CSC 148H5 S 2016 Test 2
Duration — 50 minutes
Aids allowed: none

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

Last Name: _____ **First Name:** _____

☐ Lecture Section: L0102 Instructor: Dan Zingaro (11:00-12:00)
☐ Lecture Section: L0101 Instructor: Tiffany Tong (10:00-11:00)
☐ Lecture Section: L0103 Instructor: Sadia Sharmin (9:00-10:00)

---

*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 4 questions on 8 pages (including this page). *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: _____/ 4
2: _____/ 4
3: _____/ 6
4: _____/ 6

TOTAL: _____/20

Question 1. [4 MARKS]

The **preorder** traversal for a given binary tree t is 8, 2, 1, 9. The **inorder** traversal for the **same** binary tree t is 2, 1, 8, 9.

Draw the binary tree t corresponding to the above preorder and inorder traversals.

Question 2. [4 MARKS]

Explain in plain English the purpose of the following `mystery` code. (Remember: this means that we want the overall purpose of the code, **not** a line-by-line description of what the code does.) `t` is a binary tree in nodes-and-references form.

```
def mystery(t):
  if not t:
    return True
  if not t.left and not t.right:
    return True
  if t.left and t.right:
    return False
  return mystery(t.left) and mystery(t.right)
```

Question 3. [6 MARKS]

Here is the way linked list nodes will be structured for this question. The empty linked list will be represented as `None`.

```
class Node:
  def __init__(self, value):
    self.value = value
    self.next = None
```

Write a function that takes `lst1` (the head of a linked list) and `lst2` (the head of a different linked list), and merges the two as follows:

If `lst1` is `1 -> 2 -> 3` (i.e. a list with three elements: 1 followed by 2 followed by 3)
and `lst2` is `4 -> 5 -> 6`, then
`merge(lst1, lst2)` makes
`lst1` be `1 -> 4 -> 2 -> 5 -> 3 -> 6`
`lst2` stays unchanged.

If `lst1` is `1 -> 2 -> 3 -> 4 -> 5` and `lst2` is `6 -> 7`, then
`merge(lst1, lst2)` makes
`lst1` be `1 -> 6 -> 2 -> 7 -> 3 -> 4 -> 5`
`lst2` stays unchanged.

```
def merge(lst1: Node, lst2: Node) -> None:
  '''
  Given two linked lists, merge elements of lst2 into lst1,
  such that lst1 has all of lst2's elements inserted into it
  in the way shown above.
  lst2 is NOT changed.

  # HINT: You may create new nodes within this function (i.e. x = Node(value)) if needed.

  Precondition: The number of elements in lst2
  is less than or equal to the number of elements in lst1.
  '''
```

Question 4. [6 MARKS]

This question uses the following `Tree` class:

```
class Tree:
  """Tree ADT; nodes may have any number of children"""

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

    self.value = value
    if not children:
        self.children = []
    else:
        self.children = children[:]
```

Write the following function so that it satisfies its docstring. Your code *must be recursive*.

```
def nonleaf_count(t: Tree) -> int:
  """Return number of internal (non-leaf) nodes in t

  >>> tn2 = Tree(2, [Tree(4), Tree(4.5)])
  >>> tn3 = Tree(3, [Tree(6)])
  >>> tn1 = Tree(1, [tn2, tn3])
  >>> nonleaf_count(tn1)
  3
  """
```

[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.]

[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.]

Last Name: _____ **First Name:** _____

Short Python function/method descriptions:

```
__builtins__:
  input([prompt]) -> str
    Read a string from standard input; return that string with no newline.
    The prompt string, if given, is printed without a trailing newline before reading.
  max(a, b, c, ...) -> value
    With two or more arguments, return the largest argument.
  min(a, b, c, ...) -> value
    With two or more arguments, return the smallest argument.
  print(value, ..., sep=' ', end='\n') -> NoneType
    Prints the values. Optional keyword arguments:
      sep:  string inserted between values, default a space.
      end:  string appended after the last value, default a newline.
int:
  int(x) -> int
    Convert a string or number to an integer, if possible.
    A floating point argument will be truncated towards zero.
str:
  S.count(sub[, start[, end]]) -> int
    Return the number of non-overlapping occurrences of substring sub in string S[start:end].
    Optional arguments start and end are interpreted as in slice notation.
  S.find(sub[,i]) -> int
    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.isalpha() -> bool
    Return True if and only if all characters in S are alphabetic
    and there is at least one character in S.
  S.isdigit() -> bool
    Return True if and only if all characters in S are digits
    and there is at least one character in S.
  S.islower() -> bool
    Return True if and only if all cased characters in S are lowercase and there is at least
    one cased character in S.
  S.isupper() -> bool
    Return True if and only if all cased characters in S are uppercase and there is at least
    one cased character in S.
  S.lower() -> str
    Return a copy of S converted to lowercase.
  S.replace(old, new) -> str
    Return a copy of string S with all occurrences of the string old replaced with the string new.
  S.split([sep]) -> list of str
    Return a list of the words in S, using string sep as the separator and any whitespace string
    if sep is not specified.
  S.startswith(prefix) -> bool
    Return True if S starts with the specified prefix and False otherwise.
  S.strip() -> str
    Return a copy of S with leading and trailing whitespace removed.
  S.upper() -> str
    Return a copy of S converted to uppercase.
```