PLEASE HAND IN

PLEASE HAND IN

UNIVERSITY OF TORONTO
Faculty of Arts and Science

December 2013 Examinations

CSC 148H1F
Duration — 3 hours

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.

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

# 1:	______/15
# 2:	______/12
# 3:	______/16
# 4:	______/12
# 5:	______/12
# 6:	______/ 8
TOTAL:	______/75

Good Luck!

Question 1. [15 MARKS]

The special method list._add_(L) concatenates lists:

```
>>> list([1, 2, 3]).__add__([4, 5, 6])
[1, 2, 3, 4, 5, 6]
```

Similarly, the special method int._add_(n) adds integers:

```
>>> int(5).__add__(3)
8
```

Recall that "adding" can be extended over an entire list of one or more lists by using reduce to concatenate all the lists:

```
>>> from functools import reduce
>>> reduce(list.__add__, [[1, 2, 3]])
[1, 2, 3]
>>> reduce(list.__add__, [[1, 2, 3], [4, 5, 6], [7, 8]])
[1, 2, 3, 4, 5, 6, 7, 8]
```

Suppose the following code has been evaluated:

```
from functools import reduce

def ca(L: list) -> list:
    """No meaningful docstring!!"""
    return reduce(list.__add__, [ca(x) if isinstance(x, list) else [x] for x in L])
```

For each part below, show step-by-step what the expression produces. By "step-by-step" I mean you need to show how intermediate values are produced except if the intermediate value is a recursive call to ca where you have already evaluated an equivalent case and can just plug the result in.

Part (a) [3 MARKS]

```
ca([1, 2, 3])
```

Part (b) [3 MARKS]

```
ca([1, [2, 3], 4])
```

Part (c) [3 MARKS]

```
ca([1, 2, [3, [4, 5], 6], 7])
```

Part (d) [3 MARKS]

```
ca([1, 2, [3, [4, 5, [5.1, 5.2]], 6], 7])
```

Part (e) [3 MARKS]

```
reduce(int.__add__, ca([1, 2, [3, [4, 5], 6], 7]))
```

Question 2. [12 MARKS]

Read the declaration of class BTNode, and the docstring for function filter_nodes(n, f).

```
class BTNode:
    """Node in a binary tree"""

    def __init__(self: 'BTNode', value: object, left: 'BTNode',
                 right: 'BTNode') -> None:
        """
        Create a new BTNode with value and (possibly)
        children left and right.
        """
        self.value, self.left, self.right = value, left, right


def filter_nodes(n: BTNode, f: 'boolean function') -> list:
    """
    Return inorder list of values of tree rooted at n
    that satisfy boolean function f.

    >>> def h(n: int) -> bool:
    ...     return n % 5 == 0 # is n a multiple of 5?
    ...
    >>> filter_nodes(None, h)
    []
    >>> filter_nodes(BTNode(7, BTNode(0, None,None), BTNode(15, None, None)), h)
    [0, 15]
    >>> def g(n: int) -> bool:
    ...     return n % 7 == 0 # is n a multiple of 7?
    ...
    >>> filter_nodes(BTNode(7, BTNode(0, None,None), BTNode(15, None, None)), g)
    [0, 7]
    """
```

Part (a) [8 MARKS]

Write the body for the function filter_nodes(n, f).

Part (b) [4 MARKS]

Read over, and write the body for, the function largest_filtered_value(n, f). Notice that you may assume that the tree rooted at n is a binary search tree (BST).

```
def largest_filtered_value(n: BTNode, f: 'bool function') -> object:
    """
    Return largest value in tree rooted at n that satisfies boolean function f, or None
    if there are no values that satisfy f.
    Assume the tree rooted at n is a binary search tree (BST).

    >>> def g(n: int) -> bool:
    ...     return n % 7 == 0 # is n a multiple of 7?
    ...
    >>> largest_filtered_value(BTNode(7, BTNode(0, None,None), BTNode(15, None, None)), g)
    7
    """
```

Question 3. [16 MARKS]

Read over class TreeNode and the docstring for function cycle.

```
class TreeNode:
    """Node in a tree of arbitrary arity (branching factor)"""

    def __init__(self: 'TreeNode', value: object, children: list) -> None:
        """Create a node in a tree"""
        self.value, self.children = value, children

    def __repr__(self: 'TreeNode') -> str:
        """represent TreeNode as a string"""
        return 'TreeNode(' + repr(self.value) + ', ' + repr(self.children) + ')'

def cycle(n: TreeNode) -> TreeNode:
    """
    Produce root of a new tree with same node values as the one rooted at n,
    but all children are cycled [c1, c2, ..., cn] -> [c2, ..., cn, c1].

    >>> cycle(None) is None
    True
    >>> cycle(TreeNode(5, [TreeNode(4, []), TreeNode(3, []), TreeNode(2, [])]))
    TreeNode(5, [TreeNode(3, []), TreeNode(2, []), TreeNode(4, [])])
    >>> cycle(TreeNode(5, [TreeNode(4, []), TreeNode(3, [TreeNode(2, []), TreeNode(1, [])])]))
    TreeNode(5, [TreeNode(3, [TreeNode(1, []), TreeNode(2, [])]), TreeNode(4, [])])
    """
```

Part (a) [8 MARKS]

Write the body of cycle(n).

