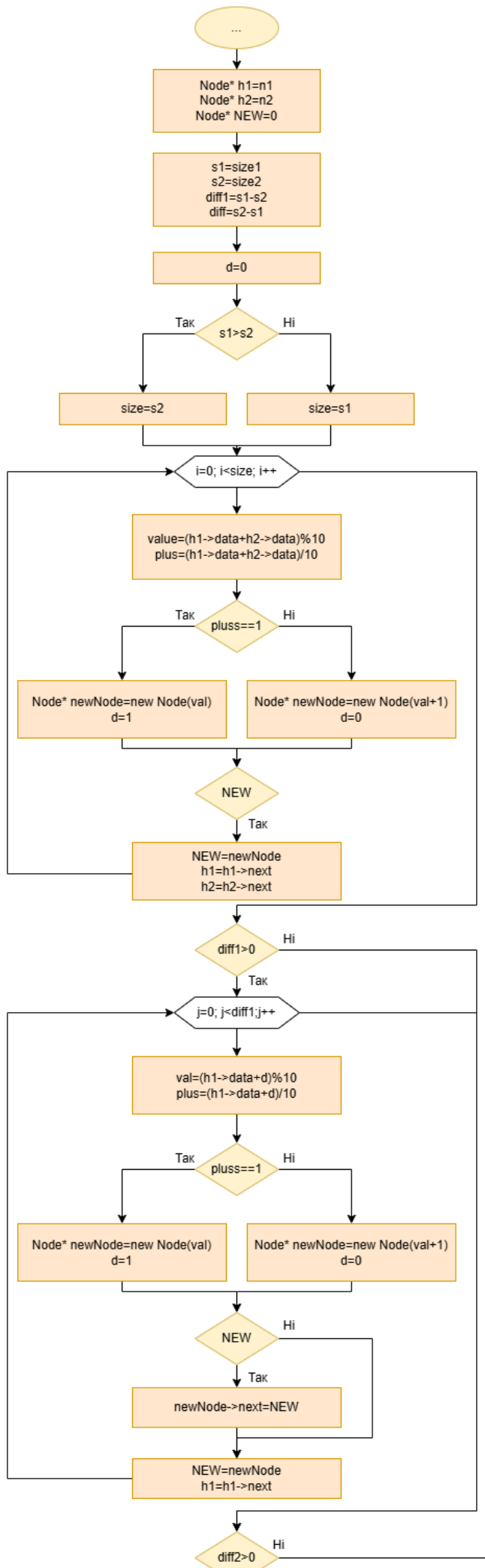
Завдання №4 Algotester Lab 7-8

- Варіант завдання : 3
- Деталі завдання : Ваше завдання - власноруч реалізувати структуру даних "Двійкове дерево пошуку".
Ви отримаєте QQ запитів, кожен запит буде починатися зі слова-ідентифікатора, після якого йдуть його параметри. Вам будуть поступати запити такого типу:
 - **Вставка:**
Ідентифікатор - insertinsert
Ви отримуєте ціле число valuevalue - число, яке треба вставити в дерево.
 - **Пошук:**
Ідентифікатор - containscontains
Ви отримуєте ціле число valuevalue - число, наявність якого у дереві необхідно перевірити.
Якщо valuevalue наявне в дереві - ви виводите YesYes, у іншому випадку NoNo.
 - **Визначення розміру:**
Ідентифікатор - sizesize
Ви не отримуєте аргументів.
Ви виводите кількість елементів у дереві.
 - **Вивід дерева на екран**
Ідентифікатор - printprint

