Stein-Odometer (Kilometerzähler)

Leonardo erfand das erste *Odometer* ("Kilometerzähler"). Dieser Wagen konnte Entfernungen messen, indem er mit den Drehungen seiner Räder Steine (*engl.:* pebbles) fallen ließ. Aus der Anzahl der Steine ergab sich die Anzahl der Drehungen und damit auch die vom Odometer zurückgelegte Entfernung. Nun haben Informatiker die Funktion des Odometers durch eine Software-Steuerung erweitert. Du sollst das Odometer programmieren, gemäß der im Folgenden angegebenen Regeln.

Arbeitsraster

Das Odometer bewegt sich auf einem Raster mit 256 × 256 Feldern. Auf jedem Feld können höchstens 15 Steine liegen. Ein Feld wird durch Koordinaten (Zeile, Spalte) identifiziert, wobei die Koordinaten jeweils im Bereich 0, …, 255 liegen. Ein Feld (i, j) hat folgende benachbarten Zellen (falls sie existieren): (i - 1, j), (i + 1, j), (i, j - 1) und (i, j + 1). Alle Felder in der ersten oder letzten Zeile bzw. in der ersten oder letzten Spalte heißen *Randfelder*. Zu Beginn steht das Odometer immer auf Feld (0, 0) (die nordwestliche Ecke) und zeigt nach Norden.

Basis-Befehle

Das Odometer kann mit Hilfe der folgenden Befehle programmiert werden:

- `left` — dreht das Odometer um 90 Grad nach links (entgegen dem Uhrzeigersinn), belässt das Odometer jedoch im selben Feld (falls es beispielsweise vorher nach Süden zeigte, zeigt es danach in Richtung Osten).
- `right` — dreht das Odometer um 90 Grad nach rechts (im Uhrzeigersinn), belässt das Odometer jedoch im selben Feld (falls es beispielsweise vor diesem Befehl nach Westen zeigte, zeigt es danach in Richtung Norden).
- `move` — bewegt das Odometer ein Feld nach vorn (in die Richtung, in die es im Moment zeigt). Falls es kein solches Feld gibt (also falls der Rand des Rasters bereits erreicht ist), hat der Befehl keine Auswirkungen.
- `get` — entfernt einen Stein vom aktuellen Feld. Falls kein Stein auf dem aktuellen Feld liegt, so hat der Befehl keine Auswirkungen.
- `put` — platziert einen (zusätzlichen) Stein auf dem aktuellen Feld. Falls bereits 15 Steine auf dem aktuellen Feld liegen, hat der Befehl keine Auswirkungen. Der Steinvorrat des Odometers ist unerschöpflich.

- `halt` — beendet die Ausführung des Programms.

Das Odometer führt ein gegebenes Programm Zeile für Zeile aus; in jeder Zeile darf nur eine Anweisung stehen. Leerzeilen werden ignoriert. Das Symbol # kennzeichnet den Beginn eines Kommentars; jeglicher folgender Text in der selben Zeile wird bei der Programmausführung ignoriert. Wenn das Odometer das Ende des Programms erreicht, wird die Ausführung beendet.

Beispiel 1

Wir betrachten das folgende Programm für das Odometer. Das Programm bewegt das Odometer nach (0, 2), wobei es am Ende nach Osten zeigt. (Man beachte, dass die erste `move`-Anweisung ignoriert wird, da das Odometer sich in der nordwestlichen Ecke befindet und nach Norden blickt.)

```
move  # Kein Effekt.
right
# Jetzt zeigt das Odometer nach Osten.
move
move
```

Kontrollfluss mit Labels

Um die Programmausführung zu kontrollieren, kannst du Labels benutzen. Das sind Strings (case-sensitiv) aus höchstens 128 der folgenden Zeichen: `a`, …, `z`, `A`, …, `Z`, `0`, …, `9`. Hier nun die Befehle, die Labels verwenden; *L* bezeichnet ein beliebiges gültiges Label.

L`:` (d.h. *L* gefolgt von einem Doppelpunkt '`:`') — bezeichnet eine Programmstelle mit dem Label *L*. Labels müssen eindeutig sein. Eine Programmzeile mit einem Label hat keine Auswirkung auf das Odometer.

- `jump` *L* — springe zur Zeile mit Label *L* und fahre dort mit der Ausführung des Programms fort.
- `border` *L* — falls das Odometer auf einem Randfeld steht und nach außen zeigt (d.h., ein `move` Befehl hätte keine Auswirkung): springe zur Zeile mit Label *L* und fahre dort mit der Ausführung des Programms fort; andernfalls hat dieser Befehl keine Auswirkung.
- `pebble` *L* — falls auf dem aktuellen Feld mindestens ein Stein liegt: springe zur Zeile mit Label *L* und fahre dort mit der Ausführung des Programms fort; andernfalls hat dieser Befehl keine Auswirkung.

