UNIVERSITY OF TORONTO
Faculty of Arts & Science

Fall 2024 Mock Midterm

CSC 148 H1S

Duration: 70 min.

Aids Allowed: None

*Do **not** turn this page until you have received the signal to start.*
In the meantime, fill in your name, student number, and UTORid, and carefully read all the information below.

First (Given) Name(s):

Last (Family) Name(s):

10-Digit Student Number:

UTORid (e.g., `pitfra12`):

This test consists of 4 questions on 9 pages (including this one), printed on both sides of the paper.

Make sure to read each question carefully to ensure you are answering the question being asked.

Please write legibly, and show your work where appropriate. We can only give you credit for what you write on the page.

You may write in either pen or pencil.

Good luck! We want you to do well!

Marking Guide

Nº 1: _____/ 6

Nº 2: _____/ 7

Nº 3: _____/ 6

Nº 4: _____/ 7

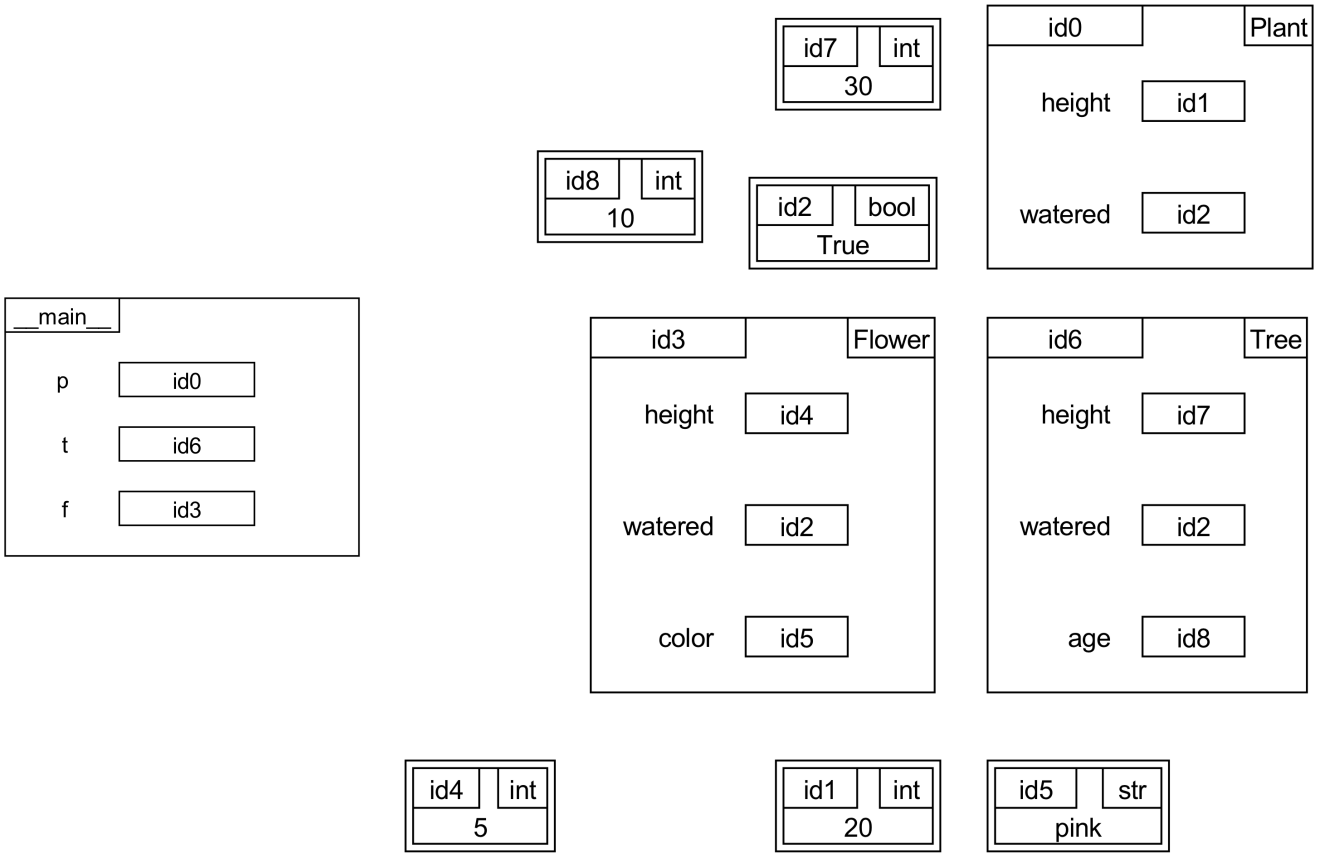
TOTAL: _____/26

STUDENTS MUST HAND IN ALL EXAMINATION MATERIALS AT THE END

Question 1. [6 MARKS]

Consider the following client code which produces the memory model shown below.
On the next page, write code for the three classes required by the client code. Do not write any docstrings, but you must include type annotations. There must be an inheritance relationship in your code, and the parent class does not have to be abstract.

