/\*

Divya and Shruti are playing a dice rolling game. Find the winner based on

the user given game mode.

mode = "target" then the winner is who reaches the target score first

mode = "counter" then the winner is who scores more after counter times rolling.

\*/

import java.io.IOException;

import java.io.BufferedReader;

import java.io.InputStreamReader;

import java.util.ArrayList;

import java.util.Collection;

import java.util.Collections;

import java.util.LinkedList;

import java.util.Random;

public class First {

static void dice\_game() throws IOException {

String mode;

int d\_score = 0, s\_score = 0;

String winner = "";

Random r = new Random();

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

mode = input.readLine(); // Reading string

if(mode.equals("target")) {

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

int target = Integer.parseInt(input.readLine());

while(true) {

d\_score = d\_score + r.nextInt(1, 6);

if(d\_score >= target) {

winner = "Divya";

break;

}

s\_score = s\_score + r.nextInt(1, 6);

if(s\_score >= target) {

winner = "Shruti";

break;

}

}

}

else if(mode.equals("counter")) {

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

int counter = Integer.parseInt(input.readLine());

while(counter != 0) {

d\_score = d\_score + r.nextInt(1, 6);

s\_score = s\_score + r.nextInt(1, 6);

counter--;

}

if(d\_score > s\_score)

winner = "Divya";

else if(d\_score == s\_score)

winner = "Tie";

else

winner = "Shruti";

}

else {

System.out.println("Game mode is invalid!");

}

System.out.println("Divya score => " + d\_score);

System.out.println("Shruti score => " + s\_score);

System.out.println("Winner is " + winner);

}

static void array\_demo() {

int[] arr = new int[5];

}

static void chefs() throws IOException {

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

int n = Integer.parseInt(rd.readLine());

ArrayList<Integer> a = new ArrayList<>();

ArrayList<Integer> b = new ArrayList<>();

ArrayList<Integer> c = new ArrayList<>();

while(n != 0) {

String[] chef\_details = rd.readLine().split(" ");

if(chef\_details[0].equals("A"))

a.add(Integer.parseInt(chef\_details[1]));

if(chef\_details[0].equals("B"))

b.add(Integer.parseInt(chef\_details[1]));

if(chef\_details[0].equals("C"))

c.add(Integer.parseInt(chef\_details[1]));

n--;

}

if((Collections.min(a)+Collections.min(b)) > Collections.min(c))

System.out.println(Collections.min(c));

else

System.out.println(Collections.min(a)+Collections.min(b));

}

static void arrayList\_demo() {

// Collection\_name object\_name = new Collection\_name();

// collection\_name<data\_type> object\_name = new collection<data\_type>();

ArrayList info = new ArrayList();

ArrayList<Integer> data = new ArrayList<Integer>();

ArrayList<Integer> zero = new ArrayList<Integer>();

zero.add(0);

info.add(10);

info.add(5);

info.add(20);

info.add(30);

info.add(33);

info.add(15);

data.addAll(info);

System.out.println(info);

System.out.println(data);

}

static void sales() throws IOException {

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

String[] tmp = rd.readLine().split(" ");

int N = Integer.parseInt(tmp[0]);

int M = Integer.parseInt(tmp[1]);

ArrayList<Integer> result = new ArrayList<>();

while(N != 0) {

String t[] = rd.readLine().split(" ");

ArrayList<Integer> day\_sales = new ArrayList<>();

for(String s:t)

day\_sales.add(Integer.parseInt(s));

result.add(Collections.max(day\_sales));

N--;

}

for(int value:result)

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

}

static void goldCoins() throws IOException {

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

int n = Integer.parseInt(rd.readLine());

String[] tmp = rd.readLine().split(" ");

ArrayList<Integer> creates = new ArrayList<>();

ArrayList<Integer> zero = new ArrayList<>();

zero.add(0);

for(String s:tmp)

creates.add(Integer.parseInt(s));

System.out.println(creates);

int collected\_coins = 0;

while(creates.size() != 0) {

collected\_coins = collected\_coins + Collections.max(creates);

int index = creates.indexOf(Collections.max(creates));

if(creates.size() == 1) {

creates.set(index, 0);

} else if(index == 0) {

creates.set(index, 0);

creates.set(index+1, 0);

} else if(index == creates.size()-1) {

creates.set(index-1, 0);

creates.set(index, 0);

} else {

creates.set(index-1, 0);

creates.set(index, 0);

creates.set(index+1, 0);

}

creates.removeAll(zero);

System.out.println("Coins collected = " + collected\_coins);

System.out.println(creates);

}

}

static void collection\_demo() {

LinkedList ll = new LinkedList();

LinkedList l2 = new LinkedList();

l2.add(11);

l2.add(22);

l2.add(33);

ll.add(10);

ll.add(20);

ll.addFirst(30);

ll.addLast(40);

ll.addAll(2, l2);

System.out.println(ll);

}

// How to create a collection?

