CSC 148H5 S 2014 Test 2
Duration — 50 minutes
Aids allowed: none

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

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

Lecture Section: L0101 Instructor: Dan

---

*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. No error checking is required: assume all user input and all argument values are valid.
If you use any space for rough work, indicate clearly what you want marked.

1: _____/ 3

2: _____/ 6

3: _____/ 6

4: _____/ 5

TOTAL: _____/20

Question 1. [3 MARKS]

Here is a claim: the list of nodes visited in an inorder traversal of a BST is always a sorted list.

Is this claim true? If yes, explain why. If not, give a BST whose inorder traversal is not a sorted list.

Question 2. [6 MARKS]

A **plus expression** (plex) is a string whose allowed characters are `'0'`, `'1'`, `'('`, `')'`, and `'+'`. Here are the rules for determining whether a string is a plex:

- `'0'` and `'1'` are plexes
- If `p` is a plex, then so is `'+' + p`
- If `p` and `q` are plexes, then so is `'(' + p + '+' + q + ')'`

For example, `'(0+1)'` and `'++(1+1)'` are plexes.

Write the following function. You may use helper functions as you wish.

```
def is_plex(s: str) -> bool:
  '''Return True iff s is a plex according to the above rules.
  '''
```

Question 3. [6 MARKS]

Here is a node in a binary tree (this code is from your lab).

```
class BTNode:
  '''A node in a binary tree.'''

  def __init__(self: 'BTNode', item: object,
               left: 'BTNode'=None, right: 'BTNode'=None) -> None:
    '''Initialize this node.'''
    self.item, self.left, self.right = item, left, right
```

Write the following function to remove all leaves from the binary tree rooted at `t`. Note that as you remove leaves, you may cause other nodes to become leaves. Do **not** remove those new leaves; only remove the nodes that were leaves originally.

```
def remove_leaves(t: 'BTNode') -> 'BTNode':
  '''Remove the leaves of the binary tree rooted at t.
  Return the root of the tree; return None if the tree
  becomes empty.
  '''
```

Question 4. [5 MARKS]

Write the `append` method of the `LinkedList` class.

We already solved it recursively in lecture. You **must** therefore write an iterative version here; you are not permitted to use recursion.

```
class LinkedList:
  '''Linked list class'''

  def __init__(self: 'LinkedList', head: object=None,
               rest: 'LinkedList'=None) -> None:
    '''Create a new LinkedList.
    head - first element of linked list
    rest - linked list of remaining elements
    The empty linked list has head None
    '''
    # a linked list is empty if and only if it has no head
    self.empty = head is None
    if not self.empty:
      self.head = head
      if rest is None:
        self.rest = LinkedList()
      else:
        self.rest = rest

  def prepend(self: 'LinkedList', newhead: object) -> None:
    '''Add new head to front of LinkedList'''
    if not self.empty:
      temp = LinkedList(self.head, self.rest)
    else:
      temp = LinkedList()
    self.head = newhead
    self.rest = temp
    self.empty = False

  def append(self: 'LinkedList', newlast: object) -> None:
    '''Add newlast to end of LinkedList'''
```

[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.
```