Lab 6
Preview

23 February 2017

This week you'll spend a bunch of time reading and playing with my code, and continuing to learn your way around some of tools that will save you some time and help you work with larger programs. To get started, create your `lab6` directory and copy into it all the files from `/home/shared/162-1/lab6/`.

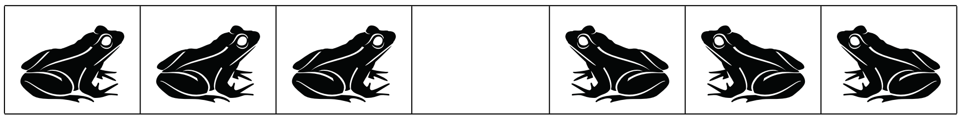
Sometime before lab, read all the files and take notes (in your notebook) on any C++ feature that we haven't talked about before. Include filenames and line numbers.

Then, spend at least a little while specifically looking at the `solvefrog` function (in `solvefrog.cpp`) to figure out how it works. Write either a 2–3 sentence high-level description of how it works, or else a 2–3 sentence description of which parts of the function you don't yet understand (and your best guess as to what they're similar to).

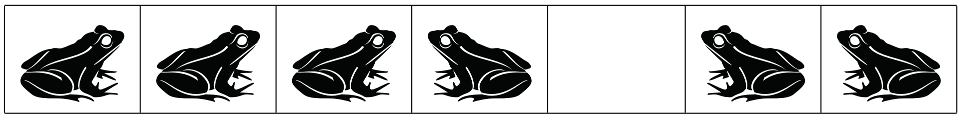
Below is a description of the puzzle that this code is implementing.

The leapfrog puzzle

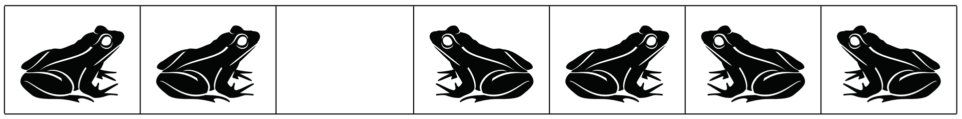
As you may have noticed, the files in this directory have frog-related names; they are all related to a solitaire puzzle that works as follows. There are seven cells filled by frogs facing to the left and right, initially as in this diagram:



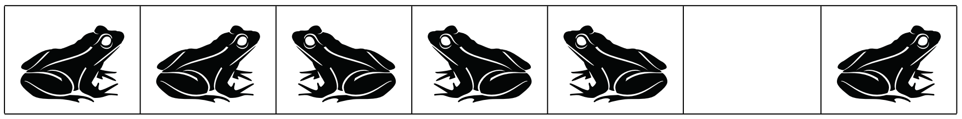
A frog can only be moved into the empty square, and only if the frog is immediately in front of it or can get to it by leaping over a single other frog. For instance, if we choose the third from the end to step forward, the result would be as follows:



and then the frog in front of that one could jump over him:



If you're not careful, you can get yourself stuck:



But if you choose the moves in the correct order, you can make all the frogs swap places before getting stuck:

In this directory, the `Leapfrog` class is designed to simulate the puzzle itself, and the `solvefrog` function (and file) is designed to simulate a player of the puzzle, i.e. someone trying to solve it.

Credit: Frog image by Vectorportal.com