22C:16 CS:1210 Exam 2

April 5th, 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 8 pages.
- 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:

SCA (M, W evening)	A01 (9:30-10:20)	A02 (11-11:50)	A03 (12:30-1:20)
	A04 (2:00-2:50)	A06 (3:30-4:20)	

---

1. Evaluate each expression and write down its *value*. Assume that (i) `concat` is a function that takes two arguments `a` and `b` and returns `b+a`, (ii) `isLong` is a function that takes an argument `x` and returns `len(x) > 3`, and (iii) `L = [["This", ["is", "a"]], "nested", ["list", "with"], ["several", [["different"]]], ["levels", "of"], "nesting"]`.

   (a) `L[:5:2][-1]`

   (b) `[x for y in L for x in y][:5]`

   (c) `filter(isLong, L)`

For your convinience, here is the list `L` from the previous page: `L = [["This", ["is", "a"]], "nested", ["list", "with"], ["several", [["different"]]], ["levels", "of"], "nesting"]`

(d) `reduce(concat, map(len, L))`

(e) `[x.split("e") for x in L if len(x) > 3]`

(f) `L.count("nest")`

(g) `reduce(concat, filter(isLong, L))`

(h) `[i for i in range(len(L)) if "nest" in L[i]]`

(i) `L[3][1][0][0][1:3]`

(j) `[x[0] for x in L if type(x) == list]`

2. This problem has two parts. For each part, you are given some code and asked to figure out what output it produces.

   (a) Consider the following function:

```
def areConnected(w1, w2):
    words = [w1[:i+1] + w1[j:i:-1] + w1[j+1:]
             for i in range(len(w1))
             for j in range(i+1, len(w1))
             ]
    print words
    return w2 in words
```

   (i) What output do we get when we call this function as:

```
areConnected("help", "hlep")
```

   Note that you are not being asked what the function *returns*; you are being asked about what gets printed.

   (ii) What do the following function calls return?

   * `areConnected("writers", "rwiters")`

   * `areConnected("writers", "wrtiers")`

   * `areConnected("writers", "writes")`

   * `areConnected("writers", "sriterw")`

(b) Here is the code for *binary search*. Notice the `print`-statement that we 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 left, mid, right

        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
```

(i) What output does the following function call produce:

```
binarySearch([3, 5, 7, 8, 10, 20, 30, 100, 1000, 1001], 79)
```

(ii) What output does the following function call produce:

```
binarySearch([3, 5, 7, 8, 10, 20, 30, 100, 25, 30, 1000, 1001], 8)
```

(iii) What output does the following function call produce:

```
binarySearch([3, 5, 7, 8, 10, 20, 30, 100, 15, 30, 1000, 1001], 15)
```

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

   (a) Here is a partially completed program that reads a text file that looks like:

   ```
   Theodore Roosevelt 1858-1919
   Woodrow Wilson 1856-1924
   Herbert Hoover 1874-1964
   Richard Nixon 1913-1994
   ```

   Thus each line in the text file contains the name of a U.S. president followed by that president's life span. You may assume that each president's name consists of a first name and a last name separated by one or more blanks. The president's name is followed by one or more blanks and then the life span of the president is given in the format shown above. Your task is to write a program that reads a file such as this and outputs the list of presidents in decreasing order of how long they have lived. For the above shown input file, your program should produce the following output:

   ```
   Herbert Hoover
   Richard Nixon
   Woodrow Wilson
   Theodore Roosevelt
   ```

   Here is the partially complete program. It contains 4 blanks that you need to fill – three in the `while`-loop and one in the `print`-statement.

