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

Львів 2024

Тема роботи:

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

Мета роботи:

Освоєння основних принципів робити 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 Memmory 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). Сформувати 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 куль і для кожної читають магічне заклинання, 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 a 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

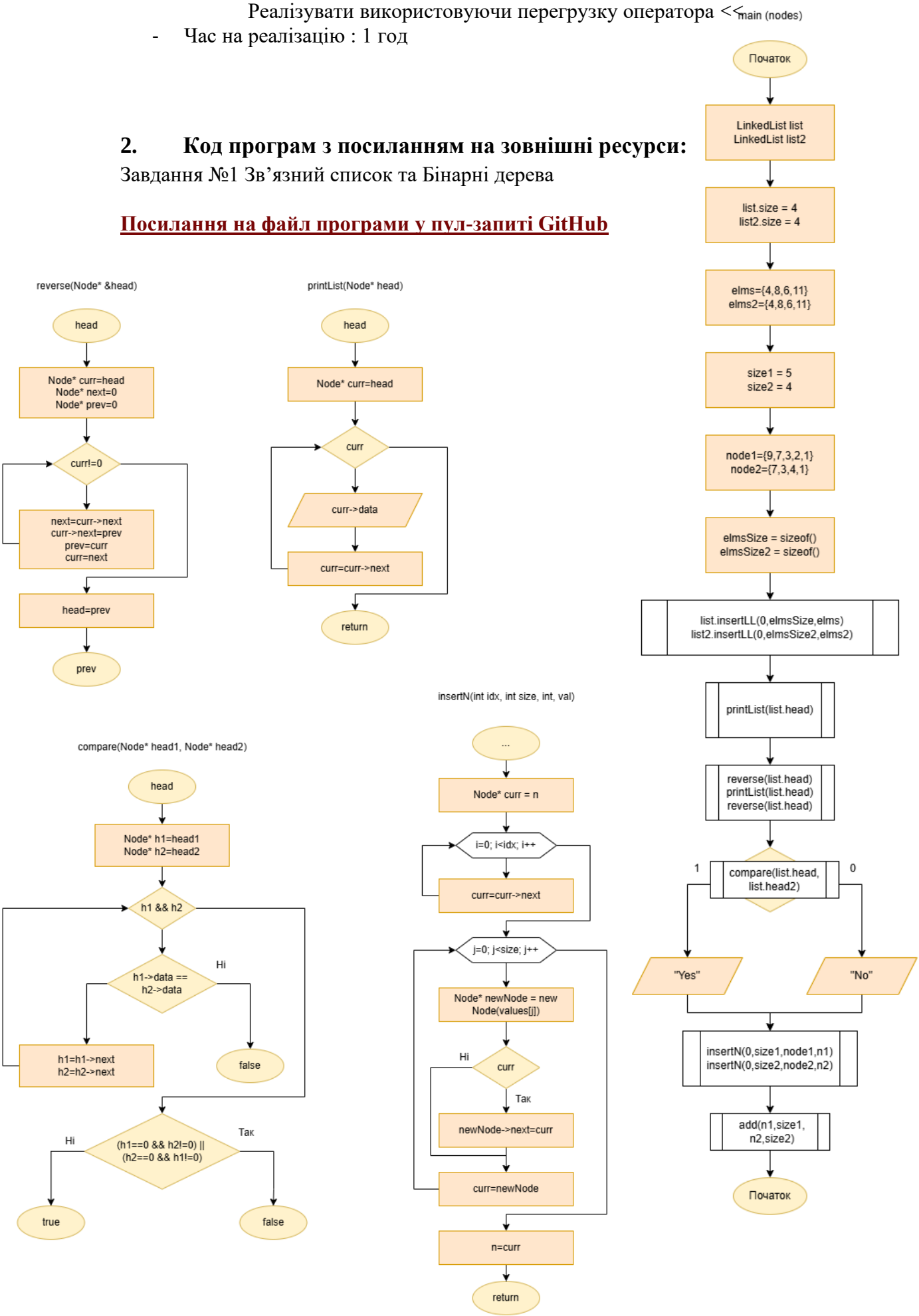
Ви не отримуєте аргументів.
Ви виводите усі елементи дерева через пробіл.
Реалізувати використовуючи перегрузку оператора <<

