CS 61A Spring 2017

Structure and Interpretation of Computer Programs

Test 1 Solutions

INSTRUCTIONS

- You have 2 hours to complete the exam.
- The exam is open book, open notes, closed computer, closed calculator.
- Mark your answers **on the exam itself**. We will *not* grade answers written on scratch paper.

Last name	
First name	
Student ID number	
CalCentral email (`@berkeley.edu`)	
TA	
Name of the person to your left	
Name of the person to your right	
Room in which you are taking exam	
Seat number in the exam room	
I pledge my honor that during this examination I have neither given nor received assistance. **(please sign)**	

Reference Material.

```
# Linked Lists (implementations not shown)

empty = ...  # The empty list

def link(first, rest):
    """A linked list whose first element is FIRST and the linked list
    REST is the rest of the list."""

def first(lnklst):
    """The first item in linked list LNKLST."""

def rest(lnklst):
    """The list following the first item in linked list LNKLST."""

def isempty(lnklst):
    """True if LNKLST is empty."""

def print_link(lnklst):
    """Prints the linked list LNKLST in the format (v0, v1, ...)."""
```

1. (12 points) Evaluate These!

For each of the expressions in the table below, write the output displayed by the interactive Python interpreter when the expression is evaluated. The output may have multiple lines. If an error occurs, write "`Error`". If an expression yields (or prints) a function, write "`<Function>`". No answer requires more than 3 lines. (It's possible that all of them require even fewer.) The first two rows have been provided as examples.

The interactive interpreter displays the value of a successfully evaluated expression, unless it is `None`, plus all values passed to `print`.

Assume that `python3` has executed the statements on the left:

```
def w(L):
  if len(L) == 0:
    return L
  elif L[0] in L[1:]:
    return w(L[1:])
  else:
    return [L[0]] + w(L[1:])

def reduce(f, L, init):
  if len(L) == 0:
    return init
  else:
    return reduce(f, L[1:],
                  f(init, L[0]))
```

Expression	**Interactive Output**
`pow(2, 3)`	8
`print(4, 5) + 1`	4 5 Error
`1 + (4 and 6) + (5 or 0) + (0 and 8)`	12

`f = lambda x: x`
`g = lambda y: f(g)`
`g(2)(2)`
	`<Function>`

`0 and print(2)`	0
`range(1,20)[-2]`	18
`w([1, 2, 3, 1, 8, 2])`	[3,1,8 2]

`f = lambda x: lambda y: \`
`  lambda z: x+y+z`
`reduce(lambda g, x: g(x),`
`  [1, 2, 3], f)`
	6

