**Stack and Queue**

**Question 1:**

**Stack 2 : Array Implementation : Push and Pop**

Write a program to implement push and pop operation on stack and to display the contents of the stack.  
  
Create a class named Stack and include the following methods

|  |  |
| --- | --- |
| **Method Name** | **Description** |
| void push(int data) | This method takes the data to be pushed in the stack and Print the message “Stack is full” in the push function when an attempt is made to push a data into a full stack. |
| int pop() | This method to pop the data from the stack and Print the message “Stack is empty” in the pop function and return the value -1000 when an attempt is made to pop data from an empty stack. |
| void display() | This method to display all the data in the stack. |

In the Stack function intialize the value of top to -1 and initialize the value of maxSize.

Refer function specifications for further details.

**Input and Output Format:**

Refer sample input and output for formatting specifications.  
  
Note that the statement “The contents of the stack are” is in the main function. In the display function, if the stack is empty, print “ {}”.

[All text in bold corresponds to input and the rest corresponds to output]

**Sample Input and Output:**

Enter the maximum size of the stack

**3**

Choice 1 : Push

Choice 2 : Pop

Choice 3 : Display

Any other choice : Exit

Enter your choice

**1**

Enter the element to be pushed

**1**

Choice 1 : Push

Choice 2 : Pop

Choice 3 : Display

Any other choice : Exit

Enter your choice

**1**

Enter the element to be pushed

**2**

Choice 1 : Push

Choice 2 : Pop

Choice 3 : Display

Any other choice : Exit

Enter your choice

**1**

Enter the element to be pushed

**3**

Choice 1 : Push

Choice 2 : Pop

Choice 3 : Display

Any other choice : Exit

Enter your choice

**1**

Enter the element to be pushed

**4**

Stack is full

Choice 1 : Push

Choice 2 : Pop

Choice 3 : Display

Any other choice : Exit

Enter your choice

**3**

The contents of the stack are 1 2 3

Choice 1 : Push

Choice 2 : Pop

Choice 3 : Display

Any other choice : Exit

Enter your choice

**2**

The popped element is 3

Choice 1 : Push

Choice 2 : Pop

Choice 3 : Display

Any other choice : Exit

Enter your choice

**2**

The popped element is 2

Choice 1 : Push

Choice 2 : Pop

Choice 3 : Display

Any other choice : Exit

Enter your choice

**2**

The popped element is 1

Choice 1 : Push

Choice 2 : Pop

Choice 3 : Display

Any other choice : Exit

Enter your choice

**2**

Stack is empty

Choice 1 : Push

Choice 2 : Pop

Choice 3 : Display

Any other choice : Exit

Enter your choice

**3**

The contents of the stack are {}

Choice 1 : Push

Choice 2 : Pop

Choice 3 : Display

Any other choice : Exit

Enter your choice

**4**

**Program**

**Main.java**

import java.util.\*;

public class Main{

public static void main(String[] args) {

    int choice=0;

    Scanner sc = new Scanner(System.in);

    System.out.println("Enter the maximum size of the stack");

    int n=sc.nextInt();

    Stack s = new Stack(n);

    do{

        System.out.println("Choice 1 : Push\nChoice 2 : Pop\nChoice 3 : Display\nAny other choice : Exit");

        System.out.println("Enter your choice");

        choice = sc.nextInt();

        switch(choice)

        {

            case 1:

            {

                System.out.println("Enter the element to be pushed");

                int a=sc.nextInt();

                sc.nextLine();

                s.push(a);

                break;

            }

            case 2:

            {

                int b=s.pop();

                if(b!=-1000){

                    System.out.println("The popped element is "+b);

                }

                break;

            }

            case 3:

            {

                System.out.print("The contents of the stack are"+" ");

                s.display();

                System.out.println();

                break;

            }

            default:

            {

                System.exit(0);

                break;

            }     }

    }while(choice<4);

} }

**Stack.java**

import java.util.\*;

public class Stack{

    int top=-1;

    int[] arr;

    int n;

    public Stack(int n){

        this.n=n;

        arr=new int[n];

    }

        public void push(int data)

    {

    if(top==arr.length-1){

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

    }else{

        top++;

        arr[top]=data;

    }

    }

    public int pop()

    {

     if(top==-1){

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

         return -1000;

     }else{

         int temp=top;

         top--;

         return arr[temp];

     }

    }

    public void display()

    {

    if(top==-1){

        System.out.println(" {}");

    }else{

        for(int i=0;i<=top;i++){

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

        }

    }

    }

}

**Question 2:**

**Queue 2 : Array Implementation using rear index**

Consider implementing a fixed size circular queue of maximum size 5 using an array.

