**Day 6 : Special Programs Series:**

**Searching**

1. Linear Search [Solution]
2. **package** Assig6;
3. **public** **class** Q1 {
4. **public** **static** **int** linearSearch(**int**[] arr, **int** target) {
5. **for** (**int** i = 0; i < arr.length; i++) {
6. **if** (arr[i] == target) {
7. **return** i;
8. }
9. }
10. **return** -1;
11. }
12. **public** **static** **void** main(String[] args) {
13. **int**[] arr = { 5, 10, 15, 20, 25, 30, 35, 40, 45, 50 };
14. **int** target = 30;
15. **int** index = *linearSearch*(arr, target);
16. **if** (index != -1) {
17. System.***out***.println("Element found at index " + index);
18. } **else** {
19. System.***out***.println("Element not found in the array");
20. }
21. }
22. }

OUTPUT:

Element found at index 5

2. Binary Search [Solution]

**package** Assig6;

**public** **class** Q2 {

**public** **static** **int** binarySearch(**int**[] arr, **int** target) {

**int** left = 0;

**int** right = arr.length - 1;

**while** (left <= right) {

**int** mid = left + (right - left) / 2;

**if** (arr[mid] == target)

**return** mid;

**if** (arr[mid] < target)

left = mid + 1;

**else**

right = mid - 1;

}

**return** -1;

}

**public** **static** **void** main(String[] args) {

**int**[] arr = { 2, 4, 6, 8, 10, 12, 14, 16, 18, 20 };

**int** target = 12;

**int** index = *binarySearch*(arr, target);

**if** (index != -1) {

System.***out***.println("Element found at index " + index);

} **else** {

System.***out***.println("Element not found in the array");

}

}

}

OUTPUT:

Element found at index 6

3. Sort elements by frequency

4. Sort an array of 0s, 1s and 2s

5. Java Program to Check for balanced parenthesis by using Stacks

**package** Assig6;

**import** java.util.\*;

**public** **class** Q7 {

**public** **static** **boolean** isBalanced(String s) {

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

**for** (**char** c : s.toCharArray()) {

**if** (c == '(' || c == '[' || c == '{') {

stack.push(c);

}

**else** **if** (c == ')' || c == ']' || c == '}') {

**if** (stack.isEmpty() || !*isMatching*(stack.pop(), c)) {

**return** **false**;

}

}

}

**return** stack.isEmpty();

}

**private** **static** **boolean** isMatching(**char** opening, **char** closing) {

**return** (opening == '(' && closing == ')') || (opening == '[' && closing == ']') || (opening == '{' && closing == '}');

}

**public** **static** **void** main(String[] args) {

String[] testCases = {"{[()]}", "{[()]", "([)]", "{[(])}", "{[()]}"};

**for** (String testCase : testCases) {

System.***out***.println("Expression: " + testCase + ", Balanced: " + *isBalanced*(testCase));

}

}

}

OUTPUT:

Expression: {[()]}, Balanced: true

Expression: {[()], Balanced: false

Expression: ([)], Balanced: false

Expression: {[(])}, Balanced: false

Expression: {[()]}, Balanced: true

6. Java Program to Implement Stack

**package** Assig6;

**class** Q3

{

**static** **final** **int** ***MAX*** =5;

**int** top;

**int** stack[] = **new** **int**[***MAX***];

Q3()

{

top = -1;

}

**boolean** isEmpty()

{

**return** (top < 0);//true

}

**boolean** push(**int** x)

{

**if**(top >= (***MAX*** -1))

{

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

**return** **false**;

}

**else**

{

stack[++top] = x;

System.***out***.println(x+ " Push ...");

**return** **true**;

}

}

**int** pop()

{

**if**(top < 0)

{

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

**return** 0;

}

**else**{

**int** x = stack[top--];

**return** x;

}

}

**int** peek()

{

**if**(top<0)

{

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

**return** 0;

}

**else**{

**int** x = stack[top];

**return** x;

}

}

**public** **static** **void** main(String args[])

{

Q3 s1 = **new** Q3();

System.***out***.println(s1.isEmpty());

s1.push(20);

s1.push(30);

s1.push(40);

s1.push(50);