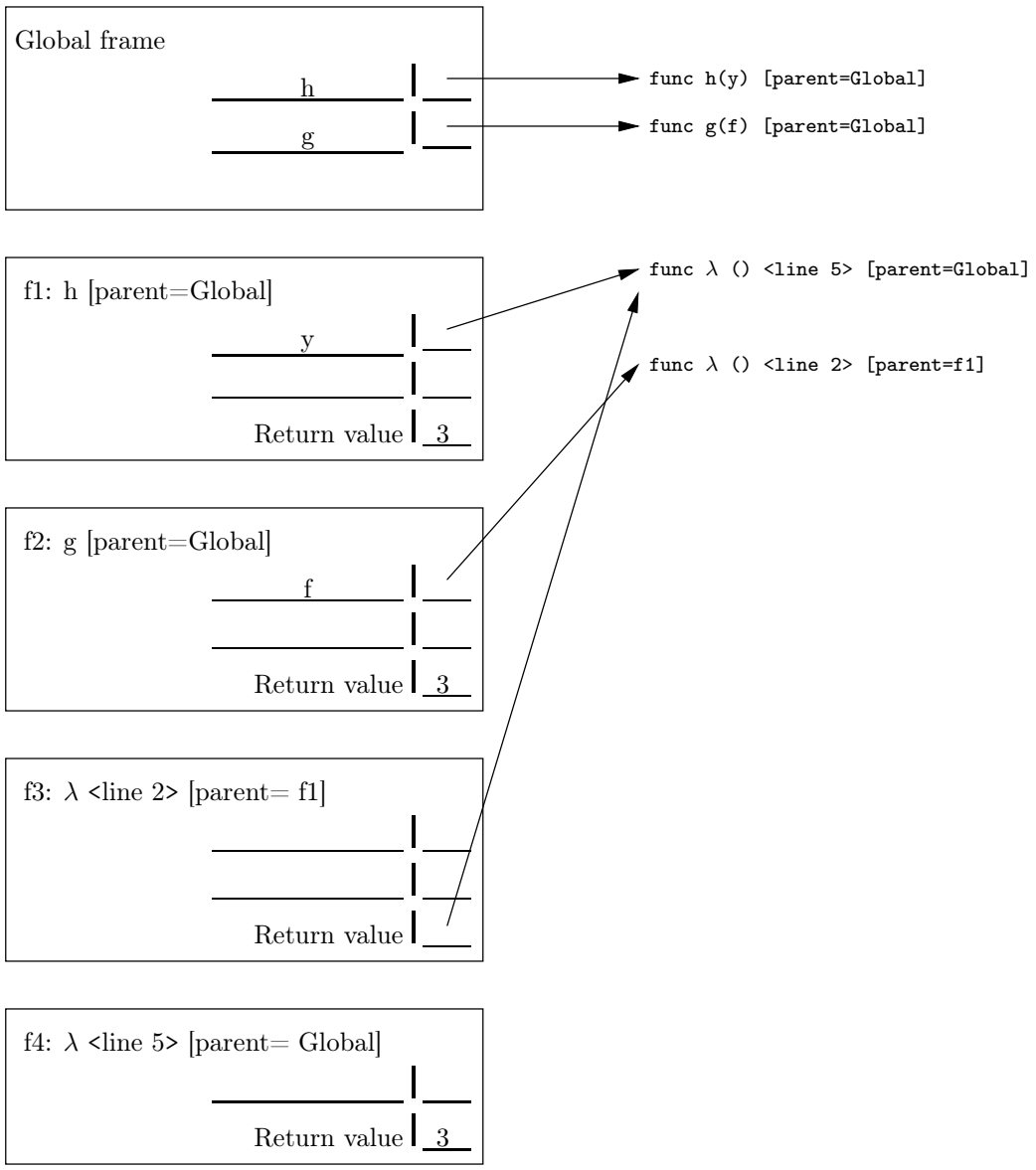
2. (12 points) Environmental Policy

(a) (6 pt) 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, frames, or blank lambda function values.*

