

|  |
| --- |
| | Lab No | Date | Topic |
|  |
| Lab 1 | 31-01-25 | Arrays |
| Lab 2 | 07-02-25 | Linked List |
| Lab 3 | 14-02-25 | Doubly Linked List |
| Lab 4 | 21-02-25 | Circular Linked List |
| Lab 5 | 28-02-25 | Reverse Codes |
| Lab 6 | 07-03-25 | Stack Operations |
| Lab 7 | 12-03-25 | Queues |
| Lab 8 | 28-03-25 | Priority Queues |
| Lab 9 | 04-04-25 | Trees |
| Lab10 | 11-04-25 | Binary Search Tree |
| Lab11 | 25-04-25 | AVL |

LAB-1(31-01-25)

1.Malloc

#include <stdio.h>

#include <stdlib.h>

int main() {

int n, i, \*ptr, sum = 0;

printf("Enter number of elements: ");

scanf("%d", &n);

ptr = (int\*)malloc(n \* sizeof(int));

if (ptr == NULL) {

printf("Error! Memory not allocated.");

exit(0);

}

printf("Enter elements of array: ");

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

scanf("%d", ptr + i);

sum += \*(ptr + i);

}

printf("Sum = %d", sum);

free(ptr);

return 0;

}

A screenshot of a computer program

AI-generated content may be incorrect.

2.Calloc

#include <stdio.h>

#include <stdlib.h>

int main() {

int n, i, \*ptr, sum = 0;

printf("Enter number of elements: ");

scanf("%d", &n);

ptr = (int\*)calloc(n, sizeof(int));

if (ptr == NULL) {

printf("Error! Memory not allocated.");

exit(0);

}

printf("Enter elements of array: ");

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

scanf("%d", ptr + i);

sum += \*(ptr + i);

}

printf("Sum = %d", sum);

free(ptr);

return 0;

}

A screenshot of a computer program

AI-generated content may be incorrect.

3.Realloc

#include <stdio.h>

#include <stdlib.h>

int main() {

int \*ptr, i, n1, n2;

printf("Enter size of array: ");

scanf("%d", &n1);

ptr = (int\*)malloc(n1 \* sizeof(int));

printf("Address of previously allocated memory:\n");

for (i = 0; i < n1; ++i) {

printf("%p\t", (ptr + i));

}

printf("\nEnter new size of array: ");

scanf("%d", &n2);

ptr = realloc(ptr, n2 \* sizeof(int));

printf("Address of newly allocated memory:\n");

for (i = 0; i < n2; ++i) {

printf("%p\t", (ptr + i));

}

free(ptr);

return 0;

}

A computer screen with white text

AI-generated content may be incorrect.

4. Calloc to print 1 to 5 nuimbers

#include <stdio.h>

#include <stdlib.h>

int main() {

int \*ptr, i;

ptr = (int\*)calloc(5, sizeof(int));

if (ptr == NULL) {

printf("Error! Memory not allocated.");

exit(0);

}

for (i = 0; i < 5; i++) {

ptr[i] = i + 1;

}

printf("Numbers from 1 to 5 (using calloc): ");

for (i = 0; i < 5; i++) {

printf("%d ", ptr[i]);

}

free(ptr);

return 0;

}

A screenshot of a computer

AI-generated content may be incorrect.

5.Malloc to print 1to 5 numbers

#include <stdio.h>

#include <stdlib.h>

int main() {

int \*ptr, i;

ptr = (int\*)malloc(5 \* sizeof(int));

if (ptr == NULL) {

printf("Error! Memory not allocated.");

exit(0);

}

for (i = 0; i < 5; i++) {

ptr[i] = i + 1;

}

printf("Numbers from 1 to 5 (using malloc): ");

for (i = 0; i < 5; i++) {

printf("%d ", ptr[i]);

}

free(ptr);

return 0;

}A screenshot of a computer

AI-generated content may be incorrect.

6. Real World scenarios involving array

#include <stdio.h>

#define DAYS\_IN\_WEEK 7

int main() {

// Declare an array to store temperatures for each day of the week

float temperatures[DAYS\_IN\_WEEK];

// Input temperatures for each day

printf("Enter temperatures for each day of the week:\n");

for (int i = 0; i < DAYS\_IN\_WEEK; ++i) {

printf("Day %d: ", i + 1);

scanf("%f", &temperatures[i]);

}

// Display the recorded temperatures

printf("\nRecorded temperatures for the week:\n");

for (int i = 0; i < DAYS\_IN\_WEEK; ++i) {

printf("Day %d: %.2f\n", i + 1, temperatures[i]);

}

// Calculate and display the average temperature

float totalTemperature = 0;

for (int i = 0; i < DAYS\_IN\_WEEK; ++i) {

totalTemperature += temperatures[i];

}

float averageTemperature = totalTemperature / DAYS\_IN\_WEEK;

printf("\nAverage temperature for the week: %.2f\n", averageTemperature);

return 0;

}A screenshot of a computer

AI-generated content may be incorrect.

7. Reverse an array

#include <stdio.h>

int main() {

int n = 5; // Size of the array

int rev[5]; // Array to store the reversed elements

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

// Reverse the array

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

rev[i] = arr[n - i - 1];

}

// Print the reversed array

printf("Reversed Array:\n");

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

printf("arr[%d] = %d\n", i, rev[i]);

}

return 0;

}

A screenshot of a computer program

AI-generated content may be incorrect.

8. Another logic for Reversing array

#include <stdio.h>

int main() {

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

int n = 5;

printf("Reversed Array:\n");

for (int i = n - 1; i >= 0; i--) {

printf("%d ", arr[i]);

}

printf("\n");

return 0;

}A screenshot of a computer

AI-generated content may be incorrect.

9. Merge 2 arrays

include <stdio.h>

int main() {

int arr1[10], arr2[10], arr3[20];

int i, n1, n2, m, index = 0;

// Input size and elements of the first array

printf("Enter the number of elements in array1: ");

scanf("%d", &n1);

printf("Enter the elements of the first array:\n");

for (i = 0; i < n1; i++) {

printf("arr1[%d] = ", i);

scanf("%d", &arr1[i]);

}

// Input size and elements of the second array

printf("Enter the number of elements in array2: ");

scanf("%d", &n2);

printf("Enter the elements of the second array:\n");

for (i = 0; i < n2; i++) {

printf("arr2[%d] = ", i);

scanf("%d", &arr2[i]);

}

// Merge the first array into the third array

for (i = 0; i < n1; i++) {

arr3[index++] = arr1[i];

}

// Merge the second array into the third array

for (i = 0; i < n2; i++) {

arr3[index++] = arr2[i];

}

// Print the merged array

m = n1 + n2;

printf("The merged array is:\n");

for (i = 0; i < m; i++) {

printf("arr[%d] = %d\n", i, arr3[i]);

}

return 0;

}

A screenshot of a computer program

AI-generated content may be incorrect.

10. Another logic

#include <stdio.h>

int main() {

int arr1[10], arr2[10], arr3[20];

int n1, n2, i;

// Input size and elements of the first array

printf("Enter number of elements in array1: ");

scanf("%d", &n1);

printf("Enter elements of array1:\n");

for (i = 0; i < n1; i++) {

scanf("%d", &arr1[i]);

}

// Input size and elements of the second array

printf("Enter number of elements in array2: ");

scanf("%d", &n2);

printf("Enter elements of array2:\n");

for (i = 0; i < n2; i++) {

scanf("%d", &arr2[i]);

}

// Merge the arrays

for (i = 0; i < n1; i++) {

arr3[i] = arr1[i];

}

for (i = 0; i < n2; i++) {

arr3[n1 + i] = arr2[i];

}

// Print the merged array

printf("Merged Array:\n");

for (i = 0; i < n1 + n2; i++) {

printf("%d ", arr3[i]);

}

printf("\n");

return 0;

}

A screenshot of a computer program

AI-generated content may be incorrect.

11. Array 3d

#include <stdio.h>

int main() {

// Define a 3D array with dimensions 2x3x4

// It contains 2 layers, each with 3 rows and 4 columns

int arr[2][3][4] = {

{

{1, 2, 3, 4}, // First row of first layer

{5, 6, 7, 8}, // Second row of first layer

{9, 10, 11, 12} // Third row of first layer

},

{

{13, 14, 15, 16}, // First row of second layer

{17, 18, 19, 20}, // Second row of second layer

{21, 22, 23, 24} // Third row of second layer

}

};

// Access a specific element in the 3D array

printf("Element at [1][2][3]: %d\n", arr[1][2][3]); // This prints 24

// Print all elements in the 3D array

printf("3D Array Elements:\n");

for (int i = 0; i < 2; i++) { // Loop through each layer

for (int j = 0; j < 3; j++) { // Loop through each row

for (int k = 0; k < 4; k++) { // Loop through each column

printf("arr[%d][%d][%d] = %d\n", i, j, k, arr[i][j][k]);

}

}

}

return 0;

}

A screenshot of a computer

AI-generated content may be incorrect.

12. Weather data management

#include <stdio.h>

int main() {

// 3D array to store weather data

// Dimensions: 2 cities, 7 days, 3 parameters (temperature, humidity, wind speed)

int weather[2][7][3];

// Input weather data for each city and day

for (int city = 0; city < 2; city++) {

for (int day = 0; day < 7; day++) {

printf("Enter data for City %d, Day %d (Temp, Humidity, Wind Speed):\n", city + 1, day + 1);

for (int parameter = 0; parameter < 3; parameter++) {

scanf("%d", &weather[city][day][parameter]);

}

}

}

// Display the collected weather data

printf("\nWeather Data Summary:\n");

for (int city = 0; city < 2; city++) {

printf("City %d:\n", city + 1);

for (int day = 0; day < 7; day++) {

printf(" Day %d - Temp: %d, Humidity: %d, Wind Speed: %d\n",

day + 1,

weather[city][day][0],

weather[city][day][1],

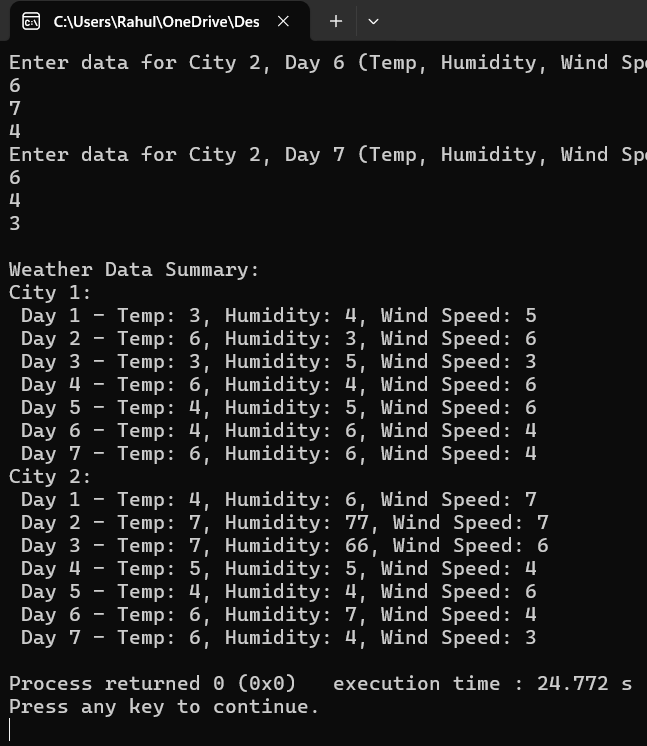
weather[city][day][2]);

}

}

return 0;

}



13. Insertion in an array

INSERT AT START

#include <stdio.h>

// Function to insert an element at the start of the array

int insertAtStart(int arr[], int n, int value) {

// Shift all elements to the right to make space at the start

for (int i = n; i > 0; i--) {

arr[i] = arr[i - 1];

}

// Place the new value at the start

arr[0] = value;

// Return the updated size of the array

return n + 1;

}

int main() {

int arr[10] = {20, 30, 40, 50}; // Array with initial elements

int n = 4; // Current size of the array

int value = 10; // Value to insert at the start

// Insert at the start

n = insertAtStart(arr, n, value);

// Display the updated array

printf("Array after insertion at the start: ");

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

printf("%d ", arr[i]);

}

printf("\n");

return 0;

}A screenshot of a computer program

AI-generated content may be incorrect.

14. insert at specific position

#include <stdio.h>

// Function to insert an element at a specific position

int insertAtPosition(int arr[], int n, int pos, int value) {

// Shift elements to the right starting from the end up to position

for (int i = n; i > pos; i--) {

arr[i] = arr[i - 1];

}

// Place the new value at the specified position

arr[pos] = value;

// Return the updated size of the array

return n + 1;

}

int main() {

int arr[10] = {10, 20, 30, 50}; // Array with initial elements

int n = 4; // Current size of the array

int pos = 2; // Position to insert (0-based index)

int value = 40; // Value to insert at the position

// Insert at specified position

n = insertAtPosition(arr, n, pos, value);

// Display the updated array

printf("Array after insertion at position %d: ", pos);

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

printf("%d ", arr[i]);

}

printf("\n");

return 0;

}

A screenshot of a computer

AI-generated content may be incorrect.

15. Insert at END

#include <stdio.h>

// Function to insert an element at the end of the array

int insertAtEnd(int arr[], int n, int value) {

// Place the new value at the end

arr[n] = value;

// Return the updated size of the array

return n + 1;

}

int main() {

int arr[10] = {10, 20, 30, 40}; // Array with initial elements

int n = 4; // Current size of the array

int value = 50; // Value to insert at the end

// Insert at the end

n = insertAtEnd(arr, n, value);

// Display the updated array

printf("Array after insertion at the end: ");

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

printf("%d ", arr[i]);

}

printf("\n");

return 0;

}

A screenshot of a computer program

AI-generated content may be incorrect.

16. DELETION FROM AN ARRAY

DELETE FROM START

#include <stdio.h>

// Function to delete the first element from the array

int deleteFromStart(int arr[], int n) {

// Shift elements to the left to overwrite the first element

for (int i = 0; i < n - 1; i++) {

arr[i] = arr[i + 1];

}

// Return the updated size of the array

return n - 1;

}

int main() {

int arr[10] = {10, 20, 30, 40, 50}; // Initial elements

int n = 5; // Size of array

// Delete the first element

n = deleteFromStart(arr, n);

// Print updated array

printf("Array after deletion from the start: ");

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

printf("%d ", arr[i]);

}

printf("\n");

return 0;

}

A screenshot of a computer

AI-generated content may be incorrect.

17. Delete from specific position

#include <stdio.h>

// Function to delete an element from a specific position

int deleteFromPosition(int arr[], int n, int pos) {

// Check for invalid position

if (pos < 0 || pos >= n) {

printf("Invalid position!\n");

return n;

}

// Shift elements to the left from the position

for (int i = pos; i < n - 1; i++) {

arr[i] = arr[i + 1];

}

// Return the updated size

return n - 1;

}

int main() {

int arr[10] = {10, 20, 30, 40, 50}; // Initial array

int n = 5; // Size

int pos = 2; // Index to delete (0-based)

// Delete from specified position

n = deleteFromPosition(arr, n, pos);

// Print updated array

printf("Array after deletion from position %d: ", pos);

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

printf("%d ", arr[i]);

}

printf("\n");

return 0;

}

A screenshot of a computer

AI-generated content may be incorrect.

18.Delete from the END

#include <stdio.h>

// Function to delete the last element of the array

int deleteFromEnd(int arr[], int n) {

if (n <= 0) {

printf("Array is already empty!\n");

return 0;

}

// Just decrease size

return n - 1;

}

int main() {

int arr[10] = {10, 20, 30, 40, 50}; // Initial elements

int n = 5; // Size

// Delete the last element

n = deleteFromEnd(arr, n);

// Print updated array

printf("Array after deletion from the end: ");

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

printf("%d ", arr[i]);

}

printf("\n");

return 0;

}

A screenshot of a computer

AI-generated content may be incorrect.

