**Question 1:**

**Implement a queue class, as a circular array. The queue will have only positive integer**

**elements. Your class should be called myQueue , and it should support the following**

**member methods:**

**Queue() : A constructor, that creates an object of type Queue**

**void Enqueue (int elem) : adds elem at the end of the queue**

**void Dequeue() : removes the element from the front of the queue**

**int Front () : returns (without removing) the element from the front of the queue**

**int getSize () : returns the number of elements in the queue**

**bool isEmpty() : returns true if the queue is empty, else false**

**bool isFull() : returns true if the queue is full, else false**

**void Display() : displays all the elements of the queue**

**You will be given N queries as follows. A query may contain two integers 'a x' or just a**

**single integer 'a'**

**1 x - Enqueue element x into the queue.**

**2 - Dequeue the front element of the queue.**

**3 - front operation which prints the front element of the queue.**

**4 - Displays the queue.**

**5 - Checks if queue is empty.**

**6 - displays size of queue.**

**7 - Checks if queue is full, if true then exit**

**ANSWER:**

import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStreamReader;

public class myQueue

{

int end;

int front;

int[] cBuff;

static final int Q\_MAX = 31;

public myQueue() {

this.end = 0;

this.front = 0;

this.cBuff = new int[Q\_MAX];

}

public static void myExit(String code) {

System.out.println(code);

System.exit(1);

}

public void Enqueue(int elem) {

if (this.isFull())

myExit("-1");

this.cBuff[this.front] = elem;

this.front = (this.front + 1) % Q\_MAX;

}

public int Dequeue() {

if (this.isEmpty())

myExit("-2");

int value = this.cBuff[this.end];

this.end = (this.end + 1) % Q\_MAX;

return value;

}

public int Front() {

if (this.isEmpty())

myExit("-3");

if (this.front > 0)

return this.cBuff[this.front - 1];

return this.cBuff[Q\_MAX - 1];

}

public int getSize() {

if (this.isEmpty())

return 0;

if (this.end < this.front)

return this.front - this.end;

return Q\_MAX - this.end + this.front;

}

public boolean isEmpty() {

return this.front == this.end;

}

public boolean isFull() {

return this.getSize() == Q\_MAX - 1;

}

public void Display() {

int i = 0;

int curr = this.end;

int qSize = this.getSize();

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

System.out.print(this.cBuff[curr] + " ");

curr = (curr + 1) % Q\_MAX;

}

System.out.println();

}

public static void main(String[] args) {

myQueue queue = new myQueue();

BufferedReader reader = new BufferedReader(new InputStreamReader(System.in));

String input = "";

try {

input = reader.readLine();

} catch (IOException e) {

myExit("IO Exception");

}

int queries = Integer.parseInt(input.trim());

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

try {

input = reader.readLine();

} catch (IOException e) {

myExit("IO Exception");

}

String[] inputArr = input.trim().split(" ");

int operation = Integer.parseInt(inputArr[0]);

switch (operation) {

case 1:

queue.Enqueue(Integer.parseInt(inputArr[1]));

break;

case 2:

System.out.println(queue.Dequeue());

break;

case 3:

System.out.println(queue.Front());

break;

case 4:

queue.Display();

break;

case 5:

if (queue.isEmpty())

System.out.println("1");

else

System.out.println("0");

break;

case 6:

System.out.println(queue.getSize());

break;

case 7:

if (queue.isFull())

System.exit(0);

System.out.println("0");

break;

}

}

}

}

**Question 2:**

**Given a string of characters including parentheses of 3 types: ( ), { } and [ ]. You have to**

**find whether the parentheses are balanced or not. Write a function called**

**checkBalanced which takes a string as an argument and if the string is balanced, returns**

**1, otherwise returns 0. Assume that the string will not contain any white space.**

**ANSWER:**

import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStreamReader;

public class CheckBalanced {

int top;

char[] stack;

static final int S\_MAX = 31;

public CheckBalanced() {

this.top = -1;

this.stack = new char[S\_MAX];