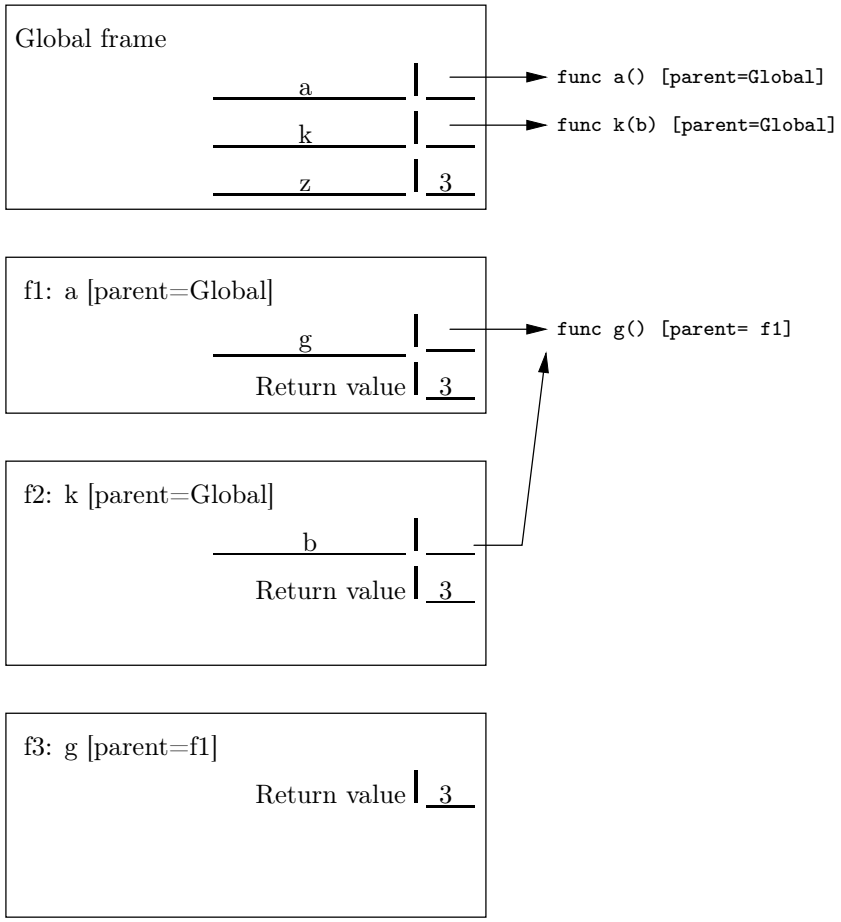
A complete answer will:

- Add all missing names and parent annotations to all local frames.
- Add all missing values created or referenced during execution.
- Show the return value for each local frame.

```
1 def h(y):
2     return g(lambda: y)
3 def g(f):
4     return f()()
5 h(lambda: 3)
```



(b) (6 pt) The environment diagram below corresponds to the execution of a certain program. The frames shown were created in top-to-bottom order, and at one point they were all simultaneously active (their functions had not returned). The diagram shows the situation right after all the functions have returned. Write a Python program whose execution corresponds to this environment diagram. Many answers are possible, but as a guideline, fewer than 10 lines are really needed. In any case, your answer must not create extra frames or additional functions.



Write solution here.

```
def a():
    def g():
        return 3
    return k(g)

def k(b):
    return b()

z = a()
```

3. (4 points) Sequence Checking

Fill in the following function so that it fulfills its comment.

```
def make_checker(relation, start, end):
    """Assumes that START and END are integers and RELATION is a two-argument
    function that returns true/false values.  Returns a function that, given
    a function f as input, returns True if RELATION returns true for
    all adjacent values in the sequence f(START), f(START+1), ... f(END-1).
    For example, eq_chk, below, checks that the values returned by
    its argument function for values 0-4 are all equal, while up_chk checks
    that the values returned by the argument function are in strictly
    increasing order.

    >>> eq_chk = make_checker(lambda x, y: x == y, 0, 5) # Check all equal
    >>> eq_chk(lambda x: 3)
    True
    >>> eq_chk(lambda x: x)
    False
    >>> up_chk = make_checker(lambda x, y: x < y, 0, 5) # Check increasing
    >>> up_chk(lambda x: x)
    True
    >>> up_chk(lambda x: 3)
    False
    """

    def checker(f):
        for k in range(start + 1, end):
            if not relation(f(k-1), f(k)):
                return False
        return True
    return checker
```

4. (1 points) Extra

In the formula $a = 0.4 + 0.3 \cdot 2^m$ ($m = -\inf, 0, 1, 2, \ldots$), what does a refer to?

The approximate positions of planets (in AU). This is known as Bode's Law.