Реалізувати використовуючи перегрузку оператора `<<main(nodes)`



Node* add(Node* n1, int size1, Node* n2, int size2)



```

1  #include <iostream>
2
3  class Node {
4  public:
5      int data;
6      Node* next;
7
8      Node(int value) {
9          data = value;
10         next = nullptr;
11     }
12 };
13
14
15
16 class LinkedList {
17
18     public:
19     Node* head;
20     int size;
21     LinkedList() {
22         size = 0;
23         head = nullptr;
24     }
25
26     void insertLL(int index, int listSize, int values[]) {
27         if (index < 0 || index > size || listSize <= 0) {
28             return;
29         }
30         Node* current = head;
31         for (int i = 0; i < index; i++) {
32             current = current->next;
33         }
34         for (int j = 0; j < listSize; j++) {
35             Node* newNode = new Node(values[j]);
36             if (current) {
37                 newNode->next = current;
38             }
39             current = newNode;
40             size++;
41         }
42         head = current;
43     }
44 };
45
46
47
48 // реверс списка
49 Node* reverse(Node* &head){
50     Node* current = head;
51     Node* next = nullptr;
52     Node* prev = nullptr;
53     while(current != nullptr){
54         next = current->next;
55         current->next = prev;
56         prev = current;
57         current = next;
58     }
59     head = prev;
60     return prev;
61 }
62
63
64
65 // друк списка
66 void printList(Node* head) {
67     Node* current = head;
68     while(current) {
69         std::cout << current->data << " ";
70         current = current->next;
71     }
72     std::cout << std::endl;
73 }
74

```

```

77 // порівняння списків
78 bool compare(Node* h1, Node* h2){
79     Node* head1 = h1;
80     Node* head2 = h2;
81     while(head1 && head2) {
82         if(head1->data == head2->data){
83             head1 = head1->next;
84             head2 = head2->next;
85         } else {
86             return false;
87         }
88     }
89     if((head2==nullptr && head1!=nullptr) || (head2!=nullptr && head1==nullptr)){
90         return false;
91     }
92
93     return true;
94 }
95
96
97 //створення нодів
98 void insertN(int index, int listSize, int values[], Node* &n) {
99     Node* current = n;
100     for (int i = 0; i < index; i++) {
101         current = current->next;
102     }
103     for (int j = 0; j < listSize; j++) {
104         Node* newNode = new Node(values[j]);
105         if (current) {
106             newNode->next = current;
107         }
108         current = newNode;
109     }
110     n = current;
111 }
112
113 // додавання
114 Node* add(Node* n1, int size1, Node* n2, int size2){
115     Node* head1 = n1;
116     Node* head2 = n2;
117     Node* NEW = nullptr;
118     int s1 = size1;
119     int s2 = size2;
120     int diff1 = s1-s2;
121     int diff2 = s2-s1;
122     int d=0;
123     int size;
124     if(s1>s2){
125         size = s2;
126     } else {
127         size = s1;
128     }
129
130     for(int i = 0; i<size; i++){
131         int value = (head1->data + head2->data)%10;
132         int plus= (head1->data + head2->data)/10;
133         Node* newNode = new Node(value);
134         if(plus == 1){
135             Node* newNode = new Node(value);
136             d=1;
137         } else {
138             Node* newNode = new Node(value+1);
139             d=0;
140         }
141         if (NEW) {
142             newNode->next = NEW;
143         }
144         NEW = newNode;
145         head1 = head1->next;
146         head2 = head2->next;
147     }
148     if(diff1>0){
149         for(int j=0; j<diff1; j++){
150             int value = (head1->data+d)%10;
151             int plus =(head1->data+d)/10;
152             Node* newNode = new Node(value);
153             if(plus==1){
154                 Node* newNode = new Node(value);
155                 d=1;
156             } else {
157                 Node* newNode = new Node(value+d);
158                 d=0;
159             }
160             if (NEW) {
161                 newNode->next = NEW;
162             }
163             NEW = newNode;
164             head1 = head1->next;
165         }

```

```

167     if(diff2>0){
168         for(int j=0; j<diff2; j++){
169             int value = (head2->data+d)%10;
170             int plus =(head2->data+d)/10;
171             Node* newNode = new Node(value);
172             if(plus==1){
173                 Node* newNode = new Node(value);
174                 d=1;
175             } else {
176                 Node* newNode = new Node(value+d);
177                 d=0;
178             }
179             if (NEW) {
180                 newNode->next = NEW;
181             }
182             NEW = newNode;
183             head2 = head2->next;
184         }
185     }
186     if(d==1){
187         Node* newNode = new Node(d);
188         if (NEW) {
189             newNode->next = NEW;
190         }
191         NEW = newNode;
192     }
193     printList(NEW);
194     return NEW;
195 }
200 int main(){
201     LinkedList list;
202     LinkedList list2;
203     Node* n1;
204     Node* n2;
205     list.size=4;
206     list2.size=4;
207     int node1[]={9, 7, 3, 2, 1};
208     int size1 = 5;
209     int size2 = 4;
210     int node2[]={7, 3, 4, 1};
211     int elements[] = {4, 8, 6, 11};
212     int elements2[] = {4, 8, 6, 11};
213     int elementsSize = sizeof(elements)/sizeof(elements[0]);
214     int elementsSize2 = sizeof(elements2)/sizeof(elements2[0]);
215     list.insertLL(0, elementsSize, elements);
216     list2.insertLL(0, elementsSize2, elements2);
217
218
219     printList(list.head);
220
221     reverse(list.head);
222     printList(list.head);
223     reverse(list.head);
224
225     std::cout << "Чи однаковими є списки 1 та 2 ? : "
226     << ( compare(list.head, list2.head) ? "Yes" : "No")
227     << "\n";
228
229     insertN(0, size1, node1, n1);
230     insertN(0, size2, node2, n2);
231
232     add(n1, size1, n2, size2);
233
234     return 0;
235 }

