First Exam
CS 101 Computer Science I

KEY
Friday February 8, 2013
Instructor Muller
Boston College
Fall 2012

Please write your name at the top. Do problem 1 and **either** of problems 2 or 3 **but not both** and do problems 4, 5 and 6. You are allowed to use one 8.5 x 11 sheet of notes. Computers, calculators, books and notes are prohibited. In solving problems involving repetition, you are free to use any form 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		5
2		5 or
3		5
4		7
5		7
6		8
Total		32

1. (5 Points) Consider the following python functions for computing the volume of a cylinder:

```
def area(radius):
  return 3.14 * radius ** 2

def volume(radius, height):
  return area(radius) * height
```

Use the plug-it-in method to show the evaluation of `volume(2, 4 - 1)`. Your answer should include one line for each individual step in the computation.

```
volume(1 + 1, 3)
 = volume(2, 3)
 = return area(2) * 3
 = return (return 3.14 * 2 ** 2) * 3
 = return (return 3.14 * 4) * 3
 = return (return 12.56) * 3
 = return 12.56 * 3
 = return 37.68
 = 37.68
```

2. (5 Points) Temperatures are usually expressed using either *Fahrenheit* or *Celcius* scales. To convert Fahrenheit temperature T to Celcius, subtract 32 from T and multiply the result by 5.0 / 9.0. Write a function `fToC : float -> float` that accepts a temperature in Fahrenheit and returns the temperature in Celcius.

```
def fToC(farenheit): return (farenheit - 32.0) * (5.0 / 9.0)
```

3. (5 Points) Write a python function `member : a * a list -> bool` such that a call `member(item, list)` will return `True` if `item` occurs in `list`. Otherwise, `member` should return `False`.

```
def member(item, list):
  if list == []:
    return False
  else:
    if item == list[0]:
      return True
    else:
      return member(item, list[1:])
```

4. (7 Points) An *association list* is a list of pairs `[(key`$_1$`, value`$_1$`), ..., (key`$_n$`, value`$_n$`)]` where each key is associated with a value. For example, the association list

```
a = [(4, [1, 2, 4]), (5, [1, 5]), (6,[1, 2, 3, 6])]
```

associates the integer keys `4`, `5` and `6` with their integer factors. Write a python function

`assoc : a * (a * b) list -> b` such that given a call `assoc(key, alist)`, the function `assoc` returns the value in the pair with the matching key. For example, `assoc(5, a)` should evaluate to `[1, 5]`. You may assume that the keys in the association list are unique but you may not assume that `key` is actually associated with anything in the `alist`. If `key` isn't associated with any value your function should return the special built-in value `None`.

```
def assoc(key, alist):
  for pair in alist:
    if key == pair[0]:
      return pair[1]
  return None
```

or

```
def assoc(key, alist):
  if alist == []:
    return None
  else:
    pair = alist[0]
    if key == pair[0]:
      return pair[1]
    else:
      return assoc(key, alist[1:])
```

5. (7 Points) Association lists are an especially simple implementation of a *dictionary*. They work well enough for small applications but they aren't particularly efficient when it comes to finding keys. On average, the `assoc` function above has to look halfway through the list to find a given `key`. A more efficient way to represent associations between keys and values is to use a *binary search tree*.

   A simple scheme for implementing a dictionary using a binary search tree in Python could work as follows:

   - An empty dictionary is the built-in python constant `None`,
   - A non-empty dictionary is a 4-tuple of the form `(dictionary, key, value, dictionary)`.

   So if `d` is a non-empty dictionary, `d[0]` is another dictionary, `d[1]` is a key, `d[2]` is a value, and `d[3]` is another dictionary. For example, using integer keys and string values, dictionary `d` below is a dictionary recording 3 associations:

```
leftD = (None, 100, 'Alice', None)
rightD = (None, 300, 'Mary', None)
d = (leftD, 200, 'Bob', rightD)
```

   The essential idea of the scheme is that the keys in the dictionary are arranged so the smaller keys are always on the left while the larger keys are on the right. For example, in dictionary `d` above, the single key in dictionary `d[0]` is 100 which is less than 200 while the single key in dictionary `d[3]` is 300 which is greater than 200.

   (a) Write a function `find(key, dictionary)` which finds the value associated with the key in the dictionary. Your function should return `None` if the key isn't associated with a value in the dictionary.

```
def find(key, dictionary):
  if dictionary == None:
    return None
  else:
    k = dictionary[1]
    if key == k:
      return dictionary[2]
    else:
      if key < k:
        return find(key, dictionary[0])
      else:
        return find(key, dictionary[3])
```