Lab-2 (07-2-25)

19. BUBBLE SORT  
 SELECTION SORT

Linked list creation

Display

class Node {

int data;

Node next;

Node(int value) {

this.data = value;

this.next = null;

}

}

class SinglyLinkedList {

private Node head;

private Node tail;

// Add a node at the end

public void add(int value) {

Node newNode = new Node(value);

if (head == null) {

head = newNode;

tail = newNode;

} else {

tail.next = newNode;

tail = newNode;

}

}

// Display the list

public void display() {

if (head == null) {

System.out.println("Linked list is empty.");

return;

}

Node temp = head;

System.out.print("Linked list: ");

while (temp != null) {

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

temp = temp.next;

}

System.out.println();

}

}

// Main class containing the main method

public class Main {

public static void main(String[] args) {

SinglyLinkedList list = new SinglyLinkedList();

list.add(10);

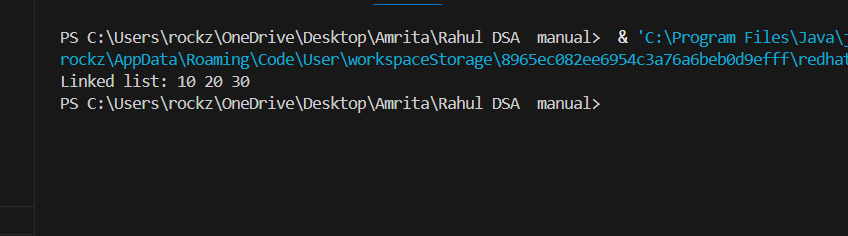
list.add(20);

list.add(30);

list.display();

}

}



20. Insert at start

class Node {

int data;

Node next;

Node(int value) {

this.data = value;

this.next = null;

}

}

class SinglyLinkedList {

private Node head;

private Node tail;

// Add a node at the end

public void add(int value) {

Node newNode = new Node(value);

if (head == null) {

head = newNode;

tail = newNode;

} else {

tail.next = newNode;

tail = newNode;

}

}

// Insert at beginning

public void insertatbegining(int value) {

Node newNode = new Node(value);

if (head == null) {

head = newNode;

tail = newNode;

} else {

newNode.next = head;

head = newNode;

}

}

// Display the list

public void display() {

if (head == null) {

System.out.println("Linked list is empty.");

return;

}

Node temp = head;

System.out.print("Linked list: ");

while (temp != null) {

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

temp = temp.next;

}

System.out.println();

}

}

// Main class

public class Main {

public static void main(String[] args) {

SinglyLinkedList list = new SinglyLinkedList();

list.add(10);

list.add(20);

list.add(30);

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

list.display();

list.insertatbegining(5);

System.out.println("After inserting at beginning:");

list.display();

}

}

A screen shot of a computer program

AI-generated content may be incorrect.

21. At End

class Node {

int data;

Node next;

Node(int value) {

this.data = value;

this.next = null;

}

}

class SinglyLinkedList {

private Node head;

private Node tail;

// Add a node at the end

public void add(int value) {

Node newNode = new Node(value);

if (head == null) {

head = newNode;

tail = newNode;

} else {

tail.next = newNode;

tail = newNode;

}

}

public void insertatend(int value)

{

Node newNode=new Node(value);

if(head==null)

{

head=newNode;

tail=newNode;

}else{

tail.next=newNode;

tail=newNode;

}

}

// Display the list

public void display() {

if (head == null) {

System.out.println("Linked list is empty.");

return;

}

Node temp = head;

System.out.print("Linked list: ");

while (temp != null) {

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

temp = temp.next;

}

System.out.println();

}

}

// Main class

public class Main {

public static void main(String[] args) {

SinglyLinkedList list = new SinglyLinkedList();

list.add(10);

list.add(20);

list.add(30);

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

list.display();

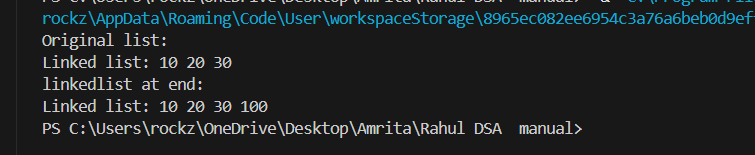
list.insertatend(100);

System.out.println("linkedlist at end:");

list.display();

}

}



22. At Specific Position

class Node {

int data;

Node next;

Node(int value) {

this.data = value;

this.next = null;

}

}

class SinglyLinkedList {

private Node head;

private Node tail;

// Add a node at the end

public void add(int value) {

Node newNode = new Node(value);

if (head == null) {

head = newNode;

tail = newNode;

} else {

tail.next = newNode;

tail = newNode;

}

}

// Insert at the beginning

public void insertatbegining(int value) {

Node newNode = new Node(value);

if (head == null) {

head = newNode;

tail = newNode;

} else {

newNode.next = head;

head = newNode;

}

}

// Insert at specific position (1-based index)

public void insertatposition(int value, int pos) {

if (head == null || pos <= 1) {

insertatbegining(value);

return;

}

Node newNode = new Node(value);

Node temp = head;

int count = 1;

// Traverse to (pos - 1)th node or end of list

while (temp != null && count < pos - 1) {

temp = temp.next;

count++;

}

if (temp == null || temp.next == null) {

// If position is beyond current size, add at end

add(value);

} else {

newNode.next = temp.next;

temp.next = newNode;

}

}

// Display the list

public void display() {

if (head == null) {

System.out.println("Linked list is empty.");

return;

}

Node temp = head;

System.out.print("Linked list: ");

while (temp != null) {

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

temp = temp.next;

}

System.out.println();

}

}

// Main class

public class Main {

public static void main(String[] args) {

SinglyLinkedList list = new SinglyLinkedList();

list.add(10);

list.add(20);

list.add(30);

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

list.display();

list.insertatbegining(5);

System.out.println("After inserting at beginning:");

list.display();

list.insertatposition(67, 3);

System.out.println("After inserting 67 at position 3:");

list.display();

}

}

A screen shot of a computer

AI-generated content may be incorrect.

23. Delete at start

class Node {

int data;

Node next;

Node(int value) {

this.data = value;

this.next = null;

}

}

class SinglyLinkedList {

private Node head;

private Node tail;

// Add a node at the end

public void add(int value) {

Node newNode = new Node(value);

if (head == null) {

head = newNode;

tail = newNode;

} else {

tail.next = newNode;

tail = newNode;

}

}

// Insert at the beginning

public void insertatbegining(int value) {

Node newNode = new Node(value);

if (head == null) {

head = newNode;

tail = newNode;

} else {

newNode.next = head;

head = newNode;

}

}

// Insert at specific position (1-based index)

public void deleteatbegining()

{

if(head==null)

{

System.out.print("empty");

return;

}

if(head==tail)

{

head=null;

tail=null;

return;

}

Node temp=head;

head=head.next;

temp.next=null;

}

// Display the list

public void display() {

if (head == null) {

System.out.println("Linked list is empty.");

return;

}

Node temp = head;

System.out.print("Linked list: ");

while (temp != null) {

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

temp = temp.next;

}

System.out.println();

}

}

// Main class

public class Main {

public static void main(String[] args) {

SinglyLinkedList list = new SinglyLinkedList();

list.add(10);

list.add(20);

list.add(30);

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

list.display();

list.deleteatbegining();

System.out.println("linkedlist at deleteatsatart:");

list.display();

}

}

A screen shot of a computer

AI-generated content may be incorrect.

24. Delete END

class Node {

int data;

Node next;

Node(int value) {

this.data = value;

this.next = null;

}

}

class SinglyLinkedList {

private Node head;

private Node tail;

// Add a node at the end

public void add(int value) {

Node newNode = new Node(value);

if (head == null) {

head = newNode;

tail = newNode;

} else {

tail.next = newNode;

tail = newNode;

}

}

// Insert at the beginning

public void insertatbegining(int value) {

Node newNode = new Node(value);

if (head == null) {

head = newNode;

tail = newNode;

} else {

newNode.next = head;

head = newNode;

}

}

public void deleteatend()

{

if(head==null)

{

System.out.print("empty");

return;

}

if(head==tail)

{

head=null;

tail=null;

return;

}

Node temp=head;

while(temp.next!=tail)

{

temp=temp.next;

}

tail=temp;

temp.next=null;

}

// Display the list

public void display() {

if (head == null) {

System.out.println("Linked list is empty.");

return;

}

Node temp = head;

System.out.print("Linked list: ");

while (temp != null) {

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

temp = temp.next;

}

System.out.println();

}

}

// Main class

public class Main {

public static void main(String[] args) {

SinglyLinkedList list = new SinglyLinkedList();

list.add(10);

list.add(20);

list.add(30);

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

list.display();

list.deleteatend();

System.out.println("linkedlist at delete endposition:");

list.display();

}

}

A screen shot of a computer

AI-generated content may be incorrect.

25. Delete specific POS

class Node {

int data;

Node next;

Node(int value) {

this.data = value;

this.next = null;

}

}

class SinglyLinkedList {

private Node head;

private Node tail;

// Add a node at the end

public void add(int value) {

Node newNode = new Node(value);

if (head == null) {

head = newNode;

tail = newNode;

} else {

tail.next = newNode;

tail = newNode;

}

}

// Insert at the beginning

public void insertatbegining(int value) {

Node newNode = new Node(value);

if (head == null) {

head = newNode;

tail = newNode;

} else {

newNode.next = head;

head = newNode;

}

}

// Delete from beginning

public void deleteatbegining() {

if (head == null) {

System.out.println("List is empty. Nothing to delete.");

return;

}

if (head == tail) {

head = null;

tail = null;

} else {

Node temp = head;

head = head.next;

temp.next = null;

}

}

// Delete at specific position (1-based index)

public void deleteatposition(int pos) {

if (head == null) {

System.out.println("List is empty. Nothing to delete.");

return;

}

if (pos <= 1) {

deleteatbegining();

return;

}

Node temp = head;

int count = 1;

// Traverse to (pos - 1)th node

while (temp != null && temp.next != null && count < pos - 1) {

temp = temp.next;

count++;

}

// Check if position is valid

if (temp.next == null) {

System.out.println("Invalid position. Node does not exist.");

return;

}

// If deleting the last node, update tail

if (temp.next == tail) {

tail = temp;

}

temp.next = temp.next.next;

}

// Display the list

public void display() {

if (head == null) {

System.out.println("Linked list is empty.");

return;

}

Node temp = head;

System.out.print("Linked list: ");

while (temp != null) {

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

temp = temp.next;

}

System.out.println();

}

}

// Main class

public class Main {

public static void main(String[] args) {

SinglyLinkedList list = new SinglyLinkedList();

list.add(10);

list.add(20);

list.add(30);

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

list.display();

list.deleteatbegining();

System.out.println("After deleting from start:");

list.display();

list.deleteatposition(3);

System.out.println("After deleting at position 3:");

list.display();

}

}

A screen shot of a computer

AI-generated content may be incorrect.

26. Contact List Management

// Node class for contacts

class Contact {

String name;

Contact next;

public Contact(String name) {

this.name = name;

this.next = null;

}

}

// Linked List class to manage contacts

class ContactList {

Contact head;

// Add contact at the beginning

void addContact(String name) {

Contact newContact = new Contact(name);

newContact.next = head;

head = newContact;

}

// Display contacts

void displayContacts() {

Contact temp = head;

System.out.println("Contact List:");

while (temp != null) {

System.out.print(temp.name + " -> ");

temp = temp.next;

}

System.out.println("NULL");

}

}

// Main class

public class ContactListApp {

public static void main(String[] args) {

ContactList contacts = new ContactList();

contacts.addContact("Alice");

contacts.addContact("Bob");

contacts.addContact("Charlie");

contacts.displayContacts();

}

}

A screen shot of a computer

AI-generated content may be incorrect.

27. Book borrowing System in library

// Node class for books

class Book {

String title;

Book next;

public Book(String title) {

this.title = title;

this.next = null;

}

}

// Linked List for borrowed books

class BorrowedBooks {

Book head;

void borrowBook(String title) {

Book newBook = new Book(title);

newBook.next = head;

head = newBook;

}

void displayBorrowedBooks() {

Book temp = head;

System.out.println("Borrowed Books:");

while (temp != null) {

System.out.print(temp.title + " -> ");

temp = temp.next;

}

System.out.println("NULL");

}

}

// Main class

public class LibrarySystem {

public static void main(String[] args) {

BorrowedBooks library = new BorrowedBooks();

library.borrowBook("The Alchemist");

library.borrowBook("Data Structures and Algorithms");

library.displayBorrowedBooks();

}

}

A computer screen with white text

AI-generated content may be incorrect.

28. Music playlist System

// Node class for songs

class Song {

String title;

Song next;

public Song(String title) {

this.title = title;

this.next = null;

}

}

// Linked List for playlist

class Playlist {

Song head;

void addSong(String title) {

Song newSong = new Song(title);

newSong.next = head;

head = newSong;

}

void displayPlaylist() {

Song temp = head;

System.out.println("Music Playlist:");

while (temp != null) {

System.out.print(temp.title + " -> ");

temp = temp.next;

}

System.out.println("NULL");

}

}

// Main class

public class MusicPlayer {

public static void main(String[] args) {

Playlist playlist = new Playlist();

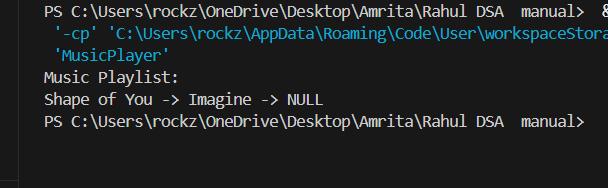
playlist.addSong("Imagine");

playlist.addSong("Shape of You");

playlist.displayPlaylist();

}

}



29. Patient Record System in Hospital

// Node class for patients

class Patient {

String name;

Patient next;

public Patient(String name) {

this.name = name;

this.next = null;

}

}

// Linked list class to manage patient records

class Hospital {

Patient head;

void admitPatient(String name) {

Patient newPatient = new Patient(name);

newPatient.next = head;

head = newPatient;

}

void displayPatients() {

Patient temp = head;

System.out.println("Admitted Patients:");

while (temp != null) {

System.out.print(temp.name + " -> ");

temp = temp.next;

}

System.out.println("NULL");

}

}

// Main class

public class HospitalManagement {

public static void main(String[] args) {

Hospital hospital = new Hospital();

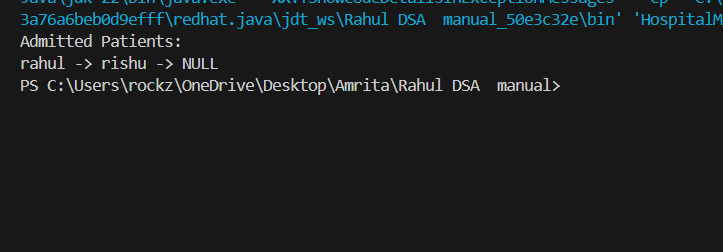
hospital.admitPatient("rishu");

hospital.admitPatient("rishi");

hospital.displayPatients();

}

}



30. Browser History Navigation

// Node class for web pages

class WebPage {

String url;

WebPage next;

public WebPage(String url) {

this.url = url;

this.next = null;

}

}

// Linked list class for managing browser history

class BrowserHistory {

WebPage head;

void visitPage(String url) {

WebPage newPage = new WebPage(url);

newPage.next = head;

head = newPage;

}

void displayHistory() {

WebPage temp = head;

System.out.println("Browser History:");

while (temp != null) {

System.out.print(temp.url + " -> ");

temp = temp.next;

}

System.out.println("NULL");

}

}

// Main class

public class BrowserNavigation {

public static void main(String[] args) {

BrowserHistory history = new BrowserHistory();

history.visitPage("google.com");

history.visitPage("github.com");

history.visitPage("stackoverflow.com");

history.displayHistory();

}

}

A computer screen shot of a computer

AI-generated content may be incorrect.

31. Train Ticket booking system

// Node class for train tickets

class Ticket {

String passengerName;

Ticket next;

public Ticket(String passengerName) {

this.passengerName = passengerName;

this.next = null;

}

}

// Linked list class to manage train tickets

class TrainTickets {

Ticket head;

void bookTicket(String passengerName) {

Ticket newTicket = new Ticket(passengerName);

newTicket.next = head;

head = newTicket;

}

void displayTickets() {

Ticket temp = head;

System.out.println("Booked Tickets:");

while (temp != null) {

System.out.print(temp.passengerName + " -> ");

temp = temp.next;

}

System.out.println("NULL");

}

}

// Main class

public class TrainBookingSystem {

public static void main(String[] args) {

TrainTickets train = new TrainTickets();

train.bookTicket("vishnu");

train.bookTicket("vamsi");

train.bookTicket("virat");

train.displayTickets();

}

}

A computer screen shot of a computer code

AI-generated content may be incorrect.

