CS 558: Programming Languages **Practice Exam** **March 12, 2014**

This is a practice. It has 5 questions. On the real exam you will be allowed one (8.5" x 11") page of notes.

---

1. **Short Answers** (5 points each = 30 points total.).

   (a) Explain the difference between static and dynamic typing. List at least one real language which uses each approach.

   (b) Explain the difference between over-riding a method and over-loading a function. List at least one real language which uses each approach.

   (c) What makes a function "higher-order". What issues arises (in languages with higher order functions) that requires some care when implementing higher-order functions?

   (d) What advantages does a two-space collector have over a mark-and-sweep collector. What disadvantages does a two-space collector have over a mark-and-sweep collector.

   (e) What is a composite type? Give three examples.

   (f) Explain the differences between parametric polymorphism (found in languages like ML and Haskell), and subtype polymorphism (found in languages like C++, Smalltalk, and Java).

2. **Type Inference** (7 points each = 21 points total).

   Below we outline several type inference rules for a language similar to E7. Each rule is made from one or more typing judgments (and other auxiliary judgments). A typing judgment has the following form.

   $\Gamma \vdash e : t$

   Where Γ is an environment mapping variable names to types. e is syntactic term of the language, and t is a type. A judgment states that the term e has type t in a given context Γ (which tells about the types of term variables that appear in e). We also have several auxiliary judgments.

   - $\Gamma(v) = t$, which states that the environment Γ maps the term variable v to the type t.
   - $\Gamma(x \Rightarrow t)$, describes an environment with the same behavior as Γ, except that it maps the variable x to t. If x was already mapped to some type s in Γ, then this old mapping for x is hidden in $\Gamma(x \to t)$

   Example rules for function application and anonymous functions are given below.

   $$\frac{\Gamma \vdash f : (a \to t) \quad \Gamma \vdash x : a}{\Gamma \vdash (f\ x) : t} \qquad \frac{\Gamma(x \Rightarrow t) \vdash e : t}{\Gamma \vdash (\lambda(x)e) : (a \to t)}$$

   Complete the following rules for pair creation, if-the-else, and while-loop.

$$\frac{}{\Gamma \vdash (\texttt{pair}\ x\ y) : t}$$

$$\frac{}{\Gamma \vdash (\texttt{if}\ e\ \texttt{then}\ x\ \texttt{else}\ y) : t}$$

$$\frac{}{\Gamma \vdash (\texttt{while}\ e\ \texttt{do}\ x) : t}$$

3. **Modules** (6 points each = 18 points).

   Primary purpose of a Module System is to divide large programs into (somewhat) independent sections, offering: (1) name space control, (2) an abstraction barrier, and (3) separate compilation.

   For each of these purposes, do three things: (a) Explain what it is. (b) Give examples or reasons why a programmer might want such control. (c) List issues that need to be addressed when implementing this purpose.

   - Name space control.
   - Abstraction Barrier.
   - Separate compilation .

4. **Small programs in language E8** (8 points each = 16 points total).

   Using the object oriented language E8, write object definitions for the following constructs. For each construct, provide 1 method that might be useful on such a construct As an example I have provided the object definition for *lists*, and provided the method *length*.

```
(val list
  (object
     (def nil
        (object (method length () 0)   ))

     (method cons (y x)
        (object (def head y)
                (def tail x)
                (method length () (x.length.+ 1))   ))
  ))
```

   (a) A data structure representing arithmetic expressions. There are 4 kinds of arithmetic expressions. Variables (like x), Constants (like 3), Additions (like $(x + 5)$) and Negations (like (-6)).

   (b) A data structure for 2-3-Trees. A 2-3-Tree has three forms. It is empty, in which case it stores no data and no sub trees. It is a Node2, in which case it stores 1 Integers and 2 sub trees. t is a Node3, in which case its stores 2 Integers and 3 sub trees.

5. **Language Types and Parameter Passing.** (15 points total)

   We have seen three different types of languages: imperative, functional, and object oriented. We have studied many forms of parameter passing mechanisms such as: call by value, call be reference, call by copy return, and call by name. It is a fact that some parameter passing mechanisms are more widely used in some types of languages than others. Discuss why this is so. Use examples where appropriate. Use only the space provided on this page. Think carefully about what you want to say. Short concise answers that get to the important issues will get more credit than long rambling ones.