UCLA Computer Science 131 midterm - Spring 2020

Name:_____

Student ID:_____

90 minutes total, 1 minute = 1 point. Open book, open notes, open computer. Answer all questions yourself, without assistance from other students or outsiders.

The exam is not easy, and you are not expected to answer all the questions completely. In your answers, overall approach and intuition will count more than trivial detail. Budget your time while taking the exam. It may help to skip questions that are harder than their point count would suggest.

Print the exam, read the first page, then write your starting time on the first page. Then take at most 90 minutes to answer the questions and write your answers on the exam. (CAE students with x% extra time should add to the 90 minutes accordingly.) When you're done, write your finishing time on the first page, sign the first page, scan the completed exam, and upload your scans to CCLE Gradescope as quickly as you can. If you lack a scanner, carefully photograph the sheets of paper with your cell phone and upload the photographs. Save your filled-out exam until the class is over, and do not give or show it to anybody other than an instructor or TA.

If you lack a printer, read the exam on your laptop's screen, write your answers on blank sheets of paper (preferably 8½″×11″) with one page per question, and upload the scanned sheets of paper. At the end of the exam, you should have scanned and uploaded as many photographs as there are questions. If you do not answer a question, scan a blank sheet of paper as the answer. You can type your answers if you like; all we need is something that you can upload on Gradescope as a PDF.

The exam is open book, open notes, and open computer. You can use your laptop to use a search engine for answers, and to run programs designed to help you answer questions. However, do not use your computer or any other method to communicate with other students or outsiders, or anything like that. Communicate only via CCLE and Gradescope to obtain your exam and upload your scanned results, or via Zoom or email with the instructor or TAs.

_____ Time (Los Angeles time) you started the exam

_____ Time that you ended the exam

IMPORTANT Before submitting the exam, certify that you have read and followed the above rules by signing and dating here:

Signature:_____ Date:_____

1 Explain each of the diagnostics in the following OCaml session.
Also, for each diagnostic, propose a minimal extension to OCaml that would mean the program was not erroneous, and give a downside for your proposal that helps to explain why OCaml made it erroneous.

1a (3 minutes).

```
        if 2 + 2 = 4 then print_endline "OK" else "BAD";;
                                                 ^^^^^
    Error: This expression has type string but an expression was
             expected of type unit
```

1b (4 minutes).

```
        let a x = x a;;
                    ^
    Error: Unbound value a
    Hint: If this is a recursive definition,
    you should add the 'rec' keyword on line 1
```

1c (5 minutes).

```
    let f = function | a::_, __ -> a | _,b -> [[b]] | _,[[c]] -> c;;
                                                              ^
    Error: This expression has type 'a but an expression was expected
           of type
             'a list list list list
           The type variable 'a occurs inside 'a list list list list
```

2 (6 minutes). Consider the following two OCaml function definitions:

```
    let rec a = fun x -> x a
    let rec b = fun x -> fun y -> y x
```

One is valid, the other is not. Explain why, both in terms of types and in terms of what the functions mean. Give a use of the valid function, and simplify and shorten its definition as much as you can.

[page 4]
3 (10 minutes). Suppose you have a working solution to Homework 2.
Define a *generous matcher* to be like a Homework 2 matcher, except
that its argument is a list of acceptors instead of being a single
acceptor. A generous matcher will match any prefix such that the
corresponding suffix is accepted by any of the acceptors in the list.
The generous matcher tries the acceptors in list order, and returns
the result of the first acceptor that succeeds, or None if none of
them succeed. Define a function (make_generous_matcher G) that
returns a generous matcher for G. Your implementation should be as
simple and clear as possible. It can use the functions that already
exist in a working implementation of Homework 2.

4 (15 minutes). Write an OCaml function (revhalf L) that returns a copy of L, except that its even-numbered elements (counting from 1), if any, are in reverse order. For example, revhalf [1;2;3;4;5;6;7] should return [1;6;3;4;5;2;7]. As much as possible, avoid creating objects; and keep your solution as simple and elegant as you can. You may use the Pervasives and List modules, but no other modules.

