HW 6: The Banker's Algorithm

Your job is to implement the banker's algorithm, to keep deadlock from occuring. Your source code should be called `banker.c`. The program will take a single command-line argument: the name of a scenario file to read. Here is a sample scenario file:

```
2	1	1
2	1	0
1	1	1

+0A
+0A
+1B
+0B
+1A
-1A
!1
+0C
```

The first part of the file is the setup. It contains $(p + 1)$ lines, where p is the number of processes. These lines are tab-separated, and the number of columns is the number of resources. Processes are numbered consecutively starting at 0, and resources are lettered consecutively starting at A. (There will only be 26 resources at most.) The first line indicates the total number of resources there are of each type. In the example there are 2 of type A, and one each of types B and C. The remaining lines are the declarations each process has made, as to the maximum number of each resource it will require. So process 0 has declared that it will only ever need 2 of resource A, 1 of B, and 0 of C.

The second part begins after the skipped line. It is the precise sequence of events that occurs. There are 3 kinds of events:

- **Request:** These lines begin with a `+`, followed by a number and a letter with no spaces. Note that the number might be multiple characters. The number indicates the process making the request, and the character is the resource requested. The first event line in the example indicates that process 0 has requested a resource A.
- **Relinquish:** These begin with a `-`, but otherwise look like requests. In this event, a process is giving a resource back. It will always succeed (unless the process has already terminated). In the 6th event above, process 1 is relinquishing a resource A.
- **Termination:** These begin with a `!`, and they indicate that the given process has finished. All remaining resources that it held are relinquished. Any further requests from it should be ignored.

Every event that is not ignored will result in at least one line being printed to the screen. Upon receiving a request, your banker will print out the request and how it was handled, all in one line. The response will be in parentheses, as follows:

- If the request is valid and there are enough resources to grant it according to the banker's algorithm, do so. Finish the output line by printing `(GRANTED)` at the end of the line.
- If the request is valid but there aren't enough resources to grant it, the process will be frozen. Finish the output line with `(FROZEN)`, and place the event on a queue. Further events involving that process will also be placed on the queue until it is unfrozen.
- If the request is invalid, because it has exceeded its declared needs, terminate the process immediately (reclaiming all its resources and ignoring it from now on). Finish the output line with `(INVALID: PROCESS TERMINATED)`.

When a resource is relinquished or several are relinquished (possibly due to termination), the event is printed as shown in the example. Then, the queue of frozen processes will be checked to see if one or more processes can be unfrozen and their requests can accomodated. Each event should be checked in strict first-come-first-served order. If a change occurs in any of the frozen processes, start searching the queue again for another satisfiable event. Only stop when you have gotten through the queue with no changes. Note that each process must have its events run in strict order, so if a frozen process's prior event cannot occur, nor can any further ones. If a request is granted, print out that the process was `UNFROZEN`, followed by the granted request as normal.

Here is the expected output from the above sample file:

```
Process 0 requested a Resource A. (GRANTED)
Process 0 requested a Resource A. (GRANTED)
Process 1 requested a Resource B. (FROZEN)
Process 0 requested a Resource B. (GRANTED)
Process 0 requested a Resource C. (INVALID: PROCESS TERMINATED)
Process 1 was UNFROZEN: Process 1 requested a Resource B. (GRANTED)
Process 1 requested a Resource A (GRANTED).
Process 1 relinquished a Resource A.
Process 1 terminated.
```