Name:________________________ andrewID:______________ Recitation:___

15-112 F24
Midterm2 FR Writeups

Read these instructions carefully before starting:

1. This exam has two handouts to submit:
    - The main exam where you write your solutions.
    - An additional handout with the FR writeups, which you also must submit!

2. This is the FR Writeups handout. See the Main Exam for more instructions.

3. Be sure to write your name, andrewId, and recitation on this handout.

4. We will not grade anything written on this handout, but you may use this handout to write scratch work

5. Be sure to submit this handout at the end of the exam, along with your Main Exam.

6. Good luck!

Free Response / FR1: getColorMap [15 pts]

Background: Two friends drove through a few states on a recent road trip. For each state, they recorded the number of Mountain Dews they drank in that state, organized by the different colors, like so:

```
stateMap = {
      'PA' : { 'red' : 3,
               'green' : 2 },
      'OH' : { 'red' : 1,
               'blue' : 2 },
      'NY' : { 'green' : 4 }
     }
```

For example, the friends drank 3 red Mountain Dews in PA.

Now, we can instead arrange this same data by mapping each color to a string listing the total number for that color and a comma-separated, alphabetically sorted list of the states for that color, like so:

```
colorMap = {
    'red' :   '4 in OH,PA',
    'green' : '6 in NY,PA',
    'blue' :  '2 in OH'
}
```

With this in mind, write the function getColorMap(stateMap) that takes a stateMap and returns the corresponding colorMap.

You are assured that:

- Every dict in the stateMap is non-empty,
- Every state name is 2 uppercase letters, and
- All the counts are positive.

Write your answer to FR1 in the Main Exam and not here.

Free Response / FR2: nthPerfect [15 pts]

Note: for this problem, you must not use iteration (for or while loops).

For full credit, you also may not create any lists.

Background: A positive integer n is "perfect" if n equals the sum of all its "proper" factors -- that is, the positive factors of n that are less than n.

For example, the proper factors of 6 are 1, 2, 3. Since 1+2+3 == 6, 6 is perfect.

The first several perfect numbers are: 6, 28, 496.

With that in mind, and not using iteration, write the function nthPerfect(n) that takes a non-negative integer n and returns the nth perfect number, where nthPerfect(0) returns 6.

Here are some test cases:

```
    assert(nthPerfect(0) == 6)
    assert(nthPerfect(1) == 28)
    assert(nthPerfect(2) == 496)
```

You will want to write some helper functions. These also must not use iteration (while or for loops), and for full credit must not create any lists.

Write your answer to FR2 in the Main Exam and not here.

Free Response / FR3: makeFactorish [15 pts]

Note: for this problem you must use backtracking properly to receive credit (even if there are other ways to solve the problem).

You must use recursion here, but you are also free to use iteration if it helps.

Background: given a list L of non-zero integers, we will say that L is "factorish" (a coined term) if L is non-empty and for each value k from 1 to len(L) (inclusive), k is a factor of the sum of the first k values in L.

For example, say L = [1, 3, 2]:

```
    k    first k values   sum of first k values
    1       [1]                   1
    2       [1, 3]                4
    3       [1, 3, 2]             6
```

We see that 1 is a factor of 1, 2 is a factor of 4, and 3 is a factor of 6 so this list is factorish.

As another example, say L = [1, 2, 3]:

```
    k    first k values   sum of first k values
    1       [1]                   1
    2       [1, 2]                3
    3       [1, 2, 3]             6
```

Here, we see that 2 is NOT a factor of 3, so this list is not factorish.

With this in mind, using backtracking properly, write the function makeFactorish(L) that takes a possibly-empty list L of non-zero integers, and returns an ordering of the list L that is factorish, or None if no such order exists.

Be sure that you do not mutate L.

Also, note that there are several legal results for most lists, so our test cases check if you returned any one of them.

For example, if L = [1, 2, 3], the possible orders of L that are factorish are [3, 1, 2] and [1, 3, 2], so:

```
    assert(makeFactorish([1, 2, 3]) in [[1, 3, 2],
                                         [3, 1, 2]])
```

Here are some other test cases for you:

```
    assert(makeFactorish([1, 2, 3, 2]) in [[1, 3, 2, 2],
                                            [1, 3, 2, 2],
                                            [3, 1, 2, 2],
                                            [3, 1, 2, 2]])

    assert(makeFactorish([42]) in [[42]])

    assert(makeFactorish([ ]) == None)

    assert(makeFactorish([1, 1, 2]) == None)
```

Also, note this test case, which assures makeFactorish is non-mutating:

```
    L = [1, 2, 3]
    assert(makeFactorish(L) in [[1, 3, 2], [3, 1, 2]])
    assert(L == [1, 2, 3]) # do not mutate L!
```

Write your answer to FR3 in the Main Exam and not here.

Free Response / FR4: File and Folder Classes [15 pts]

Write the classes File and Folder so the following test code passes.
Do not hardcode any test cases. Your solution must work in general.
However, you only have to implement the methods required to make the test code pass.

```
file1 = File('a.txt', 100) # 100 is the size of the file
assert(str([file1]) == '[File(name=a.txt, size=100)]')
assert(file1.getSize() == 100)

file2 = File('b.pdf', 200)
assert(file2 == File('b.pdf', 200))
assert(file2 != File('b.pdf', 300))
assert(file2 != 'do not crash here')

file3 = File('c.doc', 500)

folder1 = Folder('A')
assert(folder1.files == [ ])
assert(folder1.folders == [ ])
# the size of a folder is the sum of the sizes of
# all the files it contains, including the files
# in any folders it contains (or any folders they contain).
assert(str([folder1]) == '[Folder(name=A, size=0)]')
assert(folder1.getSize() == 0)

folder1.addFile(file1)
assert(folder1.files == [file1])
assert(str([folder1]) == '[Folder(name=A, size=100)]')
assert(folder1.getSize() == 100)

folder1.addFile(file2)
assert(folder1.files == [file1, file2])
assert(str([folder1]) == '[Folder(name=A, size=300)]')
assert(folder1.getSize() == 300)
```

```
folder2 = Folder('B')
folder2.addFile(file3)
assert(str([folder2]) == '[Folder(name=B, size=500)]')
assert(folder2.getSize() == 500)

folder1.addFolder(folder2)
assert(folder1.folders == [folder2])
assert(str([folder1]) == '[Folder(name=A, size=800)]')

folder1.removeFolderByName('B')
assert(folder1.folders == [ ])
assert(str([folder1]) == '[Folder(name=A, size=300)]')
assert(str([folder2]) == '[Folder(name=B, size=500)]')

folder1.moveFile(file1, folder2)
assert(str([folder1]) == '[Folder(name=A, size=200)]')
assert(str([folder2]) == '[Folder(name=B, size=600)]')
```

Write your answer to FR4 in the Main Exam and not here.

This page is intentionally blank.