Programming Languages [Fall 2019]
Test II

NAME: _____

Instructions:

1) This test is 7 pages in length.

2) You have 75 minutes to complete and turn in this test.

3) Short-answer and essay questions include a guideline for how much to write. Respond in complete English sentences and avoid using bulleted and itemized lists.

4) For full credit on ML-response questions, implementations must avoid the @ operator and run in linear (O(n)) time.

5) This test is closed books, notes, laptops, phones, smartwatches, friends, neighbors, etc.

6) Use the backs of pages in this test packet for scratch work. If you write more than a final answer in the area next to a question, circle your final answer.

7) Write and sign the following: "I pledge my Honor that I have not cheated, and will not cheat, on this test."

_____

_____

Signed: _____

1. [3 points]
At the level of detail discussed in class, what makes one programming language "lower level" than another? [1 sentence]

2. [9 points]
At the level of detail discussed in class, how do Turing Machines work? Include a description of what their transition functions input and output. [3-5 sentences]

3. [6 points]
Why are almost all dynamically typed PLs type safe? [2-4 sentences]

4. [4 points]
In class we discussed type safety as guaranteeing a relationship between the static and dynamic semantics. Explain this relationship. [1-2 sentences]

5. [10 points]
Function c takes, in Curried form, a list L and a value v and returns the number of elements of L equal to v.

a) What is the type of c?

b) Implement c according to the constraints of Assignment 1 (i.e., with recursion).

c) Implement c according to the constraints of Assignment 2 (i.e., with map/fold).

6. [10 points]
Function f takes a list of lists and returns a single list containing all the elements in the sublists. Function f's output must be in the same order as its input. For example, f [[1,2],[],[3,4]] returns [1,2,3,4].
a) Implement f according to the constraints of Assignment 1 (i.e., with recursion).

b) Implement f according to the constraints of Assignment 2 (i.e., with map/fold).

7. [9 points]
For each of the following ML expressions, write the expression's type, or, if the expression is ill typed, write "no type".

a) `fun i j k = j k (j,k)`

b) `fun i j k = j k (k,k)`

c) `fun i j k = i k (k=j)`

8. [12 points]
a) Define a λ_{UT} function that takes a Church numeral *N* and returns the Church boolean `true` if *N* is odd and `false` if *N* is even. You may use abbreviations for Church booleans and numerals when convenient (e.g., you may write `true` instead of λ`t.`λ`f.t`).

b) Using the normal-order strategy, trace the evaluation of applying your function from Part (a) to the Church encoding of the number 2. Show each step of the evaluation, underline redexes, and define and use abbreviations when convenient.

9. [25 points]
The language L from the theory assignments has the following first-order abstract syntax:

types τ ::= bool | $\tau_1 \times \tau_2$

exprs e ::= x | true | false | e_1 NOR e_2 | (e_1,e_2) | let val $(x_1,x_2) = e_1$ in e_2 end

a) Define higher-order abstract syntax for L, but *only show the cases that are used for variable and let expressions*. (Hint: Define FV, CAS, and alpha-equivalence rules.)

b) Prove the e_1 NOR e_2 case of the Weakening Lemma for L.

10. [12 points]
Prove the following case of the Progress Lemma for diML. Recall that Progress says that if $e{:}\tau$ then either $e{=}v$ (for some value v) or $e \rightarrow e'$ (for some expression e').

Case $\dfrac{e_1{:}\tau_1 \rightarrow \tau_2 \qquad e_2{:}\tau_1}{(e_1\ e_2){:}\tau_2}$

Assume that the Canonical-forms Lemma has already been proved. When your proof needs to refer to names of dynamic/static semantics rules, just make up reasonable names (e.g., D-Apply1 can be the first search rule for evaluating application expressions).

[Undergraduates stop here. The remaining problem is for graduate students.]

11. [8 points]
State the Substitution Lemma for diML. Then prove or disprove that the converse of the Substitution Lemma is also true for diML. Recall that, for a formula of the form "for all x, if F(x) then F'(x)", the converse says "for all x, if F'(x) then F(x)".