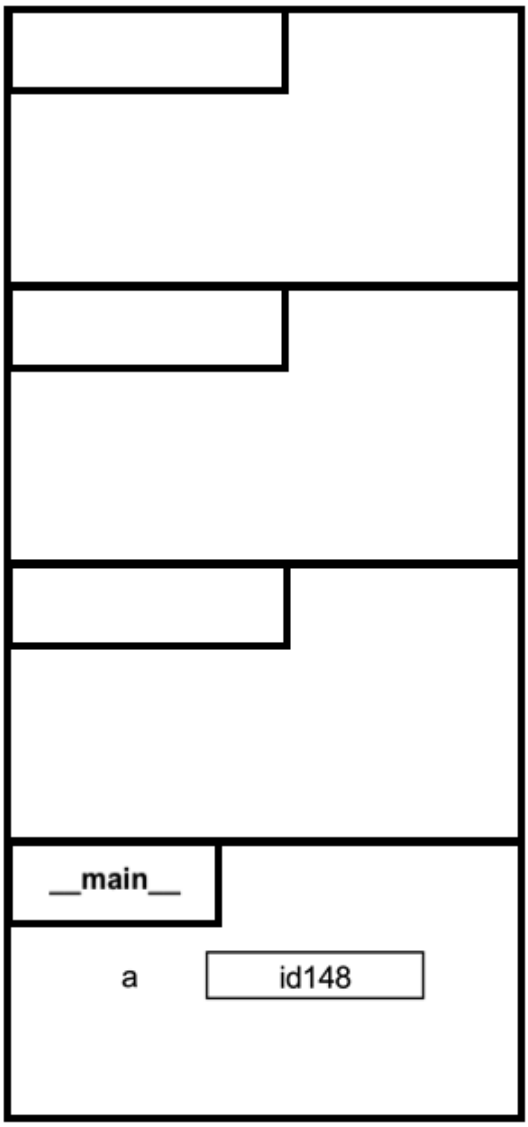
```
if __name__ == '__main__':

    p = Plant(20, True)

    t = Tree(30, 10)

    f = Flower(5, 'pink', True)
```



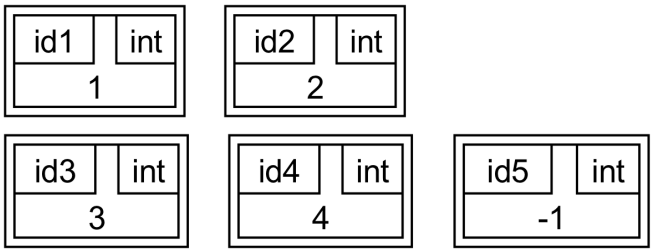
In the space below, write your classes. You do not need to include docstrings.

Question 2. [7 MARKS]

Part (a) [5 MARKS] Consider the following code. Draw the contents of the call stack **immediately before** `have_fun` returns. Draw any objects as needed in the space below to complete the memory model. If any stack frames were popped before `have_fun` returns, cross them out to indicate that they were popped. We have executed the first line of code in the stack frame for `__main__` to get you started, but you will have to draw the object with `id148`.




```
def d_fun(lst: list[int], v: int) -> None:
    z = v + 1
    lst.append(z)

def z_fun(d: list[int]) -> int:
    d_fun(d, 3)
    return d[-1]

def have_fun(lst: list[int]) -> int:
    rslt = z_fun(lst)
    return rslt

if __name__ == "__main__":
    a = [1, 2, 3]
    c = a[:]
    print(have_fun(a))
    print(c)
```



Part (b) [2 MARKS] In the space below, write what is printed when the code is executed.

Question 3. [6 MARKS]

Part (a) [1 MARK] Consider the `MyList` class below.
What is the name of the relationship between the `MyList` class and the `Queue` class? _____

Part (b) [5 MARKS] Implement the `remove` method according to its docstring. Do not make any assumptions about the behaviour of the `Queue` class other than that it has the API specified on the aid sheet. You may not define any other methods or functions. You may not use any builtin python lists, sets, tuples, or dictionaries — use `Stack` or `Queue` objects instead. Comments are not required but are encouraged.

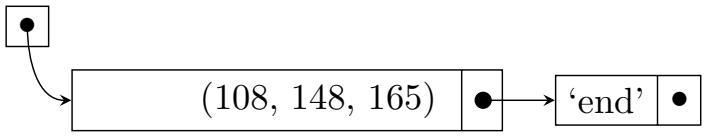
```
class MyList:
    """A list implemented using a Queue.
       The item at index 0 in this MyList is stored at the front of _queue"""
    _queue: Queue

    def __init__(self) -> None:
        """ Initialize <self> to represent an empty list. """
        self._queue = Queue()

    def remove(self, item: Any) -> None:
        """Remove the first occurence of <item> from this list.
        Raise a ValueError if <item> is not in this list."""
```

Question 4. [7 MARKS]

Consider the following docstring description of a new `LinkedList` method:

```
def expand_all_tuples(self) -> None:
    """Mutate <self> so that every tuple has been "expanded".
    Expanding a tuple of length n results in n-1 new nodes, each containing
    one item from the original tuple, in the original order. The original node
    contains the first item from the original tuple.

    Preconditions:
    - each tuple in self has length > 0

    >>> lst = LinkedList([('hi', 'there'), 42])
    >>> print(lst)  # the original list
    ('hi', 'there') -> 42
    >>> lst.expand_all_tuples()
    >>> print(lst)  # the mutated list after calling the method
    hi -> there -> 42
    """
```

Part (a) [2 MARKS] Modify the simplified `LinkedList` object diagram below to show how this `LinkedList` object will look after `expand_all_tuples` is executed. Hint: `self._first.item` will be mutated.



Part (b) [5 MARKS] