Title:	Methods and Tools for Software Engineering
Course ID:	ECE 650
Term:	Winter 2021
WWW:	`https://ece.uwaterloo.ca/~rbabaeec/ece650/w21`
Campuswire	`https://campuswire.com/p/G9003B726` Enter 3253 when prompted
Lectures:	Wednesday, 14:30 – 17:20 EST
Instructor:	Reza Babaee, `rbabaeec@uwaterloo.ca`
TAs:	An Qi Zhang, `an.zhang@uwaterloo.ca`

Assignment 1 - Due Sunday, January 26, 2020, 23:59

The Repository for Assignment 1

Simply run the following command to clone your repository:

```
git clone ist-git@git.uwaterloo.ca:ece650-1211/a1/USERID.git
```

where `USERID` is the ID you use to login to quest (without @uwaterloo).

This is the first in a series of assignments that is part of a single large project. The project is to help the local police department with their installation of security cameras at traffic intersections. You will solve a particular kind of optimization problem, called the Vertex Cover problem, in this context. The idea is for the police to be able to minimize the number of cameras they need to install, and still be as effective as possible with their monitoring.

For this assignment, you need to:

- Take as input a series of commands that describe streets.
- Use that input to construct a particular kind of undirected graph.
- Write your code in Python (Version 3.x).
- Ensure that it works on `eceubuntu.uwaterloo.ca`. (You are allowed to use only those Python libraries that are already installed on that machine. You are not allowed to import any new libraries. [1])

Input

Your program should take input from standard input. The input comprises lines each of which specifies a command. There are 4 kinds of commands. (1) add a street, (2) change a street, (3) remove a street, and, (4) generate a graph. There could be any whitespace before the commands. Each command is separated from its arguments with at least one whitespace. There could be any whitespace between the numbers that represent the coordinates.

Sample Input

Here is an example of how your program should work. Visualizing this example using the Cartesian coordinate system may help you understand what's going on [2].

```
a "Weber Street" (2,-1) (2,2) (5,5) (5,6) (3,8)
a "King Street S" (4,2) (4,8)
a "Davenport Road" (1,4) (5,8)
g
```

[1] You will need a VPN if you are working off-campus; follow the instructions at `https://uwaterloo.ca/information-systems-technology/services/virtual-private-network-vpn` on how to install and run a VPN client on your machine.

[2] You can use `https://www.desmos.com/calculator` to create a visualization for the street grid.

```
V = {
  1:  (2,2)
  2:  (4,2)
  3:  (4,4)
  4:  (5,5)
  5:  (1,4)
  6:  (4,7)
  7:  (5,6)
  8:  (5,8)
  9:  (3,8)
  10: (4,8)
}
E = {
  <1,3>,
  <2,3>,
  <3,4>,
  <3,6>,
  <7,6>,
  <6,5>,
  <9,6>,
  <6,8>,
  <6,10>
}
c "Weber Street" (2,1) (2,2)
g
V = {
  2:  (4,2)
  5:  (1,4)
  6:  (4,7)
  8:  (5,8)
  10: (4,8)
}
E = {
  <2,6>,
  <6,5>,
  <6,8>,
  <6,10>
}
r "King Street S"
g
V = {
}
E = {
}
```

Commands

- `a` is used to add a street. It is specified as: "`a "Street Name"` (x_1, y_1) (x_2, y_2) ... (x_n, y_n)". Each (x_i, y_i) is a GPS coordinate. We interpret the coordinates as a poly-line segment. That

is, we draw a line segment from (x_i, y_i) to (x_{i+1}, y_{i+1}). You are allowed to assume that each x_i and y_i is an integer. (Note, however, that the coordinates of an intersection may not be integers.)

- `c` is used to change the specification of a street. Its format is the same as for `a`. It is a new specification for a street you've specified before.
- `r` is used to remove a street. It is specified as "`r "Street Name"`".
- `g` causes the program to output the corresponding graph.

Output

Your program should output to the standard output. All your processes should terminate gracefully (and quietly) once you see `EOF` at stdin. Your program should not generate any extraneous output; for example, do not print out prompt strings such as "`please enter input`" and things like that. Since we are using an automatic marking script the output format should be exactly the same as the on provided here (the sample output). Do not print any extra whitespace or any other character after the output.

Errors

Error should be output to standard error. You can use exceptions in your code to catch errors. The above example is that of a "perfect" user — someone that did not make any mistakes with specifying the input. You should account for errors in the input. If a line in the input is erroneous, you should immediately output an error message. The format of the message is to be the string "`Error:`" followed by a brief descriptive message about the error. For example:

```
Error: 'c' or 'r' specified for a street that does not exist.
```

Your program should recover from the error as well. That is, your program should reject the errorneous line, but continue to accept input. Your program should not crash because of an error. Any erroneous input we try will be of a relatively benign nature that mimics honest mistakes a user makes. We will not try malicious input, such as unduly long lines or weird control characters.

The Output Graph

There is a vertex corresponding to: (a) each intersection, and, (b) the end-point of a line segment of a street that intersects with another street. An example of (a) from above is Vertex 3. An example of (b) is Vertex 1. The identity of a vertex can be any string of letters or integers (but no special characters). For example, `v1xyz` is acceptable as the identity of a vertex, but not `v1` `!!#xyz`. (The space is unacceptable, as are '`!`' and '`#`'.

There is an edge between two vertices if: (a) at least one of them is an intersection, (b) both lie on the same street, and, (c) one is reachable from the other without traversing another vertex. An example from above is the edge $\langle 1, 3 \rangle$, which connects the end-point of a line segment to an intersection. Another example is the edge $\langle 3, 6 \rangle$, which connects two intersections.

Marking

Your output has to perfectly match what is expected. You should also follow the submissions instructions carefully. The reason is that our marking is automated.

- Does not compile/make/crashes: automatic 0
- Your program runs, awaits input and does not crash on input: + 20
- Passes Test Case 1: + 20
- Passes Test Case 2: + 20
- Passes Test Case 3: + 15

- Correctly detects errors: + 25
- Programming style: + 0 for this assignment, but will be > 0 for future assignments.

Submission Instructions

Your code can be in multiple Python source files. The main python file should be `a1ece650.py`. The assignment should be submitted as a `master` branch on gitlab. Don't forget to enter your student identification and hours you worked on the assignment in `user.yml` and follow any additional instructions in `README.md`.