Note that the array contents holds the contents of the queue and the integer front stores the index of the front element in the queue and the integer rear stores the index of the last element in the queue.

Write a java program to implement enQueue and deQueue operation on queue and to display the contents of the queue.

Create a class named Queue and include the following methods

|  |  |
| --- | --- |
| **Method Name** | **Description** |
| void enQueue(int data) | This method takes the data to be inserted in the queue and Print the message “Queue is full” in the enQueue function when an attempt is made to insert a data into a full queue. |
| int deQueue() | This method to delete the data from the queue and Print the message “Queue is empty” in the deQueue function and return the value -1000 when an attempt is made to delete data from an empty queue. |
| void display() | This method to display all the data in the queue. |

In the Queue function intialize the value of front and rear to -1.

Refer function specifications for further details.

**Input and Output Format:**

Refer sample input and output for formatting specifications.

Note that the statement “The contents of the queue are” is in the main function. In the display function, if the queue is empty, print “ {}”.

**[All text in bold corresponds to input and the rest corresponds to output]**

**Sample Input and Output:**

Choice 1 : Enter element into Queue

Choice 2 : Delete element from Queue

Choice 3 : Display

Any other choice : Exit

Enter your choice

**3**

The contents of the queue are {}

Choice 1 : Enter element into Queue

Choice 2 : Delete element from Queue

Choice 3 : Display

Any other choice : Exit

Enter your choice

**1**

Enter the element to be inserted/entered

**10**

Choice 1 : Enter element into Queue

Choice 2 : Delete element from Queue

Choice 3 : Display

Any other choice : Exit

Enter your choice

**1**

Enter the element to be inserted/entered

**20**

Choice 1 : Enter element into Queue

Choice 2 : Delete element from Queue

Choice 3 : Display

Any other choice : Exit

Enter your choice

**1**

Enter the element to be inserted/entered

**30**

Choice 1 : Enter element into Queue

Choice 2 : Delete element from Queue

Choice 3 : Display

Any other choice : Exit

Enter your choice

**1**

Enter the element to be inserted/entered

**40**

Choice 1 : Enter element into Queue

Choice 2 : Delete element from Queue

Choice 3 : Display

Any other choice : Exit

Enter your choice

**1**

Enter the element to be inserted/entered

**50**

Choice 1 : Enter element into Queue

Choice 2 : Delete element from Queue

Choice 3 : Display

Any other choice : Exit

Enter your choice

**1**

Enter the element to be inserted/entered

**60**

Queue is full

Choice 1 : Enter element into Queue

Choice 2 : Delete element from Queue

Choice 3 : Display

Any other choice : Exit

Enter your choice

**2**

The deleted element is 10

Choice 1 : Enter element into Queue

Choice 2 : Delete element from Queue

Choice 3 : Display

Any other choice : Exit

Enter your choice

**2**

The deleted element is 20

Choice 1 : Enter element into Queue

Choice 2 : Delete element from Queue

Choice 3 : Display

Any other choice : Exit

Enter your choice

**1**

Enter the element to be inserted/entered

**60**

Choice 1 : Enter element into Queue

Choice 2 : Delete element from Queue

Choice 3 : Display

Any other choice : Exit

Enter your choice

**3**

The contents of the queue are 30 40 50 60

Choice 1 : Enter element into Queue

Choice 2 : Delete element from Queue

Choice 3 : Display

Any other choice : Exit

Enter your choice

**2**

The deleted element is 30

Choice 1 : Enter element into Queue

Choice 2 : Delete element from Queue

Choice 3 : Display

Any other choice : Exit

Enter your choice

**2**

The deleted element is 40

Choice 1 : Enter element into Queue

Choice 2 : Delete element from Queue

Choice 3 : Display

Any other choice : Exit

Enter your choice

**2**

The deleted element is 50

Choice 1 : Enter element into Queue

Choice 2 : Delete element from Queue

Choice 3 : Display

Any other choice : Exit

Enter your choice

**3**

The contents of the queue are 60

Choice 1 : Enter element into Queue

Choice 2 : Delete element from Queue

Choice 3 : Display

