Final Exam
CS 1103 Computer Science I Honors
Fall 2016

KEY

Friday December 16, 2016

Instructor Muller
Boston College

Before reading further, please arrange to have an empty seat on either side of you. Now that you are seated, please write your name **on the back** of this exam.

This is a closed-notes and closed-book exam. Computers, calculators, and books are prohibited.

This is a 30 point exam.

- Partial credit will be given so be sure to show your work.
- Feel free to write helper functions if you need them.
- **Please write neatly.**

Problem	Points	Out Of
1 Snippets		**6**
2 Storage		**8**
3 Lists		**12**
3 SVM		**4**
Total		**30**

Section 1: Snippets (6 Points Total)

1. (1 Point) In a sentence or two, what is a *value*? Give an example of an OCaml expression that is a value, an example of an expression that is not a value but which has a value and give an example of an expression which has no value.

   Answer: A *value* is an expression that cannot be further simplified. For example, `(2, 3)` is value. The expression `(2, 1  2)`+ is an expression which has a value but is not a value and `(2, 1 / 0)` is an expression that does not have a value.

2. (1 Point) In a sentence or two, what is a *variable*? Are there legal OCaml expressions containing more than one variable with the same name? If so, show one.

   Answer: A variable is a symbol that can be associated with a value.

   ```
   let x = 1 in (let x = 2 in x) + x
   ```

   has two different variables named `x`.

3. (1 Point) Is the following expression well formed? If so, simplify the expression, one step at a time.

   ```
   let x = match (1 + 1) = 3 with | true -> 4 | false -> 5 in x * 2
   ```

   Answer:

   ```
   let x = match (1 + 1) = 3 with | true -> 4 | false -> 5 in x * 2 ->
     let x = match 2 = 3 with | true -> 4 | false -> 5 in x * 2 ->
     let x = match false with | true -> 4 | false -> 5 in x * 2 ->
     let x = 5 in x * 2 ->
     5 * 2 ->
     10
   ```

4. (1 Point) Is the following function well-defined? If so, what is its type?

```
let f (x, y) = (y + 1, x)
```

Answer: Yes it is well typed with type `'a * int -> int * 'a`.

5. (1 Point) $200_4 = X_{16}$. Solve for X.

Answer: X = 20.

6. (1 Point) $123_5 = 212_X$. Solve for X.

Answer: $X = 4$**.**

Section 2: Storage Diagrams (8 Points)

1. (2 Points) Consider the following code:

```
let f x =
  let a = [x; x; x]             (1)
  in
  a                             (2)

f (1, 2, 3)
```

Show the state of the stack and the heap after (1) has executed but before (2) has executed.

Answer:

```
         Stack                                           Heap


+---+
| f |
+---+---+                                    +---+
| x | o-+----------------------------------->| 1 |<---------+
+---+---+                                    +---+          |
| a | o-+-------+                            | 2 |          |
+-------+       |                            +---+          |
                |                            | 3 |          |
                |                            +---+          |
                |                                           |
                |    +---+---+   +---+---+   +---+---+      |
                +--->| o | o-+-->| o | o-+-->| o | o-+-+    |
                     +-+-+-+-+   +-+-+-+-+   +-+-+---+ =    |
                       |           |           |            |
                       +---------->+---------->+------------+
```

2. (2 Points) Consider the following code:

```
let rec f n =
  match n = 0 with
  | true  ->
    let a = (1, 2)                        (1)
    in
    a                                     (2)
  | false -> f (n - 1)

f (1 + 1)
```

Show the state of the stack and the heap after (1) has executed but before (2) has executed.

Answer:

```
         Stack                                       Heap
+---+
| f |
+---+---+
| n | 0 |
+---+---+                             +---+
| a | o-+---------------------------->| 1 |
+---+---+                             +---+
                                      | 2 |
+---+                                 +---+
| f |
+---+---+
| n | 1 |
+---+---+
| a |   |
+---+---+

+---+
| f |
+---+---+
| n | 2 |
+---+---+
| a |   |
+---+---+
```

3. (2 Points) Consider the following definitions.

```
let g (b, c) =
  b.(1) <- (3, 4) ;                (3)
  b                                (4)

let f a =
  let b = ([| a; a |], [a; a])     (1)
  in
  g b                              (2)

f (1, 2)
```

Show the state of the stack and the heap after (1) has executed but before (2) has executed.

