CS:1210 Exam 2

April 3rd, 6:30 pm to 8:30 pm

Instructions:

- This is an open notes exam and you have 2 hours to complete it. There are 4 problems in the exam and these appear on 9 pages. The exam is worth 100 points (10% of your grade); the first two problems are worth 20 points each and the last two are worth 30 points each.
- Make sure that you do not have any electronic devices (laptops, phones, calculators, dictionaries, etc.) on your desk; you are not allowed to access these during your exam.
- Write as neatly as you can.
- Show all your work, especially if you want to receive partial credit.

Name:

Circle your discussion section:

A01 (8:30-9:20, 221 MLH)	A02 (9:30-10:20 213 MLH)	A03 (11-11:50, 116 MH)
A04 (12:30-1:20, 66 SH)	A05 (3:30-4:20, 248 JH)	A06 (5-5:50, 346 JH)

1. [**20 points**] Suppose that `L` is the list

```
[["This", ["is", "a"]], "nested", ["list", "with"], ["few", [["dish"]]], ["nets", "of"], "nesting"].
```

For parts (a)-(f), write down the *value* of each of the given expressions. For parts (g)-(j) write down the value of `L` after the given statement has been executed. For each part, start with the list `L` given above.

(a) `[len(x) for x in L].count(2)`

(b) `"a".join(L[2][0].split("i") + L[2][1].split("i"))`

(c) `[x+y for x in range(4) for y in range(6) if y - x > 2]`

Recall that the list `L` is:

```
[["This", ["is", "a"]], "nested", ["list", "with"], ["few", [["dish"]]], ["nets", "of"], "nesting"].
```

(d) `[str(x)[::-1] for x in range(20, 25)]`

(e) `[y for x in L for y in x if type(x) == list and type(y) == str]`

(f) `[x.split("ste") for x in L if type(x) == str]`

(g) `L[1:6:2]=["a","b","c"]`

(h) `L[0::2] = list(range(10,40,10))`

(i) `L.remove(L[3])`

(j) For this part, you have to execute the following two lines of code. Recall that you have to write down the value of `L`.

```
LL = L
LL[:]=range(len(LL))
```

2. [**20 points**] You are given some code and asked to answer questions about this code.

(a) What output does this program produce? Here is the code for *binary search.* Notice the `print`-statement that I have added inside the `while`-loop and the `print` statement at the bottom of the function.

```
def binarySearch(L, k):
    left = 0
    right = len(L)-1

    while left <= right:
        mid = (left + right)//2 # index of the middle element
        print(L[left:mid], L[mid], L[mid+1:right+1])

        if L[mid] == k:
            return mid # element is found at mid, so return this index
        elif L[mid] < k: # look for element in right half
            left = mid + 1
        elif L[mid] > k: # look for element in the left half
            right = mid -1

    print(left, right)
    return -1 # element is not found in the list

# main program
result = binarySearch([3, 5, 7, 8, 10, 20, 30, 100, 102, 111], 63)
```

(b) What output does this program produce?

```
def foo(s):
    upperCaseLetters = [ch for ch in s if ch.isupper()]
    for ch in upperCaseLetters:
        s = s.replace(ch, "")
    return s

line = "The Bed is In the Top Floor, Next to The Red Box."
L = [x for x in [foo(s) for s in line.split()] if len(x) <= 2]

for e in L:
    print(e)
```

3. **[30 points]** In this problem, you are given two partially completed programs. Your task is to complete each program.

   (a) You are given a list of elements – some of these elements are strings and some are themselves lists containing strings. For example, the list you are given might be `["for", ["them", "object"], "graph", ["survey"]]`. The problem is to write a function called `computeLengths` that takes such a list as a parameter and returns a list of corresponding string lengths. For the list in the above example, `computeLengths` should return `[3, [4, 6], 5, [6]]`. Here is another example: `computeLengths([["ok", "two"], ["jail", "empty"]])` should return `[[2, 3], [4, 5]]`. Notice that the structure of the sublists is preserved by the returned list.

   There are two blanks in the code below that you need to fill, with one line of code in each.

```
def computeLengths(L):
    answer = []
    for x in L:
        if type(x) == str:
            # Append a length to answer


            ------------------------------------------------
        else:
            # x is a list of strings and so we need to append a
            # list of lengths to answer


            -------------------------------------------------------

    return answer
```

(b) Write a function called `removeNonPalindromes` that takes as parameter a list, say `L`, of strings and returns a new list with all non-palindromes in `L` removed. A *palindrome* is a string whose first half equals the reverse of its second half. Examples of palindromes are `"mom"`, `"civic"`, `"radar"`, `"anna"`, etc. Note that palindromes can be of even or odd length. Thus if `L` is `["madam", "mad", "anna", "civic", "license"]` then the function call `removeNonPalindromes(L)` should return `["madam", "anna", "civic"]`.