Any other choice : Exit

Enter your choice

**2**

The deleted element is 60

Choice 1 : Enter element into Queue

Choice 2 : Delete element from Queue

Choice 3 : Display

Any other choice : Exit

Enter your choice

**2**

Queue is empty

Choice 1 : Enter element into Queue

Choice 2 : Delete element from Queue

Choice 3 : Display

Any other choice : Exit

Enter your choice

**3**

The contents of the queue are {}

Choice 1 : Enter element into Queue

Choice 2 : Delete element from Queue

Choice 3 : Display

Any other choice : Exit

Enter your choice

**4**

**Program**

**Main.java**

import java.util.\*;

public class Main{

public static void main(String[] args) {

    int choice=0;

    Scanner sc = new Scanner(System.in);

    Queue q = new Queue();

    do{

        System.out.println("Choice 1 : Enter element into Queue\nChoice 2 : Delete element from Queue\nChoice 3 : Display\nAny other choice : Exit");

        System.out.println("Enter your choice");

        choice = sc.nextInt();

        switch(choice)

        {

            case 1:

            {

                System.out.println("Enter the element to be inserted/entered");

                int a=sc.nextInt();

                sc.nextLine();

                q.enQueue(a);

                break;

            }

            case 2:

            {

                int b=q.deQueue();

                if(b!=-1000)

                System.out.println("The deleted element is "+b);

                break;

            }

            case 3:

            {

        System.out.print("The contents of the queue are");

                q.display();

                System.out.println();

                break;

            }

            default:

            {

                System.exit(0);

                break;

            }

    }

    }while(choice<4);

}

}

**Queue.java**

import java.util.\*;

public class Queue{

     int[] arr=new int[5];

     int N=arr.length;

     int front=-1,rear=-1;

     public boolean isEmpty(){

    if(front==-1&&rear==-1) return true;

    return false;

     }

     public boolean isFull(){

         if((rear+1)%N==front)return true;

         return false;

     }

    public void enQueue(int data)

    {

    if(isFull()){

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

        return;

    }if(isEmpty()){

        front=0;

        rear=0;

    }else{

        rear=(rear+1)%N;

    }arr[rear]=data;

    }

    public int deQueue()

    {

     if(isEmpty()){

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

         return -1000;

     }

     int v=arr[front];

     if(front==rear){

         front=rear=-1;

     }else{

         front=(front+1)%N;

     }return v;

    }

    public void display()

    {

    /\* Fill your code here \*/

    if(isEmpty()){

        System.out.println(" {}");

         return ;

    }int i=front;

    while(true){

        System.out.print(" "+arr[i]);

        if(i==rear)break;

        i=(i+1)%N;

    }

    }

}

**Question 3:**

**David and Cakes**

David loves Cakes and is much interested in baking. Hence he joined as a Manager at a famous bakery 'Cake Walk' which is called the cake lovers' paradise in the town. Because of the bakery's tantalizing and delicious wide range of freshly baked cakes served in attractive cake boxes, people are seen always queuing up to have one of those boxes, which is why David's job is not so easy. Each cake box has a cost associated with it. The cake boxes are kept as a pile. He needs to handle two types of queries:

1) Customer Query:  
When a customer demands a cake box, the box on the **top** of the pile is given and the customer is charged according to the cost of the box. This reduces the height of the pile by 1.  
In case the pile is empty, the customer goes away empty-handed.

2) Baker Query:  
The baker prepares a cake, wraps it in a box and adds it on top of the pile. And reports the cost of the box to David.

Help David manage the process.

**Input Format:**  
First-line contains an integer Q, the number of queries. Q lines follow.  
A Type-1 (Customer) Query, is indicated by a single integer 1 in the line.  
A Type-2 (Baker) Query, is indicated by two space-separated integers 2 and C where C is the cost of the box prepared.