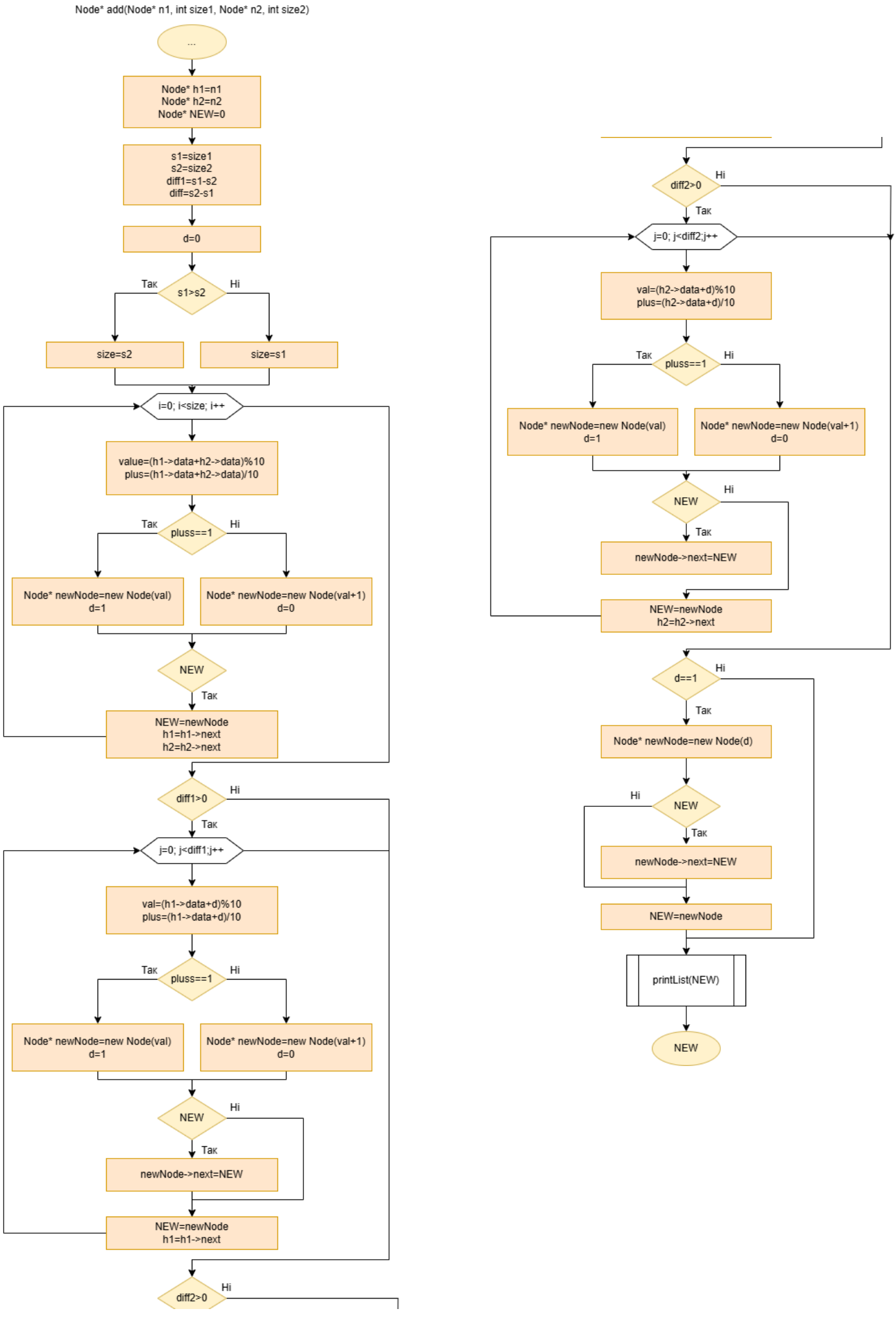
- Час на реалізацію : 1 год

2. Код програм з посиланням на зовнішні ресурси:

