CS 61A Fall 2014

Structure and Interpretation of Computer Programs

Final Exam Solutions

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 3 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	Q. 5	Q. 6	Total
/14	/16	/12	/12	/18	/8	/80

1. (14 points) Representing Scheme Lists

For each row below, write the output displayed by the interactive Python interpreter when the expression is evaluated. Expressions are evaluated in order, and **expressions may affect later expressions**.

Whenever the interpreter would report an error, write Error. You *should* include any lines displayed before an error. *Reminder*: The interactive interpreter displays the `repr` string of the value of a successfully evaluated expression, unless it is `None`.

The `Pair` class from Project 4 is described on your final study guide. Recall that its `__str__` method returns a Scheme expression, and its `__repr__` method returns a Python expression. The full implementation of `Pair` and `nil` appear at the end of the exam as an appendix. Assume that you have started Python 3, loaded `Pair` and `nil` from `scheme_reader.py`, then executed the following:

```
blue = Pair(3, Pair(4, nil))
gold = Pair(Pair(6, 7), Pair(8, 9))

def process(s):
    cal = s
    while isinstance(cal, Pair):
        cal.bear = s
        cal = cal.second
    if cal is s:
        return cal
    else:
        return Pair(cal, Pair(s.first, process(s.second)))

def display(f, s):
    if isinstance(s, Pair):
        print(s.first, f(f, s.second))

y = lambda f: lambda x: f(f, x)
```

Expression	Output
`Pair(1, nil)`	`Pair(1, nil)`
`print(Pair(1, nil))`	`(1)`
`1/0`	Error
`print(print(3), 1/0)`	`3` Error
`print(Pair(2, blue))`	`(2 3 4)`
`print(gold)`	`((6 . 7) 8 . 9)`

Expression	Output
`process(blue.second)`	`Pair(nil, Pair(4, nil))`
`print(process(gold))`	`(9 (6 . 7) 9 8 . 9)`
`gold.second.bear.first`	`8`
`y(display)(gold)`	`8 None` `(6 . 7) None`

