Problem statement :

Murali krishna is trying to understand the concept of “register” storage class specifier in C language. After several experiments he noticed that, At compilation stage itself **C compiler** is taking care of generating instructions such a way that weather to allocate “stack or register” for the variable. For example, in x86 architecture 6 general purpose registers are free for “operands”. If programmer requests for 7 registers, the gcc compiler is designed in such a way that it can able to assume the scarcity of registers and it generates 6 instructions for register allocation and 1 instruction for stack allocation.

Murali wants to implement the same algorithm for his **chota compiler**. Please, help him by finding an effecient way.

Input :

The first line of input contains an integer value **N** which tells about no of statements will be given.
Eachline of next **N** statements contain **3** space seperated **integers.**
vid <space> resourse <space> status.

vid(Variable id)
resourse (0 or 1) '**0**' means regsiter and '**1**' means stack.
status (0 or 1) '**1**' means allocation and '**0**' means deallocation.

Example satement : 1
5 0 1
Allocate **variable 5** in stack**.**

Example statement : 2
2 0 0
Deallocate **variable2** from register.

Output :

For each statement, print a single line containing the string
"**RA**" if the variable is allocated in register.
“**SA**” if variable is allocated in stack.
“**DA**” if deallocated from **register or stack.**
“**AA**” if variable is already allocated in **stack or register.**
“NA” if variable is not available to deallocate.
“**INV**” if user enters invalid input.

Constraints :

0 < **vid** < 10^9
- 10^9 < **resourse** < 10^9
- 10^9 < **status <** 10^9

Example 1 :

9
1 0 1
2 0 1
3 0 1
4 0 1
5 0 1
6 0 1
7 0 1
3 0 0
7 0 1
15 0 1

output :

RA
RA
RA
RA
RA
RA
SA
DA
AA
RA

Explanation :

In the above example user requested register allocation for varaible '1'. Because of there are 6 free registers . So chota compiler allocated one register to variable '1' .

After 6 statements 6 registers are allocated for variables 1,2,3,4,5,6. Now variable 7 is also requesting for registers. But there is no free registers to allocate. so now chota compiler pushes this variable into stack and prints SA.

By seeing **Statement 8** compiler understands that variable 3 is deallocated. So now one register is free .

In s**tatement 9** variable 7 is requesting for register. But it is already allocated in stack. So chota compiler prints “AA” on the screen.

In **statement 10** variable 15 is requesting for register. Now chota compiler knows that one register is free so it will allocate that register to variable 15.

Example 2 :

5
99 1 1
1 1 0
1 1 1
1 1 1
5 23 0

output :

SA
NA
SA
AA
INV

Explanation :

In the above example in **statement 1** user requested stack allocation for varaible '99'. Because of Stack is unlimited, chota compiler allocated variable 99 in stack.

In s**tatement 2** variable 1 is requesting for deallocation from stack. But in Past there is no allocation for variable 1 in stack. So **NA** is printed.

In s**tatement 5** resourse value is other than 0 or 1. So it will be considered as invalid input **(INV)**.