32. Task Management System

// Node class for tasks

class Task {

String taskName;

Task next;

public Task(String taskName) {

this.taskName = taskName;

this.next = null;

}

}

// Linked list class for managing tasks

class TaskManager {

Task head;

void addTask(String taskName) {

Task newTask = new Task(taskName);

newTask.next = head;

head = newTask;

}

void displayTasks() {

Task temp = head;

System.out.println("Task List:");

while (temp != null) {

System.out.print(temp.taskName + " -> ");

temp = temp.next;

}

System.out.println("NULL");

}

}

// Main class

public class TaskManagerApp {

public static void main(String[] args) {

TaskManager tasks = new TaskManager();

tasks.addTask("Complete Java assignment");

tasks.addTask("Attend team meeting");

tasks.addTask("Review project code");

tasks.displayTasks();

}

}

A screen shot of a computer

AI-generated content may be incorrect.

LAB-3 (14-2-25)

33. Creation Doubly Linked list

// Node class for Doubly Linked List

class DNode {

int data;

DNode next;

DNode prev;

DNode(int value) {

this.data = value;

this.next = null;

this.prev = null;

}

}

// Doubly Linked List class

public class DoublyLinkedList {

private DNode head;

private DNode tail;

// Add node to the end

public void add(int value) {

DNode newDNode = new DNode(value);

if (head == null) {

head = newDNode;

tail = newDNode;

} else {

tail.next = newDNode;

newDNode.prev = tail;

tail = newDNode;

}

}

// Display the list

public void display() {

if (head == null) {

System.out.println("Linked list is empty.");

return;

}

DNode temp = head;

System.out.print("Linked list: ");

while (temp != null) {

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

temp = temp.next;

}

System.out.println();

}

// Main method inside the class

public static void main(String[] args) {

DoublyLinkedList dlist = new DoublyLinkedList();

dlist.add(10);

dlist.add(20);

dlist.add(30);

dlist.display();

}

}

A screen shot of a computer

AI-generated content may be incorrect.

34. DLL insertion (start)

// Node class for Doubly Linked List

class DNode {

int data;

DNode next;

DNode prev;

DNode(int value) {

this.data = value;

this.next = null;

this.prev = null;

}

}

// Doubly Linked List class

public class DoublyLinkedList {

private DNode head;

private DNode tail;

// Add node to the end

public void add(int value) {

DNode newDNode = new DNode(value);

if (head == null) {

head = newDNode;

tail = newDNode;

} else {

tail.next = newDNode;

newDNode.prev = tail;

tail = newDNode;

}

}

// Display the list

public void display() {

if (head == null) {

System.out.println("Linked list is empty.");

return;

}

DNode temp = head;

System.out.print("Linked list: ");

while (temp != null) {

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

temp = temp.next;

}

System.out.println();

}

// Insert at the beginning

public void insertatbegining(int value) {

DNode newDNode = new DNode(value);

if (head == null) {

head = newDNode;

tail = newDNode;

} else {

newDNode.next = head;

head.prev = newDNode;

head = newDNode;

}

}

// Main method

public static void main(String[] args) {

DoublyLinkedList dlist = new DoublyLinkedList();

dlist.add(10);

dlist.add(20);

dlist.add(30);

dlist.display();

dlist.insertatbegining(5);

System.out.print("After inserting at start: ");

dlist.display();

}

}

A screen shot of a computer

AI-generated content may be incorrect.

35. DLL END

// Node class for Doubly Linked List

class DNode {

int data;

DNode next;

DNode prev;

DNode(int value) {

this.data = value;

this.next = null;

this.prev = null;

}

}

// Doubly Linked List class

public class DoublyLinkedList {

private DNode head;

private DNode tail;

// Add node to the end

public void add(int value) {

DNode newDNode = new DNode(value);

if (head == null) {

head = newDNode;

tail = newDNode;

} else {

tail.next = newDNode;

newDNode.prev = tail;

tail = newDNode;

}

}

// Insert at end (alternative to add)

public void insertatend(int value) {

DNode newDNode = new DNode(value);

if (head == null) {

head = newDNode;

tail = newDNode;

} else {

tail.next = newDNode;

newDNode.prev = tail;

tail = newDNode;

}

}

// Display the list

public void display() {

if (head == null) {

System.out.println("Linked list is empty.");

return;

}

DNode temp = head;

System.out.print("Linked list: ");

while (temp != null) {

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

temp = temp.next;

}

System.out.println();

}

// Main method

public static void main(String[] args) {

DoublyLinkedList dlist = new DoublyLinkedList();

dlist.add(10);

dlist.add(20);

dlist.add(30);

dlist.display();

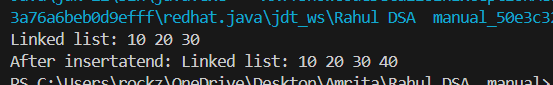
dlist.insertatend(40);

System.out.print("After insertatend: ");

dlist.display();

}

}



36. DLL specific position

// Node class for Doubly Linked List

class DNode {

int data;

DNode next;

DNode prev;

DNode(int value) {

this.data = value;

this.next = null;

this.prev = null;

}

}

// Doubly Linked List class

public class DoublyLinkedList {

private DNode head;

private DNode tail;

// Add node to the end

public void add(int value) {

DNode newDNode = new DNode(value);

if (head == null) {

head = newDNode;

tail = newDNode;

} else {

tail.next = newDNode;

newDNode.prev = tail;

tail = newDNode;

}

}

// Display the list

public void display() {

if (head == null) {

System.out.println("Linked list is empty.");

return;

}

DNode temp = head;

System.out.print("Linked list: ");

while (temp != null) {

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

temp = temp.next;

}

System.out.println();

}

// Insert at the beginning

public void insertatbegining(int value) {

DNode newDNode = new DNode(value);

if (head == null) {

head = newDNode;

tail = newDNode;

} else {

newDNode.next = head;

head.prev = newDNode;

head = newDNode;

}

}

// Insert at the end

public void insertatend(int value) {

add(value); // Using existing add() method

}

// Insert at specific position (1-based index)

public void insertatposition(int value, int pos) {

if (head == null || pos <= 1) {

insertatbegining(value);

return;

}

DNode newDNode = new DNode(value);

DNode temp = head;

// Traverse to (pos - 1)th node

for (int i = 1; i < pos - 1 && temp.next != null; i++) {

temp = temp.next;

}

// If inserting at the end

if (temp.next == null) {

insertatend(value);

return;

}

// Insert in the middle

newDNode.next = temp.next;

newDNode.prev = temp;

temp.next.prev = newDNode;

temp.next = newDNode;

}

// Main method

public static void main(String[] args) {

DoublyLinkedList dlist = new DoublyLinkedList();

dlist.add(10);

dlist.add(20);

dlist.add(30);

dlist.display();

dlist.insertatbegining(5);

System.out.print("After inserting at start: ");

dlist.display();

dlist.insertatposition(15, 3); // Inserting at position 3

System.out.print("After inserting at position 3: ");

dlist.display();

}

}

A screen shot of a computer screen

AI-generated content may be incorrect.

37. DLL (deletion form start)

// Node class for Doubly Linked List

class DNode {

int data;

DNode next;

DNode prev;

DNode(int value) {

this.data = value;

this.next = null;

this.prev = null;

}

}

// Doubly Linked List class

public class DoublyLinkedList {

private DNode head;

private DNode tail;

// Add node to the end

public void add(int value) {

DNode newDNode = new DNode(value);

if (head == null) {

head = newDNode;

tail = newDNode;

} else {

tail.next = newDNode;

newDNode.prev = tail;

tail = newDNode;

}

}

public void deletefrombegining()

{

if(head==null)

{

System.out.println("empty");

return;

}

if(head==tail)

{

head=null;

tail=null;

}

head=head.next;

head.prev=null;

}

// Display the list

public void display() {

if (head == null) {

System.out.println("Linked list is empty.");

return;

}

DNode temp = head;

System.out.print("Linked list: ");

while (temp != null) {

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

temp = temp.next;

}

System.out.println();

}

// Main method

public static void main(String[] args) {

DoublyLinkedList dlist = new DoublyLinkedList();

dlist.add(10);

dlist.add(20);

dlist.add(30);

dlist.display();

dlist.deletefrombegining();

System.out.print("deleteatstart:");

dlist.display();

}

}

A screen shot of a computer code

AI-generated content may be incorrect.

38. DLL deletion at end

// Node class for Doubly Linked List

class DNode {

int data;

DNode next;

DNode prev;

DNode(int value) {

this.data = value;

this.next = null;

this.prev = null;

}

}

// Doubly Linked List class

public class DoublyLinkedList {

private DNode head;

private DNode tail;

// Add node to the end

public void add(int value) {

DNode newDNode = new DNode(value);

if (head == null) {

head = newDNode;

tail = newDNode;

} else {

tail.next = newDNode;

newDNode.prev = tail;

tail = newDNode;

}

}

// Delete from the end

public void deletefromend() {

if (head == null) {

System.out.println("List is already empty.");

return;

}

if (head == tail) {

head = null;

tail = null;

} else {

tail = tail.prev;

tail.next = null;

}

}

// Display the list

public void display() {

if (head == null) {

System.out.println("Linked list is empty.");

return;

}

DNode temp = head;

System.out.print("Linked list: ");

while (temp != null) {

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

temp = temp.next;

}

System.out.println();

}

// Main method

public static void main(String[] args) {

DoublyLinkedList dlist = new DoublyLinkedList();

dlist.add(10);

dlist.add(20);

dlist.add(30);

dlist.add(40);

dlist.display();

dlist.deletefromend(); // deletes 40

System.out.print("After deletefromend: ");

dlist.display();

}

}

A screen shot of a computer program

AI-generated content may be incorrect.

39. DLL deletion from specific pos

// Node class for Doubly Linked List

class DNode {

int data;

DNode next;

DNode prev;

DNode(int value) {

this.data = value;

this.next = null;

this.prev = null;

}

}

// Doubly Linked List class

public class DoublyLinkedList {

private DNode head;

private DNode tail;

// Add node to the end

public void add(int value) {

DNode newDNode = new DNode(value);

if (head == null) {

head = newDNode;

tail = newDNode;

} else {

tail.next = newDNode;

newDNode.prev = tail;

tail = newDNode;

}

}

// Delete from the beginning

public void deletefrombegining() {

if (head == null) {

System.out.println("List is already empty.");

return;

}

if (head == tail) {

head = null;

tail = null;

} else {

head = head.next;

head.prev = null;

}

}

// Delete from the end

public void deletefromend() {

if (head == null) {

System.out.println("List is already empty.");

return;

}

if (head == tail) {

head = null;

tail = null;

} else {

tail = tail.prev;

tail.next = null;

}

}

// Delete from a given position

public void deleteatposition(int pos) {

if (head == null) {

System.out.println("List is empty.");

return;

}

if (pos <= 1) {

deletefrombegining();

return;

}

DNode temp = head;

for (int i = 1; i < pos - 1; i++) {

if (temp.next == null) {

System.out.println("Position out of bounds.");

return;

}

temp = temp.next;

}

if (temp.next == null || temp.next == tail) {

deletefromend();

return;

}

DNode toDelete = temp.next;

temp.next = toDelete.next;

if (toDelete.next != null) {

toDelete.next.prev = temp;

}

}

// Display the list

public void display() {

if (head == null) {

System.out.println("Linked list is empty.");

return;

}

DNode temp = head;

System.out.print("Linked list: ");

while (temp != null) {

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

temp = temp.next;

}

System.out.println();

}

// Main method

public static void main(String[] args) {

DoublyLinkedList dlist = new DoublyLinkedList();

dlist.add(10);

dlist.add(20);

dlist.add(30);

dlist.add(40);

dlist.display();

dlist.deletefrombegining();

System.out.print("After deletefrombegining: ");

dlist.display();

dlist.deleteatposition(2); // deletes 30

System.out.print("After deleteatposition(2): ");

dlist.display();

dlist.deletefromend(); // deletes 40

System.out.print("After deletefromend: ");

dlist.display();

}

}

A screen shot of a computer program

AI-generated content may be incorrect.

LAB\_4 (21-02-25)

40. Circular Singly Linked list

Insertion at start

class CNode

{

int data;

CNode next;

CNode(int value)

{

this.data=value;

this.next=null;

}

}

public class CircularLinkedList

{

private CNode head;

private CNode tail;

public void add(int value)

{

CNode newCNode=new CNode(value);

if(head==null)

{

head=newCNode;

tail=newCNode;

}else{

tail.next=newCNode;

tail=newCNode;

tail.next=head;

}

}

public void display()

{

if(head==null)

{

System.out.print("empty");

return;

}

CNode temp=head;

do{

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

temp=temp.next;

}while(temp!=head);

{

System.out.println();

}

}

public void insertatbegining(int value)

{

CNode newCNode=new CNode(value);

if(head==null)

{

head=newCNode;

tail=newCNode;

newCNode.next=head;

}else{

newCNode.next=head;

head=newCNode;

tail.next=newCNode;

}

}

public static void main(String[] args)

{

CircularLinkedList clist = new CircularLinkedList();

clist.add(10);

clist.add(20);

clist.add(30);

clist.add(40);

clist.add(50);

clist.add(60);

clist.add(70);

System.out.print("After creation: ");

clist.display();

clist.insertatbegining(10);

System.out.print("After inserting 10 at beginning: ");

clist.display();

}

}

A computer screen shot of a computer program

AI-generated content may be incorrect.

41. Insertion at end

class CNode {

int data;

CNode next;

CNode(int value) {

this.data = value;

this.next = null;

}

}

public class CircularLinkedList {

private CNode head;

private CNode tail;

public void add(int value) {

CNode newCNode = new CNode(value);

if (head == null) {

head = newCNode;

tail = newCNode;

tail.next = head;

} else {

tail.next = newCNode;

tail = newCNode;

tail.next = head;

}

}

public void deletefromend() {

if (head == null) {

System.out.println("List is empty.");

return;

}

if (head == tail) {

head = null;

tail = null;

return;

}

CNode temp = head;

while (temp.next != tail) {

temp = temp.next;

}

tail = temp;

tail.next = head;

}

public void insertatbegining(int value) {

CNode newCNode = new CNode(value);

if (head == null) {

head = newCNode;

tail = newCNode;

newCNode.next = head;

} else {

newCNode.next = head;

head = newCNode;

tail.next = head;

}

}

public void insertatend(int value) {

add(value); // Reuse the add method since it already handles insertion at end

}

public void display() {

if (head == null) {

System.out.println("List is empty.");

return;

}

CNode temp = head;

System.out.print("Circular Linked List: ");

do {

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

temp = temp.next;

} while (temp != head);

System.out.println();

}

public static void main(String[] args) {

CircularLinkedList clist = new CircularLinkedList();

clist.add(10);

clist.add(20);

clist.add(30);

clist.add(40);

clist.add(50);

clist.add(60);

clist.add(70);

System.out.print("After creation: ");

clist.display();

clist.insertatbegining(10);

System.out.print("After inserting 10 at beginning: ");

clist.display();

clist.insertatend(60);

System.out.print("After inserting 60 at end: ");

clist.display();

}

}

A screen shot of a computer

AI-generated content may be incorrect.