5. Consider the following EBNF grammar for a subset of Java. This grammar is a simplified version of the full Java grammar. In this grammar, [X] means zero or one occurrences of X, and {X} means zero or more occurrences of X; in both of these metanotations there are no spaces between the "X" and the surrounding brackets. Nonterminals are capitalized, and terminals start with a lower-case letter or are individual symbols like ';'. The start symbol is Statement.

```
  Statement:
    Block
    ;
    id : Statement
    Expression ;
    if ParExpression Statement else Statement
    assert Expression [: Expression] ;
    while ParExpression Statement
    do Statement while ParExpression ;
    break [id] ;
    continue [id] ;
    return [Expression] ;
    throw Expression ;
    try Block Catches
    try Block [Catches] Finally

  Catches:
    CatchClause {CatchClause}

  CatchClause:
    catch ( CatchType id ) Block

  CatchType:
    id {| id}

  Finally:
    finally Block

  Block:
    { BlockStatements }

  BlockStatements:
    {Statement}

  ParExpression:
    ( Expression )

  Expression:
    id
```

5a (5 minutes). Translate this grammar from EBNF to straight BNF. To save time you can write your translation atop the existing grammar, replacing only the parts that need translation.

[continued on next page]

5b (4 minutes). Would your straight-BNF grammar work with a working solution to Homework 2? Briefly explain why or why not.

For each of the following changes to the grammar, state whether the change would make the grammar ambiguous. Consider each change independently. If the change would make the grammar ambiguous, prove it; if not, state why not.

5c (3 minutes). Remove the “(” token from the ParExpression rule.

5d (5 minutes). Change the parentheses tokens “(” “)” in the ParExpression rule to brace tokens “{” “}”.

5e (5 minutes). Remove the “finally” token from the Finally rule.

6. Suppose Java's 'wait' and 'notifyAll' primitives were implemented as follows:

```
   private boolean flag;
   void wait(void) { flag = true; while (flag) continue; }
   void notifyAll(void) { flag = false; }
```

and suppose a Java program uses this implementation and never uses the 'notify' primitive.

6a (3 minutes). Explain why this implementation would not be correct for this usage, due to a potential race condition.

6b (5 minutes). Suppose we assume that our program is free of race conditions so that the correctness problem mentioned in (a) cannot occur. Explain why this implementation has a significant efficiency problem that is independent of the correctness issue.

6c (5 minutes). Suppose the 'synchronized' keyword were placed before the 'wait' and 'notifyAll' implementation shown above. Would this fix the correctness and/or efficiency problems? Explain.

7. You are optimizing a large Java server app to run on lnxsrv10.seas.ucla.edu, an Intel Xeon Silver 4116 which has 12 cores, each with (dual) hyperthreading, 32 KiB 8-way L1 I cache, a similar L1 D cache, and a 1 MiB L2 cache; and which has 16.5 MiB shared L2 cache and 64 GiB DDR4-2400 DRAM. All caches are write-back.

Your app is using a just-in-time Java compiler that operates via a machine-code cache M, i.e., an area of memory containing machine-code translations of your app's byte-code methods. Your JVM lets you place a limit on M's size; by default there is no practical limit, but before your JVM starts up you can specify a limit MLIM to M's size, and if you do the JVM will periodically discard rarely-used machine code from the cache so that the cache size stays under MLIM.

7a (4 minutes). Explain why it would not make much sense to set MLIM to 32 KiB even though that would mean M would fit in the L1 I cache.

7b (8 minutes). Give a scenario where it would make sense to set MLIM to 4 MiB, and give another scenario where it make sense to set MLIM to 48 MiB. Your scenarios should contrast the type of code and data accesses the two scenarios have, and why the stated MLIM values are appropriate for each scenario.