PLEASE HAND IN

PLEASE HAND IN

UNIVERSITY OF TORONTO
Faculty of Arts and Science

Term test #2

CSC 148H1, Section L0101
Duration — 50 minutes

| | |
|---|---|
| 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 test consists of 4 questions on 10 pages (including this one). *When you receive the signal to start, please make sure that your copy of the test 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. We think we have provided a lot of space for your work, but please do not feel you need to fill all available space.

*Good Luck!*

## Question 1. [8 MARKS]

Read the docstring below for method **remove_first_double**. You may assume that classes **LinkedListNode** and **LinkedList** from the API have been imported. Implement **remove_first_double**. **Note**: The only **LinkedList** and **LinkedListNode** methods provided are those in the API.

```
def remove_first_double(self):
    """
    Remove second of two adjacent nodes with duplicate values.
    If there is no such node, leave self as is.  No need
    to deal with subsequent adjacent duplicate values.

    @param LinkedList self: this linked list
    @rtype: None

    >>> list_ = LinkedList()
    >>> list_.append(3)
    >>> list_.append(2)
    >>> list_.append(2)
    >>> list_.append(3)
    >>> list_.append(3)
    >>> print(list_.front)
    3 -> 2 -> 2 -> 3 -> 3 ->|
    >>> list_.remove_first_double()
    >>> print(list_.front)
    3 -> 2 -> 3 -> 3 ->|
    """
```

This page is left (mainly) blank for things that don't fit elsewhere.

## Question 2. [8 marks]

Read the docstring for function **contains_satisfier** below, and then implement it.

```
def contains_satisfier(list_, predicate):
    """
    Return whether possibly-nested list_ contains a non-list element
    that satisfies (returns True for) predicate.

    @param list list_: list to check for predicate satisfiers
    @param (object)->bool predicate: boolean function
    @rtype: bool

    >>> list_ = [5, [6, [7, 8]], 3]
    >>> def p(n): return n > 7
    >>> contains_satisfier(list_, p)
    True
    >>> def p(n): return n > 10
    >>> contains_satisfier(list_, p)
    False
    """
```

This page is left (mainly) blank for things that don't fit elsewhere.

CONT'D...

## Question 3. [8 MARKS]

Read the docstring below for function **count_odd_above**, as well as the API for class **Tree**. You may assume that class **Tree** has been imported. Implement function **count_odd_above**. **Hint**: The depth of a node is 1 less than the depth of its children.

```
def count_odd_above(t, n):
    """
    Return the number of nodes with depth less than n that have odd values.

    Assume t's nodes have integer values.

    @param Tree t: tree to list values from
    @param int n: depth above which to list values
    @rtype: int

    >>> t1 = Tree(4)
    >>> t2 = Tree(3)
    >>> t3 = Tree(5, [t1, t2])
    >>> count_odd_above(t3, 1)
    1
    """
```

This page is left (mainly) blank for things that don't fit elsewhere.

cont'd...

## Question 4. [6 MARKS]

Draw a diagram of a binary search tree of minimum height containing the following integer values:

7, 1, 9, 13, 5, 3, 15, 11

This page is left (mainly) blank for things that don't fit elsewhere.

This page is left (mainly) blank for things that don't fit elsewhere.

|  |  |
|---|---|
| # 1: | ______/ 8 |
| # 2: | ______/ 8 |
| # 3: | ______/ 8 |
| # 4: | ______/ 6 |
| TOTAL: | ______/30 |