

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 ІІІ-12

Стефанович Євгеній

Тема лабораторної роботи:

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

Мета лабораторної роботи:

Метою цієї лабораторної роботи є освоєння основних принципів роботи з динамічними структурами даних, а також отримання навичок реалізації та використання таких структур, як стек, черга, зв'язний список і дерево. Я прагну зрозуміти, як динамічні структури дозволяють ефективніше використовувати пам'ять шляхом виділення ресурсів у динамічній області пам'яті (heap), а також які основні операції можна виконувати з кожною структурою. Я маю на меті навчитися обирати відповідну структуру даних для конкретної задачі, враховуючи її особливості та поведінку в пам'яті, а також зрозуміти алгоритми роботи з динамічними структурами, такі як додавання, видалення елементів і пошук. Це допоможе мені закласти основу для подальшого вивчення складніших структур даних та алгоритмів їх обробки.

Джерела:

-University lectures

-aCode – data structures

-Google + ChatGPT for learning about different types of trees, stacks and queues with their implementations.

Виконання:

Lab# programming: VNS Lab 10

Time expected: 3h

Time spent: 3h

Порядок виконання роботи

1. Написати функцію для створення списку. Функція може створювати порожній список, а потім додавати в нього елементи.
2. Написати функцію для друку списку. Функція повинна передбачати вивід повідомлення, якщо список порожній.
3. Написати функції для знищення й додавання елементів списку у відповідності зі своїм варіантом.

4. Виконати зміни в списку й друк списку після кожної зміни.
5. Написати функцію для запису списку у файл.
6. Написати функцію для знищення списку.
7. Записати список у файл, знищити його й виконати друк (при друці повинне бути видане повідомлення "Список порожній").
8. Написати функцію для відновлення списку з файлу.
9. Відновити список і роздрукувати його.
10. Знищити список.
16. Записи в лінійному списку містять ключове поле типу `*char` (рядок символів). Сформувані двонаправлений список. Знищити елемент із заданим ключем. Додати K елементів у кінець списку.

```
Створення списку...
Друк списку...
Node 1 Node 2 Node 3 Node 4 Node 5 Node 6
Видалення елемента: Node 4 зі списку...
Друк списку...
Node 1 Node 2 Node 3 Node 5 Node 6
Додавання 3 елементів в кінець списку...
Друк списку...
Node 1 Node 2 Node 3 Node 5 Node 6 Node 7 Node 8 Node 9
Запис списку в файл: file.txt...
Знищення списку...
Друк списку...
Список пустий.
Відновлення списку з файлу: file.txt...
Друк списку...
Node 1 Node 2 Node 3 Node 5 Node 6 Node 7 Node 8 Node 9
Знищення списку...
Друк списку...
Список пустий.
PS C:\Users\Eugene\Downloads\output> █
```

```

C: > Users > Eugene > Desktop > epic_6 > G+ vns_lab_10_task_eugenie_stefanovich.cpp > main()
1  #include <iostream>
2  #include <cstring>
3  #include <cstdint>
4
5  using namespace std;
6
7  struct node {
8      char* data;
9      node* next;
10     node* prev;
11 };
12
13 void createLL(node*& head) {
14     cout << "Создания списка..." << endl;
15     const char* values[] = {"Node 1", "Node 2", "Node 3", "Node 4", "Node 5", "Node 6"};
16
17     for(int i = 0; i < 6; i++){
18         node* newNode = new node;
19         newNode->data = strdup(values[i]);
20         newNode->next = nullptr;
21
22         if(head == nullptr){
23             newNode->prev = nullptr;
24             head = newNode;
25         }else{
26             node* tmp = head;
27             while(tmp->next != nullptr){
28                 tmp = tmp->next;
29             }
30
31             tmp->next = newNode;
32             newNode->prev = tmp;
33         }
34     }
35 }
36
37 void deleteByData(node*& head, const char* data) {
38     cout << "Удаления элемента: " << data << " из списка..." << endl;
39     if(head == nullptr)
40         return;
41     node* tmp = head;
42     node* toDelete;
43
44     while (tmp != nullptr) {
45         if (strcmp(tmp->data, data) == 0) {
46             toDelete = tmp;
47
48             if (toDelete == head) {
49                 head = toDelete->next;
50                 head->prev = nullptr;
51             }
52             else {
53                 if (toDelete->prev != nullptr) {
54                     toDelete->prev->next = toDelete->next;
55                 }
56                 if (toDelete->next != nullptr) {
57                     toDelete->next->prev = toDelete->prev;
58                 }
59             }
60
61             free(toDelete->data);
62             delete toDelete;
63             tmp = (head != nullptr) ? head : nullptr;
64         } else {
65             tmp = tmp->next;
66         }
67     }
68 }
69
70 void printLL(node* head) {
71     cout << "Печать списка..." << endl;
72
73     if(head == nullptr){
74         cout << "Список пустой." << endl;
75         return;
76     }
77
78     node* tmp = head;
79     while(tmp != nullptr){

