

Duration: **50 minutes**
Aids Allowed: **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 read the instructions below carefully.

This term test consists of 4 questions on 10 pages (including this one), printed on both sides of the paper. *When you receive the signal to start, please make sure that your copy of the test is complete, fill in the identification section above, and write your name on the back of the last page.*

Answer each question directly on the test paper, in the space provided, and use one of the “blank” pages for rough work. If you need more space for one of your solutions, use a “blank” page and *indicate clearly the part of your work that should be marked.*

In your answers, you may use any of the Python builtin functions and standard modules listed on the accompanying Python reference sheet. You must write your own code for everything else.

Comments and docstrings are *not required*, except where indicated, although they may help us grade your answers. Also, they may be worth part marks if you cannot write a complete answer.

If you are unable to answer a question (or part of a question), remember that you will get 10% of the marks for any solution that you leave *entirely blank* (or where you cross off everything you wrote to make it clear that it should not be marked).

MARKING GUIDE

1: _____/12

2: _____/10

3: _____/10

4: _____/ 8

TOTAL: _____/40

Good Luck!

Question 1. [12 MARKS]

Implement each method in class **Stack** below. Use instances of class **LLNode** to store the items in a linked list. *Do not use any builtin Python container in your solution!* (No `list`, `set`, `dict`, `file`, or even `str`.) **Your code will be marked on its style and design in addition to its correctness.**

```
class LLNode:
    """A node in a linked list."""
    def __init__(self, item, link=None):
        self.item = item
        self.link = link

class EmptyStackError(Exception):
    pass

class Stack:
    """Implementation of the Stack ADT."""

    def __init__(self):
        """Initialize this stack to be empty."""

    def push(self, item):
        """Add item to the top of this stack."""
```

[...continued on the next page...]

Question 1. (CONTINUED)

```
# ...class Stack, continued...
```

```
def pop(self):  
    """Remove the top item from this stack and return that item.  
    Raise EmptyStackError if this stack is empty."""
```

```
def is_empty(self):  
    """Return True iff this stack is empty."""
```

Question 2. [10 MARKS]

Consider the following node class (for linked lists).

```
class LLNode:
    """A node in a linked list."""
    def __init__(self, item, link=None):
        self.item = item
        self.link = link
```

Write the body of the function below to satisfy its docstring. This will require a bit of thinking, so it would be a good idea to provide some internal comments to describe what you are trying to do.

Your code will be marked on its style and design in addition to its correctness.

```
def split_even(head):
    """(LLNode) -> NoneType
    Modify the linked list starting at node head so that every even integer in
    the list becomes two separate values, each equal to half of the original
    (this will require adding new nodes to the list). Precondition: every item
    in the linked list is an integer. For example, if head refers to a linked
    list as follows:
        head: 1 -> 2 -> 3 -> 4
    then after the call split_even(head), the linked list should look like this
    (the 2 has been split into two 1's and the 4 into two 2's):
        head: 1 -> 1 -> 1 -> 3 -> 2 -> 2
    (Note: this function does NOT return anything.)
    """
```

*Use the space on this “blank” page for scratch work, or for any solution that did not fit elsewhere.
Clearly label each such solution with the appropriate question and part number.*

Question 3. [10 MARKS]

Consider the following node class (for binary trees).

```
class BTreeNode:
    """A node in a binary tree."""
    def __init__(self, item, left=None, right=None):
        self.item = item
        self.left = left
        self.right = right
```

Part (a) [4 MARKS]

Write the body of the method below to satisfy its docstring. (*Note: This is a **method** inside class BTreeNode.*)
Your code will be marked on its style and design in addition to its correctness.

```
def reverse(self):
    """(BTreeNode) -> NoneType
    Replace every item in the subtree rooted at this node with its reverse.
    Precondition: every item in the subtree rooted at this node is a
    string. For example, if self is a reference to the root of the tree
    pictured on the left below, then after this method finishes executing,
    the tree's items have changed as pictured on the right.
```

```
before reverse()
    "this"
   /   \
 "test"  "is"
      /
    "fun"
```

