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

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

Lab day, time, room: _____

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

Lecture Section: L0101 **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 8 pages (including this one). *When you receive the signal to start, please make sure that your copy is complete.*
Comments 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]

The textbook discussed two ways to represent binary trees: using nested lists, and using node objects. For example, a tree with `'A'` as the root value, `'B'` as `'A'`s left child, and no right child, could be represented using this list: `['A', ['B', None, None], None]`. An empty tree is just `None`.

Here is a `Node` class:

```
class Node:
    def __init__(self, v, L=None, R=None):
        '''A new Node with value v, left child Node L, and right child Node
        R.'''
        self.value = v
        self.left = L
        self.right = R
```

Complete the recursive function below. The solution should **easily** fit in the space provided.

```
def to_list(root):
    '''Return a nested list representation of the tree rooted at Node root,
    where the root is at index 0, the left subtree is at index 1, and the
    right subtree is at index 2.'''
```

Question 2. [10 MARKS]

Part (a) [6 MARKS]

Follow the Huffman tree-building process for the message `"bwahaha"`. (Don't bother with EOF.)

- Initial forest:

- Forest after one step in the tree creation (after two nodes are combined):

- Continue drawing the forests until the process is finished. Separate each step with a horizontal line.

Part (b) [2 MARKS] What is the binary representation for the letter `'w'`? _____

Part (c) [2 MARKS] Is your Huffman tree the only possible one for this message? Circle the answer:

`Yes` `No`

Question 3. [10 MARKS]

This question is on both this page and the following one.

`type(o)` returns `o`'s type. `type(1)`, for example, returns the `int` type.

Sometimes it might be nice to make sure that a data structure such as a queue holds only values of a particular type. Below is the beginning of such a queue type.

```
class TypesafeQueue:
    def __init__(self, my_type):
        '''A Queue that only holds items of type my_type.'''
        self.qtype = my_type
        self.queue = []

    def enqueue(self, o):
        '''Append o to me; raise a QueueTypeError if this queue does not hold
        items of type(o).'''
        self.queue.append(o)

    def dequeue(self):
        '''Remove and return the top item.'''
        return self.queue.pop(0)

    def front(self):
        '''Return the top item.'''
        return self.queue[0]

    def isEmpty(self):
        '''Return whether there are any items in this queue.'''
        return self.queue == []

    def size(self):
        '''Return the number of items in this queue.'''
        return len(self.queue)
```

This might be used as follows to make a queue that only holds `str`s: `q = TypesafeQueue(str)`.

The problem is that the functions don't yet prevent other types from being put into the queue.

Part (a) [3 MARKS]

Write a test function that tests whether a `QueueTypeError` is thrown when a `TypesafeQueue` raises an exception when expected.

Part (b) [4 MARKS]

Rewrite `enqueue` so that it raises a `QueueTypeError` if an item of the wrong type is inserted. Also define your exception class. For full marks, the exception output should include both the value of the wrong type, and the type of item the `TypesafeQueue` holds.

Part (c) [3 MARKS]

Now write the body of the following function. You must use try/except to deal with `QueueTypeError`s.

```
def add_all(q, L):
    '''Enqueue all items in list L to TypesafeQueue q. Print any items that
    are the wrong type.'''
```

Question 4. [10 MARKS]

Here is the order in which nodes in a particular tree are visited in an inorder traversal:

```
D  G  B  A  E  C
```

Here is the order in which nodes are visited in that same tree, but in a preorder traversal:

```
A  B  D  G  C  E
```

Here is the order in which nodes are visited in that same tree, but in a postorder traversal:

```
G  D  B  E  C  A
```

Draw the tree:

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

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