Lab 02 Specification – Exploring Scope
Due (via your git repo) no later than 8 a.m., Tuesday, 18 September 2018.
50 points

Lab Goals

- To understand the scope rules and issues.
- To experiment, visualize and study the stack structure in JavaScript and in Java.
- To analyze the code optimization. To reflect on the lab experience with scoping.

Suggestions for Success

- Take a look at the suggestions for successfully completing the lab assignment, which is available at: `https://www.cs.allegheny.edu/sites/amohan/resources/suggestions.pdf`

Learning Assignment

To do well on this assignment, you should also read

- PLP chapter 03, section (3.1 - 3.3)

Tasks

- **[Begin a document.]** You will hand in a document with answers to several questions, so go ahead and create this document. In your document, place your name and your honor code pledge at the top. Number your answers according to the question numbers on this lab handout.
- **[1- More about scope in JavaScript.]** Consider the following JavaScript program—what value is printed by the final line—is it `"hello"` or is it `"10"`?

```
var x = "hello";

function f() {
   x = 10;
   console.log("x ="+x);
   var x;
}

f();
console.log("x="+x); /* What gets printed, "hello" or 10? */
```

  In your document, answer the question above (easy! just run it!), then (a) explain how JavaScript's "function scope" rule is interpreted, and (b) state whether or not JavaScript requires "declare before use" for variables.

- **[2- A nifty visualization.]** Go to the website `http://www.pythontutor.com` and click on the link to "Visualize your code and get live help now!"

  From the dropdown menu, choose JavaScript. Enter the following code:

```
function f(x) {
  g(x+1);
}
function g(x) {
  console.log("in g, x ="+x);
}
f(3);
```

  Click the "Visualize Execution" button and then keep pressing the "Forward" button, watching the animation on the right side at each step.

  In your document, provide a brief (one paragraph) explanation of the various stack frames that appear and disappear.

- **[3- Look at the stack structure in Java.]** Generate the JVM bytecode for the Java class `Stack1.java` (A copy of the file is provided in the lab git repository). There is no "`main`" here, so you can't run this, but you can still compile it and view the bytecode with either `jbe` or the `javap -c` command (refer to lab 1).

  In bytecode, the instructions "`iload`", "`dload`", "`istore`", and "`dstore`" take a numeric argument that specifies a location in the activation record (or frame). (Sometimes this argument is part of the instruction name, e.g, "`iload_1`"). In your document, "Draw" the portion of the frame containing the paramaters and local variables of function `f`, showing where each parameter `i`, `j`, `a`, etc., and each local variable `sum`, etc., appears in the frame. By "draw" I just mean something like this:

```
1:   name of variable in frame location 1
2:   name of variable in frame location 2
3:   name of variable in frame locations 3 and 4*
5:     etc.
```

  *Recall that a `double` takes twice as many spaces as an `int`.

- **[4- A stack machine computation.]** Generate the byte code for program `Stack2.java` (A copy of the file is provided in the lab git repository). Assume that function `f` is called with `x = 10`, `y = 20`. "Draw" the frame for `f` (as in the previous question). Then "draw" the contents of the stack after each line of bytecode in function `f`. If the stack contains the value of a variable, name it. For instance, after the first two lines of bytecode are executed, the stack looks like this (I'm assuming the stack grows upward):

```
20 (y)
10 (x)
```

  and after the third line it looks like:

```
30
```

  Remember how a stack machine works: operators like "`add`" pop their operands off the stack and then push the result (the above example shows how `x` and `y` are pushed on the stack, then they are popped and the value of `x+y` is pushed onto the stack.)

- **[5- One more look at optimization.]** Look once more at Java program `Stack2.java` and its bytecode. How could this bytecode be optimized, i.e., shortened to fewer instructions? Answer this question, explaining and showing your optimized bytecode, in your document. (Feel free to experiment with the `jbe` editor.)

Required Deliverables

Please submit electronic versions of the following deliverables to your GitHub repository by the due date:

1. The report document with the answers to the questions posed in the lab assignment.
2. Reflection (in the report document) on the biggest learning points and any challenges that you have encountered during this lab.
3. Your solution file should strictly be typed and any images of hand-written text is not acceptable. Your solution file should be in PDF format, any other formats will not be acceptable. Although not a requirement, it is highly recommended to use **LATEX** to create your solution file. **LATEX** is an unique text editor that provides an exclusive feature to publish highly professional PDF files based on TEX programming commands. One distinguishing feature of TEX commands are that, it provides a platform to write complex mathematical equations in a very elegant and professional manner. A sample tex file created by Prof. Roberts from University of Adelaide is provided in the repository for your reference. There are other great samples on the web, a google search should lead you to several other examples. Also if you are interested, the link to a quick start by Prof. Roberts on latex is provided below:

   `http://www.maths.adelaide.edu.au/anthony.roberts/LaTeX/ltxqstart.php`

Grading Rubric

1. If you complete Task 1 completely as per the requirement outlined above, you will receive 5 points.
2. If you complete Task 2 completely as per the requirement outlined above, you will receive 5 points.
3. If you complete Task 3 completely as per the requirement outlined above, you will receive 15 points.
4. If you complete Task 4 completely as per the requirement outlined above, you will receive 15 points.
5. If you complete Task 5 completely as per the requirement outlined above, you will receive 10 points.
6. If you fail to upload the lab solution file to your git repo by the due date, there will no points awarded for your submission towards this lab assignment. Late submissions will be accepted based on the late submission policy described in the course syllabus.
7. Partial credit will be awarded, based on the work demonstrated in the lab submission file.
8. If you needed any clarification on your lab grade, talk to the instructor. The lab grade may be changed if deemed appropriate.