ECE 479/579
Spring 2018

(Die Hard) Homework #1
Due: Feb 13, 2018, 11:30pm in D2L

Recall the water jug problem discussed in class. Write a program in C, C++, or Java, that takes an input file specifying:

a) containers' capacities,
b) initial state, and
c) goal state (all as integer values).

The capacities and input state's values are provided as precise amounts. That is, for example, an initial state could be a pair (0, 3). The goal state, however, can be entered as (2, -1) where 2 denotes exactly 2 gallons in the first container and -1 denotes any amount (less than or equal to the allowed capacity) of water in the second container. Similarly, (-1, 3) would denote any amount in the first container (again subject to the capacity constraint) and exactly 3 gallons in the second container). For instance, the goal state (2, 1) would require exactly 2 gallons in the first container and exactly 1 gallon in the second.

The program should output a sequence of actions necessary to solve the problem, i.e., actions that allow you to arrive at the goal state given your initial state, for two control strategies: a) random – given the list of operators ordered in ascending order, randomly generate the next operator's number and check to see if that operator applies to your current state (if so, apply it, otherwise randomly select another operator), and b) a systematic state expansion.

Example input file
Capacity of jug A: 4
Capacity of jug B: 3
Initial state: 0 0
Goal state: 2 -1

Sample output
```
…
>Strategy A
>Starting out with a 4-gal jug and a 3-gal jug          --- state: (4, 0)
>Empty the 4-gal jug                                    --- state: (0, 0)
…
>Strategy B
>Starting out with a 4-gal jug and a 3-gal jug          --- state: (4, 0)
>Pour water from the 4-gal jug into the 3-gal jug       --- state: (1, 3)
…
```

Run your test cases twice, that is for the strategies a) and b) as defined above. For strategy a) if the search is unsuccessful after you have tried 250 state expansions, terminate it. **Please use the public cases(in D2L) to check your result. Grading will use not only these public cases.**

***** Output format must follow the sample output. Otherwise, you won't receive any credit.**

Submission Requirements

File Format and Naming Rules

- All assignments must be submitted as a single ZIP (.zip) using the naming convention: hw01_NetID.zip. NetID is your UA NetID.
  Ex. My email: kspeng@email.arizona.edu,
  My submission filename: hw01_kspeng.zip.
- Include a README.txt in your submission that specifies your programming language.
  Ex. Programming: C++
  If you use C++11, please specify C++11. Ex. Programming: C++
- The submitted archive must use the following directory structure:

```
C\C++:      hw01_NetID
            |____\ README.TXT
            |____\ src
                      |____\ CMakeLists.txt
                      |____\ .cpp or .c or .h files
Java:       hw01_NetID
            |____\ README.TXT
            |____\ src
                      |____\ .java files
```

- The argument is only the **input file name**.
- The result must be **printed in the console** and written into an **output.txt** file.
- The source file should be named as following: **hw01_NetID.xxx**
- Sample run:
  Ex. hw1_kspeng test_case.txt
- Do not include any build files, packages, or .exe in the directory.

***** If you do not follow the format and naming rules, we can't recognize your works and you will not get any credit.**

Compiling Rules

- All assignments will be compiled and tested using the ECE department server ece3/compute (ece3.ece.arizona.edu/compute.ece.arizona.edu), on which the CMake tools and java compile tools are already available. You can access the ece3/compute server using any secure shell (SSH) client. All students have your own ID and PASSWORD to access ece3 server. You are strongly encouraged to test your programs using the ece3 server before submission.
  - You can use ssh in command window or putty (download here) to connect to ece3 server.
  - ID and PASSWORD are you current NetID and PASSWORD.

Rubrics:

Must follow the Formatting/Submission Requirements. If not, 0 point.	
Must run/compile on ece3 (**Must be no error and warning.**)	5%
Actions must be valid	20%
State must follow from the action	20%
Must stop when found an acceptable end state	15%
For strategy a) stop unsuccessful searches after 250 state expansions	15%
For strategy b) must never repeat a previous state	25%