UNIVERSITY OF TORONTO
Faculty of Arts and Science

Midterm 1
CSC148H1F – L0101 (Horton)

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 **8 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 designing a class called `Club` for keeping track of the members of a club. We plan to have a private attribute that is a list of the members' email addresses. Write the portion of the docstring for class `Club` that describes this attribute (you can choose its name).

   (b) **[1 mark]** Suppose the `Club` class will also have a method called `sign_up` which is called when a new member joins the club, and will add their name to the front of the list of members. Assuming that we are committed to using a list of email addresses, what simple change to the class will make the `sign_up` method run faster?

   (c) **[1 mark]** Name a class from Assignment 1 that was abstract (approximate name is fine):

   (d) **[1 mark]** What is the one thing that client code should never do with an abstract class?

   (e) **[2 marks]** Sometimes a class is so abstract that it has no attributes and no method bodies. What do we gain by defining such a seemingly useless class?

   (f) **[2 marks]** Suppose we have a `Queue` class (a regular queue, not a priority queue), and it includes this method:

```
def num_items(self):
    count = 0
    while not self.is_empty():
        temp = self.dequeue()
        count += 1
        self.enqueue(temp)
    return count
```

   Explain why the loop in this method is an infinite loop.

   (g) **[1 mark]** Suppose we fix method `num_items`. But rather than the usual way of calling a method, such as `q.num_items()`, we want to be able to call it as `len(q)`. How must we change the method to make this possible?

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

   - It has a new attribute `is_special`, which is a **function** that takes an object and returns a boolean value. This attribute is initialized from a parameter to the `__init__` method.
   - When enqueueing a new item into a `DoubleQueue`, its `is_special` attribute is first called on the item. If this returns `True` the item is goes into the `DoubleQueue` twice, otherwise it goes in once as usual.

   Here is a Python interactive session that demonstrates the usage of `DoubleQueue`.

```
>>> def loud_word(s):
...     return s in ['crash', 'bang', 'bellow', 'holler', 'honk']
...
>>> q = DoubleQueue(loud_word)
>>> q.enqueue('quiet')
>>> q.enqueue('honk')
>>> q.enqueue('peaceful')
>>> q.dequeue()
'quiet'
>>> q.dequeue()
'honk'
>>> q.dequeue()
'honk'
>>> q.dequeue()
'peaceful'
```

   In the space below, show how to override the relevant `Queue` methods in the `DoubleQueue` class. You *must* properly call superclass methods when appropriate.

```
class DoubleQueue(Queue):
    def __init__(self, is_special):









    def enqueue(self, item):
```

   (You don't need any further space to answer this question.)

3. **[9 marks]**

You are responsible for designing a class to keep track, for each value in a set of values, the number of times it occurs. (This is called a "frequency distribution".) Here is an example of how we want to use it:

```
>>> d = Distribution('Grades for Lab 3', ['A', 'B', 'C', 'D', 'F'])
>>> d.add_occurrence('A')
>>> d.add_occurrence('C')
>>> d.add_occurrence('B')
>>> d.add_occurrence('A')
>>> d.add_occurrence('B')
>>> d.add_occurrence('B')
>>> d.num_occurrences('D')
0
>>> d.num_occurrences('B')
3
>>> d.num_occurrences('A')
2
```

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

- In the class docstring fill in all the **attributes** of the `Distribution` class. You may choose any reasonable way to store the necessary data. Make all attributes private.
- Implement the `__init__` method. A method docstring is not necessary.
- Implement method `add_occurrence`. It's up to you to decide what happens when this method is called on an item that is was not in the list provided when the `Distribution` was constructed.
- Complete the docstring for `add_occurrence`.

Ensure that the code above would run, assuming we defined the additional method `num_occurences`.

```
class Distribution:
    """A frequency distribution.

    === Attributes ===
    # TODO: YOUR ATTRIBUTES GO HERE.









    """
```

```
    # TODO: Implement method __init__ here.
    # Remember that a docstring is not necessary.
```

```
def add_occurrence(self, value):
    """
    Record that <value> occurred once / once more.

    TODO: Complete the docstring so that it describes the method's behaviour
    in all circumstances.

    @type self: Distribution
    @type value: object
    @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) [**3 marks**] Suppose we have done this:

   ```
   >>> linky = LinkedList([6, 12, 18, 24, 30])
   ```

   Fill in the gaps below to show the the value of each Python expression. If an expression raises an error, explain why.

   ```
   >>> type(linky.next.next.item)
   ```

   ```
   >>> linky.next.next.item
   ```

   ```
   >>> type(linky.item.next)
   ```

   ```
   >>> linky.item.next
   ```

   ```
   >>> type(linky._first.next.item)
   ```

   ```
   >>> linky._first.next.item
   ```

   (b) [**1 mark**] In the gap below, write a one-line snippet of code to remove the first node in the linked list and ensure the output shown.

   ```
   >>> linky = LinkedList([5, 8, 2, 9, 3])

   >>> print(linky)

   [5 -> 8 -> 2 -> 9 -> 3]

   >>>                                                    # YOUR WORK GOES ON THIS LINE

   >>> print(linky)

   [8 -> 2 -> 9 -> 3]
   ```

   (c) [**1 mark**] Why does the time required to insert at the front of a linked list **not** grow in proportion to the length of the list?

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