Part (b) [8 MARKS]

Read the docstring for function equivalent, and write its body.

```
def equivalent(n1: TreeNode, n2: TreeNode) -> bool:
    """
    Does the tree rooted at n1 have == values at corresponding nodes to the
    tree rooted at n2?

    >>> equivalent(None, None)
    True
    >>> n1 = TreeNode(5, [TreeNode(3, []), TreeNode(2, []), TreeNode(4, [])])
    >>> n2 = TreeNode(5, [TreeNode(3, []), TreeNode(2, []), TreeNode(4, [])])
    >>> n1 == n2
    False
    >>> equivalent(n1, n2)
    True
    """
```

Question 4. [12 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 to confirm your choice.

Part (a) [3 MARKS]

linear search(L: list, x: object) Compare each element of L to value x and return the index of the first match, or else len(L) if there is no match.

Part (b) [3 MARKS]

binarysearch(L: list, x: object) If L has 1 element, return 0 if that element is x, otherwise raise Exception('not found'). If non-decreasing list L has 2 or more elements, compare element L[n/2] to x. If x is less than L[n/2], binarysearch list L[:n/2], otherwise binarysearch list L[n/2:].

Part (c) [3 MARKS]

selectionsort(L: list) For each index i: 0 .. len(L)-1 of the list, swap the minimum element of L[i:] with the element currently at position i.

Part (d) [3 MARKS]

mergesort(L: list) If L has fewer than 2 elements, we're done. Otherwise, split L in half, mergesort each half, then merge each half (merge is known to have linear complexity).

Question 5. [12 MARKS]

Read over the class declaration of Point:

```
# module math may be handy...
import math


class Point:
    """represent a point in space"""

    def __init__(self: 'Point', coords: list) -> None:
        """create a new point"""
        self.coords = [float(x) for x in coords]
```

Write class declarations for TwoDPoint and ThreeDPoint that subclass Point. These new classes must be able to create corresponding points:

```
>>> p2a = TwoDPoint([3,4])
>>> p3 = ThreeDPoint([0,3,4])
```

However, both p2 = TwoDPoint([0, 3, 4]) and p3 = ThreeDPoint([3, 4]) should raise Exceptions, since there is a mismatch between the length of coords and the dimension. Any such mismatch should raise an Exception.

Also, both TwoDPoint and ThreeDPoint should have method distance() that is able to report distance from the origin, using the formula:

$$\texttt{p2.distance()} = \sqrt{\texttt{p2.coord[0]}^2 + \texttt{p2.coord[1]}^2} \quad \text{(for TwoDPoint p2)}$$

$$\texttt{p3.distance()} = \sqrt{\texttt{p3.coord[0]}^2 + \texttt{p3.coord[1]}^2 + \texttt{p3.coord[2]}^2} \quad \text{(for ThreeDPoint p3)}$$

Finally, both TwoDPoint and ThreeDPoint should have a method _equal_(other) that returns True if other is a Point whose coordinates match those of the current Point instance.

You will receive the most credit for implementations that place as much common code as possible into superclass Point itself, although you are not allowed to modify its _init_ method.

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

Question 6. [8 MARKS]

Read the docstring for function index_list below, then write its body. You may not create new lists in your implementation — you must change TL itself.

```
def index_list(TL: list, i: int) -> int:
    """
    Replace TL's string elements, from left to right, with consecutive
    integers, starting at i.  Return 1 + the largest integer you used in the
    replacements, or i itself if no replacments were made.

    TL is a TreeList.  A TreeList is either None or a 3-element list where:
    Element 0 is a string
    Element 1 and Element 2 are TreeLists

    >>> TL = None
    >>> index_list(TL, 2)
    2
    >>> TL is None
    True
    >>> TL = ["five", None, None]
    >>> index_list(TL, 3)
    4
    >>> TL
    [3, None, None]
    >>> TL = ["five", ["seven", None, None], ["three", None, None]]
    >>> index_list(TL, 4)
    7
    >>> TL
    [4, [5, None, None], [6, None, None]]
    """
```

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

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

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

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

Total Marks = 75