Simpler Memory Manager

You are given a task to design a memory manager for a new multitasking OS to support the following features:

1. A process may request N blocks of memory at any point that it is alive
2. A process may free previously allocated blocks of memory at any point in its lifetime
3. A process may request N contiguous blocks of memory. Fail if unavailable.
4. A process may **not** request more than a fraction of the total memory: ***25%***

Input Format:

```
<total-block-count>
<command-1> <process> <args>
:
<command-N> <process> <args>
```

Keep in mind that your inputs can vary in size - from 1 to N, N <= 10^6

The available commands & their formats are described below:

1. **allocate**
   allocate <process> <variable> <blocks-requested>
2. **free**
   free <process> <variable>
3. **kill**
   kill <process>
4. **inspect**
   inspect <process>

Output Format:
<command-result> <allocated-space-block-count> / <free-space-block-count>

Example:
Input is in Regular styled font. Output is in the bold-italicized font.

```
100
allocate P1 var_w 1000
error 0 / 100
allocate P1 var_w 10
success 10 / 90
allocate P1 var_x 50
error 10 / 90
allocate P2 var_y 25
success 35 / 65
```

```
free P1 var_x
success 25 / 75
kill P2
success 0 / 100
allocate P1 var_z 10
success 10 / 90
allocate P4 var_x 5
success 15 / 85
allocate P1 var_w 5
success 20 / 80
free P4 var_x
success 15 / 85
allocate P1 var_y 6
success 21 / 79
inspect P1
var_z 0-9
var_w 15-19
var_y 20-25
```

Expectations

1. **Code should be demo-able (very important).**
2. **Code should be functionally correct and complete.**
    a. At the end of this interview round, an interviewer will provide multiple inputs to your program for which it is expected to work
3. Code should be readable, modular, testable and use proper naming conventions. It should be easy to add/remove functionality without rewriting entire codebase.
4. Create the sample data yourself. You can put it into a file, test case or **main driver program** itself.
5. Code should handle edge cases properly and fail gracefully. Add suitable exception handling, wherever applicable.
6. Avoid writing monolithic code.

Guidelines

1. Please discuss the solution with an interviewer
2. Input can be read from a file or STDIN or coded in a driver method.
3. Output can be written to a file or STDOUT.
4. Feel free to store all interim/output data in-memory.
5. Restrict internet usage to looking up syntax
6. You are free to use the language of your choice.

7. Save your code/project by your name and e-mail it to aurobindo.m@flipkart.com. Your program will be executed on another machine. So, explicitly specify dependencies, if any, in your e-mail.
8. Do not use console input for test cases, everything can go through driver class.