Answer:

```
         Stack                                       Heap
                                                                +------+
                               +------------------------------->|  o--+---+
+----+                +------+ |                                +------+   |
| f |          +------>|  o--+--+                               |  o  |    |
+---+---+      |      +------+                                  +--+--+   |
| b | o-+------+      |  o--+--+                                   |      |
+---+---+             +------+ |     +-+-+----+    +-+-+----+      |      |
| a | o-+------+               +---->| o | o-+---->| o | o-+--+    |      |
+---+---+      |                     +-+-+----+    +-+-+----+  =   |      |
               |                       |             |             |      |
               |                       +------------>+------------>+<-----+
               |                                                   |
               |                                                   v
               |                                                +------+
               +----------------------------------------------->|  1  |
                                                                +------+
                                                                |  2  |
                                                                +------+
```

4. (2 Points) Consider the following definition.

```
let g (b, c) =
  b.(1) <- (3, 4) ;                 (3)
  b                                 (4)

let f a =
  let b = ([| a; a |], [a; a])      (1)
  in
  g b                               (2)

f (1, 2)
```

Show the state of the stack and the heap after (3) has executed but before (4) has executed.

Answer:

```
         Stack                                       Heap
+---+
| g |
+---+---+
| b | o-+-----------------------------------------------------+
+---+---+                                                     |
| c | o-+--------------------------+                          |
+---+---+                          |                          v
                                   |                        +-----+
                           +-------+----------------------->|  o--+---+
+---+             +-----+  |       |                        +-----+   |   +-----+
| f |       +---->|  o--+--+       |                        |  o--+---+-->|  3  |
+---+---+   |     +-----+          |                        +-----+   |   +-----+
| b | o-+---+     |  o--+--+       v                                  |   |  4  |
+---+---+         +-----+  |     +-+-+---+     +-+-+---+              |   +-----+
| a | o-+---+              +---->| o | o-+---->| o | o-+--+           |
+---+---+   |                    +-+-+---+     +-+-+---+  =           |
            |                      |             |                    |
            |                      +------------>+------------>+<-----+
            |                                                  |
            |                                                  v
            |                                               +-----+
            +---------------------------------------------->|  1  |
                                                            +-----+
                                                            |  2  |
                                                            +-----+
```

Section 3: Lists, Trees and Arrays

1. (3 Points) Write an OCaml function `oddsToZero : int list -> int list` such that a call of the function `(oddsToZero ns)` returns a list like `ns` but in which all of the odd integers have been replace by zeros. For example, the call `(oddsToZero [1; 2; 3; 4; 5])` would return the list `[0; 2; 0; 4; 0]`.

   Answer:

   ```
   let rec oddsToZero ns = List.map (fun n -> if n mod 2 = 0 then n else 0) ns
   ```

2. (3 Points) Let `a` and `b` be arrays of integers that are in ascending order. Write the function

   ```
   merge : int array -> int array -> int array
   ```

   such that a call `(merge a b)` returns a new array containing the elements of both `a` and `b` in ascending order. For example, the call `(merge [|1; 3|] [|2; 4; 5|])` should evaluate to the array `[|1; 2; 3; 4; 5|]`.

   Answer:

   ```
   let merge a b =
     let m = Array.length a in
     let n = Array.length b in
     let c = Array.make (m + n) 0 in
     let i = ref 0 in
     let j = ref 0
     in
     while (!i < m || !j < n) do
       match !i = m with
       | true  -> c.(!i + !j) <- b.(!j) ; j := !j + 1
       | false ->
         (match !j = n with
          | true  -> c.(!i + !j) <- a.(!i) ; i := !i + 1
          | false ->
            (match a.(!i) < b.(!j) with
             | true  -> c.(!i + !j) <- a.(!i); i := !i + 1
             | false -> c.(!i + !j) <- b.(!j); j := !j + 1))
     done
     ; c
   ```

3. (3 Points) Consider the following representation of binary trees. This definition differs from those discussed in class in that here we use an explicit empty node `Empty` while the in-class version had a leaf `(Leaf t)`. In this representation a leaf would be `Node{info = t; left = Empty; right = Empty}`.

```
type t = A | B | C | D | E | F

type tree = Empty
          | Node of {info : t; left : tree; right : tree}

let t0 = Node {info = F; left = Empty; right = Empty}
let t1 = Node {info = B; left = Empty; right = Empty}
let t2 = Node {info = D; left = Empty; right = t1}
let t3 = Node {info = E; left = Empty; right = Empty}
let t4 = Node {info = C; left = t2; right = t3}
let t5 = Node {info = A; left = t1; right = t4}
```

```
 t5 = A
     / \
    B   C
       / \
      D   E
       \
        F
```

The *root* of tree `t5` is `A`. The length of the path from `A` to `A` is 0. The length of the path from `A` to `C` is 1. The length of the path from `A` to `F` is 3. The *height* of a tree is the length of the longest path from the root. Write the function `height : tree -> int` such that a call `(height tree)` returns the height of the tree.

