CSCI415—Systems Programming

Spring 2017

Programming Assignment 6

Boiling Beetles with a Shell Script

Shell scripts are often used to extend the function of a binary. In this project, you will be given a binary for the beetle program of project one, and you will embed it in a shell script in order to add substantial capability. Note that you will *not* be able to modify the basic beetle program.

The `boiler` Script – Simple Use

Your shell script will be named `boiler`. It's simplest invocation is in the form

```
% ./boiler -n num -i min max step
```

The argument `num` is a positive non-zero integer; it indicates the number of beetles to be simulated in each run of the `beetle` program caused by execution of the script. The three arguments `min`, `max`, and `step` are also positive non-zero integers. These three arguments specify a sequence of square sizes on which the beetles' lives are to be simulated:

- the smallest square should be `min` inches on each side;
- the next smallest square should be `min` + `step`;
- the sequence ends if the next square would have a side dimension greater than `max`.

Here are some concrete examples of the simple use of the script:

```
% ./boiler -n 10000 -i 1 6 2
```

This would boil 10000 beetles on squares of dimension 1, 3, and 5 inches.

```
% ./boiler -n 10000 -i 3 1 1
```

This would boil 10000 beetles on a 3 inch square. Note that, regardless of the values for `step` and `max`, at least one run of the `beetle` program will take place.

The argument sequences can be given in any order, so

```
% ./boiler -i 1 6 2 -n 10000
```

is legal, and behaves as you would expect (just like the earlier example).

The final output of the script (on stdout) is required to appear as shown below. The output is generated only after **all** simulations have completed. A line of output will contain the size of a side of the square as an integer number of inches and a floating point mean beetle lifetime. The two numbers should be separated by whitespace. The mean beetle lifetime should have one digit to the right of the decimal point in your output.

```
square_dim_1 mean_life_1
square_dim_2 mean_life_2
. . .
square_dim_n mean_life_n
```

Note that for a large number of beetles and several square sizes, the script may run for a long time. Accordingly, your script must respond to several signals:

`SIGINT:` terminate the activity of the script immediately and clean up all temporary files the script may have created.

`SIGUSR1:` as soon as possible, output (to standard output) the square size that is currently being worked on and then continue with normal operations.

`SIGUSR2:` dump a restorable image (see below) of the current state of your beetle simulation to the file `./beetle.state`, terminate the script, and clean up any temporary files.

The `boiler` Script – Graphical Output

If the optional argument sequence `-p filename` is asserted in addition to the basic required arguments described above, then your script will generate a PNG graphical plot of your beetle simulations. This plot should display square size on the x axis and mean beetle lifetime on the y axis. The axes should be labeled appropriately. The `filename` designates where the PNG image will be stored.

The `-p filename` command line argument sequence may appear in any order relative to the `-i` and `-n` argument sequences.

Your script should use the powerful `gnuplot(1)` program to produce the PNG plot. There is a short manual page for `gnuplot`, but more complete documentation can be accessed with the command "`info gnuplot`."

