PLEASE HAND IN

UNIVERSITY OF TORONTO
Faculty of Arts and Science

April 2014 Examinations

CSC 148H1S
Duration — 3 hours

PLEASE HAND IN

Allowed aids: one 8.5"x11" handwritten aid sheet (both sides)

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

Last Name: ______________________

First Name: ______________________

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.)

This exam consists of 6 questions on 16 pages (including this one). *When you receive the signal to start, please make sure that your copy of the exam is complete.*

Please answer questions in the space provided. You will earn 20% for any question you leave blank or write "I cannot answer this question," on. You may earn substantial part marks for writing down the outline of a solution and indicating which steps are missing.

You must achieve 40% of the marks on this final exam to pass this course.

Write your student number at the bottom of pages 2-16 of this exam.

# 1:	______/12
# 2:	______/12
# 3:	______/12
# 4:	______/ 8
# 5:	______/10
# 6:	______/15
TOTAL:	______/69

Good Luck!

Question 1. [12 MARKS]

A MultiIterator is a list that also maintains a number of indices, each of which represents the current position of an iteration over the elements of the list from first to last. Write a MultiIterator class that has all the methods of list, takes a list in its initializer, and has the following additional methods:

- `new_iterator(self) -> int`
  Starts (another) new iteration over the underlying list, and returns an integer label that is used to refer to this new iteration when calling `reset` and `next`.
- `reset(self,i:int) -> None`
  Restarts (sets to 0) the iteration with label i, if there is one. Throws an exception if there is no iteration i.
- `next(self, i:int) -> object`
  Returns the element at the current position of iteration i, and either increments the current position by 1 or, if the current position is already at the end of the list, resets iteration i to 0. Throws an exception, with a useful message, if there is no iteration i, or if the current position of iteration i is out-of-bounds of the underlying list (since the underlying list can be modified while an iteration is in-progress!).

Here's an example of a MultiIterator in use:

```
>>> m = MultiIterator([0,2])
>>> i = m.new_iterator()
>>> m.next(i)
0
>>> j = m.new_iterator()
>>> m.next(i)
2
>>> m.next(j)
0
>>> m.next(i)
0
```

Write a class MultiIterator that satisfies the above criteria. The only methods that you need to write are `__init__`, `new_iterator`, `next`, and `reset`.

Question 2. [12 MARKS]

Read over the declaration of class Tree and the docstring for function same_leaves below:

```
class Tree:
    """Bare-bones Tree ADT"""

    def __init__(self: 'Tree',
                 value: object =None, children: list =None):
        """Create a node with value and any number of children"""

        self.value = value
        if not children is None:
            self.children = []
        else:
            self.children = children[:] # quick-n-dirty copy of list

def same_leaves(t1: Tree, t2: Tree) -> bool:
    """Return True if and only if the set of t1's leaves' values
    are equal to the set of t2's leaves' values

    precondition: all leaves of t1 have distinct string values
    precondition: all leaves of t2 have distinct string values
    """
```

Part (a) [2 MARKS]

Give an example of two trees t1 and t2 that have the same set of node values but same_leaves(t1, t2) should return False.

Part (b) [2 MARKS]

Give an example of two trees t1 and t2 that have different sets of node values but same_leaves(t1, t2) should return True

Part (c) [8 MARKS]

Write the body of same_leaves.

Question 3. [12 MARKS]

Read over the declaration of class Tree and then answer questions part (a) and (b):

```
class Tree:
    """Bare-bones Tree ADT"""

    def __init__(self: 'Tree',
                 value: object =None, children: list =None):
        """Create a node with value and any number of children"""

        self.value = value
        if not children is None:
            self.children = []
        else:
            self.children = children[:] # quick-n-dirty copy of list

    def __repr__(self: 'Tree') -> str:
        """Return a string representation of Tree self
        >>> t = Tree(1, [Tree(2), Tree(3, [Tree(4)])])
        >>> repr(t)
        'Tree(1, [Tree(2), Tree(3, [Tree(4)])])'
        """
        if len(self.children) > 0:
            return 'Tree({}, {})'.format(repr(self.value), repr(self.children))
        else:
            return 'Tree({})'.format(repr(self.value))
```

Part (a) [8 MARKS]

Implement the method purge_clones:

```
    def purge_clones(self: Tree) -> None:
        """Remove every child that has the same value as its parent
        (from self and self's descendents).

        >>> t = Tree(1, [Tree(2, [Tree(1), Tree(2)]), Tree(1)])
        >>> t.purge_clones()
        >>> repr(t)
        'Tree(1, [Tree(2, [Tree(1)])])'
        """
```

Part (b) [4 MARKS]

Describe four more test cases that would increase your confidence that purge_clones works as specified. Each of your test cases should have the form:

```
t = Tree(...) # where you fill in something meaningful for ...
t.purge_clones()
repr(t) == 'Tree(...)' # again, fill in something meaningful for ...
```

Question 4. [8 MARKS]

