Exercises for *defining semantics for complex systems*

The next exercises are meant to give you some feeling for iTasks and the way we handle semantics in Clean. Do not hesitate to raise questions or to ask for help.

1. Getting started with iTasks.
   Login to the windows system and go to the directory C:/CleanCEFP. You will find there the application CleanIDE.exe. Start it by double clicking. Answer yes to its question about integrating this version in the system (if necessary).
   Open Projects/iTasks2/AllExamples.prj form the CleanIDE (by pressing ctrl-o, or via the file menu).
   Run this project (by pressing ctrl-r, or selecting *Update and Run* from the Project menu.
   Windows will ask you if you want to unblock this program, allow this permanently.
   Locate your browser to http://localhost. You should get a login screen. Login as *root*, with an empty password.
   Succeeded? Congratulations, this was the hardest part of the exercise. In case of problems check the steps above, or call for help.
   Tips:
   - If the CleanIDE cannot link your program the old version of the program is probably still running.
   - If your project has a "heap full" error you should increase the heap (and stack space) via the "project option…" dialogue in the project menu.
   - If you program complains about port 80 another program is using it. Most likely one of your other iTask programs (or Skype).
   - If your program behaves strange delete the folder named *project-**iStore*** and try again.
   - If the system does not behave as expected check if you project is in the same folder as the AllExamples-project (C:/CleanCEFP/Projects/iTasks2). To work in other folders you should change the definition of ResourceDir in iDataSettings.dcl.
   - Check if you have a project and if your project is running in the environment iTasks2.
2. Make your own task.
   Open a new file in the directory C:/CleanCEFP/Projects/iTasks2. You can choose the name, but the extension should be ***icl***. Create your own task that asks your name (using an editTask), and print "hello" followed by your name. Look at the previous exercise to see how you start your own task.
3. Change your task form the pervious exercise such that it only allows nonempty names.
4. [Optional] Invent your own task and implement it.

5. On www.cs.ru.nl/~pieter/cefp09 you will find the files exprSem.icl and exprSem.prj that contain the various versions of the semantics used in the lecture. Download these files to your computer and execute the project. It should produce the values of the used variables after executing the `facStmt`. Extend the language by conditional expressions of the form `If BExpr AExpr AExpr` by removing the comment symbols in the data type `AExpr`. Update the semantics in order to assign a proper meaning to this statement and test if your semantics is correct by executing one or more examples.
6. [Optional, harder and much work] Add pure and strict functions to the semantics. The constructor for defining these functions is given as comment in the type `Stmt`. The constructor to apply such a function is given in the type AExpr as a comment.
   The least painful way to do this is probably to turn the state intoa function with typ `Var -> Res` using the type `:: Res = Val Int | Fun [Var] AExpr`. This implies that functions and values live in the same name space.