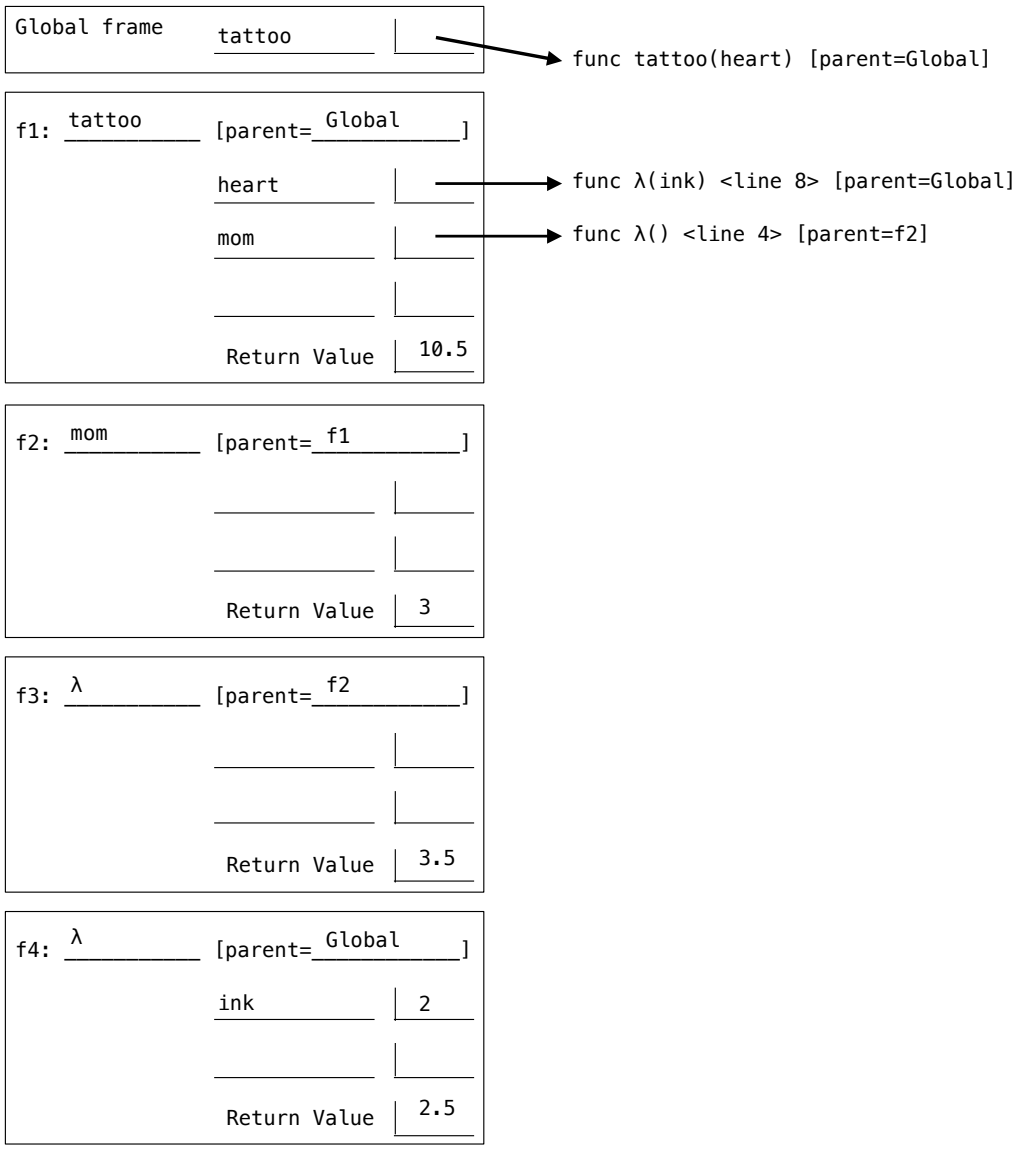
2. (16 points) Environments

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

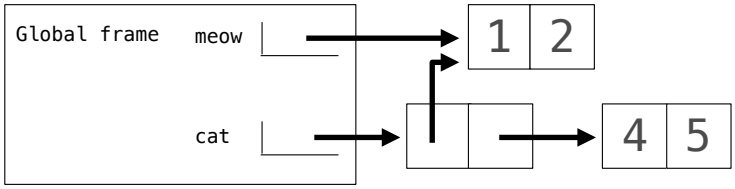
```
1 def tattoo(heart):
2     def mom():
3         nonlocal mom
4         mom = lambda: heart(2) + 1
5         return 3
6     return mom() + mom() + 4
7
8 tattoo(lambda ink: ink + 0.5)
```



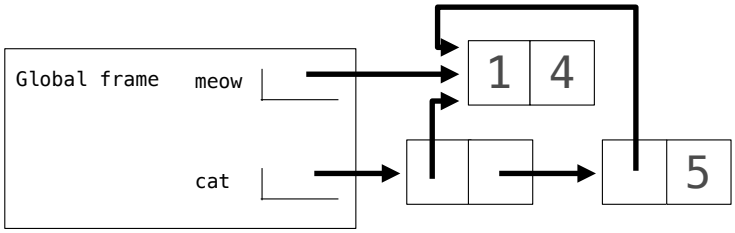
(b) **(6 pt)** For the six-line program below, fill in the three environment diagrams that would result after executing each pair of lines in order. **You must use box-and-pointer diagrams to represent list values. You do not need to write the word "list" or write index numbers.**

Important: All six lines of code are executed in order! Line 3 is executed after line 2 and line 5 after line 4.

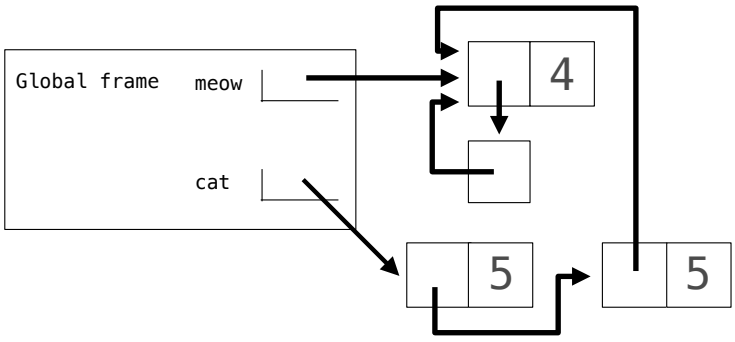
```
1 meow = [1, 2]
2 cat =  [meow, [4, 5]]
```



```
3 cat[0][1] = cat[1][0]
4 cat[meow[0]][0] = meow
```



```
5 meow[0] = [cat.pop(0)]
6 cat.extend(cat[0][1:])
```



(c) **(2 pt)** Circle the value, `True` or `False`, of each expression below when evaluated in the environment created by executing all six lines above. If you leave this question blank, you will receive 1 point.

