# Accidentes aéreos

Estamos programando un *bot* de Telegram que enviará noticias a sus seguidores y ahora nos toca tratar el escabroso tema de los accidentes de aviación. Cuando se produce uno de estos accidentes, suelen hacerse comentarios del estilo "*Este es el accidente más grave desde febrero de 1995*".

Nos han pasado un listado (ordenado cronológicamente) de accidentes ocurridos en el pasado y queremos estar preparados para que, cuando ocurra el siguiente accidente, podamos producir un comentario como el anterior.

De hecho, para probar esta funcionalidad, queremos saber cuál habría sido el comentario cuando se produjo cada uno de los accidentes conocidos.

## Entrada

La entrada está formada por una serie de casos. Cada caso comienza con el número $N$ de accidentes conocidos (un número entre 1 y 250.000). A continuación aparecen $N$ líneas con la descripción de cada uno de ellos: una fecha (con formado `DD/MM/AAAA`) y el número de víctimas en ese accidente. Todas las fechas son distintas y el listado está ordenado cronológicamente de menor a mayor.

## Salida

Para cada caso se escribirán $N$ líneas. La $i$-ésima línea contendrá la fecha del último accidente anterior que tuvo (estrictamente) más víctimas que el accidente $i$-ésimo. Si no existe tal accidente (en particular, eso ocurre siempre para el primero), se escribirá `NO HAY` en su lugar.

Después de cada caso se escribirá una línea con tres guiones (`---`).

## Entrada de ejemplo

```
6
19/12/1990 50
01/02/2000 80
10/05/2001 30
20/10/2005 10
08/07/2007 60
10/07/2007 40
```

## Salida de ejemplo

```
NO HAY
NO HAY
01/02/2000
10/05/2001
01/02/2000
08/07/2007
---
```

**Autor:** Alberto Verdejo.

```
//
//  stack_eda.h
//
//  Implementación del TAD pila con array dinámico
//
//  Estructuras de Datos y Algoritmos
//  Facultad de Informática
//  Universidad Complutense de Madrid









                    ______        ____
    size_t nelems;

    // tamaño del array
    size_t capacidad;

    // puntero al array que contiene los datos (redimensionable)
    T * array;

public:

    // constructor: pila vacía
    stack() : nelems(0), capacidad(TAM_INICIAL), array(new T[capacidad]) {}

    // destructor
    ~stack() {
        libera();
    }

    // constructor por copia
```

```
    void push(T const& elem) {
        if (nelems == capacidad)
            amplia();
        array[nelems] = elem;
        ++nelems;
    }

    // consultar la cima
    T const& top() const {
        if (empty())
            throw std::domain_error("la pila vacia no tiene cima");
        return array[nelems - 1];
    }

    // desapilar el elemento en la cima
    void pop() {
        if (empty())
            throw std::domain_error("desapilando de la pila vacia");
        --nelems;
    }

    // consultar si la pila está vacía
    bool empty() const {
        return nelems == 0;
    }

    // consultar el tamaño de la pila
    size_t size() const {
        return nelems;
    }

protected:

    void libera() {
        delete[] array;
        array = nullptr;
    }

    // this está sin inicializar
    void copia(stack const& other) {
        capacidad = other.nelems;
        nelems = other.nelems;
        array = new T[capacidad];
        for (size_t i = 0; i < nelems; ++i)
            array[i] = other.array[i];
    }

    void amplia() {
        T * viejo = array;
        capacidad *= 2;
        array = new T[capacidad];
        for (size_t i = 0; i < nelems; ++i)
            array[i] = std::move(viejo[i]);
        delete[] viejo;
    }
};
```

```
#endif // stack_eda_h
```

```
#ifndef E07_H
#define E07_H

#include <iostream>
#include <fstream>
#include <iomanip>

//Clase Fecha para la fecha de los accidentes
class Fecha{
private:
    int dia, mes, anyo;
public:
    Fecha(){};
    Fecha(int d, int m, int a) : dia(d), mes(m), anyo(a){};

    int getD() const { return dia; };
    int getM() const { return mes; };
    int getA() const { return anyo; };
};
class Accidentes{
private:
    Fecha fecha, anterior;
    int victimas;

public:
    Accidentes(){};
    Accidentes(Fecha f, int v) : fecha(f), victimas(v){};
    Accidentes(Fecha f, int v, Fecha a) : fecha(f), victimas(v), anterior(a){};

    int getV() const { return victimas; };
    Fecha getF() const { return fecha; };

};

inline std::ostream & operator<< (std::ostream & out, Fecha const & h) {
    out << std::setfill('0') << std::setw(2) << h.getD() << '/' << std::setfill('0') << std::
    setw(2) << h.getM() << '/' << std::setfill('0') << std::setw(4) << h.getA();
    return out;
}
#endif
```

```cpp
#include "E07.h"
#include "stack_eda.h"

bool resuelveCaso(){
    char c;
    int d, m, a, v, nAcc;
    stack<Accidentes> pila;

    std::cin >> nAcc;

    if (!std::cin)
        return false;

    while (nAcc > 0){
        bool enc = false;
        std::cin >> d >> c >> m >> c >> a >> v;
        Fecha f(d, m, a);
        Accidentes acc(f, v);

        while (!enc && !pila.empty()){
            if (pila.top().getV() > v){
                std::cout << pila.top().getF() << std::endl;
                pila.push(acc);
                enc = true;
            }
            else {
                pila.pop();
            }
        }

        if (pila.empty()){
            std::cout << "NO HAY" << std::endl;
            pila.push(acc);
        }

        nAcc--;
    }

    std::cout << "---" << std::endl;
    return true;
}


int main() {

    // ajuste para que cin extraiga directamente de un fichero
#ifndef DOMJUDGE
    std::ifstream in("casos.txt");
    auto cinbuf = std::cin.rdbuf(in.rdbuf());
#endif

    while (resuelveCaso());
    system("PAUSE");
    // restablecimiento de cin
#ifndef DOMJUDGE
    std::cin.rdbuf(cinbuf);
#endif
```

```
    return 0;
}
```