UCLA CS 131 Midterm, Fall 2017
100 minutes total, open book, open notes
closed computer

Name:_____ Student ID:_____

1	2	3	4	5	6	total

1a (10 minutes). Write an OCaml function merge_sorted that merges two sorted lists. Its first argument should be a comparison function 'lt' that compares two list elements and returns true if the first element is less than the second. Its second and third arguments should be the lists to be merged. For example, (merge_sorted (<) [21; 49; 49; 61] [-5; 20; 25; 49; 50; 100]) should yield [-5; 20; 21; 25; 49; 49; 49; 50; 61; 100].

1b (3 minutes). What is the type of merge_sorted?

1c (3 minutes). What does the following expression yield, and what is its type?

```
merge_sorted (fun a b -> List.length a < List.length b)
```

1d (8 minutes). Is your implementation of merge_sorted tail-recursive? If so, briefly say why it won't have any problem with stack overflow. If not, briefly say why not, and explain any problems you would have in rewriting your implementation to make it tail-recursive.

2 (9 minutes). Consider the following top-level OCaml definitions:

```
let f f = f 1 1
let g g = g 0.0 g
let h h = h f "x"
```

For each identifier declared in this code, give the identifier's scope and type. Or, if there is a scope or type error, briefly explain the error.

3a (5 minutes). In Java, is the subtype relation transitive? That is, if A is a subtype of B and B is a subtype of C, is A a subtype of C? If so, explain why; if not, give a counterexample.

3b (5 minutes). In Java, is the graph of the subtype relation a tree? If so, explain why, and say what the root is; if not, give a counterexample.

4. Consider the following grammar for declarations in a subset of C. Ths grammar uses a form of EBNF in which the left hand side is not indented and is followed by ":", each right hand alternative is indented, nonterminals are strings of letters and "-", terminal symbols are either surrounded by single quotes or are INT (meaning an integer constant) or ID (meaning an identifier), and X? stands for zero or one instances of X.

```
declaration:
  declaration-specifiers init-declarator-list? ';'
```

```
  declaration-specifiers:
    storage-class-specifier declaration-specifiers?
    type-specifier declaration-specifiers?
    type-qualifier declaration-specifiers?
    function-specifier declaration-specifiers?

  storage-class-specifier:     type-specifier:
    'typedef'                    'void'
    'static'                     'char'
                                 'int'
  type-qualifier:
    'const'                    function-specifier:
    'volatile'                   'inline'
                                 '_Noreturn'
  init-declarator-list:
    init-declarator
    init-declarator-list ',' init-declarator

  init-declarator:
    declarator
    declarator '=' initializer

  declarator:
    pointer? direct-declarator

  direct-declarator:
    ID
    '(' declarator ')'
    direct-declarator '[' INT ']'
    direct-declarator '(' 'void' ')'

  pointer:
    '*' type-qualifier-list? pointer?

  type-qualifier-list:
    type-qualifier-list? type-qualifier

  initializer:
    ID
    INT
```

4a (2 minutes). What makes this grammar EBNF and not simply BNF?

4b (8 minutes). Give an example declaration that is syntactically correct (i.e., it is produced by this grammar) but is semantically incorrect for C. Prove that it is syntactically correct. Briefly explain why it is semantically incorrect.

4c (5 minutes). Suppose we changed the grammar by replacing the ruleset for type-qualifier-list with the following:

```
  type-qualifier-list:
    type-qualifier type-qualifier-list?
```

Would this cause any problems? If so, describe a problem and give an example. If not, briefly explain why not.

4d (10 minutes). Suppose we changed the original grammar by replacing the two rulesets for declarator and direct-declarator with the following single ruleset:

```
  declarator:
    pointer? declarator
    ID
    '(' declarator ')'
```

```
    declarator '[' INT ']'
    declarator '(' 'void' ')'
```

Would this cause any problems? If so, describe a problem and give an example. If not, briefly explain why not.

4e (10 minutes). Draw a syntax chart for the original grammar.

5 (10 minutes). Suppose we write Java code in a purely functional style, in that we never assign to any variables except when initializing them. That is, we always initialize local variables and never assign to them later, and we always initialize instance variables once at the start of constructors and never assign to them later.

In our purely-functional Java programs, is the Java Memory Model still relevant, or can we ignore it? If it's still relevant, explain which parts of it still apply and give an example. If not, briefly explain why not.

6 (12 minutes). Consider the following code, taken from the answer to the older version of Homework 2.

```
  let match_empty frag accept = accept frag

  let match_nothing frag accept = None

  let rec match_star matcher frag accept =
    match accept frag with
      | None ->
          matcher frag
                  (fun frag1 ->
                     if frag == frag1
                     then None
                     else match_star matcher frag1 accept)
      | ok -> ok

  let match_nucleotide nt frag accept =
    match frag with
      | [] -> None
      | n::tail -> if n == nt then accept tail else None

  let append_matchers matcher1 matcher2 frag accept =
    matcher1 frag (fun frag1 -> matcher2 frag1 accept)

  let make_appended_matchers make_a_matcher ls =
    let rec mams = function
      | [] -> match_empty
      | head::tail -> append_matchers (make_a_matcher head) (mams tail)
    in mams ls
```

In this code, a matcher is a curried function taking two arguments: first, a fragment 'frag' and second, an acceptor 'accept'. Suppose we change the API for matchers by interchanging their arguments, so that the acceptor comes first (all the functions remain curried). Rewrite the above code to use the altered API, and simplify the resulting code as much as possible.