CSC 151 INTRODUCTION TO PROGRAMMING LANGUAGES

SPRING 2020 FINAL WRITTEN ASSESSMENT

Instructions:

This is an open-book assessment. You may use the textbook, class notes, and any resources linked from the course home page in answering the questions. You have three options to choose from.

Option A. Take the exam on your own, without any assistance from others. Under this option, the exam is due on May 8 at midnight. (All deadlines for this assignment are EDT.)

Option B. Participate in a group exam process. Under this option, you must turn in your individual effort on the exam by Monday, May 4 at midnight. You will then be assigned to a group, and groups will have until Wednesday, May 6 at midnight to submit a single revised solution. Finally, the entire set of individuals who have chosen Option B will collaborate to produce their best combined solution by Friday, May 8 at midnight. Aim to spend 2-4 hours on each stage. Group work may be conducted via Zoom or by document sharing. For grading, I will randomly choose two questions (consistent across the class) to grade for each phase.

Option C. Waive the exam. I will count this as a C in computing your final grade. (Remember that like all others, this class is S/U for spring 2020. For those who choose options A or B, the lowest possible grade you will receive on this assessment is also a C.)

I. Vocabulary

Categorize each of the four main languages we have worked with (Python, OCaml, C, and Java) according to the following distinctions.

a.) Statically typed vs. dynamically typed
b.) Object-oriented vs. non-object-oriented
c.) Active memory management vs. passive management
d.) Primarily imperative vs. primarily functional

II. Type Inference

For each OCaml value below, fill in the blank for the type annotation or else write "ill typed" if there is a type error on that line. Your answer should be the *most generic* type for the value—i.e. if **`int list`** and **`bool list`** are both possible types of an expression, you should write **`'a list`**.

The first one is done for you.

```
let z : _________ 'a list list ________ = [[]]

let a : ___________________________________ = true::false

let b : ___________________________________ = [] :: [] :: []
```

```
let c : ____________________________ = begin match [] with
                                         | [] -> "foo"
                                         | h::t -> h+3
                                       end

let d : ____________________________ = transform (fun x -> x>2)

let e : ____________________________ = fold (fun x y -> x::y)

let f : ____________________________ = (fun x -> fun y -> x) 42

let g : ____________________________ = let f x = [x] in f (f 5)

let h : ____________________________ = let sum x y = x+y in
                                       transform (transform sum)
```

(Note: in the above, recall our definitions of transform and fold as shown below.)

```
let rec transform (f : 'a -> 'b) (ls : 'a list) : 'b list =
  begin match ls with
    | [] -> []
    | h::t -> (f h) :: (transform f t)
  end

let rec fold (combine: 'a -> 'b -> 'b) (base: 'b) (l: 'a list): 'b =
  begin match l with
  | [] -> base
  | h :: t -> combine h (fold combine base t)
  end
```

III. Conditional Expressions

Consider the Java code below. Underline the calls to **`getVal()`** that will actually be executed when the following code runs.

```
public class Cond {
    private int value;
    public Cond(int v) {
        value = v;
    }
    public int getVal() {
        return value;
    }
    public static void main(String[] args) {
        Cond c = new Cond(7);
        if ((c.getVal()<8)&&(c.getVal()>6)) {
            System.out.println(c.getVal());
        }
        c = new Cond(5);
        if ((c.getVal()<8)||(c.getVal()>6)) {
            System.out.println(c.getVal());
        }
        c = new Cond(3);
```

```
            if ((c.getVal()>8)&&(c.getVal()<6)) {
                System.out.println(c.getVal());
            }
            c = new Cond(6);
            if ((c.getVal()>8)||(c.getVal()<6)) {
                System.out.println(c.getVal());
            }
            c = new Cond(2);
            if (((c.getVal()>8)||(c.getVal()<6))&&((c.getVal()<5)||(c.getVal()<9))) {
                System.out.println(c.getVal());
            }
            c = new Cond(8);
            if (((c.getVal()>2)&&(c.getVal()<6))||(c.getVal()>4)||(c.getVal()<3)) {
                System.out.println(c.getVal());
            }
        }
    }
```

IV. Iteration

Consider the pseudocode below in answering the questions that follow. Assume that the value of **n** has been set previously. It may have any integer value.

```
i = 0
print i
while i < n do
    i = i+1
    print i
end
```

a.) Rewrite to get the same behavior using **repeat…until**
b.) Rewrite to get the same behavior using **do…while**
c.) Rewrite to get the same behavior using a Python-style **range** loop
d.) Rewrite to get the same behavior using a **while true do…** loop with a mid-loop **break** to exit.

