CS 558: Programming Languages **Midterm Exam** **Feb. 11, 2014**

This exam has 5 questions. One (8.5" x 11") page of notes allowed. Answer all 5 questions in the space provided.

Student Name (neatly in block letters): ______________________________

1. **Short Answers** (3 points each = 30 points total.).

   (a) Explain the difference between a binding-site of a variable and a reference (use) of a variable. How many of each can a variable have?

   (b) What determines if an object is allocated statically, on the stack, or in the heap?

   (c) Explain the closest nested scope rule.

   (d) What is an alias? Why are they considered a problem in language design and implementation?

   (e) Explain the differences among *overloading*, *coercion*, and *polymorphism*.

   (f) List the principal uses of *goto*, and the structured alternatives to each.

(g) Why is sequencing an relatively unimportant form of control flow in Lisp?

(h) What are the advantages of making a loop index variable local to the loop it controls?

(i) What is the difference between formal and actual parameters?

(j) What is a higher order function? Give three examples we saw in class, the reading, or homework.

2. **Operational semantics** (6 points each = 26 points).

   Below we outline the operational semantics for an E4 like language These semantics describe a language much like the one we interpreted in HW4.hs, where the state of the machine contains both a stack and heap. Pairs are allocated in the heap at cnsecutive locations, and function parameters are allocated on the stack. The operational semantic judgment looks like

   $F; E; S; H; e \downarrow S; H; v$

   Where E is an environment mapping variable names to locations. F maps function names to their formal parameter and body (all function have exactly one argument). A location may be either on the stack or in the heap. S is the stack, mapping locations to values, and H is the heap, also mapping locations to values. e is an expression of the language, and v is a value (or result). We have several side judgments.

   - $push\ v\ S \rightarrow S'; l$ computes a new stack S', which is like S except with the new fresh location l on the top of the stack containing v.
   - $pop\ S \rightarrow S'$ which returns a new stack S', like S but with its top element removed
   - $alloc\ H_1\ n\ v_1 \cdots v_n \rightarrow l; H_2$ which allocates n fresh location in the Heap H_1 for values $v_1 \cdots v_n$. the location l is the address of the first of these consecutive n locations. and H_2 is this new heap..
   - $l \in H$ or $l \in S$, which tests if l is in the stack or heap.

- $S(l)$ or $H(l)$ which fetches the value a location l from the stack or heap, and $E(n)$ fetches the location for name n.
- $E + (v, l)$ returns a new environment, like E, except it now maps variable v to location l.
- $F(f) \to (p, b)$ looks up the formal parameter p and body b for a function f.

An example rule for function application is given below.

$$\frac{\begin{array}{c} F(f) \to (p, b) \\ F; E; S; H; x \downarrow S_1; H_1; v \\ push\ v\ S_1 \to S_2 \\ F; E; S_2; H_1; b \downarrow S_3; H_3; u \\ pop\ S_3 \to S_4 \end{array}}{F; E; S; H; (@fx) \downarrow S_4; H_3; u}$$

Complete the following rules for variables, pair creation, and pair access.

$$\frac{}{F; E; S; H; (\text{variable}) \downarrow}$$

$$\frac{}{F; E; S; H; (\text{pair } e_1\ e_2) \downarrow}$$

$$\frac{}{F; E; S; H; (\text{fst } e) \downarrow}$$

$$\frac{}{F; E; S; H; (\text{snd } e) \downarrow}$$

3. **Parameter passing mechanisms** (6 points each = 18 points).

   Choose three different parameter passing mechanisms. Discuss the advantages and disadvantages of each mechanism.

   - Mechanism 1 = ______________

- Mechanism 2 = ____________

- Mechanism 3 = ____________

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

   (a) Using the language Haskell, write a small function `filter` that is passed two arguments. The first is a predicate returning `True` or `False`, and the second is a list. The function should return all (and only) those elements that meet the predicate. For example the call `(filter even [1,2,3,4,5,6,7,8])` should return `[2,4,6,8]`.

   The call `(filter (\ x -> x > 5) [6,3,1,7])` should return `[6,7]`.

   The type of `filter` is `(a -> Bool) -> [a] -> [a]`.

   (b) Using one of the languages E3, E4, or E5, write a small function that computes all rotations of a list of any length. Use the list conventions we developed in class. I.e. a list is set of right-nested pairs ending in 0. For example the call `(@ rotate (@ list3 1 2 3))` on a list of length 3, returns the nested pairs `((1.(2.(3.0))).((2.(3.(1.0))).((3.(1.(2.0))),0)))`. You may assume any of the functions we saw in class or homework, including but not limited to null, cons. head, tail, isnil, list1 , list2, list3, and append.

   If these languages had types, the type of rotate would be `[a] -> [[a]]`.

   Hint: Write a helper function with two parameters, and use it to define rotate.

5. **Interpreters.** (10 points total)

   In this class we have stressed the role of definitional interpreters in the process of language design and study. Answer the following questions. What is a definitional interpreter? What are they used for? How can they be used as a pedagogical tool? In what way are they superior to a compiler? In what way are they inferior to a compiler. What features of the host language (the language one writes the definitional interpreter in) would you want?