**Assignment 01:**

You have an empty sequence, and you will be given

queries. Each query is one of these three types:

1 x  -Push the element x into the stack.

2    -Delete the element present at the top of the stack.

3    -Print the maximum element in the stack.

**Function Description**

Complete the *getMax* function in the editor below.

*getMax* has the following parameters:

- *string operations[n]:* operations as strings

**Returns**

- *int[]:* the answers to each type 3 query

**Input Format**

The first line of input contains an integer,

. The next

lines each contain an above mentioned query.

**Constraints**

**1<=n<=105**

**1<=x<=109**

**1<=type<=3**

All queries are valid.

**Sample Input**

STDIN   Function

-----   --------

10      operations[] size n = 10

1 97    operations = ['1 97', '2', '1 20', ....]

2

1 20

2

1 26

1 20

2

3

1 91

3

**Sample Output**

26

91

**Code:**

package leet;

import java.util.\*;

import java.util.Scanner;

class Solution

{

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

int maxEle;

public void getMax()

{

if (s.empty())

return;

else

System.*out*.print(maxEle + "\n");

}

public void peek()

{

if (s.empty())

{

return ;

}

int t = s.peek();

if(t > maxEle)

System.*out*.print(maxEle);

else

System.*out*.print(t);

}

public void pop()

{

if (s.empty())

{

return;

}

int t = s.peek();

s.pop();

if (t > maxEle)

{

System.*out*.print(maxEle + "\n");

maxEle = 2 \* maxEle - t;

}

else

System.*out*.print(t + "\n");

}

public void push(int x)

{

if (s.empty())

{

maxEle = x;

s.push(x);

System.*out*.print( x + "\n");

return;

}

if (x > maxEle)

{

s.push(2 \* x - maxEle);

maxEle = x;

}

else

s.push(x);

System.*out*.print( x + "\n");

}

public static void main(String[] args)

{

Solution s = new Solution();

Scanner sc = new Scanner(System.*in*);

int ch = sc.nextInt();

switch(ch){

case 1:

int a = sc.nextInt();

s.push(a);

break;

case 2:

s.pop();

break;

case 3:

s.getMax();

}

}

}