1. (**10 points**) Let $A_1 = \{D_1\#D_2 \mid D_1$ and D_2 describe DFAs and $L(D_1) \subseteq L(D_2)\}$. In English, $D_1\#D_2 \in A_1$ iff D_1 and D_2 describe DFAs and every string that is recognized by the DFA described by D_1 is also recognized by the DFA described by D_2. You can assume that D_1 and D_2 are described as in the Oct. 24 lecture notes, or any other reasonable description. Show that A_1 is Turing decidable.

2. (**15 points**) Let $A_2 = \{G \mid G$ describe CFG and $L(G) \supseteq L(1^*)\}$. In other words, G generates every string that consists of zero or more 1's (it may generate other strings as well). Show that A_2 is Turing decidable.

3. (**15 points**) In class we considered a Java method, `boolean halt(String src, String input)`, that is supposed to return true if the Java program with the source code given by string `src` halts when run with input `input` and returns false otherwise. We showed in class that it is impossible to write such a method. Our proof involved passing the source code for a Jave program as both the `src` and `input` arguments to `halt`.

   Now consider a new method, `boolean haltNoJavaAsInput(String src, String input)`. This method returns `false` if `input` is a syntactically correct Java program. Otherwise, `haltNoJavaAsInput` returns true if the program described by `src` halts when run with input `input` and returns false otherwise (just line `halt` described above). Note that the question of whether or not a program is syntactically correct Java is Turing decidable – this is what a Java compiler does. More formally, `haltNoJavaAsInput` is a decider for the language A_3, with

$$\begin{array}{rcll} A_3 & = & \{J\#I \mid & J \text{ is the source code for a Java program} \\ & & \wedge & I \text{ is a string that is } \textbf{not} \text{ a syntactically correct Java program} \\ & & \wedge & \text{Program } J \text{ halts when run with input } I \\ & & \} & \end{array}$$

   Show that it is impossible to write a method `haltNoJavaAsInput` as described above. Equivalently, show that language A_3 is not Turing decidable.

4. 2. (**15 points**) In class, we constructed one example that must cause a proposed function for `halt` to give the wrong answer or never terminate. Show that for any proposed implementation of `halt` there must be an infinite number of inputs that cause it to give the wrong answer or never terminate.

5. 3. (**35 points**) Download the program `mystery.java` from

   http://www.ugrad.cs.ubc.ca/∼cs421/hw/7/mystery.java

   Look over the code, compile it, and run it – I promise that it's not malicious.

   (a) (**5 points**) What does the program do? Just give a one-sentence description of the output that it produces. You'll get to explain *how* it does it in the rest of the question.

   (b) (**5 points**) What is string `s` for?

   (c) (**5 points**) What does method `x()` do?
   A one sentence answer is enough. You'll get to explain the details in the next three questions.

   (d) (**5 points**) What do the first four `buf.append(...)`'s in `x()` do?

   (e) (**5 points**) What does the first `for` loop in `x()` do?

   (f) (**5 points**) What does the second `for` loop in `x()` do?

   (g) (**5 points**) What does method `fix(String)` fix?

6. (**20 points**) A 2-PDA is a PDA with two stacks.

   (a) (**10 points**) Describe a 2-PDA that recognizes the language $\{w \in \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}^* \mid \exists n.\ w = a^n b^n c^n\}$. This shows that a 2-PDA is more powerful than a 1-PDA.

   (b) (**10 points**) Show that the class of languages recognized by 2-PDAs is exactly the same as the set of Turing recognizable languages. (Hint: Show that any Turing machine can be simulated by a 2-PDA and vice-versa).

7. (**20 points**, *extra credit*) A *ray automaton* consists of an infinite number of DFAs, D_0 D_1, D_2, ... arranged in a line. The automata all have the same set of states, Q, the same start state $q_0 \in Q$, and the same transition function $\delta : Q \times Q \times Q \rightarrow Q$. A configuration of a ray automaton is a function $\mathcal{C} : \mathbb{Z}^{\geq 0} \rightarrow Q$ where $\mathcal{C}(i)$ gives the state of DFA D_i. The automaton moves from configuration $\mathcal{C}$ to configuration $\mathcal{C}'$ iff

$$\begin{array}{rcl} \mathcal{C}'(0) & = & \delta(q_0, \mathcal{C}(0), \mathcal{C}(1)) \\ \mathcal{C}'(i) & = & \delta(\mathcal{C}(i-1), \mathcal{C}(i), \mathcal{C}(i+1)), \quad i > 0 \end{array}$$

   In other words, at each step, each DFA makes a transition according to its own state and the states of its left and right neighbours. Because DFA D_0 has no left neighbor, it always uses q_0 as its left input. There is a special state q_f, and the ray automaton halts iff it reaches a configuration, $\mathcal{C}$ where D_0 is in state q_f, i.e. $\mathcal{C}(0) = q_f$.

   (a) (**10 points**) Prove that the halting problem for ray automata is undecidable.

   (b) (**10 points**) Is the halting problem for ray automate Turing recognizable? Justify your answer.

   Note: This problem is adapted from a similar problem from *Automata and Computability* by Dexter C. Kozen.