International Olympiad in Informatics 2012
23-30 September 2012
Sirmione - Montichiari, Italy
Competition tasks, day 1: Leonardo's inventions and projects

odometer

Japanese — 1.2

小石オドメーター (Pebbling odometer)

レオナルド (Leonardo) は**オドメーター** (odometer) を発明した：オドメーターとは，車輪が回るときに小石を落として距離を測れるカートのようなものである．小石を数えて，車輪の回転数がわかり，オドメーターが動いた距離を計算できる．私たち計算機科学者は，オドメーターにソフトウェアによる制御を加え，機能を大きく拡張した．あなたの課題は，以下のルールに従ってオドメーターをプログラムすることである．

オドメーターが動作するマス目 (Operation grid)

オドメーターは架空の 256 × 256 の正方形状のマス目の上で動く．各々のマスは 15 個以下の小石を含むことができ，(行，列) の座標で表される．それぞれの座標は 0, …, 255 の範囲である．マス (i, j) と隣接するマスは (存在すれば) (i - 1, j), (i + 1, j), (i, j - 1), (i, j + 1) である．最初の行，最後の行，最初の列，最後の列にあるマスは**境界** (border) と呼ばれる．オドメーターは常にマス (0, 0) (北西の隅) で北を向いてスタートする．

基本的なコマンド (Basic commands)

オドメーターは以下のコマンドを使ってプログラムできる．

- `left` — 90 度左 (反時計回り) に回り，今いるマスに残る (例えば，南を向いていたら東を向く).
- `right` — 90 度右 (時計回り) に回り，今いるマスに残る (例えば，西を向いていたら北を向く).
- `move` — 1 マス前方 (オドメーターの向いている方向) に隣接するマスに移動する．そのようなマスが存在しない場合 (その方向の境界に到達していた場合)，何もしない.
- `get` — 今いるマスから小石を 1 個取り除く．小石がない場合，何もしない.
- `put` — 今いるマスに小石を 1 個置く．小石が既に 15 個ある場合，何もしない．置くための小石が足りなくなることはない.
- `halt` — 実行を終了する.

オドメーターはプログラムで与えられた順番でコマンドを実行する．プログラムは各行にコマンドを高々 1 個含む．空行は無視される．記号 # はコメントを表し，それ以降の行末までの文字列は無視される．オドメーターがプログラムの最後に達すると，実行を終了する.

例 1 (Example 1)

以下のオドメーターのプログラムは，オドメーターをマス (0, 2) に動かし，東を向かせる (初めオドメーターは北西の隅で北を向いているので，最初の move は無視されることに注意せよ).

```
move  # 何もしない
right
# 今，オドメーターは東を向いている
move
move
```

ラベル・境界・小石 (Labels, borders and pebbles)

現在の状態によってプログラムの流れを変えるために，ラベルを使うことができる．ラベルは 128 文字以下の a, …, z, A, …, Z, 0, …, 9 からなる文字列である (小文字と大文字は区別される)．ラベルに関係するコマンドの一覧が以下に示されている．以下の説明で，*L* は任意のラベルを表す.

- *L*: (*L* の後にコロン ':') — ラベル *L* のプログラム中での位置を宣言する．すべての宣言されたラベルは異ならなければならない．ラベルの宣言はオドメーターに影響を与えない.
- jump *L* — 無条件にラベル *L* の行にジャンプして実行を続ける.
- border *L* — オドメーターが境界のマスでマス目の端を向いている場合 (すなわち，move をしても何も起こらない場合)，ラベル *L* の行にジャンプして実行を続ける．そうでない場合，このコマンドは何もせず，通常通り実行を続ける.
- pebble *L* — 今いるマスに少なくとも 1 個の小石がある場合，ラベル *L* の行にジャンプして実行を続ける．そうでない場合，このコマンドは何もせず，通常通り実行を続ける.