Завдання №1 Зв’язний список та Бінарні дерева

Посилання на файл програми у пул-запиті GitHub




Node* add(Node* n1, int size1, Node* n2, int size2)
...
Node* h1=n1
Node* h2=n2
Node* NEW=0
s1=size1
s2=size2
diff1=s1-s2
diff=s2-s1
d=0
Так
s1>s2
Hi
size=s2
size=s1
i=0; i<size; i++
value=(h1->data+h2->data)%10
plus=(h1->data+h2->data)/10
Так
pluss==1
Hi
Node* newNode=new Node(val)
d=1
Node* newNode=new Node(val+1)
d=0
NEW
Так
NEW=newNode
h1=h1->next
h2=h2->next
diff1>0
Hi
Так
j=0; j<diff1;j++
val=(h1->data+d)%10
plus=(h1->data+d)/10
Так
pluss==1
Hi
Node* newNode=new Node(val)
d=1
Node* newNode=new Node(val+1)
d=0
NEW
Hi
Так
newNode->next=NEW
NEW=newNode
h1=h1->next
diff2>0
Hi
Так
j=0; j<diff2;j++
val=(h2->data+d)%10
plus=(h2->data+d)/10
Так
pluss==1
Hi
Node* newNode=new Node(val)
d=1
Node* newNode=new Node(val+1)
d=0
NEW
Hi
Так
newNode->next=NEW
NEW=newNode
h2=h2->next
d==1
Hi
Так
Node* newNode=new Node(d)
Hi
NEW
Так
newNode->next=NEW
NEW=newNode
printList(NEW)
NEW

```cpp
#include <iostream>

class Node {
public:
    int data;
    Node* next;

    Node(int value) {
        data = value;
        next = nullptr;
    }
};



class LinkedList {

    public:
    Node* head;
    int size;
    LinkedList() {
        size = 0;
        head = nullptr;
    }
    void insertLL(int index, int listSize, int values[]) {
        if (index < 0 || index > size || listSize <= 0) {
            return;
        }
        Node* current = head;
        for (int i = 0; i < index; i++) {
            current = current->next;
        }
        for (int j = 0; j < listSize; j++) {
            Node* newNode = new Node(values[j]);
            if (current) {
                newNode->next = current;
            }
            current = newNode;
            size++;
        }
        head = current;
    }


};

// реверс списку
Node* reverse(Node* &head){
    Node* current = head;
    Node* next = nullptr;
    Node* prev = nullptr;
    while(current != nullptr){
        next = current->next;
        current->next = prev;
        prev = current;
        current = next;
    }
    head = prev;
    return prev;
}


// друк списку
void printList(Node* head) {
    Node* current = head;
    while(current) {
        std::cout << current->data << " ";
        current = current->next;
    }
    std::cout << std::endl;
}
```

```cpp
// порівняння списків
bool compare(Node* h1,  Node* h2){
    Node* head1 = h1;
    Node* head2 = h2;
    while(head1 && head2) {
        if(head1->data == head2->data){
            head1 = head1->next;
            head2 = head2->next;
        } else {
            return false;
        }
    }
    if((head2==nullptr && head1!=nullptr) || (head2!=nullptr && head1==nullptr)){
        return false;
    }

    return true;
}


//створення нодів
void insertN(int index, int listSize, int values[], Node* &n) {
        Node* current = n;
        for (int i = 0; i < index; i++) {
            current = current->next;
        }
        for (int j = 0; j < listSize; j++) {
            Node* newNode = new Node(values[j]);
            if (current) {
                newNode->next = current;
            }
            current = newNode;
        }
        n = current;
    }
// додавання
Node* add(Node* n1, int size1, Node* n2, int size2){
    Node* head1 = n1;
    Node* head2 = n2;
    Node* NEW = nullptr;
    int s1 = size1;
    int s2 = size2;
    int diff1 = s1-s2;
    int diff2 = s2-s1;
    int d=0;
    int size;
    if(s1>s2){
        size = s2;
    } else {
        size = s1;
    }
    for(int i = 0; i<size; i++){
        int value = (head1->data + head2->data)%10;
        int plus= (head1->data + head2->data)/10;
        Node* newNode = new Node(value);
        if(plus == 1){
            Node* newNode = new Node(value);
            d=1;
        } else {
            Node* newNode = new Node(value+1);
            d=0;
        }
        if (NEW) {
            newNode->next = NEW;
        }
        NEW = newNode;
        head1 = head1->next;
        head2 = head2->next;
    }
    if(diff1>0){
        for(int j=0; j<diff1; j++){
            int value = (head1->data+d)%10;
            int plus =(head1->data+d)/10;
            Node* newNode = new Node(value);
            if(plus==1){
                Node* newNode = new Node(value);
                d=1;
            } else {
                Node* newNode = new Node(value+d);