(*True* / *False*) `meow is cat[0]`

(*True* / *False*) `meow[0][0] is cat[0][0]`

3. (12 points) Expression Trees

Your partner has created an interpreter for a language that can add or multiply positive integers. Expressions are represented as instances of the `Tree` class and must have one of the following three forms:

- (**Primitive**) A positive integer `entry` and no branches, representing an integer
- (**Combination**) The `entry '+'`, representing the sum of the values of its branches
- (**Combination**) The `entry '*'`, representing the product of the values of its branches

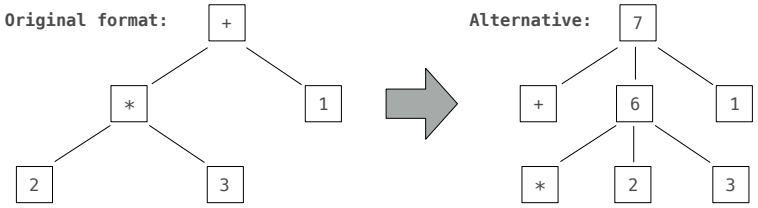
The `Tree` class is on the Midterm 2 Study Guide. The sum of no values is 0. The product of no values is 1.

(a) (6 pt) Unfortunately, multiplication in Python is broken on your computer. Implement `eval_with_add`, which evaluates an expression without using multiplication. You may fill the blanks with names or call expressions, but the only way you are allowed to combine two numbers is using addition.

```
def eval_with_add(t):
    """Evaluate an expression tree of * and + using only addition.

    >>> plus = Tree('+', [Tree(2), Tree(3)])
    >>> eval_with_add(plus)
    5
    >>> times = Tree('*', [Tree(2), Tree(3)])
    >>> eval_with_add(times)
    6
    >>> deep = Tree('*', [Tree(2), plus, times])
    >>> eval_with_add(deep)
    60
    >>> eval_with_add(Tree('*'))
    1
    """
    if t.entry == '+':

        return sum([eval_with_add(b) for b in t.branches])

    elif t.entry == '*':

        total = 1


        for b in t.branches:


            term, total = total, 0


            for _ in range(eval_with_add(b)):


                total = total + term

        return total

    else:

        return t.entry
```

(b) (6 pt) A TA suggests an alternative representation of an expression, in which the `entry` is the value of the expression. For combinations, the operator appears in the left-most (index 0) branch as a leaf.

Note: This question was updated from its original published version, which contained a flaw in the provided code.



Implement `transform`, which takes an expression and mutates all combinations so that their entries are values and their first branches are operators. In addition, `transform` should return the value of its argument. You may use the `calc_apply` function defined below.

```
def calc_apply(operator, args):
    if operator == '+':
        return sum(args)
    elif operator == '*':
        return product(args)

def product(vals):
    total = 1
    for v in vals:
        total *= v
    return total

def transform(t):
    """Transform expression tree t to have value entries and operators leaves.

    >>> seven = Tree('+', [Tree('*', [Tree(2), Tree(3)]), Tree(1)])
    >>> transform(seven)
    7
    >>> seven
    Tree(7, [Tree(+), Tree(6, [Tree(*), Tree(2), Tree(3)]), Tree(1)])
    """
    if t.entry in ('*', '+'):

        args = []

        for b in t.branches:


            args.append(transform(b))


        t.branches = [Tree(t.entry)] + t.branches


        t.entry = calc_apply(t.entry, args)


    return t.entry
```

4. (12 points) Lazy Sunday

(a) (4 pt) A *flat-map* operation maps a function over a sequence and flattens the result. Implement the `flat_map` method of the `FlatMapper` class. You may use at most 3 lines of code, indented however you choose.

```
class FlatMapper:
    """A FlatMapper takes a function fn that returns an iterable value. The
    flat_map method takes an iterable s and returns a generator over all values
    that are within the iterables returned by calling fn on each element of s.

    >>> stutter = lambda x: [x, x]
    >>> m = FlatMapper(stutter)
    >>> g = m.flat_map((2, 3, 4, 5))
    >>> type(g)
    <class 'generator'>
    >>> list(g)
    [2, 2, 3, 3, 4, 4, 5, 5]
    """


    def __init__(self, fn):


        self.fn = fn


    def flat_map(self, s):


        for x in s:


            for r in self.fn(x):


                yield r
```