// How to insert?

// How to update?

// How to delete?

// How to access?

// What are additional features?

public static void main(String args[]) throws IOException {

collection\_demo();

}

}

—----

// A linked list is a collection elements in which each element has a link to its

// next element.

public class LL {

Node head;

int s = 0;

static class Node {

int data;

Node next;

Node(int data) {

this.data = data;

this.next = null;

}

}

void addFirst(int data) {

Node newNode = new Node(data);

if(head == null)

head = newNode;

else {

newNode.next = head;

head = newNode;

}

s++;

}

void addLast(int data) {

Node newNode = new Node(data);

if(head == null) {

head = newNode;

} else {

Node tmp = head;

while(tmp.next != null)

tmp = tmp.next;

tmp.next = newNode;

}

s++;

}

void addAt(int index, int data) {

Node newNode = new Node(data);

if(head == null)

head = newNode;

else if(index < 0 || index >= size()) {

addLast(data);

} else {

Node tmp = head;

while(index-1 != 0) {

tmp = tmp.next;

index--;

}

newNode.next = tmp.next;

tmp.next = newNode;

}

}

int removeFirst() {

if(head == null)

return -1;

int deleted = head.data;

head = head.next;

return deleted;

}

int removeLast() {

if(head == null)

return -1;

Node tmp = head;

Node previous = null;

while(tmp.next != null) {

previous = tmp;

tmp = tmp.next;

}

previous.next = null;

return tmp.data;

}

void reverse() {

if(head == null)

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

else {

Node previous = null;

Node current = head;

Node tmp = null;

while(current != null) {

tmp = current.next;

current.next = previous;

previous = current;

current = tmp;

}

head = previous;

}

}

int size() {

return s;

}

void printList() {

if(head == null) {

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

} else {

Node tmp = head;

while(tmp != null) {

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

tmp = tmp.next;

}

System.out.println("null");

}

}

public static void main(String args[]) {

LL list = new LL();

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

list.addFirst(10);

list.addFirst(20);

list.addFirst(30);

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

list.printList();

list.addLast(40);

list.printList();

list.addLast(50);

list.printList();

list.addFirst(5);

—---

// A linked list is a collection elements in which each element has a link to its

// next element.

public class LL {

Node head;

int s = 0;

static class Node {

int data;

Node next;

Node(int data) {

this.data = data;

this.next = null;

}

}

void addFirst(int data) {

Node newNode = new Node(data);

if(head == null)

head = newNode;

else {

newNode.next = head;

head = newNode;

}

s++;

}

void addLast(int data) {

Node newNode = new Node(data);

if(head == null) {

head = newNode;

} else {

Node tmp = head;

while(tmp.next != null)

tmp = tmp.next;

tmp.next = newNode;

}

s++;

}

void addAt(int index, int data) {

Node newNode = new Node(data);

if(head == null)

head = newNode;

else if(index < 0 || index >= size()) {

addLast(data);

} else {

Node tmp = head;

while(index-1 != 0) {

tmp = tmp.next;

index--;

}

newNode.next = tmp.next;

tmp.next = newNode;

}

}

int removeFirst() {

if(head == null)

return -1;

int deleted = head.data;

head = head.next;

return deleted;

}

int removeLast() {

if(head == null)

return -1;

Node tmp = head;

Node previous = null;

while(tmp.next != null) {

previous = tmp;

tmp = tmp.next;

}

previous.next = null;

return tmp.data;

}

void reverse() {

if(head == null)

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

else {

Node previous = null;

Node current = head;

Node tmp = null;

while(current != null) {

tmp = current.next;

current.next = previous;

previous = current;

current = tmp;

}

head = previous;

}

}

int size() {

return s;

}

void printList() {

if(head == null) {

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

} else {

Node tmp = head;

while(tmp != null) {

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

tmp = tmp.next;

}

System.out.println("null");

}

}

public static void main(String args[]) {

LL list = new LL();

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

list.addFirst(10);

list.addFirst(20);

list.addFirst(30);

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

list.printList();

list.addLast(40);

list.printList();

list.addLast(50);

list.printList();

list.addFirst(5);

list.printList();

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

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

list.addAt(2, 100);

list.printList();

list.addAt(5, 500);

list.printList();

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

list.printList();

list.reverse();

list.printList();

}

}