Second Exam
CS 101 Computer Science I

KEY
Wednesday March 27, 2013
Instructor Muller
Boston College
Spring 2013

Please write your name at the top. You are allowed to use one 8.5 x 11 sheet of notes. Computers, calculators, books and notes are prohibited. Partial credit will be given so be sure to show your work. **Please try to write neatly.**

| Problem | Points | Out Of |
|---|---|---|
| 1 | | 3 |
| 2 | | 3 |
| 3 | | 6/7 |
| 4 | | 3 |
| 5 | | 3 |
| 6 | | 4 |
| Total | | 22 |

1. (3 Points) The **stdaudio** library represents an audio signal with 2 channels sampled at 44100 kHz with individual amplitude samples ranging between -1 and +1. The stduadio library includes a **makeSound** function that accepts two lists of amplitude samples and returns a *sound*. What would you hear from the following:

```
>>> ch = [ 1 for _ in range(88200) ]
>>> sound = makeSound(ch, ch)
>>> play(sound)
```

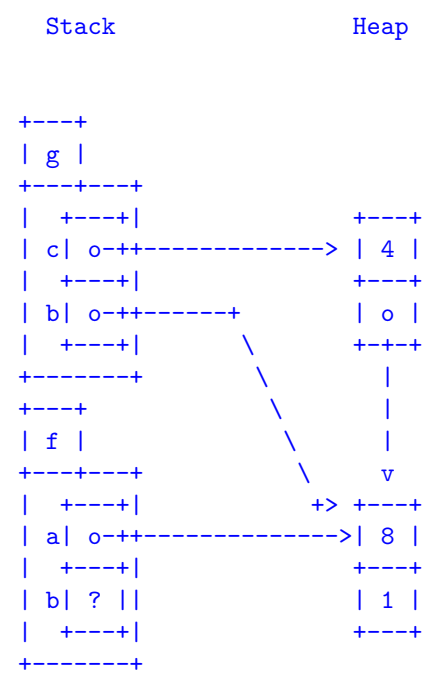
Silence.

2. (3 Points) Show the state of the stack and the heap after line (1) is complete but before line (2) has been executed.

```
def f(a):
  b = g(a)
  return b

def g(b):
  b[0] = 8
  c = (4, b)     (1)
  return c       (2)

f([0, 1])
```



3. (6/7 Points) The function **isAscending : int list → bool** returns **True** if the input list is in strictly ascending order. Otherwise it returns **False**. For example, the call **isAscending(range(4))** should return **True**. In general, **isAscending** should return **True** for any list of length less than 2. The call **isAscending([1, 2, 2, 3])** would return **False**.

   Do problem **(a)** and **either** problem **(b)** or **(c)** but not both. Problem **(c)** is worth one extra point.

   (a) **(3 Points)** Write the function **isAscending** using either a **while**-loop or a **for**-loop.

```
def isAscending(list):
  N = len(list)
  if N < 2: return True
  for i in range(N - 1):
    if list[i] >= list[i + 1]:
      return False
  return True
```

```
def isAscending(list):
  N = len(list)
  if N < 2: return True
  i = 0
  while i < N - 1:
    if list[i] >= list[i + 1]:
      return False
    i = i + 1
  return True
```

   (b) **(One-of: 3 Points)** Write the function **isAscending** using recursion.

```
def isAscending(list):
  if len(list) < 2:
    return True
  elif list[0] >= list[1]:
    return False
  else:
    return isAscending(list[1:])
```

   (c) **(One-of: 4 Points)** Write the function **isAscending** without using a **while**-loop, a **for**-loop or recursion.

```
def isAscending(list):
  N = len(list) - 1
  a = [ 1 for i in range(N) if list[i] < list[i + 1]]
  return sum(a) == N
```

4. **(3 Points)** A point in the Cartesian plane is represented as a pair $(x, y)$. The *distance* between two points $(x_0, y_0)$ and $(x_1, y_1)$ is given by the Pythagorean theorem:

$$\sqrt{(x_0 - x_1)^2 + (y_0 - y_1)^2}$$

In Python, a point in the Cartesian plane can be represented by a value of type **point = float * float**.

Write a function **distance : point → float** such that if **p** is the point **(x, y)**, the call **distance(p)** returns the distance from **p** to the origin $(0, 0)$. Note that the **distance** function accepts one argument that is a pair. It does not accept two arguments. Feel free to use the **math.sqrt** function.

5. **(3 Points)** Write a function **distances : (float * float) list → float list** such that the call **distances([(x1, y1), ..., (xn, yn)])** will return a list of the distances of the points to the origin. Feel free to use the **distance** function as a helper even if you weren't able to write it.

6. **(4 Points)** Write a function **within : float * (float * float) list → int** such that the call **within(d, [(x1, y1), ..., (xn, yn)])** will return the number of points in **[(x1, y1), ..., (xn, yn)]** that are strictly within distance **d** of the origin.