Second Exam
CS 1101 Computer Science I
Spring 2016

Section 04

KEY

Thursday April 14, 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 26 point exam. Answer all questions in Sections 1 and 2. Answer either 3.1 or 3.2 but not both. Circle the number of the problem that you want graded.

- 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		**6**
2.1		**4**
2.2		**4**
3		**8**
4		**4**
Total		**26**

Section 1 (6 Points Total)

1. (1 Point) Digital computers are "digital" because they use discrete values — digits — to represent things rather than continuous values. Digital computers are actually *binary* digital computers because the discrete values are binary digits (bits) — *everything* in a binary digital computer is represented using the common substrate of bits. For example, the letter **A** is represented as the sequence of 8 bits `0100 0001`. This very same bit sequence is also the representation of the decimal integer `65`, it can also represent 8 booleans, maybe a machine instruction such as `Beq` as well as untold other items.

   In a sentence or two, explain how a digital computer can keep all of this straight. How does it know when to interpret `0100 0001` as a `Beq` rather than `A`?

   Answer: The program counter together with the stored instructions impose an interpretation on the bits.

2. (1 Point) Is the following well-defined? If so, what is its value?

   ```
   [ (a, 8) for a in ["Boston", "College"]]
   ```

   Answer: Yes, answer is [("Boston", 8), ("College", 8)]

3. (1 Point) Solve for X.

   (a) $C5_{16} = X_4$.

   Answer: We could convert $C5_{16}$ to the equivalent decimal numeral, 197, and then use the iterative algorithm to convert 197 to base 4. However, since 4 is a power of 2, there is an easier way. $C5_{16} = 1100\ 0101_2$. Since $4 = 2^2$, we can regroup the bits in groups of 2: $11\ 00\ 01\ 01 = 3011_4$, so $X = 3011$.

   (b) $C5_{16} = X_8$.

   Answer: As above $C5_{16} = 1100\ 0101_2$. Since $8 = 2^3$, we can regroup the bits in groups of 3: $11\ 000\ 101 = 305_8$, so $X = 305$.

4. (3 Points) Show the state of the stack and heap after (1) has executed but before (2) has executed.

```
def f(xs, ys):
  zs = (ys, 8, xs)                (1)
  return zs                       (2)

f([4, 5, 6], (1, (2, (3, None)))

      Stack                                      Heap
```

Answer:

```
+---+
| f |
+--------+                           +---+---+    +---+---+    +---+---+
| ys | o-+-------------------------> | 1 | o-+--->| 2 | o-+--->| 3 | o-+--+
+----+---+                           +---+---+    +---+---+    +---+---+  |
| zs | o-+--------------------+        ^                                  =
+----+---+                    v        |
| xs | o-+---------+        +---+      |
+----+---+         |        | o-+------+
                   |        +---+
                   v        | 8 |
                 +---+      +---+
                 | 4 |<-----+-o |
                 +---+      +---+
                 | 5 |
                 +---+
                 | 6 |
                 +---+
```

Section 2 (8 Points Total)

1. (4 Points) Write a function `masterList : (string * 'a list) list -> 'a list` such that a given call `masterList(pairs)` returns a list combining all of the lists in the pairs. For example, the call

   ```
   masterList([("Haiming", [1.0, 2.0]), ("Alice", [3.0, 4.0])])
   ```

   should evaluate to the list `[1.0, 2.0, 3.0, 4.0]`.

   Answer:

   ```
   def masterList(pairs):
     if pairs == []:
       return []
     else:
       (_, xs) = pairs[0]
       rest = pairs[1:]
       return xs + masterList(rest)

   def masterList(pairs):
     ys = map(lambda (_, zs): zs, pairs)
     return reduce(operator.concat, ys, [])
   ```

2. (4 Points) Write a function `arrayDiff : 'a array * 'a array -> 'a array` such that a given call `arrayDiff(a1, a2)` returns a new array containing all of the elements of `a1` that are not in `a2`. For example, the call `arrayDiff([2, 3, 4], [3, 5])` should evaluate to the array `[2, 4]`.

   Answer:

   ```
   def arrayDiff(a1, a2):
     return [ a for a in a1 if not(a in a2)]
   ```

Section 3 (8 Points Total)

Do either problem 1 or 2. Do not do both.

1. (8 Points) In statistics, the *population standard deviation* of a set of k numbers n_1, n_2, ..., n_k, is a measure of the *range* of values in the set. Computation of the population standard deviation starts with computing the mean

$$a = (n_1 + n_2 + \ldots + n_k)/k.$$

Then, for each member n_i of the population, we compute a measure of n_i's *deviation* from the mean. The deviation for n_i is $d_i = (n_i - a)^2$. Finally, the population standard deviation is the square root of the average of the d_i:

$$\text{psd} = \sqrt{(d_1 + d_2 + \ldots + d_k)/k}$$

For the purposes of the following three problems, we're going to take the population as a list of floating point numbers. Feel free to use the `math.sqrt` function in solving these problems. Also feel free to use the function in (a) in your solution to (b) and/or (c) and feel free to use the function in (b) in your solution to (c), even if you weren't able to complete (a) and (b).

