CS 61A Spring 2014

Structure and Interpretation of Computer Programs

TEST 2 SOLUTIONS

INSTRUCTIONS

- You have 2 hours to complete the exam.
- The exam is open book and open notes. You may not use a computer, calculator, or anything responsive.
- Mark your answers ON THE EXAM ITSELF. If you are not sure of your answer you may wish to provide a *brief* explanation.
- Since some students will be taking the exam later, PLEASE DO NOT DISCUSS IT before discussion sections.

Last name	
First name	
SID	
Login	
TA & section time	
Name of the person to your left	
Name of the person to your right	
All the work on this exam is my own. **(please sign)**	

For staff use only

Q. 1	Q. 2	Q. 3	Q. 4	Q. 5	Q. 6	Total
/12	/15	/8	/11	/	/4	/50

1. (12 points) What would Python Do?

(a) (9 points)

Assume that you have started Python 3 and executed the following statements:

```
class Doge:
    much = "wow"
    def bark(self):
        return self.much

class Puppye(Doge):
    very = "meow"
    def __init__(self, friend):
        self.friend = friend
        self.much = [Doge.much, friend.much]

    def bark(self):
        return "puppye" + Doge.bark(self)[1]


such = Doge()
such.much = "interesting"
fido = Puppye(such)
flora = Puppye(fido)
```

Expression	Evaluates to
`abs(-3)`	3
`1/0`	ERROR
`Doge.much`	'wow'
`Puppye.much`	'wow'
`such.very`	Error ('Doge' has no 'very')
`such.bark()[0:2]`	'in'
`fido.bark()`	puppyeinteresting
`fido.friend.much`	'interesting'
`fido.friend.bark()`	'interesting'
`flora.friend.bark()`	puppyeinteresting
`flora.friend.friend.much`	'interesting'

(b) (3 points) Consider the following Python function:

```
def foo(a, b):
    return map(lambda x: a[x//2] if x % 2 == 0 else b[x//2], range(len(a+b)))
```

What is the value of `tuple(foo( (10, 5, 25), (8, 7, 11) ))`?

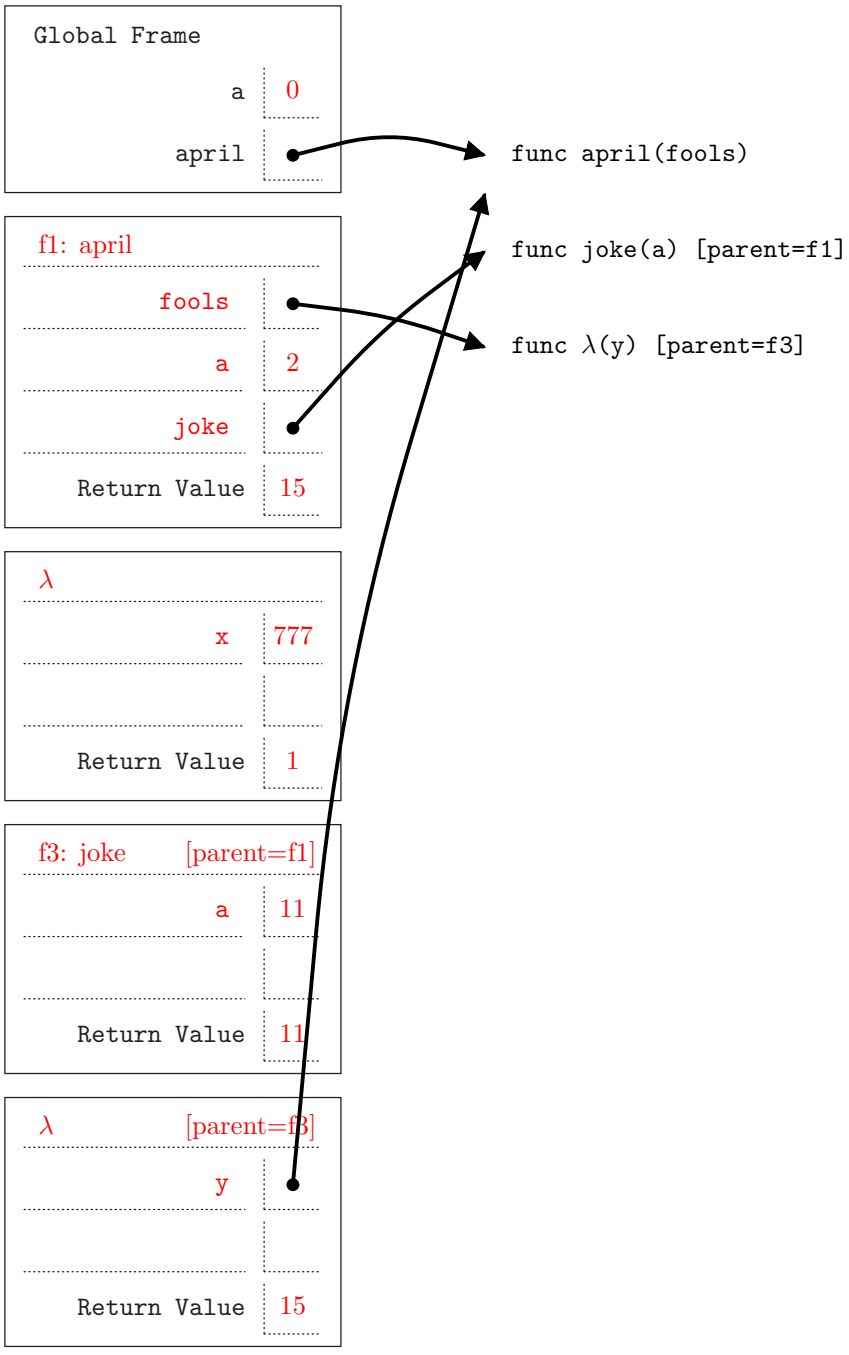
Ans: (10, 8, 5, 7, 25, 11)

