# CS 61A Structure and Interpretation of Computer Programs Summer 2016 MIDTERM SOLUTIONS

# INSTRUCTIONS

- You have 2 hours and 50 minutes to complete the exam.
- The exam is closed book, closed notes, closed computer, closed calculator, except one 8.5"  $\times$  11" cheat sheet of your own creation.
- Mark your answers on the exam itself. We will not grade answers written on scratch paper.

Last name	
First name	
Student ID number	
Instructional account (cs61a)	
BearFacts email (_@berkeley.edu)	
ТА	
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)	

#### 1. (6 points) Olympic Games

(a) (4 pt) For each of the statements below, write the output displayed by the interactive Python interpreter when the statement is executed. The output may have multiple lines. No answer requires more than four lines. The first two have been provided as examples.

Assume that you have started python3 and executed the following statements:

```
gold = [0, 1]
def go(dream, team):
    print('Steps to success:')
    def medal(lion):
        return dream(team, lion)
    return medal
def win(big):
    print('Eat vegetables.')
    return big[-1] + big[-2]
def get(it, done):
    do = it
    while done < 2:
        print("Don't cheat!")
        do, done = do + [win(do)], done + 1
    return do
```

>>> print(4, 5) + 1 4 5 Error	>>> riooo = go(get, gold) Steps to success:
>>> gold[-2] 0	<pre>&gt;&gt;&gt; usa = riooo(0) Don't cheat! Eat vegetables. Don't cheat! Eat vegetables.</pre>
>>> win(gold) Eat vegetables. 1	>>> usa [0, 1, 1, 2]

(b) (2 pt) What would be the return value of riooo(-2), after evaluating the expressions above?
 [0, 1, 1, 2, 3, 5]

## 2. (6 points) The Name Game

Fill in the environment diagram that results from executing the code below 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 and parent annotations to all frames.
- Add all missing values created or referenced during execution.
- Show the return value for each local frame.



#### 3. (3 points) High Scores

Write expressions involving the list lst that evaluate to the following values. Your expressions must use lst. For the second and third lines, you must use a list comprehension.

```
>>> lst = [[1, 2], [3, 4], [5, 6], [7, 8], [9, 10]]
>>> len(lst)
5
>>> [[11 - x, 11 - y] for x, y in lst]
[[10, 9], [8, 7], [6, 5], [4, 3], [2, 1]]
>>> [2 * x for x in lst[0]]
[2, 4]
```

4. (4 points) Kerbal Space Program

Define the function  $is\_sorted$  that takes in a non-negative integer n and returns True if the digits of n are in non-increasing order from left to right, and False otherwise. See the doctests for details.

You may only use the lines provided. You may not use any Python built-in sorting functions.

```
def is_sorted(n):
    """Returns whether the digits in n are in non-increasing order
    from left to right.
    >>> is_sorted(4)
    True
    >>> is_sorted(55555)
    True
    >>> is_sorted(9876543210)
    True
    >>> is_sorted(9087654321)
    False
    .....
    if n < 10:
        return True
    elif n % 10 > (n // 10) % 10:
        return False
    else:
        return is_sorted(n // 10)
```

4

#### 5. (6 points) Katamari

(a) (4 pt) Define the function aggregate that takes in a two-argument function fn, a sequence seq, and a predicate function pred and returns the result of using fn to combine the elements in seq for which pred returns True. See the doctests for details. You may assume fn is commutative and never returns None.

You may only use the lines provided. You may not need to fill all the lines. You may not use an import statement.

```
def aggregate(fn, seq, pred):
    """Aggregates using fn the elements in seq that satisfy pred.
    >>> def is_even(x):
            return x % 2 == 0
    . . .
    >>> def sum_plus_one(x, y):
            return x + y + 1
    . . .
    >>> aggregate(sum_plus_one, [2, 4, 6], is_even) # (2 + 4 + 1) + 6 + 1
    14
    >>> # If no elements satisfy pred, return None
    >>> aggregate(sum_plus_one, [1, 3, 5, 7, 9], is_even)
    >>> # If only one element satisfies pred, return that element
    >>> aggregate(sum_plus_one, [1, 2, 3], is_even)
    2
    .....
    result = None
    for elem in seq:
        if pred(elem):
            if result == None:
                result = elem
            else:
                result = fn(result, elem)
    return result
```

(b) (2 pt) Use the aggregate function above, which you can assume works correctly, to define the fact function that returns the factorial of non-negative integers. Use only one line of code. If you need more space, you can continue your line of code on the second blank. The add and mul functions, which may be useful, have been imported for you.

