CSC 148H1 Winter 2008 Midterm Test
Duration — 50 minutes
Aids allowed: none

Student Number: |_|_|_|_|_|_|_|_|_|_|_|

Lab day, time, room: _____

Last Name: _____ **First Name:** _____

Lecture Section: L5101 **Instructor:** Gries

---

*Do **not** turn this page until you have received the signal to start.*
(Please fill out the identification section above, **write your name on the back of the test**, and read the instructions below.)
Good Luck!

This test consists of 4 questions on 10 pages (including this one). *When you receive the signal to start, please make sure that your copy is complete.*
Comments and docstring are not required except where indicated, although they may help us mark your answers. They may also get you part marks if you can't figure out how to write the code.
If you use any space for rough work, indicate clearly what you want marked.

1: _____/10
2: _____/10
3: _____/10
4: _____/10

TOTAL: _____/40

Question 1. [10 MARKS]

This question is about binary trees that may or may not be binary *search* trees. The problem here is to examine a binary tree and determine whether or not it satisfies the conditions required for a BST (binary search tree): for every node, all the keys in that node's left subtree are less than the node's key, and all the keys in its right subtree are greater.

Part (a) [2 MARKS]

In one or two *short* English sentences, state a strategy for deciding whether a binary tree is a BST.

Part (b) [8 MARKS]

Complete the function `is_bst` begun below, using the strategy you stated in part (a). The parameter for `is_bst` is a `Node`; the class definition for `Node` is given for you.

You may assume that all keys are integers. Very likely you will find it best to write one or more helper functions.

```
class Node(object):
    def __init__(self, key):
        self.key = key
        self.left = None
        self.right = None

def is_bst(root):
    '''Return True or False according to whether the binary tree rooted
    at root satisfies the requirements of a binary search tree.
    '''
```

Question 2. [10 MARKS]

In this question we are managing a warehouse — specifically, we are keeping track of the inventory in the warehouse. An object-oriented design is needed, to model the inventory and also the individual items in the warehouse.

The items have names, which we will consider to be unique keys. The inventory will store information about the items in a dictionary, using the item names as keys and the quantity of each item as the value for that key.

Part (a) [2 MARKS]

Write the `Warehouse` class, including the `__init__` method, which must set up the dictionary, called "`inventory`". No other methods are needed at this point.

Part (b) [3 MARKS]

Write the `Item` class, representing an item that can be stored in the warehouse. Every item must have a name; assume the names are unique (that is, different from the name of every other item), but do not bother trying to check this. The name will be set in the `__init__` method.

Besides `__init__`, an `Item` needs one other method, `add_to_inventory`, which takes one parameter: the inventory, which is a dictionary with item names as keys.

Part (c) [2 MARKS]

In part (b), you told an `Item` how to add itself to an inventory. Now, you need to write a new method for the `Warehouse` class, so that a `Warehouse` can be given an `Item` and tell the item to add itself to the `Warehouse`'s inventory.

Here is the beginning of the method; please complete it.

```
# in class Warehouse:
    def add_item(self, item):
```

Part (d) [3 MARKS]

Some items aren't counted but measured; for example, you might add a tricycle to the inventory, but you would add wheat by the ton, not by counting grains.

Define a new class, `MeasuredItem`, that is a subclass of `Item`. Each instance of `MeasuredItem` has not only a name but also a quantity.

Write the class `MeasuredItem` so that it can be used in the same way as `Item` without requiring changes to `Warehouse` or `Item`.

Question 3. [10 MARKS]

Consider the following method.

```
def average(L):
    '''Return the average of the numbers in L.'''
    sum = 0.0
    count = 0
    for value in L:
        sum = sum + value
        count = count + 1
    return sum / count
```

Part (a) [3 MARKS]

There are at least two different possible errors that might occur during execution of `average`. One is when the list is empty. Describe another one.

Part (b) [5 MARKS] Here is the output when you call `average` with an empty list:

```
ZeroDivisionError: integer division or modulo by zero
```

That isn't very helpful. Rewrite `average` (including the docstring) to raise an `EmptyListError` with a better error message when given an empty list; also define class `EmptyListError`.

Part (c) [3 MARKS]

Write two `nose` tests that test whether the appropriate exceptions are raised in the two error situations as expected. (Note: we don't expect you to use all the space on this page.)

Question 4. [10 MARKS]

Part (a) [4 MARKS] Consider the following code.

```
def mystery(s):
    if len(s) == 0:
        return ""
    elif len(s) == 1:
        return s
    elif s[0] == s[1]:
        return mystery(s[1:])
    else:
        return s[0] + mystery(s[1:])
```

What does `mystery` do? **Describe in one English sentence.**

Part (b) [6 MARKS]

Write a recursive function called `to_words` that takes a single `int` parameter and returns a `str` that contains the digits in the `int` in English.

For example, the call

```
to_words(23561)
```

would result in the following:

```
'two three five six one '
```

Assume the following list of `str`s is defined in the file that contains the function you are writing.

```
NUMBERS = ['zero', 'one', 'two', 'three', 'four', 'five', 'six', 'seven',
       'eight', 'nine']
```

Hint: Use the `/` and `%` operators.

Use this page for rough work and for any answers that didn't fit.

Last Name: _____ **First Name:** _____