2. (15 points) Assorted Diagrams

(a) (8 points) Fill in the environment diagram that results from executing the following segment until the entire program is finished, an error occurs, or all frames are filled. *You may not need to use all of the spaces or frames.* A complete answer will:

- Add all missing names, labels, and parent annotations to all local frames.
- Add all missing values created during execution.
- Show the return value for each local frame resulting from a call that has completed when execution stops (leave other return values blank).

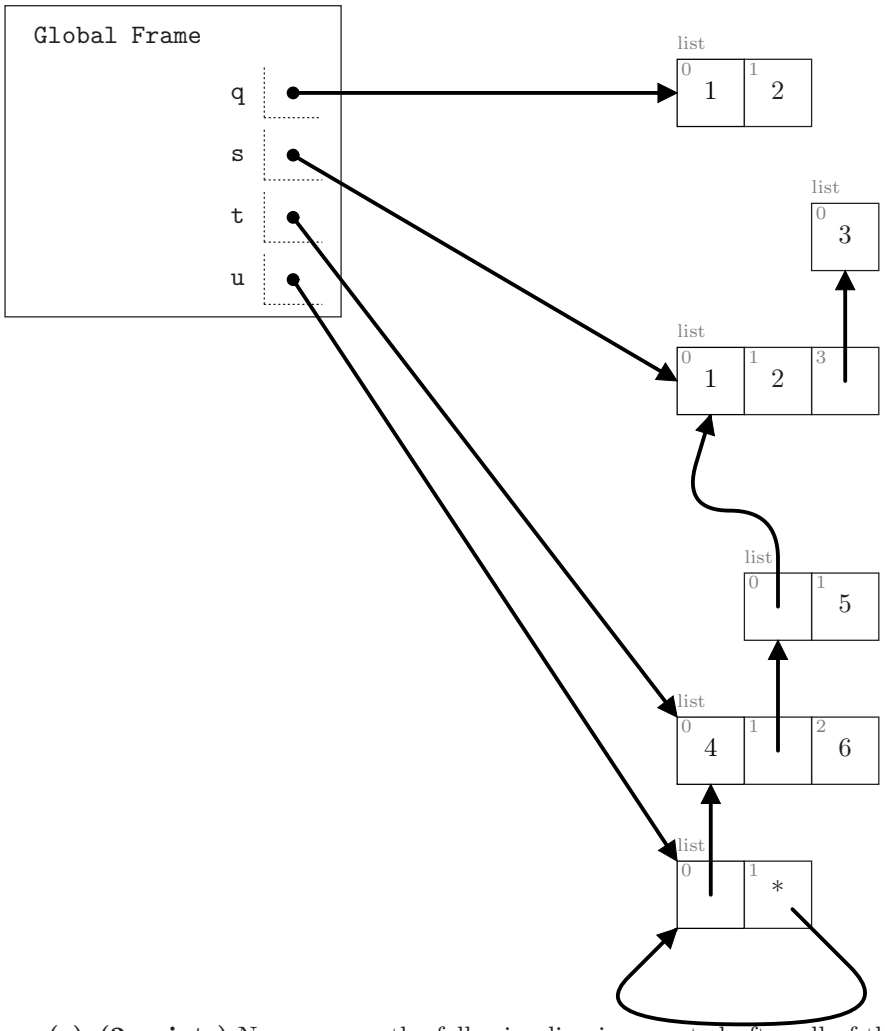
```
a = 0
def april(fools):
    a = 2
    def joke(a):
        nonlocal fools
        fools = lambda y: a + 4
        return a
    if fools(777) == 1:
        joke(a + 9)
    return fools(april)
april(lambda x: a + 1)
```



(b) (5 points) Draw the box-and-pointer diagram that results from executing the following code:

```
q = [1, 2]
s = [1, 2, [3]]
t = [4, [s, 5], 6]
u = [t]
```

We've already drawn the diagram for `q` as an example.