*Hint:* You may need the ternary operator <expr1> if <cond> else <expr2>.

```
>>> from operator import add, mul
>>> fact = lambda n: aggregate(mul, range(1, n+1), lambda x: True) if n > 0 else 1
>>> fact(0)
1
>>> fact(5)
120
```

#### 6. (6 points) Legend of Zelda

This question uses the following linked list data abstraction. We have provided an example linked list.

```
>>> empty = 'empty'
                                             >>> link(1, link(2, link(3)))
>>> def link(first, rest=empty):
         return [first, rest]
. . .
. . .
                                           1
                                                        2
                                                                    3
>>> def first(lnk):
         return lnk[0]
. . .
. . .
>>> def rest(lnk):
         return lnk[1]
. . .
```

Define the function linked\_sum that takes in a linked list of positive integers lnk and a non-negative integer total and returns the number of combinations of elements in lnk that sum up to total. You may use each element in lnk zero or more times. See the doctests for details. Do not violate abstraction barriers.

You may only use the lines provided. You may not need to fill all the lines.

```
def linked_sum(lnk, total):
    """Return the number of combinations of elements in lnk that
    sum up to total.
    >>> # Four combinations: 1 1 1 1 1 1 1 2 , 1 3 , 2 2
    >>> linked_sum(link(1, link(2, link(3, link(5)))), 4)
    4
    >>> linked_sum(link(2, link(3, link(5))), 1)
    0
    >>> # One combination: 2 3
    >>> linked_sum(link(2, link(4, link(3))), 5)
    1
    .....
    if total == 0:
        return 1
    elif lnk == empty or total < 0:
        return 0
    else:
        with_first = linked_sum(lnk, total - first(lnk))
        without_first = linked_sum(rest(lnk), total)
        return with_first + without_first
```

#### Name:

## 7. (9 points) Game of Thrones

This question uses the following tree data abstraction. We have provided an example tree.



(a) (6 pt) Define the function track\_lineage that takes in a tree of strings family\_tree and a string name. Assume that there is a unique node with entry name. track\_lineage returns a list with the entries of the parent and grandparent of that node.<sup>1</sup> If the node with entry name does not have a parent or grandparent, return None for that element in the list. See the doctests for details. Do not violate abstraction barriers. You may only use the lines provided. You may not need to fill all the lines.

```
def track_lineage(family_tree, name):
    """Return the entries of the parent and grandparent of
    the node with entry name in family_tree.
    >>> t = tree('Tytos', [
                 tree('Tywin', [
    . . .
                     tree('Cersei'), tree('Jaime'), tree('Tyrion')
    . . .
                 ]),
    . . .
                 tree('Kevan', [
    . . .
                     tree('Lancel'), tree('Martyn'), tree('Willem')
    . . .
                 ])])
    . . .
    >>> track_lineage(t, 'Cersei')
    ['Tywin', 'Tytos']
    >>> track_lineage(t, 'Tywin')
    ['Tytos', None]
    >>> track_lineage(t, 'Tytos')
    [None, None]
    .....
    def tracker(t, p, gp):
        if name == entry(t):
            return [p, gp]
        for c in children(t):
             res = tracker(c, entry(t), p)
             if res:
                 return res
    return tracker(family_tree, None, None)
```

<sup>&</sup>lt;sup>1</sup>The grandparent of a node is the parent of the node's parent.

(b) (3 pt) Assuming that track\_lineage works correctly, define the function are\_cousins that takes in a tree of strings family\_tree and two strings name1 and name2 and returns True if the node with entry name1 and the node with entry name2 are cousins in family\_tree. Assume that there are unique nodes with entries name1 and name2 in family\_tree. See the doctests for details.

Two nodes are cousins if they have the same grandparent but different parents.

You may only use the lines provided. You may not need to fill all the lines.

```
def are_cousins(family_tree, name1, name2):
    """Return True if a node with entry name1 is a cousin of a node with
    entry name2 in family_tree.
    >> are_cousins(t, 'Kevan', 'Tytos') # same tree as before
    False
    >> are_cousins(t, 'Cersei', 'Lancel')
    True
    >> are_cousins(t, 'Jaime', 'Lancel')
    True
    >> are_cousins(t, 'Jaime', 'Tyrion')
    False
    """
    p1, gp1 = track_lineage(family_tree, name1)
    p2, gp2 = track_lineage(family_tree, name2)
    return p1 != p2 and gp1 is not None and gp1 == gp2
```

## 8. (0 points) Games of Berkeley

In the box below, write a positive integer. The student who writes the lowest unique integer will receive one extra credit point. In other words, write the smallest positive integer that you think no one else will write.