System.***out***.println("Delete element = "+s1.pop());

System.***out***.println("Tos element = "+s1.peek());

}

}

OUTPUT:

true

20 Push ...

30 Push ...

40 Push ...

50 Push ...

Delete element = 50

Tos element = 40

7. Java Program to Implement Queue

**package** Assig6;

**class** Q4{

**int** size = 5;

**int** Q[] = **new** **int**[size];

**int** rear, front;

Q4()

{

front=-1;

rear=-1;

}

**boolean** isEmpty()

{

**if**(front == -1)

**return** **true**;

**else**

**return** **false**;

}

**boolean** isFull()

{

**if**(front == 0 && rear == size-1)

**return** **true**;

**else**

**return** **false**;

}

**void** enqueue(**int** x)

{

**if**(isFull())

{

System.***out***.println("Queue is full");

}

**else**

{

**if**(front == -1)

front =0;

rear++;

Q[rear] = x;

System.***out***.println(x+" Inserted.");

}

}

**int** dequeue()

{

**int** x;

**if**(isEmpty())

{

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

**return** -1;

}

**else**

{

x=Q[front];

**if**(front >= rear )

{

front = -1;

rear = -1;

}

**else**{

front++;

}

System.***out***.println(x+"Deleted.");

**return** x;

}

}

**void** display()

{

**if**(isEmpty())

System.***out***.println("Queue is Empty");

**else**

{

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

System.***out***.println(Q[i]);

}

}

**public** **static** **void** main(String args[])

{

Q4 q1 = **new** Q4();

q1.enqueue(11);

q1.enqueue(12);

q1.enqueue(13);

q1.enqueue(14);

q1.enqueue(15);

q1.enqueue(15);

q1.enqueue(16);

q1.display();

q1.dequeue();

q1.display();

q1.enqueue(16);

q1.display();

}

}

OUTPUT:

11 Inserted.

12 Inserted.

13 Inserted.

14 Inserted.

15 Inserted.

Queue is full

Queue is full

11

12

13

14

15

11Deleted.

12

13

14

15

8. Java Program to Implement Dequeue.

**package** Assig6;

//Deque implementation in Java

**class** Q5 {

**static** **final** **int** ***MAX*** = 100;

**int** arr[];

**int** front;

**int** rear;

**int** size;// Take array size = n

**public** Q5(**int** size) {

arr = **new** **int**[***MAX***];

//set pointers front and rear

front = -1;

rear = 0;

**this**.size = size;

}

**boolean** isFull() {

**return** ((front == 0 && rear == size - 1) || front == rear + 1);

}

**boolean** isEmpty() {

**return** (front == -1);

}

//Insert at the front

**void** insertfront(**int** key) {

//check for full queue

**if** (isFull()) {

System.***out***.println("Full");

**return**;

}

//check front position

**if** (front == -1) {

front = 0;

rear = 0;

}

//front < 1, reinitialize it to n-1(last index)

**else** **if** (front == 0)

front = size - 1;

//else decrese by 1

**else**

front = front - 1;

//insert new value at front i.e., arr[front]

arr[front] = key;

}

**void** insertrear(**int** key) {

// check array is full

**if** (isFull()) {

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

**return**;

}

**if** (front == -1) {

front = 0;

rear = 0;

}

//check deque is full rear = 0

**else** **if** (rear == size - 1)

rear = 0;

//increase the pointer by 1

**else**

rear = rear + 1;

//insert element arr[rear]

arr[rear] = key;

}

**void** deletefront() {

**if** (isEmpty()) {

System.***out***.println("Queue Underflow\n");

**return**;

}

// Deque has only one element

**if** (front == rear) {

front = -1;

rear = -1;

} **else** **if** (front == size - 1)

front = 0;

**else**

front = front + 1;

}

**void** deleterear() {

**if** (isEmpty()) {

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

**return**;

}

**if** (front == rear) {

front = -1;

rear = -1;

} **else** **if** (rear == 0)

rear = size - 1;

**else**

rear = rear - 1;

}

**int** getFront() {

**if** (isEmpty()) {

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

**return** -1;

}

**return** arr[front];

}

**int** getRear() {

**if** (isEmpty() || rear < 0) {

System.***out***.println(" Underflow\n");

