|  |
| --- |
| 1.Write a program in Java to right rotate an array by 5 steps |

class RotateRight {

public static void main(String[] args) {

int[] arr = {1, 2, 3, 4, 5};

int n = 5;

System.out.println("Original Array: ");

for (int i = 0; i < arr.length; i++) {

System.out.print(arr[i] + " ");

}

for (int i = 0; i < n; i++) {

int j, last;

last = arr[arr.length - 1];

for (j = arr.length - 1; j > 0; j--) {

arr[j] = arr[j - 1];

}

arr[0] = last;

}

System.out.println();

System.out.println("Array after right rotation: ");

for (int i = 0; i < arr.length; i++) {

System.out.print(arr[i] + " ");

}

}

}

2.Write a program in Java to find the fourth smallest element in an unsorted list

import java.util.Arrays;

public class FourthSmallest {

public static void main(String[] args) {

int[] arr = { 5, 3, 1, 4, 2 };

int k = 4;

Arrays.sort(arr);

System.out.println("The fourth smallest element is: " + arr[k - 1]);

}

}

3.Write a program in Java to find the sum of n number of elements in the range of L and R where 0 <= L <= R <= n-1

import java.util.Scanner;

public class SumOfElementsInRange {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

System.out.print("Enter the value of n: ");

int n = sc.nextInt();

int[] arr = new int[n];

System.out.println("Enter the elements of the array:");

for (int i = 0; i < n; i++) {

arr[i] = sc.nextInt();

}

System.out.print("Enter the value of L: ");

int L = sc.nextInt();

System.out.print("Enter the value of R: ");

int R = sc.nextInt();

int sum = 0;

for (int i = L; i <= R; i++) {

sum += arr[i];

}

System.out.println("The sum of elements in the range of L and R is: " + sum);

}

}

4.Write a program in Java to multiply two matrices

**Input :** mat1[][] = {{1, 2},

{3, 4}}

mat2[][] = {{1, 1},

{1, 1}}

**Output :** {{3, 3},

{7, 7}}

**Input :** mat1[][] = {{2, 4},

{3, 4}}

mat2[][] = {{1, 2},

{1, 3}}

**Output :** {{6, 16},

{7, 18}}

5.Write a program in Java to delete the first occurrence of a key in a singly linked list

public class LinkedList {

Node head;

static class Node {

int data;

Node next;

Node(int d) {

data = d;

next = null;

}

}

void deleteNode(int key) {

Node temp = head, prev = null;

if (temp != null && temp.data == key) {

head = temp.next;

return;

}

while (temp != null && temp.data != key) {

prev = temp;

temp = temp.next;

}

if (temp == null) {

return;

}

prev.next = temp.next;

}

public void printList() {

Node n = head;

while (n != null) {

System.out.print(n.data + " ");

n = n.next;

}

}

public static void main(String[] args) {

LinkedList list = new LinkedList();

list.head = new Node(1);

list.head.next = new Node(2);

list.head.next.next = new Node(3);

list.head.next.next.next = new Node(4);

list.head.next.next.next.next = new Node(5);

System.out.println("Original Linked list ");

list.printList();

list.deleteNode(3);

System.out.println("\nLinked List after Deletion of 3 ");

list.printList();

}

}

6.Write a program in Java to insert a new element in a sorted circular linked list

// A class to represent a node of the circular linked list

class Node {

int data; // The value of the node

Node next; // The reference to the next node

// A constructor to create a new node with given data

Node(int data) {

this.data = data;

this.next = null;

}

}

// A class to represent the circular linked list

class CircularLinkedList {

Node head; // The reference to the head node

// A method to insert a new element at the end of the list

void insertAtEnd(int data) {

// Create a new node with the given data

Node newNode = new Node(data);

// If the list is empty, make the new node as the head and tail

if (head == null) {

head = newNode;

tail = newNode;

} else {

// Otherwise, find the last node of the list

Node last = head;

while (last.next != null) {

last = last.next;

}

// Make the new node as the next of the last node and update its tail pointer

newNode.next = last.next;

tail.nextNode = newNode;

// Update the tail pointer to point to the new node

tail = newNode;

// Make the new node as the head and update its head pointer

head.data = newNode.data;

newNode.nextNode = head;

// Update the head pointer to point to the new node

head.nextNode = newNode;

}

}

// A method to print all elements of the list in order

void printList() {

Node current = head; // Start from the head node

while (current != null) { // Loop until current is null

System.out.print(current.data + " "); // Print current's data followed by a space

current = current.nextNode; // Move to next node

}

System.out.println(); // Print a new line at end

current = tail; // Start from tail again

while (current != null) { // Loop until current is null

System.out.print(current.data + " "); // Print current's data followed by a space

current = current.nextNode; // Move to next node

}

System.out.println(); // Print a new line at end

}

}

// A class to test our program with some examples

class TestCircularLinkedList {

public static void main(String[] args) {

CircularLinkedList cll1 = new CircularLinkedList();

cll1.insertAtEnd(10);

cll1.insertAtEnd(20);

cll1.insertAtEnd(30);

cll1.printList();

}

}

7. Write a program in Java to traverse a doubly linked list in the forward and backward directions

public class DoublyLinkedList {

Node head; // head of list

/\* Doubly Linked list Node\*/

class Node {

int data;

Node prev;

Node next;

// Constructor to create a new node

// next and prev is by default initialized as null

Node(int d) { data = d; }

}

// Function to traverse the doubly linked list in forward direction

public void printListForward() {

Node node = head;

while (node != null) {

System.out.print(node.data + " ");

node = node.next;

}

}

// Function to traverse the doubly linked list in backward direction

public void printListBackward() {

Node node = head;

while (node.next != null) {

node = node.next;

}

while (node != null) {

System.out.print(node.data + " ");

node = node.prev;

}

}

// Driver Code

public static void main(String[] args) {

DoublyLinkedList dll = new DoublyLinkedList();

dll.head = dll.new Node(1);

Node second = dll.new Node(2);

Node third = dll.new Node(3);

dll.head.next = second;

second.next = third;

second.prev = dll.head;

third.prev = second;

System.out.println("Doubly linked list in forward direction:");

dll.printListForward();

System.out.println("\nDoubly linked list in backward direction:");

dll.printListBackward();

}

}

8.Write a program in Java to insert and remove elements in a stack

// Import the Stack class

import java.util.Stack;

// Create a class named StackDemo

public class StackDemo {

// Create a main method

public static void main(String[] args) {

// Create an empty stack of integers

Stack<Integer> stack = new Stack<>();

// Insert some elements into the stack using push() method

stack.push(10);

stack.push(20);

stack.push(30);

// Print the original stack

System.out.println("Original stack: " + stack);

// Remove an element from the top of the stack using pop() method

int removed = stack.pop();

// Print the removed element and the updated stack

System.out.println("Removed element: " + removed);

System.out.println("Updated stack: " + stack);

}

}

9. Write a program in Java to insert and remove elements in a queue

// Using Queue interface

import java.util.Queue;

import java.util.LinkedList;

public class QueueExample {

public static void main(String[] args) {

// Creating a Queue object using LinkedList class

Queue<Integer> numbers = new LinkedList<>();

// Inserting some elements at the rear of the queue

numbers.offer(1);

numbers.offer(2);

numbers.offer(3);

// Printing the queue before deletion

System.out.println("Queue before deletion: " + numbers);

// Removing and returning an element from the front of the queue

int removedNumber = numbers.poll();

// Printing the removed element

System.out.println("Removed element: " + removedNumber);

// Printing the queue after deletion

System.out.println("Queue after deletion: " + numbers);

}

}