CS 61A Structure and Interpretation of Computer Programs

Summer 2015 FINAL

INSTRUCTIONS

- You have 3 hours to complete the exam.
- The exam is closed book, closed notes, closed computer, closed calculator, except one hand-written 8.5" × 11" crib sheet of your own creation and the official CS 61A midterm 1, midterm 2, and final exam study guides.
- Mark your answers **on the exam itself**. We will *not* grade answers written on scratch paper.

Last name	
First name	
Student ID number	
BearFacts email	
TA	
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. (14 points) What Would Scheme Do?

Assume that you have started the interactive Scheme interpreter from Project 4 using `python3 scheme.py` and executed the following statements:

```
(define lst '(1 2 (+ 3 4)))

(define y 1)

(define f (mu (x) (+ x y)))

(define g (lambda (x y) (f (+ x x))))

(define (h z) (z 3 7))
```

For each of the expressions in the table below, write the output displayed by the interpreter when the expression is evaluated. If an error occurs, write "Error". Assume each is executed in the same interactive session started above; **the result of executing one line may affect the result of other lines**.

Expression	**Interactive Output**
`(+ 1 2 3)`	6
`(/ 1 0)`	Error
`lst`	
`(car (cdr lst))`	
`(cons (cons 3 (cons 4 nil)) 5)`	
`(or 1 (/ 1 0))`	
`(and 1 (/ 1 0))`	
`(h +)`	
`(define x 5)`	
`(f (+ x x))`	
`(h g)`	
`(let ((a 1) (2 2)) a)`	
`(let ((x 1) (y x)) y)`	
`(f y)`	

