fullName: _____ andrewID: _____ section: ____

15-112 S26 Quiz9 Version B
Time: 25 Minutes

You must write your name on this paper and hand this back in immediately after the assessment. If we do not receive it immediately, you will receive a zero on the assessment. Do not unstaple any pages. All pages must be handed in intact.

Do not use your own scrap paper. You should not need it, but if you must absolutely have scrap paper, raise your hand and we will provide some. Write your andrewID clearly on it and hand it in with your quiz. We will not grade anything on scrap paper.

You may not ask questions during the quiz, except for English-language clarification questions. If you are unsure about a problem, take your best guess.

Before and during the quiz, you may not view any other notes, prior work, websites or resources, including any form of AI. You may not use calculators, phones, laptops, or any other devices. You may not communicate with anyone else except for current 112 TAs or faculty during the assessment. All syllabus policies apply.

You may not discuss this test with anyone else, even briefly, in any form, until we have released grades. Failure to abide by these rules may result in an academic integrity violation.

Do not use material we have not yet covered. Thus, do not use OOP.

Do not open this or look inside (even briefly) before you are ready to begin. Do not spend more than the specified time noted above on this assessment.

Code Tracing [30 pts, 7.5 pts each]

For each of the following, indicate what the code prints. Place your answer (and nothing else) in the box below each block of code. If a line of code crashes, just print "crash" (without quotes) and stop the CT at that point.

CT1: [7.5 pts]

```
def ct1(n):
    if n == 0:
        return 0
    elif n % 2 == 0:
        return 2 * ct1(n-1)
    else:
        return n + ct1(n-1)

print(ct1(5))
```

CT2: [7.5 pts]

```
def ct2(n):
    if n == 0:
        return [ ]
    else:
        return [n] + ct2(n//2) + [n]

print(ct2(3))
```

CT3: [7.5 pts]

```
def ct3(L, d):
    if len(L) < 2:
        return L
    else:
        i = len(L)//2
        left = L[:i]
        right = L[i:]
        if sum(left) > sum(right):
            return [ct3(left, d+1) + ['L', d]]
        else:
            return ['R', d] + ct3(right,d+1)

print(ct3([3,5,7,2], 0))
```

CT4: [7.5 pts]

```
from cmu_cpcs_utils import *

def ct4(T, d, k):
    if T.isLeaf():
        return (T.value, d)
    else:
        return ct4(T.children[k], d+1, k)

T = Tree('A',
         Tree('B',
              Tree('C'),
              Tree('D')),
         Tree('E',
              Tree('F'),
              Tree('G')))

print(ct4(T, 0, 0))
```

Fill in the Blank [15 pts, 5 pts each]

FitB #1: Fill in the blank for this code from Towers of Hanoi in the course notes:

```
def move(n, source, target, temp):
    if n == 1:
        return [(source, target)]
    else:
        return (move(n-1, source, temp,   target) +

                ____________________________________________________ +

                move(n-1, temp,   target, source))
```

FitB #2: Fill in the blank for this code from Merge Sort in the course notes:
You can assume any required helper functions are written for you.

```
def mergeSort(L):
    if len(L) < 2:
        return L
    else:
        i = len(L)//2
        sortedLeftHalf = mergeSort(L[:i])
        sortedRightHalf = mergeSort(L[i:])

        return ____________________________________________________
```

FitB #3: Fill in the blank in h(n) so that f(n) == g(n) for all integer values of n:

```
def f(n):
    # hint: f(n) is a familiar function from the course notes
    if n < 2:
        return False
    for factor in range(2, n):
        if (n % factor == 0):
            return False
    return True

def g(n):
    if n <= 1:
        return False
    else:
        return h(n, 2)

def h(n, f):
    if f == n:
        return True
    elif n % f == 0:
        return False
    else:
        return _____________________________________________________
```

FR1: alternatesParity(L) [25 pts]

Note: to receive any credit for this problem, you must not use iteration (for or while loops). Just use recursion.

Background: the "parity" of an int is 0 if it is even and 1 if it is odd.

With that, write the recursive non-mutating function alternatesParity(L) that takes a list L of ints and returns True if the values in L alternate parity (so each value has a different parity than the previous value), and False otherwise. We will say that the empty list does alternate parity, as does any list with just one int in it.

Thus:

```
L = [1, 2, 1, 4, 3]
assert(alternatesParity(L) == True)

L = [1, 2, 4]
assert(alternatesParity(L) == False) # 4 follows 2

L = [ ]
assert(alternatesParity(L) == True) # empty list alternates parity

L = [2, 1, 4]
assert(alternatesParity(L) == True)
assert(L == [2, 1, 4]) # non-mutating
```

Write your solution on the next page:

FR2: makeChildCountsEven (T) [30 pts]

Background: given a tree T and a tree C, we can call T.addChild(C) to mutatingly add C as a child of T. For example:

```
T = Tree(1)
C = Tree(2)
T.addChild(C)
assert(T == Tree(1,
                 Tree(2)))
```

With this in mind, write the function makeChildCountsEven(T) that takes a tree T and mutates T by finding each node N in T where N has an odd number of children and adds one more child to N with the value 0. As with most mutating functions, this should return None. For example:

```
T = Tree(1,
         Tree(2),
         Tree(3,
              Tree(4)),
         Tree(5,
              Tree(6,
                   Tree(7)),
              Tree(8)))
```

The nodes with an odd number of children are:

- 1 (with 3 children)
- 3 (with 1 child)
- 6 (with 1 child)

Thus:

```
assert(makeChildCountsEven(T) == None)
assert(T == Tree(1,
                 Tree(2),
                 Tree(3,
                      Tree(4),
                      Tree(0)),      # <-- Since 3 had 1 child
                 Tree(5,
                      Tree(6,
                           Tree(7),
                           Tree(0)), # <-- Since 6 had 1 child
                      Tree(8))),
                 Tree(0))            # <-- Since 1 had 3 children
```

Write your solution here:

BonusCT: These CTs are optional, and intended to be very challenging. They are worth very few points. Indicate what the following code prints. Place your answers (and nothing else) in the boxes below each code block. If a line of code crashes, just print "crash" (without quotes) and stop the CT at that point.

bonusCT1: [1pt]

```
def bonusCt1(n):
    def f(n):
        return n - f(n-1) if n else 0
    def g(n):
        return f(n)//g(n-1) if n>1 else 1
    def h(n):
        return h(n-1) + [g(n)] if n > 2 else [ ]
    return h(n)

print(bonusCt1(8))
```

bonusCT2: [1pt]

```
def bonusCt2(L):
    def g(L, i):
        if i > 0:
            L[0] += L[i]
            g(L, i-1)
    def f(L):
        if len(L) == 1:
            return L[0]
        else:
            g(L, len(L)-1)
            L.pop()
            return f(L)
    return f(L)

print(bonusCt2([1,3,2,4]))
```