掉下小石頭的里程計

李奧納多發明了第一部里程計: 那是一台車，利用輪子轉動時掉下小石頭來測量距離。只要算小石頭的個數就可得知輪子的轉動次數，進而得知走過的距離。身為電腦科學家，我們在里程計加入軟體的控制，已擴充它的功能。你的任務就是依下列的規則設計里程計的程式。

運作網格範圍

里程計在一個假想的256 x 256格子的正方形區塊上移動。每個格子最多可容納15粒小石頭，且以 (橫列, 直行) 座標標示，座標值範圍是0, ..., 255。格子 (i, j) 相鄰的格子(如果存在的話)是 (i-1, j), (i+1, j), (i, j-1) 和 (i, j+1)。任何落在第一橫列或最後橫列，第一直行或最後直行的格子，稱之為"邊界"。里程計一定自 (0, 0) 格子(西北角)起動，面朝北。

"基本指令"

里程計程式可使用下列指令

- `left` — 向左轉90度 (逆時鐘方向) 且留在目前格子 (例如: 如果本來朝南，指令完成後將朝東)。
- `right` — 向右轉90度 (順時鐘方向) 且留在目前格子 (例如: 如果本來朝西，指令完成後將朝北)。
- `move` — 朝里程計目前的方向，向前移動一格到相鄰的格子。如果無此格子存在 (也就是已在該方向的邊界)，則此指令沒有作用。
- `get` — 從目前格子移除一個小石頭。如果目前格子沒有小石頭，則此指令沒有作用。
- `put` — 從目前格子增加一個小石頭。如果目前格子已有15顆小石頭，則此指令沒有作用。里程計的小石頭決不會用光。
- `halt` — 終止執行。

里程計依程式中的順序執行指令，一個指令一行，空白行可忽略，`#` 表示註解; 任何後續的文字，直到該行結尾，都可忽略。如果里程計到達程式結尾，則執行終止。

"範例 1"

考慮下列里程計程式。它把里程計帶到(0, 2)，面朝東。(注意第一個 `move` 被忽略，因為里程計位在西北角，面朝北方)。

```
move  # 沒有效果
right
# 現在里程計面朝東。
move
move
```

"標籤、邊界、和小石頭"

使用標籤可以依目前狀態改變程式的流程。標籤是一個包含至多128個分大小寫符號的字串，符號可包含 `a`, …, `z`, `A`, …, `Z`, `0`, …, `9`. 和標籤有關的新指令如下所列。在下列敘述中， *L* 表示任意有效的標籤。

- *L*`:` (也就是 *L* 後面接著一個冒號‘:’) — 宣告標籤*L* 在程式中的位置。所有的標籤必須是獨一無二的。宣告標籤對里程計沒有效果。
- `jump` *L* — 無條件跳到標籤*L*的那一行繼續執行。
- `border` *L* — 如果里程計位於邊界，且面向網格邊緣(也就是`move` 指令將會沒有效果)，則跳到標籤*L*的那一行繼續執行; 否則程式繼續正常執行，此指令沒有效果。
- `pebble` *L* — 如果目前的格子含有至少一顆小石頭，則跳到標籤*L*的那一行繼續執行; 否則程式繼續正常執行，此指令沒有效果。

"範例 2"

下列程式找到橫列0的第一顆小石頭，並停在該處; 如果橫列0沒有小石頭，則停在橫列末尾的邊界。它使用 `leonardo` 和 `davinci` 兩個標籤。

```
right
leonardo:
pebble davinci  # 找到小石頭
border davinci  # 橫列末端
move
jump leonardo
davinci:
halt
```

里程計開始時先向右轉。迴圈由宣告 `leonardo:` 標籤處開始，在 `jump leonardo` 指令處結束。在迴圈中，里程計檢查小石頭是否存在，或者是否在橫列末端的邊界; 若不是，里程計就透過 `move` 從格子 (0, j) 移動到相鄰的格子 (0, j + 1)，因為後者確實存在。(`halt` 指令在此不一定需要，因為程式無論如何都會終止)

聲明

你必須繳交如上面描述的里程計語言的程式，使里程計按照預期的行動。 每個子任務(如下)規定了一項里程計必須達成的行動，以及所繳交的解答 必須滿足的限制。限制和下面兩點有關:

- "程式大小" — 程式要夠短。程式大小是裡面指令的數目。標籤、註解、和空白行*不算在* 程式大小裡。
- "執行長度" — 程式必須終止得夠快。執行長度是執行*步驟'的數目': 每個單一指令的執行算做一個步驟，無論指令是否有效果; 標籤、註解、和空白行不算步驟。*

範例一的程式大小為4，執行長度為4。範例二的程式大小為6，且在只有一顆小石頭在格子(0, 10)的網格執行時,執行長度為43: `right`, 迴圈經過十次，每次4個步驟 (`pebble davinci; border davinci; move; jump leonardo`)，和最後的`pebble davinci`與`halt`。

子任務 1 [9 分]