Here is an example:

```
 % ./boiler -i 1 10 1 -n 10000 -p plotfile.png
```

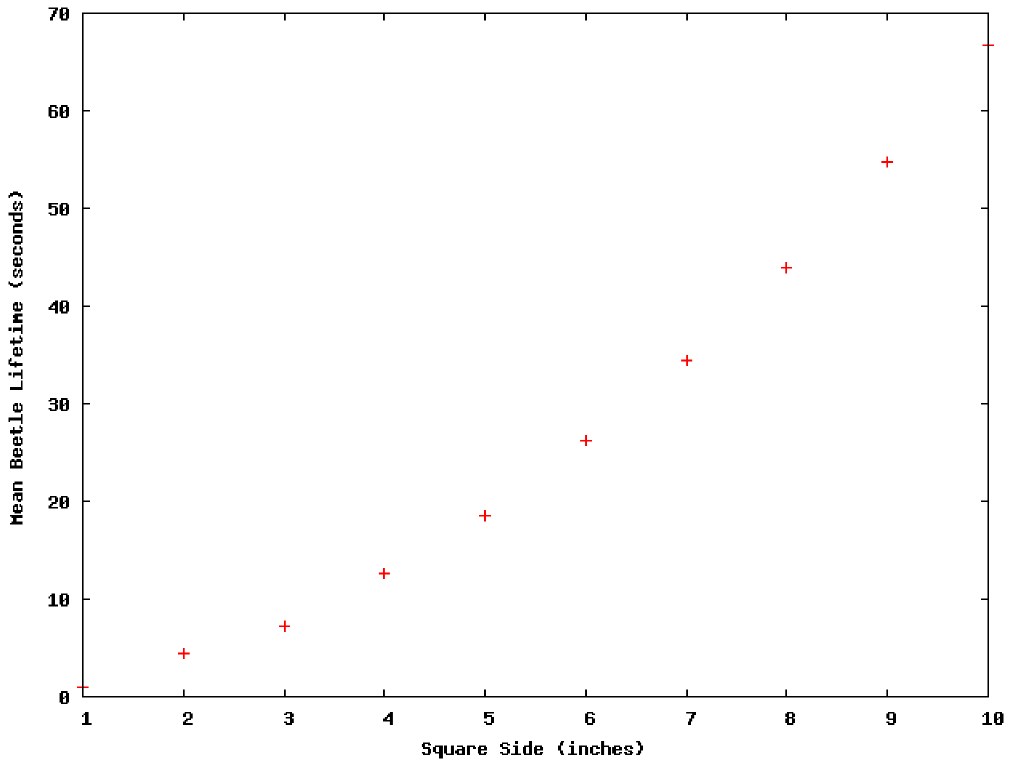
In this invocation, ten `beetle` simulations will be run, each with 10000 beetles, on ten different square sizes. The standard output will still look like it does in a basic invocation of `boiler`, but a PNG image containing the required plot will be produced in `plotfile.png`.

This run of the `boiler` script should produce the following on its standard output:

```
1 1.0
2 4.5
3 7.2
4 12.6
5 18.5
```

```
6 26.2
7 34.4
8 43.9
9 54.8
10 66.7
```

The image in `plotfile.png` should look something like the following:



The `boiler` Script – Saving/Restoring

Given that the script may run a long time, it is useful to be able to stop the execution and save its state in a file. At a later time (and maybe on a different machine) the file in which the script's state was dumped can serve as the basis for restarting the script at the point where it was stopped previously. As specified above, if your script receives a `SIGUSR2` signal, it needs to stop execution and dump its state into the file named `beetle.state` in the current working directory. Exactly what is meant by "state" depends on the design of your script, but it would probably include the command line arguments used when the script was invoked, the size of the square being simulated when the `SIGUSR2` was received, and perhaps the contents of any significant temporary files. In any case, the saved state should be sufficient to restart the script when the following invocation is used (`statefile` is the name of the state file in this example):

```
% ./boiler -r statefile
```

This should restart the beetle simulation (approximately) at the point where it was interrupted by the `SIGUSR2`. Note that there is no need to re-output anything that had been sent to standard output before the `SIGUSR2`. After being restarted, the script should behave just as if it had never been interrupted.

If the `-r` option is asserted on an invocation of `boiler`, no other option sequences are allowed.

Notes

The beetle program operates as specified in P1; accordingly, your script can rely upon the argument checking of the beetle binary, it doesn't have to check that positive non-zero integers are given as arguments where required. You are responsible for checking general argument validity, however.

If a run of the `beetle` program inside the script terminates abnormally, the script should terminate immediately after cleaning up. A diagnostic message and any partial output should be produced. If a plot was requested, it should not be produced.

Your script must invoke the `beetle` binary located at

```
/home/f85/kearns/public/415/p6/beetle
```

The first line of `boiler` must be `#!/bin/sh`. This is necessary for finding the pid of the script so signals can be sent to it.

If you develop on your own system, please reserve time to make sure that your submission behaves as you expect on a departmental Linux system. Software version differences can be problematic.

Restrictions

The `boiler` program must be a shell script, interpreted by `/bin/sh` on a departmental system. You may use `awk` scripts as part of your solution as long as they are embedded in the shell script (you may only submit a single file, `boiler`). No other scripting language nor non-system binary may be used in your submission.

Submission

In the directory where you develop `boiler`, do the following:

- `gpg -r CSCI415 --sign -e boiler`
- `cp boiler.gpg ~`
- `chmod 0644 ~/boiler.gpg`

You must create your gpg file by the deadline stated on the web page.

Please note that the final project (P7) submission will be on April 28, the last day of classes. You will not be able to use slip days on that submission.