Quiz #6: Linked Lists, Trees, Inheritance ICS-33 Spring 2017

Print only this page, answer problem #1 and #2 on it, and submit it on **Friday**, at the **start of lecture**, in a pile for your Lab at the front of class.

Name (Last, First) _____

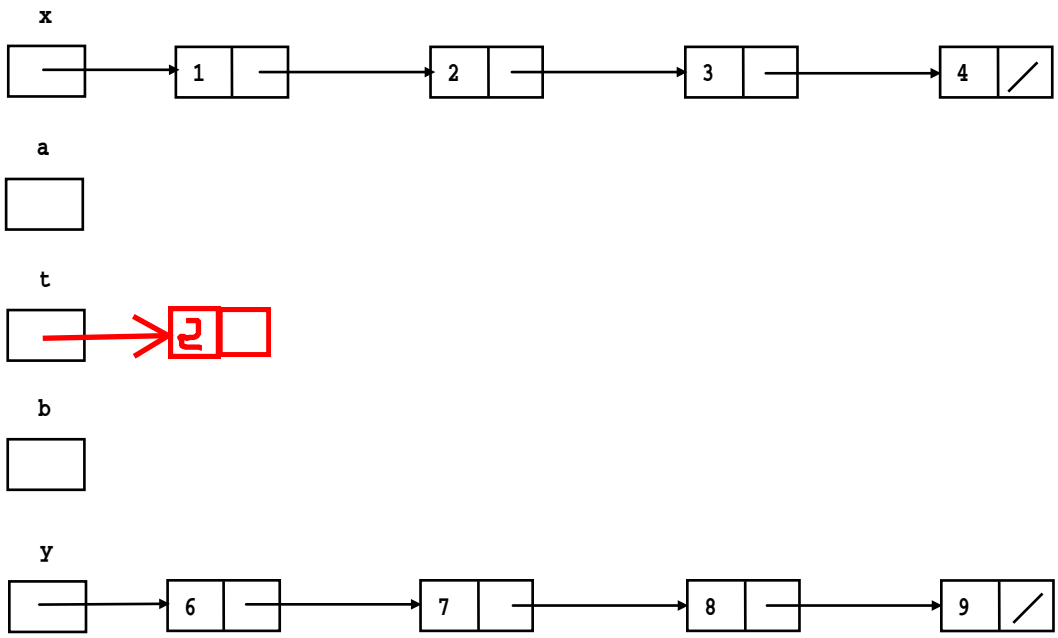
Lab # _____

When working on this quiz, recall the rules stated on the Academic Integrity statement that you signed. You can download the **`q6helper`** project folder (available for Friday, on the **Weekly Schedule** link) in which to write/test/debug your code. Submit your completed **`q6solution`** module online by Thursday, 11:30pm. I will post my solutions to EEE reachable via the **Solutions** link on Friday afternoon.

---

1. (6 pts) Examine the **`mystery`** method and hand simulate the call **`mystery(x,y);`** using the linked list below. **Lightly cross out** ALL references that are replaced and **Write in** new references: don't erase any references. It will look a bit messy, but be as neat as you can. Show references to **`None`** as **`/.`** Do your work on scratch paper first.

```
def mystery(a,b):
    while b != None:
        t = a.next.next
        a.next.next = b
        a = b
        b = t
```



2. (3 pts) Draw the binary search tree that results from adding the following values (in the following order) into an empty binary search tree: **`7`**, **`6`**, **`9`**, **`3`**, **`5`**, **`15`**, **`13`**, **`14`**, **`1`**, **`8`**, **`11`**, **`12`**, **`2`**, **`10`**, and **`4`**. Draw just the number for each tree node, with lines down to its children nodes. Try to space-it-out to be easy to read. Also, answer the questions on the right.