There are three blanks in the code below that you need to fill, the first two with assignment statements and the third with a boolean expression.

```
def removeNonPalindromes(L):
    answer = []
    for x in L:
        # if length of x is odd
        if len(x) % 2 == 1:
            # assign to y the second half of x, exluding the
            # middle character

            ------------------------------

        # else if the length of x is even
        else:
            # assign to y the second half of x

            ------------------------------

        # compare the first half of x with the reversed version of y
        # if length of x is odd, then the first half of x should
        # exclude the middle character

        if ___________________________________________:

            answer.append(x)

    return answer
```

4. **[30 points]** For each part of this problem, you are required to write some code.

   (a) A transportation company has 125 square feet of space in its truck to transport items of two types, Type A and Type B. Each item of Type A occupies 7 square feet and each item of Type B occupies 11 square feet of space. The company makes profit p_A for each item of Type A and profit p_B for each item of Type B. The company's goal is to find quantities of Type A and Type B items that will fit within its truck *and* maximize its profit. In other words, the company wants to find a quantity x_A of Type A items and a quantity x_B of Type B items such that (i) x_A and x_B are nonnegative integers, (ii) $7 \cdot x_A + 11 \cdot x_B \leq 125$ so that the space constraint of the truck is satisfied, and (iii) $p_A \cdot x_A + p_B \cdot x_B$ is maximized over all appropriate choices of x_A and x_B.

   Write a function `computeQuantities` that takes profits p_A and p_B as parameters and returns a size-2 list of the number of Type A items and the number of Type B items that the company should transport. For example, `computeQuantities(3.25, 4.25)` returns `[16, 1]` because it is possible to fit 16 Type A items and 1 Type B item in the truck (since $16 \times 7 + 1 \times 11 = 123 \leq 125$). Furthermore, this leads to a profit of \$56.25 (since $16 \times 3.25 + 1 \times 4.25 = 56.25$) and this is more than the profit that we could get from using any other combination of Type A and Type B quantities that would fit within the 125 square feet space.

   Note: My code for this problems is 5 lines long, including the function header. I first construct a list of all (x_A, y_A) pairs satisfying the space constraints and then construct a list of profits associated with these (x_A, y_A) pairs. Then I find the index of the maximum profit and return the (x_A, y_A) pair at this index.

(b) You are given an input file called `filenames.txt` that contains a bunch of file names of student homework submissions. The name of each student homework submission contains a time stamp and the student's name; here are the contents of an example input file:

```
4-10-15.17:30:25.John.Doe.hw1a.py
4-10-15.16:59:25.Ann.Best.hw1a.py
4-10-15.16:10:25.Ela.Gray.hw1a.py
3-3-15.22:10:25.Joe.Beck.hw1a.py
4-13-15.22:10:25.Bob.Cook.hw1a.py
4-10-15.12:10:25.Alice.Miller.hw1a.py
4-10-15.17:25:25.Ann.Best.hw1a.py
4-3-15.17:10:25.Joe.Beck.hw1a.py
```

The first line of the input file tells us that a student named "John Doe" submitted his solution to problem (a) in homework 1 on April 10th at time 17:30:25 (i.e., 5:30:25 pm). Notice that a student may submit homework solution several times.

Write a program that reads the input file and outputs the list of names of all students who have made at least one submission at or before the deadline of April 10th, 16:59:59. You can assume that you are given a boolean function `withinDeadline` that returns `True` if the time stamp of the student submission indicates that they met the deadline. The function, `withinDeadline` takes 6 integer parameters and has the following function header:

```
def withinDeadline(year, month, day, hour, minute, second):
```

While processing the first line in the input, we would make the function call

```
withinDeadline(15, 4, 10, 17, 30, 25)
```

to figure out that "John Doe" did not submit his homework in time. For the input file above, the output should be

```
['Ann Best', 'Ela Gray', 'Joe Beck', 'Alice Miller']
```

Note that even though "Joe Beck" had two submissions that met the deadline, his name appears only once in the output.

Note: My code for this problem is a 10-line program. Please use the next page to write your code neatly.

Space for your solution to Problem 4(b)