Beispiel 2

Das folgende Programm lässt das Odometer den ersten (westlichsten) Stein in Zeile 0 finden und dann anhalten; falls in Zeile 0 kein Stein liegt, hält das Odometer am Ende der Zeile an. Das Programm enthält die Labels `leonardo` und `davinci`.

```
right
leonardo:
pebble davinci  # Stein gefunden
border davinci  # Ende der Zeile
move
jump leonardo
davinci:
halt
```

Das Odometer dreht sich zuerst nach rechts. Die Schleife beginnt mit dem Label `leonardo:` und endet mit dem Befehl `jump leonardo`. In der Schleife prüft das Odometer, ob ein Feld mit einem Stein oder ein Randfeld am Ende der Zeile erreicht wurde; falls nicht, bewegt sich das Odometer von Feld (0, j) zum benachbarten Feld (0, j + 1), welches existiert. (Der Befehl `halt` ist nicht zwingend nötig, da die Ausführung des Programms ohnehin beendet wird.)

Aufgabe

Du sollst mehrere Programme in der Odometer-Programmiersprache einsenden, die jeweils dafür sorgen, dass sich das Odometer wie gewünscht verhält. Für jede Teilaufgabe (siehe unten) ist ein gewünschtes Verhalten angegeben und die Beschränkungen, die das Programm erfüllen muss. Die Beschränkungen betreffen die beiden folgenden Größen:

- *Programmlänge* — das Programm muss kurz genug sein. Die Programmlänge entspricht der Anzahl der Anweisungen. Labels, Kommentare und Leerzeilen werden nicht mitgezählt.
- *Ausführungslänge* — das Programm muss schnell genug beendet werden. Die Ausführungslänge ist die Anzahl der ausgeführten Anweisungen: Jede Befehlsausführung zählt als Anweisung, unabhängig davon, ob der Befehl eine Auswirkung hatte. Labels, Kommentare und Leerzeilen zählen nicht als Anweisung.

Bei Beispiel 1 ist die Programmlänge 4 und die Ausführungslänge 4. Bei Beispiel 2 ist die Programmlänge 6 und die Ausführungslänge 43, wenn es auf einem Raster mit einem einzelnen Stein im Feld (0, 10) ausgeführt wird: Die ausgeführten Anweisungen sind in diesem Fall `right`, darauf 10 Wiederholungen der Schleife, wobei jede Wiederholung die Anweisungen (`pebble davinci`; `border davinci`; `move`; `jump leonardo`) ausführt, und schließlich `pebble davinci` sowie `halt`.

Teilaufgabe 1 [9 Punkte]

Am Anfang sind x Steine auf dem Feld (0, 0) und y auf dem Feld (0, 1) und alle übrigen Felder sind leer. Schreib ein Programm, welches mit dem Odometer auf Feld (0, 0) terminiert, falls $x \leq y$, und andernfalls auf Feld (0, 1). (Es spielt keine Rolle in welche Richtung das Odometer am Ende ausgerichtet ist. Es spielt auch keine Rolle wie viele Steine am Ende auf dem Raster liegen und wo sie sich befinden.)

Beschränkungen: Programmlänge ≤ 100, Ausführungslänge ≤ 1 000

Teilaufgabe 2 [12 Punkte]

Gleiche Aufgabe wie vorhin, aber nachdem das Programm ausgeführt wurde muss das Feld (0, 0) genau x Steine und das Feld (0, 1) genau y Steine enthalten.

Beschränkungen: Programmlänge ≤ 200, Ausführungslänge ≤ 2 000

Teilaufgabe 3 [19 Punkte]

Es gibt genau zwei Steine irgendwo in Zeile 0: Ein Stein befindet sich auf dem Feld (0, x), der andere auf dem Feld (0, y). x und y sind verschieden und x + y ist gerade. Schreibe ein Programm, welches das Odometer auf dem Feld (0, (x + y)/2) stehen lässt, d.h. genau im Mittelpunkt der beiden Felder, welche Steine enthalten. Der Endzustand des Rasters ist nicht relevant.

Beschränkungen: Programmlänge ≤ 100, Ausführunglänge ≤ 200 000.

Teilaufgabe 4 [bis zu 32 Punkte]

Es gibt maximal 15 Steine auf dem Raster. Keine zwei dieser Steine befinden sich im selben Feld. Schreibe ein Programm, welches alle diese Steine im nordwestlichen Eckfeld sammelt. Genauer gesagt, falls es am Anfang x Steine auf dem Raster gibt, müssen am Ende genau x Steine auf dem Feld (0, 0) liegen, und es dürfen keine Steine sonstwo sein.

