CS 61A Structure and Interpretation of Computer Programs
Summer 2017

Mock Final

- You have 1 hours and 20 minutes to complete this exam.
- This exam is closed book, closed notes, closed computer, closed calculator, except *four* 8.5” × 11” cheat sheets.
- Mark your answers **on the exam itself**. We will *not* grade answers written on scratch paper.
- For multiple choice questions, **fill in each option or choice completely.**
  - □ means mark **all options** that apply
  - ○ means mark a **single choice**

Last name	
First name	
Student ID number	
CalCentral email (`_@berkeley.edu`)	
Teaching Assistant	○ Alex Stennet ○ Angela Kwon ○ Ashley Chien ○ Joyce Luong ○ Karthik Bharathala ○ Kavi Gupta ○ Kelly Chen ○ Michael Gibbes ○ Michelle Hwang ○ Mitas Ray ○ Rocky Duan ○ Samantha Wong
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)**	

0. (0 points) Determination What’s been fun? What are you grateful for?

1. (10 points) Talking to Ducks

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", but include all output displayed before the error. If a function value is displayed, write "Function".

```
class Duck:
    family = []
    ducks = 0

    def __init__(self, name):
        self.name = name
        Duck.family.append(self)
        Duck.ducks += 1

    def __iter__(self):
        while True:
            yield Duck.family[0]
            first = Duck.family.pop(0)
            Duck.family.append(first)

    def __str__(self):
        return 'A Duck'

    def __repr__(self):
        return 'Duck("' + self.name + '")'

def clean(self):
    self.family = []
    return self
```

```
class Duckling:
    mother_duck = Duck

    def __init__(self, name):
        Duck.__init__(self, name)
        ducks = 0

    def __repr__(self):
        return 'Duckling("' + self.name + '")'

class Swan(Duckling):
    def __init__(self, name='Autumn'):
        Duckling.__init__(self, name)
        self.mother_duck = self.mother_duck('Swan')

    def __iter__(self):
        Duckling.next = Duck.__iter__(self)
        while True:
            yield next(Duckling.next)

drake = Duckling('Drake')
helen = drake.mother_duck('Helen')
drakerator = iter(helen)
```

Expression	Interactive Output
`pow(2, 3)`	8
`print(4, 5) + 1`	4 5 Error
`print(print(next(drakerator)), next(drakerator))`	
`Duck.ducks`	

`different = Swan()`
`different.mother_duck.ducks`	

`next(iter(different))`	
`Duck.family[-2:]`	
`clean(different.mother_duck)`	
`Duck.family[-1]`	

2. (8 points) A Link to the Past

Implement `slice_reverse` which takes a linked list `s` and mutatively reverses the elements on the interval, $[i, j)$ (including i but excluding j). Assume `s` is zero-indexed, $0 < i < j$, and that `s` has at least j elements.

You must use mutation; solutions which call the `Link` constructor will not receive credit. The `Link` class reference is provided below.

```
class Link:

    empty = ()

    def __init__(self, first, rest=empty):
        assert rest is Link.empty or isinstance(rest, Link)
        self.first = first
        self.rest = rest

def slice_reverse(s, i, j):
    """
    >>> s = Link(1, Link(2, Link(3)))
    >>> slice_reverse(s, 1, 2)
    >>> s
    Link(1, Link(2, Link(3)))
    >>> s = Link(1, Link(2, Link(3, Link(4, Link(5)))))
    >>> slice_reverse(s, 2, 4)
    >>> s
    Link(1, Link(2, Link(4, Link(3, Link(5)))))
    """

    start = ____________________________________________________

    for ________________________________________________________:

        start = ________________________________________________

    reverse = Link.empty

    current = __________________________________________________

    for ________________________________________________________:

        ____________________________________________________________________________________________________

        current.rest = ________________________________________________________________________

        reverse = _____________________________________________________________________________

        current = _____________________________________________________________________________

    ____________________________________________________________

    ____________________________________________________________

    ____________________________________________________________
```

3. (6 points) Lost Woods

(a) (4 pt) A **Binary Search Tree** is a tree where each node contains either 0, 1, or 2 nodes and where the left branch (if present) contains values *strictly less than* (<) the root value, and the right branch (if present) contains values *strictly greater than* (>) the root value. The definition is recursive: both the left and right branches must themselves also be BST for the entire tree to be a BST.

Implement `is_binary` which that takes in a Tree `t`, and returns `True` if `t` is a Binary Search Tree and `False` otherwise. Trees can contain any number of branches, but if a tree contains only one branch, interpret it as a left branch.

```
class Tree:

    def __init__(self, root, branches=[]):
        self.root = root
        self.branches = list(branches)

    def is_leaf(self):
        return not self.branches

def is_binary(t):

    def binary(t, lo, hi):

        if _____________________________________:

            if t.is_leaf():

                return True

            elif ___________________________________________________________________________:

                return ______________________________________________________________________

            elif ___________________________________________________________________________:

                return ______________________________________________________________________

        return False

    return binary(t, float('-inf'), float('inf'))
```

(b) (1 pt) Choose the $\Theta(\cdot)$ expression that best describes the runtime of `is_binary` on a well-formed **binary search tree** with n nodes. Assume the implementation of `is_binary` is optimal.