5. (4 points) Insert

Fill in the following function to fulfill its comment. The linked-list interface that you should use is given on page **??**. **Warning:** this problem deals with linked lists, *NOT* Python lists or tuples. You cannot use `+`, `len()`, indexing (`L[k]`), or list construction (`[...]`).

```
def link_insert(lnklst, value, before):
    """Return a linked list identical to LNKLST, but with VALUE inserted just
    before the first occurrence of BEFORE in the list, if any.  The returned
    list is identical to LNKLST if BEFORE does not occur in LNKLST.
    The operation is non-destructive.
    >>> L = link(2, link(3, link(7, link(1))))
    >>> print_link(L)
    (2, 3, 7, 1)
    >>> Q = link_insert(L, 19, 7)
    >>> print_link(Q)
    (2, 3, 19, 7, 1)
    >>> print_link(link_insert(L, 19, 20))
    (2, 3, 7, 1)
    """

    if isempty(lnklst):
        return lnklst
    elif first(lnklst) == before:
        return link(value, lnklst)
    else:
        return link(first(lnklst), link_insert(rest(lnklst), value, before))
```

6. (4 points) Longest Nondecreasing Suffix

Consider a function `up_suffix` that is supposed to return the longest suffix of a Python list of integers such that the suffix consists of nondecreasing values. For example, when applied to

```
[1, 2, 3, 4, 5, 1, 3, 3, 4]
```

it should yield `[1, 3, 3, 4]`.

Fill in the following function to do this:

```
def up_suffix(L):
    """Returns the longest non-descending suffix of Python list L. """

    def longest_suffix_start(L):
        """The index in L of the beginning of the longest nondecreasing suffix
        of L. For the empty list, returns 0.
        >>> longest_suffix_start([1, 2, 3, 4, 5, 1, 3, 3, 4])
        5
        >>> longest_suffix_start([2, 4, 6, 8, 10])
        0
        """

        k = len(L) - 1

        while k > 0 and L[k-1] <= L[k]:
            k -= 1
        return max(k, 0)
        # We were not fussy about people who returned -1 for the empty list,
        # so just return k was all right as well.

    return L[longest_suffix_start(L):]
```

7. (4 points) Post's Problem

Consider two Python lists of strings, where the two lists have equal length, N:

```
A = [ "a", "ab", "bba"]
B = [ "baa", "aa", "bb" ]
```

Is there a sequence of integers—i_1, i_2, ... i_m—where $m > 0$ and $0 \leq i_k < N$ for all k, such that

$$A[i_1] + A[i_2] + \cdots + A[i_m] = B[i_1] + B[i_2] + \cdots + B[i_m]?$$

This is called the *Post Correspondence Problem*. For this `A` and `B`, the answer is yes for $m = 4$: (2, 1, 2, 0), because

```
A[2] + A[1] + A[2] + A[0] == "bba" + "ab" + "bba" + "a"  == "bbaabbbaa"
B[2] + B[1] + B[2] + B[0] == "bb" + "aa" + "bb" + "baa" == "bbaabbbaa"
```

On the other hand, the answer is no if we limit m to 3 (so we cannot add more than three strings), or if we shorten the lists by removing `A[0]` and `B[0]`. Fill in the following function to determine whether there is a solution.

```
def correspond(A, B, M):
    """Assuming A and B are lists of strings with len(A) == len(B), and M
    is an integer, returns true iff there is a sequence of indices into A
    and B, (i1, i2, ..., im), where 1 <= m <= M, such that
        A[i1] + A[i2] + ... + A[im] == B[i1] + B[i2] + ... + B[im].
    """
    N = len(A)

    def can_correspond(sa, sb, M):
        """Return true iff there is some sequence of indices into A and
        B, (i1, i2, ..., im), where 1 <= m <= M, such that
        SA + A[i1] + A[i2] + ... + A[im] == SB + B[i1] + B[i2] + ... + B[im].
        Assumes M is a non-negative integer.
        """

        if M <= 0:
            return False
        else:
            for i in range(N):
                ta = sa + A[i]
                tb = sb + B[i]
                if ta == tb:
                    return True
                elif can_correspond(ta, tb, M - 1):
                    return True
            return False

    return can_correspond("", "", M)
```