Answer:

```
let rec height tree =
  match tree with
  | Empty -> 0
  | Node{left = Empty; right = Empty} -> 0
  | Node{left; right} -> 1 + max (height left) (height right)
```

4. (3 Points) Early on we wrote the predicate `isPrime : int -> bool` which tested `n` for primality. Our definition looked something like this:

```
let sqrtInt n = int_of_float (sqrt (float n))
let isFactor m n = n mod m == 0

let isPrime n =
  let top = sqrtInt n in
  let rec repeat candidate =
    match candidate > top with
    | true  -> true
    | false ->
      (match isFactor candidate n with
       | true  -> false
       | false -> repeat (candidate + 1))
  in
  repeat 2
```

Rewrite the `isPrime` function without using recursion.

Answer:

```
let isPrime n =
  let top = sqrtInt n in
  let i = ref 2
  in
  while (!i) <= top && not(isFactor !i n) do i := !i + 1 done;
  !i > top
```

Section 4: SVM (4 Points Total)

Assume that the data segment contains a list of non-zero numbers ending with a sentinal zero, something like `[3; 1; 2; 4; 8; 0]`. Write an SVM program that when called with a number n in `R1`, will `Hlt` with a `1` in register `R0` if n is in the data segment and a `0` in `R0` if it isn't.

Answer:

```
Li  R0, 1
Mov R2, Zero
Lod R3, 0(R2)
Cmp R3, Zero
Beq 4
Cmp R3, R1
Beq 3
Add R2, R2, R0
Jmp -7
Mov R0, Zero
Hlt
```

1 The Simple Virtual Machine

The instruction set of SVM is as follows.

- `Lod Rd, offset(Rs)`: Let `base` be the contents of register `Rs`. Then this instruction loads the contents of data segment location `offset + base` into register `Rd`.
- `Sto Rs, offset(Rd)`: Let `base` be the contents of register `Rd`. Then this instruction stores the contents of register `Rs` into data segment location `offset + base`.
- `Li Rd, number`: loads `number` into register Rd.
- `Mov Rd, Rs`: copies the contents of register Rs into register Rd.
- `Add Rd, Rs, Rt`: adds the contents of registers Rs and Rt and stores the sum in register Rd.
- `Sub Rd, Rs, Rt`: subtracts the contents of register Rt from Rs and stores the difference in register Rd.
- `Mul Rd, Rs, Rt`: multiplies the contents of register Rt by Rs and stores the product in register Rd.
- `Div Rd, Rs, Rt`: divides the contents of register Rs by Rt and stores the integer quotient in register Rd.
- `Cmp Rs, Rt`: sets PSW = Rs - Rt. Note that if Rs > Rt, then PSW will be positive, if Rs == Rt, then PSW will be 0 and if Rs < Rt, then PSW will be negative.
- `Blt disp`: if PSW is negative, causes the new value of PC to be the sum PC + disp. Note that if disp is negative, this will cause the program to jump backward in the sequence of instructions. If PSW $\geq$ 0, this instruction does nothing.
- `Beq disp`: if PSW == 0, causes the new value of PC to be the sum PC + disp. Note that if disp is negative, this will cause the program to jump backward in the sequence of instructions. If PSW $\neq$ 0, this instruction does nothing.
- `Bgt disp`: if PSW, is positive, causes the new value of PC to be the sum PC + disp. Note that if disp is negative, this will cause the program to jump backward in the sequence of instructions. If PSW $\leq$ 0, this instruction does nothing.
- `Jmp disp`: causes the new value of PC to be the sum PC + disp.
- `Jsr disp`: Jump subroutine: `RA := PC` then `PC := PC + disp`.
- `R`: Return from subroutine: `PC := RA`.
- `Hlt`: causes the svm machine to print the contents of registers PC, PSW, R0, R1, R2 and R3. It then halts.