# CS 61A Fall 2013 — Structure and Interpretation of Computer Programs

Midterm 2 Solutions

**INSTRUCTIONS**

- You have 2 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 two official 61A midterm study guides attached to the back of this exam.
- Mark your answers ON THE EXAM ITSELF. If you are not sure of your answer you may wish to provide a *brief* explanation.

| | |
|---|---|
| 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 | Total |
|---|---|---|---|---|
| /14 | /14 | /12 | /10 | /50 |

**1. (14 points) Classy Costumes**

For each of the following expressions, write the value to which it evaluates. The first two rows have been provided as examples. If evaluation causes an error, write Error. If evaluation never completes, write Forever.

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

```
class Monster:
    vampire = {2: 'scary'}
    def werewolf(self):
        return self.vampire[2]

class Blob(Monster):
    vampire = {2: 'night'}
    def __init__(self, ghoul):
        vampire = {2: 'frankenstein'}
        self.witch = ghoul.vampire
        self.witch[3] = self

spooky = Blob(Monster)
spooky.werewolf = lambda self: Monster.vampire[2]
```

| Expression | Evaluates to |
|---|---|
| `square(5)` | 25 |
| `1/0` | Error |
| `[k+2 for k in range(4)]` | [2, 3, 4, 5] |
| `Monster.vampire[2][3]` | 'r' |
| `repr(len(spooky.witch))` | '2' |
| `spooky.witch[3] is not spooky` | False |
| `spooky.witch[2][0:4]` | 'scar' |
| `spooky.werewolf()` | Error |
| `Monster.werewolf(spooky)` | 'night' |

