Assignment 8: the process of program development: `simwalk`

By now, you have experience writing programs in bash scripts, some experience in Python, and reasonable understanding good programming practices. Now it is time for you to apply the top-down approach for the process of program development by writing a program that requires parsing through a text file, which is provided by the user as a file argument to the script at the time of execution.

Imagine a medieval city's town square surrounded by four walls. Suppose that the town square has a grid-like layout, whose units are the stride of an average person's steps.

- The center of the grid is the origin – the starting point of the walk.
- Each grid point is a square tile of unit size 1 step by 1 step.
- The walls that surround the town square enclose a rectangle of size H by L whose center is at the origin.
    - Both H and L are odd numbers.
- There is a gate in the east wall that will always span 5 tiles, centered on the east-west axis of the grid.
- This town square has no buildings to impede a person's path – it does not any obstacles of any kind.

Shown below is a ***graphical example*** ofa town square of size 11-steps (height) by 13-steps (length):

Length

Height

Wall	Wall	Wall	Wall	Wall	Wall	Wall	Wall	Wall	Wall	Wall	Wall	Wall
Wall												Wall
Wall												Wall
Wall												Gate
Wall												Gate
Wall						Origin						Gate
Wall												Gate
Wall												Gate
Wall												Wall
Wall												Wall
Wall	Wall	Wall	Wall	Wall	Wall	Wall	Wall	Wall	Wall	Wall	Wall	Wall

A very drunk person starting at the center, the origin, takes one step at a time either to the north, east, south, or west. Each step is random, and puts the drunk on an adjacent tile. This person is so drunk, that each step is completely independent of the preceding ones. The future of this drunken person will vary based on their walking path. They[1] might:

- reach one of the walls purely by chance, or
- reach the gate purely by chance, or
- walk around without ever reaching a wall, tiring of exhaustion, and fall asleep.

We are interested in knowing how many times the drunk returns to the origin before reaching one of the walls, if at all, and whether they reach the gate. Reaching the gate is the goal.

Instructions

Write a Python program named `simwalk.py`, which will be given three command line arguments:

- the *dimensions of a hypothetical town square as its first two arguments,* as well as
- the *third argument being the pathname of a file that contains a text string that describes such a random walk.*

The contents of the input files describing the random walk are as follows:

- Each step of a walking path in the input file is represented by one of the letter characters `n`, `e`, `s`, or `w`, respectively meaning north, east, south, or west.
- A walk is a sequence of these letters with no white-spaces separating them. For example,

  ```
  nnneeewwssws
  ```

  is a walk that goes north three steps, then east three steps, then west two steps, then south two steps, then west one step, then south and stops. This walk ends at the origin.
- There is no bound on the length of the walk.

If the user executes the program with `401` steps and `601` steps and provides a random walk file named `walkfile`,

```
./simwalk.py 401 601 walkfile
```

The program should simulate the walking path contained in `walkfile` in the specified dimensions of the town square.

Notice the town square dimensions have not yet been clearly specified for their exact meaning.

[1] "They" in this circumstance is used as a third-person singular, gender-neutral pronoun.

There are two interpretations when passing in the dimensions of the town square:

- ***Interpretation A:*** Arbitrarily, the programmer designates the first argument and the second argument as the height dimension and the length dimension, respectively. In the example,
  the ***height*** would be `401` steps, and
  the ***length*** would be `601` steps.

- ***Interpretation B:*** Alternatively, a different programmer might designate the converse argument specification instead. Now, the first argument is for the length dimension, and the second argument is for the height dimension. In the example,
  the ***length*** would *be* `601` steps, and
  the **height** would be `401` steps.

Throughout your program, the specification of the city dimensions should always reflect your decided interpretation style. It should also be explained in the usage of the program for the error-checking conditions.

Error checking:
The program must report any errors on the standard error stream.

When the user incorrectly executes the program, the program should exit back to the command prompt. It should exit with an appropriate description of the error, and inform the user the proper usage of the program:

- If three arguments are not provided to `simwalk.py`, it is an error.
- If either of the first two command-line arguments are not positive integers, it is an error.
- If either the height or the length specified is less than 10, it is an error.

When the user does properly execute the program, the appropriate pathname to the input file, given as the third argument, should be opened and closed in a safe and proper manner to read its contents, and tested for invalid input.

- If the file cannot be read (no read permissions), it is an error.
- If the file does not exist, it is an error.
- If the input file contains any symbols or characters other than the letters `n`, `e`, `s`, or `w`, the program should also exit, with an error message, stating that the input file has corrupt data.
  - The input file can be assumed to contain no newlines or white-space characters; the program does not have to check that this is true.
  - The program should still be able to execute anyway if there are inclusions of newlines or whitespace within the file.

Program considerations:
If `simwalk.py` is a given dimension of the town square that is not an odd number, it should round up the specific value to the smallest odd number greater than it.

If there are no errors, the program should read the input file for its contents and simulate the walk.

Each time it returns to the origin, it must update a counter.

Reaching a wall is defined as trying to step into the wall. If the drunk is at a tile adjacent to a wall, they have not reached the wall. Reaching a gate is similarly defined. The walk stops if either

- they reach the gate,
- they reach a wall, or
- the sequence of steps ends – there are no more steps in the file that contains the random walk.

The output of `simwalk.py` when it is properly executed without any errors should be in the form of the statement,

```
The walk was <n> steps long returned to the origin <x> times. It
ended because it <...>
```

where

- `<n>` is the length of the walk,
- `<x>` is replaced by the count of how many times they return to the origin, and
- `<...>` is one of the three reasons that the walk stops.

There are some random walk files that you can use as test file for your program in the Linux Lab network in the `cs132` course directory:

```
/data/biocs/b/student.accounts/cs132/data/randomwalks
```

Grading Rubric

This assignment is graded on a 100 point scale.

The script will be graded on its correctness foremost. This means that it does exactly what the assignment states it must do, in detail. Correctness is worth 70% of the grade. Then it is graded on its clarity, simplicity, and efficiency, worth 30% of the grade.

The objectives when writing any script, are

- *clarity* – it should be easy to understand by someone with a basic knowledge of Python
- *efficiency* – it should use the least resources possible
- *simplicity* – it should be as simple as possible

Submitting Requirements

- **Due date:** Sunday November 8, 11:59PM Eastern Standard Time
- Late submissions will be docked total points at the rate of 1 point for every day it is late. No submissions will be accepted after Sunday November 15, 11:59PM Eastern Standard Time.
- Accepted formats:
    - Screenshot image(s) in JPEG, PNG, or TIF of the script, its execution for a sampling various input and the resulting output of each execution.
    - A zip archive file containing the `simwalk.py` script.
      The steps to create a zip archive and to secure-copy files is outlined in the tutorials called '`zip` command for beginners' and '`scp` command for beginners', that are located in the Course Materials section on the course Blackboard.