(a) (2 Points) Write a function `deviations : float list -> float list` such that a function call `deviations(population)` returns a list of the deviations. For example, the call

```
deviations([5.0, 3.0, 7.0])
```

should evaluate to the list `[0.0, 4.0, 4.0]` because the average `a = (5.0 + 3.0 + 7.0) / 3` is `5.0`, and

```
(5.0 - 5.0) ** 2.0 = 0.0,
(3.0 - 5.0) ** 2.0 = 4.0 and
(7.0 - 5.0) ** 2.0 = 4.0.
```

Answer:

```
def average(ns): return sum(ns) / len(ns)

def deviations(ns):
  ave = average(ns)
  return map(lambda n : (n - ave) ** 2.0, ns)
```

(b) (2 Points) Write a function `standardDeviation : float list -> float` such that a call

`standardDeviation(population)`

will evaluate to the population standard deviation of `population`. For example, the call

`(standardDeviation [5.0, 3.0, 7.0])`

should evaluate to `1.63` because the deviations of `[5.0, 3.0, 7.0]` are `[0.0, 4.0, 4.0]` and

```
math.sqrt((0.0 + 4.0 + 4.0) / 3.) = math.sqrt(8.0 / 3.)
                                  = math.sqrt(2.66)
                                  = 1.63.
```

Answer:

```
def standardDeviation(ns):
  devs = deviations(ns)
  return sqrt(average(devs))
```

(c) (4 Points) This problem is concerned with computing summary data for student scores. Let's say the input data is a list of pairs:

```
[("Haiming", [80., 75.]), ("Alice", [99., 98.]), ("Mark", [70., 60.])]
```

The desired output is a list of pairs recording how many standard deviations a student's average exam score is from the class average. In this example, the desired result would be

```
[("Haiming", -0.19), ("Alice", 1.27), ("Mark", -1.08)]
```

The overall average of `[80., 75, 99., 98., 70., 60.]` is

$$(80. + 75. + 99. + 98. + 70. + 60.)/6. = 482./6. = 80.3$$

and the standard deviation is `14.19`. The averages for each of the students (resp.) are

```
(80. + 75.) / 2.0 = (155. / 2.0) = 77.5,
(99. + 98.) / 2.0 = (197. / 2.0) = 98.5 and
(70. + 60.) / 2.0 = (130. / 2.0) = 65.0.
```

So the deviations of the average exam score for each of the students from the overall average are:

```
[77.5 - 80.3, 98.5 - 80.3, 65.0 - 80.3] = [-2.79, 18.20, -15.29]
```

And finally, the number of standard deviations for each student are

```
[-2.79 / 14.19, 18.20 / 14.19, -15.29 / 14.19] = [-0.19, 1.27, -1.08]
```

Write the function `summary : (string * float list) list -> (string * float) list`.

Answer:

```
def summary(pairs):
  ns = masterList(pairs)
  overallAverage = average(ns)
  psd = standardDeviation(ns)
  def compute(pair):
    (name, scores) = pair
    dev = average(scores) - overallAverage
    return (name, dev / psd)
  return map(compute, pairs)
```

2. (8 Points) Write a function `permutations : 'a list * int -> ('a list) list` such that a call

```
permutations(symbols, n)
```

returns a list of all n-length permutations of symbols drawn from `symbols`. For example, the function call `permutations([0, 1], 3)` should return the 8-element list of length-3 lists:

```
[[0, 0, 0], [0, 0, 1], [0, 1, 0], [0, 1, 1]
 [1, 0, 0], [1, 0, 1], [1, 1, 0], [1, 1, 1]]
```

The ordering of the list elements is unimportant. This is a challenging problem.

Answer:

```
def permutations(symbols, n):
  if n == 0:
    return [[]]
  else:
    perms = permutations(symbols, n - 1)
    almost = map(lambda symbol : map(lambda perm : [symbol] + perm, perms), symbols)
    return reduce(operator.concat, almost, [])
```

Section 4 (4 Points)

Let M and N be non-negative integers and consider a data segment `Data = [M, N, ...]`. Write an SVM program to compute `(isFactor M N)` halting with a 0 in register R0 if M is not a factor of N and halting with a 1 in R0 if M is a factor of N. Feel free to place any values that you want in the data segment after M and N. (Yes, this is exactly Part A of problem set 7.)

Answer:

```
data = [M, N]

Lod R0, 0(Zero)    # R0 := M
Lod R1, 1(Zero)    # R1 := N
Cmp R1, R0         # R1 < R0 ?
Blt 2
Sub R1, R1, R0     # R1 := R1 - R0  (N := N - M)
Jmp -4
Cmp R1, Zero       # See R0 is a factor
Beq 2
Li  R0, 0
Hlt
Li  R0, 1
Hlt
```

	Instruction	Meaning
1.	Lod Rd, offset(Rs)	Let base be the contents of register Rs and let address = base + offset. Then register Rd gets the contents of location address in RAM.
2.	Li Rd, number	Rd := number.
3.	Sto Rs, offset(Rt)	Let base be the contents of register Rt and let address = base + offset. Then RAM location address gets Rs.
4.	Mov Rd, Rs	Rd := Rs.
5.	Add Rd, Rs, Rt	Rd := Rs + Rt.
6.	Sub Rd, Rs, Rt	Rd := Rs - Rt.
7.	Mul Rd, Rs, Rt	Rd := Rs * Rt.
8.	Div Rd, Rs, Rt	Rd := Rs / Rt.
9.	Cmp Rs, Rt	PSW := Rs - Rt.
10.	Beq disp	PC := PC + disp if PSW = 0.
11.	Blt disp	PC := PC + disp if PSW < 0.
12.	Bgt disp	PC := PC + disp if PSW > 0.
13.	Jmp disp	PC := PC + disp.
14.	Jsr disp	RA := PC, PC := PC + disp.
15.	R	PC := RA.
16.	Hlt	SVM halts.

Table 1: The SVM Instruction Set. Notation: Rd is a destination register, Rs and Rt are source registers. All of offset, number and disp are integers.