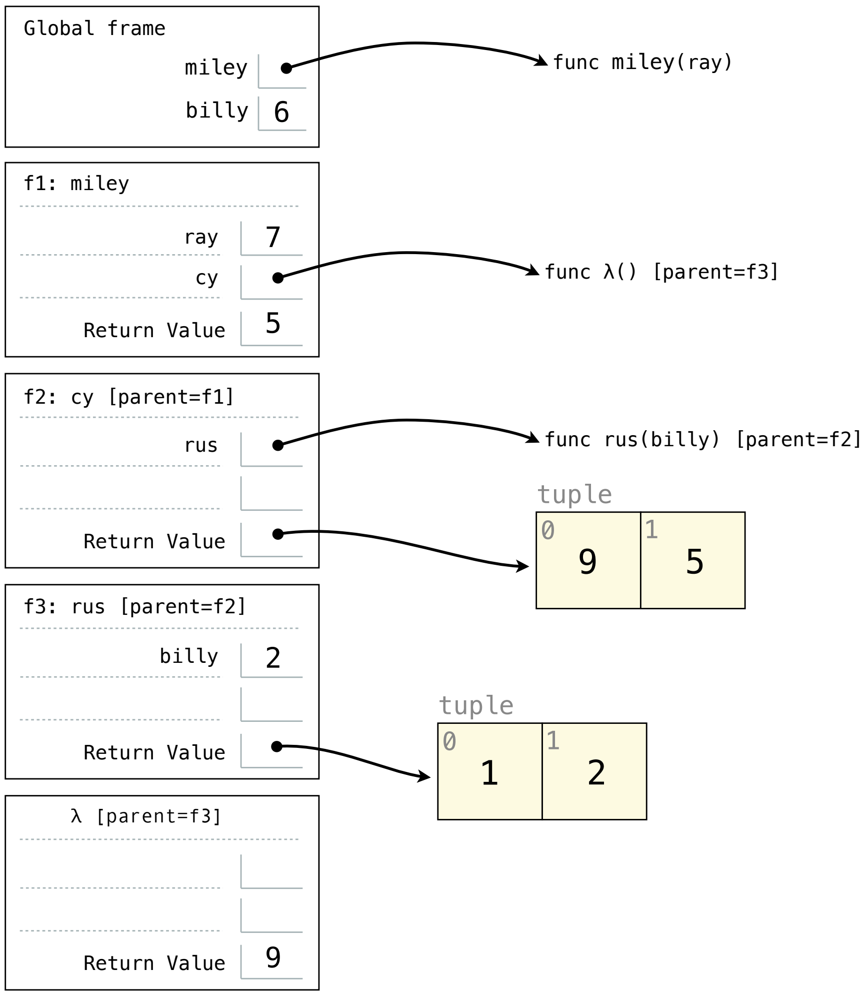
**2. (14 points) Wreaking Ball**

**(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 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.

```
def miley(ray):
    def cy():
        def rus(billy):
            nonlocal cy
            cy = lambda: billy + ray
            return (1, billy)
        if len(rus(2)) == 1:
            return (3, 4)
        else:
            return (cy(), 5)
    return cy()[1]
billy = 6
miley(7)
```



(b) **(6 pt)** Write the letter of the environment diagram that would result from executing each code snippet below, just after starting Python. The first blank is filled for you. Two different snippets may result in the same environment diagram. If none of the environment diagrams are correct, write N.

```
s = [1, 2]
t = list(s)
```

A

```
s = [[1, 2]]
t = s[0]
t[1] = list(s)
```

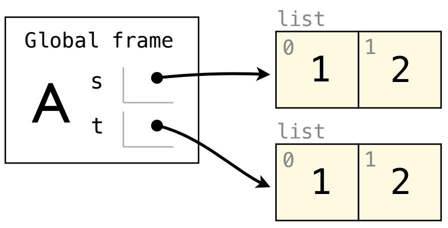
F

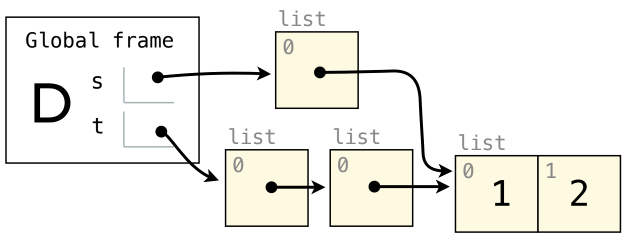
```
s = [[1, 2]]
t = s
t[0][1] = list(s)
```

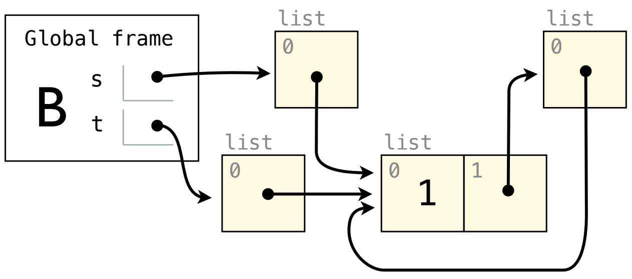
C

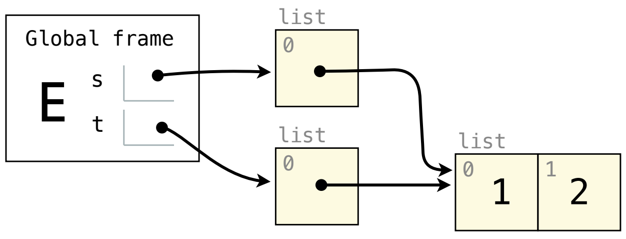
```
s = [[1, 2]]
t = list(s)
t[0] = list(s)
```

