CS 61A Fall 2014

Structure and Interpretation of Computer Programs

MIDTERM 1 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 official 61A midterm 1 study guide 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
/12	/14	/8	/6	/40

1. (12 points) World Cup

(a) (10 pt) For each of the expressions in the tables below, write the output displayed by the interactive Python interpreter when the expression is evaluated. The output may have multiple lines.

Whenever the interpreter would report an error, write Error. You *should* include any lines displayed before an error.

Reminder: the interactive interpreter displays the value of a successfully evaluated expression, unless it is `None`.

The first three rows have been provided as examples.

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

```
def square(x):
    return x * x

def argentina(n):
    print(n)
    if n > 0:
        return lambda k: k(n+1)
    else:
        return 1 / n

def germany(n):
    if n > 1:
        print('hallo')
    if argentina(n-2) >= 0:
        print('bye')
    return argentina(n+2)
```

Expression	Interactive Output
`5*5`	25
`print(5)`	5
`1/0`	Error
`print(1, print(2))`	2 1 None
`argentina(0)`	0 Error

Expression	Interactive Output
`argentina(1)(square)`	1 4
`germany(1)(square)`	-1 3 16
`germany(2)(germany)`	hallo 0 Error

(b) (2 pt) Fill in the blank with an expression so that the whole expression below evaluates to a number.
Hint: The expression `abs > 0` causes a `TypeError`.

```
(lambda t: argentina(t)(germany)(square))( 0.5 )
```

