CSE305 **Sample Final Exam** **Spring 2023**

One notes binder allowed, otherwise no book, no electronics, closed neighbors, 170 minutes. Do *all five* questions in the exam booklets provided. The exam totals 200 pts., subdivided as shown. *Show your work*, and explain your reasoning where it is called for—doing so may help for partial credit.

(1) (3+0+6+9+15+6 = 39 pts.)

Consider the following function written in OCaml

```
let rec findDoubles(ell) = match ell with
    [] -> []
  | x::[] -> []
  | x::y::rest -> if x = y then x::findDoubles(rest)
                           else findDoubles(y::rest)
;;
```

(a) Calculate `findDoubles [1;3;3;3;5;5;2;2;2;2;6;5;5]`. (3 pts.)

(b) Suppose your original intent was to make a list `findRepeaters(L)` of all the repeating elements, so that you'd get `[3;5;2;5]` instead of your actual answer to (a). How would you modify the code? Wait—don't answer this yet. (0 pts.)

(c) Make a tail-recursive "helper function" `fdh(L,R)` that does the same thing as `findDoubles`, except that the output list is accumulated onto the given list `R`. (6 pts.)

(d) Now modify your `fdh` to answer part (b). Did using your answer to (c) make things easier than your first thoughts in (b), and if so, how? (9 pts.)

(e) Now write a predicate `findDoubles(L,R)` in Prolog, where `L` is given and the output is to be stored in `R`. You may use the built-in equality predicate `=`, but must use a cut in place of also using the inequality predicate `\=`. (15 pts.)

(f) Now in OCaml, you can use the built-in `option` datatype:

```
type 'a option = None | Some of 'a
```

to make `findFirstDouble(L)` return `None` if `L` has no doubles, or `Some d` if `d` is the first double. But why might this be annoying? Write the OCaml type signature of `findFirstDouble` as part of your answer. (6 pts.)

(2) (1+12+9+6+9+12 = 49 pts.)

Consider the following expression, in the syntax of C/C++/Java/C♯. Note that it has an internal assignment—recall that assignment is treated as a binary expression operation in all of these languages.

```
z = (x - y + z) / (y = x + 2) * (y - 3*z)
```

And recall part of the standard BNF for expressions in these languages:

```
E1 ::=  Var = E1 | E
E  ::=  E + T | E - T | T
T  ::=  T*F | T/F | F
F  ::=  (E1) | Var | any_numeric_literal
Var::=  x | y | z | etc.
```

(a) Evaluate the expression when `x = 4`, `y = 6`, and `z = 2`. (1 pt.)

(b) Write a parse tree for the above expression in this grammar. You may abbreviate some productions, e.g. `F` can be a direct child of `E1`. (12 pts.)

(c) Now write an expression tree for the expression itself. (9 pts.)

(d) Use your tree in (c) to convert the expression into Postfix form. (6 pts.)

(e) Now compile the Postfix into our "rudimentary stack language." (9 pts.)

(f) Show the evaluation of your stack code in (e) when `x = 4`, `y = 6`, and `z = 2`. (12 pts.)

(3) (12+9+6+6+6 = 39 pts.)

The program overleaf is written in a mythical language that mixes syntax from Scala and Javascript and C/C++/Java, plus mimics Ada in that the body of a subprogram comes after any nested subprogram definitions.

```
def Main()
    int k,x
    def A(int y) return int
        def B(int x) return int
            return x*(y + 1)
        end B
        def C(int y) return int
            int k = 2*y
            return B(k)
        end C
    /* Main body of A begins here */
    begin A
        if y = 1 then return C(3*y)
                 else return A(y - 1)
        end if
    end A
    def D(int x)
        Console.WriteLine("Answer is: " + (x + k))
    end D
/* Body of Main() begins here. */
begin Main
    k = 3
    x = A(k)
    D(x)
end Main
```

(a) For each of the four sub-programs `A,B,C,D`, draw up a table showing which of the three variable names `k`, `x`, and `y` are visible within that sub-program, and if so, which block it belongs to (i.e., was declared in). For example, the table for `Main` is: `k = Main.k`, `x = Main.x`, but no `y` is visible. (12 pts.)

(b) Trace out the allocation of stack frames during the execution of this program. Show the static and dynamic links from each frame. (Everything except the particular values of variables within the frames is independent of parts (c) or (d); you should be able to use this trace to help you solve (c) and (d) without having to re-draw it each time. 9 pts.)

