Kiezelmeter

Leonardo heeft de eerste *kilometerteller* uitgevonden: een wagentje dat afstanden kan bepalen door kiezeltjes te laten vallen als de wielen van de wagen draaien. Uit het aantal kiezels volgt het aantal omwentelingen van het wiel, waardoor je de afgelegde afstand van het wagentje kunt bepalen. Als informatici hebben we software toegevoegd aan deze *kiezelmeter* die de functionaliteit uitbreidt. Het is jouw taak om de kiezelmeter te programmeren volgens de volgende regels.

het rooster waarop je beweegt

De kiezelmeter beweegt op een denkbeeldig vierkant rooster van 256 x 256 cellen. Elke cel kan maximaal 15 kiezels bevatten. Elke cel wordt aangeduid door een paar coördinaten (rij, kolom), waarbij elke coördinaat een waarde tussen 0 en 255 (inclusief) heeft. Gegeven een cel (i, j), dan zijn de aangrenzende cellen (als deze bestaan): (i - 1, j), (i + 1, j), (i, j - 1) en (i, j + 1). Elke cel die op de eerste of laatste rij, of de eerste of laatste kolom ligt heet een *grens*. De kiezelmeter begint altijd in cel (0, 0) (de noord-westelijke hoek), en is gericht naar het noorden.

Basis-instructies

De kiezelmeter kan geprogrammeerd worden met de volgende instructies:

`left` — draai 90 graden naar links (tegen de klok) en blijf in de huidige cel (bvb. als je naar het zuiden keek, dan kijk je na deze instructie naar het oosten).

`right` — draai 90 graden naar rechts (met de klok mee) en blijf in de huidige cel (bvb. als je naar het westen keek, dan kijk je na deze instructie naar het noorden).

`move` — beweeg vooruit naar de naastgelegen cel (in de richting waarin je kijkt). Als deze cel niet bestaat (als je de grens in die richting hebt bereikt) dan gebeurt er niets.

`get` — verwijder een kiezel van de huidige cel. Als in de huidige cel geen kiezels liggen, dan heeft deze instructie geen effect.

`put` — plaats een kiezel in de huidige cel. Als de huidige cel al 15 kiezels bevat dan doet deze instructie niets. De kiezelmeter heeft altijd voldoende kiezels.

`halt` — stop het programma.

De kiezelmeter voert de instructies uit in de volgorde waarin ze gegeven zijn in het programma. Het programma mag maximaal één instructie per regel bevatten. Lege regels worden genegeerd. Het symbool `#` geeft commentaar aan; alle tekst die er op volgt tot aan het einde van de regel, wordt genegeerd. Als de kiezelmeter het einde van het programma bereikt, stopt hij vanzelf.

Voorbeeld 1

Beschouw het volgende programma voor de kiezelmeter. Dit programma brengt de kiezelmeter naar de cel (0,2), gericht op het oosten. Merk op dat de eerste `move` geen effect heeft, omdat de kiezelmeter in de noord-westelijke hoek begint en dan op het noorden is gericht.

```
move  # doet niets
right
# Nu is de kiezelmeter op het oosten gericht.
move
move
```

Labels, grenzen en kiezels

Om het verloop van het programma aan te passen afhankelijk van de huidige staat kan je labels gebruiken. Dit zijn hoofdlettergevoelige strings van maximaal 128 tekens gekozen uit de karakters `a`, …, `z`, `A`, …, `Z`, `0`, …, `9`. We voegen de volgende instructies toe betreffende labels. In deze instructies duidt *`L`* een geldig label aan.

- *`L`*`:` (dus *`L`* gevolgd door een dubbelpunt '`:`') — de declaratie van de locatie in het programma die het label *`L`* krijgt. Alle gedeclareerde labels moeten uniek zijn. Een label declareren heeft verder geen effect op de kiezelmeter.
- `jump` *`L`* — ga verder met het uitvoeren van instructies door te springen naar de regel met het label *`L`*.
- `border` *`L`* — als je op een grens staat en je kijkt naar de buitenkant van het vierkant (dus, als een `move` instructie niets zou doen) spring je naar de regel met label *`L`*, anders gaat je programma verder op de volgende regel en doet deze instructie niets.
- `pebble` *`L`* — Als de huidige cel minimaal één kiezel bevat, spring dan naar de regel met label *`L`*; anders gaat je programma verder op de volgende regel en doet deze instructie niets.