一開始有 x 個小石頭在格子 (0, 0) 和 y 個小石頭在格子 (0,1) 而其他的格子都是空的。請記得任何一個格子裡面最多只能有15個小石頭。請寫一個程式讓里程計最後停在 (0, 0) 如果 $x \leq y$。如果是其他的狀況則停在 (0, 1)。(我們不管最後里程計朝向哪一個方向; 我們也不管最後網格裡面有多小顆小石頭或者是這些小石頭落在何處)。

Limits: 程式大小 ≤ 100, 執行長度 ≤ 1

子任務 2 [12 分]

與前一個子任務相同，但是當程式執行結束，格子(0,0) 必須包含 x 個小石頭而且 (0,1) 必須包含 y 顆小石頭。

Limits: 程式大小 ≤ 200, 執行長度 ≤ 2

子任務 3 [19 分]

在橫列 0 (row 0) 有兩顆小石頭; 一顆在格子 (0, x) 另外一顆在格子 (0, y); x 和 y 不相等而且 x+y 是雙數。 請寫一個程式讓里程計停在格子 (0, (x + y) / 2), 也就是在兩個含有小石頭的格子的中點。網格最後的狀態無關緊要。

Limits: 程式大小 ≤ 200, 執行長度 ≤ 2

子任務 4 [至多 32 分]

在網格當中最多只能有15顆小石頭，而且沒有兩顆小石頭落在同一個格子。請寫一個程式將小石頭全部蒐集在西北角; 更精確一點來說，如果一開始網格中有 x 顆小石頭，在結束的時候，格子(0, 0) 必須要有x 顆小石頭，而其他格子都不能有小石頭。

這個子任務的得分將依所繳交程式的執行長度而定。更確實來說，如果 L 是在各種不同測試下的最大執行長度，你的得分將是:

- 32 分 if $L \leq 200\,000$;
- $32 - 32 \log_{10}(L / 200\,000)$ 分 if $200\,000 < L < 2\,000\,000$;

- 0分 if $L \geq 2\ 000\ 000$.

限制: 程式大小 ≤ 200.

子任務 5 [至多 28 分]

每一格可以有任意數目的小石頭(當然仍在 0 與 15 之間)。寫個找出最小值的程式，也就是在格子(i, j)終止，使得其它的格子都有至少和格子(i, j)一樣多顆小石頭。程式執行後，每個格子裡的小石頭數目必須和執行前相同。

這個子任務的得分將依所繳交的程式大小而定。更確實來說，得分將是:

- 28 分 if $P \leq 444$;
- $28 - 28 \log_{10}(P / 444)$ 分 if $444 < P < 4\ 440$;
- 0 分 if $P \geq 4\ 440$.

限制: 執行長度 $\leq 44\ 400\ 000$.

實作細節

每個子任務必須繳交剛好一個檔案，根據上述語法規則撰寫。每個繳交的檔案最大5 MiB。於每個子任務，你的里程計程式碼將會用多筆測試資料測試， 而你會收到你的程式碼用到哪些資源的回饋。如果程式碼因語法錯誤而無法執行時，你將收到特定語法錯誤的資訊。

你所繳交的里程計程式可以不必包含全部的子任務。如果你這次的繳交未包含子任務 X 的里程計程式，則你最近一次的子任務 X 會自動列入; 如果沒有這個程式，則此子任務得分為零。

一如往常，每次繳交的分數是每項子任務得分的總和。本題最後總分是已知繳交的分數和最後一次分數的最大值。

模擬器

我們提供了一個以測試為目的的里程計模擬器，讓你可以用來測試你的程式以及輸入網格。里程計程式將使用相同的格式進行撰寫(也就是如上所描述的)。

網格的描述使用以下的格式: 檔案中的每一行必須包含三個數，R, C, 以及P，代表在第 R 列 C 行的格子含有 P 個小石頭。沒有被描述到的格子代表該格子沒有任何小石頭。例如，請參考下面的檔案:

```
0 10 3
4 5 12
```

這個檔案描述的網格包含15顆小石頭: 3個在格子 (0, 10)，12顆在格子(4, 5)。

在你的工作目錄，你可以藉用呼叫`simulator.py`啟動測試模擬器，將程式檔案名稱當成參數。這個模擬器可以接受下列的命令行選項:

- `-h` 簡明概述可能的選項;
- `-g GRID_FILE` 從檔案載入網格的描述 `GRID_FILE` (內定: 空網格);
- `-s GRID_SIDE` 設定網格的大小為 `GRID_SIDE x GRID_SIDE` (內定值: 256, 如同之前問題的描述); 使用小一點的網格可以對你的程式除錯有幫助。
- `-m STEPS` 限制模擬器的執行步驟數量到最多 `STEPS` 步;
- `-c` 指定編譯模式。在編譯模式中，模擬器會回傳一樣的輸出結果，但是不會利用Python進行模擬，反而產生並編譯成一個小的C程式。這在程式開始時會產生大量的負擔，但之後會顯著的得到較快速的結果。如果你的程式執行超過10 000 000步驟時，建議你使用這選項。

繳交上傳次數

本題允許繳交上傳的最多次數為128.