```
f = open("presidents.txt", "r")
line = f.readline().rstrip()
presidentList = []
while line:

    [firstName, lastName, lifeSpan] = ___________

    [birthYear, deathYear] = ____________

    presidentList.append(_________________________________________)
    line = f.readline()

presidentList.sort(reverse = True)

for president in presidentList:

    print ____________________

f.close()
```

(b) In this problem we provide a partially complete program that reads a 2-dimensional $n \times n$ *symmetric* matrix from a text file and stores this as a "nested" list (i.e., a list of lists). For example, the input file might look like the following:

```
3 4 1 7
3 2 1
2 4
1
```

This input file describes a 4×4 matrix. You would expect the input file to contain 16 numbers (4 per line), but it contains fewer numbers because the matrix is *symmetric*. Recall that in a symmetric matrix, the element in row i, column j is the same as the element in row j, column i. In the above example, the first line specifies the first row of the matrix. The second line, "`3 2 1`" specifies the elements in the second row of the matrix that occur in column 2, column 3, and column 4 respectively. There is no need to provide the element in row 2, column 1 because it is the same as the element in row 1, column 2, which we know to be 4. Similarly, the third line "`2 4`" specifies the elements in the third row of the matrix that occur in column 3 and column 4 respectively. Elements that occur in the third row in columns 1 and 2 need not be explicitly specified because they can be inferred by symmetry. Thus the matrix specified by the input file is:

$$\begin{bmatrix} 3 & 4 & 1 & 7 \\ 4 & 3 & 2 & 1 \\ 1 & 2 & 2 & 4 \\ 7 & 1 & 4 & 1 \end{bmatrix}$$

In the following program (shown in the next page), the matrix is constructed in two stages. In the first stage, the input file is read and zeroes are used for the missing numbers. For the input file shown above, the constructed matrix, after the first stage would be:

```
[[3, 4, 1, 7], [0, 3, 2, 1], [0, 0, 2, 4], [0, 0, 0, 1]]
```

In the second stage, the zeroes that appear in the beginning of each row are replaced by valid elements. The constructed matrix after the second stage would be:

```
[[3, 4, 1, 7], [4, 3, 2, 1], [1, 2, 2, 4], [7, 1, 4, 1]]
```

In the partially completed program, the first `for`-loop comprises the first stage described above and the "`print mat`" statement after this `for`-loop produces output:

```
[[3, 4, 1, 7], [0, 3, 2, 1], [0, 0, 2, 4], [0, 0, 0, 1]]
```

The second stage is implemented by the nested `for`-loops and the "`print mat`" statement at the bottom of the program produces output:

```
[[3, 4, 1, 7], [4, 3, 2, 1], [1, 2, 2, 4], [7, 1, 4, 1]]
```

Fill in the four blanks in this program.

```
f = open("matrix.txt", "r")

mat = []
zeroes = 0
for line in f:

   mat.append(____________________________________)

   zeroes = zeroes + 1

print mat

for i in ___________________________:

    for j in ___________________________:

        ________________________

print mat
f.close()
```

4. A sequence of numbers

$$x_1, x_2, \ldots, x_n$$

is called *bitonic* if there is an integer j, $1 \leq j \leq n$, such that $x_1 < x_2 < \cdots < x_j$ and $x_j > x_{j+1} > \cdots > x_n$. In other words, a bitonic sequence consists of a bunch of numbers at the beginning that are in strictly ascending order followed by a bunch of numbers in strictly descending order. For example, 1, 3, 7, 9, 15, 10, 2 is a bitonic sequence. Also, 3, 7, 8, 10, 20, 21 is a bitonic sequence.

Write a function called `findBitonicMax` that takes a list `L` of numbers that forms a bitonic sequence and returns an index of the maximum number in `L`. Of course, it is possible to find the maximum number by simply doing a left-to-right scan of the elements in `L`. A *much* more efficient way would be to solve this problem by using a "binary search" like algorithm and this is what you are required to implement. Specifically, you could examine the element in the middle of `L` along with its two neighbors on either side and based on what you see, you can determine if (i) the maximum is the middle element, (ii) the maximum is on the left side of the middle element, and (iii) the maximum is on the right side of the middle element.

Implement this "binary search" like algorithm using the following function header:

```
def findBitonicMax(L):
```