CSC 148H1 Summer 2008 Midterm Test
Duration — 60 minutes
Aids allowed: none

Student Number: ____________

Lab day: ____________

Last Name: ____________ **First Name:** ____________

Lecture Section: L0101 **Instructor:** R. Danek

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 5 questions on 10 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:	______/ 8
# 4:	______/10
# 5:	______/10
TOTAL:	______/48

Question 1. [10 marks]

Why is searching for an item in a binary search tree sometimes more efficient than searching for an item in a list? Under what conditions is searching a BST no more efficient than searching a list? Use a diagram or two in your explanation. **(Note: Your explanation should be at a high-level and contain no more than a few sentences. Do not write any code for this question.)**

Question 2. [10 MARKS]

A casino has acquired a new slot machine that has a very nice payout scheme, at least from the perspective of the gambler. Whenever a person puts in 1 coin, the machine pays out 1 coin. Whenever a person puts in 3 coins, it pays out 4 coins. Whenever a person puts in n coins (for n other than 1 or 3): if n is even, then the machine pays out the same number of coins that would have been paid out if the person had put in $2n + 1$ coins; and if n is odd, then the machine pays out 3 coins **plus** the same number of coins that would have been paid out if the person had put in $n - 2$ coins.

Write a recursive function `payout` that takes n as an argument and returns the number of coins that the slot machine will payout if n coins are put into it. Clearly point out what the base cases are in your function.

Hint: To test if a number n is even or odd, determine n's remainder when divided by 2. Recall that % is the remainder operator.

Question 3. [8 MARKS]

This question is on both this page and the next.

Assume that you have made the following definitions in a module:

```
class BarException(Exception):
    pass
class FooException(Exception):
    pass

x = 5

def g(n):
    return n * 2

def f(n):
    try:
        def g(n):
            return n * 3

        x = g(n)
        try:
            if n == 0:
                raise BarException

            if n == 1:
                raise FooException

            print "XYZ"
        except FooException:
            print x
        except:
            print "ABC"
            raise
    except BarException:
        print x

def k():
    print x

# *** PLACE CODE FROM EACH SUBQUESTION HERE ***
```

Question is continued on next page...

For each of the following subquestions, write the output you would expect on the screen if you put the code in the specified position above and ran the module.

Part (a) [2 marks]

```
f(0)
```

Part (b) [2 marks]

```
f(1)
k()
```

Part (c) [2 marks]

```
f(2)
print g(x)
```

Part (d) [2 marks]

```
f(x)
```

Question 4. [10 MARKS]

Assume that $A < B < C < ... < Z$ (as in the natural ordering of the alphabet).

Here is the order in which nodes in a particular **binary search tree** are visited in a postorder traversal:

```
A E D C J K H F
```

Draw the tree:

Question 5. [10 MARKS]

This question is on both this page and the next.

In this question, assume you have access to a Queue implementation with the following class and method definitions:

```
class Queue:
    def __init__(self):
        '''Make a new empty queue'''
        # implementation details omitted ...

    def enqueue(self, o):
        '''add o to the end of the queue'''
        # implementation details omitted ...

    def dequeue(self):
        '''remove the front element from the queue and return it'''
        # implementation details omitted ...

    def front(self):
        '''return the front element from the queue'''
        # implementation details omitted ...

    def isEmpty(self):
        '''return True if the queue is empty, False otherwise'''
        # implementation details omitted ...

    def size(self):
        '''return the number of elements in the queue'''
        # implementation details omitted ...
```

On the next page you are given a partially complete implementation of the Stack ADT. As you can see, it doesn't use a python list like we did in class; instead, it makes use of a `Queue` instance.

You must complete the implementation of the `pop` and `size` methods, obeying the following rules:

- You cannot change the `push` method or `__init__` method.
- You cannot use python lists or dictionaries anywhere in your code.
- You **can** create a new instance of the `Queue` class if you want.

Don't worry about the efficiency of your implementation. In fact, to correctly implement `pop` while obeying the given constraints, your solution will necessarily be inefficient. Also, don't worry about throwing/handling exceptions, and you can ignore the fact that the `peek` and `isEmpty` methods aren't defined.

```
class Stack:
    def __init__(self):
        '''make a new empty stack'''
        self.container = Queue()

    def push(self, value):
        '''add an element onto the top of the stack'''
        self.container.enqueue(value)

    def size(self):                                        # (2 marks)
        '''return the number of elements in the stack'''
        # add your code here




    def pop(self):                                         # (8 marks)
        '''remove the top element from the stack and return it'''
        # add your code here
```

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

Last Name: ______________________ **First Name:** ______________________