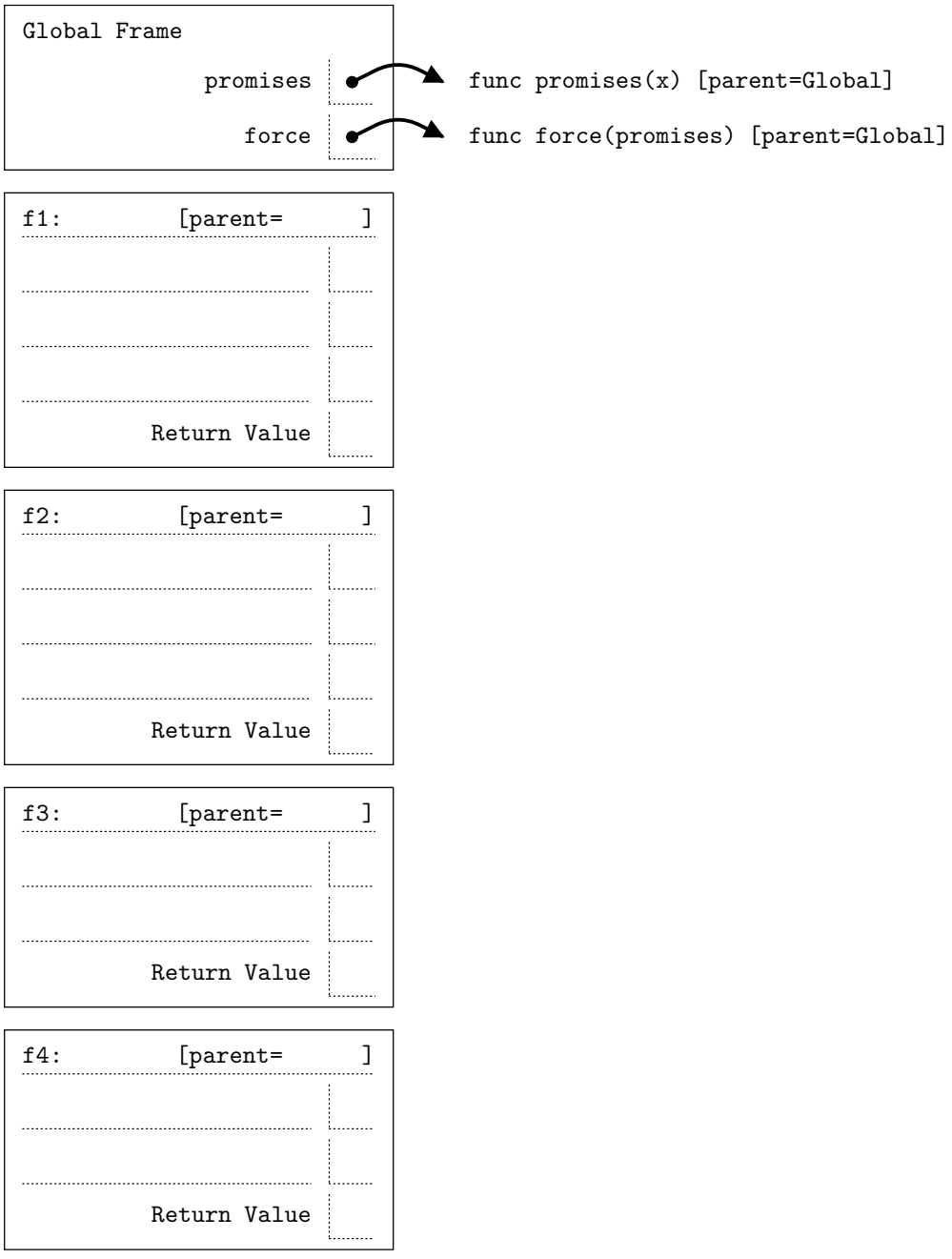
2. (16 points) Broken promises

(a) (8 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 only need to show the final state of each frame. *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 local frames.
- Add all missing values created or referenced during execution.
- Show the return value for each local frame.

```
1  def promises(x):
2      delayed = [0]
3      while x > 0:
4          delayed.append(
5              lambda y: y - x)
6          x = x - 1
7      return delayed
8
9  def force(promises):
10     res = [0]
11     x = 1
12     while x < len(promises):
13         res.append(
14             promises[x](7))
15         x = x + 1
16
17 force(promises(2))
```

