Quiz 2
CSCI 1103 Computer Science I Honors

KEY

Tuesday November 8, 2016
Instructor Muller
Boston College

Fall 2016

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		6
2 Storage Diagrams		4
3 Repetition		3
4 SVM		5
Total		18

1 Snippets (6 Points Total)

1. (2 Points) OCaml's `Char` module has a function `Char.chr : int -> char` which accepts an integer ASCII code for a character and returns the character. For example, the call `(Char.chr 65)` evaluates to the character `'A'`.

   Recall that the `List.map : ('a -> 'b) -> 'a list -> 'b list` function is defined as in:

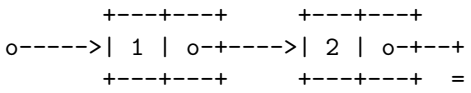
   ```
   let rec map f xs =
     match xs with
     | [] -> []
     | x :: xs -> (f x) :: map f xs
   ```

   Is the expression `(List.map Char.chr)` well-typed? If so, what is its type? And, in a sentence, what does it do?

   Answer:

   Yes, the type is `int list -> char list`. It converts lists of ASCII codes to their characters.

2. (2 Points) True or false? The expressions `1 :: 2 :: []` and `[1; 2]` are represented identically as a chain of "cons" nodes as in the storage diagram:

   Answer:

   True

3. (2 Points) Solve for X. $X_{16} = BC_{16} + 10111100_2$.

   Answer:

   X = 178

2 Storage Diagrams (4 Points)

Show the state of the Stack and the Heap after `(1)` has executed but before `(2)` has executed. Note: between `(1)` and `(2)` the function `append` is called. There is no need to show any of the stack records for the calls of `append` since these records would be popped off the stack at the completion of `(1)`. This question is particularly concerned with the value of `zs`.

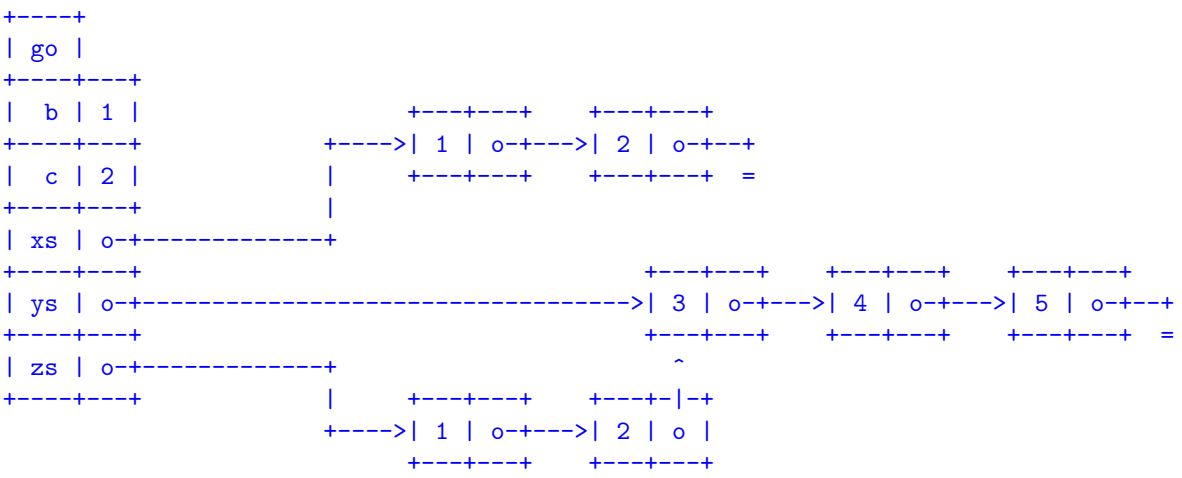
```
let rec append xs ys =
  match xs with
  | [] -> ys
  | x :: xs -> x :: append xs ys

let go b c =
  let xs = [b; c] in
  let ys = [3; 4; 5] in
  let zs = append xs ys            (1)
  in
  zs                               (2)

go 1 2
```

Stack Heap

Answer:



3 Repetition (3 Points)

Write a function `compactList : int list -> int list` such that a call `(compactList ns)` returns a list exactly like `ns` but in which all of the zeros are at the end. For example, the call `(compactList [1; 0; 2; 0; 3])` should evaluate to the list `[1; 2; 3; 0; 0]`. Note that the non-zero values must remain in their original order.

Answer:

```
let compactList ns =
  let xs = List.filter (fun n -> n <> 0) ns in
  let ys = List.filter (fun n -> n =  0) ns
  in
  xs @ ys
```

4 The Simple Virtual Machine (5 Points)

In October we learned about storage and about the basic machine model that underlies all of our computers. We saw the assembly language for a simple virtual computer (SVM) with a RAM and a CPU with an ALU and 8 registers: `pc`, `psw`, `R0`, ..., `R3`, `Zero` and `ra`. The SVM instruction set is specified on the attached sheet.

Consider a data segment with integers trailed by a single 0. E.g., `data = [2, 3, 4, 0]`. Write an SVM program that halts after having squared all of the numbers in the data segment. In the example, the data segment would be mutated so that when the program halted the data segment contained `[4, 9, 16, 0]`.

Answer:

```
0: Mov R1, Zero
1: Li  R2, 1
2: Lod R0, 0(R1)
3: Cmp R0, Zero
4: Beq 4
5: Mul R0, R0, R0
6: Sto R0, 0(R1)
7: Add R1, R1, R2
8: Jmp -7
9: Hlt
```

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.