42. Insertion at Specific position

class CNode {

int data;

CNode next;

CNode(int value) {

this.data = value;

this.next = null;

}

}

public class CircularLinkedList {

private CNode head;

private CNode tail;

public void add(int value) {

CNode newCNode = new CNode(value);

if (head == null) {

head = newCNode;

tail = newCNode;

tail.next = head;

} else {

tail.next = newCNode;

tail = newCNode;

tail.next = head;

}

}

public void deletefromend() {

if (head == null) {

System.out.println("List is empty.");

return;

}

if (head == tail) {

head = null;

tail = null;

return;

}

CNode temp = head;

while (temp.next != tail) {

temp = temp.next;

}

tail = temp;

tail.next = head;

}

public void insertatbegining(int value) {

CNode newCNode = new CNode(value);

if (head == null) {

head = newCNode;

tail = newCNode;

newCNode.next = head;

} else {

newCNode.next = head;

head = newCNode;

tail.next = head;

}

}

public void insertatend(int value) {

add(value);

}

public void insertatposition(int value, int pos) {

if (head == null || pos <= 1) {

insertatbegining(value);

return;

}

CNode newCNode = new CNode(value);

CNode temp = head;

for (int i = 1; i < pos - 1 && temp.next != head; i++) {

temp = temp.next;

}

newCNode.next = temp.next;

temp.next = newCNode;

if (temp == tail) {

tail = newCNode;

}

}

public void display() {

if (head == null) {

System.out.println("List is empty.");

return;

}

CNode temp = head;

System.out.print("Circular Linked List: ");

do {

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

temp = temp.next;

} while (temp != head);

System.out.println();

}

public static void main(String[] args) {

CircularLinkedList clist = new CircularLinkedList();

clist.add(10);

clist.add(20);

clist.add(30);

clist.add(40);

clist.add(50);

clist.add(60);

clist.add(70);

System.out.print("After creation: ");

clist.display();

clist.insertatbegining(10);

System.out.print("After inserting 10 at beginning: ");

clist.display();

clist.insertatend(60);

System.out.print("After inserting 60 at end: ");

clist.display();

clist.insertatposition(35, 2);

System.out.print("After inserting 35 at position 2: ");

clist.display();

}

}

A screen shot of a computer

AI-generated content may be incorrect.

43. Deletion from start

class CNode {

int data;

CNode next;

CNode(int value) {

this.data = value;

this.next = null;

}

}

public class CircularLinkedList {

private CNode head;

private CNode tail;

public void add(int value) {

CNode newCNode = new CNode(value);

if (head == null) {

head = newCNode;

tail = newCNode;

tail.next = head;

} else {

tail.next = newCNode;

tail = newCNode;

tail.next = head;

}

}

public void deletefromend() {

if (head == null) {

System.out.println("List is empty.");

return;

}

if (head == tail) {

head = null;

tail = null;

return;

}

CNode temp = head;

while (temp.next != tail) {

temp = temp.next;

}

tail = temp;

tail.next = head;

}

public void deletefrombegining()

{

if(head==null)

{

System.out.print("empty");

return;

}

if(head==tail)

{

head=null;

tail=null;

return;

}

head=head.next;

tail.next=head;

}

public void display() {

if (head == null) {

System.out.println("List is empty.");

return;

}

CNode temp = head;

System.out.print("Circular Linked List: ");

do {

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

temp = temp.next;

} while (temp != head);

System.out.println();

}

public static void main(String[] args) {

CircularLinkedList clist = new CircularLinkedList();

clist.add(10);

clist.add(20);

clist.add(30);

clist.add(40);

clist.add(50);

clist.add(60);

clist.add(70);

System.out.print("After creation: ");

clist.display();

clist.deletefrombegining();

System.out.print("After deleting from beginning: ");

clist.display();

}

A screenshot of a computer program

AI-generated content may be incorrect.

44. Deletion at End

class CNode {

int data;

CNode next;

CNode(int value) {

this.data = value;

this.next = null;

}

}

public class CircularLinkedList {

private CNode head;

private CNode tail;

public void add(int value) {

CNode newCNode = new CNode(value);

if (head == null) {

head = newCNode;

tail = newCNode;

tail.next = head;

} else {

tail.next = newCNode;

tail = newCNode;

tail.next = head;

}

}

public void deletefromend() {

if (head == null) {

System.out.println("List is empty.");

return;

}

if (head == tail) {

head = null;

tail = null;

return;

}

CNode temp = head;

while (temp.next != tail) {

temp = temp.next;

}

tail = temp;

tail.next = head;

}

public void display() {

if (head == null) {

System.out.println("List is empty.");

return;

}

CNode temp = head;

System.out.print("Circular Linked List: ");

do {

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

temp = temp.next;

} while (temp != head);

System.out.println();

}

public static void main(String[] args) {

CircularLinkedList clist = new CircularLinkedList();

clist.add(10);

clist.add(20);

clist.add(30);

clist.add(40);

clist.add(50);

clist.add(60);

clist.add(70);

System.out.print("After creation: ");

clist.display();

clist.deletefromend();

System.out.print("After deleting from end: ");

clist.display();

}

}

A screen shot of a computer program

AI-generated content may be incorrect.

45. Deletion at specific position

class CNode {

int data;

CNode next;

CNode(int value) {

this.data = value;

this.next = null;

}

}

public class CircularLinkedList {

private CNode head;

private CNode tail;

public void add(int value) {

CNode newCNode = new CNode(value);

if (head == null) {

head = newCNode;

tail = newCNode;

tail.next = head;

} else {

tail.next = newCNode;

tail = newCNode;

tail.next = head;

}

}

public void deletefrombegining() {

if (head == null) {

System.out.println("List is empty.");

return;

}

if (head == tail) {

head = null;

tail = null;

return;

}

head = head.next;

tail.next = head;

}

public void deletefromend() {

if (head == null) {

System.out.println("List is empty.");

return;

}

if (head == tail) {

head = null;

tail = null;

return;

}

CNode temp = head;

while (temp.next != tail) {

temp = temp.next;

}

tail = temp;

tail.next = head;

}

public void deleteatposition(int pos) {

if (head == null) {

System.out.println("List is empty.");

return;

}

if (pos <= 1) {

deletefrombegining();

return;

}

CNode temp = head;

for (int i = 1; i < pos - 1 && temp.next != head; i++) {

temp = temp.next;

}

// If trying to delete the last node

if (temp.next == tail) {

deletefromend();

} else {

temp.next = temp.next.next;

}

}

public void display() {

if (head == null) {

System.out.println("List is empty.");

return;

}

CNode temp = head;

System.out.print("Circular Linked List: ");

do {

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

temp = temp.next;

} while (temp != head);

System.out.println();

}

public static void main(String[] args) {

CircularLinkedList clist = new CircularLinkedList();

clist.add(10);

clist.add(20);

clist.add(30);

clist.add(40);

clist.add(50);

clist.add(60);

clist.add(70);

System.out.print("After creation: ");

clist.display();

clist.deletefromend();

System.out.print("After deleting from end: ");

clist.display();

clist.deleteatposition(1);

System.out.print("After deleting at position 1: ");

clist.display();

clist.deletefrombegining();

System.out.print("After deleting from beginning: ");

clist.display();

}

}

A screen shot of a computer

AI-generated content may be incorrect.

46. Circular Doubly Linked LIST

Insertion at start

class CDNode

{

int data;

CDNode next;

CDNode prev;

CDNode(int value)

{

this.data=value;

this.next=null;

this.prev=null;

}

}

public class CircularDoublyLinkedList

{

private CDNode head;

private CDNode tail;

public void add(int value)

{

CDNode newNode=new CDNode(value);

if(head==null)

{

head=newNode;

tail=newNode;

head.next=head;

head.prev=head;

}else{

tail.next=newNode;

newNode.prev=tail;

tail=newNode;

tail.next=head;

head.prev=tail;

}

}

public void display()

{

if(head==null)

{

System.out.print("empty");

return;

}

CDNode temp=head;

do {

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

temp=temp.next;

} while (temp!=head);

System.out.println();

}

public void insertatbegining(int value)

{

CDNode newNode=new CDNode(value);

if(head==null)

{

head=newNode;

tail = newNode;

newNode.next = newNode;

newNode.prev = newNode;

} else {

newNode.next = head;

newNode.prev = tail;

head.prev = newNode;

tail.next = newNode;

head = newNode;

}

}

public static void main(String[] args) {

CircularDoublyLinkedList cdList = new CircularDoublyLinkedList();

cdList.add(10);

cdList.add(20);

cdList.add(30);

cdList.add(40);

cdList.add(50);

System.out.print("Initial List: ");

cdList.display();

cdList.insertatbegining(5);

System.out.print("Initial List after insert at begining: ");

cdList.display();

}

}

A computer screen shot of a computer program

AI-generated content may be incorrect.

47. Insertion at End

class CDNode

{

int data;

CDNode next;

CDNode prev;

CDNode(int value)

{

this.data=value;

this.next=null;

this.prev=null;

}

}

public class CircularDoublyLinkedList

{

private CDNode head;

private CDNode tail;

public void add(int value)

{

CDNode newNode=new CDNode(value);

if(head==null)

{

head=newNode;

tail=newNode;

head.next=head;

head.prev=head;

}else{

tail.next=newNode;

newNode.prev=tail;

tail=newNode;

tail.next=head;

head.prev=tail;

}

}

public void display()

{

if(head==null)

{

System.out.print("empty");

return;

}

CDNode temp=head;

do {

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

temp=temp.next;

} while (temp!=head);

System.out.println();

}

public void insertAtEnd(int value) {

CDNode newNode = new CDNode(value);

if (head == null) {

head = newNode;

tail = newNode;

newNode.next = newNode;

newNode.prev = newNode;

} else {

newNode.next = head;

newNode.prev = tail;

tail.next = newNode;

head.prev = newNode;

tail = newNode;

}

}

public static void main(String[] args) {

CircularDoublyLinkedList cdList = new CircularDoublyLinkedList();

cdList.add(10);

cdList.add(20);

cdList.add(30);

cdList.add(40);

cdList.add(50);

System.out.print("Initial List: ");

cdList.display();

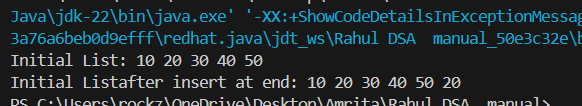
cdList.insertAtEnd(20);

System.out.print("Initial Listafter insert at end: ");

cdList.display();

}

}



48. Insertion at specific pos

class CDNode

{

int data;

CDNode next;

CDNode prev;

CDNode(int value)

{

this.data=value;

this.next=null;

this.prev=null;

}

}

public class CircularDoublyLinkedList

{

private CDNode head;

private CDNode tail;

public void add(int value)

{

CDNode newNode=new CDNode(value);

if(head==null)

{

head=newNode;

tail=newNode;

head.next=head;

head.prev=head;

}else{

tail.next=newNode;

newNode.prev=tail;

tail=newNode;

tail.next=head;

head.prev=tail;

}

}

public void display()

{

if(head==null)

{

System.out.print("empty");

return;

}

CDNode temp=head;

do {

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

temp=temp.next;

} while (temp!=head);

System.out.println();

}

public void insertatbegining(int value)

{

CDNode newNode=new CDNode(value);

if(head==null)

{

head=newNode;

tail = newNode;

newNode.next = newNode;

newNode.prev = newNode;

} else {

newNode.next = head;

newNode.prev = tail;

head.prev = newNode;

tail.next = newNode;

head = newNode;

}

}

public void insertAtEnd(int value) {

CDNode newNode = new CDNode(value);

if (head == null) {

head = newNode;

tail = newNode;

newNode.next = newNode;

newNode.prev = newNode;

} else {

newNode.next = head;

newNode.prev = tail;

tail.next = newNode;

head.prev = newNode;

tail = newNode;

}

}

public void insertAtPosition(int value, int pos) {

if (head == null || pos == 1) {

insertatbegining(value);

return;

}

CDNode newNode = new CDNode(value);

CDNode temp = head;

for(int i=1;i<pos-1;i++)

{

temp=temp.next;

}

newNode.next = temp.next;

newNode.prev = temp;

temp.next.prev = newNode;

temp.next = newNode;

if(head==tail)

{

insertAtEnd(value);

return;

}

}

public static void main(String[] args) {

CircularDoublyLinkedList cdList = new CircularDoublyLinkedList();

cdList.add(10);

cdList.add(20);

cdList.add(30);

cdList.add(40);

cdList.add(50);

System.out.print("Initial List: ");

cdList.display();

cdList.insertatbegining(5);

System.out.print("Initial List after insert at begining: ");

cdList.display();

cdList.insertAtEnd(20);

System.out.print("Initial Listafter insert at end: ");

cdList.display();

cdList.insertAtPosition(15, 3);

System.out.print("Initial List afetr insert at position: ");

cdList.display();

}

}

A screen shot of a computer

AI-generated content may be incorrect.

49.Deletion from start

class CDNode

{

int data;

CDNode next;

CDNode prev;

CDNode(int value)

{

this.data=value;

this.next=null;

this.prev=null;

}

}

public class CircularDoublyLinkedList

{

private CDNode head;

private CDNode tail;

public void add(int value)

{

CDNode newNode=new CDNode(value);

if(head==null)

{

head=newNode;

tail=newNode;

head.next=head;

head.prev=head;

}else{

tail.next=newNode;

newNode.prev=tail;

tail=newNode;

tail.next=head;

head.prev=tail;

}

}

public void display()

{

if(head==null)

{

System.out.print("empty");

return;

}

CDNode temp=head;

do {

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

temp=temp.next;

} while (temp!=head);

System.out.println();

}

public void deleteFromBeginning() {

if (head == null) {

System.out.println("List is empty!");

return;

}

if (head == tail) {

head = null;

tail = null;

return;

} else {

head = head.next;

tail.next = head;

head.prev = tail;

}

}

public static void main(String[] args) {

CircularDoublyLinkedList cdList = new CircularDoublyLinkedList();

cdList.add(10);

cdList.add(20);

cdList.add(30);

cdList.add(40);

cdList.add(50);

System.out.print("Initial List: ");

cdList.display();

cdList.deleteFromBeginning();

System.out.print("After deleting from beginning: ");

cdList.display();

}

}

A screen shot of a computer program

AI-generated content may be incorrect.

50. deletion from end

class CDNode {

int data;

CDNode next;

CDNode prev;

CDNode(int value) {

this.data = value;

this.next = null;

this.prev = null;

}

}

public class CircularDoublyLinkedList {

private CDNode head;

private CDNode tail;

public void add(int value) {

CDNode newNode = new CDNode(value);

if (head == null) {

head = newNode;

tail = newNode;

head.next = head;

head.prev = head;

} else {

tail.next = newNode;

newNode.prev = tail;

tail = newNode;

tail.next = head;

head.prev = tail;

}

}

public void display() {

if (head == null) {

System.out.println("List is empty!");

return;

}

CDNode temp = head;

do {

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

temp = temp.next;

} while (temp != head);

System.out.println();

}

public void deleteFromEnd() {

if (head == null) {

System.out.println("List is empty!");

return;

}

if (head == tail) {

head = null;

tail = null;

} else {

tail = tail.prev;

tail.next = head;

head.prev = tail;

}

}

public static void main(String[] args) {

CircularDoublyLinkedList cdList = new CircularDoublyLinkedList();

cdList.add(10);

cdList.add(20);

cdList.add(30);

cdList.add(40);

cdList.add(50);

System.out.print("Initial List: ");

cdList.display();

cdList.deleteFromEnd();

System.out.print("After deleting from end: ");

cdList.display();

}

}

A screen shot of a computer program

AI-generated content may be incorrect.