}

public static void myExit(String code) {

System.out.println(code);

System.exit(1);

}

public void Push(char elem) {

if (this.isFull())

myExit("-1");

this.top++;

this.stack[this.top] = elem;

}

public void Pop() {

if (this.isEmpty())

myExit("-2");

this.top--;

}

public int Peek() {

if (this.isEmpty())

return '\0';

return this.stack[top];

}

public boolean isEmpty() {

return this.top == -1;

}

public boolean isFull() {

return this.top == S\_MAX - 1;

}

public void Reset() {

this.top = -1;

}

public static void main(String[] args) {

CheckBalanced stack = new CheckBalanced();

BufferedReader reader = new BufferedReader(new InputStreamReader(System.in));

String input = "";

try {

input = reader.readLine();

} catch (IOException e) {

myExit("IO Exception");

}

int queries = Integer.parseInt(input.trim());

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

stack.Reset();

try {

input = reader.readLine();

input = input.trim();

} catch (IOException e) {

myExit("IO Exception");

}

for (int j = 0; j < input.length(); j++) {

char ch = input.charAt(j);

switch (ch) {

case '(':

case '{':

case '[':

stack.Push(ch);

break;

case ')':

if (stack.Peek() == '(')

stack.Pop();

else

stack.Push(ch);

break;

case '}':

if (stack.Peek() == '{')

stack.Pop();

else

stack.Push(ch);

break;

case ']':

if (stack.Peek() == '[')

stack.Pop();

else

stack.Push(ch);

break;

}

}

System.out.println(stack.isEmpty() ? 1 : 0);

}

}

}

**Question 3:**

**Implement a Linked list class, as a circular array. The queue will have only positive**

**integer elements. Your class should be called myList , and it should support the**

**following member methods:**

1. **insertEnd(val) [Return type void]**
2. **insertAtInd(ind,val) [Return type bool]**
3. **count() [Return type int]**
4. **print() [Return type void]**
5. **deleteAll(val) [Return type int]**
6. **reverse() [Return type void]**

**ANSWER :**

import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStreamReader;

class Node {

int data;

Node next;

public Node(int data, Node next) {

this.data = data;

this.next = next;

}

}

public class LinkedList {

Node head;

int length;

public LinkedList() {

this.head = null;

this.length = 0;

}

public static void myExit(String code) {

System.out.println(code);

System.exit(1);

}

public void insertEnd(int data) {

if (head == null) {

head = new Node(data, null);

length++;

return;

}

Node tmp = head;

while (tmp.next != null) {

tmp = tmp.next;

}

Node newNode = new Node(data, null);

tmp.next = newNode;

length++;

}

public boolean insertAtInd(int index, int data) {

if (isEmpty() || index < 0 || index > length) {

return false;

}

Node tmp = head;

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

tmp = tmp.next;

}

Node newNode = new Node(data, tmp.next);

tmp.next = newNode;

length++;

return true;

}

public int count() {

return length;

}

boolean isEmpty() {

return head == null;

}

public void print() {

if (isEmpty()) {

System.out.println("null");

return;

}

Node tmp = head;

while (tmp != null) {

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

tmp = tmp.next;

}

System.out.println();

}

void deleteNext(Node node) {

Node tmp = node.next;

node.next = tmp.next;

length--;

}

public int deleteAll(int data) {

int count = 0;

if (isEmpty())

return 0;

while (head.data == data) {

head = head.next;

count++;

length--;

if (isEmpty())

return count;

}

Node node = head;

while (node.next != null) {

if (node.next.data == data) {

deleteNext(node);

count++;

continue;

}

node = node.next;

}

return count;

}

Node reverseAll(Node node) {

Node prev = null;

Node curr = node;

Node next = null;

while (curr != null) {

next = curr.next;

curr.next = prev;

prev = curr;

curr = next;

