Law Firm

A low firm has established the following three levels of priorities of their cases by their urgency: 0 (high priority), 1 (medium priority) and 2 (low priority). All the cases of a given priority will be handled before the cases of the next priority. Each case has a unique ID, a name and a type (it can be any word). You are asked to define an abstract data type (ADT) with the following operations:

- Create(): Builder to create a law firm with no cases
- addCase (id, name, type, priority): It adds a new case with a certain "id", "name" and "type", all of these represented as strings. The priority is represented with an integer value from 0 to 2.
- getCase(id, name, type): In "name" and "type" output parameters, it stores the information of the case with the given "id". If there is not such case, it should raise an error "This case does not exist"
- nextCase(id, priority): It stores the id of the next case and its priority to be handled, in the two output parameters. Notice that high-priority cases are handled before medium-priority cases, and medium-priority cases are handled before low-priority cases. Inside each priority, the first case that was added, should be the first one that should be handled. If there are no cases, it should raise an error "There are no cases".
- removeNextCase(): It removes the next case from the whole system, leaving no trace in the system. If there is not any case, it does nothing.
- empty(): It returns true if there are not any case to be handled, and false otherwise.

It is mandatory to define a C++ class for implementing this ADT. This ADT can use any of the ADTs of the virtual campus. It is recommended to use exceptions for handling errors. You can use exceptions of the updated "Exceptions.h" file in virtual campus (e.g. EWrongKey and EEmpty) if you find it appropriate, or define your new exceptions, as you prefer. You need to define a main function that uses the class of the new ADT for handling the input and generating the output.

Indicate the **complexity cost of each operation explaining it**, as a comment prior to each operation in the programming code.

Input

The first line will indicate the number of cases. Each case will be represented with n + 1 lines: In each case, the first line will indicate the number of operations referred as n. The next n lines have the operations.

The syntax of the operations will be the following ones:

add <id> <name> <type> <priority>: It adds a new case with certain id, name, type and priority, without printing anything.

get <id>: It gets a case with a given "id" and prints the name and type separated with a space and a line break at the end. It prints "This case does not exist" if there is no case with this id.

next: It prints the id and priority of the next case to be handled separated with a space, and breaks the line at the end. It prints "There are no cases" if there is not any case to be handled.

remove: It removes the next case if there is any to be removed, without printing anything.

empty: It prints "yes" if there are no cases to be handled or "no" otherwise, breaking the line at the end.

Output

The output of each case will be just the printed messages of all the operations without adding new extra line breaks, as each operation prints its break line if printing anything.

Example of input

```
2
16
add  c77 Bridge traffic 2
add c100 Vigo criminal 0
add c111 Juan traffic 2
add c112 Susana divorce 1
get c111
get g22
next
remove
next
remove
next
remove
next
remove
next
empty
6
add a2 Esther criminal 0
empty
remove
remove
empty
get a2
```

Example of output

Juan Traffic
This case does not exist
c100 0
c112 1
c77 2
c111 2
There are no cases
yes
no
yes
This case does not exist

Remember an example of throwing a new exception:
throw EEmpty("The data structure is empty");
Remember an example of catching an exception:
try{

```
…
} catch (EEmpty & e){
      cout<<e.msg()<<endl;
}
```