Title:	Methods and Tools for Software Engineering
Course ID:	ECE 650
WWW:	`https://ece.uwaterloo.ca/~wdietl/teaching/2015f/ece650/`
Lectures:	Thursday, 17:30 – 20:20
Instructor:	Dr. Werner Dietl, `wdietl@uwaterloo.ca`, DC 2522
TA:	Parsa Pourali, `ppourali@uwaterloo.ca`

Office hours by appointment. Begin all email subjects with `[ECE650]`.

Assignment 3 - Due Tuesday, November 10, 12:00 noon

For this assignment, you are to:

- Modify the output format of your Python script from Assignment 1 to match the input format of your C program from Assignment 2.
- Write a program in C to generate random input for your Python script.
- Write a driver program in C that, using Inter-Process Communication (IPC), links the output of the random input generator to the input of the Python script, and the output of the Python script to the input of the C program from Assignment 2.

Resources

The online book, "Advanced Linux Programming", will be useful for this assignment and beyond.

Sample Run

You should name the drivers executable `a3-ece650`. In the following, we assume that "`$`" is the shell command-prompt.

```
$ make clean
$ make
$ chmod u+x ./a1-ece650.py
$ ./a3-ece650 -s 5 -n 4 -l 5
V = 8
E = {<0,2>,<0,3>,<0,4>,<1,3>,<4,7>,<5,2>,<5,6>}
s 2 4
2-0-4
```

In the above, the first four lines make your executable, ensure your Python script is executable, and run the driver program with some command-line arguments. Then, the lines "`V = ...`", "`E = ...`", and "`2-0-4`" are output. The input the user provided to your program via stdin is "`s 2 4`".

Input, Output and Error

Your program should take input from stdin, and output to stdout. Errors should be output to stderr. Errors should always start with "`Error:`" followed by a brief description. 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.

As the example above indicates, there are two kinds of inputs the user provides. One is via the command-line arguments, like `-s` and `-n`. This is done once only, when your program is run. The other is the '`s`' input at stdin, which may be issued repeatedly, just as in Assignment 2. Your program should output a shortest path.

We will not test your program for format errors in the input. That is, the command-line arguments, if specified, will be formatted correctly, and the `s` input will also be formatted correctly.

Of course, we may omit command-line arguments (see below for what to do in such cases), and specify vertex IDs to `s` that do not exist, or between whom a path does not exist. The latter two cases should cause your program to report an error.

Marking

- Marking script for uncompressing etc. fails: automatic 0
- Does not compile/make/crashes: automatic 0
- Your program runs, awaits input and does not crash on input: + 20
- Run in default mode: + 15
- Error check arguments: + 5
- Test functionality: + 20
- Test `rgen`: + 20
- Replace `rgen`: + 20

Makefile

As discussed below under "Submission Instructions", you should create a `Makefile`. Your `Makefile` should contain the target "`all`" that makes a final executable named `a3-ece650`. Your `Makefile` should contain also a target called "`clean`", that removes all object and executable files. We will first issue a "`make clean`", and then a "`make`" to compile your code. If your code is not compiled from scratch (i.e., from the C sources), you get an automatic 0.

Submission Instructions

You should place all your files in a single folder (directory), `a3-ece650`. The folder should contain:

- All your C and Python source-code files. IMPORTANT NOTE: the executable for your random generator must be named `rgen`. The reason is that part of our testing will replace your generator with ours.
- A `Makefile`, that builds the final executables.

You should then `tar` and `gzip` the folder using the command:
`tar czf a3-ece650.tar.gz a3-ece650`.
(To `untar` and `gunzip`, we will issue `tar xzf a3-ece650.tar.gz`.)
You should upload `a3-ece650.tar.gz` to the dropbox on Learn for Assignment 3.

File names

There are two file names that are important. One is your main executable. This must be named `a3-ece650`. Our marking script will simply try and run this after doing a `make`. The other executable file whose name is important is your random input generator. The executable for this must be named `rgen`. The reason is that for some of our tests, we will replace your `rgen` with ours.

Things to be done

Python script

You should edit your Python script from Assignment 1 so that its output format for the specification of the graph matches the input format for your C program from Assignment 2. Think of it as though your Python script provides command-line input to the C program. The way to do this is to simply map each vertex to an index in $[0, n-1]$ (where n is the number of vertices), and rename your edges accordingly.

Also, the only output that your Python script should produce to stdout is in response to "`g`", for which it outputs the specification of the graph (i.e., `V` and `E`). Error output should go to stderr.

Random input generator

Your random input generator should generate random inputs of street specifications for your Python script. It takes four command-line arguments. All are optional.

- `-s k` — where `k` is an integer ≥ 2. The number of streets should be a random integer in $[2, \texttt{k}]$. If this option is not specified, you should use a default of $\texttt{k} = 10$; that is, the number of streets should be a random integer in $[2, 10]$.
- `-n k` — where `k` is an integer ≥ 1. The number of line-segments in each street should be a random integer in $[1, \texttt{k}]$. Default: $\texttt{k} = 5$.
- `-l k` — where `k` is an integer ≥ 5. Your process should wait a random integer in $[5, \texttt{k}]$ seconds before generating the next (random) input. Default: $\texttt{k} = 5$.
- `-c k` — where `k` is an integer ≥ 1. Your process should generate x, y coordinates such that every x and y value is in the range $[-\texttt{k}, \texttt{k}]$. For example, if $\texttt{k} = 15$, all of your coordinate values should be integers between -15 and 15. Default: $\texttt{k} = 20$.

Your program should generate a specification of streets in the format that your Python script expects (see Assignment 1). You can name the streets whatever you want. You should ensure that your input does not have errors. For example, if you generate a line-segment that overlaps with a line-segment (across all streets) generated earlier, you should regenerate that line-segment. Similarly, you should not have any zero-length line segments.

Also, note that your random generator could go into an infinite loop looking for a valid specification. You should disallow this from happening by limiting the number of tries. That is, if your random generator fails to generate a valid spec for a continuous A number of attempts, it should `exit()` with an error message to stderr. A reasonable A to adopt may be 25. Whatever A you adopt, your error message should mention it. That is, your error message should be something like, "`Error: failed to generate valid input for 25 simultaneous attempts`".

Before a new specification to your Python script, your generator should issue "`r`" commands to your Python script to remove all previous streets and replace them with the new specification of streets.

You must use `/dev/urandom` as the source of your random data. Also, see under "Submission Instructions" how your executable for the random generator must be named.

Driver

Your driver program has overall control. You have at least three programs that run concurrently: (1) the random generator, (2) your Python script, and, (3) your program from Assignment 2 that takes as input a graph-specification and computes shortest paths. Your driver program should `fork()` two processes and `exec()` two of those programs, and then `exec()` the third (so it turns into a process that corresponds to the third program). It should set up all IPC appropriately beforehand.

It should send normal output to stdout, error output to stderr, and take input from stdin. As we mention above, the only input it takes are "`s`" commands, that ask for a shortest path between vertices.

Gotcha warning: note that I say "you have at least three programs that run concurrently." You may need more than three that run concurrently. (Why?)