```

11 6 8 4

4 8 6 11

Чи однаковими є списки 1 та 2 ? : Yes

1 0 4 6 6 2


```

1  #include <iostream>
2
3
4  struct TreeNode {
5      int data;
6      TreeNode* left;
7      TreeNode* right;
8
9      TreeNode(int value): data(value), left(nullptr), right(nullptr) {}
10 };
11
12 struct BinarySearchTree {
13
14 public:
15     BinarySearchTree() : root(nullptr) {}
16
17     void insert(int value) {
18         root = insert(root, value);
19     }
20
21     void inorderTraverse() {
22         inorderTraverse(root);
23     }
24
25
26     void create_mirror_flip(){
27         inorderTraverse(create_mirror_flip(root));
28     }
29
30     void tree_sum(){
31         tree_sum(root);
32     }
33
34 private:
35     TreeNode* root;
36
37     TreeNode* insert(TreeNode* node, int value) {
38         if (node == nullptr) {
39             return new TreeNode(value);
40         }
41
42         if (value < node->data) {
43             node->left = insert(node->left, value);
44         } else if (value > node->data) {
45             node->right = insert(node->right, value);
46         }
47
48         return node;
49     }
50
51     void inorderTraverse(TreeNode* node) {
52         if (node != nullptr) {
53             inorderTraverse(node->left);
54             std::cout << node->data << " ";
55             inorderTraverse(node->right);
56         }
57     }
58
59     TreeNode* create_mirror_flip(TreeNode* node){
60         if(node == nullptr){
61             return nullptr;
62         }
63         TreeNode* NewNode = new TreeNode(node->data);
64         NewNode->left = create_mirror_flip(node->right);
65         NewNode->right = create_mirror_flip(node->left);
66
67         return NewNode;
68     }
69
70     void tree_sum(TreeNode* &root){
71         if(root == nullptr){
72             return;
73         }
74         tree_sum(root->left);
75         tree_sum(root->right);
76         if(root->left && root->right){
77             root->data = root->left->data + root->right->data;
78         }
79         if(root->left && !root->right){
80             root->data = root->left->data;
81         }
82         if(!root->left && root->right){
83             root->data = root->right->data;
84         }
85     }
86 };

```

```

92 int main(){
93
94
95     BinarySearchTree myTree;
96
97     myTree.insert(4);
98     myTree.insert(2);
99     myTree.insert(5);
100    myTree.insert(1);
101    myTree.insert(6);
102    myTree.insert(3);
103
104    std::cout << "The tree : ";
105    myTree.inorderTraverse();
106    std::cout << "\n";
107
108
109    std::cout << "Flipped : ";
110    myTree.create_mirror_flip();
111    std::cout << "\n";
112
113
114    std::cout << "The sum of dauthers of node : ";
115    myTree.tree_sum();
116    myTree.inorderTraverse();
117    std::cout << "\n";
118
119
120    return 0;
121 }

```

```

The tree : 1 2 3 4 5 6
Flipped : 6 5 4 3 2 1
The sum of dauthers of node : 1 4 3 10 6 6

```

Завдання №2 VNS Labs 10

[Посилання на файл програми у пул-запиті GitHub](#)

```

1  #include <iostream>
2  #include <stdio.h>
3  #include <fstream>
4  #include <cstring>
5
6
7
8  class Node {
9  public:
10     char* data;
11     Node* prev;
12     Node* next;
13
14     Node(char* &value) {
15         data = value;
16         prev=nullptr;
17         next = nullptr;
18     }
19 };
20
21
22 class LinkedList {
23
24 public:
25     LinkedList() {
26         size = 0;
27         head = nullptr;
28         tail=nullptr;
29     }

