CptS 580 Advanced Programming Languages
Midterm Exam
March 7, 2012

Instructions
Time limit: 2 hours – once you look at the exam you have 2 hours to complete it (honor system). I think that each of the 4 questions should take 15-25 minutes incorporating thinking, researching, and writing.
Due date: midnight, tonight, March 7. Turn it in by emailing it to me – hauser@eecs.wsu.edu, or putting it under my office door (EME-53)
Resouces: open book, open notes. You are encouraged to use the haskell system to check your answers when appropriate.
Questions: I expect to be in my office from 2:00 onward this afternoon. I will also be reading email into the evening; but don't expect answers to questions emailed to me after 9PM.

Turn in format: put your name on your exam paper at the top of each page. Please also write at the top of the first page, clearly labeled, the *time that you started the exam*, *the time that you finished*, and *the time that you turned it in under my door* (if using that method of submission).
Please start the answer to each *numbered* question on a new page – i.e. your submission should have at least 4 pages.

1. Grammars
a) Function application in Haskell is normally denoted by placing the function argument after the function name (juxtaposition), i.e. `f x`. Function application associates to the left, so `f1 f2 x` means `((f1 f2) x)`. Assuming that non-terminal `<t>` will generate (through some unspecified grammar whose details don't matter) any identifier, number, string, etc., give a grammar that generates this kind of function application Haskell expressions with the appropriate associativity.

b) Function application can also be expressed in Haskell using the `$` infix operator. `$` associates to the right so `f1 $ f2 $ x` means `(f1 (f2 x))`. Again using non-terminal `<t>` as before, give a grammar for function application expressions where the application is denoted by `$` having the correct associativity.

c) Now write a grammar that allows for both kinds of function application, preserving the associativities of `$` and juxtaposition and giving `$` lower precedence.

d) Finally, extend your grammar from c) to allow parenthesized sub-expressions, with the contents of parentheses having higher precedence than anything else.

2. Separate compilation
Describe the separate-compilation process for your preferred *compiled* language – Java, C, C++, C#, Haskell, ML, etc. – from source code to running program. Questions to help guide your answer: What are the tools involved? (compiler, etc.) What are the inputs and outputs of each of the tools? What information beyond the actual compiled code must be present in each of the files produced and consumed by the different tools?

3. Haskell type classes
Consider the Haskell data type give by

```
datatype Tree a = Leaf a | Interior (Tree a) (Tree a)
```

a) Give an instance declaration for `Tree` as a `Functor` and write the necessary function definition(s).
b) Discuss the possibility and usefulness of making Tree an instance of `Applicative`: what functions would have to be defined? describe plausible behavior of those functions keeping in the spirit of the `Applicative` type class (you don't have to write the function definitions – just describe what they should do). Do you think that making `Tree` an instance of `Applicative` is worthwhile?

4. Function application and composition

a) Assume that functions `f1`, `f2`, and `f3` are defined with appropriate types so that `f3 $ f2 $ f1 2` is type-correct. Show that `f3 . f2 $ f1 2` has the same value as `f3 $ f2 $ f1 2` by working out the value of each using the definition of function composition `(.)` and the definiton of `$`, reducing each to a parenthesized expression not involving either `.` or `$`.

b) The equality of the two expressions in part a) might lead you to believe that `f3 . f2` and `f3 $ f2` mean the same thing, which they most assuredly do not. Use the types of `.` and `$` and perhaps an example to explain the difference in meaning between `f3 . f2` and `f3 $ f2`.