Voorbeeld 2

Het volgende programma zoekt de eerste (meest westelijke) kiezel in rij 0 en stopt daar; als er geen kiezels in rij 0 zijn dan stopt het programma op de grens aan het einde van die rij. Het programma gebruikt twee labels `leonardo` en `davinci`.

```
right
leonardo:
pebble davinci  # kiezel gevonden
border davinci  # einde van de rij
move
jump leonardo
davinci:
halt
```

De kiezelmeter begint met een draai naar rechts. De lus begint met het declareren van het label `leonardo:` en eindigt met de instructie `jump leonardo`. In de lus controleert de kiezelmeter of er een kiezel aanwezig is of of het einde van de rij is bereikt. Is geen van beide het geval, dan doet de kiezelmeter een `move` van de huidige cel (0, j) naar de aangrenzende cel (0, j + 1). (De instructie `halt` is hier niet echt noodzakelijk omdat je programma toch eindigt.)

Opdracht

Schrijf een programma in de taal van de kiezelmeter zoals hierboven beschreven, dat ervoor zorgt dat de kiezelmeter zich gedraagt zoals verwacht. Elke subtaak (zie verderop) beschrijft specifiek gedrag van de kiezelmeter en de randvoorwaarden waaraan je oplossing moet voldoen. De randvoorwaarden betreffen de volgende twee zaken.

- *Lengte van je programma* — je programma moet kort genoeg zijn. De lengte van je programma is het aantal instructies dat het bevat. Labels, commentaar en lege regels *tellen niet mee* bij het bepalen van deze lengte.
- *Lengte van uitvoering* — je programma moet snel genoeg zijn. De lengte van uitvoering is het aantal *stappen* dat werd uitgevoerd bij het draaien van je programma. Elke uitgevoerde instructie, ongeacht of die enig effect heeft gehad, telt mee als stap. Declaraties van labels, commentaar en lege regels worden niet meegerekend.

In voorbeeld 1 is de lengte van je programma 4 en de lengte van uitvoering ook 4. In voorbeeld 2 is de lengte van je programma 6. Als je dit programma uitvoert op een rooster met een enkele kiezel in cel (0,10) dan is de lengte van uitvoering 43 stappen: `right`, 10 herhalingen van de lus waarbij in elke herhaling 4 instructies worden uitgevoerd (`pebble davinci`; `border davinci`; `move`; `jump leonardo`), en tot slot, `pebble davinci` en `halt`.

Subtaak 1 [9 punten]

In het begin liggen er x kiezels in cel (0, 0) en y in cel (0, 1), en alle andere cellen zijn leeg. Denk eraan dat er hoogstens 15 kiezels in één cel kunnen liggen! Schrijf een programma dat eindigt met de kiezelmeter in cel (0, 0) als $x \leq y$, en anders in cel (0, 1). Het maakt niet uit in welke richting de kiezelmeter gericht is op het einde. Het maakt ook niet uit hoeveel kiezels er aan het einde nog op het rooster liggen, of waar ze liggen.

Limieten: lengte van je programma ≤ 100, lengte van uitvoering ≤ 1 000.

Subtaak 2 [12 punten]

Dezelfde opdracht als hierboven, maar als je programma stopt moet de cel (0, 0) precies x kiezels bevatten, en de cel (0, 1) moet precies y kiezels bevatten.

Limieten: lengte van je programma ≤ 200, lengte van uitvoering ≤ 2 000.

Subtaak 3 [19 punten]

Er zijn precies twee kiezels ergens in rij 0: één in cel (0, x) en de andere in cel (0, y); x en y zijn verschillend en x + y is even. Schrijf een programma dat de kiezelmeter achterlaat in cel (0, (x + y) / 2), dus precies in het midden tussen de beide kiezels. De eindtoestand van het rooster zelf maakt niets uit.

Limieten: lengte van je programma ≤ 100, lengte van uitvoering ≤ 200 000.