```

```

31 // Створення списку.
32 void create(LinkedList *list){
33     int n = 4;
34     size = n;
35     char* elements[n] = {"sgyd", "lnaswer", "nskjdhdhdi", "aaaaaaa"};
36     for(int i = 0; i < n; i++){
37         char* value = elements[i];
38         Node* newNode = new Node(value);
39         newNode->data = strdup(value);
40         newNode->next = nullptr;
41
42         if(head == NULL){
43             newNode->prev = nullptr;
44             head = newNode;
45         }else{
46             Node* tmp = head;
47             while(tmp->next != nullptr){
48                 tmp = tmp->next;
49             }
50
51             tmp->next = newNode;
52             newNode->prev = tmp;
53         }
54     }
55 }
57 void insert(char index_char, char* value) {
58     Node* current = head;
59     int index = size+2;
60
61     for (int i = 0; i < size; i++) {
62         if (current && *(current->data) == index_char) {
63             index = i+1;
64             break;
65         }
66         if (current) {
67             current = current->next;
68         }
69     }
70     if(index == size+2) return;
71
72     Node* newNode = new Node(value);
73
74     if (index == size+1) {
75         if (tail) {
76             tail->next = newNode;
77         }
78         newNode->prev = tail;
79         tail = newNode;
80         if (!head) {
81             head = newNode;
82         }
83     } else { // Вставка в середину списку
84         Node* current = head;
85         for (int i = 0; i < index - 1; i++) { // Знайти елемент перед позицією вставки
86             current = current->next;
87         }
88         Node* nextNode = current->next;
89         current->next = newNode;
90         newNode->prev = current;
91         newNode->next = nextNode;
92         if (nextNode) {
93             nextNode->prev = newNode;
94         }
95     }
96
97     size++; // Збільшити розмір списку
98 }

```

```

103 // Знищення елемента зі списку.
104 void erase(int K){
105     if (K <= 0 || K > size) {
106         return;
107     }
108     Node* current = head;
109     for (int i = 0; i < K; i++) {
110         if (!current) break;
111         Node* nextNode = current->next;
112         delete current;
113         current = nextNode;
114         head = current;
115         size--;
116     }
117     head = current;
118     if (head) {
119         head->prev = nullptr;
120     }
121 }
122
123
124 // Друк списку.
125 void print(){
126     Node* current = head;
127     if(current==nullptr){
128         std::cerr << "Немає елементів у списку\n";
129         return;
130     }
131     while(current) {
132         std::cout << current->data << " ";
133         current = current->next;
134     }
135     std::cout << std::endl;
136 }
137
138 // Запис списку у файл.
139 void write_to_file(const char* name){
140     std::ofstream f(name);
141     if (!f.is_open()) {
142         std::cerr << "Не вдалося відкрити файл для запису" << std::endl;
143         return;
144     }
145     Node* current = head;
146     while(current) {
147         f << current->data << "\n";
148         current = current->next;
149     }
150     f.close();
151 }
152
153
154
155 // Знищення списку.
156 void deleteList(LinkedList **list){
157     Node *tmp = (*list)->head;
158     Node *next = NULL;
159     while (tmp) {
160         next = tmp->next;
161         delete tmp;
162         tmp = next;
163         head = tmp;
164         size--;
165     }
166 }
167
168 // Відновлення списку з файлу.
169 LinkedList from_file(const char* name){
170     std::ifstream f(name);
171     if(!name){
172         std::cerr << "Error opening file";
173         return *this;
174     }
175     char string[50];
176     while(f >> string){
177         char* value = string;
178         Node* newNode = new Node(value);
179         newNode->data = strdup(value);
180         newNode->next = nullptr;
181
182         if(head == NULL){
183             newNode->prev = nullptr;
184             head = newNode;
185         }else{
186             Node* tmp = head;
187             while(tmp->next != nullptr){
188                 tmp = tmp->next;
189             }
190             tmp->next = newNode;
191             newNode->prev = tmp;
192         }
193     }
194 }