**Output Format:**  
For each Type-1 Query, output the price that the customer has to pay i.e. cost of the box given to the customer in a new line. If the pile is empty, print "No Cake" (without the quotes).

**Sample Input:**  
**6  
1  
2 5  
2 7  
2 9  
1  
1**

**Sample Output:**  
No Cake  
9  
7

**Explanation:**

Initially, The pile is empty.  
Baker adds a box with cost=5.  
Baker adds a box with cost=7.  
Baker adds a box with cost=9.  
The customer takes the box on the top i.e. cost=9. Now box of cost=7 on top. The customer takes the box on the top i.e. cost=7.

**Program**

**Main.java**

import java.util.\*;

import java.util.Stack;

class Main

{

    public static void main(String args[])

    {

        //Fill your code here

    Scanner sc=new Scanner(System.in);

    int N=sc.nextInt();

    Stack<Integer> s=new Stack<>();

    for(int i=0;i<N;i++){

        int q=sc.nextInt();

        if(q==1){

            if(s.isEmpty()){

                System.out.println("No Cake");

            }else{

                System.out.println(s.pop());

            }

        }if(q==2){

            int c=sc.nextInt();

            s.push(c);

        }

    }

        sc.close();

    }

}

**Question 4:**

**Corporate Fiesta**

Corporate Fiesta is the first of its kind Carnival that has a unique blend of sports and arts, catering to varied interest of employees belonging to the Corporate companies of all sizes across the City. The Carnival had many events for different sections of employees.  
  
One notable event is a simple ball passing game called FORWARD and REVERSE organised for the smart Coders in the Corporate. In this event, the coders were given N passes and players having ids between 1 and 1000000. Initially some player with a given id had the ball in his possession. The coders had to make a program to display the id of the player who possessed the ball after exactly N passes.  
  
Description of the passes:  
There were two kinds of passes:  
1. F ID  
2. R  
  
**Explanation :**  
For the first kind of pass, the player in possession of the ball passes the ball to player with id=ID, while for the second kind of a pass, the player in possession of the ball passes the ball back to the player who had passed the ball to him.  
  
**Note:**  
It is guaranteed that the given order of all the passes will be a valid order .

**Define a structure**  
struct PIDs  
{  
int \*idStack;  
int top;  
int size;  
};

**Function Specifications:**

void init(struct PIDs\*, int );  
void push(struct PIDs\* , int );  
void pop(struct PIDs \*);  
int peek(PIDs);

**Input Format :**  
The first line of the input contains two space separated integers N, the number of passes and ID of the player possessing the ball in the very beginning. (1≤N≤100000 and 1≤ID≤1000000)  
N lines follow describing the passes. ( for description of the passes, refer the statement above. )  
  
**Output Format :**  
Output to each test case should be a single line containing the "Player" ID (quotes for clarity) of the player who possesses the ball after N passes.  
Refer sample input and output for formatting specifications.  
  
**Sample Input:**  
10 23  
F 86  
F 63  
F 60  
R  
F 47  
R  
F 99  
F 9  
R  
R  
  
**Sample Output:**  
Player 9  
  
**Explanation:**  
Initially, Player having id=23 possesses the ball. After pass 1, Player having id=86 possesses the ball. After pass 2, Player having id=63 possesses the ball. After pass 3, Player having id=60 possesses the ball. After pass 4, Player having id=63 possesses the ball. After pass 5, Player having id=47 possesses the ball. After pass 6, Player having id=63 possesses the ball. After pass 7, Player having id=99 possesses the ball. After pass 8, Player having id=9 possesses the ball. After pass 9, Player having id=99 possesses the ball. After pass 10, Player having id=9 possesses the ball. So the output is 9.