```


C:\Users\Eugene\Desktop> epic_6 > C:\vrs_lab_10_task_eugenie_stefanovich.cpp > main()

```
78 void printLL(node* head) {
79     while(tmp != nullptr){
80         cout << tmp->data << " ";
81         tmp = tmp->next;
82     }
83     cout << endl;
84 }
85
86 void addElementsToEnd(node*& head, int K, ...) {
87     cout << "Додавання " << K << " елементів в кінець списку..." << endl;
88
89     va_list args;
90     va_start(args, K);
91
92     for (int i = 0; i < K; i++) {
93         const char* data = va_arg(args, const char*);
94         node* newNode = new node;
95
96         newNode->data = strdup(data);
97         newNode->next = nullptr;
98         newNode->prev = nullptr;
99
100         if (head == nullptr) {
101             head = newNode;
102         }
103         else {
104             node* tmp = head;
105             while (tmp->next != nullptr) {
106                 tmp = tmp->next;
107             }
108             tmp->next = newNode;
109             newNode->prev = tmp;
110         }
111     }
112
113     va_end(args);
114 }
115
116 void writeLLToFile(node*& head, const char* filename) {
117     cout << "Занес список в файл: " << filename << "..." << endl;
118     FILE* f = fopen(filename, "w");
119     if(f == nullptr){
120         cerr << "Can't open the file.";
121         exit(1);
122     }
123
124     node* tmp = head;
125     while(tmp != nullptr){
126         const char* buffer = tmp->data;
127         fputs(buffer, f);
128         fputs("\n", f);
129         tmp = tmp->next;
130     }
131     fclose(f);
132 }
133
134 void deleteLL(node*& head) {