(c) Suppose the language uses static scoping. Using the information in your answers to (a) and (b), carefully work out the value printed by this program. Show your work. (6 pts.)

(d) Now suppose the langauage is using dynamic scoping. Work out the value that would be printed now. (Showing where a value would change is enough for full credit. 6 pts.)

(e) Finally, and going back to static scoping, suppose that the header of `A` were `def A(int& y)` for call-by-reference. Would the program be legal? If you say no, indicate the kind and place of error that would be reported by the compiler. If you say yes, trace what happens to the storage objects `k` and `y` as it is run. (6 pts.)

(4) (9+9 = 18 pts.)

Consider the following Prolog facts and rules:

```
father(abou,barak).
father(abou,filia).
father(barak,nina).
mother(filia,mara).
mother(mara,haroun).

%% X is a parent of Y
parent(X,Y) :- mother(X,Y).
parent(X,Y) :- father(X,Y).

%% X is a grandfather of Z
grandfather(X,Z) :- father(X,Y), parent(Y,Z).
```

(a) What does Prolog print in response to the following two queries? Show all answers, i.e., as if you kept entering `;` at the prompt. (9 pts. total)

```
| ?- grandfather(abou, mara).
| ?- grandfather(abou, haroun).
```

(b) Using a cut, write a predicate `hasMissingParent(X)` that matches all X for which exactly one parent is listed in the current database. (Alternatively you may use the built-in Prolog predicate `not(p)` where `p` can be a Prolog predicate, but cut is easier. 9 pts.)

(5) ($4 \times 4 = 16$ pts.)

True/False **with** *justifications*. Please write out the words `true` and `false` in full (2 pts.) and then give a brief justification (2 pts.).

(a) Upon execution of a pointer assignment `p = q`, the value of `p` is the address of the storage object `q`, so that `p` now points at `q`.

(b) In Java, the loop variable `i` in the for-loop `for (int i = 1; i <= 15; i++) {...}` has value `i = 16` after the loop exits.

(c) On executing a statement `Foo x = new Foo();` in C# or Java (or without the semicolon in Scala), where `Foo` is a class, a storage object is created on the system stack as well as on the system heap.

(d) In OCaml, in a function call of the form `f(X,Y)` where `X` and `Y` stand for arbitrary expressions, both `X` and `Y` are evaluated before the body of `f` is executed.

(6) (15+18+6 = 39 pts.)

Consider the following OCaml datatype for the representation of assignment statements and conditional control structures, which could apply to many programming languages not just C/C++/Java. Here we assume that datatypes `'a exp` and `'a lvalue` have already been defined to model expressions and targets of some unknown type `'a`, and this problem does not depend on details of their definition.

```
type 'a stmt = Assign of 'a lvalue * 'a exp
  | IfThen of bool exp * 'a stmt
  | IfThenElse of bool exp * 'a stmt * 'a stmt
  | Block of 'a stmt list;
```

(a) For each OCaml term, say yes/no whether it is a legal construct of this datatype, i.e., whether OCaml compiles it and gives it an inferred type. For example, assuming `c,x,y,z` are bound to appropriate values, `IfThenElse(c,Assign(x,3),Assign(y,z))` is legal, and OCaml will give it type `int stmt`. Moreover, in pattern matching, OCaml will match `c` to values of type `bool exp`, `x` and `y` to `int lvalue`, and `z` to `int exp`. (3 pts. each)

1. `Block [Assign(x,3), Assign(y,4)]`
2. `Block [Assign(x,"three"), Assign(y,4)]`
3. `IfThen(c, IfThenElse(d, Assign(x, 3.0), Assign(x, 4.0)))`
4. `IfThenElse(c, IfThen(d, Assign(x, 3.0)), Assign(x, 4.0))`
5. `IfThenElse(c, Block [], Assign(x, true))`

(b) Write an (E)BNF grammar G with start symbol `ASTMT` such that $L(G)$ equals the set of all legal constructs of this datatype. You may use nonterminals `AEXP`, `BOOLEXP`, and `ALVALUE` without expanding them further, and assume they derive all legal constructs of types `'a exp`, `bool exp`, and `'a lvalue` (including identifiers and constants as shown above). Treat `Assign, IfThen, IfThenElse, Block` and the punctuation `( [ , ] )` as terminal tokens, and (optionally) introduce a new nonterminal to generate a list. (18 pts.)

(c) Is your grammar G ambiguous? Why or why not? (6 pts.)

End of Exam