```

```

196         return *this;
197     }
198
199     friend std::ostream& operator<<(std::ostream& os, const LinkedList& list) {
200         Node* current = list.head;
201
202         while(current) {
203             os << current->data << " ";
204             current = current->next;
205         }
206         os << std::endl;
207         return os;
208     }
209
210 private:
211     Node* head;
212     Node* tail;
213     int size;
214 };
215
216 int main(){
217     const char* name = "File with list. txt";
218     LinkedList* l = new LinkedList();
219
220     l->create(1);
221     //std::cout << "" << list;
222     l->print();
223
224     std::cout << "Введіть елемент, який хочете ввести : ";
225     char value[50];
226     scanf("%s", value);
227     while(getchar() != '\n');
228     std::cout << "Введіть символ, за яким знайти елемент : ";
229     char index_char;
230     scanf("%c", &index_char);
231     while(getchar() != '\n');
232     l->insert(index_char, value);
233     l->print();
234
235     std::cout << "Введіть кількість елементів, які потрібно видалити із початку списку: ";
236     int K;
237     std::cin >> K;
238     while(getchar() != '\n');
239     l->erase(K);
240     l->print();
241
242     std::cout << "Видалений список:\n";
243     l->write_to_file(name);
244     l->deleteList(&l);
245     l->print();
246
247     std::cout << "Відновлений список з файлу:\n";
248     l->from_file(name);
249     l->print();
250
251     return 0;
252 }

```

```

sgyd lnaswer nskjdhdi aaaaaaa
Введіть елемент, який хочете ввести : abrakadabra
Введіть символ, за яким знайти елемент : l
sgyd lnaswer abrakadabra nskjdhdi aaaaaaa
Введіть кількість елементів, які потрібно видалити із початку списку: 3
nskjdhdi aaaaaaa
Видалений список:
Немає елементів у списку
Відновлений список з файлу:
nskjdhdi aaaaaaa

```

Завдання №3 Algotester Lab 5

Посилання на файл програми у пул-запиті GitHub

```
1  #include <iostream>
2  #include <vector>
3  #include <cmath>
4  #include <cinttypes>
5  #include <cstdio>
6
7  int main(){
8      unsigned long long a, x;
9      uint64_t b=0;
10     int N, r=0, c=0;
11     std::vector<bool> row(8, 0);
12     std::vector<int> answer;
13     std::vector<std::vector<bool>> board(8, row);
14     std::cin >> a;
15     std::cin >> N;
16     std::vector<int> R(N), C(N);
17     for(int i=0; i<N; i++){
18         std::cin >> R[i] >> C[i];
19         R[i]--;
20         C[i]--;
21     }
22
23     while(a>0){
24         board[r][c]=a%2;
25         a/=2;
26         c++;
27         if(c==8){
28             c = 0;
29             r++;
30         }
31     }
32
33     for(int n=0; n<N; n++){
34         for(int i=0; i<8; i++){
35             if(board[R[n]][i]==1){
36                 board[R[n]][i]=0;
37             } else {
38                 board[R[n]][i]=1;
39             }
40         }
41         for(int i=0; i<8; i++){
42             if(board[i][C[n]]==1){
43                 board[i][C[n]]=0;
44             } else {
45                 board[i][C[n]]=1;
46             }
47         }
48         if(board[R[n]][C[n]]==1){
49             board[R[n]][C[n]]=0;
50         } else {
51             board[R[n]][C[n]]=1;
52         }
53     }
54
55     for(int i=0; i<8; i++){
56         for(int j=0; j<8; j++){
57             if(board[i][j]==1){
58                 x = pow(2, j+i*8);
59                 b+= x;
60             }
61         }
62     }
63
64     std::cout << b;
65
66     return 0;
67 }
```

| Created | Compiler | Result | Time (sec.) | Memory (MiB) | Actions |
|------------|----------|----------------|-------------|--------------|----------------------|
| 3 days ago | C++ 23 | Accepted | 0.003 | 1.426 | View |
| 3 days ago | C++ 23 | Wrong Answer 3 | 0.002 | 1.270 | View |