135     cout << "Видалення списку..." << endl;
136     node* tmp = head;
137     while(tmp != nullptr){
138         node* nextNode = tmp->next;
139         free(tmp->data);
140         delete(tmp);
141         tmp = nextNode;
142     }
143     head = nullptr;
144 }
145
146 void fromFileToLL(node*& head, const char* filename) {
147     cout << "Відновлення списку з файлу: " << filename << "..." << endl;
148     FILE* f = fopen(filename, "r");
149     if(f == nullptr){
150         cerr << "Can't open the file.";
151         exit(1);
152     }
153
154     char buffer[256];
155     node* tmp = nullptr;
156     while(fgets(buffer, sizeof(buffer), f)){
```

```

while(fgets(buffer, sizeof(buffer), f)){
    buffer[strcspn(buffer, "\n")] = '\0';

    node* newNode = new node;
    newNode->data = strdup(buffer);
    newNode->next = nullptr;
    newNode->prev = tmp;

    if(head == nullptr){
        head = newNode;
    }else{
        tmp->next = newNode;
    }
    tmp = newNode;
}
fclose(f);
}

int main() {
    node* head = nullptr;
    const char* dataToDelete = "Node 4";
    const char* filename = "file.txt";

    createLL(head);
    printLL(head);

    deleteByData(head, dataToDelete);
    printLL(head);

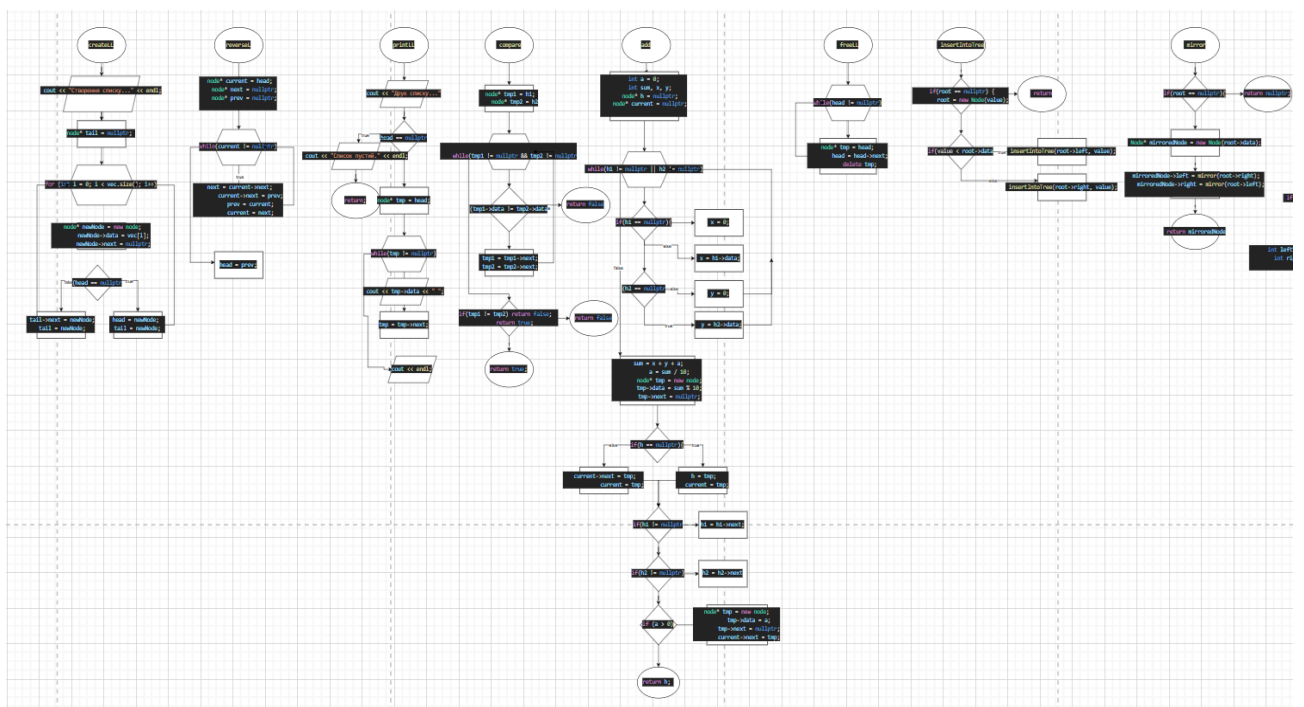
    addKElementsToEnd(head, 3, "Node 7", "Node 8", "Node 9");
    printLL(head);

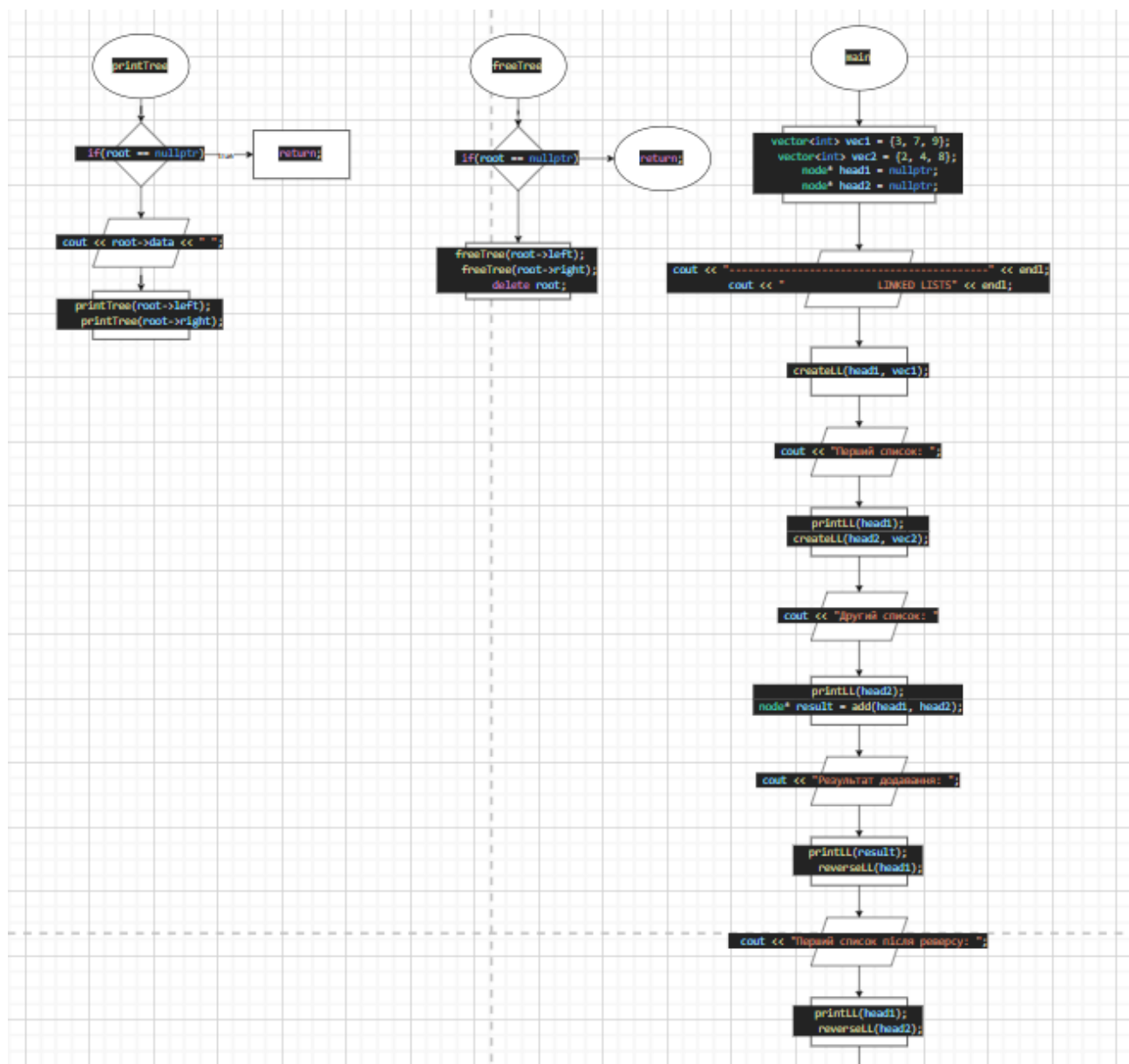
    writeLLToFile(head, filename);
    deleteLL(head);
    printLL(head);

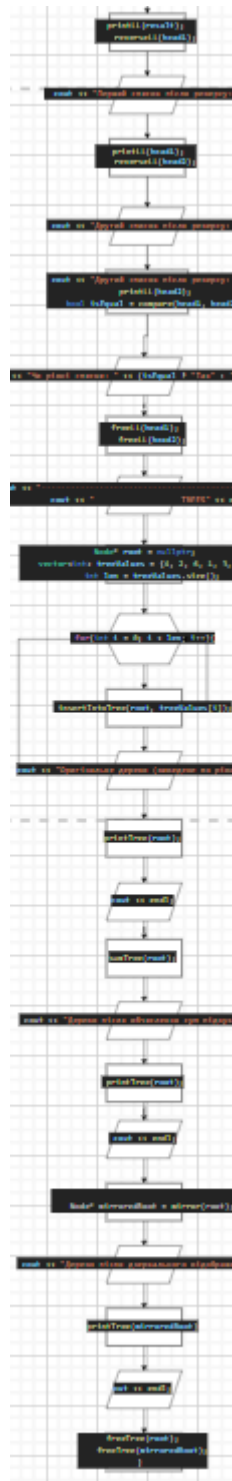
    fromFileToLL(head, filename);
    printLL(head);

    deleteLL(head);
    printLL(head);
}