例 2 (Example 2)

以下のプログラムは，行 0 にある最初の (最も西の) 小石の場所にオドメーターを動かす．行 0 に小石がない場合は，オドメーターは行の端で止まる．このプログラムは 2 つのラベル leonardo と davinci を使っている.

```
right
leonardo:
pebble davinci  # 小石が見つかったか
border davinci  # 行の終わりか
move
jump leonardo
davinci:
halt
```

まず，オドメーターは右に回る．続いて，ループがラベル leonardo: の宣言で始まりコマンド jump leonardo で終わる．ループ中で，オドメーターはマスに小石があるか，行の最後の境界

であるかを調べる．そのどちらでもない場合，オドメーターは現在のマス (0, j) から隣のマス (0, j+1) に移動する `move` を行う (マス (0, j+1) が存在するため)．いずれにせよプログラムは終了するので，コマンド `halt` はこの場合は必ずしも書かれていなくてもよい．

問題文 (Statement)

あなたは，上記で説明されたオドメーターの言語で書かれた，期待通りの動作をするプログラムを提出する．以下の各小課題は，オドメーターの動作および提出する解答が満たすべき制約を指定する．制約には次の 2 つがある．

- **プログラムのサイズ** (Program size) — プログラムは十分短くなければならない．プログラムのサイズは，コマンドの個数である．ラベルの宣言，コメント，空行はサイズには**数えない**．
- **実行の長さ** (Execution length) — プログラムは十分速く実行を終了しなければならない．実行の長さは，行われた**ステップ**の個数である： 1 回のコマンドの実行は (影響があったかなかったかにかかわらず) 1 ステップと数えられる．ラベルの宣言，コメント，空行はステップには数えない．

例 1 では，プログラムのサイズは 4 であり，実行の長さは 4 である．例 2 では，プログラムのサイズは 6 であり，マス (0, 10) に小石が 1 個だけあるときに実行すると，実行の長さは 43 ステップである： `right`，10 回のループ (1 回のループは 4 ステップかかる： `pebble davinci`，`border davinci`，`move`，`jump leonardo`)，`pebble davinci`，`halt` である．

小課題 1 [9 点]

初め，マス (0, 0) に x 個，マス (0, 1) に y 個の小石があり，他のマスには何もない．どのマスにも小石は 15 個以下であることに注意せよ．x ≤ y ならばマス (0, 0) に，そうでなければマス (0, 1) にオドメーターがいる状態で実行が終了するプログラムを作成せよ．実行終了時のオドメーターの向き，マス目の上の小石の個数や位置は何でも構わない．

制限： プログラムのサイズ ≤ 100，実行の長さ ≤ 1 000.

小課題 2 [12 点]

小課題 1 と同じ課題だが，プログラムが終了したとき，マス (0, 0) にちょうど x 個，マス (0, 1) にちょうど y 個の小石がなければならない．

制限： プログラムのサイズ ≤ 200，実行の長さ ≤ 2 000.

小課題 3 [19 点]

行 0 にちょうど 2 個の小石がある： 1 個はマス (0, x) に，もう 1 個はマス (0, y) にある．x と y は異なり，x + y は偶数である．オドメーターをマス (0, (x + y) / 2) (すなわち，小石が

あった 2 個のマスの中点) にいる状態で実行が終了するプログラムを作成せよ．マス目の最後の状態は何でも構わない．

制限： プログラムのサイズ ≤ 100，実行の長さ ≤ 200 000.

小課題 4 [最大で 32 点]

マス目に 15 個以下の小石があり，どの 2 つも異なるマスにある．すべての小石を北西の隅に集めるプログラムを作成せよ．より正確には，最初にマス目に x 個の小石があった場合，最後にはマス (0, 0) にちょうど x 個の小石があり，他の場所には小石がない状態にしなければならない．

この小課題の得点は提出されたプログラムの実行の長さによる．より正確には，様々なテストケースに対する実行の長さの最大値を L とすると，あなたの得点は以下のように決まる：

- L ≤ 200 000 ならば 32 点,
- 200 000 < L < 2 000 000 ならば $32 - 32 \log_{10} (L / 200\,000)$ 点,
- L ≥ 2 000 000 ならば 0 点.