}

node = prev;

return node;

}

public void reverse() {

if (isEmpty())

return;

head = reverseAll(head);

}

void printAtIndex(int index, int count) {

if (isEmpty()) {

System.out.println("NONE");

return;

}

Node tmp = new Node(head.data, head.next);

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

tmp = tmp.next;

}

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

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

tmp = tmp.next;

}

System.out.println();

}

public void midValue() {

if (length % 2 == 0)

printAtIndex(length / 2 - 1, 2);

else

printAtIndex(length / 2, 1);

}

public static void main(String[] args) {

LinkedList list = new LinkedList();

BufferedReader reader = new BufferedReader(new InputStreamReader(System.in));

String input = "";

while (true) {

try {

input = reader.readLine();

} catch (IOException e) {

myExit("IO Exception");

}

String[] inputArr = input.trim().split(" ");

char operation = inputArr[0].charAt(0);

switch (operation) {

case 'i':

list.insertEnd(Integer.parseInt(inputArr[1]));

break;

case 'n':

int status = list.insertAtInd(Integer.parseInt(inputArr[1]), Integer.parseInt(inputArr[2]))? 1 : 0;

System.out.println(status);

break;

case 'p':

list.print();

break;

case 'r':

list.reverse();

break;

case 'c':

System.out.println(list.count());

break;

case 'm':

list.midValue();

break;

case 'd':

System.out.println(list.deleteAll(Integer.parseInt(inputArr[1])));

break;

case 'q':

System.exit(0);

break;

}

}

}

}

**Question 4:**

**Given a binary tree, print its preorder, inorder and postorder traversals. Your input will**

**be list of numbers, which must be inserted in such a way that it forms a Binary Search**

**Tree. You have to implement a class with the three member functions:**

**1. preOrder()**

**2. inOrder()**

**3. postOrder()**

**Print the tree's traversals as a single line of space-separated values**

**ANSWER :**

import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStreamReader;

class Node {

int data;

Node left;

Node right;

public Node(int data) {

this.data = data;

this.left = null;

this.right = null;

}

}

public class Traversal {

Node root;

public Traversal() {

this.root = null;

}

public static void myExit(String code) {

System.out.println(code);

System.exit(1);

}

public void Insert(int data) {

this.root = this.InsertData(this.root, data);

}

Node InsertData(Node root, int data) {

if (root == null) {

Node tmp = new Node(data);

return tmp;

}

if (data < root.data) {

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

} else if (data > root.data) {

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

}

return root;

}

public void inOrder() {

inOrderTraverse(this.root);

}

void inOrderTraverse(Node root) {

if (root != null) {

inOrderTraverse(root.left);

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

inOrderTraverse(root.right);

}

}

public void preOrder() {

preOrderTraverse(this.root);

}

void preOrderTraverse(Node root) {

if (root != null) {

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

preOrderTraverse(root.left);

preOrderTraverse(root.right);

}

}

public void postOrder() {

postOrderTraverse(this.root);

}

void postOrderTraverse(Node root) {

if (root != null) {

postOrderTraverse(root.left);

postOrderTraverse(root.right);

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

}

}

public static void main(String[] args) {

Traversal tree = new Traversal();

BufferedReader reader = new BufferedReader(new InputStreamReader(System.in));

String input = "";

try {

input = reader.readLine();

} catch (IOException e) {

myExit("IO Exception");

}

int queries = Integer.parseInt(input.trim());

try {

input = reader.readLine();

} catch (IOException e) {

myExit("IO Exception");

}

String[] inputArr = input.trim().split(" ");

for (String s: inputArr) {

tree.Insert(Integer.parseInt(s));

}

tree.preOrder();

System.out.println();

tree.inOrder();

System.out.println();

tree.postOrder();

System.out.println();