```







Lab# programming: Algotester Lab 5

Time expected: 15 min

Time spent: 1h+

Lab 5v3

Limits: 1 sec., 256 MB

У вас є карта гори розміром $N \times M$.

Також ви знаєте координати $\{x, y\}$, у яких знаходиться вершина гори.

Ваше завдання - розмалювати карту таким чином, щоб найнижча точка мала число 0, а пік гори мав найбільше число.

Клітинки які мають суміжну сторону з вершиною мають висоту на один меншу, суміжні з ними і не розфарбовані мають ще на 1 меншу висоту і так далі.

Input

У першому рядку 2 числа N та M - розміри карти

у другому рядку 2 числа x та y - координати піку гори

Output

N рядків по M елементів в рядку через пробіл - висоти карти.

```
C: > Users > Eugene > Desktop > epic_6 > G+ algotester_lab_5_task_eugenie_stefanovich.cpp > ...
1  < #include <iostream>|
2  #include <cmath>
3
4  using namespace std;
5
6
7  < int main(){
8      int N, M, x, y , maxi = 0 , buf;
9      cin >> N >> M;
10     cin >> x >> y;
11     (x-1)*M + y;
12     cout <<endl;
13     buf = (y - 1) + x - 1;
14     if(maxi < buf)
15         maxi = buf;
16     buf = ( M - y ) + x - 1;
17     if(maxi < buf)
18         maxi = buf;
19     buf = ( M - y ) + (N-x);
20     if(maxi < buf)
21         maxi = buf;
22     buf = (y - 1) + (N-x);
23     if(maxi < buf)
24         maxi = buf;
25
26     < for(int i = 1 ; i <= N*M ; i++){
27         int otkl;
28
29         if(i%M ==0)
30             otkl = abs(M -y) + abs((i/M ) - x);
31         else
32             otkl = abs(i%M -y) + abs((i/M +1) - x);
33
34         cout << maxi - otkl <<" ";
35         if(i % M == 0)
36             cout <<endl;
37     }
38
39     return 0;
40 }
```

```
3 9
1 2
8 9 8 7 6 5 4 3 2
7 8 7 6 5 4 3 2 1
6 7 6 5 4 3 2 1 0
PS C:\Users\user\Do
```

Lab78 Var – 3

Lab 78v3

Limits: 1 sec., 256 MiB

Ваше завдання - власноруч реалізувати структуру даних "Двійкове дерево пошуку".

Ви отримаєте Q запитів, кожен запит буде починатися зі слова-ідентифікатора, після якого йдуть його параметри.

Вам будуть поступати запити такого типу:

- **Вставка:**

Ідентифікатор - *insert*

Ви отримуєте ціле число *value* - число, яке треба вставити в дерево.