(c) (2 points) Now assume the following line is executed after all of the statements above are executed.

```
u.append(u)
```

Show the result on your diagram. Mark any added list cells with an asterisk (*) to show what you change.

3. (8 points) Data Abstraction

Albert loves to create salads, and decides to create functional abstractions representing vegetables and salads.

(a) (2 points) Unfortunately, Albert only had time to finish writing the constructors for his abstract data types. Help him out by writing in the corresponding selector functions.

```
def make_vegetable(name, color):
    """Returns a representation of a vegetable with given NAME and COLOR,
    which must both be strings."""
    return [[name], [color]]

def get_name(vegetable):

    return ____vegetable[0][0]____

def get_color(vegetable):

    return ____vegetable[1][0]____


def make_salad(name, vegetables):
    """Return a salad having the given NAME, and constituent VEGETABLES (a list of
    vegetables."""
    return [[name], vegetables]

def get_salad_name(salad):

    return ____salad[0][0]____

def get_vegetables(salad):

    return ____salad[1]____
```

Condensation of the methods from part (a) for reference:

```
def make_vegetable(name, color): return [[name], [color]]
def get_name(vegetable): ...
def get_color(vegetable): ...
def make_salad(name, vegetables): return [[name], vegetables]
def get_salad_name(salad): ...
def get_vegetables(salad): ...
```

(b) (5 points) Steven decides to use Albert's data abstractions to create a delicious lunch. However, he doesn't feel confident that he understands how to use Albert's abstract data types. Help him out by both:

- Crossing out any mistakes or data abstraction violations he has made.
- Writing in code that correctly conveys what he was trying to do, either on the line of code itself or below the line of code where the issue occurs.

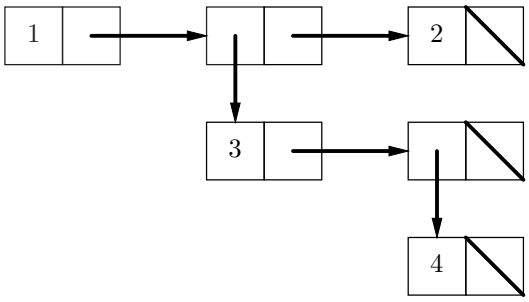
```
 ingredients = [ make_vegetable("lettuce", "green"), make_vegetable("carrot" , "orange")]
ingredients = [ make_vegetable(["lettuce"], ["green"]), [["carrot"] , ["orange"]] ]

 first_vegetable_color = get_color(ingredients[0])
first_vegetable_color = ingredients[0][1][0]


lunch = make_salad( "simple_salad" , ingredients + [make_vegetable("cucumber", "green")] )

 salad_name = get_salad_name(lunch)
salad_name = lunch[0][0]

 for veg in get_vegetables(lunch):
for xxxxx in lunch[1]: #Line 5. See part (c)

     vegetable_name = get_name(veg)
    vegetable_name = get_salad_name(xxxxx)  # Line 6

     vegetable_color = get_color(veg)
    vegetable_color = xxxxx[1][0]


    print("From " + salad_name + ", I have a " + vegetable_color + " " + vegetable_name)


last_ingredient = get_vegetables(lunch)[len(get_vegetables(lunch)) - 1]

 print(get_name(last_ingredient) " was a great finishing touch!")
print( last_ingredient.name + " was a great finishing touch!" )
```

(c) (1 point) What meaningful variable name could be used in place of `xxxxx` on lines 5 and 6?

See above: "vegetable", "veg", or "ingredient", etc.

4. (11 points) Programs

(a) (5 points) Given a nested list of rlists, return the sum of all integers weighted by their depth. In the `rlist` below, 1 and 2 are at depth 1, 3 is at depth 2, and 4 is at depth 3:



Here is the relevant part of the `rlist` implementation from lecture:

```
empty_rlist = ...
def rlist(first, rest = empty_rlist): ...
def first(r): ...
def rest(r): ...
```