51. Deletion at specific position

class CDNode {

int data;

CDNode next;

CDNode prev;

CDNode(int value) {

this.data = value;

this.next = null;

this.prev = null;

}

}

public class CircularDoublyLinkedList {

private CDNode head;

private CDNode tail;

public void add(int value) {

CDNode newNode = new CDNode(value);

if (head == null) {

head = tail = newNode;

head.next = head.prev = head;

} else {

newNode.prev = tail;

newNode.next = head;

tail.next = newNode;

head.prev = newNode;

tail = newNode;

}

}

public void deleteFromBeginning() {

if (head == null) {

System.out.println("List is empty!");

return;

}

if (head == tail) {

head = tail = null;

} else {

head = head.next;

head.prev = tail;

tail.next = head;

}

}

public void deleteFromEnd() {

if (head == null) {

System.out.println("List is empty!");

return;

}

if (head == tail) {

head = tail = null;

} else {

tail = tail.prev;

tail.next = head;

head.prev = tail;

}

}

public void deleteFromPosition(int pos) {

if (head == null) {

System.out.println("List is empty, can't delete at position " + pos);

return;

}

if (pos == 1) {

deleteFromBeginning();

return;

}

CDNode current = head;

int count = 1;

do {

if (count == pos) {

if (current == tail) {

deleteFromEnd();

} else {

current.prev.next = current.next;

current.next.prev = current.prev;

}

return;

}

current = current.next;

count++;

} while (current != head);

System.out.println("Position " + pos + " is out of bounds.");

}

public void display() {

if (head == null) {

System.out.println("List is empty!");

return;

}

CDNode temp = head;

do {

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

temp = temp.next;

} while (temp != head);

System.out.println();

}

public static void main(String[] args) {

CircularDoublyLinkedList cdList = new CircularDoublyLinkedList();

cdList.add(10);

cdList.add(20);

cdList.add(30);

cdList.add(40);

cdList.add(50);

System.out.print("Initial List: ");

cdList.display();

cdList.deleteFromEnd();

System.out.print("After deleting from end: ");

cdList.display();

cdList.deleteFromPosition(2);

System.out.print("After deleting from position 2: ");

cdList.display();

cdList.deleteFromPosition(10); // Test out-of-bounds

}

}

A screen shot of a computer

AI-generated content may be incorrect.

LAB\_5 (28-02-25)

52. REVERSING A SLL

class Node {

int data;

Node next;

Node(int value) {

this.data = value;

this.next = null;

}

}

class SinglyLinkedList {

private Node head;

private Node tail;

// Add a node at the end

public void add(int value) {

Node newNode = new Node(value);

if (head == null) {

head = newNode;

tail = newNode;

} else {

tail.next = newNode;

tail = newNode;

}

}

void reverse() {

Node prev = null;

Node temp = head;

Node next = null;

while (temp != null)

{

next = temp.next;

temp.next = prev;

prev = temp;

temp = next;

}

head = prev;

}

// Display the list

public void display() {

if (head == null) {

System.out.println("Linked list is empty.");

return;

}

Node temp = head;

System.out.print("Linked list: ");

while (temp != null) {

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

temp = temp.next;

}

System.out.println();

}

}

// Main class

public class Main {

public static void main(String[] args) {

SinglyLinkedList list = new SinglyLinkedList();

list.add(10);

list.add(20);

list.add(30);

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

list.display();

list.reverse();

list.display();

}

}

A black screen with white text

AI-generated content may be incorrect.

53. Reversing a DLL

// Node class for Doubly Linked List

class DNode {

int data;

DNode next;

DNode prev;

DNode(int value) {

this.data = value;

this.next = null;

this.prev = null;

}

}

// Doubly Linked List class

public class DoublyLinkedList {

private DNode head;

private DNode tail;

// Add node to the end

public void add(int value) {

DNode newDNode = new DNode(value);

if (head == null) {

head = newDNode;

tail = newDNode;

} else {

tail.next = newDNode;

newDNode.prev = tail;

tail = newDNode;

}

}

public void reverse()

{

if(head==null)

{

System.out.println("empty");

return;

}

DNode temp=tail;

while(temp!=null)

{

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

temp=temp.prev;

}

System.out.println();

}

// Display the list

public void display() {

if (head == null) {

System.out.println("Linked list is empty.");

return;

}

DNode temp = head;

System.out.print("Linked list: ");

while (temp != null) {

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

temp = temp.next;

}

System.out.println();

}

// Main method

public static void main(String[] args) {

DoublyLinkedList dlist = new DoublyLinkedList();

dlist.add(10);

dlist.add(20);

dlist.add(30);

dlist.add(40);

dlist.display();

dlist.reverse();

}

}

A screen shot of a computer program

AI-generated content may be incorrect.

54. Reversing a Circular SLL

class CNode {

int data;

CNode next;

CNode(int value) {

this.data = value;

this.next = null;

}

}

public class CircularLinkedList {

private CNode head;

private CNode tail;

public void add(int value) {

CNode newCNode = new CNode(value);

if (head == null) {

head = newCNode;

tail = newCNode;

tail.next = head;

} else {

tail.next = newCNode;

tail = newCNode;

tail.next = head;

}

}

public void reverse() {

if (head == null || head.next == head) {

return; // no need to reverse if list is empty or has only 1 node

}

CNode prev = tail;

CNode current = head;

CNode next;

do {

next = current.next;

current.next = prev;

prev = current;

current = next;

} while (current != head);

tail = head;

head = prev;

}

public void display() {

if (head == null) {

System.out.println("List is empty.");

return;

}

CNode temp = head;

System.out.print("Circular Linked List: ");

do {

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

temp = temp.next;

} while (temp != head);

System.out.println();

}

public static void main(String[] args) {

CircularLinkedList clist = new CircularLinkedList();

clist.add(10);

clist.add(20);

clist.add(30);

clist.add(40);

clist.add(50);

clist.add(60);

clist.add(70);

System.out.print("After creation: ");

clist.display();

clist.reverse();

System.out.print("After reversing the list: ");

clist.display();

}

}

A screen shot of a computer

AI-generated content may be incorrect.

55. Reversing of Circular DLL

class CDNode {

int data;

CDNode next;

CDNode prev;

CDNode(int value) {

this.data = value;

this.next = null;

this.prev = null;

}

}

public class CircularDoublyLinkedList {

private CDNode head;

private CDNode tail;

public void add(int value) {

CDNode newNode = new CDNode(value);

if (head == null) {

head = tail = newNode;

head.next = head.prev = head;

} else {

newNode.prev = tail;

newNode.next = head;

tail.next = newNode;

head.prev = newNode;

tail = newNode;

}

}

public void Reverse() {

if (head == null)

{

System.out.println("List is empty.");

return;

}

CDNode temp = tail;

do {

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

temp = temp.prev;

} while (temp != tail);

System.out.println();

}

public void display() {

if (head == null) {

System.out.println("List is empty!");

return;

}

CDNode temp = head;

do {

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

temp = temp.next;

} while (temp != head);

System.out.println();

}

public static void main(String[] args) {

CircularDoublyLinkedList cdList = new CircularDoublyLinkedList();

cdList.add(10);

cdList.add(20);

cdList.add(30);

cdList.add(40);

cdList.add(50);

System.out.print("Initial List: ");

cdList.display();

cdList.Reverse();

}

}

A screen shot of a computer

AI-generated content may be incorrect.

LAB\_6 (7-03-25)

56. Stack operation in java

Implementation stack using array

import java.util.Scanner;

class Stack {

private int[] stack;

private int top;

private int max;

public Stack(int size) {

max = size;

stack = new int[max];

top = -1;

}

public void push(int val) {

if (top == max - 1) {

System.out.println("STACK OVERFLOW! Cannot push " + val);

} else {

stack[++top] = val;

System.out.println(val + " pushed onto the stack.");

}

}

public int pop() {

if (top == -1) {

System.out.println("STACK UNDERFLOW! No elements to pop.");

return -1;

} else {

System.out.println(stack[top] + " popped from the stack.");

return stack[top--];

}

}

public int peek() {

if (top == -1) {

System.out.println("STACK IS EMPTY! No elements to peek.");

return -1;

} else {

return stack[top];

}

}

public void display() {

if (top == -1) {

System.out.println("STACK IS EMPTY!");

} else {

System.out.println("Stack elements:");

for (int i = top; i >= 0; i--) {

System.out.println(stack[i]);

}

}

}

}

public class StackDemo {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

Stack stack = new Stack(3);

int option, val;

do {

// Displaying menu options

System.out.println("\n\*\*\*\*\* MAIN MENU \*\*\*\*\*");

System.out.println("1. PUSH");

System.out.println("2. POP");

System.out.println("3. PEEK");

System.out.println("4. DISPLAY");

System.out.println("5. EXIT");

System.out.print("Enter your option: ");

option = scanner.nextInt();

switch (option) {

case 1:

System.out.print("Enter the number to be pushed on stack: ");

val = scanner.nextInt();

stack.push(val);

break;

case 2:

stack.pop();

break;

case 3:

val = stack.peek();

if (val != -1) {

System.out.println("The value stored at the top of the stack is: " + val);

}

break;

case 4:

stack.display();

break;

}

} while (option != 5);

scanner.close();

}

A screenshot of a computer program

AI-generated content may be incorrect.

57. Implementation of stack using linked list

class StackNode {

int data;

StackNode next;

public StackNode(int data) {

this.data = data;

this.next = null;

}

}

public class Stack {

private StackNode top;

public Stack() {

this.top = null;

}

public void push(int ele) {

StackNode newNode = new StackNode(ele);

newNode.next = top;

top = newNode;

System.out.println(ele + " pushed to stack");

}

public int pop() {

if (top == null) {

System.out.println("STACK IS EMPTY");

return -1;

}

int popped = top.data;

top = top.next;

System.out.println(popped + " is deleted");

return popped;

}

public void display() {

if (top == null) {

System.out.println("STACK IS EMPTY");

return;

}

StackNode temp = top;

System.out.println("Stack elements:");

while (temp != null) {

System.out.println(temp.data);

temp = temp.next;

}

}

public static void main(String[] args) {

Stack stack = new Stack();

stack.push(10);

stack.push(20);

stack.push(30);

stack.display();

stack.pop();

stack.display();

}

}

A screen shot of a computer

AI-generated content may be incorrect.

58. Infix to Postfix conversion

import java.util.Scanner;

import java.util.Stack;

public class expression2 {

static int getPrecedence(char ch) {

if (ch == '+' || ch == '-') return 1;

if (ch == '\*' || ch == '/' || ch == '%') return 2;

if (ch == '^') return 3;

return -1;

}

static String convertToPostfix(String infix) {

Stack<String> postfixStack = new Stack<>();

Stack<Character> operatorStack = new Stack<>();

int i = 0;

while (i < infix.length()) {

char ch = infix.charAt(i);

if (Character.isDigit(ch)) {

String num = "";

while (i < infix.length() && Character.isDigit(infix.charAt(i))) {

num += infix.charAt(i);

i++;

}

postfixStack.push(num);

continue;

}

else if (Character.isLetter(ch)) {

postfixStack.push(String.valueOf(ch));

}

else if (ch == '(') {

operatorStack.push(ch);

}

else if (ch == ')') {

while (!operatorStack.isEmpty() && operatorStack.peek() != '(') {

postfixStack.push(String.valueOf(operatorStack.pop()));

}

operatorStack.pop();

}

else {

while (!operatorStack.isEmpty() && getPrecedence(ch) <= getPrecedence(operatorStack.peek()) && operatorStack.peek() != '(') {

postfixStack.push(String.valueOf(operatorStack.pop()));

}

operatorStack.push(ch);

}

i++;

}

while (!operatorStack.isEmpty()) {

postfixStack.push(String.valueOf(operatorStack.pop()));

}

Stack<String> reversedStack = new Stack<>();

while (!postfixStack.isEmpty()) {

reversedStack.push(postfixStack.pop());

}

String result = "";

while (!reversedStack.isEmpty()) {

result += reversedStack.pop() ;

}

return result;

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.print("Enter an infix expression: ");

String infix = scanner.nextLine();

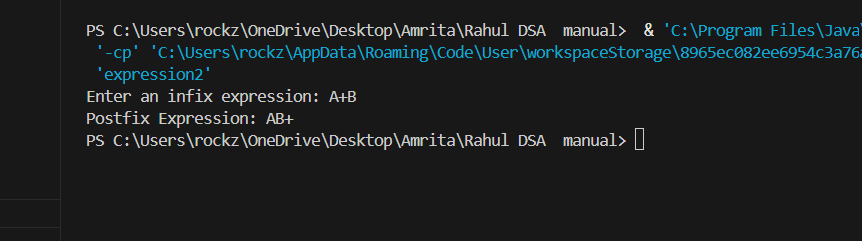
String postfix = convertToPostfix(infix);

System.out.println("Postfix Expression: " + postfix);

scanner.close();

}

}



59. Postfix Expression Evaluation code

import java.util.Stack;

import java.util.Scanner;

public class Evaluatepostfix {

public static double evaluatePostfix(String postfix) {

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

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

char ch = postfix.charAt(i);

if (Character.isDigit(ch)) {

stack.push((double)(ch - '0'));

}

else {

double operand2 = stack.pop();

double operand1 = stack.pop();

switch (ch) {

case '+': stack.push(operand1 + operand2); break;

case '-': stack.push(operand1 - operand2); break;

case '\*': stack.push(operand1 \* operand2); break;

case '/': stack.push(operand1 / operand2); break;

case '%': stack.push(operand1 % operand2); break;

case '^': stack.push(Math.pow(operand1, operand2)); break;

default: throw new IllegalArgumentException("Invalid operator: " + ch);

}

}

}

return stack.pop();

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.print("Enter a postfix expression: ");

String postfix = scanner.nextLine();

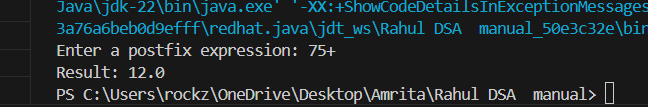
double result = evaluatePostfix(postfix);

System.out.println("Result: " + result);

scanner.close();

}

}



60.Infix to Prefix Expression

import java.util.Stack;

public class InfixToPrefix {

public static String infixToPrefix(String infix) {

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

StringBuilder prefix = new StringBuilder();

for (int i = infix.length() - 1; i >= 0; i--) {

char ch = infix.charAt(i);

if (Character.isLetterOrDigit(ch)) {

prefix.append(ch);

} else if (ch == ')') {

stack.push(ch);

} else if (ch == '(') {

while (stack.peek() != ')') {

prefix.append(stack.pop());

}

stack.pop();

} else {

while (!stack.isEmpty() && precedence(ch) < precedence(stack.peek())) {

prefix.append(stack.pop());

}

stack.push(ch);

}

}

while (!stack.isEmpty()) {

prefix.append(stack.pop());

}

return prefix.reverse().toString();

}

private static int precedence(char ch) {

switch (ch) {

case '+':

case '-':

return 1;

case '\*':

case '/':

return 2;

case '^':

return 3;

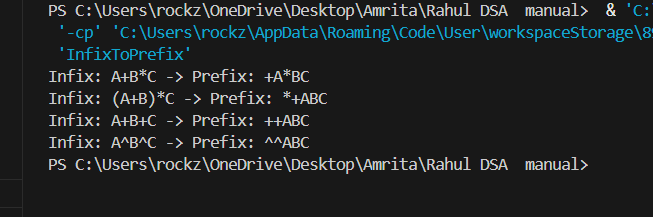
default:

return 0;

}

}

}



61. Prefix Evaluation Code

import java.util.Stack;

public class PrefixEvaluation {

public static int evaluatePrefix(String prefix) {

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

for (int i = prefix.length() - 1; i >= 0; i--) {

char ch = prefix.charAt(i);

if (Character.isDigit(ch)) {

stack.push(ch - '0');

} else {

int a = stack.pop();

int b = stack.pop();

switch (ch) {

case '+':

stack.push(a + b);

break;

case '-':

stack.push(a - b);

break;

case '\*':

stack.push(a \* b);

break;

case '/':

stack.push(a / b);

break;

}

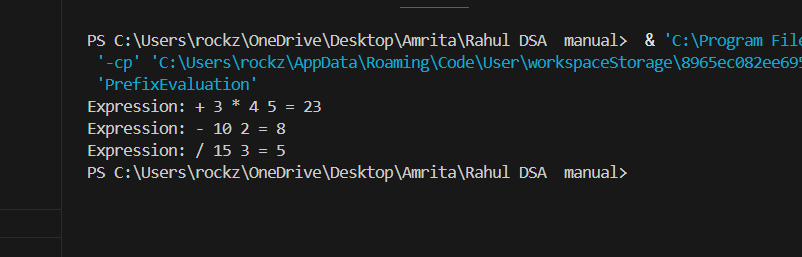
}

}

return stack.pop();

}

}



62. ALL in one code (infix,postfix,prefix)

import java.util.Stack;

public class ExpressionConverterEvaluator {

// Infix to Postfix Conversion

public static String infixToPostfix(String infix) {

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

StringBuilder postfix = new StringBuilder();

for (char ch : infix.toCharArray()) {

if (Character.isLetterOrDigit(ch)) {

postfix.append(ch);