- **Пошук:**

Ідентифікатор - *contains*

Ви отримуєте ціле число *value* - число, наявність якого у дереві необхідно перевірити.

Якщо *value* наявне в дереві - ви виводите *Yes*, у іншому випадку *No*.

- **Визначення розміру:**

Ідентифікатор - *size*

Ви не отримуєте аргументів.

Ви виводите кількість елементів у дереві.

- **Вивід дерева на екран**

Ідентифікатор - *print*

Ви не отримуєте аргументів.

Ви виводите усі елементи дерева через пробіл.

Реалізувати використовуючи перегрузку оператора <<

Input

Ціле число Q - кількість запитів.

У наступних рядках Q запитів у зазначеному в умові форматі.

```

C:\Users\Eugene\Desktop> epic_6 > C:\algotester_lab_7_8_variant_1_eugenie_stefanovich.cpp > binaryTree<T> > insertRecursively(N
1  #include <iostream>
2  #include <string>
3  #include <algorithm>
4
5  using namespace std;
6
7  enum Operation {
8      INSERT,
9      SIZE,
10     PRINT,
11     CONTAINS,
12     UNKNOWN
13 };
14
15 Operation getOperation(const string& command) {
16     if (command == "insert") return
17         INSERT;
18     if (command == "size") return
19         SIZE;
20     if (command == "print") return
21         PRINT;
22     if (command == "contains") return
23         CONTAINS;
24     return UNKNOWN;
25 }
26
27 template<typename T = int>
28 class binaryTree {
29 private:
30     struct Node {
31         T wid;
32         Node* left;
33         Node* right;
34         Node(T val) : wid(val), left(nullptr), right(nullptr) {}
35     };
36
37     Node* kor;
38     int killusl;
39
40
41     void insertRecursively(Node* node, T wid) {
42         if (wid < node->wid) {
43             if (node->left == nullptr) {
44                 node->left = new Node(wid);
45                 killusl++;
46             } else {
47                 insertRecursively(node->left, wid);
48             }
49         } else if (wid > node->wid) {
50             if (node->right == nullptr) {
51                 node->right = new Node(wid);
52                 killusl++;
53             } else {
54                 insertRecursively(node->right, wid);
55             }
56         }
57     }
58
59     bool containsRecursively(Node* node, T wid) {
60         if (node == nullptr)
61             return false;
62         if (wid == node->wid)
63             return true;
64         if (wid < node->wid)
65             return containsRecursively(node->left, wid);
66         return containsRecursively(node->right, wid);
67     }
68
69     void printTree(Node* node, ostream& os) const {
70         if (node != nullptr) {
71             printTree(node->left, os);
72             os << node->wid << " ";
73             printTree(node->right, os);
74         }
75     }
76
77 public:

```

```

7 public:
8     binaryTree() : kor(nullptr), killusl(0) {}
9
10
11
12 void insert(T wid) {
13     if (kor == nullptr) {
14         kor = new Node(wid);
15         killusl++;
16     } else {
17         insertRecursively(kor, wid);
18     }
19 }
20
21 bool contains(T wid) {
22     return containsRecursively(kor, wid);
23 }
24
25 int getSize() const {
26     return killusl;
27 }
28
29 friend ostream& operator<<(ostream& os, const binaryTree& tree) {
30     tree.printTree(tree.kor, os);
31     return os;
32 }
33 };
34
35 int main() {
36     int Q;
37     cin >> Q;
38     binaryTree<int> tree;
39
40     for (int i = 0; i < Q; i++) {
41         string option;
42         cin >> option;
43         Operation operation = getOperation(option);
44
45         switch (operation) {
46             case INSERT: {
47                 int wid;
48                 cin >> wid;
49                 tree.insert(wid);
50                 break;
51             }
52             case SIZE: {
53                 cout << tree.getSize() << endl;
54                 break;
55             }
56             case CONTAINS: {
57                 int wid;
58                 cin >> wid;
59                 cout << (tree.contains(wid) ? "Yes" : "No") << endl;
60                 break;
61             }
62             case PRINT: {
63                 cout << tree << endl;
64                 break;
65             }
66             default:
67                 break;
68         }
69     }
70 }

```

Class Practice Task

Time expected: 2 h

Time spent: 3.5h

Задача №1 - Реверс списку (Reverse list)

Реалізувати метод реверсу списку: Node reverse(Node *head);*

Умови задачі:

- використовувати цілочисельні значення в списку;
- реалізувати метод реверсу;
- реалізувати допоміжний метод виведення вхідного і обернутого списків;

Задача №2 - Порівняння списків

```
bool compare(Node *h1, Node *h2);
```

Умови задачі:

- використовувати цілочисельні значення в списку;
 - реалізувати функцію, яка ітеративно проходить по обох списках і порівнює дані в кожному вузлі;
- якщо виявлено невідповідність даних або якщо довжина списків різна (один список закінчується раніше іншого), функція повертає **false**.