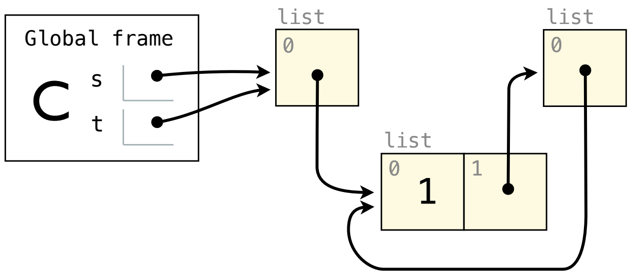
D

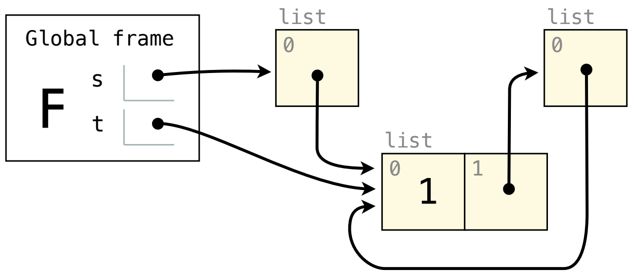












N: None of the above

**3. (12 points) Mutants**

**(a)** **(4 pt)** Given two `Rlist` arguments `a` and `b`, the function `merge(a, b)` changes `a` so that it also includes all elements of `b` at the end, but *does not* change `b`. After merging, changes to `b` *should not* affect `a`. Assume that `a` is not empty, but `b` may be empty. Complete the implementation by filling the blanks with expressions. The `Rlist` class is defined in your study guide.

```
def merge(a, b):
    """Add the elements of b to the end of a, mutating a but not b.

    >>> a = Rlist(1, Rlist(2, Rlist(3)))
    >>> b = Rlist(4, Rlist(5, Rlist(6)))
    >>> merge(a, b)
    >>> a  # a should be modified
    Rlist(1, Rlist(2, Rlist(3, Rlist(4, Rlist(5, Rlist(6))))))
    >>> b  # b should not be modified
    Rlist(4, Rlist(5, Rlist(6)))
    >>> b.first = 7  # modify the elements of b
    >>> b  # b should be modified
    Rlist(7, Rlist(5, Rlist(6)))
    >>> a  # a should not be modified
    Rlist(1, Rlist(2, Rlist(3, Rlist(4, Rlist(5, Rlist(6))))))
    """
    assert a is not Rlist.empty
    if b is Rlist.empty:
        return  # No entries to add to a.


    elif a.rest is Rlist.empty:


        a.rest = Rlist(b.first)


        merge(a.rest, b.rest)


    else:


        merge(a.rest, b)
```

**(b)** **(2 pt)** Define a mathematical function $f(m, n)$ such that evaluating a correct and efficient implementation of `merge(a, b)` on Rlist `a` of length $m$ and Rlist `b` of length $n$ requires $\Theta(f(m, n))$ function calls.

$f(m, n) = m + n$

**(c)** **(6 pt)** The function `fold_tree` takes in a three-argument function, a zero value, and a `Tree`. It returns the value of replacing `Tree` with the function and empty branches with the zero value. For each of `size`, `reverse`, and `repeated`, complete the inner function `f`. Each `f` *cannot* be recursive.

```
def fold_tree(fn, zero, tree):
    """Replaces the tree constructor with a 3-argument function.

    >>> t = Tree(3, Tree(5, None, Tree(3)), Tree(2))
    >>> f = lambda a, b, c: a + b + c
    >>> fold_tree(f, 0, t)  # is equivalent to the expression...
    13
    >>> f(3, f(5, 0, f(3, 0, 0)), f(2, 0, 0))
    13
    """
    if tree is None:
        return zero
    return fn(tree.entry, fold_tree(fn, zero, tree.left),
                          fold_tree(fn, zero, tree.right))

def size(tree):
    """Return the number of trees contained in tree.

    >>> size(Tree(3, Tree(5, None, Tree(3)), Tree(2)))
    4
    """
    def f(entry, left, right):
        return 1 + left + right


    return fold_tree(f, 0, tree)

def reverse(tree):
    """Return a new tree swapping all left and right branches of tree.

    >>> reverse(Tree(3, Tree(5, None, Tree(3)), Tree(2)))
    Tree(3, Tree(2), Tree(5, Tree(3), None))
    """
    def f(entry, left, right):
        return Tree(entry, right, left)


    return fold_tree(f , None, tree)


def repeated(tree):
    """Return how many times the root entry of tree appears in tree.

    >>> repeated(Tree(3, Tree(5, None, Tree(3)), Tree(2)))  # 3 appears twice.
    2
    """
    def f(entry, left, right):
        return left + right + (1 if tree.entry == entry else 0)


    return fold_tree(f, 0, tree)
```

**4. (10 points) Expansion Mansion**

For a fraction `n/d` with `n < d`, its decimal expansion is written as a series of digits following a decimal point.