} else if (ch == '(') {

stack.push(ch);

} else if (ch == ')') {

while (stack.peek() != '(') {

postfix.append(stack.pop());

}

stack.pop();

} else {

while (!stack.isEmpty() && precedence(ch) <= precedence(stack.peek())) {

postfix.append(stack.pop());

}

stack.push(ch);

}

}

while (!stack.isEmpty()) {

postfix.append(stack.pop());

}

return postfix.toString();

}

// Infix to Prefix Conversion

public static String infixToPrefix(String infix) {

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

StringBuilder prefix = new StringBuilder();

for (int i = infix.length() - 1; i >= 0; i--) {

char ch = infix.charAt(i);

if (Character.isLetterOrDigit(ch)) {

prefix.append(ch);

} else if (ch == ')') {

stack.push(ch);

} else if (ch == '(') {

while (stack.peek() != ')') {

prefix.append(stack.pop());

}

stack.pop();

} else {

while (!stack.isEmpty() && precedence(ch) < precedence(stack.peek())) {

prefix.append(stack.pop());

}

stack.push(ch);

}

}

while (!stack.isEmpty()) {

prefix.append(stack.pop());

}

return prefix.reverse().toString();

}

// Postfix Expression Evaluation

public static int evaluatePostfix(String postfix) {

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

for (char ch : postfix.toCharArray()) {

if (Character.isDigit(ch)) {

stack.push(ch - '0');

} else {

int b = stack.pop();

int a = stack.pop();

switch (ch) {

case '+':

stack.push(a + b);

break;

case '-':

stack.push(a - b);

break;

case '\*':

stack.push(a \* b);

break;

case '/':

stack.push(a / b);

break;

}

}

}

return stack.pop();

}

// Prefix Expression Evaluation

public static int evaluatePrefix(String prefix) {

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

for (int i = prefix.length() - 1; i >= 0; i--) {

char ch = prefix.charAt(i);

if (Character.isDigit(ch)) {

stack.push(ch - '0');

} else {

int a = stack.pop();

int b = stack.pop();

switch (ch) {

case '+':

stack.push(a + b);

break;

case '-':

stack.push(a - b);

break;

case '\*':

stack.push(a \* b);

break;

case '/':

stack.push(a / b);

break;

}

}

}

return stack.pop();

}

// Helper method to determine operator precedence

private static int precedence(char ch) {

switch (ch) {

case '+':

case '-':

return 1;

case '\*':

case '/':

return 2;

case '^':

return 3;

default:

return 0;

}

}

public static void main(String[] args) {

String infixExpression = "a+b\*(c^d-e)^(f+g\*h)-i";

String postfixExpression = infixToPostfix(infixExpression);

String prefixExpression = infixToPrefix(infixExpression);

System.out.println("Infix Expression: " + infixExpression);

System.out.println("Postfix Expression: " + postfixExpression);

System.out.println("Prefix Expression: " + prefixExpression);

// Example evaluation (assuming single-digit numbers for simplicity)

String postfixExample = "23\*5+"; // Equivalent to (2\*3)+5

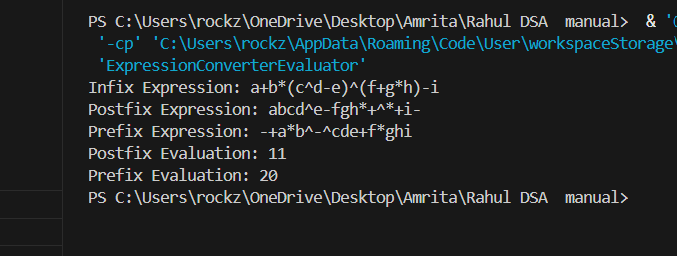
String prefixExample = "\*+234"; // Equivalent to (2+(3\*4))

System.out.println("Postfix Evaluation: " + evaluatePostfix(postfixExample));

System.out.println("Prefix Evaluation: " + evaluatePrefix(prefixExample));

}

}



LAB\_7 (12-03-25)

63. Queue Using Array

import java.util.Scanner;

class QueueUsingArray {

int[] ar = new int[10];

int n = 10;

int front = -1;

int rear = -1;

void enqueue(int item) {

if (rear == n - 1) {

System.out.println("Overflow!");

return;

}

if (front == -1 && rear == -1) {

front = 0;

rear = 0;

} else {

rear++;

}

ar[rear] = item;

System.out.println("Element inserted.");

}

void dequeue() {

if (front == -1 || front > rear) {

System.out.println("Underflow!");

return;

}

System.out.println("Element deleted from queue is: " + ar[front]);

if (front == rear) {

front = -1;

rear = -1;

} else {

front++;

}

}

void display() {

if (front == -1 || front > rear) {

System.out.println("Queue is empty.");

return;

}

System.out.print("Elements are: ");

for (int i = front; i <= rear; i++) {

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

}

System.out.println();

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

QueueUsingArray q = new QueueUsingArray();

System.out.println("Queue Operations:");

System.out.println("1: Enqueue");

System.out.println("2: Dequeue");

System.out.println("3: Display");

System.out.println("4: Exit");

int choice;

do {

System.out.print("Enter your choice: ");

choice = sc.nextInt();

switch (choice) {

case 1:

System.out.print("Enter element to insert: ");

int item = sc.nextInt();

q.enqueue(item);

break;

case 2:

q.dequeue();

break;

case 3:

q.display();

break;

case 4:

System.out.println("Exiting...");

break;

default:

System.out.println("Invalid choice. Try again.");

}

} while (choice != 4);

sc.close();

}

}

A screenshot of a computer

AI-generated content may be incorrect.

64. Queue using linked list

class Node {

int data;

Node next;

Node(int data) {

this.data = data;

this.next = null;

}

}

public class Queue {

private Node front;

private Node rear;

public void EnQueue(int value) {

Node newNode = new Node(value);

if (front == null) {

front = newNode;

rear = newNode;

} else {

rear.next = newNode;

rear = newNode;

}

}

public void DeQueue() {

if (front == null) {

System.out.println("Queue is empty, can't delete at beginning. (Underflow)");

return;

}

System.out.println("The element to be deleted is " + front.data);

if (front == rear) {

front = null;

rear = null;

} else {

front = front.next;

}

}

public void display() {

if (front == null) {

System.out.println("Queue is empty!");

return;

}

Node temp = front;

while (temp != null) {

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

temp = temp.next;

}

System.out.println();

}

public static void main(String[] args) {

Queue queue = new Queue();

queue.EnQueue(10);

queue.EnQueue(20);

queue.EnQueue(30);

queue.EnQueue(40);

queue.EnQueue(50);

System.out.println("After creation:");

queue.display();

queue.DeQueue();

System.out.println("After deleting from beginning:");

queue.display();

}

}

A screen shot of a computer

AI-generated content may be incorrect.

65. Circular queue using Array

Circular queue using Linked list

class CNode {

int data;

CNode next;

CNode(int value) {

this.data = value;

this.next = null;

}

}

public class CircularQueue {

private CNode front, rear;

public void enqueue(int value) {

CNode newNode = new CNode(value);

if (front == null) {

front = rear = newNode;

rear.next = front;

} else {

rear.next = newNode;

rear = newNode;

rear.next = front;

}

System.out.println("Inserted: " + value);

}

public void dequeue() {

if (front == null) {

System.out.println("Queue is empty, can't delete.");

return;

}

System.out.println("Element deleted: " + front.data);

if (front == rear) {

front = rear = null;

} else {

front = front.next;

rear.next = front;

}

}

public void display() {

if (front == null) {

System.out.println("Queue is empty.");

return;

}

System.out.print("Queue elements: ");

CNode temp = front;

do {

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

temp = temp.next;

} while (temp != front);

System.out.println("(back to front)");

}

public static void main(String[] args) {

CircularQueue queue = new CircularQueue();

queue.enqueue(10);

queue.enqueue(20);

queue.enqueue(30);

queue.enqueue(40);

queue.enqueue(50);

System.out.print("After creation: ");

queue.display();

queue.enqueue(60);

System.out.print("After inserting 60: ");

queue.display();

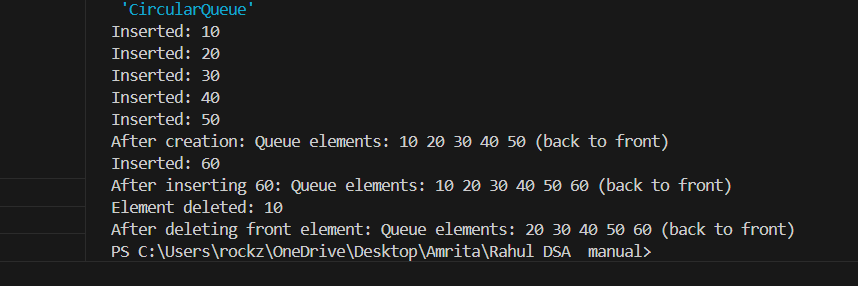
queue.dequeue();

System.out.print("After deleting front element: ");

queue.display();

}

}



LAB\_8 (28-03-25)

66. Priority Queue using Array

class PNode {

int data, priority;

PNode next;

PNode(int data, int priority) {

this.data = data;

this.priority = priority;

this.next = null;

}

}

public class PriorityQueue {

private PNode front;

public PriorityQueue() {

front = null;

}

public void enqueue(int data, int priority) {

PNode newNode = new PNode(data, priority);

if (front == null || priority < front.priority) {

newNode.next = front;

front = newNode;

} else {

PNode temp = front;

while (temp.next != null && temp.next.priority <= priority) {

temp = temp.next;

}

newNode.next = temp.next;

temp.next = newNode;

}

}

public int dequeue() {

if (isEmpty()) {

System.out.println("Priority Queue is empty!");

return -1;

}

int data = front.data;

front = front.next;

return data;

}

public void display() {

if (isEmpty()) {

System.out.println("Priority Queue is empty!");

return;

}

PNode temp = front;

while (temp != null) {

System.out.print(temp.data + "(" + temp.priority + ") -> ");

temp = temp.next;

}

System.out.println("null");

}

public boolean isEmpty() {

return front == null;

}

public static void main(String[] args) {

PriorityQueue pq = new PriorityQueue();

pq.enqueue(10, 2);

pq.enqueue(20, 1);

pq.enqueue(30, 3);

pq.enqueue(40, 0);

System.out.println("Priority Queue after enqueuing elements:");

pq.display();

System.out.println("Dequeued element: " + pq.dequeue());

System.out.println("Priority Queue after dequeuing an element:");

pq.display();

}

}

A computer screen with white text

AI-generated content may be incorrect.

67. Code for Priority Queue Descending

import java.util.Scanner;

class Node {

int data;

int priority;

Node next;

Node(int data, int priority) {

this.data = data;

this.priority = priority;

this.next = null;

}

}

public class PriorityQueueDescending {

static Node front = null;

static void enqueue(int data, int priority) {

Node newNode = new Node(data, priority);

if (front == null || priority > front.priority) {

newNode.next = front;

front = newNode;

} else {

Node temp = front;

while (temp.next != null && temp.next.priority >= priority) {

temp = temp.next;

}

newNode.next = temp.next;

temp.next = newNode;

}

System.out.println("Element inserted.");

}

static void dequeue() {

if (front == null) {

System.out.println("Underflow! Queue is empty.");

return;

}

System.out.println("Element deleted from queue is: " + front.data);

front = front.next;

}

static void display() {

if (front == null) {

System.out.println("Queue is empty.");

return;

}

System.out.println("Elements in queue (in priority order):");

Node temp = front;

while (temp != null) {

System.out.println("Value: " + temp.data + " | Priority: " + temp.priority);

temp = temp.next;

}

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

int ch;

do {

System.out.println("\n1: Insert (Enqueue)");

System.out.println("2: Delete (Dequeue)");

System.out.println("3: Display Queue");

System.out.println("4: Exit");

System.out.print("Enter your choice: ");

ch = sc.nextInt();

switch (ch) {

case 1:

System.out.print("Enter element to insert: ");

int data = sc.nextInt();

System.out.print("Enter priority: ");

int priority = sc.nextInt();

enqueue(data, priority);

break;

case 2:

dequeue();

break;

case 3:

display();

break;

case 4:

System.out.println("Exiting...");

break;

default:

System.out.println("Invalid choice.");

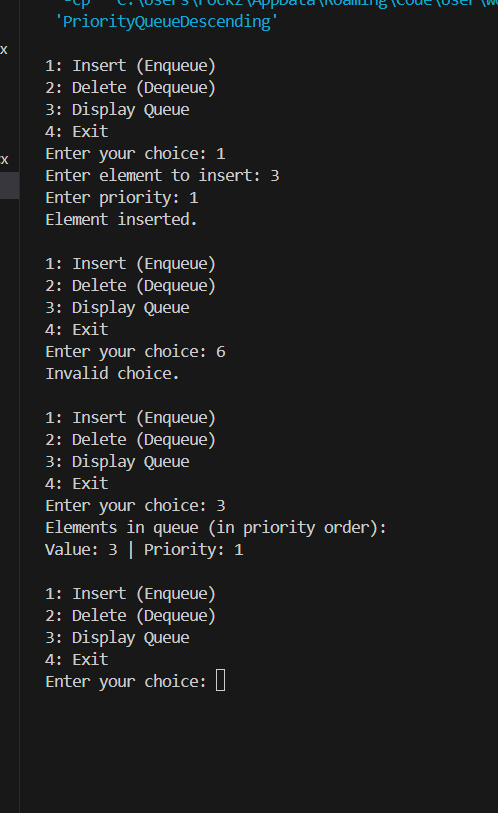
}

} while (ch != 4);

sc.close();

}

}



68. Ascending Priority Queue

import java.util.Scanner;

class Node {

int data;

int priority;

Node next;

Node(int data, int priority) {

this.data = data;

this.priority = priority;

this.next = null;

}

}

public class PriorityQueueAscending {

static Node front = null;

static void enqueue(int data, int priority) {

Node newNode = new Node(data, priority);

if (front == null || priority < front.priority) {

newNode.next = front;

front = newNode;

} else {

Node temp = front;

while (temp.next != null && temp.next.priority <= priority) {

temp = temp.next;

}

newNode.next = temp.next;

temp.next = newNode;

}

System.out.println("Element inserted.");

}

static void dequeue() {

if (front == null) {

System.out.println("Underflow! Queue is empty.");

return;

}

System.out.println("Element deleted from queue is: " + front.data);

front = front.next;

}

static void display() {

if (front == null) {

System.out.println("Queue is empty.");

return;

}

System.out.println("Elements in queue (in priority order):");

Node temp = front;

while (temp != null) {

System.out.println("Value: " + temp.data + " | Priority: " + temp.priority);

temp = temp.next;

}

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

int ch;

do {

System.out.println("\n1: Insert (Enqueue)");

System.out.println("2: Delete (Dequeue)");

System.out.println("3: Display Queue");

System.out.println("4: Exit");

System.out.print("Enter your choice: ");

ch = sc.nextInt();

switch (ch) {

case 1:

System.out.print("Enter element to insert: ");

int data = sc.nextInt();

System.out.print("Enter priority: ");

int priority = sc.nextInt();

enqueue(data, priority);

break;

case 2:

dequeue();

break;

case 3:

display();

break;

case 4:

System.out.println("Exiting...");

break;

default:

System.out.println("Invalid choice.");

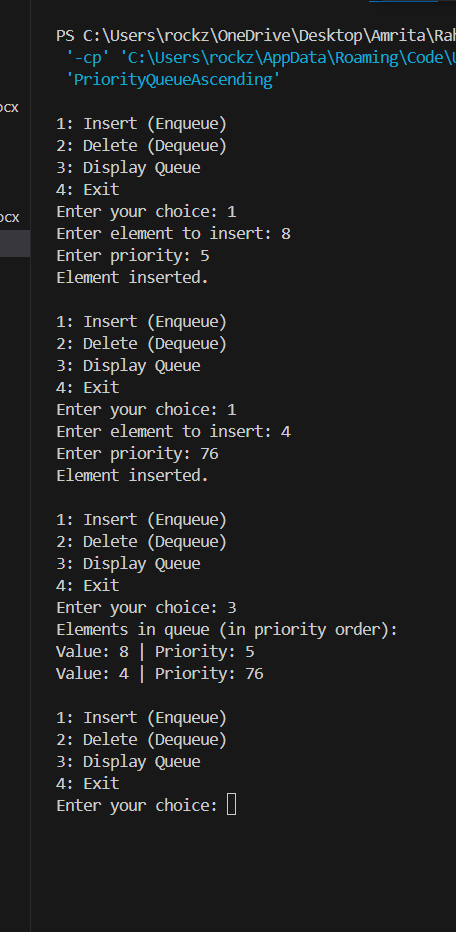
}

} while (ch != 4);

sc.close();

