CS 115 Exam 3, Spring 2013

Your name: ______________________________________

Rules

- You may use one handwritten 8.5 x 11" cheat sheet (front and back). This is the only resource you may consult during this exam.
- Explain/show work if you want to receive partial credit for wrong answers.
- As long as your code is correct and obeys the specifications, you will get full credit. No points for style.
- When you write code, be sure that the indentation level of each statement is clear.

Grade (instructor use only)

	Your Score	**Max Score**
Problem 1		10
Problem 2		10
Problem 3		10
Problem 4		10
Problem 5		30
Problem 6		30
Total		100

Problem 1: Recursion (10 points)

Consider the following function definition:

```
def mystery_function(x, y):
    if x == 0:
        return 1
    return y * mystery_function(x - 1, y)
```

A. What does the following function call return?

```
mystery_function(0, 100)
```

B. What does the following function call return? Show your work if you're worried about your arithmetic.

```
mystery_function(2, 3)
```

C. How would you summarize what this function does in just a few words?

Don't explain the code line-by-line. Provide a higher-level description like "adds *x* and *y*" or "computes *x* factorial."

Problem 2: Selection sort (10 points)

Consider the following list:

```
L = ['Peter',
     'Chris',
     'Meg',
     'Lois',
     'Stewie',
     'Joe',
     'Quagmire',
     'Brian'  ]
```

In the diagrams below, show the contents of the list after each of the first 4 iterations of the for-loop in `selection_sort`. If the list does not change from one iteration to the next, you can write "SAME" for the next iteration.

Selection sort code is on the next page for your reference, but you won't need it if you understand the algorithm.

INDEX	INITIAL ORDER	AFTER i=0 ITERATION	AFTER i=1	AFTER i=2	AFTER i=3
0	Peter				
1	Chris				
2	Meg				
3	Lois				
4	Stewie				
5	Joe				
6	Quagmire				
7	Brian				

This selection sort code works identically to the selection sort code in your lab:

```
def selection_sort(list_to_sort):
  for i in range(len(list_to_sort) - 1):
    min_index = find_min_index(list_to_sort, i)
    list_to_sort[i], list_to_sort[min_index] =
      list_to_sort[min_index], list_to_sort[i]


def find_min_index(L, s):
     min_index = s
     for i  in range(s, len(L)):
        if L[i] < L[min_index]:
            min_index = i
    return min_index
```

Problem 3: Binary search (10 points)

Consider the following sorted list:

```
L = ['Brian',
     'Chris',
     'Joe',
     'Lois',
     'Meg',
     'Peter',
     'Quagmire',
     'Stewie' ]
```

and the following binary search (which is essentially identical to your lab code):

```
# binary_search()
# Finds the position of an item in a list
# Parameters: the list; the item to search for
# Returns: the item's position (or None)
def binary_search(search_list, value_to_find):
     first = 0
     last = len(search_list) - 1

     while first <= last:
          middle = (first + last) // 2
          if value_to_find == search_list[middle]:
               return middle
          elif value_to_find < search_list[middle]:
               last = middle - 1
          else:
               first = middle + 1
     return None
```

You may want to label the elements of L with their numeric index values before proceeding.

Answer the questions on the next page.

(a) Fill out the following table tracing a binary search for `'Joe'` in this list. You should fill out one row per iteration of the loop. If there are more rows than iterations, leave the extra rows blank.

Iteration	**Value of `first`**	**Value of `last`**	**Value of `middle`**	**Value of `L[middle]`**
1				
2				
3				
4				
5				

(b) Fill out the following table tracing a binary search for `'Mayor West'` in this list.

Iteration	**Value of `first`**	**Value of `last`**	**Value of `middle`**	**Value of `L[middle]`**
1				
2				
3				
4				
5				

Problem 4: Object-oriented programming terminology (10 points)

Consider the following Python code:

```
class K:
   def __init__(self, a):
       self.a = a

   def __str__(self):
       return str(self.a)

   def k(self):
      return 10

b = 5                   # Calls method ______________

c = K(6)                # Calls method ______________

d = K(10)               # Calls method ______________

print(b)                # Calls method ______________

print(c)                # Calls method ______________
```

A. What is the data type of the variable `b`?

B. What is the data type of the variable `c`?

C. What is the data type of the variable `d`?

D. List the methods of class `K`.

E. In each comment above, fill in the blank with the method(s) of class `K` that are called in the execution of each line.
If a line does not call a method of class `K`, write N/A.

Problem 5: Creating classes (30 points)

In this problem, you will define a class to represent an item on a food menu.

If you use the input() or print() functions in your solution to this problem, you're doing it wrong!

Your class should be named `Food`, and you should define the following methods:

`__init__`: This method initializes a `Food` object.

- Parameters:
  - Food name (e.g. `'eggplant parmigiana'`)
  - Price (e.g. `12.95`)
  - Whether the item is vegetarian (will be either `True` or `False`)
- Should initialize internal variables for the food name, price, and whether or not it's vegetarian, based on the information that was passed in.

`__str__`: This method returns a string that contains the object's information in the format shown below. You should add the (V) when you print the name of a vegetarian item:

```
Eggplant parmigiana (V): $12.95
```

or

```
Bacon-wrapped steak: $22.0
```

`__lt__`: This method compares `self` to another `Food` object.
It returns `True` if `self`'s price is lower than the other object's price and `False` otherwise.
For example, if `self` is the eggplant parmigiana and `other` is the bacon-wrapped steak, it should return `True`.

`get_name`: This method returns the name of the food item

`get_price`: This method returns the price of the food item

`is_veg`: This method returns `True` if the item is vegetarian and `False` otherwise

`add_bacon`: This method does not return anything, but it changes the `self` object in three ways:

- The name of the object gets 'with bacon' added to it. So `eggplant parmigiana` becomes `eggplant parmigiana with bacon`.
- The price goes up by $3.
- The vegetarian indicator should become `False`.

[WRITE YOUR PROBLEM 5 CODE HERE]

Problem 6: Using classes (30 points)

For this problem, you must write a **complete program**. However, you can assume that the `Food` class from Problem 5 has already been correctly defined for you.

To earn full credit, you must use the methods of the `Food` class whenever possible.

Read the instructions carefully before you start coding!

Your program should contain the following:

1. A function called `CreateFood` to do the following:
    - Ask the user to enter the item and its price on two separate lines.
      ```
      bacon-wrapped steak
      22
      ```
    - If the first word of the item's name is a capital V, then the item is vegetarian. Remove this V from the name of the item. For example:
      ```
      V eggplant parmigiana
      12.95
      ```
    - If the name of the food is a blank line, return `None`.
    - If the price is not a positive number, exit the program with an error message.
    - Otherwise, create ***and return*** a `Food` object that uses the information the user entered.

2. A `main` function to do the following:
    - Call `CreateFood` repeatedly until the user enters a blank line instead of the menu information.
    - Use the results of `CreateFood` to build a list of foods.
    - Using your `Food` methods, do the following:
        - Find the most expensive item on the menu
        - Add bacon to it
        - Print its info in the following format:

      ```
      You ordered:
      Bacon-wrapped steak with bacon: $25
      ```

[WRITE YOUR PROBLEM 6 CODE HERE]

[EXTRA SPACE]