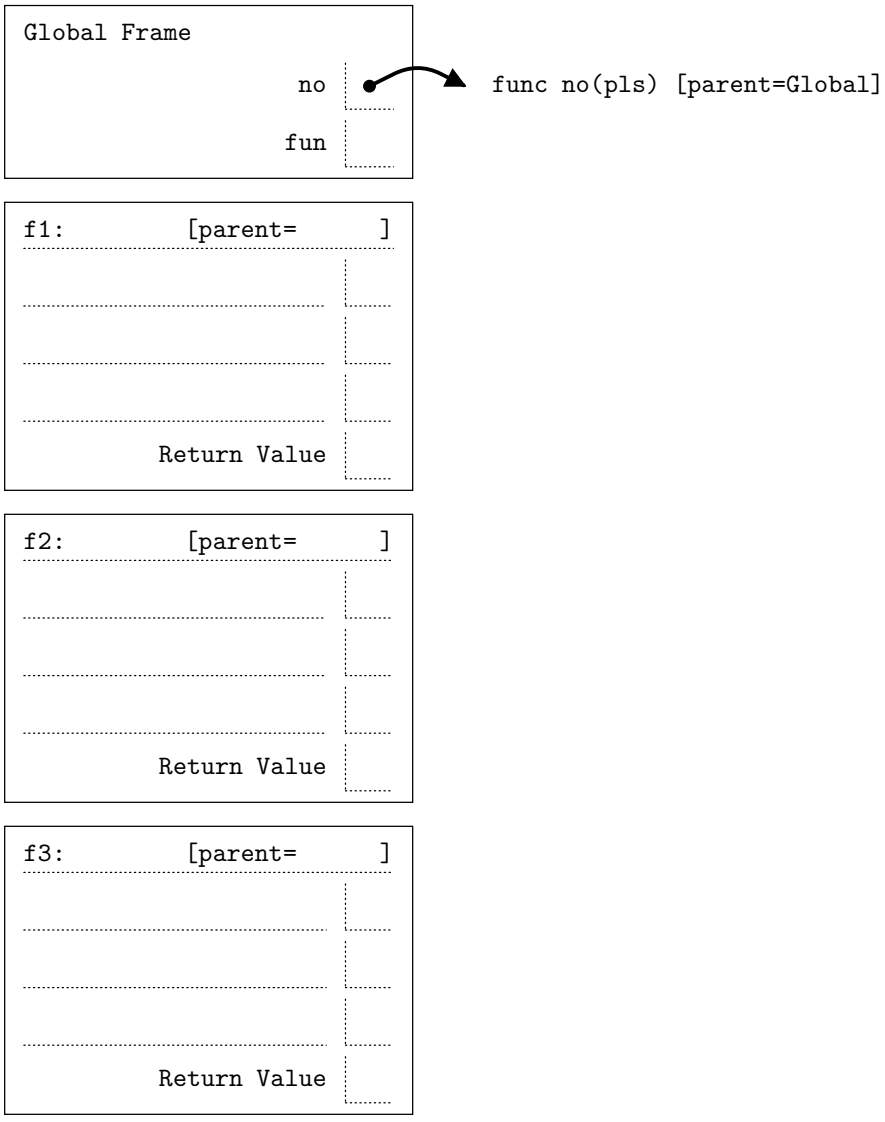


(b) **(8 pt)** Fill in the environment diagram that results from executing the code below until the entire program is finished, an error occurs, or all frames are filled. You only need to show the final state of each frame. *You may not need to use all of the spaces or frames.*

A complete answer will:

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

```
1  def no(pls):
2      def y(y):
3          fun.append(y)
4          pls[0] = pls
5          def y(y):
6              nonlocal pls
7              pls = pls + [y]
8          return y
9      return y
10
11 fun = [1, [2]]
12 no(fun)(3)(0)
```



3. (4 points) Cracking the Coding Interview

The `fizzbuzz` question is a standard software engineering interview question that goes as follows:

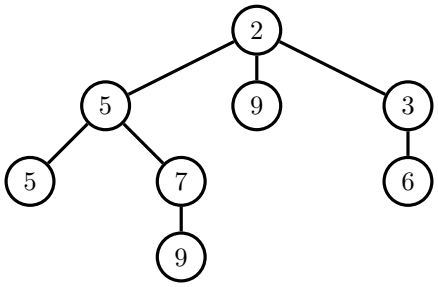
Implement `fizzbuzz(n)`, which prints numbers from 1 to `n` (inclusive). However, for numbers divisible by 3, print "fizz". For numbers divisible by 5, print "buzz". For numbers divisible by both 3 and 5, print "fizzbuzz'.

A recent Reddit thread expressed concern that UC Berkeley computer science students did not know how to solve `fizzbuzz`. Prove them wrong.

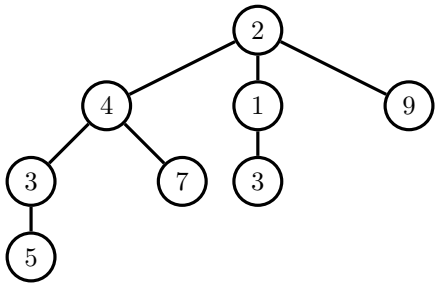
```
def fizzbuzz(n):
    """
    >>> result = fizzbuzz(16)
    1
    2
    fizz
    4
    buzz
    fizz
    7
    8
    fizz
    buzz
    11
    fizz
    13
    14
    fizzbuzz
    16
    >>> result is None
    True
    """
```

4. (8 points) Trees

(a) (4 pt) In computer science, a *min heap* is a tree such that every node's entry is less than or equal to all of the entries in that node's subtrees.



This is a valid min heap



Not a valid min heap. 4 is greater than 3. Also, 2 is greater than 1.