**Program**

**Main.java**

import java.util.\*;

import java.util.Stack;

import java.io.\*;

class Main

{

    public static void main(String args[])

    {

        //Fill your code here

    Scanner sc=new Scanner(System.in);

    int n=sc.nextInt();

    int c=sc.nextInt();

    Stack<Integer> s=new Stack<>();

    s.push(c);

    int dup,player;

    for(int i=0;i<n;i++){

        String line=sc.next();

        if(line.equals("F")){

            int newId=sc.nextInt();

            s.push(newId);

        }if(line.equals("R")){

            dup=s.pop();

            player=s.peek();

            s.push(dup);

            s.push(player);

        }

    }System.out.println("Player "+s.peek());

    sc.close();

    }

}

**Question 5:**

**Shankar and his new Project**

Shankar is a diligent, hardworking IT professional working for a reputed Software concern. He sets clear targets to achieve his tasks and is strongly committed to attaining it. Shankar's Boss has assigned him with a new, most vital project of the quarter related to Banking services and it has N processes in it. All the processes have a unique number assigned to them from **1 to N**.

He is given two things:

* The **calling** order in which all the processes are called.
* The **ideal** order in which all the processes should have been executed.

Now, let us demonstrate this by an example. Let's say that there are **3 processes**, the calling order of the processes is: **3 - 2 - 1.** The ideal order is: **1 - 3 - 2,** i.e., process number 3 will only be executed after process number 1 has been completed; process number 2 will only be executed after process number 3 has been executed.

* *Iteration #1:* Since the ideal order has process #1 to be executed firstly, the calling ordered is changed, i.e., the first element has to be pushed to the last place. Changing the position of the element takes 1 unit of time. The new calling order is: 2 - 1 - 3. Time is taken in step #1: 1.
* *Iteration #2:* Since the ideal order has process #1 to be executed firstly, the calling ordered has to be changed again, i.e., the first element has to be pushed to the last place. The new calling order is: 1 - 3 - 2. Time is taken in step #2: 1.
* *Iteration #3:* Since the first element of the calling order is the same as the ideal order, that process will be executed. And it will be thus popped out. Time is taken in step #3: 1.
* *Iteration #4:* Since the new first element of the calling order is the same as the ideal order, that process will be executed. Time is taken in step #4: 1.
* *Iteration #5:* Since the last element of the calling order is the same as the ideal order, that process will be executed. Time is taken in step #5: 1.

Total time taken: 5 units.

**Note:** Executing a process takes 1 unit of time. Changing the position takes 1 unit of time.

Shankar's initial task in the project is to estimate the total time taken for the entire queue of processes to be executed. Can you help him on the same?

**Input format:**  
The first line of the input is an integer **N**, denoting the number of processes; 1<=**N**<=100.

The second line contains the calling order of the processes.

The third line contains the ideal order of the processes.

**Output format:**  
Print the total time taken for the entire queue of processes to be executed.

Refer to sample input and output for formatting specifications.

**[All text in bold corresponds to input and the rest corresponds to output]**

**Sample Input and Output:**

3

3 2 1

1 3 2

5

**Program**

**Main.java**

import java.util.\*;

public class Main{

public static void main(String[] args) {

    int n,j=0,tim=0,i,last;

    Scanner sc = new Scanner(System.in);

    n=sc.nextInt();

    Queue q = new Queue(n);

    int[] a=new int[n];

    int[] b=new int[n];

    for(i=0;i<n;i++)

    a[i]=sc.nextInt();

    for(i=0;i<n;i++)

    //Fill your code here

    q.enQueue(a[i]);

    for(i=0;i<n;i++)

    b[i]=sc.nextInt();

     while(j!=n){

        if(q.front()!=b[j]){

            //Fill your code here

            int temp=q.deQueue();

            q.enQueue(temp);

            tim++;

            }

        else{

            //Fill your code here

            q.deQueue();

            tim++;

            j++;

        }

    }

    System.out.println(tim);

}

}

**Queue.java**

import java.util.\*;

public class Queue{

     int[] arr;

     int front,rear,size,capacity;

     public Queue(int cap){

         arr=new int[cap];

         capacity=cap;

         front=0;

         rear=0;

         size=0;

     }

    public void enQueue(int data)

    {

    if(size==capacity)return ;

    arr[rear]=data;

    rear=(rear+1)%capacity;

    size++;

    }

    public int deQueue()

    {

     if(size==0)return -1;

     int val=arr[front];

     front=(front+1)%capacity;

     size--;

     return val;

    }

    public int front()

        {

    if(size==0)return -1;

    return arr[front];

    }

}