                d=0;
            }
            if (NEW) {
                newNode->next = NEW;
            }
            NEW = newNode;
            head1 = head1->next;
        }
```

```
    if(diff2>0){
        for(int j=0; j<diff2; j++){
            int value = (head2->data+d)%10;
            int plus =(head2->data+d)/10;
            Node* newNode = new Node(value);
            if(plus==1){
                Node* newNode = new Node(value);
                d=1;
            } else {
                Node* newNode = new Node(value+d);
                d=0;
            }
            if (NEW) {
                newNode->next = NEW;
            }
            NEW = newNode;
            head2 = head2->next;
        }
    }
    if(d==1){
        Node* newNode = new Node(d);
        if (NEW) {
            newNode->next = NEW;
        }
        NEW = newNode;
    }
    printList(NEW);
    return NEW;
}
int main(){

    LinkedList list;
    LinkedList list2;
    Node* n1;
    Node* n2;
    list.size=4;
    list2.size=4;
    int node1[]={9, 7, 3, 2, 1};
    int size1 = 5;
    int size2 = 4;
    int node2[]={7, 3, 4, 1};
    int elements[] = {4, 8, 6, 11};
    int elements2[] = {4, 8, 6, 11};
    int elementsSize = sizeof(elements)/sizeof(elements[0]);
    int elementsSize2 = sizeof(elements2)/sizeof(elements2[0]);
    list.insertLL(0, elementsSize, elements);
    list2.insertLL(0, elementsSize2, elements2);

    printList(list.head);

    reverse(list.head);
    printList(list.head);
    reverse(list.head);

    std::cout << "Чи однаковими є списки 1 та 2 ? : "
    << ( compare(list.head, list2.head) ? "Yes" : "No")
    << "\n";

    insertN(0, size1, node1, n1);
    insertN(0, size2, node2, n2);

    add(n1, size1, n2, size2);

    return 0;
}
```

```
11 6 8 4
4 8 6 11
Чи однаковими є списки 1 та 2 ? : Yes
1 0 4 6 6 2
```

```cpp
#include <iostream>


struct TreeNode {
    int data;
    TreeNode* left;
    TreeNode* right;

    TreeNode(int value): data(value), left(nullptr), right(nullptr) {}
};

struct BinarySearchTree {

public:
    BinarySearchTree() : root(nullptr) {}

    void insert(int value) {
        root = insert(root, value);
    }

    void inorderTraverse() {
        inorderTraverse(root);
    }


    void create_mirror_flip(){
        inorderTraverse(create_mirror_flip(root));
    }

    void tree_sum(){
        tree_sum(root);
    }
private:
    TreeNode* root;

    TreeNode* insert(TreeNode* node, int value) {
        if (node == nullptr) {
            return new TreeNode(value);
        }

        if (value < node->data) {
            node->left = insert(node->left, value);
        } else if (value > node->data) {
            node->right = insert(node->right, value);
        }

        return node;
    }

    void inorderTraverse(TreeNode* node) {
        if (node != nullptr) {
            inorderTraverse(node->left);
            std::cout << node->data << " ";
            inorderTraverse(node->right);
        }
    }

    TreeNode* create_mirror_flip(TreeNode* node){
        if(node == nullptr){
            return nullptr;
        }
        TreeNode* NewNode = new TreeNode(node->data);
        NewNode->left = create_mirror_flip(node->right);
        NewNode->right = create_mirror_flip(node->left);

        return NewNode;
    }
    void tree_sum(TreeNode* &root){
        if(root == nullptr){
            return;
        }
        tree_sum(root->left);
        tree_sum(root->right);
        if(root->left && root->right){
            root->data = root->left->data + root->right->data;
        }
        if(root->left && !root->right){
            root->data = root->left->data;
        }
        if(!root->left && root->right){
            root->data = root->right->data;
        }
    }
};
```

```cpp
int main(){


    BinarySearchTree myTree;

    myTree.insert(4);
    myTree.insert(2);
    myTree.insert(5);
    myTree.insert(1);
    myTree.insert(6);
    myTree.insert(3);

    std::cout << "The tree : ";
    myTree.inorderTraverse();
    std::cout << "\n";


    std::cout << "Flipped : ";
    myTree.create_mirror_flip();
    std::cout << "\n";


    std::cout << "The sum of dauthers of node : ";
    myTree.tree_sum();
    myTree.inorderTraverse();
    std::cout << "\n";


    return 0;
}
```

```
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

