Exam 2
CSCI 1103 Computer Science I Honors

KEY

Thursday November 9, 2017
Instructor Muller
Boston College

Fall 2017

Please do not write your name on the top of this quiz. Before reading further, please arrange to have an empty seat on either side of you. Now that you are seated, please note the number on top of your test and write it together with your name on the sheet that is circulating.

This is a closed-book and closed-notes quiz. Computers, calculators and books are prohibited. Feel free to use a solution to one problem in solving subsequent problems. And unless otherwise specified, feel free to use any repetition idiom that you would like.

Partial credit will be given so be sure to show your work. **Please try to write neatly.**

Problem	Points	Out Of
1 Snippets		4
2 Storage Diagrams		4
3 Repetition		4
4 SVM		5
Total		17

1 Snippets (4 Points Total)

1. (1 Point) Solve for X, $X_{10} = 312_4$.

   Answer:

   X = 54

2. (1 Point) Solve for X, $X_8 = AA_{16}$.

   Answer:

   X = 252

3. (1 Point) Is the following well-formed? If so, what is its value? If it's not well-formed, what's wrong with it?

```
# let x = 2.9 in (x, int_of_float x);;
```

   Answer:

   (2.9, 2)

4. (1 Point) Is the following well formed? If so, what is its value? If it's not well-formed, what's wrong with it?

```
# let a = (let a = 1 in a + a) + a in a + a;;
```

   Answer:

   Unbound variable a

2 Storage Diagrams (4 Points)

Show the state of the Stack and the Heap after (1) has executed but before (2) has executed.

```
let what ns =
  let treble n = n * 3 in
  let ms = List.map treble ns                   (1)
  in
  ms                                            (2)

let go () =
  let ns = [1; 2; 3]
  in
  what ns

go ()
```

```
        Stack                                                    Heap
```

Answer:

```
+-------+
| what |
+------+-----+             +---+---+      +---+---+      +---+---+
|   ms |  o--+------------>| 3 | o-+----->| 6 | o-+----->| 9 | o-+-+
|      +-----+             +---+---+      +---+---+      +---+---+ =
|   ns |  o--+-----------+
+------------+            \
                           \
+----+                      \
| go |                       v
+----+-+-----+             +---+---+      +---+---+      +---+---+
|   ns |  o--+------------>| 1 | o-+----->| 2 | o-+----->| 3 | o-+-+
+------+-----+             +---+---+      +---+---+      +---+---+ =
```

3 Repetition (4 Points)

Do either problem 1 or problem 2 but not both.

1. (4 Points Total) Consider a simple binary tree of integers:

```
type tree = Empty
          | Node { left  : tree; data  : int; right : tree }

let s = Node { left  = Node { left = Empty; data = 5; right = Empty}    s is  4
             ; data  = 4                                                     / \
             ; right = Node { left = Empty; data = 6; right = Empty}        5   6
             }

let t = Node { left  = s                                                t is  8
             ; data  = 8                                                     / \
             ; right = Node { left = Empty; data = 2; right = Empty}        4   2
             }                                                             / \
                                                                          5   6
```

(a) (3 Points) Write a function `add : tree -> int` such that a call `(add tree)` returns the sum of the integers in `tree`. For example, the call `(add t)` should return `25`.

Answer:

```
(* file: 3.1.a.ml
   author: Bob Muller

   A solution to problem 3.1.a for the 2nd midterm exam, Fall 2017.
*)
type tree = Empty
          | Node of { left  : tree
                    ; data  : int
                    ; right : tree
                    }

(* add : tree -> int
*)
let rec add tree =
  match tree with
  | Empty -> 0
  | Node {left; data; right} ->
     data + (add left) + (add right)
```

(b) (1 Point) Write a function `bfs : tree -> int list` such that a call `(bfs tree)` returns a list of the integers in `tree` in *breadth first* order. For example, `(bfs Empty)` would evaluate to `[]`, `(bfs s)` would evaluate to `[4; 5; 6]` and `(bfs t)` would evaluate to `[8; 4; 2; 5; 6]`.

Answer:

```
(* file: 3.1.b.ml
   author: Bob Muller

   A solution to problem 3.1.b for the 2nd midterm exam, Fall 2017.
*)
type tree = Empty
          | Node of { left  : tree
                    ; data  : int
                    ; right : tree
                    }

(* bfs : tree -> int list
*)
let bfs tree =
  let rec repeat trees =
    match trees with
    | [] -> []
    | Empty :: trees -> repeat trees
    | Node {left; data; right} :: trees ->
       data :: repeat (trees @ [left; right])
  in
  repeat [tree]
```

2. (4 Points Total)

   (a) (2 Points) Let's say we're given an array `ns` containing a very large number of integers **in ascending order**:

   ```
   let ns = [| 2; 8; 21; ...; 408; 500; ... |]
   ```

   Write a function `find : int -> int array -> bool` such that a call `(find n ns)` returns `true` if `n` is in `ns`. Otherwise `find` should return `false`.

   Answer:

   ```
   (* file: 3.2.a.ml
      author: Bob Muller

      A simple solution to problem 3.2.a for the 2nd midterm exam, Fall 2017.

     find : int -> int array -> bool
   *)
   let find n ns =
     let answer = ref false
     in
     for i = 0 to Array.length ns - 1 do
       match n = ns.(i) with
       | true  -> answer := true
       | false -> ()
     done;
     !answer
   ```

   (b) (1 Point) How much work does your definition of `find` do?

   Answer:
   The above code is linear.

(c) (1 Point) Can you write a fast, i.e., *sublinear*, version of `find`?

Answer:

```
(* file: 3.2.c.ml
   author: Bob Muller

   A solution to problem 3.2.c for the 2nd midterm exam, Fall 2017.
*)

(* find : int -> int array -> bool

   The call (find n ns) implements a binary search, performs
   log_2 N units of work.
*)
let find n ns =
  let length = Array.length ns in
  let rec repeat lo hi =
    match lo = hi with
    | true  -> n = ns.(lo)
    | false ->
       let middle = (lo + hi) / 2
       in
       match n < ns.(middle) with
       | true  -> repeat lo (middle - 1)
       | false -> n = ns.(middle) || (repeat (middle + 1) hi)
  in
  (length > 0) && repeat 0 (length - 1)
```

4 The Simple Virtual Machine (5 Points)

The SVM instruction set is specified on the attached sheet. Register `R3` contains an integer n and the data segment contains non-negative integers trailed by a single -1. E.g., `data = [2, 3, 4, 2, 2, -1]`. Write an SVM program that replaces all occurrences of n in the data segment by 0. For example, let `R3` contain 2. Then running the program with the data segment above, your program should halt with `data = [0, 3, 4, 0, 0, -1]`.

Answer:

```
 0:   Mov   R2, Zero        # points to datum
 1:   Li    R1, 1           # for incrementing
 2:   Lod   R0, 0(R2)       # R0 <- data
 3:   Cmp   R0, Zero        # check for done
 4:   Blt   6
 5:   Cmp   R0, R3          # not done, check for target
 6:   Beq   2
 7:   Add   R2, R2, R1      # not target, increment data index
 8:   Jmp   -7              # and go back
 9:   Sto   Zero, 0(R2)     # found target, replace it with 0
10:   Jmp   -4              # head back, incrementing first
11:   Hlt                   # done
```

5 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.