PLEASE HAND IN

PLEASE HAND IN

UNIVERSITY OF TORONTO
Faculty of Arts and Science

APRIL/MAY EXAMINATIONS

CSC 148H1S
Instructors: Clarke, Gries

Duration — 3 hours

Examination Aids: None.

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

Last (Family) Name(s): _____

First (Given) Name(s): _____

---

*Do **not** turn this page until you have received the signal to start.*
(In the meantime, please fill out the identification section above, and read the instructions below *carefully*.)

This examination consists of 5 questions on 16 pages (including this one).

Instructions:

- Check to make sure that you have all 16 pages.
- **Read the entire exam before you start.** Not all questions are of equal value, so budget your time accordingly.
- You do not need to add `import` lines or do error checking unless explicitly required to do so.
- You do not need to write comments or docstrings, although they may help us understand your code or enable us to give you partial marks.
- Helper functions are always allowed.
- If you use any space for rough work, indicate clearly what you want marked.

MARKING GUIDE

# 1:	_____/ 5
# 2:	_____/ 6
# 3:	_____/10
# 4:	_____/23
# 5:	_____/10
TOTAL:	_____/54

Good Luck!

CONT'D...

Question 1. [5 MARKS]

Here is a program that runs successfully, though "success" might include raising uncaught exceptions:

```
class StackEmptyException(Exception): pass
class BadOpException(Exception): pass

class Stack(object):
    def __init__(self):
        self.data = []

    def push(self, item):
        self.data.append(item)

    def pop(self):
        if len(self.data) == 0:
            raise StackEmptyException
        return self.data.pop()

def calc(stack, op):
    '''Return the result of doing arithmetic on the top two
    elements of the stack, using operation op, one of '+',
    '-', '*', '/'.
    '''
    a = stack.pop()
    b = stack.pop()
    if op == '+': return a + b
    elif op == '-': return a - b
    elif op == '*': return a * b
    elif op == '/': return a / b
    else:
        raise BadOpException('bad operator ' + op)

if __name__ == '__main__':
    s = Stack()
    s.push(0)
    s.push(1)
    s.push(2)
    s.push(3)
    s.push(4)
    s.push(5)
    s.push(6)
    s.push(7)
    s.push(8)
    s.push(9)

    for op in '+!-*/@':
        try:
            print calc(s, op)
        except StackEmptyException, e:
            print 'empty stack'
        except BadOpException, e:
            print e.message
```

Below, write what is printed when you run the program, including messages about exceptions. If you think the program will print a message about an exception that is raised and not caught, and you can't remember the name of the exception, giving a brief description is good enough.

Question 2. [6 MARKS]

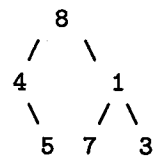
Here is a class to represent a node in a binary tree:

```
class Node(object):
    '''A node in a binary tree.'''

    def __init__(self, value, left=None, right=None):
        '''A new Node with value v and left and right children (which are Nodes themselves).'''

        self.value = value
        self.left = left
        self.right = right
```

Write a *non-recursive* function called `level_order` that takes the root `Node` of a binary tree as a parameter and prints the contents in *level order*: the root's value is printed first, then the root's children from left to right, then the children's children, and so on. For example, if this is the tree:



Then your function, given the node containing 8, would print this: `8 4 1 5 7 3`

Hint: use a queue and put `Node` objects into it. Print the values as you dequeue. Remember that a `Queue` has methods `enqueue`, `dequeue`, and `is_empty`. You can assume that the `Queue` class has been imported.

Continue your answer to the level order question here.

Question 3. [10 MARKS]

The following class can be used in either a binary tree or a doubly-linked list.

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

Write a function called `bst_to_LL` that takes the root `Node` of a binary tree as a parameter and turns it into a doubly-linked list where the nodes are in the same order as when the tree is visited using an inorder traversal. The function should return a tuple containing the front `Node` and end `Node` of the doubly-linked list. (Hint: what information do you get back from the recursive calls on the left and right subtrees?)

