Homework 1: The Programming Language Core

Due: Problems 2, 3(a), 4(a) on Feb 10; 3(b), 4(b), 6 on Feb 17; 10 and 21(a) on Feb 24; 14,16, 19 on Feb 26; 21(b) on Mar 3.

In this homework you will learn about the basics of type systems and semantics, from Schmidt's chapter 1. Please read that.

Feel free to substitute other work that you think may be more interesting or relevant for the material in this homework. However, if you do that, please state the problem you are solving clearly, and make it clear why you wanted to make a change.

You can work either in groups or on your own for all the problems, or for any combination; but please make it clear what you're doing with whom.

For work that you do involving computer programs, please use the `hugs` Haskell interpreter. Please also provide evidence that your program is correct (for example, test cases).

Hand in a printout of your code and your testing. Some questions just require an answer in English. Be sure to put your name on everything and to clearly label what problem each function solves with a comment.

1. (suggested practice) Do problems 1.1, 1.2, and 2.1 on page 24 of Schmidt's book.

2. For each of the following commands, write a proof that it type checks.

   (a) `loc 3 := (@loc 1) + 4 + 5`

   (b) `while not ((@loc 1) = 5) do loc 1 := (@loc 1) + 1 od`

   These can be done by hand on paper, but you may find it more convenient to work with a text editor, where you can copy text from one line to the next. For example, here's a proof that `loc 1 := 2 + 3` type checks.

```
                2: int      3: int
                ---------   ----------
                2: int exp, 3: int exp
                ----------------------
loc 1: int loc,  2 + 3: int exp
------------------------------
loc 1 := 2 + 3 : comm
```

3. Do problem 2.2 on page 24 of Schmidt's book. (Note the correction to this problem in the errata for the book.) Do this:

   (a) on paper, as suggested in the problem, using the usual notation from the book, and

   (b) then translating your ideas into Haskell. See the class's web page or the files in the following directory to start.

   `~leavens/WWW/ComS641/lib/`

   You should devise your own test cases to show that your code works.

   Hint: do the paper version first, and for that version, use the following steps (which you may need to iterate until they work).

   - Decide on the new abstract syntax you want to add, if any. Write this out in a grammar, as in Schmidt's Figure 1.1. Use "..." for the old, unchanged productions.
   - Decide on the new type attributes, and write out a grammar for them.

- Decide on the type checking rules, and write out the new axioms and inference rules (as in Figure 1.4).
- For at least one example, write out a proof that a phrase has a type, using your new type checking rules (and the old ones). If you encounter problems, with your rules go back and revise as necessary.
- Think about the bigger issues here, and how programs in the languge might be affected by your change. Can the new things you added be used effectively in programs? If not, then revise as necessary, and check again as in the previous step.

When translating your paper solution into Haskell, take the following steps.

- Decide on a concrete syntax for your abstract syntax. Copy and modify the parser (in `CoreLangParser.lhs`), and in that file change the abstract syntax, and change the parser to parse your concrete syntax. You'll also have to change the keywords and symbols in the definition of the scanner, and the definition of the Show instances for any syntactic categories you changed.

  If you added new syntactic categories, you will want to add new instances of the Show and Read classes for them.

  Don't forget to put your name in this file, and to change the comment at the top about what problem it's solving.

- Test your parser now. For example, at the prompt type something like

  ```
  read "loc 3 := 5" :: Command
  ```

  If you get the error "No parse" it may mean that you typed a syntax error in your test. If not, try something like

  ```
  scanner "loc 3 := 5"
  ```

  and look to see if all of what you think of as keywords are being recognized as such. If not, fix the definition of scanner. You can also run individual parsers on your input, like the following.

  ```
  expression (scanner "6 + 4 + 1")
  ```

  If this doesn't help see me; I don't want you to get bogged down in parsing.

- If you need to, copy and modify the type attributes (in `CoreTypeAttributes.lhs`). Also change the Show instance in that file if you made changes.

  Again, don't forget to change the comments.

- Copy and modify the type rules (in `CoreTyping.lhs`). For each typing rule you added or changed, add or change the annotate function declared in the appropriate instance of Annotatable.

  If you added a new syntactic category, then you will also have to change the definition of the data type Syntax, and the instances of Annotatable and Show for it, and you'll also have to add a new instance of Annotatable for your new syntactic category.

- For testing, it's easiest to put some test cases in a file. For example,

  ```
  module Problem_2_2_a where
  import CoreTypeHelpers
  import CoreTyping
  import CoreLangParser

  type_check :: String -ATree Command
  type_check = annotate . read

  pretty_type = show . type_check
  ```

```
test 1 = pretty_type "int loc 3 := 2"
test 2 = pretty_type "bool loc 2 := (1 = 2)"
```

Once this is done, you can run tests by typing Haskell expressions like

```
test 1
```

at the Haskell prompt.

4. Do problem 2.3 on page 24 of Schmidt's book. As in the previous problem, do this (a) on paper, and then (b) translate your solution to Haskell.

   (This might be a place to substitute some more interesting feature instead of trying this problem as written in the book.)

5. (suggested practice) Do problem 4.1 on page 25.

6. Do problem 6.1 on page 25 of Schmidt's book. Be sure to do each part 2 ways (once with the given store, and once without a store). This is to be done in Schmidt's notation. Your answer can be handwritten, although you may find it easier to manipulate this kind of calculation from inside a text editor.

   (You may want to check your work with the program.)

7. Do problem 6.2, parts a and b on page 25 of Schmidt's book. This is to be done in Schmidt's notation. Your answer can be handwritten.

8. (extra credit) Do problem 6.2, part c on page 25 of Schmidt's book.

9. (extra credit) Do problem 6.3 on page 25 of Schmidt's book. Do this (a) on paper, and then (b) translate your solution to Haskell.

10. Do problem 6.4 on page 25 of Schmidt's book. Do this (a) on paper, and then (b) translate your solution to Haskell. For error meanings, you can either use the Haskell error primitive, or define a data type to hold the error values.

11. (extra credit) Do problem 6.5 on pages 25-26 of Schmidt's book. Do this (a) on paper, and then (b) translate your solution to Haskell.

12. (extra credit) Do problem 7.2 on page 27 of Schmidt's book.

13. (extra credit) Do problem 7.3 on page 27 of Schmidt's book.

14. Do problem 8.1 on page 25 of Schmidt's book. For this you should only use the given store, as it makes no sense to do such a calculation without it. This is to be done in Schmidt's notation. Your answer can be handwritten, although you may find it easier to manipulate this kind of calculation from inside a text editor.

15. (extra credit) Do problem 8.2 on page 27-28 of Schmidt's book.

16. Do problem 8.4, part a, on pages 28-29 of Schmidt's book.

17. (extra credit) Automate, in Haskell or lambda Prolog, the semantics in problem 8.5 page 29 of Schmidt's book.

18. (extra credit) Do problem 8.5, part a, on page 29 of Schmidt's book.

19. Do problem 8.6, parts a-c, on pages 29-30 of Schmidt's book. Note that there is a typo in the "if" rule, and that the first $@loc_2$ in part (a) should be $@loc_1$. In part (c), the first $@loc1$ should be $@loc_1$ and the *do* should be **do**.

20. (extra credit) Do problem 8.6, part d, on pages 29-30 of Schmidt's book.

21. Select one of the alternatives of problem 9.2 of Schmidt's book, on page 30, and work the problem for that alternative. Do this both (a) on paper, in Schmidt's notation, and (b) in Haskell. Test your Haskell code. (You may want to do this in a group.)

    You will be showing your solution to this problem to other class members, so make sure it's clear.