(b) (2 pt) Define `cycle` that returns a `Stream` repeating the digits 1, 3, 0, 2, and 4. **Hint**: `(3+2)%5` equals 0.

```
def cycle(start=1):
    """Return a stream repeating 1, 3, 0, 2, 4 forever.

    >>> first_k(cycle(), 11) # Return the first 11 elements as a list
    [1, 3, 0, 2, 4, 1, 3, 0, 2, 4, 1]
    """


    def compute_rest():


        return cycle((start+2) % 5)


    return Stream(start, compute_rest)
```

(c) (4 pt) Implement the Scheme procedure `directions`, which takes a number `n` and a symbol `sym` that is bound to a nested list of numbers. It returns a Scheme expression that evaluates to `n` by repeatedly applying `car` and `cdr` to the nested list. Assume that `n` appears exactly once in the nested list bound to `sym`.

Hint: The implementation searches for the number `n` in the nested list `s` that is bound to `sym`. The returned expression is built during the search. See the tests at the bottom of the page for usage examples.

```
(define (directions n sym)

    (define (search s exp)

        ; Search an expression s for n and return an expression based on exp.

        (cond ((number? s) (if (= s n) exp nil))

              ((null? s) nil)

              (else (search-list s exp))))

    (define (search-list s exp)

        ; Search a nested list s for n and return an expression based on exp.

        (let ((first (search (car s) (list 'car exp)))

              (rest  (search (cdr s) (list 'cdr exp))))

             (if (null? first) rest first)))

    (search (eval sym) sym))

(define a '(1 (2 3) ((4))))
(directions 1 'a)
; expect (car a)
(directions 2 'a)
; expect (car (car (cdr a)))
(define b '((3 4) 5))
(directions 4 'b)
; expect (car (cdr (car b)))
```

(d) (2 pt) What expression will `(directions 4 'a)` evaluate to?

(car (car (car (cdr (cdr a)))))

5. (18 points) Basis Loaded

Ben Bitdiddle notices that any positive integer can be expressed as a sum of powers of 2. Some examples:

$$\begin{aligned} 11 &= 8 + 2 + 1 \\ 23 &= 16 + 4 + 2 + 1 \\ 24 &= 16 + 8 \\ 45 &= 32 + 8 + 4 + 1 \\ 2014 &= 1024 + 512 + 256 + 128 + 64 + 16 + 8 + 4 + 2 \end{aligned}$$

A `basis` is a linked list of decreasing integers (such as powers of 2) with the property that any positive integer `n` can be expressed as the sum of elements in the `basis`, starting with the largest element that is less than or equal to `n`.

(a) (4 pt) Implement `sum_to`, which takes a positive integer `n` and a linked list of decreasing integers `basis`. It returns a linked list of elements of the `basis` that sum to `n`, starting with the largest element of `basis` that is less than or equal to `n`. If no such sum exists, raise an `ArithmeticError`. **Each number in `basis` can only be used once (or not at all).** The `Link` class is described on your Midterm 2 Study Guide.

```
def sum_to(n, basis):
    """Return a sublist of linked list basis that sums to n.

    >>> twos = Link(32, Link(16, Link(8, Link(4, Link(2, Link(1))))))
    >>> sum_to(11, twos)
    Link(8, Link(2, Link(1)))
    >>> sum_to(24, twos)
    Link(16, Link(8))
    >>> sum_to(45, twos)
    Link(32, Link(8, Link(4, Link(1))))
    >>> sum_to(32, twos)
    Link(32)
    """


    if n == 0:


        return Link.empty


    elif basis == Link.empty:


        raise ArithmeticError


    elif basis.first > n:


        return sum_to(n, basis.rest)


    else:


        return Link(basis.first, sum_to(n-basis.first, basis.rest))
```