Read the declaration of class Tree, then answer the questions that follow.

```
class Tree:
    """Bare-bones Tree ADT"""

    def __init__(self: 'Tree',
                 value: object =None, children: list =None):
        """Create a node with value and any number of children"""

        self.value = value
        if not children:
            self.children = []
        else:
            self.children = children[:] # quick-n-dirty copy of list

    def __repr__(self: 'Tree') -> str:
        """Return a string representation of Tree self
        >>> t = Tree(1, [Tree(2), Tree(3, [Tree(4)])])
        >>> repr(t)
        'Tree(1, [Tree(2), Tree(3, [Tree(4)])])'
        """
        if len(self.children) > 0:
            return 'Tree({}, {})'.format(repr(self.value), repr(self.children))
        else:
            return 'Tree({})'.format(repr(self.value))
```

Implement the method deepen.

```
def deepen(t:Tree) -> None:
    """Modify t, doubling its depth by adding a node just below every node in t.
    If u is a node in the original t, then in the new tree u will have as its
    only child a new node v, with u.value == v.value, and v's children will u's
    former children.

    >>> t = Tree(1, [Tree(2), Tree(3)])
    >>> deepen(t)
    >>> repr(t)
    'Tree(1, [Tree(1, [Tree(2, [Tree(2)]), Tree(3, [Tree(3)])])])'
    """
```

Question 5. [10 MARKS]

Read over the declarations of classes BTNode and LListNode. Then answer the questions below and on the next page.

```
class BTNode:
    """Binary Tree node."""

    def __init__(self: 'BTNode', data: object,
                 left: 'BTNode'=None, right: 'BTNode'=None) -> None:
        """Create BT node with data and children left and right."""
        self.data, self.left, self.right = data, left, right


class LListNode:
    """Node to be used in linked list"""

    def __init__(self: 'LListNode', value: object,
                 nxt: 'LListNode' =None) -> None:
        """Create a new LListNode containing value and with next node nxt

        nxt --- None if and only if we are on the last node
        value --- always a Python object, there are no empty nodes
        """
        self.value, self.nxt = value, nxt

    def __repr__(self: 'LListNode') -> str:
        """Represent LListNode self as a string"""
        return 'LListNode({}, {})'.format(repr(self.value), repr(self.nxt))
```

Part (a) [2 MARKS]

Implement the function contains:

```
def contains(node: LListNode, value: object) -> bool:
    """Return whether some LListNode in the linked list starting at node
    contains value

    >>> lnk = LListNode(1, LListNode(3, LListNode(5)))
    >>> contains(lnk, 3)
    True
    >>> contains(lnk, 7)
    False
    """
```

Part (b) [8 MARKS]

Implement the function **append_unique_data**:

```
def append_unique_data(bt: BTNode, n: LListNode) -> LListNode:
    """Return node that starts the linked list formed by appending
    unique data (no duplicates) from the tree rooted at t to
    the linked list that starts at n.

    >>> t = BTNode(5, BTNode(3), BTNode(4, BTNode(3)))
    >>> lnk = append_unique_data(t, None)
    >>> repr(lnk)
    'LListNode(4, LListNode(3, LListNode(5, None)))'
    """
```

Question 6. [15 MARKS]

Read the text descriptions of the four algorithms below. For each algorithm explain which of the following complexity classes best describe the worst-case time performance for a list of n elements:

$\mathcal{O}(1)$ $\mathcal{O}(\lg n)$ $\mathcal{O}(n)$ $\mathcal{O}(n \lg n)$ $\mathcal{O}(n^2)$

Explain why the text description leads you to choose the complexity class you did. Also explain what behaviour an implementation of each algorithm should exhibit when run on a computer on a list of size 2n versus a list of size n.

You may assume that == and < are $\mathcal{O}(1)$ operations for floats.

Part (a) [3 MARKS]

Mergesort a list of n floats, then find and return a maximum element.

Part (b) [3 MARKS]

Given a python list L of floats, and an integer $0 \leq i < \text{len}(L) - 1$, return True if the ith element of L is less than the $(i + 1)$th element of L.

Part (c) [6 MARKS]

Given a length-n list of length-n lists of floats, for example, if n = 3:

```
[[1.1, 1.1, 2.0], [1.2, 9.1, 2.0], [1.1, 1.1, 2.0]]
```

in the most straight-forward way check:

1. whether the the first list is equivalent to at least one of the other lists (same elements in the same order).

2. whether there is at least a pair of equivalent lists.

Part (d) [3 MARKS]

The following function f when run on a list of floats.

```
def f(L:list) -> float:
    def f2(n:int) -> float
        if n == 0:
            return L[n]
        else:
           return L[n] * f2(n/2)
  return f2(len(l) - 1)
```

This page has been left intentionally (mostly) blank, in case you need

This page has been left intentionally (mostly) blank, in case you need

This page has been left intentionally (mostly) blank, in case you need space.

Total Marks = 69