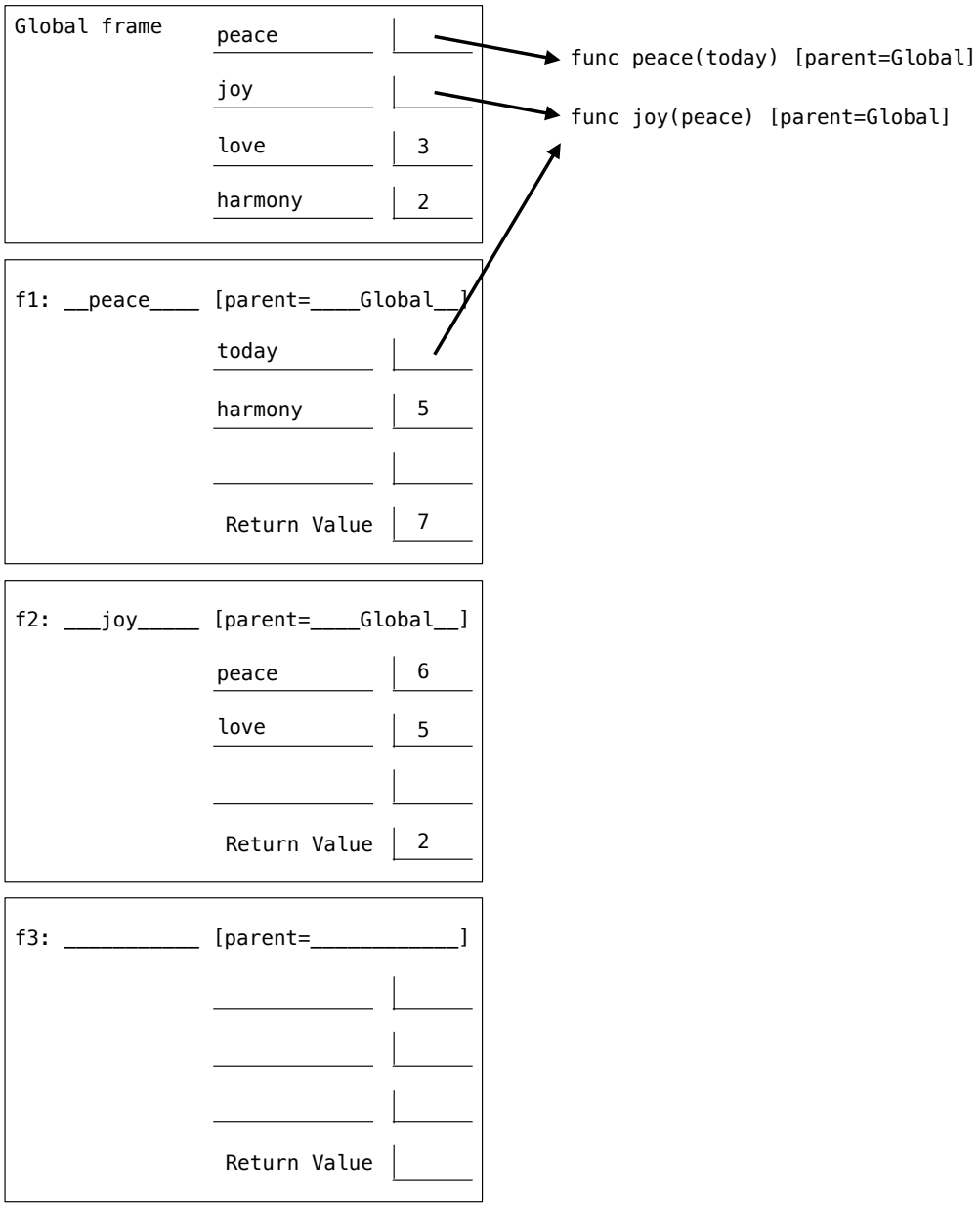
2. (14 points) Envy, Iron, Mint

(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 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.

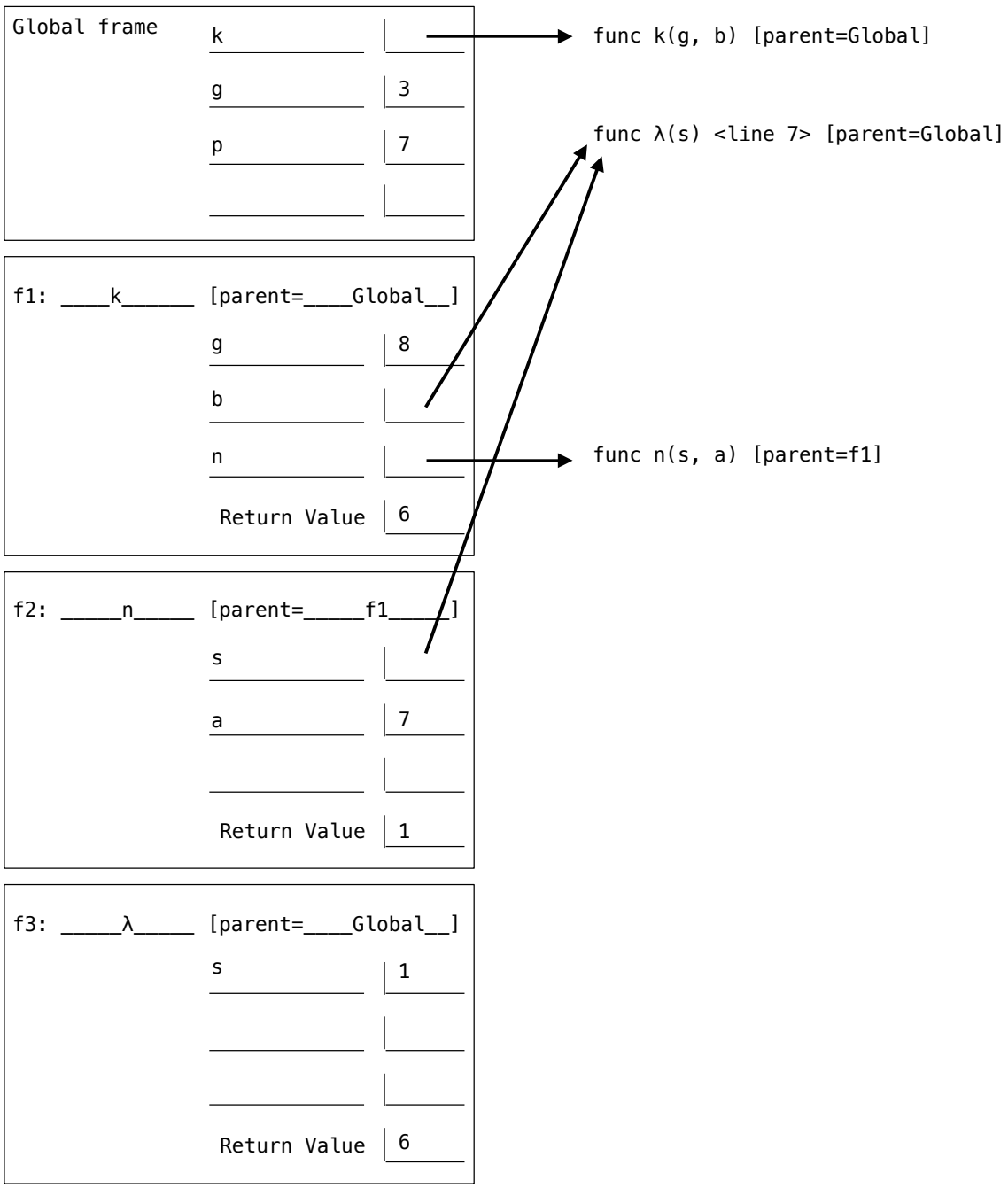
```
1  def peace(today):
2      harmony = love+2
3      return harmony + today(love+1)
4
5  def joy(peace):
6      peace, love = peace+2, peace+1
7      return love // harmony
8
9  love, harmony = 3, 2
10 peace(joy)
```