}

}

**Question 5:**

**Given a positive integer your task is to implement the following function**

**F(X) = X / 2 if X is even**

**= (3 \* X) + 1 if X is odd**

**Now in main program take an input integer N and keep applying the function you**

**made, F, until the result is one. (It is guaranteed that the result will eventually reach 1**

**for the test cases that we have given). Output the value after each execution of the**

**function (including the input integer N and the final value 1) on a new line.**

ANSWER :

import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStreamReader;

public class Reduction {

public static void myExit(String code) {

System.out.println(code);

System.exit(1);

}

public static void main(String[] args) {

BufferedReader reader = new BufferedReader(new InputStreamReader(System.in));

String input = "";

try {

input = reader.readLine();

} catch (IOException e) {

myExit("IO Exception");

}

int N = Integer.parseInt(input.trim());

while (N != 1) {

System.out.println(N);

if (N % 2 == 0) {

N /= 2;

} else {

N = 3 \* N + 1;

}

}

System.out.println(N);

}

}

**Question 6:**

**Write a program that given a square matrix of odd dimensions, prints all elements of**

**the matrix, starting from the center, in the order of a spiral as shown in the examples**

**below.**

**Matrix Size: 7x7**

**Matrix Size: 5 x 5**

**ANSWER :**

import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStreamReader;

class Index {

public int i;

public int j;

}

public class Matrix {

int top;

int[][] matrix;

Index[] stack;

public static final int DOWN = 0;

public static final int RIGHT = 1;

public static final int UP = 2;

public static final int LEFT = 3;

public Matrix(int N) {

top = -1;

matrix = new int[N][N];

stack = new Index[N \* N + 1];

}

public void Push(Index t) {

this.top++;

Index obj = new Index();

obj.i = t.i;

obj.j = t.j;

this.stack[this.top] = obj;

}

public void Pop() {

Index obj = stack[top];

System.out.print(matrix[obj.i][obj.j] + " ");

this.top--;

}

public boolean isEmpty() {

return this.top == -1;

}

public static void myExit(String code) {

System.out.println(code);

System.exit(1);

}

public void set(int i, int j, int val) {

this.matrix[i][j] = val;

}

public int get(int i, int j) {

return this.matrix[i][j];

}

public void TraceLine(Index t, int dir, int squareLen) {

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

this.Push(t);

switch (dir) {

case UP:

t.i--;

break;

case DOWN:

t.i++;

break;

case LEFT:

t.j--;

break;

case RIGHT:

t.j++;

break;

}

}

}

public void TraceSquare(Index obj, int squareLen) {

Index t = new Index();

t.i = obj.i;

t.j = obj.j;

this.TraceLine(t, DOWN, squareLen);

this.TraceLine(t, RIGHT, squareLen);

this.TraceLine(t, UP, squareLen);

this.TraceLine(t, LEFT, squareLen);

}

public void Traverse(int N) {

int squareLen = N - 1;

Index obj = new Index();

obj.i = 0;

obj.j = 0;

while (squareLen > 0) {

this.TraceSquare(obj, squareLen);

squareLen -= 2;

obj.i++;

obj.j++;

}

this.Push(obj);

while (!this.isEmpty()) {

this.Pop();

}

}

public void Display(int N) {

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

for (int j = 0; j < N; j++) {

System.out.print(this.matrix[i][j] + " ");

}

System.out.println();

}

}

public static void main(String[] args) {

BufferedReader reader = new BufferedReader(new InputStreamReader(System.in));

String input = "";

try {

input = reader.readLine();

} catch (IOException e) {

myExit("IO Exception");

}

int N = Integer.parseInt(input.trim());

Matrix m = new Matrix(N);

for (int row = 0; row < N; row++) {

try {

input = reader.readLine();

} catch (IOException e) {

myExit("IO Exception");

}

String[] inputArr = input.trim().split(" ");

int col = 0;

for (String s: inputArr) {

m.set(row, col, Integer.parseInt(s));

col++;

}

}

//m.Display(N);//Debug

m.Traverse(N);

System.out.println();

}

}