**return** -1;

}

**return** arr[rear];

}

**public** **static** **void** main(String[] args) {

Q5 dq = **new** Q5(4);

System.***out***.println("Insert element at rear end : 12 ");

dq.insertrear(12);

System.***out***.println("insert element at rear end : 14 ");

dq.insertrear(14);

System.***out***.println("get rear element : " + dq.getRear());

dq.deleterear();

System.***out***.println("After delete rear element new rear become : " + dq.getRear());

System.***out***.println("inserting element at front end");

dq.insertfront(13);

System.***out***.println("get front element: " + dq.getFront());

dq.deletefront();

System.***out***.println("After delete front element new front become : " + +dq.getFront());

}

}

OUTPUT:

Insert element at rear end : 12

insert element at rear end : 14

get rear element : 14

After delete rear element new rear become : 12

inserting element at front end

get front element: 13

After delete front element new front become : 12

9. Java Program to Implement Stack Using Two Queues

**package** Assig6;

**import** java.util.LinkedList;

**import** java.util.Queue;

**public** **class** Q9<T> {

**private** Queue<T> queue1;

**private** Queue<T> queue2;

**public** Q9() {

queue1 = **new** LinkedList<>();

queue2 = **new** LinkedList<>();

}

**public** **void** push(T item) {

queue2.add(item);

**while** (!queue1.isEmpty()) {

queue2.add(queue1.remove());

}

Queue<T> temp = queue1;

queue1 = queue2;

queue2 = temp;

System.***out***.println("Pushed element: " + item);

}

**public** T pop() {

**if** (isEmpty()) {

System.***out***.println("Stack Underflow: Unable to pop element. Stack is empty.");

**return** **null**;

}

System.***out***.println("Popped element: " + queue1.peek());

**return** queue1.remove();

}

**public** **boolean** isEmpty() {

**return** queue1.isEmpty();

}

**public** **static** **void** main(String[] args) {

Q9<Integer> stack = **new** Q9<>();

stack.push(10);

stack.push(20);

stack.push(30);

stack.pop();

stack.pop();

stack.pop();

stack.pop();

}

}

OUTPUT:

Pushed element: 10

Pushed element: 20

Pushed element: 30

Popped element: 30

Popped element: 20

Popped element: 10

Stack Underflow: Unable to pop element. Stack is empty.

10. Java Program to Implement Queue Using Two Stacks

**package** Assig6;

**import** java.util.Stack;

**public** **class** Q6<T> {

**private** Stack<T> stack1; // Stack for enqueue operation

**private** Stack<T> stack2; // Stack for dequeue operation

// Constructor to initialize the two stacks

**public** Q6() {

stack1 = **new** Stack<>();

stack2 = **new** Stack<>();

}

// Method to enqueue an element into the queue

**public** **void** enqueue(T item) {

stack1.push(item); // Push the element onto stack1

System.***out***.println("Enqueued element: " + item);

}

// Method to dequeue an element from the queue

**public** T dequeue() {

**if** (isEmpty()) {

System.***out***.println("Queue is empty. Unable to dequeue.");

**return** **null**;

}

// If stack2 is empty, transfer elements from stack1 to stack2

**if** (stack2.isEmpty()) {

**while** (!stack1.isEmpty()) {

stack2.push(stack1.pop());

}

}

// Pop the top element from stack2 (which is the front of the queue)

T dequeuedItem = stack2.pop();

System.***out***.println("Dequeued element: " + dequeuedItem);

**return** dequeuedItem;

}

// Method to check if the queue is empty

**public** **boolean** isEmpty() {

**return** stack1.isEmpty() && stack2.isEmpty();

}

**public** **static** **void** main(String[] args) {

Q6<Integer> queue = **new** Q6<>();

queue.enqueue(10);

queue.enqueue(20);

queue.enqueue(30);

queue.dequeue();

queue.dequeue();

queue.dequeue();

queue.dequeue(); // Trying to dequeue from an empty queue

}

}

OUTPUT:

Enqueued element: 10

Enqueued element: 20

Enqueued element: 30

Dequeued element: 10

Dequeued element: 20

Dequeued element: 30

Queue is empty. Unable to dequeue.