Задача №3 – Додавання великих чисел

```
Node* add(Node *n1, Node *n2);
```

Умови задачі:

- використовувати цифри від 0 до 9 для значень у списку;
- реалізувати функцію, яка обчислює суму двох чисел, які збережено в списку; молодший розряд числа записано в голові списка (напр. 379 \Rightarrow 9 \rightarrow 7 \rightarrow 3);
- функція повертає новий список, передані в функцію списки не модифікуються.

Задача №4 - Віддзеркалення дерева

```
TreeNode *create_mirror_flip(TreeNode *root);
```

Умови задачі:



- використовувати цілі числа для значень у вузлах дерева
- реалізувати функцію, що проходить по всіх вузлах дерева і міняє місцями праву і ліву вітки дерева
- функція повертає нове дерево, передане в функцію дерево не модифікується

Задача №5 - Записати кожному батьківському вузлу суму підвузлів

```
void tree_sum(TreeNode *root);
```

Умови задачі:

- використовувати цілочисельні значення у вузлах дерева;
- реалізувати функцію, яка ітеративно проходить по бінарному дереві і записує у батьківський вузол суму значень підвузлів
- вузол-листок не змінює значення
- значення змінюються від листків до кореня дерева

> Users > Eugene > Desktop > epic_6 >  practice_work_team_tasks_eugenie_stefanovich.cpp >  compare(node *, node *)

```
1  #include <iostream>
2  #include <cstring>
3  #include <vector>
4
5  using namespace std;
6
7  struct node {
8      int data;
9      node* next;
10 };
11
12 struct Node {
13     int data;
14     Node* left;
15     Node* right;
16
17     Node(int value) {
18         data = value;
19         left = nullptr;
20         right = nullptr;
21     }
22 };
23
24 void createll(node*& head, vector<int>& vec) {
25     cout << "Создание списка..." << endl;
26     node* tail = nullptr;
27
28     for (int i = 0; i < vec.size(); i++) {
29         node* newNode = new node;
30         newNode->data = vec[i];
31         newNode->next = nullptr;
32
33         if (head == nullptr) {
34             head = newNode;
35             tail = newNode;
36         } else {
37             tail->next = newNode;
38             tail = newNode;
39         }
40     }
41 }
42
43 void reversell(node*& head){
44     node* current = head;
45     node* next = nullptr;
46     node* prev = nullptr;
47
48     while(current != nullptr){
49         next = current->next;
50         current->next = prev;
51         prev = current;
52         current = next;
53     }
54     head = prev;
55 }
56
57 void printll(node* head) {
58     cout << "Друк списка..." << endl;
59
60     if(head == nullptr){
61         cout << "Список пустой." << endl;
62         return;
63     }
64
65     node* tmp = head;
66     while(tmp != nullptr){
67         cout << tmp->data << " ";
68         tmp = tmp->next;
69     }
70     cout << endl;
71 }
72
73 bool compare(node* h1, node* h2){
74     node* tmp1 = h1;
75     node* tmp2 = h2;
76
77     while(tmp1 != nullptr && tmp2 != nullptr){
78         if(tmp1->data != tmp2->data) return false;
```



```

5
6
7 while(tmp1 != nullptr && tmp2 != nullptr){
8     if(tmp1->data != tmp2->data) return false;
9     tmp1 = tmp1->next;
10    tmp2 = tmp2->next;
11 }
12 if(tmp1 != tmp2) return false;
13 return true;
14 }
15

```

```

16 node* add(node* h1, node* h2){
17     int a = 0;
18     int sum, x, y;
19     node* h = nullptr;
20     node* current = nullptr;
21
22     while(h1 != nullptr || h2 != nullptr){
23         if(h1 == nullptr){
24             x = 0;
25         }else{
26             x = h1->data;
27         }
28         if(h2 == nullptr){
29             y = 0;
30         }else{
31             y = h2->data;
32         }
33
34         sum = x + y + a;
35         a = sum / 10;
36         node* tmp = new node;
37         tmp->data = sum % 10;
38         tmp->next = nullptr;
39
40         if(h == nullptr){
41             h = tmp;
42             current = tmp;
43         }else{
44             current->next = tmp;
45             current = tmp;
46         }
47
48         if(h1 != nullptr) h1 = h1->next;
49         if(h2 != nullptr) h2 = h2->next;
50     }
51
52     if (a > 0) {
53         node* tmp = new node;
54         tmp->data = a;
55         tmp->next = nullptr;
56         current->next = tmp;
57     }
58
59     return h;
60 }
61