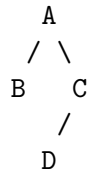
```
after reverse()
    "siht"
   /   \
 "tset"  "si"
      /
    "nuf"
```

Reminder: if *s* refers to a string, then *s[::-1]* is the reverse of *s*.
 """

Question 3. (CONTINUED)**Part (b)** [6 MARKS]

Write the body of the function below to satisfy its docstring. (*Note:* This is a **function** *outside* class `BTNode`.) **Your code will be marked on its style and design, in addition to its correctness.**

```
def leaf_list(root):  
    """(BTNode) -> [object]  
    Return a list of every item stored in a leaf of the binary tree  
    rooted at root. For example, if root refers to the root of the  
    binary tree pictured on the right, then the return value should  
    be [B, D] (or [D, B] -- order does not matter).  
    """
```

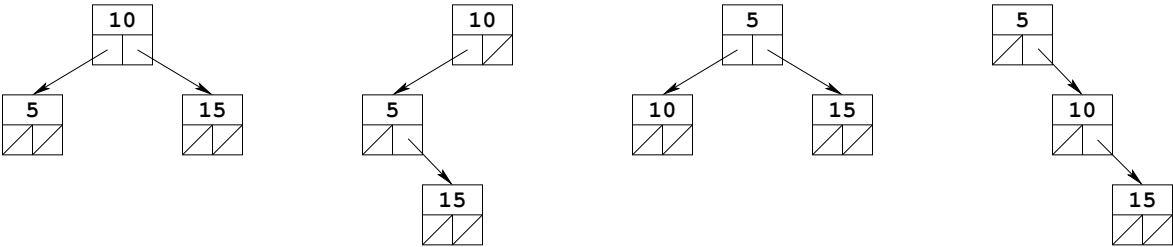


```
          A  
        / \  
       B  C  
      /   \  
     D    
```

Question 4. [8 MARKS]

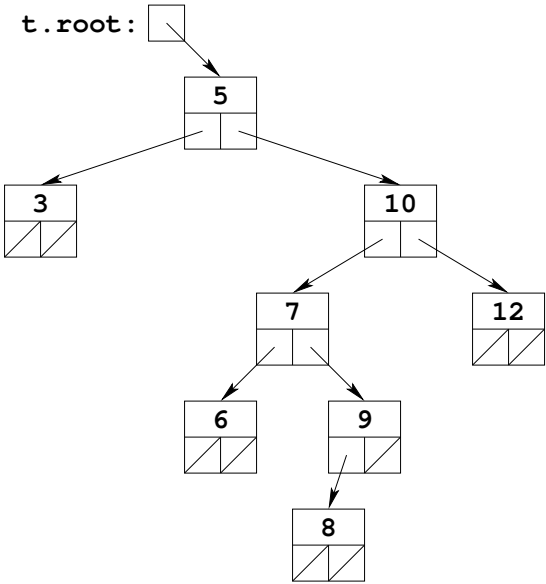
Part (a) [4 MARKS]

Some of the trees pictured below are Binary Search Trees; others are not. Circle every tree that is a Binary Search Tree.



Part (b) [4 MARKS]

Let `t` be the binary search tree pictured below. Use the rest of the space on this page to draw a picture of the tree after performing the operation `t.remove(10)`. (For your reference, we provide parts of the BST code from class on the next page.)



Question 4. (CONTINUED)**Part (b)** (CONTINUED)

```
class BST:
    # ...other methods omitted...
    def remove(self, item):
        self.root = _remove(self.root, item)

def _remove(root, item):
    if root is None:
        pass
    elif item < root.item:
        root.left = _remove(root.left, item)
    elif item > root.item:
        root.right = _remove(root.right, item)
    else: # item == root.item
        if root.left is None or root.right is None:
            root = root.left or root.right
        else:
            root.item, root.left = _remove_max(root.left)
    return root

def _remove_max(root):
    if root.right is None:
        return root.item, root.left
    else:
        largest, root.right = _remove_max(root.right)
        return largest, root
```

On this page, please write nothing except your name.

Last (Family) Name(s): _____

First (Given) Name(s): _____

Total Marks = 40