V. Subtypes

Consider the code below in answering the questions.

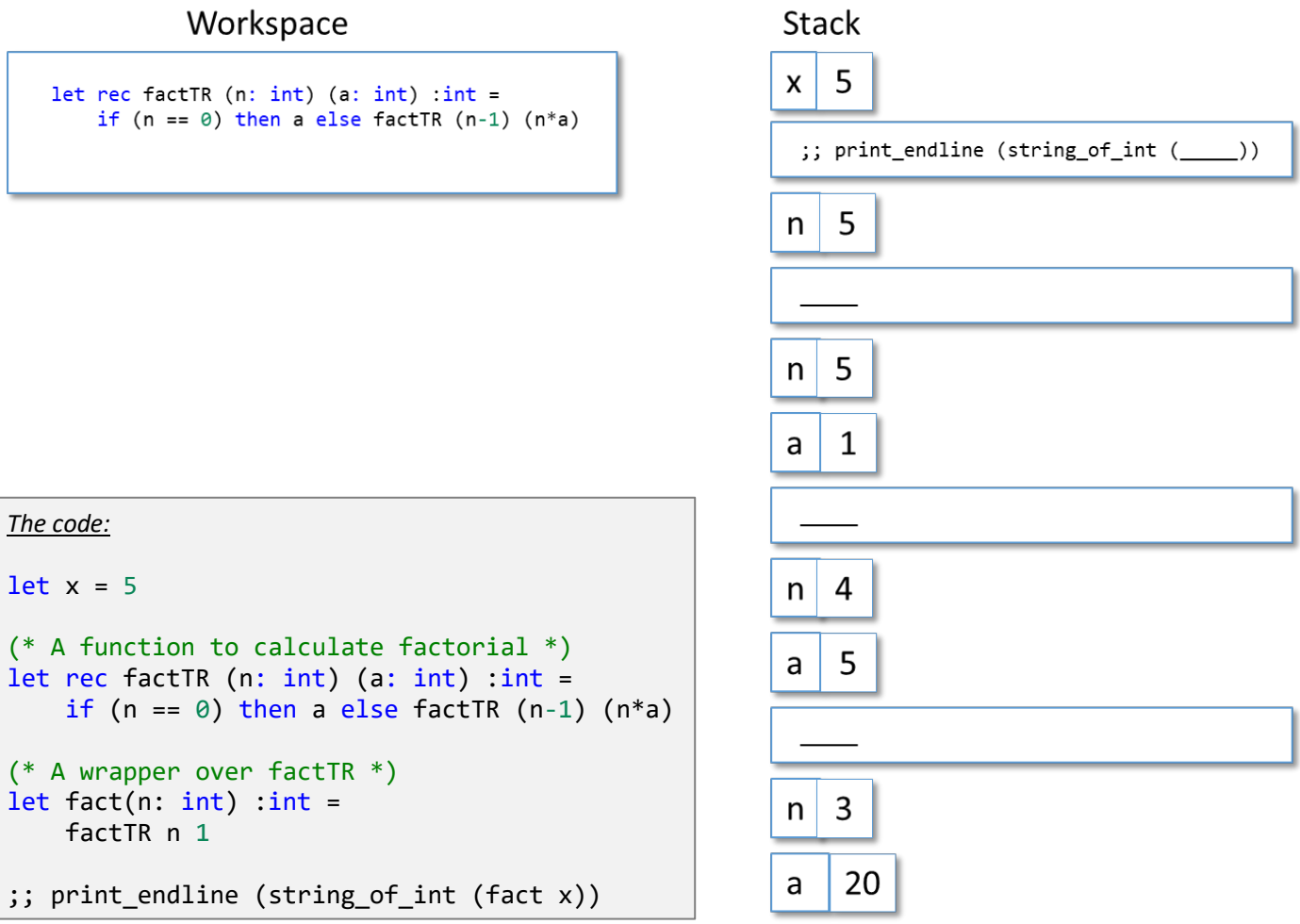
a.) What are the subtypes of Alpha?
b.) What are the subtypes of Beta?
c.) What are the subtypes of Gamma?
d.) What are the subtypes of Delta?
e.) What are the supertypes of Theta?
f.) What are the supertypes of Lambda?

```
class Alpha {
    // details omitted
}
```

```
class Beta {
    // details omitted
}

interface Gamma {
    // details omitted
}

interface Delta {
    // details omitted
}

class Epsilon extends Alpha {
    // details omitted
}

class Zeta extends Beta implements Gamma {
    // details omitted
}

class Eta implements Delta {
    // details omitted
}

class Theta extends Zeta implements Delta {
    // details omitted
}

class Iota extends Beta implements Delta {
    // details omitted
}

class Kappa extends Iota {
    // details omitted
}

class Lambda extends Epsilon implements Gamma, Delta {
    // details omitted
}
```