制限： プログラムのサイズ ≤ 200.

小課題 5 [最大で 28 点]

各マスにある小石の個数は 0 以上 15 以下の任意の値である．オドメーターが最小値を見つける，すなわち，実行が終了したとき次の条件を満たすマス (i, j) にいるようなプログラムを作成せよ： どの他のマスにも，マス (i, j) にある小石の個数以上の小石がある．プログラムの実行が終わった後，各マスにある小石の個数は初期状態と同じでなければならない．

この小課題の得点は提出されたプログラムのサイズ P による．より正確には，あなたの得点は以下のように決まる：

- P ≤ 444 ならば 28 点,
- 444 < P < 4 440 ならば $28 - 28 \log_{10} (P / 444)$ 点,
- P ≥ 4 440 ならば 0 点.

制限： 実行の長さ ≤ 44 400 000.

実装の詳細 (Implementation details)

あなたは 1 つの小課題につき上記で指定された文法ルールに従って書かれた 1 つのファイルを提出しなければならない．提出するファイルは 1 つあたり 5 MiB 以下でなければならない．それぞれの小課題に対して，あなたのオドメーターのコードは数個のテストケースに対してテストされ，使われた資源についてのフィードバックが得られる．コードが文法的に正しく

ない場合は，具体的な文法エラーの情報が得られる．

あなたの提出は，必ずしもすべての小課題に対するプログラムを含んでいなくてもよい．あなたの現在の提出が小課題 X のプログラムを含んでいなければ，小課題 X への最も新しい提出が自動的に入れられる．該当するプログラムがなければ，その小課題の得点は 0 点となる．

通常通り，各提出の得点は小課題の得点の和であり，最終的な得点はリリーステストされた提出の得点と最後の提出の得点のうちの最大値である．

シミュレータ (Simulator)

テスト実行のために，オドメーターのシミュレータが提供される．あなたは，プログラムと入力のマス目を与えることができる．オドメーターのプログラムは，提出のためのものと同じ (上記で説明された) 書式に従う．

マス目の記述は以下の書式で与えられる： ファイルの各行は 3 個の数 R, C, P を含む．これは，R 行 C 列のマスに P 個の小石があることを表す．マス目の記述に書かれていないマスは小石を含まないとされる．例えば，ファイル

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

で記述されるマス目は，マス (0, 10) に 3 個，マス (4, 5) に 12 個の計 15 個の小石を含む．

タスクディレクトリで，プログラム `simulator.py` にプログラムのファイル名を引数として渡して呼び出すことにより，シミュレータを動かすことができる．シミュレータには，以下のコマンドオプションがある：

- `-h`：利用できるオプションを簡単に説明する．
- `-g GRID_FILE`：ファイル `GRID_FILE` からマス目の記述を読み込む (デフォルトでは空のマス目である)．
- `-s GRID_SIDE`：マス目の大きさを `GRID_SIDE x GRID_SIDE` する (デフォルトでは問題の設定通り 256 である)．小さいマス目をデバッグに活用できるだろう．
- `-m STEPS`：シミュレーションの実行ステップの回数を高々 `STEPS` 回に制限する．
- `-c`：コンパイルモードに入る．コンパイルモードでは，シミュレータは完全に同じ出力を返すが，Python でシミュレーションをする代わりに小さな C プログラムを生成しコンパイルする．開始時のオーバーヘッドが大きくなるが，非常に速く結果を返す．プログラムがおよそ 10 000 000 ステップ以上動作すると予想されるときは，このオプションを使うことが推奨される．

提出回数 (Number of submissions)

この課題に対して可能な最大の提出回数は 128 回である．