# Problem 4: Zombies!

In UNIX®-like operating systems, the *fork* system call creates a new process (an independent executable object) that is uniquely identified by an integer "process ID". This new process is a "child" of its parent process (the process that executed the *fork* system call). Of course, a child process could itself execute *fork* and create its own children; there is no conceptual limit to the number of generations of processes that may exist, nor is there a limit on the number of child processes a parent may have (at least for this problem).

Another system call, *wait*[1], allows a parent process to delay its execution (if necessary) until it receives notice that one of its child processes has terminated. If a child process terminates before its parent calls *wait*, then the child becomes a so-called *zombie* process – it continues to exist but does not actually execute; if the parent later calls *wait* and at least one of its children has become a zombie, then the parent continues execution and the child process becomes completely terminated. If the parent should terminate before waiting for the termination of one or more of its children, then those remaining children become "wards of the state" – they are effectively adopted by a system process that spends it life repeatedly executing the *wait* system call, efficiently "reaping" terminated child processes that were abandoned by their parents. If a parent calls *wait* and there are no child processes, the *wait* call is effectively ignored and the parent continues execution. In this problem, a process terminates (or perhaps becomes a zombie) by executing the *exit* system call. What we seek to find in this problem is the state of all processes after a sequence of fork, exit, and wait system calls has been executed.

We assume that there is initially only a single process, with a process ID of 1, running at the beginning of each case. It's "parent" is the operating system, so when it exits, it is immediately "reaped" by the system.

When a process forks, the child process it creates is assigned the smallest positive process ID that has never been used. Thus the first process created in each input case will have ID 2.

## Input

There will be multiple input cases to consider. For each case there will be one or more lines of input. Each line contains a positive integer giving a process ID (which will be no larger than 9999), a space, one of the words "fork", "wait", or "exit" (always using lower-case letters), and the end of line. A line containing only 0 (zero) and the end of line follows the last input line for each case. A line with the same content (0 and end of line) immediately follows the input for the last case.

There will be no input in which a process "exits" after a "wait" if the wait did not complete (either successfully or unsuccessfully). That is, there will be no illogical sequences of actions.

In those cases where all processes terminate, the end of input immediately follows the exit which terminates the last process.

## Output

For each case, display the case number (1, 2, …), and for each state (running, zombie, and waiting) the number of processes in the state, the state name, a colon, and the process IDs (in ascending order) of processes in that state, after all input actions for the case have been taken. If no

[1] In real systems there is a family of wait-like system calls; we consider only a single simple wait system call in this problem.

processes are left running, then state that explicitly. Display a blank line after the output for each case. Your output should be essentially identical to that shown in the sample.

**Sample Input**

```
1 exit
0
1 fork
2 exit
0
1 fork
2 exit
1 wait
0
1 fork
1 wait
2 exit
0
1 fork
2 fork
2 fork
2 exit
4 exit
0
0
```

**Output for the Sample Input**

```
Case 1:
   No processes.

Case 2:
   1 Running: 1
   1 Zombie: 2
   0 Waiting:

Case 3:
   1 Running: 1
   0 Zombie:
   0 Waiting:

Case 4:
   1 Running: 1
   0 Zombie:
   0 Waiting:

Case 5:
   2 Running: 1, 3
   1 Zombie: 2
   0 Waiting:
```