VI. Computational Models

a.) The diagram below shows one state of the abstract stack machine during execution of some OCaml code. Circle all the bindings that are accessible in the current scope.



The code:

```
let x = 5

(* A function to calculate factorial *)
let rec factTR (n: int) (a: int) :int =
    if (n == 0) then a else factTR (n-1) (n*a)

(* A wrapper over factTR *)
let fact(n: int) :int =
    factTR n 1

;; print_endline (string_of_int (fact x))
```

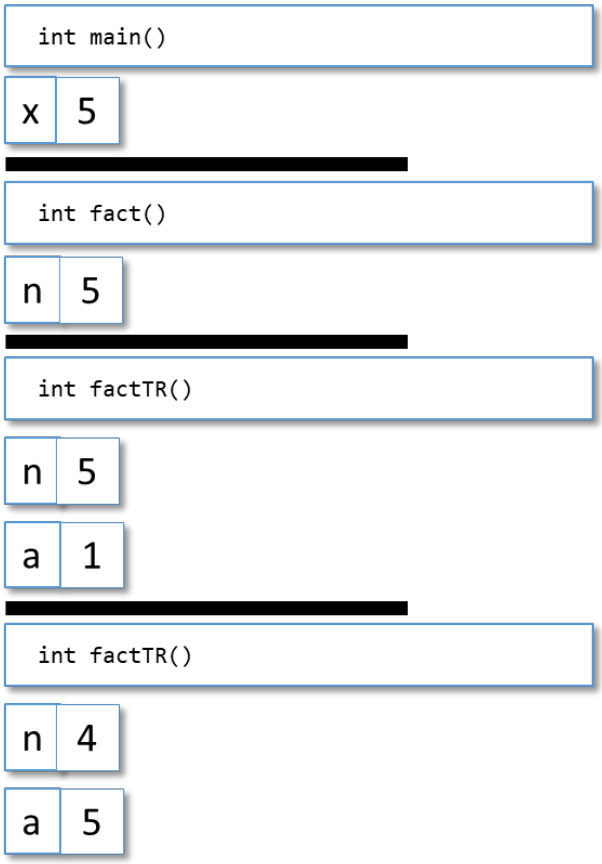
b.) If the compiler emplys a tail recursion optimization, the stack would look different. Draw the state of the stack under this assumption.

c.) The diagram below shows one state of the abstract stack machine during execution of some C code. The current line is underlined. Circle all the bindings that are accessible in the current scope.

Workspace

```c
#include<stdio.h>

// calculate factorial
int factTR(int n, int a) {
    if (n == 0)  return a;
    return factTR(n-1, n*a);
}

// A wrapper over factTR
int fact(int n) {
   return factTR(n, 1);
}

// Driver program to test
int main() {
    int x = 5;
    printf("%d\n",fact(x));
    return 0;
}
```

Stack



VII. Certification

When you have finished your exam, please sign the following statement:

In compliance with the Smith College Honor Code, I have completed this assessment without any assistance or consultation, with the exception of group work explicitly permitted in the instructions.

_____