For example, 5/8 expands to 0.625. We can compute this result recursively. The numerator 5 times 10 is 50. 50 divided by 8 is **6** with remainder 2. 20 divided by 8 is **2** with remainder 4. 40 divided by 8 is **5** with remainder 0. The quotients in bold are the digits of the expansion. Each subsequent digit is the quotient of dividing 10 times the remainder of the previous digit by the denominator of the fraction.

**(a) (4 pt)** Assume that the decimal expansion of `n/d` is finite, `n` is positive, and `n < d`. The function `expand_finite` returns the digits of the decimal expasion as an `Rlist`. Complete it by filling in each blank with an expression. The `Rlist` class is defined in your study guide.

```
def expand_finite(n, d):
    """Return the finite decimal expansion of n/d as an Rlist. Assume n<d.

    >>> expand_finite(1, 2) # 1/2 = 0.5
    Rlist(5)
    >>> expand_finite(5, 8) # 5/8 = 0.625
    Rlist(6, Rlist(2, Rlist(5)))
    >>> expand_finite(3, 40) # 3/40 = 0.075
    Rlist(0, Rlist(7, Rlist(5)))
    """

    dividend = n * 10

    quotient, remainder = dividend // d, dividend % d

    if remainder == 0:

        return Rlist(quotient)

    else:

        return Rlist(quotient, expand_finite(remainder, d))
```

**(b) (2 pt)** The function `coerce_to_float` returns the float equal to an input `Rlist` representing the series of digits following the decimal point in a finite decimal expansion. Complete its implementation below.

```
def coerce_to_float(s):
    """Return a float equal to an Rlist encoding a series of digits.

    >>> coerce_to_float(expand_finite(1, 2))
    0.5
    >>> coerce_to_float(expand_finite(3, 40))
    0.075
    """

    if s is Rlist.empty:

        return 0

    else:

        return (s.first + coerce_to_float(s.rest)) / 10
```

**Repeating Decimal Expansions.** The decimal expansion of a rational number may be infinite, but can always be described by a finite (and possibly repeating) series of digits. We can represent a series of digits as a recursive list, which may contain itself.

**(c)** **(4 pt)** Assume that `n` is positive and `n < d`. The `expand` function returns representations of both finite and infinite decimal expansions. Complete it by filling in each blank with an expression or assignment statement.

```
def expand(n, d):
    """Return the decimal expansion of n/d as an Rlist. Assume n < d.

    >>> expand(1, 2) # 1/2 = 0.5
    Rlist(5)
    >>> expand(5, 8) # 5/8 = 0.625
    Rlist(6, Rlist(2, Rlist(5)))
    >>> third = expand(1, 3)  # 1/3 = 0.333333...
    >>> [third[i] for i in range(10)]
    [3, 3, 3, 3, 3, 3, 3, 3, 3, 3]
    >>> third.rest is third  # There is only one unique digit in 1/3
    True
    >>> fourteenth = expand(1, 14)  # 1/14 = 0.0714285714285...
    >>> [fourteenth[i] for i in range(10)]
    [0, 7, 1, 4, 2, 8, 5, 7, 1, 4]
    """

    return expand_using(n, d, {})

def expand_using(n, d, known):
    """Return the decimal expansion of n/d as an Rlist.
    known -- a dictionary from integer k to the decimal expansion of k/d.
    """

    if n in known:

        return known[n]

    else:

        dividend = n * 10

        quotient, remainder = dividend // d, dividend % d


        digits = Rlist(quotient)


        known[n] = digits


        if remainder > 0:


            digits.rest = expand_using(remainder, d, known)

        return digits
```