CSC148H1 Winter 2009 Midterm Test
Duration — 50 minutes
Aids allowed: none

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

Lab day, time, room: ________________________

Family Name: ________________________ **Given 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 3 questions on 10 pages (including this one). *When you receive the signal to start, please make sure that your copy is complete.*
Comments and docstrings 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:	______/ 8
# 2:	______/ 5
# 3:	______/12
TOTAL:	______/25

Question 1. [8 marks]

Mathematical expressions can be represented as trees. Consider the following class. (This class only deals with + and * as operators.)

```
class ExprTree(object):
    '''An expression tree where the root value is either an int (with no children) or is one of
    '+' and '*', in which case the left and right subtrees are themselves ExprTree objects.'''

    def __init__(self, v, left=None, right=None):
        '''A new expression with value v.'''

        self.value = v
        self.left = left
        self.right = right

    def evaluate(self):
        '''Evaluate the expression stored in this tree and return the result.'''

        # To be completed on the next page.

    def __str__(self):
        '''Return the expression represented by this tree as a fully-parenthesized string.
        For example, if the tree has '+' at the root and 3 and 5 as the left and right children,
        the return value will be '(3+5)'; if this tree only has a leaf, the value is the int
        represented as a string.'''

        if isinstance(self.value, int):
            return str(self.value)

        # To be completed on the next page.

if __name__ == '__main__':
    leaf1 = ExprTree(1)
    leaf3 = ExprTree(3)
    leaf4 = ExprTree(4)
    leaf5 = ExprTree(5)
    i1 = ExprTree('*', leaf3, leaf4)
    i2 = ExprTree('+', leaf1, i1)
    expr = ExprTree('*', i2, leaf5)
```

Part (a) [2 marks] Draw the tree built by the program:

Part (b) [3 MARKS] Complete the body of `evaluate`.

```
def evaluate(self):
    '''Evaluate the expression stored in this tree and return the result.'''
```

Part (c) [3 MARKS] `__str__` currently only handles leaves. Complete it:

```
def __str__(self):
    '''Return the expression represented by this tree as a fully-parenthesized string.
    For example, if the tree has '+' at the root and 3 and 5 as the left and right children,
    the return value will be '(3+5)'; if this tree only has a leaf, the value is the int
    represented as a string.'''

    if isinstance(self.value, int):
        return str(self.value)
```

Question 2. [5 marks]

Complete the following function according to its docstring. Remember that `isinstance(k, T)` will tell you whether `k` is of type `T`.

```
def append_to_all(L, v):
    '''Append value v, which may be of any type, to all the nested lists in L.
    L is a list, and may contain other lists.'''
```

```
if __name__ == '__main__':
    L = [1, 2, [3]]
    append_to_all(L, 'a')
    print L  # This prints [1, 2, [3, 'a'], 'a']
```

Question 3. [12 marks]

The servers in a restaurant are starting a competition: at the end of each day, they want to know which server received the highest tip to bill ratio. For example, if a server's bills totaled $250 and they received $50 in tips, they had a good night: that's 20%.

They want a program to manage this. The program needs to keep track of each bill, including the total amount of the bill, which server served the table, and the tip amount. They want to be able to print the list of bills and tips for a particular server.

Once you're done, this code should run without error:

```
if __name__ == '__main__':
    r = Restaurant()
    r.add_bill('Paul', Bill('Paul', 110, 10))
    r.add_bill('Paul', Bill('Paul', 120, 10))
    r.add_bill('Paul', Bill('Paul', 130, 10))
    r.add_bill('Jen', Bill('Jen', 80, 10))
    r.add_bill('Jen', Bill('Jen', 90, 10))
    r.add_bill('Jim', Bill('Jim', 90, 10))
    print r.get_servers() # Should print something like ['Paul', 'Jen', 'Jim']
    print r.get_bills('Paul') # Should print a list containing three Bills.
    print winner(r)  # Should print 'Jen'
```

Part (a) [2 MARKS] Write class `Bill`.

Part (b) [3 MARKS]

Write class `Restaurant`, which has a dictionary as an instance variable. The keys in the dictionary are server names (as strings) and the values are lists of instances of `Bill`. Make sure the code at the beginning of this question will run.

Part (c) [3 marks]

Write function `ratio`, which takes a list of `Bill` objects as a parameter and returns the ratio of the sum of the tips to the sum of the totals.

Part (d) [4 marks]

Write function `winner`, which takes a `Restaurant` as a parameter and returns the name of the server who had the highest tip to bill ratio. Remember your `ratio` function.

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

Short Python function/method descriptions:

```
__builtins__:
  abs(x) -> number
    Return the absolute value of x.
  lambda: expr -> function
    Returns a function that evaluates the Python expression expr.
  len(x) -> integer
    Return the length of the list, tuple, dict, or string x.
  max(L) -> value
    Return the largest value in L.
  min(L) -> value
    Return the smallest value in L.
  open(name[, mode]) -> file object
    Open a file.  Legal modes are "r" (read), "w" (write), and "a" (append).
  range([start], stop, [step]) -> list of integers
    Return a list containing the integers starting with start and ending with
    stop - 1 with step specifying the amount to increment (or decrement).
    If start is not specified, the list starts at 0.  If step is not specified,
    the values are incremented by 1.
dict:
  D[k]  or  D.get(k) -> value
    Return the value associated with the key k in D.
  k in D  or  D.has_key(k) -> boolean
    Return True if k is a key in D and False otherwise.
  D.keys() -> list of keys
    Return the keys of D.
  D.values() -> list of values
    Return the values associated with the keys of D.
file (also called a "reader"):
  F.close()
    Close the file.
  F.read([size]) -> read at most size bytes, returned as a string.
    If the size argument is negative or omitted, read until EOF (End
    of File) is reached.
  F.readline([size]) -> next line from the file, as a string. Retain newline.
    A non-negative size argument limits the maximum number of bytes to return (an incomplete
    line may be returned then).  Return an empty string at EOF.
float:
  float(x) -> floating point number
    Convert a string or number to a floating point number, if possible.
int:
  int(x) -> integer
    Convert a string or number to an integer, if possible.  A floating point
    argument will be truncated towards zero.
list:
  L.append(x)
    Append x to the end of the list L.
  L.index(value) -> integer
    Returns the lowest index of value in L.
  L.insert(index, x)
    Insert x at position index.
  L.remove(value)
    Removes the first occurrence of value from L.
  L.sort()
```

```
    Sorts the list in ascending order.
str:
  str(x) -> string
    Convert an object into its string representation, if possible.
  S.find(sub[,i]) -> integer
    Return the lowest index in S (starting at S[i], if i is given) where the
    string sub is found or -1 if sub does not occur in S.
  S.index(sub) -> integer
    Like find but raises an exception if sub does not occur in S.
  S.isdigit() -> boolean
    Return True if all characters in S are digits and False otherwise.
  S.replace(old, new) -> string
    Return a copy of string S with all occurrences of the string old replaced
    with the string new.
  S.rstrip([chars]) -> string
    Return a copy of the string S with trailing whitespace removed.
    If chars is given and not None, remove characters in chars instead.
  S.split([sep]) -> list of strings
    Return a list of the words in S, using string sep as the separator and
    any whitespace string if sep is not specified.
  S.strip() -> string
    Return a copy of S with leading and trailing whitespace removed.
```

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