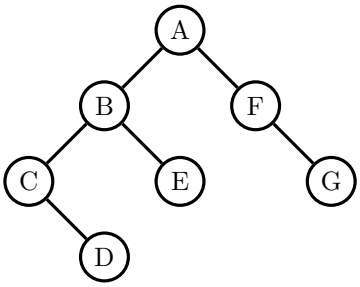
Implement `is_min_heap(t)`, where `t` is an instance of the `Tree` class (provided below). `is_min_heap` returns True if `t` is a valid min heap and returns False if it is not a valid min heap.

The `Tree` class has been provided for you below. A reasonable solution is around 6 lines, but you can use fewer or more if you want.

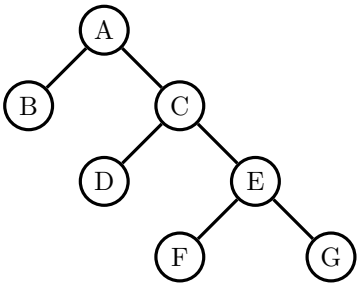
```
class Tree:
    def __init__(self, entry, subtrees=[]):
        self.entry = entry
        self.subtrees = list(subtrees)
    def is_leaf(self):
        return not self.subtrees

def is_min_heap(t):
```

(b) (4 pt)

A *full binary tree* is a `BinaryTree` where every node is either a leaf or has two children. Consider the following `BinaryTree`s:



Not a full binary tree; nodes C and F only have one child.



A full binary tree; every node is either a leaf or has two children.

Implement `is_full(b)`, where `b` is an instance of the `BinaryTree` class. `is_full` returns True if `b` is a full binary tree and returns False otherwise. You may assume `b` will not be the empty binary tree.

The `BinaryTree` class has been provided for you. A reasonable solution is around 5 to 7 lines, but you can use fewer or more if you want.

```
class BinaryTree:
    empty = ()
    def __init__(self, entry, left=empty, right=empty):
        self.entry = entry
        self.left = left
        self.right = right

def is_full(b):
```

5. (9 points) CS61A+

(a) (4 pt) The TAs are building a social networking website called CS61A+. The TAs plan to represent the network in a class called `Network` that supports the following method:

- `add_friend(user1, user2)`: adds `user1` and `user2` to each other's friends lists. If `user1` or `user2` are not in the `Network`, add them to the dictionary of friends. You may assume `user1` and `user2` are not already friends.

Help the TAs implement these two methods to make their social networking website popular!

```
class Network:
    """
    >>> cs61a_plus = Network()
    >>> cs61a_plus.add_friend('Robert', 'Jeffrey')
    >>> cs61a_plus.friends['Robert']
    ['Jeffrey']
    >>> cs61a_plus.friends['Jeffrey']
    ['Robert']
    >>> cs61a_plus.add_friend('Jessica', 'Robert')
    >>> cs61a_plus.friends['Robert']
    ['Jeffrey', 'Jessica']
    """
    def __init__(self):
        self.friends = {}       # Maps users to a list of their friends

    def add_friend(self, user1, user2):


        if ____________________________________________________:


            ________________________________________________


        if ____________________________________________________:


            ________________________________________________


        ____________________________________________________


        ____________________________________________________
```

(b) (5 pt) CS61A+ turns out to be unpopular. To attract more users, the TAs want to implement a feature that checks if two users have at most n degrees of separation. Consider the following CS61A+ `Network`:

```
self.friends = {'Robert':  ['Jeffrey', 'Jessica'],
                'Jeffrey': ['Robert', 'Jessica', 'Yulin'],
                'Jessica': ['Robert', 'Jeffrey', 'Yulin'],
                'Yulin':   ['Jeffrey', 'Jessica'],
                'Albert':  []}
```

- There are 0 degrees of separation between a person and themselves.
- There is 1 degree of separation between Robert and Jeffrey, because they are direct friends.
- There are 2 degrees of separation between Robert and Yulin (Robert → Jessica → Yulin)
- The degree of separation between Albert and anyone else is undefined, since Albert has no friends.

