Problem 2: iFlop

A major manufacturer has contracted you to implement a simulator for their new preemptive multitasking system iFlop. A run of iFlop is described by a configuration file which contains lines separated by newline; each line contains positive integer numbers and case-sensitive words separated by white space.

A configuration file starts with the following four lines, in order:

```
slice number1
ready name2 name3 ...
wait name4 name5 ...
var name6 number7
```

where *number1* is the (positive) number of instructions started within a time slice; *name2* etc. specify the user programs which are to be run, each as a separate process; *name4* etc. are the names of waiting lists for processes; and *name6* is the name and *number7* is the initial value of a global integer variable. There is at least one process and one waiting list and `var` lines may be repeated.

All waiting list and variable names are different.
Following the variables, the configuration file can specify some subprograms, for example:

```
sub Pfull
0 if f 1 3
1 dec f
2 if TRUE 4 4
3 move ready full
4 end
```

where `sub` starts a subprogram and defines its name; the program ends with `end`. All subprogram names are different.

Within each subprogram, lines are numbered consecutively from `0`.

`if` specifies a condition and two existing line numbers. Execution continues with one of these two lines depending on whether the condition is true or false. A condition is either the name of a variable and is true if the variable is not zero, or it is the name of a list (including `ready`) and is true if the list is not empty.

`dec` decrements a variable by one; there is also an `inc` statement to increment a variable by one.

`move` specifies two lists (either one can be `ready`, otherwise names defined with `wait` are used) and takes the first process from the first list (which will be not empty) and appends it as the last process on the second list.

Finally, the configuration file specifies each user program, for example:

```
prog consumer
```

```
0 call Pfull
1 print n
2 call Vempty
3 end
```

where `prog` starts a user program and specifies one of the names on the `ready` line; the program ends with `end`. All program names are different.

Within each program, lines are numbered consecutively from `0`.

`print` outputs the value of a variable to standard output.

`call` specifies the name of an existing subprogram to invoke. `call` cannot be used in a subprogram.

For the purposes of `slice`, each line (even `call`) counts as one unit. The program above requires four time units to complete because `call` counts as one unit irrespective of how many statements the subprogram contains.

Your simulator reads a configuration file from standard input and executes the programs for the
processes specified with `ready`, one at a time beginning with the first process on the `ready` list.

If a process executes the number of lines specified with `slice` without executing `call` or `end` it is moved from the beginning to the end of the `ready` list and execution continues with the process which is then first on the `ready` list.

After a process executes a `call`, execution continues with the process which is then first on the `ready` list.

If a process reaches the matching `end` in its program the process is removed from the system.

Your simulator ends once the `ready` list is empty. At this point, for each non-empty list you must print one line with the name of the list and the name of the program of each process on the list.

Sample Input see `~/2.1`

```
slice 10
ready consumer producer
wait full empty
var n 0
var f 0
var e 1
var TRUE 1
sub Pfull
0 if f 1 3
1 dec f
2 if TRUE 4 4
```

```
3 move ready full
4 end
sub Vfull
0 if full 1 3
1 move full ready
2 if TRUE 4 4
3 inc f
4 end
sub Pempty
0 if e 1 3
1 dec e
2 if TRUE 4 4
3 move ready empty
4 end
sub Vempty
0 if empty 1 3
1 move empty ready
2 if TRUE 4 4
3 inc e
4 end
prog consumer
0 call Pfull
1 print n
2 call Vempty
3 end
prog producer
0 call Pempty
1 inc n
2 call Vfull
3 end
```

Sample Output

```
1
```

Sample Input see `~/2.2`

```
slice 1
ready bad good bad good
wait blocked
sub block
0 move ready blocked
1 end
prog bad
0 call block
1 end
prog good
0 end
```

Sample Output

```
blocked bad bad
```