}

}



LAB\_9 (4-4-25)

69. binary tree using array

class BinaryTreeArray

{

int[] tree;

int size;

public BinaryTreeArray(int capacity)

{

tree = new int[capacity];

size = 0;

}

public void add(int value)

{

if (size < tree.length)

{

tree[size] = value;

size++;

}

else

{

System.out.println("Tree is full");

}

}

public void inorder(int index)

{

if (index >= size) return;

inorder(2 \* index + 1);

System.out.print(tree[index] + "->");

inorder(2 \* index + 2);

}

public void preorder(int index)

{

if (index >= size) return;

System.out.print(tree[index] + "->");

preorder(2 \* index + 1);

preorder(2 \* index + 2);

}

public void postorder(int index)

{

if (index >= size) return;

postorder(2 \* index + 1);

postorder(2 \* index + 2);

System.out.print(tree[index] + "->");

}

public static void main(String[] args)

{

BinaryTreeArray tree = new BinaryTreeArray(10);

tree.add(1);

tree.add(12);

tree.add(9);

tree.add(5);

tree.add(6);

System.out.println("Inorder traversal");

tree.inorder(0);

System.out.println("\nPreorder traversal");

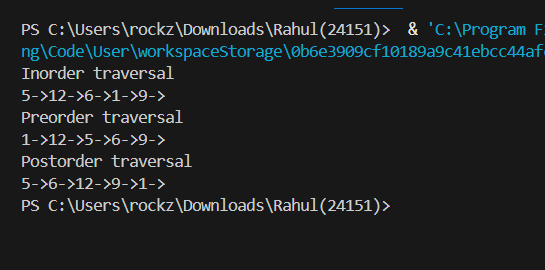
tree.preorder(0);

System.out.println("\nPostorder traversal");

tree.postorder(0);

}

}



70. **binary tree using Linked Lits**

class Node

{

int item;

Node left;

Node right;

public Node(int key)

{

item = key;

left = null;

right = null;

}

}

class BinaryTree

{

Node root;

BinaryTree()

{

root = null;

}

void postorder(Node node)

{

if (node == null)

return;

postorder(node.left);

postorder(node.right);

System.out.print(node.item + "->");

}

void inorder(Node node)

{

if (node == null)

return;

inorder(node.left);

System.out.print(node.item + "->");

inorder(node.right);

}

void preorder(Node node)

{

if (node == null)

return;

System.out.print(node.item + "->");

preorder(node.left);

preorder(node.right);

}

public static void main(String[] args)

{

BinaryTree tree = new BinaryTree();

tree.root = new Node(1);

tree.root.left = new Node(12);

tree.root.right = new Node(9);

tree.root.left.left = new Node(5);

tree.root.left.right = new Node(6);

System.out.println("Inorder traversal");

tree.inorder(tree.root);

System.out.println("\nPreorder traversal ");

tree.preorder(tree.root);

System.out.println("\nPostorder traversal");

tree.postorder(tree.root);

}

}

A screen shot of a computer

AI-generated content may be incorrect.

71. Tree Traversals (Pre,inorder,postorder)

// Node definition

class Node {

    int value;

    Node left, right;

    Node(int value) {

        this.value = value;

        left = right = null;

    }

}

// Binary tree with in-order, pre-order, post-order traversals

public class BinaryTree {

    Node root;

    // In-order: left → root → right

    void inOrder(Node node) {

        if (node == null) return;

        inOrder(node.left);

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

        inOrder(node.right);

    }

    // Pre-order: root → left → right

    void preOrder(Node node) {

        if (node == null) return;

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

        preOrder(node.left);

        preOrder(node.right);

    }

    // Post-order: left → right → root

    void postOrder(Node node) {

        if (node == null) return;

        postOrder(node.left);

        postOrder(node.right);

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

    }

    public static void main(String[] args) {

        BinaryTree tree = new BinaryTree();

        // Manually build:

        tree.root = new Node(1);

        tree.root.left = new Node(2);

        tree.root.right = new Node(3);

        tree.root.left.left = new Node(4);

        tree.root.left.right = new Node(5);

        System.out.print("In-order: ");

        tree.inOrder(tree.root);

        System.out.println();

        System.out.print("Pre-order: ");

        tree.preOrder(tree.root);

        System.out.println();

        System.out.print("Post-order: ");

        tree.postOrder(tree.root);

        System.out.println();

    }

}

A screen shot of a computer

AI-generated content may be incorrect.

Lab\_10 (11-04-25)

72. **Binary search Tree using array**

// BSTArray class to implement a binary search tree using an array

class BSTArray {

Integer[] tree; // Array to store the tree elements

int capacity; // Maximum number of nodes the tree can hold

// Constructor to initialize the tree array with a specific size

public BSTArray(int size) {

capacity = size;

tree = new Integer[capacity]; // Initially, all values are null

}

// Public method to start insertion from root (index 0)

public void insert(int key) {

insertAt(0, key); // Start recursive insertion at root

}

// Recursive helper to insert at a specific index

private void insertAt(int index, int key) {

// If index goes out of bounds, show message

if (index >= capacity) {

System.out.println("Tree capacity exceeded");

return;

}

// If position is empty, insert key here

if (tree[index] == null) {

tree[index] = key;

return;

}

// If key is less than or equal, insert to left child

if (key <= tree[index]) {

insertAt(2 \* index + 1, key); // Left child index = 2\*i + 1

} else {

insertAt(2 \* index + 2, key); // Right child index = 2\*i + 2

}

}

// Public method to search for a key in the tree

public boolean search(int key) {

return searchAt(0, key); // Start from root

}

// Recursive helper to search starting from a given index

private boolean searchAt(int index, int key) {

if (index >= capacity || tree[index] == null) {

return false; // Reached beyond leaf or empty node

}

if (tree[index] == key) {

return true; // Key found

} else if (key < tree[index]) {

return searchAt(2 \* index + 1, key); // Search in left subtree

} else {

return searchAt(2 \* index + 2, key); // Search in right subtree

}

}

// Public method to perform inOrder traversal

public void inOrder() {

System.out.print("The inOrder traversal is: ");

inOrder(0); // Start from root

System.out.println();

}

// Recursive inOrder: Left -> Root -> Right

private void inOrder(int index) {

if (index >= capacity || tree[index] == null) return;

inOrder(2 \* index + 1); // Visit left subtree

System.out.print(tree[index] + " "); // Visit root

inOrder(2 \* index + 2); // Visit right subtree

}

// Public method to perform preOrder traversal

public void preOrder() {

System.out.print("The preOrder traversal is: ");

preOrder(0);

System.out.println();

}

// Recursive preOrder: Root -> Left -> Right

private void preOrder(int index) {

if (index >= capacity || tree[index] == null) return;

System.out.print(tree[index] + " "); // Visit root

preOrder(2 \* index + 1); // Visit left subtree

preOrder(2 \* index + 2); // Visit right subtree

}

// Public method to perform postOrder traversal

public void postOrder() {

System.out.print("The postOrder traversal is: ");

postOrder(0);

System.out.println();

}

// Recursive postOrder: Left -> Right -> Root

private void postOrder(int index) {

if (index >= capacity || tree[index] == null) return;

postOrder(2 \* index + 1); // Visit left subtree

postOrder(2 \* index + 2); // Visit right subtree

System.out.print(tree[index] + " "); // Visit root

}

}

// Main class to test BSTArray

public class BinarySearchTreeArray {

public static void main(String[] args) {

BSTArray bst = new BSTArray(31); // Initialize array-based BST with 31 capacity

// Insert elements

bst.insert(10);

bst.insert(15);

bst.insert(5);

bst.insert(8);

bst.insert(18);

bst.insert(12);

bst.insert(10); // Duplicate (will be placed on left side again)

// Display all traversals

bst.preOrder(); // Root -> Left -> Right

bst.inOrder(); // Left -> Root -> Right

bst.postOrder(); // Left -> Right -> Root

// Search for two elements

search(bst, 12); // Should be found

search(bst, 9); // Should not be found

}

// Helper method for searching and printing result

private static void search(BSTArray bst, int key) {

if (bst.search(key)) {

System.out.println(key + " found");

} else {

System.out.println(key + " not found");

}

}

}

A computer screen with white text

AI-generated content may be incorrect.

73. Binary Search tree using Linked Lists

class Node

{

int key;

Node left;

Node right;

public Node(int key)

{

this.key = key;

}

}

class BST

{

private Node root;

public void insert(int key)

{

root = insert(root, key);

}

private Node insert(Node node, int key)

{

if (node == null)

{

return new Node(key);

}

if (key <= node.key)

{

node.left = insert(node.left, key);

}

else

{

node.right = insert(node.right, key);

}

return node;

}

public Node search(int key)

{

return search(root, key);

}

private Node search(Node node, int key)

{

if (node == null || node.key == key)

{

return node;

}

if (key <= node.key)

{

return search(node.left, key);

}

return search(node.right, key);

}

public void inOrder()

{

System.out.print("The inOrder traversal is: ");

inOrder(root);

System.out.println();

}

private void inOrder(Node node)

{

if (node == null)

{

return;

}

inOrder(node.left);

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

inOrder(node.right);

}

public void preOrder()

{

System.out.print("The preOrder traversal is: ");

preOrder(root);

System.out.println();

}

private void preOrder(Node node)

{

if (node == null)

{

return;

}

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

preOrder(node.left);

preOrder(node.right);

}

public void postOrder()

{

System.out.print("The postOrder traversal is: ");

postOrder(root);

System.out.println();

}

private void postOrder(Node node)

{

if (node == null)

{

return;

}

postOrder(node.left);

postOrder(node.right);

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

}

}

public class BinarySearchTree

{

public static void main(String[] args)

{

BST bst = new BST();

bst.insert(10);

bst.insert(15);

bst.insert(5);

bst.insert(8);

bst.insert(18);

bst.insert(12);

bst.insert(10);

bst.preOrder();

bst.inOrder();

bst.postOrder();

search(bst, 12);

search(bst, 9);

}

private static void search(BST bst, int key) {

if (bst.search(key) != null) {

System.out.println(key + " found");

}

else

{

System.out.println(key + " not found");

}

}

}

A screen shot of a computer

AI-generated content may be incorrect.

74. **Implement a program to insert elements into a binary search tree.**

// Import necessary classes

public class BSTInsertion {

// Node class to represent each node in the BST

static class Node {

int data; // Value of the node

Node left; // Reference to the left child

Node right; // Reference to the right child

// Constructor to create a new node

Node(int data) {

this.data = data;

this.left = null;

this.right = null;

}

}

// Method to insert a new node into the BST

public static Node insert(Node root, int data) {

// If tree is empty, create a new node and return it

if (root == null) {

return new Node(data);

}

// If data is less than root's data, insert in the left subtree

if (data < root.data) {

root.left = insert(root.left, data);

}

// If data is greater than root's data, insert in the right subtree

else if (data > root.data) {

root.right = insert(root.right, data);

}

// Return the unchanged root node

return root;

}

// Method for inorder traversal of the BST

public static void inorderTraversal(Node root) {

if (root == null) {

return; // Base case: if node is null, return

}

inorderTraversal(root.left); // Visit left subtree

System.out.print(root.data + " "); // Print current node's data

inorderTraversal(root.right); // Visit right subtree

}

// Main method to execute the program

public static void main(String[] args) {

Node root = null; // Initially the BST is empty

// Insert elements into the BST

root = insert(root, 50);

insert(root, 30);

insert(root, 20);

insert(root, 40);

insert(root, 70);

insert(root, 60);

insert(root, 80);

// Display inorder traversal (sorted order)

System.out.print("Inorder traversal: ");

inorderTraversal(root);

System.out.println();

}

}

A screenshot of a computer

AI-generated content may be incorrect.

75. **) to implement a program to search for an element in a binary search tree**

// Main class

public class BSTSearch {

// Node class definition to represent each node of the BST

static class Node {

int data; // The data value stored in the node

Node left; // Pointer to the left child

Node right; // Pointer to the right child

// Constructor to initialize a new node with data

Node(int data) {

this.data = data;

this.left = null;

this.right = null;

}

}

// Method to insert a new element into the BST

public static Node insertNode(Node root, int data) {

// If the current position is null, we place the new node here

if (root == null) {

return new Node(data); // Create and return a new node

}

// If the new data is smaller, insert in the left subtree

if (data < root.data) {

root.left = insertNode(root.left, data);

}

// If the new data is larger, insert in the right subtree

else if (data > root.data) {

root.right = insertNode(root.right, data);

}

// Return the current root node after insertion

return root;

}

// Method to search for a key in the BST

public static Node searchNode(Node root, int key) {

// Base condition: if root is null or key matches the current node's data

if (root == null || root.data == key) {

return root; // Key found or not present in the tree

}

// If key is smaller than current node's data, search in the left subtree

if (key < root.data) {

return searchNode(root.left, key);

}

// If key is greater than current node's data, search in the right subtree

return searchNode(root.right, key);

}

// Method for inorder traversal (Left -> Root -> Right)

public static void inorderTraversal(Node root) {

if (root == null) return;

inorderTraversal(root.left); // Visit left subtree

System.out.print(root.data + " "); // Print current node

inorderTraversal(root.right); // Visit right subtree

}

// Main method: Entry point of the program

public static void main(String[] args) {

Node root = null; // Initialize the BST as empty

// Insert elements into the BST

root = insertNode(root, 50);

insertNode(root, 30);

insertNode(root, 20);

insertNode(root, 40);

insertNode(root, 70);

insertNode(root, 60);

insertNode(root, 80);

// Print the BST using inorder traversal (should be sorted)

System.out.print("Inorder traversal: ");

inorderTraversal(root);

System.out.println();

// Define the key to be searched

int key = 40;

// Perform search operation in the BST

Node result = searchNode(root, key);

// Check if the element is found or not

if (result != null) {

System.out.println("Element " + key + " found in the BST.");

} else {

System.out.println("Element " + key + " not found in the BST.");

}

}

}

A screenshot of a computer

AI-generated content may be incorrect.

76. **Implement a program to delete a node from a binary search tree.**

// Define the main class

public class BSTDeletion {

// Define the structure of a BST node

static class Node {

int data; // Data value of the node

Node left; // Pointer to left child

Node right; // Pointer to right child

// Constructor to initialize a node

Node(int data) {

this.data = data;

this.left = null;

this.right = null;

}

}

// Function to perform inorder traversal (Left → Root → Right)

public static void inorderTraversal(Node root) {

if (root == null) return; // Base case: empty node

inorderTraversal(root.left); // Visit left subtree

System.out.print(root.data + " "); // Visit root

inorderTraversal(root.right); // Visit right subtree

}

// Function to perform preorder traversal (Root → Left → Right)

public static void preorderTraversal(Node root) {

if (root == null) return;

System.out.print(root.data + " "); // Visit root

preorderTraversal(root.left); // Visit left

preorderTraversal(root.right); // Visit right

}

// Function to perform postorder traversal (Left → Right → Root)

public static void postorderTraversal(Node root) {

if (root == null) return;

postorderTraversal(root.left); // Visit left

postorderTraversal(root.right); // Visit right

System.out.print(root.data + " "); // Visit root

}

// Utility function to find the minimum value node in the right subtree

public static Node findMin(Node root) {

while (root.left != null) {

root = root.left; // Go as left as possible

}

return root;

}

// Function to delete a node from the BST

public static Node deleteNode(Node root, int key) {

if (root == null) return null; // Base case: key not found

if (key < root.data) {

// If key is smaller than root, go left

root.left = deleteNode(root.left, key);

} else if (key > root.data) {

// If key is larger than root, go right

root.right = deleteNode(root.right, key);

} else {

// Node to delete found

// Case 1: Node with only right child or no child

if (root.left == null) {

return root.right; // Replace with right subtree

}

// Case 2: Node with only left child

else if (root.right == null) {

return root.left; // Replace with left subtree

}

// Case 3: Node with two children

// Find inorder successor (smallest in the right subtree)

Node temp = findMin(root.right);

// Copy the successor's value to the root

root.data = temp.data;

// Delete the inorder successor recursively

root.right = deleteNode(root.right, temp.data);

}

// Return the updated root reference

return root;

}

// Main method

public static void main(String[] args) {

// Manually constructing the BST as per the original C code

Node root = new Node(50);

root.left = new Node(30);

root.right = new Node(70);

root.left.left = new Node(20);

root.left.right = new Node(40);

root.right.left = new Node(60);

root.right.right = new Node(80);

// Print initial traversals

System.out.print("Inorder traversal: ");

inorderTraversal(root);

System.out.println();

System.out.print("Preorder traversal: ");

preorderTraversal(root);

System.out.println();

System.out.print("Postorder traversal: ");

postorderTraversal(root);

System.out.println();

// Delete node with value 50 (root)

root = deleteNode(root, 50);

// Print inorder after deletion

System.out.print("Inorder traversal after deletion: ");

inorderTraversal(root);

System.out.println();

}

}

A screenshot of a computer program

AI-generated content may be incorrect.

