**Державний вищий навчальний заклад**

**Ужгородський національний університет**

**Факультет інформаційних технологій**

**ПРАКТИЧНА РОБОТА № 2**

**Тема:** Односпрямовані та двоспрямовані списки

Виконала студентка 1 курсу спеціальності «Інженерія програмного забезпечення» Росада Марина Іванівна

Ужгород-2025

**Мета:** набути практичних навичок створення та застосування односпрямованих та двоспрямованих списків

**Завдання до роботи:**

1. Написати програму, у якій потрібно створити лінійний односпрямований список та виконати дії згідно з власним варіантом завдання табл. 1.1.
2. Написати програму, у якій потрібно створити двоспрямований список та виконати дії згідно з власним варіантом завдання табл 1.2.
3. Оформити звіт про виконання роботи та завантажити його в системі електронного навчання ДВНЗ «УжНУ» в установлений термін.
4. Підготувати відповіді на питання до лекції з односпрямованого та двоспрямованого списку.

**Хід роботи**

**Варіант 14**

**Завдання 1**

Створити лінійний односпрямований список із дійсних чисел. Визначити максимальне з елементів списку та включити його після кожного елемента, який має значення 1.

Код програми:

class Node {

constructor(data) {

this.data = data;

this.next = null;

}

}

class LinkedList {

constructor() {

this.head = null;

this.size = 0;

}

add(data) {

const newNode = new Node(data);

if (this.head === null) {

this.head = newNode;

} else {

let current = this.head;

while (current.next !== null) {

current = current.next;

}

current.next = newNode;

}

this.size++;

}

findMax() {

if (!this.head) return null;

let maxVal = this.head.data;

let current = this.head.next;

while (current) {

if (current.data > maxVal) {

maxVal = current.data;

}

current = current.next;

}

return maxVal;

}

insertAfterOnes(maxVal) {

let current = this.head;

while (current) {

if (current.data === 1) {

const newNode = new Node(maxVal);

newNode.next = current.next;

current.next = newNode;

current = newNode.next; // Пропускаємо вставлений вузол

} else {

current = current.next;

}

}

}

printList() {

let current = this.head;

let result = "";

while (current) {

result += current.data + ", ";

current = current.next;

}

return result.trim();

}

}

function main() {

const list = new LinkedList();

list.add(1);

list.add(3.5);

list.add(2);

list.add(1);

list.add(4.2);

list.add(1);

console.log("Початковий список:", list.printList());

const maxVal = list.findMax();

if (maxVal !== null) {

list.insertAfterOnes(maxVal);

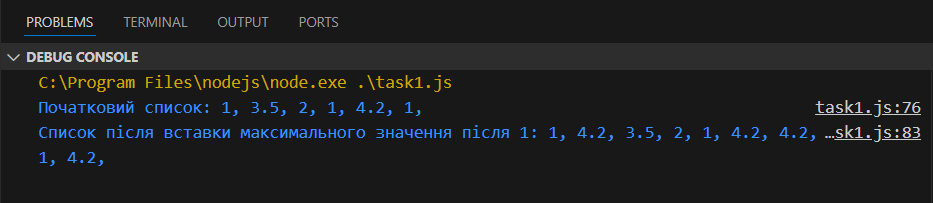
}

console.log("Список після вставки максимального значення після 1:", list.printList());

}

main();

Скріншот роботи програми:



**Завдання 2**

Дано покажчики Px і Py на два різних елементи двозв'язного списку; елемент з адресом Px знаходиться у списку поперед елемента з адресом Py, але не обов'язково поряд з ним. Перемістити елементи, які знаходяться поміж даними елементами (не долучаючи дані елементи) у новий список (в тому самому порядку). Вивести покажчик на перші елементи перетвореного та нового списків. якщо новий список стане порожнім, то пов'язаний з ним покажчик буде дорівнювати null.

Код програми:

class Node {

constructor(data) {

this.data = data;

this.next = null;

this.prev = null;

}

}

class DoublyLinkedList {

constructor() {

this.head = null;

this.tail = null;

this.size = 0;

}

add(data) {

const newNode = new Node(data);

if (this.head === null) {

this.head = newNode;

this.tail = newNode;

} else {

this.tail.next = newNode;

newNode.prev = this.tail;

this.tail = newNode;

}

this.size++;

}

printList() {

let current = this.head;

let result = "";

while (current) {

result += current.data + " ";

current = current.next;

}

return result.trim();

}

extractSublist(px, py) {

if (!px || !py || px === py) return null;

let newHead = null, newTail = null;

let current = px.next;

while (current && current !== py) {

let nextNode = current.next;

// Від'єднуємо вузол від основного списку

if (current.prev) {

current.prev.next = nextNode;

}

if (nextNode) {

nextNode.prev = current.prev;

}

// Додаємо у новий список

if (!newHead) {

newHead = newTail = current;

newHead.prev = null;

newHead.next = null;

} else {

newTail.next = current;

current.prev = newTail;

newTail = current;

newTail.next = null;

}

current = nextNode;

}

return newHead;

}

}

function main() {

const list = new DoublyLinkedList();

[1, 2, 3, 4, 5, 6, 7].forEach(num => list.add(num));

console.log("Початковий список:", list.printList());

let px = list.head.next;

let py = list.tail.prev;

let newHead = list.extractSublist(px, py);

console.log("Перетворений список:", list.printList());

console.log("Новий список:");

let current = newHead;

const newValues = [];

while (current) {

newValues.push(current.data);

current = current.next;

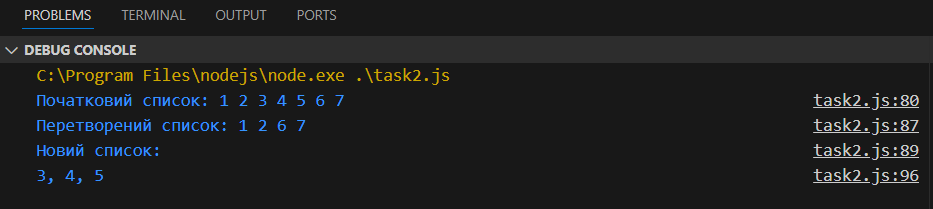
}

console.log(newValues.join(", ") || "null");

}

main();

Скріншот програми:



**Висновок:**

У ході виконання практичної роботи було розглянуто структуру та особливості односпрямованих і двоспрямованих списків, їх основні операції, такі як додавання, видалення та пошук елементів. Було реалізовано алгоритм знаходження максимального елемента списку та його вставлення після кожного елемента зі значенням 1, що дозволило закріпити навички роботи з покажчиками та динамічними структурами даних.

Односпрямовані списки забезпечують ефективне збереження даних із мінімальними витратами пам’яті, проте мають обмеження у доступі до елементів. Двоспрямовані списки, завдяки можливості переміщення в обох напрямках, дозволяють оптимізувати певні операції, але потребують більшої кількості пам’яті.

Також було реалізовано алгоритм, що виконує переміщення елементів двозв’язного списку між двома заданими вузлами у новий список, зберігаючи їхній порядок. Цей підхід дозволяє ефективно змінювати структуру списку та обробляти підмножини елементів без порушення початкових зв’язків. Реалізація цього алгоритму підкреслила важливість правильного використання покажчиків для динамічного керування пам’яттю та структурою даних.

Виконана робота продемонструвала важливість правильного керування покажчиками та звільнення пам’яті, що є критичною умовою для ефективного використання динамічних структур. Отримані знання є важливими для подальшого вивчення структур даних та розробки алгоритмів.