```

```

62 void freeLL(node*& head){
63     while(head != nullptr){
64         node* tmp = head;
65         head = head->next;
66         delete tmp;
67     }
68 }
69

```

```

70 void insertIntoTree(Node*& root, int value){
71     if(root == nullptr) {
72         root = new Node(value);
73         return;
74     }
75
76     if(value < root->data){
77         insertIntoTree(root->left, value);
78     }else{
79         insertIntoTree(root->right, value);
80     }
81 }
82

```



```

148         }else{
149             insertIntoTree(root->right, value);
150         }
151     }
152
153     Node* mirror(Node* root){
154         if(root == nullptr){
155             return nullptr;
156         }
157
158         Node* mirroredNode = new Node(root->data);
159
160         mirroredNode->left = mirror(root->right);
161         mirroredNode->right = mirror(root->left);
162
163         return mirroredNode;
164     }
165
166     void sumTree(Node* root){
167         if(root == nullptr) return;
168
169         sumTree(root->left);
170         sumTree(root->right);
171
172         if(root->left != nullptr || root->right != nullptr){
173             int leftValue = (root->left != nullptr) ? root->left->data : 0;
174             int rightValue = (root->right != nullptr) ? root->right->data : 0;
175             root->data = leftValue + rightValue;
176         }
177     }
178
179     void printTree(Node* root){
180         if(root == nullptr) return;
181         cout << root->data << " ";
182         printTree(root->left);
183         printTree(root->right);
184     }
185
186     void freeTree(Node*& root){
187         if(root == nullptr) return;
188
189         freeTree(root->left);
190         freeTree(root->right);
191         delete root;
192     }
193
194     int main() {
195         vector<int> vec1 = {3, 7, 9};
196         vector<int> vec2 = {2, 4, 8};
197
198         node* head1 = nullptr;
199         node* head2 = nullptr;
200
201         cout << "-----" << endl;
202         cout << "          LINKED LISTS" << endl;
203
204         createLL(head1, vec1);
205         cout << "Перший список: ";
206         printLL(head1);
207
208         createLL(head2, vec2);
209         cout << "Другий список: ";
210         printLL(head2);
211
212         node* result = add(head1, head2);
213         cout << "Результат додавання: ";
214         printLL(result);
215
216         reverseLL(head1);
217         cout << "Перший список після реверсу: ";
218         printLL(head1);
219
220         reverseLL(head2);

```

```

    printLL(result);

    reversell(head1);
    cout << "Перший список після реверсу: ";
    printLL(head1);

    reversell(head2);
    cout << "Другий список після реверсу: ";
    printLL(head2);

    bool isEqual = compare(head1, head2);
    cout << "Чи рівні списки: " << (isEqual ? "Так" : "Ні") << endl;
    freeLL(head1);
    freeLL(head2);

    cout << "-----" << endl;
    cout << "                TREES" << endl;

    Node* root = nullptr;
    vector<int> treeValues = {4, 2, 6, 1, 3, 5, 7};
    int len = treeValues.size();
    for(int i = 0; i < len; i++){
        insertIntoTree(root, treeValues[i]);
    }
    cout << "Оригінальне дерево (виведене по рівнях): ";
    printTree(root);
    cout << endl;

    sumTree(root);
    cout << "Дерево після обчислення сум підвузлів: ";
    printTree(root);
    cout << endl;

    Node* mirroredRoot = mirror(root);
    cout << "Дерево після дзеркального відображення: ";
    printTree(mirroredRoot);
    cout << endl;

    freeTree(root);
    freeTree(mirroredRoot);
}

```

```

-----
                        LINKED LISTS
Створення списку...
Перший список: Друк списку...
3 7 9
Створення списку...
Другий список: Друк списку...
2 4 8
Результат додавання: Друк списку...
5 1 8 1
Перший список після реверсу: Друк списку...
9 7 3
Другий список після реверсу: Друк списку...
8 4 2
Чи рівні списки: Ні
-----
                        TREES
Оригінальне дерево (виведене по рівнях): 4 2 1 3 6 5 7
Дерево після обчислення сум підвузлів: 16 4 1 3 12 5 7
Дерево після дзеркального відображення: 16 12 7 5 4 3 1
PS C:\Users\Eugene\Desktop\epic_6\output>

```

Practice# programming: Self Practice Task

Time expected: 30min

Time spent: 20 min

Верховна Рада

Обмеження: 2 сек., 256 МБ

Вже не перший рік у нашому суспільстві гостро стоїть питання кількості народних депутатів у Верховній Раді. Утримувати дуже багато дармоїдів-депутатів народ не хоче, проте все має бути справедливо і кожна політична партія повинна отримати кількість місць у парламенті пропорційну до кількості голосів, що вона отримала на виборах. Більш формально, відношення кількості голосів до кількості місць у Верховній Раді має бути однаковим для усіх партій.