Die erreichte Punktzahl für diese Teilaufgabe hängt von der Ausführungslänge des eingesendeten Programms ab. Genauer gesagt, falls L das Maximum der Ausführungslänge aller Testfälle ist, wird deine Punktzahl die folgende sein:

- 32 Punkte, falls $L \leq 200\,000$;
- $32 - 32 \log_{10}(L / 200\,000)$ Punkte, falls $200\,000 < L < 2\,000\,000$;
- 0 Punkte, falls $L \geq 2\,000\,000$.

Beschränkungen: Programmlänge ≤ 200.

Teilaufgabe 5 [bis zu 28 Punkte]

Jedes Feld des Rasters kann eine beliebige Anzahl Steine enthalten (natürlich zwischen 0 und 15). Schreibe ein Programm, welches das Minimum findet, d.h. welches das Odometer auf dem Feld (i, j) stehen lässt, sodass jedes andere Feld mindestens so viele Steine wie (i, j) enthält. Nach der Ausführung des Programms muss die Anzahl Steine auf jedem Feld dieselbe sein wie vor der Ausführung des Programms.

Die erreichte Punktzahl für diese Teilaufgabe hängt von der Programmlänge P des eingesendeten Programms ab. Genauer gesagt, deine Punktzahl wird die folgende sein:

- 28 Punkte, falls P ≤ 444;

- 28 - 28 $\log_{10}$ (P / 444) Punkte, falls 444 < P < 4 440;
- 0 Punkte, falls P ≥ 4 440.

Beschränkungen: Ausführungslänge ≤ 44 400 000.

Details zur Implementierung

Du musst genau eine Datei pro Teilaufgabe einsenden, welche den obigen Syntaxregeln genügt. Jede eingesendete Datei darf maximal 5 MiB gross sein. Für jede Teilaufgabe wird dein Odometer-Programm auf einigen Testfällen ausgeführt. Du bekommst Feedback zu den Ressourcen, die dein Programm benötigt. Falls dein Programm syntaktisch nicht korrekt ist und somit nicht getestet werden konnte, bekommst du Informationen zum spezifischen Syntaxfehler.

Es ist nicht nötig, dass deine Einsendungen Odometer-Programme für alle Teilaufgaben enthalten. Falls deine aktuelle Einsendung kein Odometer-Programm für eine Teilaufgabe X enthält, wird deine aktuellste Einsendung zu Teilaufgabe X automatisch hinzugefügt. Falls es kein solches Programm gibt, erhält du für diese Teilaufgabe keine Punkte.

Wie gewöhnlich berechnet sich die Punktzahl einer Einsendung als Summe aller Punktzahlen für die einzelnen Teilaufgaben. Schliesslich ist die endgültige Punktzahl der gesamten Aufgabe das Maximum aller Punktzahlen der release-getesten Einsendungen und der letzten Einsendung.

Simulator

Zu Testzwecken bekommst du einen Odometer-Simulator, welchem du deine Programme und Eingaberaster übergeben kannst. Odometer-Programme für den Simulator musst du im selben Format schreiben wie Programme für Einsendungen (d.h. im Format, welches oben beschrieben wurde).

Rasterbeschreibungen können im folgenden Format angegeben werden: Jede Zeile der Datei enthält drei Zahlen, R, C und P, mit der Bedeutung, dass das Feld in Zeile R und Spalte C genau P Steine enthält. Alle Felder, die nicht in der Rasterbeschreibung erwähnt werden, enthalten keine Steine. Betrachte zum Beispiel diese Datei:

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

Diese Datei beschreibt ein Raster mit 15 Steinen: 3 in Feld (0,10) und 12 in Feld (4,5).

In deinem Aufgaben-Ordner gibt es ein Programm `simulator.py`. Rufe es mit dem Namen deiner Programmdatei als Argument auf, um den Testsimulator zu starten. Das Simulator-Programm versteht folgende Kommandozeilen-Optionen:

- `-h` gibt einen kurzen Überblick über die verfügbaren Optionen.
- `-g GRID_FILE` lädt die Rasterbeschreibung aus der Datei `GRID_FILE` (standardmässig ein leeres Raster).

- `-s GRID_SIDE` setzt die Rastergrösse auf `GRID_SIDE x GRID_SIDE` (standardmässig: 256, wie in der Aufgabenbeschreibung). Die Benutzung eines kleineren Rasters kann dir beim Debuggen helfen.

- `-m STEPS` beschränkt die Anzahl Ausführungsschritte der Simulation auf maximal `STEPS` Schritte.

- `-c` aktiviert den Compilation-Modus. Dabei ist die Ausgabe der Simulation dieselbe, aber anstelle der Python-Simulation wird ein kleines C-Programm generiert und kompiliert. Das bedeutet einen grossen Overhead zu Beginn, aber erzeugt deutlich schneller Resultate. Es wird empfohlen, den Compilation-Modus zu verwenden, wenn dein Programm voraussichtlich mehr als 10 000 000 Schritte benötigen wird.

Anzahl Einsendungen

Es sind maximal 128 Einsendungen für diese Aufgabe erlaubt.