You can earn a maximum of 7 marks if you create any new `Node`s. For full marks, modify the pointers in the existing `Node`s in the tree to turn it into a doubly-linked list.

Continue your answer to the bst to linked list question here.

Question 4. [23 MARKS]

Consider this Node class:

```
class Node(object):
    '''A node in a linked list.'''

    def __init__(self, d):
        '''A new Node with data d and no next Node.'''

        self.data = d
        self.next = None
```

Part (a) [3 MARKS] Complete the following function.

```
    def print_all(front):
        '''Print all the items in the circular singly-linked list of Nodes
        pointed to by front.'''
```

Part (b) [3 MARKS]

Write a subclass of `Node` called `DoubleNode` that calls the inherited constructor and then adds `a prev` instance variable.

Part (c) [5 MARKS]

Consider the following code, which (once you write the add function) should create a circular doubly-linked list containing 6, 1, 4, 4, and 3 (in that order) with `front` pointing to the `DoubleNode` containing **6**.

```
if __name__ == '__main__':
    front = None
    front = add(front, 6)
    front = add(front, 1)
    front = add(front, 4)
    front = add(front, 4)
    front = add(front, 3)
    print_all(front)  # This should print 6, 1, 4, 4, 3
```

Complete the following function.

```
def add(front, d):
    '''Add a new DoubleNode at the end of the circular list pointed to by DoubleNode front,
    and return the new front in case front was None.'''
```

Part (d) [5 MARKS] The following function r has an incomplete docstring.

```
def r(t):
    '''t refers to a DoubleNode in a circular doubly-linked list. If
    DoubleNode t contains an odd number, return the following DoubleNode.'''

    if t.next == t and t.data % 2 == 0:
        return None
    elif t.data % 2 != 0:
        return t.next
    else:
        result = t.next
        if t.data % 2 == 0:
            t.next.prev = t.prev
            t.prev.next = t.next

        return result
```

Add sentences to the docstring to describe the remaining cases:

Part (e) [2 MARKS] Complete the following function. You should make use of function r from **Part (d)**.

```
def remove_evens(front):
    '''Remove all the Nodes containing even numbers from the circularly-linked
    list pointed to by front, and return any Node in the remaining list.'''
```

Part (f) [2 MARKS] *(Yeesh, does this question ever end?)*

Write a nose test that tests whether `remove_evens` works on a list containing one odd number and no even numbers.

Part (g) [3 MARKS]

Draw five more interesting doubly-linked lists (using either the shorthand drawing style or the full memory model) that you would use in other tests for `remove_evens`.

-
-
-
-
-

Question 5. [10 MARKS]

This question has you design a set of classes, including instance variables. You will not write any methods.

A *warranty* is a legal document that asserts that a product will operate normally for a period of time (often 1 to 3 years) and if it breaks the company will fix it or replace it.

A company has hired you to write some complaint-tracking software for the company's warranties, products, and customers. When customers purchase a warranty for a product, the warranty is given a unique id number. Each kind of product has an id as well.

Customers phone the company when they have complaints related to a product they have bought. Customers are always asked for their warranty id, which is used to retrieve information about the warranty, the customer, the product under warranty, and any previous complaints the customer has had. Every phone call counts as a separate complaint. The software should keep track of whether a complaint was successfully resolved and whether the customer received a replacement for their product as a result of the call.

Your software will keep track of all the warranties for the company, including the number of times a customer has called regarding a warranty complaint, how many times their complaints were successful, and how many times a product was replaced under a particular warranty.

Write the names of classes you would create to store this information, and underneath each class name, write the instance variables and the type of information that variable refers to. For example, if you had an instance variable for a customer name, you might write this underneath an appropriate class:

```
customer_name: str
```

Choose names and types that lend themselves to readability.

Again, **DO NOT INCLUDE ANY METHODS.** You can assume that the user interface is written separately and just makes use of your classes.

Continue your answer to the complaint-tracking question here.

*[Use the space below for rough work. This page will **not** be marked, unless you **clearly** indicate **the** part of your work that **you** want us to mark.]*

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.
```