fullName: _____ andrewID: _____ section: ____

15-112 F25 Quiz6 Version A
Time: 35 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 quiz 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 OOP.

Do not open this or look inside (even briefly) before you are ready to begin. Do not spend more than 35 minutes on this assessment.

Code Tracing [15 pts total, 5 pts each]

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

CT1:

```
def ct1(n):
    if n < 10: # note the unusual condition
        return str(n**2)
    else:
        return f'{n}.' + ct1(n//len(str(n)))

print(ct1(100))
```

CT2:

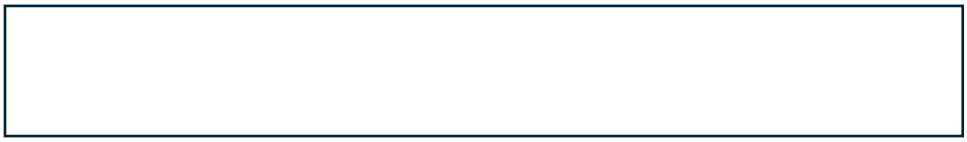
```
def ct2(L):
    if L == [ ]:
        return L
    else:
        x = min(L[0], L[-1])
        rest = L[1:-1]
        return ct2(rest) + [x]

print(ct2([1,3,5,6,2,4]))
```

CT3:

```
def ct3(L, k):
    if L == [ ]:
        return L
    else:
        i = len(L)//2
        M = L[:i]
        N = L[i+1:]
        if k%2 == 0:
            M, N = N, M
        return ct3(N, k+1) + [(L[i],k)] + ct3(M, k+1)

print(ct3([1,2,3,4,5], 0))
```



Fill in the Blank (FitB) [10 pts total, 2 pts each]

Fill in the blanks with the missing code for the following examples taken from the course notes.

FitB1:

```
def gcd(x, y):
    # Euclid's (very fast) algorithm:
    if y == 0:
        return x
    else:
        return gcd(_____________________________________________________)
```

FitB2:

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

def solveTowersOfHanoi(n):
    return move(n, 0, 2, 1)
```

FitB3:

```
# assume partition(L) is written for you:

def quickSort(L):
    if len(L) < 2:
        return L
    else:
        smaller, pivot, larger = partition(L)

        return ____________________________________________________
```

FitB4:

```
def floodFillHelper(board, row, col, oldValue, newValue):
    rows, cols = len(board), len(board[0])
    if ((row < 0) or (row >= rows) or
        (col < 0) or (col >= cols) or

        (_____________________________________________________________)):
        return
    else:
        board[row][col] = newValue
        floodFillHelper(board, row-1, col, oldValue, newValue) # up
        floodFillHelper(board, row+1, col, oldValue, newValue) # down
        floodFillHelper(board, row, col-1, oldValue, newValue) # left
        floodFillHelper(board, row, col+1, oldValue, newValue) # right
```

FitB5:

```
# assume merge(L, M) is 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 ____________________________________________________
```

FR: pruneEvens(tree) [25 pts]

For this problem, you must use recursion meaningfully. However, you may also use for or while loops if you wish. With that, write the function `pruneEvens(tree)` that takes a tree where the values are all integers, and mutatingly changes the tree so that all the subtrees with an even value are replaced with a leaf node with the value `'pruned'` (a string).

While most mutating functions return None, this function returns a set of all the even integers that were changed to 'pruned'. For example:

```
"""
          ┌─── 2 ──────── 3
─── 1 ────┤
          └─── 5 ──────── 4  ──────── 6
"""
t1 = Tree(1,
          Tree(2,
               Tree(3)
              ),
          Tree(5,
               Tree(4,
                    Tree(6)
                   ),
              ),
         )

"""
          ┌─── pruned
─── 1 ────┤
          └─── 5 ──────── pruned
"""
t2 = Tree(1,
          Tree('pruned'),
          Tree(5,
               Tree('pruned')
              ),
         )
assert(pruneEvens(t1) == {2, 4}) # Note: 6 is not in this set
assert(t1 == t2)                 # Note: t1 was mutated
```

Hint: For any tree or subtree t, you can mutate or reassign `t.value` and/or `t.children` as needed (just make sure that `t.children` is always a list).

Also: for partial credit, you can skip mutating the tree and just return a set of all the even integers that must be changed to `'pruned'`.

Write your solution here:

FR: getStateCounts(L) [25 pts]

For this problem, you must use recursion, and you must not use for or while loops. Also, your solution must not mutate L.

Background: This problem concerns a list L where each element in L is itself a list of the form `[city, listOfStates]`. For example:

```
L = [
     ['Avon', ['IL']],
     ['Bath', ['IL', 'NC', 'SC']],
     ['Cuba', ['KS']],
     ['Dale', ['IL', 'SC']]
    ]
```

Here we see that the city 'Bath' is in 3 states ('IL', 'NC', and 'SC').

With that, without using while or for loops, write the function `getStateCounts(L)` that takes a list L as described and returns a dictionary mapping each state that occurs anywhere in L to the count of the number of cities that are in that state.

In the example above, we get:

```
assert(getStateCounts(L) == {'IL': 3, 'NC': 1, 'SC': 2, 'KS': 1})
```

Hints:

1. We strongly suggest that you first write the helper function getStates(L) that takes a list L as described and returns the combined list of all the states in L (including duplicates). In the example above, getStates(L) returns:
   ```
   ['IL', 'IL', 'NC', 'SC', 'KS', 'IL', 'SC']
   ```
   See how 'IL' occurs 3 times in this list.

2. Next, use the result of `getStates(L)` to help you compute the state counts dictionary. We found it helpful to do this in another helper function.

Write your solution on the following pages.

FR: **fasterFoo [25 pts]**

The following function foo(L) runs in O(N**2). Write the function fasterFoo(L) that is equivalent to foo(L) but that runs in O(N).

```
def foo(L):
    result = [ ]
    for v in L:
        if L.count(v) == 1:
            result.append(v)
    return set(result)
```

Write your solution here:

BonusCT [2 pts total, 1 pt each]

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

```
def bonusCt1(a, b, c):
    def f(n): return n if n < 1 else f(n-1) + 2*n - 1
    def g(n, k): return n if f(n) > 10**k else g(2*n, k)
    return g(g(a, b) + g(b, a) + c, 4) + c
print(bonusCt1(3, 2, 1))
```

```
def bonusCt2(n):
    def z(n, k=ord('a')-1):
        if chr(n+k).islower(): return chr(n+k) + (z(n//2) or '')
    def y(n):
        return list(z(n) + z(n-1))
    def x(A, n=None):
        n = n or len(A)
        if n <= 1: return A
        for i in range(n-1):
            if A[i] < A[i+1]: A[i], A[i+1] = A[i+1], A[i]
        return x(A, n-1)
    return ''.join(x(y(n)))
print(bonusCt2(7))
```