Programming Languages [Fall 2019]
Test III

NAME: ____

Instructions:

1) This test is 10 pages in length.

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

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

4) Use the backs of pages in this test packet for scratch work.

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

____

____

Signed: ____

1. [4 points]
Define a monomorphic function in ML. For full credit, define the function using as few characters as possible.

2. [4 points]
How does ML avoid having a NULL value? [2-3 sentences]

3. [4 points]
For each of the following ML expressions, write the expression's type, or, if the expression is ill typed, briefly explain why.

a) `fun f(x) = if f(x) then f else (fn x=>x)`

b) `fun f(x) = if true then (fn x=>if x<1 then nil else nil) else (fn x=>[true])`

[Undergraduates skip the following problem. Problem 4 is for graduate students.]
4. [5 points]
Explain the ML value restriction. Then define functionally equivalent ML expressions e and e' such that e violates the value restriction but e' does not.

5. [12 points]

An ML API has the limited documentation shown below. Although limited, the documentation provides enough information to understand and implement the functions. Implement each of these two functions *twice*, first according to the constraints of Assignment 1 (i.e., with recursion) and second according to the constraints of Assignment 2 (i.e., with map/fold).

a) `val find : ('a -> bool)->'a list->'a option  (*finds leftmost match*)`

b) `val partition : ('a -> bool) -> 'a list -> ('a list * 'a list)`
```
(*the returned pair has the true-returning elements in the first list*)
(*elements in the returned lists are ordered as in the argument list*)
```

6. [8 points]
Prove the standard type-safety corollary, assuming that Progress and Preservation have been already proved.

7. [10 points]

Building on our encodings of natural numbers as Church numerals, encode a multiplication operator into λ_{UT}. Then trace execution of 2x3 with the normal-order evaluation strategy. As we did in class, underline redexes and use abbreviations (e.g., by writing 2 instead of the full lambda expression encoding 2).

8. [10 points]
Using evaluation contexts, define dynamic semantics for the simply typed lambda calculus having base type unit. Define two versions of the dynamic semantics: first with call-by-name evaluation and second with full-beta evaluation.

9. [13 points]

a) Define alpha-equivalence for the simply typed lambda calculus having base type unit. Assume that definitions of free variables and capture-avoiding substitution already exist. As in class, do not define explicit rules for symmetry or transitivity.

b) Prove that your definition of alpha-equivalence is in fact symmetric.

Theorem. For all expressions e and e': if $e \equiv e'$ then $e' \equiv e$.

10. [12 points]

Let S be the call-by-value simply typed lambda calculus with base type unit. State the Progress and Preservation theorems for S, and for each theorem, prove one case—the one case in which the beta-step rule is analyzed or used. Make all the assumptions we made in class, including that all the standard helper lemmas have already been proved.

11. [10 points]
Let D be diML with constructs added—as defined in class—for *binary* product and sum, unit, and recursive types. (Recall that D uses *let* expressions to eliminate product types and *case* expressions to eliminate sum types.) Implement in D a function `rev` that takes an arbitrary list L of integers and returns L reversed. In your response, please feel free to define and use abbreviations, but avoid non-D syntax (e.g., refrain from using syntactic sugar). Hint: Mimic our implementation of list-reversal in ML using difference lists.

12. [13 points]
Language U is the call-by-value simply typed lambda calculus having unit as the base type and having return expressions, as discussed in class. Define the first-order abstract syntax and complete static and dynamic semantics for U, as would be appropriate for a proof of type safety.