(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 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.

```
1 def k(g, b):
2     def n(s, a):
3         return g-p
4     return b(n(b, p))
5
6 g, p = 3, 7
7 k(p+1, lambda s: g+3)
```



3. (8 points) Express Yourself

(a) (3 pt) A k-bonacci sequence starts with K-1 zeros and then a one. Each subsequent element is the sum of the previous K elements. The 2-bonacci sequence is the standard Fibonacci sequence. The 3-bonacci and 4-bonacci sequences each start with the following ten elements:

```
             n: 0, 1, 2, 3, 4, 5, 6,  7,  8,  9, ...

kbonacci(n, 2): 0, 1, 1, 2, 3, 5, 8, 13, 21, 35, ...

kbonacci(n, 3): 0, 0, 1, 1, 2, 4, 7, 13, 24, 44, ...

kbonacci(n, 4): 0, 0, 0, 1, 1, 2, 4,  8, 15, 29, ...
```

Fill in the blanks of the implementation of `kbonacci` below, a function that takes non-negative integer `n` and positive integer `k` and returns element n of a k-bonacci sequence.

```
def kbonacci(n, k):
    """Return element N of a K-bonacci sequence.

    >>> kbonacci(1, 4)
    0
    >>> kbonacci(3, 4)
    1
    >>> kbonacci(9, 4)
    29
    >>> kbonacci(4, 2)
    3
    >>> kbonacci(8, 2)
    21
    """

    if n < k - 1:

        return 0

    elif n == k - 1:

        return 1

    else:

        total = 0


        i = n-k

        while i < n:


            total = total + kbonacci(i, k)

            i = i + 1

        return total
```

(b) (5 pt) Fill in the blanks of the following functions defined together in the same file. **Assume that all arguments to all of these functions are positive integers that do not contain any zero digits**. For example, 1001 contains zero digits (not allowed), but 1221 does not (allowed). You may assume that `reverse` is correct when implementing `remove`.

```
def combine(left, right):
    """Return all of LEFT's digits followed by all of RIGHT's digits."""
    factor = 1
    while factor <= right:
        factor = factor * 10
    return left * factor + right

def reverse(n):
    """Return the digits of N in reverse.

    >>> reverse(122543)
    345221
    """

    if n < 10:

        return n

    else:


        return combine(n%10, reverse(n//10))


def remove(n, digit):
    """Return a number that consists of all digits of N that are not DIGIT.
    Assume that DIGIT is a positive integer less than 10.

    >>> remove(243132, 3)
    2412
    >>> remove(243132, 2)
    4313
    >>> remove(remove(243132, 1), 2)
    433
    """

    removed = 0

    while n:


        n, last = n // 10, n % 10


        if last != digit:


            removed = removed * 10 + last

    return reverse(removed)
```

4. (6 points) Lambda at Last

(a) (2 pt) Fill in the blank below with an expression so that the second line evaluates to 2014. **You may only use the names `two_thousand`, `two`, `k`, `four`, and `teen` and parentheses in your expression (no numbers, operators, etc.)**.

```
two_thousand = lambda two: lambda k: k(two)(two)
two_thousand(7)(lambda four: lambda teen: 2000 + four + teen)
```

(b) (4 pt) The `if_fn` returns a two-argument function that can be used to select among alternatives, similar to an `if` statement. Fill in the return expression of `factorial` so that it is defined correctly for non-negative arguments. **You may only use the names `if_fn`, `condition`, `a`, `b`, `n`, `factorial`, `base`, and `recursive` and parentheses in your expression (no numbers, operators, etc.)**.

```
def if_fn(condition):
    if condition:
        return lambda a, b: a
    else:
        return lambda a, b: b

def factorial(n):
    """Compute n! = 1 * 2 * 3 * ... * n.

    >>> factorial(3)
    6
    >>> factorial(5)
    120
    """
    def base():
        return 1
    def recursive():
        return n * factorial(n-1)



    return if_fn(n)(recursive, base)()
```