Size =

Height =

3a. (3 pts) Define an **iterative** function named **no_adj_dup**; it is passed a linked list (**ll**) as an argument. It returns a reference to the front of a **new** linked list (comprising **new LN**s, without mutating the parameter linked list) that includes all values in the parameter linked list, except when some value occurs adjacent to itself more than once: it appears there only once. For example if we defined

```
a = list_to_ll([1,1,2,2,2,3,4,3,3,5,5,6,7,7])
```

calling **no_adj_dup(a)** returns the linked list **1->2->3->4->3->5->6->7->None**. You **may not** use Python **lists**, **tuples**, **sets**, or **dicts** in your code: **use linked list processing only**. Hints: this code both traverses the parameter linked list while building the new linked list (it is a variant of the code for copying a linked list, which appears in the course notes)**;** I had 10 lines of code, including a while loop and nested **if** statements. Think about the most general condition about when you want to add a value to the new linked list, based on a value and the value –if any- following it in the unchanging parameter linked list). Debug your code by hand simulation.

3b. (3 pts) Define a **recursive** function named **no_adj_dup** that is given the same argument and produces the same result as the iterative version above. You **may not** use Python **lists**, **tuples**, **sets**, or **dicts** in your code: **use linked lists processing only**: use no looping; you can define/use a local variable but cannot rebind it or mutate its value. Hint: use the 3 proof rules to help synthesize your code. You might try writing the recursive solution first, it is simpler (mine is 8 simple lines, including binding the recursive call).

4. (4 pts) We learned that when we declare a **class** using inheritance, the **__bases__** attribute of the **class** is bound to a **tuple** of references to its **base** classes. Write a **recursive** function named **bases** that takes a reference to any **class** and returns a **set** containing that **class** and all its base classes (back to **object**). You may not use the **__mro__** or **mro** attributes of the **class**, which would trivialize your function. You may use both iteration (over the **__bases__** list) and recursion. For example, given the following **class** definitions in a script

```
class F:pass
class C:pass
class G:pass
class B(F):pass
class D(G):pass
class A(B,C,D):pass
```

Calling **bases(A)** returns the **set**

```
{<class '__main__.A'>, <class '__main__.D'>, <class
'__main__.B'>, <class '__main__.G'>, <class 'object'>,
<class '__main__.C'>, <class '__main__.F'>}
```

Hint: use the **union** function for **sets**, which uses ***others** to compute the **union** of any number of **sets** (or use **functools.reduce**; **|** is the **set** union operator). I wrote 4 lines of code (including a comprehension).

5. (6 pts) Define a derived class named **StringVar_WithHistory**, based on the **StringVar** class in **tkinter**; it remembers what sequence of values it was set to, and is able to undo each setting.

The **StringVar** class defines 3 methods: **__init__** (**self**), **get** (**self**), and **set** (**self,value**). The **set** method changes the state of the **StringVar** object to be **value**; the **get** method returns the string it is currently set to. The **StringVar_WithHistory** derived class inherits **get** and overrides **__init__** and **set**.

Define the derived class **StringVar_WithHistory** with only the following methods (get is purely inherited):

- **__init__** (**self**): initializes the base class; creates a history **list** for storing the values **set** is called with.
- **set** (**self,value**): if the value is different from the current value, **set** the **StringVar** to **value** and remember it in the history **list** (if it is the same as the current value, do nothing: no *new* selection).
- **undo** (**self**): undo the most recently selected option by updating the **StringVar** and the history **list** (but only if the currently selected option wasn't the first one: that selection cannot be undone).

You cannot test **StringVar_WithHistory** by itself, but must test it using the **OptionMenuUndo** class, which is in the download (itself class derived from **OptionMenu**). You can simulate the GUI (how I will test it) or can actually build/test a version of the GUI that allows you to click the GUI to select options and undo selections. See the simulation in the download (for how it should behave on one complex example).

Note: if you see the error message **`AttributeError: 'NoneType' object has no attribute 'globalgetvar'`** then you have not initialized the **`StringVar`** appropriately by calling its **`__init__`** method.