Programming Languages [Fall 2017]
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 questions include a guideline for how much to write. Respond in complete English sentences.

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

5) 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.

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

____

____

Signed: ____

1. [10 points]
What is call-by-need evaluation? Hit all the main points covered in class. [1 paragraph]

2. [13 points]
In SML, implement a function getPairs that takes a list L (which you should assume contains no duplicates) and returns a list of all pairs that can be built from elements of L. For example, getPairs [1,2,3] returns [(1,1),(1,2),(1,3),(2,1),(2,2),(2,3),(3,1),(3,2),(3,3)]. The order of the elements in the returned list *does* matter. The expected type of getPairs is α list $\rightarrow$ ($\alpha \times \alpha$) list. It should be clear that the proper running time for getPairs is $O(n^2)$; to help with attaining this running time, your response may not use the @ operator.

Your implementation should be according to the constraints of Assignment II, meaning: don't use the keywords let or local, and all recursion must be in calls to map or fold.

3. [22 points]

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

4. [55 points]
Consider a language C with case expressions. Here's the first-order abstract syntax:

Types	t::= nat | int
Patterns	p::= n | i | x
Expressions	e::= case e_1 of p=>e_2 else e_3 | n | i | x

where x is a variable (identifier), n is a nat-literal, and i is an int-literal. The C expression *case* e_1 *of p*=>e_2 *else* e_3 is equivalent to the SML expression *case* e_1 *of p*=>e_2 | _=>e_3.

a) Formally define the higher-order abstract syntax of C.

C is:

Types	t::= nat | int
Patterns	p::= n | i | x
Expressions	e::= case e_1 of p=>e_2 else e_3 | n | i | x

b) Formally define static and dynamic semantics for C.

C is:

Types	t::= nat | int
Patterns	p::= n | i | x
Expressions	e::= case e_1 of p=>e_2 else e_3 | n | i | x

c) Is C Turing complete? Explain how you arrived at your answer.

d) Recall that the Weakening Lemma says: $(\Gamma_1 \vdash e{:}\tau \ \wedge\ \Gamma_1 \subseteq \Gamma_2) \Rightarrow \Gamma_2 \vdash e{:}\tau$
Prove this lemma for C. You don't need to show the complete proof; just state the proof technique and show one inductive case of the proof.

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

e) [12 points]
Besides the Weakening Lemma, what are the 3 other standard lemmas we used to prove type safety? State the 3 lemmas formally and provide the proof technique for each.