```cpp
#include <iostream>
#include <stdio.h>
#include <fstream>
#include <cstring>



class Node {
public:
    char* data;
    Node* prev;
    Node* next;

    Node(char* &value) {
        data = value;
        prev=nullptr;
        next = nullptr;
    }
};


class LinkedList {

public:
    LinkedList() {
        size = 0;
        head = nullptr;
        tail=nullptr;
    }
```

```
// Створення списку.
    void create(LinkedList *list){
        int n = 4;
        size = n;
        char* elements[n] = {"sgyd", "lnaswer", "nskjdhdi", "aaaaaaa"};
        for(int i = 0; i < n; i++){
            char* value = elements[i];
        Node* newNode = new Node(value);
        newNode->data = strdup(value);
        newNode->next = nullptr;

        if(head == NULL){
            newNode->prev = nullptr;
            head = newNode;
        }else{
            Node* tmp = head;
            while(tmp->next != nullptr){
                tmp = tmp->next;
            }

            tmp->next = newNode;
            newNode->prev = tmp;
        }
    }
    }
void insert(char index_char, char* value) {
    Node* current = head;
    int index = size+2;

    for (int i = 0; i < size; i++) {
        if (current && *(current->data) == index_char) {
            index = i+1;
            break;
        }
        if (current) {
            current = current->next;
        }
    }
    if(index == size+2) return;

    Node* newNode = new Node(value);
    if (index == size+1) {
        if (tail) {
            tail->next = newNode;
        }
        newNode->prev = tail;
        tail = newNode;
        if (!head) {
            head = newNode;
        }
    } else { // Вставка в середину списку
        Node* current = head;
        for (int i = 0; i < index - 1; i++) { // Знайти елемент перед позицією вставки
            current = current->next;
        }
        Node* nextNode = current->next;
        current->next = newNode;
        newNode->prev = current;
        newNode->next = nextNode;
        if (nextNode) {
            nextNode->prev = newNode;
        }
    }

    size++; // Збільшити розмір списку
}
```

```cpp
// Знищення елемента зі списку.
    void erase(int K){
        if (K <= 0 || K > size) {
            return;
        }
        Node* current = head;
        for (int i = 0; i < K; i++) {
            if (!current) break;
            Node* nextNode = current->next;
            delete current;
            current = nextNode;
            head = current;
            size--;
        }
        head = current;
        if (head) {
            head->prev = nullptr;
        }
    }


// Друк списку.
    void print(){
        Node* current = head;
        if(current==nullptr){
            std::cerr << "Немає елементів у списку\n";
            return;
        }
        while(current) {
            std::cout << current->data << " ";
            current = current->next;
        }
        std::cout << std::endl;
    }
// Запис списку у файл.
    void write_to_file(const char* name){
        std::ofstream f(name);
        if (!f.is_open()) {
            std::cerr << "Не вдалося відкрити файл для запису" << std::endl;
            return;
        }
        Node* current = head;
        while(current) {
            f << current->data << "\n";
            current = current->next;
        }
        f.close();
    }


// Знищення списку.
    void deleteList(LinkedList **list){
        Node *tmp = (*list)->head;
        Node *next = NULL;
        while (tmp) {
            next = tmp->next;
            delete tmp;
            tmp = next;
            head = tmp;
            size--;
        }
    }
// Відновлення списку з файлу.
    LinkedList from_file(const char* name){
        std::ifstream f(name);
        if(!name){
            std::cerr << "Error opening file";
            return *this;
        }
        char string[50];
        while(f >> string){
            char* value = string;
            Node* newNode = new Node(value);
            newNode->data = strdup(value);
            newNode->next = nullptr;

            if(head == NULL){
                newNode->prev = nullptr;
                head = newNode;
            }else{
                Node* tmp = head;
                while(tmp->next != nullptr){
                    tmp = tmp->next;
                }
                tmp->next = newNode;
                newNode->prev = tmp;
            }
        }
```

