UNIVERSITY OF TORONTO
Faculty of Arts and Science

Midterm 1
CSC148H1F – L0201 (Liu)

October 21, 2016 (**50 min.**)
Examination Aids: Provided aid sheet (back page, detachable!)

Name:

Student Number:

Please read the following guidelines carefully!

- Please write your name on the front **and back** of the exam.
- This examination has **4** questions. There are a total of **10 pages, DOUBLE-SIDED**.
- You may always write helper functions/methods unless explicitly asked not to.

Take a deep breath.

This is your chance to show us

How much you've learned.

We **WANT** to give you the credit

That you've earned.

A number does not define you.

Good luck!

1. The following questions test your understanding of the terminology and concepts from the course. You may answer in either point form or full sentences; **you do not need to write much to get full marks**!

   (a) [**2 marks**] Suppose we are reading the code of a particular class. We see a method whose name begins with an underscore _. What term do we use to describe this kind of method, and why do we write such methods?

   (b) [**2 marks**] A student asks: "why do we bother using `unittest` to test our code, when we could just write doctests?" Explain why we use `unittest` in addition to writing doctests.

   (c) [**1 mark**] When writing an abstract class, we often write methods whose body is a single line of code: `raise NotImplementedError`. What is the *purpose* of writing such a method?

   (d) [**1 mark**] Suppose we write the following function:

```
def remove_middle(lst):
    """Remove and return the middle item in <lst>.

    Precondition: len(lst) is odd.

    @type lst: list
    @rtype: object
    """
    mid = len(lst) // 2
    lst.pop(mid)
```

   Unfortunately, this function does not behaves exactly as expected. Explain why not.

(e) [**2 marks**] Do you expect the running time of the `remove_middle` function (from the previous part) to depend on the length of its input list? Explain.

(f) [**2 marks**] Consider the following function, which operates on a Python stack.

```
def size(stack):
    copy = stack
    count = 0
    while not copy.is_empty():
        copy.pop()
        count = count + 1
    return count
```

Fill in the output of the last statement below. Then underneath, explain why this happens.

```
>>> s = Stack()
>>> s.push('a')
>>> s.push('b')
>>> size(s)
2
>>> s.is_empty() # FILL IN THE OUTPUT IN THE NEXT LINE
```

Explanation:

2. [**4 marks**] We want to create a subclass of `Stack` called `GuardedStack`, which has the following differences:

   - It has a new attribute `is_good`, which is a **function** that takes an object and returns a boolean value. This attribute is initialized through a parameter to the `GuardedStack` constructor.
   - When pushing a new item onto a `GuardedStack`, the stack's `is_good` attribute is first called on the item. If this returns `True` then the item is pushed onto the stack, otherwise a `ValueError` is raised.

   Here is a sample usage of `GuardedStack`.

```
>>> def is_even(num):
...     return num % 2 == 0
...
>>> g = GuardedStack(is_even)
>>> g.push(2)
>>> g.push(4)
>>> g.push(5)
ValueError  # error raised here
>>> g.push(16)
>>> g.pop()
16
>>> g.pop()
4
```

   In the space below, show how to override the relevant `Stack` methods in the `GuardedStack` class. You *must* properly call superclass methods when appropriate. (Note: neither method is very long.)

```
def GuardedStack(Stack):
    def __init__(self, is_good):









    def push(self, item):
```

3. **[9 marks]** In this question, you are responsible for designing a class to represent an item being sold in a store. Each item has a name and original price.

   An item can be marked as "on sale", at which point there is a percentage discount on its price. Once an item is on sale, it is on sale forever. Even after an item is on sale, it must be possible to calculate (using the attributes you define) both the *original price* of an item, and its *current price*. An item's current price is equal to the item's original price, with its discount applied if it is on sale.

   An item *always starts as not on sale*; in this case, its current price is equal to its original price.

   Below and on the next page, we have a very incomplete class design for this scenario. You have four tasks:

   - Fill in all the **attributes** of the `Item` class. These can all be public. Be sure to specify a type and description for each.

     It must be possible to compute both the original and the current price from the attributes that you store, though you do not need to show how to do this.
   - Write *one* plausible **representation invariant** for your class. You have some freedom here; just make sure your representation invariant makes sense in the context of this situation.
   - Write the **constructor** of the `Item` class. A docstring is not necessary.
   - Complete the docstring and implementation of a method `put_on_sale` that allows an item to be marked as "on sale." No doctests are necessary.

     You must decide what happens when this method is called on an item that is already on sale; clearly document the expected behaviour, and make sure you implement it correctly.

```
class Item:
    """An item for sale in a store.

    === Attributes ===
    TODO: document all attributes.










    === Representation Invariant ===
    TODO: document *one* representation invariant.





    """
    # TODO: Design and implement a constructor here.
    # Remember that a docstring is not necessary.



    def put_on_sale(self, discount):
        """Mark this item as on sale with a given discount.

        TODO: document what happens if this method is called on an item that is
        already marked as on sale.



        Precondition: 0 <= discount <= 1

        @type self: Item
        @type discount: float
        @rtype: None
        """
        # TODO: implement this method.
```

4. For this question, you should refer to the documentation of the `LinkedList` class found on the aid sheet. You may use all attributes of the `LinkedList` and `_Node` classes, as well as their constructors. You may also access the `__str__` method for `LinkedList`, but no other methods.

   (a) [**1 mark**] Suppose we have created the following two linked lists:

   ```
   >>> linky = LinkedList([10, 3, 100, 0])
   >>> lando = LinkedList([44, 13, 4])
   ```

   Write an expression that accesses the *item* at index 2 in `linky`, and stores it in a variable `val`.

   (b) [**2 marks**] Write a code snippet that replaces the contents of `lando` so that it is a `LinkedList` that only stores a single item, which is equal to `val * 2`.

   ```
   >>> # After your code runs, we should be able to see the following.
   >>> print(lando)
   [200]
   ```

   (c) [**2 marks**] A student comes to you with the following implementation of a `LinkedList` method, saying that "when I ran it there was a big block of red text."

   Explain the problem with the code below, and how to fix it to satisfy its docstring.

   ```
   def count(self, item):
       """Return the number of occurrences of <item> in this linked list.

       @type self: LinkedList
       @type item: object
       @rtype: int

       >>> linky = LinkedList([5, 8, 5, 4])
       >>> linky.count(5)
       2
       """
       curr = self._first
       num = 0
       while curr is not None:
           if curr == item:
               num = num + 1
           curr = curr.next
       return num
   ```

Use this page for rough work. If you want work on this page to be marked, please indicate this clearly *at the location of the original question.*

Use this page for rough work. If you want work on this page to be marked, please indicate this clearly *at the location of the original question.*

Name:

	Q1	Q2	Q3	Q4	Total
Grade					
Out Of	10	4	9	5	28