Hint: Python's built-in method `isinstance(avalue, aclass)` will tell you if `avalue` belongs to `aclass` (or a subtype of `aclass`).

```
def nested_sum(rlst):
    """Assuming RLST to be an rlist containing only rlist or integer
    values, the sum of all integers in RLST, weighted by their depth.
    >>> simple = rlist(1, rlist(4, rlist(6)))
    >>> nested_sum(simple) # 1*1 + 1*4 + 1*6
    11
    >>> harder = rlist(1, rlist(rlist(3, rlist(rlist(4))), rlist(2)))
    >>> nested_sum(harder) # 1*1 + 2*3 + 3*4 + 1*2
    21"""
    return sum_at_depth(rlst, 1)

def sum_at_depth(rlst, d):
    """The sum of all integers in RLST, weighted by their depths, assuming
    that RLST itself is at depth D."""

    if rlst is empty_rlist :

        return 0

    elif isinstance(first(rlst), int) :

        return d * first(rlst) + sum_at_depth(rest(rlst), d)
    else:
        # (Your answer need not use all lines.)

        return sum_at_depth(first(rlst), d+1) + sum_at_depth(rest(rlst), d)
```

(b) (6 points) In lecture, we defined a class `BinTree` for describing binary trees, supporting the following operations (T denotes a `BinTree` value):

BinTree(*LABEL, LEFT, RIGHT***)** A binary tree whose label is *LABEL*, whose left child is *LEFT*, and whose right child is *RIGHT*.

BinTree.empty_tree The empty tree.

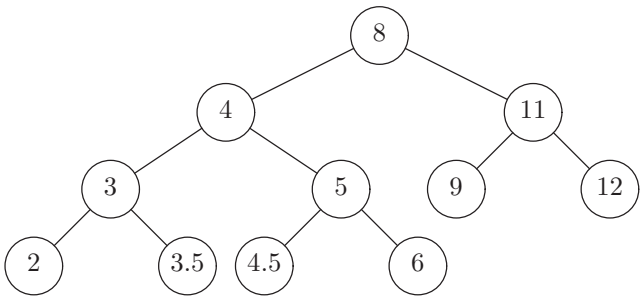
T**.is_empty** True iff T is the empty tree.

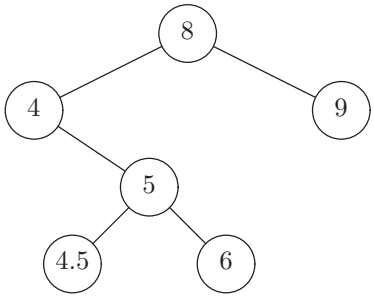
T**.label** The label of T.

T**.left** The left child of T.

T**.right** The right child of T.

Write a function `bound` that takes in a binary search tree containing real numbers, and two numbers `low` and `high`. It returns a new binary search tree that contains all and only the elements in the input tree that are between `low` and `high`, inclusive. So if `tree` is the tree on the left, `bound(tree, 4, 10)` will produce the tree on the right.






```
def bound(tree, low, high):
    """A binary search tree containing exactly the elements in binary
    search tree TREE that are between LOW and HIGH, inclusive.  Assume LOW<=HIGH.

    if ____tree.is_empty____:
        return ____BinTree.empty_tree____
    elif ____low > tree.label____:
        return ____bound(tree.right, low, high)____
    elif ____high < tree.label____:
        return ____bound(tree.left, low, high)____
    else:
        return BinTree(tree.label,
                       ____bound(tree.left, low, high), bound(tree.right, low, high)____)
```

5. (1 point) Sum of Human Knowledge

It is a little-known fact that Santa Claus actually manages to avoid the bad arctic winter flying conditions by instead using a series of perfectly straight tunnels dug (presumably by magical means) directly through the earth from the north pole. One of them, in fact, emerges near Coffs Harbour in New South Wales at about latitude 30°S. On his bathroom scale at the north pole, Santa weighs in at 100kg (or, for those who have recently taken physics, 100kg-wt). What would the scale read if he used it while exactly halfway along this particular tunnel (±2%)?

50kg-wt

6. (4 points) The Big Theta

Consider the following function (assume that parameter `S` is a list):

```
def umatches(S):
    result = set()
    for item in S:
        if item in result:
            result.remove(item)
        else:
            result.add(item)
    return result
```

For (b)–(d) below, give the tightest bounds you can (i.e., preferably (Θ) for each):

(a) (1 point) Fill in the blank: The function `umatches` returns the set of all

values in `S` that occur an odd number of times. .

(b) (1 point) Let's assume that the operations of adding to, removing from, or checking containment in a set each take roughly constant time. Give an asymptotic bound on the worst-case time for `umatches` as a function of N = `len(S)`.

$\Theta(N)$

(c) (1 point) Suppose that instead of having `result` be a set, we make it a list, (so that it is initialized to `[]` and we use `.append` to add an item). What now is the worst-case time bound? You can assume that `.append` is a constant-time operation, and `.remove` and the **in** operator require time that is $\Theta(L)$ in the worst case, where L is the length of the list operated on. Since we never add an item that is already in the list, each value appears at most once, just as for a Python set.

$\Theta(N^2)$

(d) (1 point) Now suppose that we consider only cases where the number of different values in list `S` is at most 100, and we again use a list for `result`. What is the worst-case time now?

$\Theta(N)$