LAB\_11 (25-4-25)

77. AVL insertion

// Class representing a node in the AVL Tree

class Node {

int key; // Value of the node

Node left, right; // References to left and right child nodes

int height; // Height of the node in the tree

Node(int key) {

this.key = key;

this.height = 1; // New node starts with height 1 (leaf node)

}

}

public class AVLTree {

// Function to get the height of a node

int getHeight(Node node) {

if (node == null) return 0; // Null node has height 0

return node.height;

}

// Function to get the balance factor of a node

int getBalanceFactor(Node node) {

if (node == null) return 0;

return getHeight(node.left) - getHeight(node.right);

// Balance Factor = height of left subtree - height of right subtree

}

* // Utility function to get the maximum of two integers(used in height calculation).(which side is greater left sub tree or right sub tree)

int max(int a, int b) {

* return (a > b) ? a : b; }

// Right rotation (used for LL and LR imbalance)

Node rightRotate(Node y) {

Node x = y.left; // x is left child of y

Node T2 = x.right; // T2 is the right child of x (may be null)

// Perform rotation

x.right = y; // Make y the right child of x

y.left = T2; // T2 becomes the left child of y

// Update heights

y.height = max(getHeight(y.left), getHeight(y.right)) + 1;

x.height = max(getHeight(x.left), getHeight(x.right)) + 1;

// Return new root

return x;

}

// Left rotation (used for RR and RL imbalance)

Node leftRotate(Node x) {

Node y = x.right; // y is right child of x

Node T2 = y.left; // T2 is the left child of y

// Perform rotation

y.left = x; // Make x the left child of y

x.right = T2; // T2 becomes the right child of x

// Update heights

x.height = max(getHeight(x.left), getHeight(x.right)) + 1;

y.height = max(getHeight(y.left), getHeight(y.right)) + 1;

// Return new root

return y;

}

// Function to insert a key and return new root of AVL tree

Node insert(Node node, int key) {

// 1. Standard BST Insertion

if (node == null)

return new Node(key); // If node is null, insert here

if (key < node.key)

node.left = insert(node.left, key); // Insert into left subtree

else if (key > node.key)

node.right = insert(node.right, key); // Insert into right subtree

else

return node; // Duplicates not allowed in BST

// 2. Update height of the ancestor node

node.height = 1 + max(getHeight(node.left), getHeight(node.right));

// 3. Get the balance factor to check for imbalance

int balance = getBalanceFactor(node);

// 4. Balance the tree with 4 possible cases

// Case 1: Left Left (LL)

if (balance > 1 && key < node.left.key)

return rightRotate(node);

// Case 2: Right Right (RR)

if (balance < -1 && key > node.right.key)

return leftRotate(node);

// Case 3: Left Right (LR)

if (balance > 1 && key > node.left.key) {

node.left = leftRotate(node.left); // First left rotate child

return rightRotate(node); // Then right rotate current node

}

// Case 4: Right Left (RL)

if (balance < -1 && key < node.right.key) {

node.right = rightRotate(node.right); // First right rotate child

return leftRotate(node); // Then left rotate current node

}

return node; // Return unchanged node pointer

}

// Function to print inorder traversal of AVL tree

void inOrder(Node root) {

if (root != null) {

inOrder(root.left); // Left subtree

System.out.print(root.key + " "); // Current node

inOrder(root.right); // Right subtree

}

}

// Main method

public static void main(String[] args) {

AVLTree tree = new AVLTree();

Node root = null;

// Insert nodes into AVL Tree

root = tree.insert(root, 1);

root = tree.insert(root, 2);

root = tree.insert(root, 4);

root = tree.insert(root, 5);

root = tree.insert(root, 6); // RR Rotation happens here

root = tree.insert(root, 3); // RL Rotation happens here

// Inorder Traversal of AVL Tree

System.out.print("Inorder traversal of AVL tree: ");

tree.inOrder(root); // Output will be sorted: 1 2 3 4 5 6

}

}

A screen shot of a computer

AI-generated content may be incorrect.

78. AVL DELETE

// Node class to represent each node in the AVL tree

class Node {

int key; // The value stored in the node

Node left, right; // References to left and right child nodes

int height; // Height of this node in the AVL tree

// Constructor to create a new node with a given key

Node(int k) {

key = k; // Initialize the key with given value

left = right = null; // Initially, no children

height = 1; // Height of a new node is 1 (leaf node)

}

}

// Main class containing AVL tree methods

public class Main {

// Utility method to get height of a node (returns 0 if node is null)

static int height(Node N) {

if (N == null) // If node is null

return 0; // Height is 0

return N.height; // Otherwise return node's height

}

// Right rotate subtree rooted with y

static Node rightRotate(Node y) {

Node x = y.left; // x is left child of y (new root after rotation)

Node T2 = x.right; // Temporarily store x's right subtree

// Perform rotation

x.right = y; // Make y the right child of x

y.left = T2; // Attach T2 as left child of y

// Update heights of rotated nodes

y.height = Math.max(height(y.left), height(y.right)) + 1; // y height updated first

x.height = Math.max(height(x.left), height(x.right)) + 1; // then x height updated

return x; // Return new root node after rotation

}

// Left rotate subtree rooted with x

static Node leftRotate(Node x) {

Node y = x.right; // y is right child of x (new root after rotation)

Node T2 = y.left; // Temporarily store y's left subtree

// Perform rotation

y.left = x; // Make x the left child of y

x.right = T2; // Attach T2 as right child of x

// Update heights of rotated nodes

x.height = Math.max(height(x.left), height(x.right)) + 1; // update x height

y.height = Math.max(height(y.left), height(y.right)) + 1; // update y height

return y; // Return new root node after rotation

}

// Get balance factor of node N (height of left subtree - right subtree)

static int getBalance(Node N) {

if (N == null) // If node is null

return 0; // balance is 0

return height(N.left) - height(N.right); // difference of heights

}

// Recursive method to insert a key into the subtree rooted with node and

// return the new root of the subtree after balancing

static Node insert(Node node, int key) {

// 1. Normal BST insertion

if (node == null) // If current node is null

return new Node(key); // Create a new node with key

// If key is less than node's key, insert in left subtree

if (key < node.key)

node.left = insert(node.left, key);

// If key is greater than node's key, insert in right subtree

else if (key > node.key)

node.right = insert(node.right, key);

else // Duplicate keys not allowed

return node;

// 2. Update height of this ancestor node

node.height = Math.max(height(node.left), height(node.right)) + 1;

// 3. Get the balance factor of this node to check if unbalanced

int balance = getBalance(node);

// 4. If unbalanced, then check 4 cases

// Left Left Case: imbalance caused by inserting in left subtree of left child

if (balance > 1 && key < node.left.key)

return rightRotate(node); // Perform right rotation

// Right Right Case: imbalance caused by inserting in right subtree of right child

if (balance < -1 && key > node.right.key)

return leftRotate(node); // Perform left rotation

// Left Right Case: imbalance caused by inserting in right subtree of left child

if (balance > 1 && key > node.left.key) {

node.left = leftRotate(node.left); // First left rotate left child

return rightRotate(node); // Then right rotate node

}

// Right Left Case: imbalance caused by inserting in left subtree of right child

if (balance < -1 && key < node.right.key) {

node.right = rightRotate(node.right); // First right rotate right child

return leftRotate(node); // Then left rotate node

}

// Return the unchanged node pointer

return node;

}

// Utility function to find node with minimum key value in subtree rooted with node

static Node minValueNode(Node node) {

Node current = node;

// Loop to find the leftmost leaf

while (current.left != null)

current = current.left;

return current; // Return node with minimum key

}

// Recursive method to delete a node with given key from subtree with given root

// Returns new root of the subtree after deletion and balancing

static Node deleteNode(Node root, int key) {

// STEP 1: Perform standard BST delete

if (root == null) // If tree is empty

return root; // Return null

// If key to be deleted is smaller than root's key, go to left subtree

if (key < root.key)

root.left = deleteNode(root.left, key);

// If key to be deleted is greater than root's key, go to right subtree

else if (key > root.key)

root.right = deleteNode(root.right, key);

else { // Found node to be deleted

// Node with only one child or no child

if ((root.left == null) || (root.right == null)) {

Node temp = null;

// Assign temp to non-null child if any

if (root.left != null)

temp = root.left;

else

temp = root.right;

// No child case

if (temp == null) {

temp = root; // Temporarily store root node

root = null; // Delete root (make it null)

} else // One child case

root = temp; // Copy child to root

} else {

// Node with two children:

// Get inorder successor (smallest in right subtree)

Node temp = minValueNode(root.right);

// Copy inorder successor's key to root

root.key = temp.key;

// Delete inorder successor recursively

root.right = deleteNode(root.right, temp.key);

}

}

// If tree had only one node and now root is null, return

if (root == null)

return root;

// STEP 2: Update height of current node

root.height = Math.max(height(root.left), height(root.right)) + 1;

// STEP 3: Get balance factor of current node

int balance = getBalance(root);

// STEP 4: If node unbalanced, then balance it with rotations

// Left Left Case

if (balance > 1 && getBalance(root.left) >= 0)

return rightRotate(root);

// Left Right Case

if (balance > 1 && getBalance(root.left) < 0) {

root.left = leftRotate(root.left);

return rightRotate(root);

}

// Right Right Case

if (balance < -1 && getBalance(root.right) <= 0)

return leftRotate(root);

// Right Left Case

if (balance < -1 && getBalance(root.right) > 0) {

root.right = rightRotate(root.right);

return leftRotate(root);

}

// Return the balanced node pointer

return root;

}

// Utility function for preorder traversal of the tree

static void preOrder(Node root) {

if (root != null) {

System.out.print(root.key + " "); // Print root key

preOrder(root.left); // Traverse left subtree

preOrder(root.right); // Traverse right subtree

}

}

// Main method to test the AVL tree implementation

public static void main(String[] args) {

Node root = null; // Start with empty tree

// Insert nodes into AVL tree

root = insert(root, 9);

root = insert(root, 5);

root = insert(root, 10);

root = insert(root, 0);

root = insert(root, 6);

root = insert(root, 11);

root = insert(root, -1);

root = insert(root, 1);

root = insert(root, 2);

// Print preorder traversal of the constructed AVL tree

System.out.println("Preorder traversal of the constructed AVL tree is:");

preOrder(root);

// Delete node with key 10

root = deleteNode(root, 10);

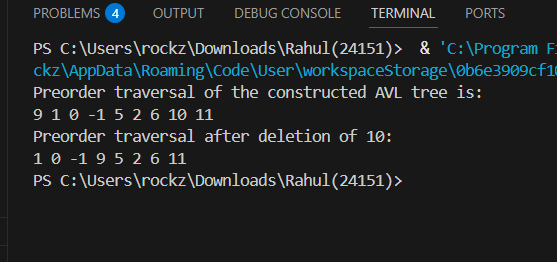
// Print preorder traversal after deletion

System.out.println("\nPreorder traversal after deletion of 10:");

preOrder(root);

}

}



79. AVL using Arrays

public class AVLArray {

int MAX = 100; // max nodes

int[] keys = new int[MAX]; // keys of nodes

int[] heights = new int[MAX]; // heights of nodes

int[] left = new int[MAX]; // left child indices

int[] right = new int[MAX]; // right child indices

boolean[] used = new boolean[MAX]; // to check used slots

int root = -1; // root index, -1 means empty tree

// Constructor initializes arrays

public AVLArray() {

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

left[i] = -1; // no left child initially

right[i] = -1; // no right child initially

heights[i] = 0; // height zero for unused nodes

used[i] = false;

}

}

// Allocate new node index with given key

int newNode(int key) {

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

if (!used[i]) {

used[i] = true;

keys[i] = key;

heights[i] = 1; // new node height = 1

left[i] = -1;

right[i] = -1;

return i;

}

}

throw new RuntimeException("Out of space!");

}

// Get height of node at index i

int height(int i) {

if (i == -1) return 0;

return heights[i];

}

// Update height of node i

void updateHeight(int i) {

heights[i] = Math.max(height(left[i]), height(right[i])) + 1;

}

// Get balance factor of node i

int getBalance(int i) {

if (i == -1) return 0;

return height(left[i]) - height(right[i]);

}

// Right rotate subtree rooted at y

int rightRotate(int y) {

int x = left[y];

int T2 = right[x];

// Perform rotation

right[x] = y;

left[y] = T2;

// Update heights

updateHeight(y);

updateHeight(x);

// Return new root

return x;

}

// Left rotate subtree rooted at x

int leftRotate(int x) {

int y = right[x];

int T2 = left[y];

// Perform rotation

left[y] = x;

right[x] = T2;

// Update heights

updateHeight(x);

updateHeight(y);

// Return new root

return y;

}

// Insert key into subtree rooted at nodeIndex, returns new root index of subtree

int insert(int nodeIndex, int key) {

if (nodeIndex == -1) {

return newNode(key);

}

if (key < keys[nodeIndex]) {

left[nodeIndex] = insert(left[nodeIndex], key);

} else if (key > keys[nodeIndex]) {

right[nodeIndex] = insert(right[nodeIndex], key);

} else {

// Duplicate keys not allowed

return nodeIndex;

}

// Update height of this ancestor node

updateHeight(nodeIndex);

// Get balance factor

int balance = getBalance(nodeIndex);

// If node is unbalanced, fix it with rotations

// Left Left Case

if (balance > 1 && key < keys[left[nodeIndex]])

return rightRotate(nodeIndex);

// Right Right Case

if (balance < -1 && key > keys[right[nodeIndex]])

return leftRotate(nodeIndex);

// Left Right Case

if (balance > 1 && key > keys[left[nodeIndex]]) {

left[nodeIndex] = leftRotate(left[nodeIndex]);

return rightRotate(nodeIndex);

}

// Right Left Case

if (balance < -1 && key < keys[right[nodeIndex]]) {

right[nodeIndex] = rightRotate(right[nodeIndex]);

return leftRotate(nodeIndex);

}

return nodeIndex;

}

// Preorder traversal of tree starting at index i

void preOrder(int i) {

if (i != -1) {

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

preOrder(left[i]);

preOrder(right[i]);

}

}

public static void main(String[] args) {

AVLArray tree = new AVLArray();

// Insert keys

tree.root = tree.insert(tree.root, 9);

tree.root = tree.insert(tree.root, 5);

tree.root = tree.insert(tree.root, 10);

tree.root = tree.insert(tree.root, 0);

tree.root = tree.insert(tree.root, 6);

tree.root = tree.insert(tree.root, 11);

tree.root = tree.insert(tree.root, -1);

tree.root = tree.insert(tree.root, 1);

tree.root = tree.insert(tree.root, 2);

System.out.println("Preorder traversal of constructed AVL tree:");

tree.preOrder(tree.root);

}

}

A screen shot of a computer

AI-generated content may be incorrect.