Subtaak 4 [maximaal 32 punten]

Er zijn maximaal 15 kiezels in het rooster, en ze liggen allemaal in verschillende cellen. Schrijf een programma dat ze allemaal verzamelt en in de noord-west-hoek neerlegt. Specifiek: als er oorspronkelijk x kiezels in het rooster liggen dan moeten er aan het einde precies x kiezels in cel (0, 0) liggen, en in de andere cellen geen.

De score voor deze subtask hangt af van de lengte van uitvoering van je ingezonden programma. Specifieker, als L het maximum is van alle lengtes van uitvoer bekomen op de verschillende testcases, dan wordt je score:

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

Limieten: lengte van je programma ≤ 200.

Subtaak 5 [maximaal 28 punten]

Er kan eender welk aantal kiezels in elke cel liggen (uiteraard tussen 0 en 15). Schrijf een programma dat het minimum vindt: het eindigt met de kiezelmeter in een cel (i, j) zodat elke andere cel minstens zoveel kiezels bevat als (i, j). Na het uitvoeren van het programma moet het aantal kiezels in elke cel hetzelfde zijn als in het begin.

De score voor deze subtaak hangt af van de lengte P van je ingezonden programma. Specifieker, je score wordt:

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

Limieten: lengte van uitvoering ≤ 44 400 000.

Implementatiedetails

Je moet precies één bestand per subtaak indienen. Dit bestand moet voldoen aan de syntax die hierboven werd beschreven. Elk ingezonden bestand mag maximaal 5MiB groot zijn. Voor elke subtaak wordt de code voor je kiezelmeter getest op een aantal testcases. Je krijgt wat feedback over het verbruik van je programma. Als je code syntaxfouten bevat en dus niet getest kan worden, krijg je informatie over de specifieke syntax error.

Je inzending hoeft niet voor elke subtaak een programma te bevatten. Als je huidige inzending geen programma voor subtaak X bevat, wordt je meest recente inzending voor subtaak X automatisch toegevoegd. Als dat programma (nog) niet bestaat krijg je 0 punten voor die subtaak bij deze inzending.

Zoals gewoonlijk is de score van je inzending de som van de scores voor elke subtaak. Je eindscore is de hoogste score van alle release-geteste inzendingen of je laatste inzending.

Simulator

Om te testen krijg je een kiezelmeter-simulator. Daaraan kan je jouw programma's en roosters als invoer geven. Deze programma's worden geschreven in hetzelfde formaat dat gebruikt wordt voor je inzending (dus zoals hierboven beschreven).

De beschrijving van een rooster wordt in het volgende formaat gegeven: op elke regel van het bestand staan drie getallen: R, C en P. Deze getallen duiden aan dat in de cel op rij R en kolom C precies P kiezels liggen. Cellen die niet genoemd worden in de beschrijving van het rooster bevatten geen kiezels. Beschouw bijvoorbeeld dit bestand:

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

Het rooster beschreven in dit bestand bevat 15 kiezels: 3 in cel (0, 10) en 12 in cel (4, 5).

Je kunt de testsimulator starten door het programma `simulator.py` aan te roepen in de map van je opdracht. Geef de naam van je programmabestand mee als argument. De simulator accepteert de volgende command line opties:

- `-h` geeft een overzicht van de beschikbare opties;
- `-g GRID_FILE` laadt de beschrijving van het rooster uit het bestand `GRID_FILE` (default: leeg rooster);
- `-s GRID_SIDE` stelt de grootte van het rooster in op `GRID_SIDE x GRID_SIDE` (default: 256, zoals vermeld in de opdrachtbeschrijving); het gebruik van kleinere roosters kan handig zijn bij het debuggen.
- `-m STEPS` limiteert het aantal uitgevoerde stappen in de simulatie tot maximaal `STEPS`;
- `-c` start de compilatie-mode. In compilatie-mode krijg je van de simulator exact dezelfde output terug, maar in plaats van de simulatie in Python uit te voeren, wordt er een klein C programma gegenereerd en gecompileerd. Dit kost veel tijd bij het starten, maar wordt veel sneller uitgevoerd. Je wordt geadviseerd deze optie te gebruiken wanneer je verwacht dat je programma meer dan 10 000 000 stappen zal uitvoeren.

Aantal submissies

Je mag maximaal 128 submissies indienen voor deze taak.