В останніх виборах до Верховної Ради взяли участь n партій. Вам відома кількість голосів, що отримала кожна з них. Допоможіть народові мінімізувати загальну кількість депутатських місць.

Вхідні дані

У першому рядку задано одне натуральне число n — кількість партій.

У другому рядку задано n натуральних чисел a_i — кількість голосів, що отримала i -та партія на останніх виборах.

Вихідні дані

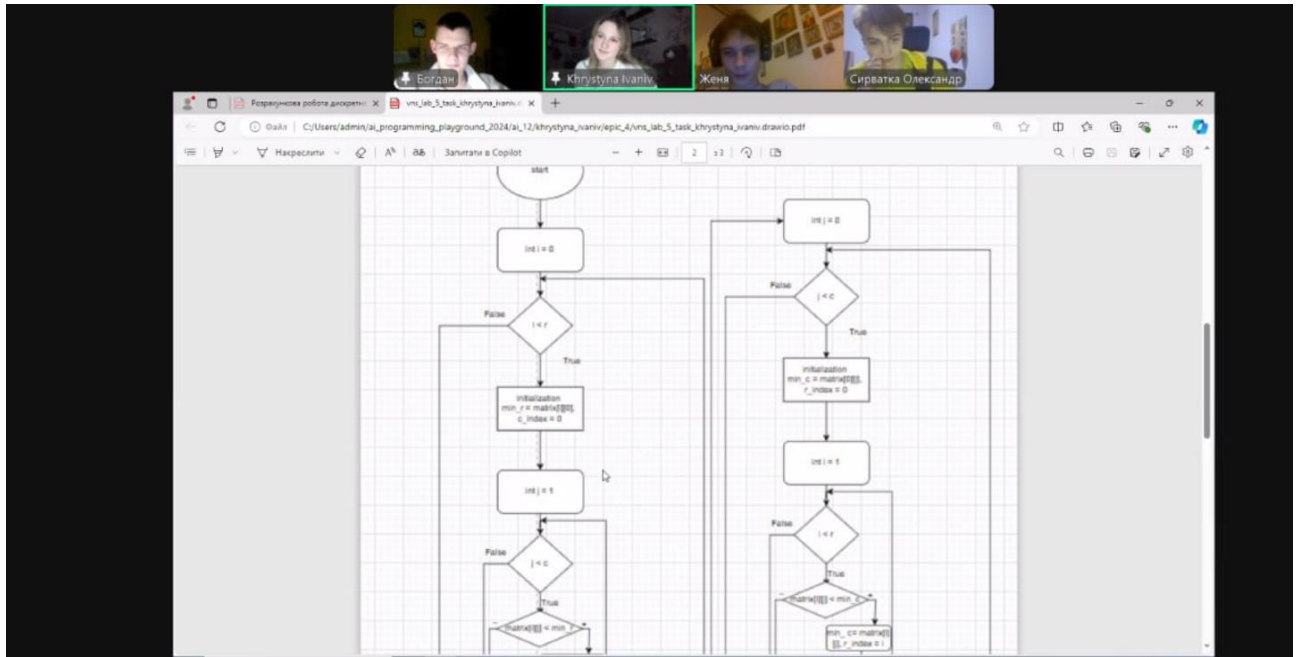
У єдиному рядку виведіть одне ціле число — мінімальну загальну кількість депутатських місць у парламенті.

~

C: > Users > Eugene > Desktop > epic_6 > G: practice_work_self_algotester_tasks_eugenie_stefanovich.cpp >

```
1  √ #include <iostream>
2  #include <vector>
3  #include <numeric>
4
5  using namespace std;
6
7  √ int main() {
8      int n ;
9      long sum = 0;
10     cin >> n;
11
12     vector<int> base(n);
13     √ for (int i = 0; i < n; i++) {
14         |     cin >> base[i];
15     }
16
17
18     int gcd = base[0];
19     √ for (int i = 1; i < n; i++) {
20         |     gcd = std::gcd(gcd, base[i]);
21     };
22     }
23     for(int i = 0; i < n; i++)
24         sum += base[i];
25
26     cout << endl << sum / gcd;
27     return 0;
28 }
29
```

Meet:



Висновок:

Завдяки цій роботі я на практиці зрозумів, як працюють основні динамічні структури даних та їхні ключові операції, і тепер усвідомлюю, чому їх використовують у завданнях, що вимагають гнучкого управління пам'яттю. Я також засвоїв відмінності між статичним та динамічним виділенням пам'яті, що дає мені краще уявлення про ефективне управління ресурсами. Практичні вправи з такими структурами, як стек, черга, зв'язний список і дерево, дозволили мені глибше зрозуміти принципи обробки даних у пам'яті, а також основи алгоритмів, що застосовуються для роботи з динамічними структурами.