Implement `degrees(user1, user2, n)`, which returns True if `user1` and `user2` are separated by **at most** n **degrees** (fewer degrees is okay). You can assume that `user1` and `user2` are already in the `Network`.

```
class Network:
    # Code from previous question
    def degrees(self, user1, user2, n):
        """In these doctests, assume cs61a_plus is a Network with the
        dictionary of friends described in the example.

        >>> cs61a_plus.degrees('Robert', 'Yulin', 2)    # Exactly 2 degrees
        True
        >>> cs61a_plus.degrees('Robert', 'Jessica', 2)  # Less than 2 degrees
        True
        >>> cs61a_plus.degrees('Yulin', 'Robert', 1)    # More than 1 degree
        False
        >>> cs61a_plus.degrees('Robert', 'Robert', 2)   # 0 degrees
        True
        >>> cs61a_plus.degrees('Albert', 'Jessica', 10) # No friends!
        False
        """
        if ____________________________________:


            return ___________


        elif ____________________________________:


            return ___________


        for friend in _________________________________:


            if ____________________________________:
                return True


        return ____________________________________
```

6. (14 points) Unique-Lo

(a) (5 pt) Implement an iterator class called `Unique`. The `__init__` method for `Unique` takes an iterable of numbers. The `Unique` iterator represents a sequence that contains each number in the iterable only once. Only the first occurrence of each element should be included in `Unique`; the order should remain the same.

See the doctests for expected behavior. A reasonable solution is around 4 to 5 lines, but you can use fewer or more if you want.

```
class Unique:
    """
    >>> list(Unique(range(5)))      # All elements are unique
    [0, 1, 2, 3, 4]

    >>> x = [1, 2, 1, 5, 2, 5]
    >>> list(Unique(x))
    [1, 2, 5]

    >>> s = Unique([3, 3, 3, 3, 3, 3, 3, 3, 4])
    >>> next(s)
    3
    >>> next(s)
    4
    """

    def __init__(self, iterable):


        self.iterator = iter(iterable)


        ------------------------------------------------------------


    def __iter__(self):

        return self

    def __next__(self):


        ------------------------------------------------------------


        ------------------------------------------------------------


        ------------------------------------------------------------


        ------------------------------------------------------------


        ------------------------------------------------------------
```

(b) **(4 pt)** Implement a Scheme function called `contains`, which takes in a number and a well-formed Scheme list of numbers. `contains` returns True if the number appears in the list.

```
scm> (contains 3 (1 2 3 4))
True
scm> (contains 42 (3 4 1))
False
```

A reasonable solution uses 3 to 4 additional lines of code, but you can use fewer or more if you want.

```
(define (contains num lst)









)
```

(c) **(5 pt)** Implement a Scheme function called `unique-stream`, which takes in a stream of numbers and returns a new stream that contains each number of the input stream only once. Only the first occurence of each number should be included, and it should be included in the order in which it appears in the original stream.

Hint: you may assume the `contains` function from the previous question is implemented correctly and use it in this question. You do not have to write a tail recursive function for this question.

```
scm> (define s (cons-stream 1 (cons-stream 2 (cons-stream 1
                              (cons-stream 5 (cons-stream 2 nil))))))
scm> (define result (unique-stream s))
scm> (stream-to-list result 3)
(1 2 5)
```

A reasonable solution uses 3 to 5 additional lines of code, but you can use fewer or more if you want.

```
(define (unique-stream s)
    (define (helper s seen)
        (cond ((null? s) nil)









        )
    (helper s nil)
)
```

7. (6 points) Counting stars

(a) (4 pt) Implement a tail-recursive Scheme function called `count`, which takes in a number and a well-formed Scheme list of numbers. `count` returns the number of times the number appears in the list.

```
scm> (count 3 (1 2 3 4 3))
2
scm> (count 42 (3 4 2))
0
```

Remember, **your solution must be tail recursive**. A reasonable solution uses 3 to 4 additional lines of code, but you can use fewer or more if you want. Make sure to fill in the blanks in the second-to-last line.

```
(define (count num lst)
    (define (helper lst total)









    )
    (helper ____________ ____________)
)
```

(b) (2 pt) When the Berts eat at a restaurant, they record a review in a SQL table called `reviews`:

restaurant	user	stars	review
Barney's	Albert	4	Used to like it
Chipotle	Robert	5	BOGO! BOGO!
Eureka	Albert	5	My favorite!
Bongo Burger	Albert	2	When I'm desperate
Umami Burger	Albert	5	I love truffle fries!