(b) (6 pt) Cross out as many lines as possible in the implementation of the `FibLink` class so that all doctests pass. A `FibLink` is a subclass of `Link` that contains decreasing Fibonacci numbers. The `up_to` method returns a `FibLink` instance whose first element is the largest Fibonacci number that is less than or equal to positive integer `n`.

```
class FibLink(Link):
    """Linked list of Fibonacci numbers.

    >>> ten = FibLink(2, FibLink(1)).up_to(10)
    >>> ten
    Link(8, Link(5, Link(3, Link(2, Link(1)))))
    >>> ten.up_to(1)
    Link(1)
    >>> six, thirteen = ten.up_to(6), ten.up_to(13)
    >>> six
    Link(5, Link(3, Link(2, Link(1))))
    >>> thirteen
    Link(13, Link(8, Link(5, Link(3, Link(2, Link(1))))))
    """
    @property
    def successor(self):
        return self.first + self.rest.first

    def up_to(self, n):
        if self.first == n:
            return self
        elif self.first > n:
            return self.rest.up_to(n)
        elif self.successor > n:
            return self
        else:
            return FibLink(self.successor, self).up_to(n)
```

(c) (2 pt) Circle the Θ expression below that describes the number of calls made to `FibLink.up_to` when evaluating `FibLink(2, FibLink(1)).up_to(n)`. The constant ϕ is $\frac{1+\sqrt{5}}{2} = 1.618...$

$\Theta(1)$ $\quad$ $\Theta(\log_\phi n)$ $\quad$ $\Theta(n)$ $\quad$ $\Theta(n^2)$ $\quad$ $\Theta(\phi^n)$

(d) (2 pt) Alyssa P. Hacker remarks that Fibonacci numbers also form a basis. How many **total** calls to `FibLink.up_to` will be made while evaluating **all** the doctests of the `fib_basis` function below? Assume that `sum_to` and `FibLink` are implemented correctly. Write your answer in the box.

```
def fib_basis():
    """Fibonacci basis with caching.

    >>> r = fib_basis()
    >>> r(11)
    Link(8, Link(3))
    >>> r(23)
    Link(21, Link(2))
    >>> r(24)
    Link(21, Link(3))
    >>> r(45)
    Link(34, Link(8, Link(3)))
    """
    fibs = FibLink(2, FibLink(1))
    def represent(n):
        nonlocal fibs
        fibs = fibs.up_to(n)
        return sum_to(n, fibs)
    return represent
```

10

(e) (4 pt) Implement `fib_sums`, a function that takes positive integer `n` and returns the number of ways that `n` can be expressed as a sum of unique Fibonacci numbers. Assume that `FibLink` is implemented correctly.

```
def fib_sums(n):
    """The number of ways n can be expressed as a sum of unique Fibonacci numbers.

    >>> fib_sums(9) # 8+1, 5+3+1
    2
    >>> fib_sums(12) # 8+3+1
    1
    >>> fib_sums(13) # 13, 8+5, 8+3+2
    3
    """
    def sums(n, fibs):

        if n == 0:

            return 1

        elif fibs == Link.empty or n < 0:

            return 0

        a = sums(n-fibs.first, fibs.rest)

        b = sums(n, fibs.rest)

        return a + b

    return sums(n, FibLink(2, FibLink(1)).up_to(n))
```

6. (8 points) Sequels

Assume that the following table of movie ratings has been created.

```
create table ratings as
  select "The Matrix" as title,    9 as rating union
  select "The Matrix Reloaded",    7           union
  select "The Matrix Revolutions", 5           union
  select "Toy Story",              8           union
  select "Toy Story 2",            8           union
  select "Toy Story 3",            9           union
  select "Terminator",             8           union
  select "Judgment Day",           9           union
  select "Rise of the Machines",   5;
```

Correct output
Judgment Day
Terminator
The Matrix
Toy Story
Toy Story 2
Toy Story 3

The correct output table for both questions below happens to be the same. It appears above to the right for your reference. **Do not hard code your solution to work only with this table!** Your implementations should work correctly even if the contents of the `ratings` table were to change.

(a) (2 pt) Select the titles of all movies that have a rating greater than 7 in alphabetical order.

```
select title from ratings where rating > 7 order by title;
```

(b) (6 pt) Select the titles of all movies for which at least 2 other movies have the same rating. The results should appear in alphabetical order. Repeated results are acceptable. *You may only use the SQL features introduced in this course.*

```
with
  groups(name, score, n) as (
    select title, rating, 1 from ratings union
    select title, score, n+1 from groups, ratings
      where title > name and rating=score
  )
select title from groups, ratings
  where rating=score and n > 2 order by title;
```