Завдання №4 Algotester Lab 7-8

Посилання на файл програми у пул-запиті GitHub

```

1  #include <iostream>
2  #include <string>
3
4
5
6  struct TreeNode {
7      int data;
8      TreeNode* left;
9      TreeNode* right;
10
11      TreeNode(int value): data(value), left(nullptr), right(nullptr) {}
12  };
13
14  struct BinarySearchTree {
15
16  public:
17      BinarySearchTree() : root(nullptr) {}
18
19      void insert(int value) {
20          root = insert(root, value);
21      }
22
23      void print() {
24          print(root);
25      }
26
27      bool contains(int value) {
28          return contains(root, value);
29      }
30      int size(){
31          return size(root);
32      }
33
34  private:
35      TreeNode* root;
36      TreeNode* insert(TreeNode* node, int value) {
37          if (node == nullptr) {
38              return new TreeNode(value);
39          }
40
41          if (value < node->data) {
42              node->left = insert(node->left, value);
43          } else if (value > node->data) {
44              node->right = insert(node->right, value);
45          }
46
47          return node;
48      }
49
50      bool contains(TreeNode* node, int value) {
51          if (node == nullptr) {
52              return false;
53          }
54
55          if (value == node->data) {
56              return true;
57          } else if (value < node->data) {
58              return contains(node->left, value);
59          } else {
60              return contains(node->right, value);
61          }
62      }
63
64      void print(TreeNode* node) {
65          if (node != nullptr) {
66              print(node->left); // LEFT
67              std::cout << node->data << " "; // CENTER
68              print(node->right); // RIGHT
69          }
70      }
71  };
72
73  int main() {
74      BinarySearchTree tree;
75
76      int n;
77      while (n < 1000000000) {
78          int value;
79          char op;
80          if (n % 1000000000 == 0) {
81              std::cout << "Enter value and operation: ";
82              std::cin >> value >> op;
83              if (op != 'i' && op != 'c' && op != 's' && op != 'p') {
84                  std::cout << "Invalid operation. Please use 'i', 'c', 's', or 'p'." << std::endl;
85                  continue;
86              }
87          }
88          if (op == 'i') {
89              tree.insert(value);
90          } else if (op == 'c') {
91              bool found = tree.contains(value);
92              if (found) {
93                  std::cout << "Value " << value << " is in the tree." << std::endl;
94              } else {
95                  std::cout << "Value " << value << " is not in the tree." << std::endl;
96              }
97          } else if (op == 's') {
98              int size = tree.size();
99              std::cout << "Size of the tree: " << size << std::endl;
100          } else if (op == 'p') {
101              tree.print();
102              std::cout << std::endl;
103          }
104          n++;
105      }
106      return 0;
107  }

```

```
73     int size(TreeNode* node) {
74         int s = 0;
75         if(node != nullptr){
76             s= 1 + size(node->left) + size(node->right);
77         } else {
78             return s;
79         }
80         return s;
81     }
82 };
83
84 int main() {
85     BinarySearchTree myTree;
86     int N;
87     std::string operation;
88
89
90     std::cin >> N;
91     for(int i=0; i<N; i++){
92         std::cin >> operation;
93         if(operation == "insert"){
94             int a;
95             std::cin >> a;
96             myTree.insert(a);
97         }
98         if(operation == "print"){
99             myTree.print();
100             std::cout << std::endl;
101         }
102         if(operation == "contains" ){
103             int a;
104             std::cin >> a;
105             if (myTree.contains(a)){
106                 std::cout << "Yes"
107                 << std::endl;
108             } else {
109                 std::cout << "No"
110                 << std::endl;
111             }
112         }
113
114     }
115     if(operation == "size"){
116         std::cout << myTree.size() << std::endl;
117     }
118 }
119 return 0;
120 }
```

| Created | Compiler | Result | Time (sec.) | Memory (MiB) | Actions |
|------------|----------|----------------|-------------|--------------|----------------------|
| 2 days ago | C++ 23 | Accepted | 0.008 | 2.426 | View |
| 3 days ago | C++ 23 | Accepted | 0.008 | 1.520 | View |
| 3 days ago | C++ 23 | Wrong Answer 2 | 0.004 | 0.926 | View |
| 3 days ago | C++ 23 | Wrong Answer 2 | 0.004 | 0.941 | View |
| 3 days ago | C++ 23 | Wrong Answer 2 | 0.004 | 1.063 | View |
| 3 days ago | C++ 23 | Wrong Answer 2 | 0.004 | 1.008 | View |
| 3 days ago | C++ 23 | Wrong Answer 2 | 0.004 | 1.027 | View |

3. Кооперація з командою:



Зустріч 28.11 18:00-20:00. Обговорювали загалини у теоретичних знаннях та допомагали один одному із кодами.

Висновки:

Після даної лабораторної роботи я навчилась використовувати динамічні структури даних, як створювати їх та ключові операції.