Write an SQL query to figure out how many restaurants Albert gave 4 or 5 stars. Using the table above, the output to your query should be the following:

stars	number of reviews
4	1
5	2

```
select ____________________________________ from reviews

  where ____________________________________

  group by _________________________________

  having ___________________________________;
```

8. (9 points) Subsequently...

(a) (5 pt) Implement a Python function called `subsequences(lst)`, which takes in a Python list of unique numbers. `subsequences` returns a list of all subsequences of `lst`.

A list `A` is a *subsequence* of list `B` if `A` contains zero or more elements of `B` in the same order that they appear in `B`.

See the doctests for expected behavior. We will not penalize you for returning the subsequences in a different order than the doctests. However, the order of elements within a subsequence must be correct.

```
def subsequences(lst):
    """
    >>> result = subsequences([1, 2, 3])
    >>> for subsequence in result:
    ...     print(subsequence)
    []
    [3]
    [2]
    [2, 3]
    [1]
    [1, 3]
    [1, 2]
    [1, 2, 3]
    """

    if lst == []:


        return ____________


    without_first = _____________________________________________


    with_first = [____________________ for ________ in ____________________]


    return ________________________________________
```

(b) **(4 pt)** In the previous question, we had to explicitly tell Python how to construct the list of subsequences, because Python is an imperative programming language.

Recall that, since Logic is a declarative programming language, we just have to describe what a subsequence looks like, and then Logic will figure out all of the subsequences for us with the following query:

```
logic> (query (subsequence (1 2 3) ?what))
what: (1 2 3)
what: (1 2)
what: (1 3)
what: (1)
what: (2 3)
what: (2)
what: (3)
what: ()
```

In this question, we will write a set of facts for the `subsequence` relation, which describes what a subsequence of a list looks looks like.

(1 pt) Write a fact for the base case:

```
(fact (subsequence ____________ ____________))
```

(2 pt) Next, write a recursive fact for the case where the first element of the list is the same as the first element in the subsequence. For example,

```
logic> (query (subsequence (1 2 3 4) (1 2 4)))
Success!
```

```
(fact (subsequence ___________________ ___________________)

      (subsequence ____________ ____________))
```

(1 pt) Finally, write a recursive fact for the case where the first element of the list might not be the same as the first element in the subsequence. For example,

```
logic> (query (subsequence (1 2 3 4) (2 3)))
Success!
```

```
(fact (subsequence ___________________ ___________________)

      (subsequence ____________ ____________))
```

9. (0 points) Extra lectures, extra credit

The following questions are each worth 0.5 extra credit points. Each question is from one of the special interest lectures given in the last week of class.

(a) (0.5 pt) Machine Learning: What is the difference between a reward function and a value function?

(b) (0.5 pt) Computability and Complexity: What is the difference between P and NP?

(c) (0.5 pt) Computational Biology: What is the edit distance between "sitting" and "biting", assuming mismatches have a cost of 1?

10. (0 points) Third time's the charm!

In each of the three boxes below, write a positive integer. If one of the numbers you pick is the lowest unique integer in the class, you get one extra credit point. In other words, you get three chances to write the smallest positive integer that you think no one else will write.

11. (0 points) Drawing to a close

Thanks for a great semester! We enjoyed teaching you this summer and we hope you enjoyed the class as well. Whether or not you plan on pursuing computer science in the future, we hope you learned a lot.

Feel free to leave a quick poem, haiku, joke, or drawing in the blank space below!

Blank