```
        return *this;
    }

    friend std::ostream& operator<<(std::ostream& os, const LinkedList& list) {
        Node* current = list.head;

        while(current) {
            os << current->data << " ";
            current = current->next;
        }
        os << std::endl;
        return os;
    }

private:
    Node* head;
    Node* tail;
    int size;
};
int main(){
    const char* name = "File with list. txt";
    LinkedList* l = new LinkedList();

    l->create(l);
    //std::cout << "" << list;
    l->print();

    std::cout << "Введіть елемент, який хочете ввести : ";
    char value[50];
    scanf("%s", value);
    while(getchar() != '\n');
    std::cout << "Введіть символ, за яким знайти елемент : ";
    char index_char;
    scanf("%c", &index_char);
    while(getchar() != '\n');
    l->insert(index_char, value);
    l->print();

    std::cout << "Введіть кількість елементів, які потрібно видалити із початку списку: ";
    int K;
    std::cin >> K;
    while(getchar() != '\n');
    l->erase(K);
    l->print();

    std::cout << "Видалений список:\n";
    l->write_to_file(name);
    l->deleteList(&l);
    l->print();

    std::cout << "Відновлений список з файлу:\n";
    l->from_file(name);
    l->print();

    return 0;
}
```

```
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

```cpp
#include <iostream>
#include <vector>
#include <cmath>
#include <cinttypes>
#include <cstdio>

int main(){
    unsigned long long a, x;
    uint64_t b=0;
    int N, r=0, c=0;
    std::vector<bool> row(8, 0);
    std::vector<int> answer;
    std::vector<std::vector<bool>> board(8, row);
    std::cin >> a;
    std::cin >> N;
    std::vector<int> R(N), C(N);
    for(int i=0; i<N; i++){
        std::cin >> R[i] >> C[i];
        R[i]--;
        C[i]--;
    }

    while(a>0){
        board[r][c]=a%2;
        a/=2;
        c++;
        if(c==8){
            c = 0;
            r++;
        }
    }

    for(int n=0; n<N; n++){
        for(int i=0; i<8; i++){
            if(board[R[n]][i]==1){
                board[R[n]][i]=0;
            } else {
                board[R[n]][i]=1;
            }
        }
        for(int i=0; i<8; i++){
            if(board[i][C[n]]==1){
                board[i][C[n]]=0;
            } else {
                board[i][C[n]]=1;
            }
        }
        if(board[R[n]][C[n]]==1){
            board[R[n]][C[n]]=0;
        } else {
            board[R[n]][C[n]]=1;
        }
    }

    for(int i=0; i<8; i++){
        for(int j=0; j<8; j++){
            if(board[i][j]==1){
                x = pow(2, j+i*8);
                b+= x;
            }
        }
    }

    std::cout << b;

    return 0;
}
```

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

```cpp
#include <iostream>
#include <string>



struct TreeNode {
    int data;
    TreeNode* left;
    TreeNode* right;

    TreeNode(int value): data(value), left(nullptr), right(nullptr) {}
};

struct BinarySearchTree {

public:
    BinarySearchTree() : root(nullptr) {}

    void insert(int value) {
        root = insert(root, value);
    }

    void print() {
        print(root);
    }

    bool contains(int value) {
        return contains(root, value);
    }
    int size(){
        return size(root);
    }

private:
    TreeNode* root;
    TreeNode* insert(TreeNode* node, int value) {
        if (node == nullptr) {
            return new TreeNode(value);
        }

        if (value < node->data) {
            node->left = insert(node->left, value);
        } else if (value > node->data) {
            node->right = insert(node->right, value);
        }

        return node;
    }

    bool contains(TreeNode* node, int value) {
        if (node == nullptr) {
            return false;
        }

        if (value == node->data) {
            return true;
        } else if (value < node->data) {
            return contains(node->left, value);
        } else {
            return contains(node->right, value);
        }
    }

    void print(TreeNode* node) {
        if (node != nullptr) {
            print(node->left); // LEFT
            std::cout << node->data << " "; // CENTER
            print(node->right);  // RIGHT
```

```
    int size(TreeNode* node) {
        int s = 0;
        if(node != nullptr){
            s= 1 + size(node->left) + size(node->right);
        } else {
            return s;
        }
        return s;
    }
};

int main() {
    BinarySearchTree myTree;
    int N;
    std::string operation;


    std::cin >> N;
    for(int i=0; i<N; i++){
        std::cin >> operation;
        if(operation =="insert"){
            int a;
            std::cin >> a;
            myTree.insert(a);
        }
        if(operation == "print"){
            myTree.print();
            std::cout << std::endl;
        }
        if(operation == "contains" ){
            int a;
            std::cin >> a;
            if (myTree.contains(a)){
                std::cout << "Yes"
                << std::endl;
            } else {
                std::cout << "No"
                << std::endl;
            }


        }
        if(operation == "size"){
            std::cout << myTree.size() << std::endl;
        }
    }
    return 0;
}
```

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.