○ $\Theta(1)$ ○ $\Theta(\log n)$ ○ $\Theta(n)$ ○ $\Theta(n \log n)$ ○ $\Theta(n^2)$ ○ $\Theta(n^3)$ ○ $\Theta(2^n)$ ○ $\Theta(3^n)$

(c) (1 pt) Choose the $\Theta(\cdot)$ expression that best describes the runtime of `is_binary` on a **tree where each node contains 3 branches** and the **overall height** of the tree is n. Assume the implementation of `is_binary` is optimal.

○ $\Theta(1)$ ○ $\Theta(\log n)$ ○ $\Theta(n)$ ○ $\Theta(n \log n)$ ○ $\Theta(n^2)$ ○ $\Theta(n^3)$ ○ $\Theta(2^n)$ ○ $\Theta(3^n)$

4. (0 points) Designated Exam Chillout Zone

5. (8 points) ▲

(a) **(4 pt)** Implement `merge` which takes in two sorted lists of numbers and returns a new sorted list containing all the values. Ties can be broken in either direction.

```
(define (merge lst1 lst2)

  (cond (__________________________________________________________________________)

        (__________________________________________________________________________)

        (__________________________________________________________________________)

        (else (_____________________________________________________________________))

  ))

scm> (merge '(1 2 3 4 5) '(6 7 8 9 10))
(1 2 3 4 5 6 7 8 9 10)
scm> (merge '(1 3 5 7 9) '(2 4 6 8 10))
(1 2 3 4 5 6 7 8 9 10)
scm> (merge '(3 4 7 9 10) '(1 2 5 6 8))
(1 2 3 4 5 6 7 8 9 10)
scm> (merge '() '())
()
```

(b) **(1 pt)** Choose the $\Theta(\cdot)$ expression that best describes the runtime of `merge` where the length of each list is n. Assume the implementation of `merge` is correct, optimized, but not tail-recursive.

○ $\Theta(1)$ ○ $\Theta(\log n)$ ○ $\Theta(n)$ ○ $\Theta(n \log n)$ ○ $\Theta(n^2)$ ○ $\Theta(n^3)$ ○ $\Theta(2^n)$ ○ $\Theta(3^n)$

(c) **(1 pt)** Choose the $\Theta(\cdot)$ expression that best describes the **number of frames opened** during the execution of `merge` where the length of each list is n. Assume the implementation of `merge` is correct, optimized, but not tail-recursive.

○ $\Theta(1)$ ○ $\Theta(\log n)$ ○ $\Theta(n)$ ○ $\Theta(n \log n)$ ○ $\Theta(n^2)$ ○ $\Theta(n^3)$ ○ $\Theta(2^n)$ ○ $\Theta(3^n)$

(d) **(1 pt)** Implement `trimerge` which takes in three sorted lists of numbers and returns a new sorted list containing all the values. You may use `merge`; assume it is implemented correctly.

```
(define (trimerge lst1 lst2 lst3)

  (________________________________________________________________________________)

)

scm> (trimerge '(1 2 3) '(4 5 6) '(7 8 9))
(1 2 3 4 5 6 7 8 9)
scm> (trimerge '(3 7 8) '(1 4 5) '(2 6 9))
(1 2 3 4 5 6 7 8 9)
scm> (trimerge '() '() '())
()
```

(e) **(1 pt)** Choose the $\Theta(\cdot)$ expression that best describes the runtime of `trimerge` where the length of each list is n. Assume the implementation of `trimerge` is optimal.

○ $\Theta(1)$ ○ $\Theta(\log n)$ ○ $\Theta(n)$ ○ $\Theta(n \log n)$ ○ $\Theta(n^2)$ ○ $\Theta(n^3)$ ○ $\Theta(2^n)$ ○ $\Theta(3^n)$

6. (8 points) Telephoney Booth

Suppose you're analyzing data collected from a regional telephone operator. Their database consists of the following *schemas* which define the columns of each table.

```
create table location(lname, areacode)
create table people(pid, pname, areacode)
create table calls(cid, date, from, to, duration)
```

- The `areacode` data in the `people` table references the area codes defined in `location`.
- The `from` and `to` columns of `calls` each reference one person's `pid`.

(a) (2 pt) Select the names of all pairs of people who called each other on the date `2017-07-04`.

```
select _________________________________________

    from _______________________________________________________________________

    where ______________________________________________________________________;
```

(b) (3 pt) Select the name of the location and the number of calls to that location for the location which has had the most phone calls made to it.

```
select _________________________________________________________________________

    from _______________________________________________________________________

    where ______________________________________________________________________

    group by ___________________________________

    order by _____________________ desc limit 1;
```

(c) (3 pt) Select the name of the location of the person who made the longest call.

```
with durations as (select ______________________________________________________

                   from ________________________________________

                   where _______________________________________)

select _________________________________________________________

    from _______________________________________________________

    where ______________________________________________________

    order by _____________________________________ desc limit 1;
```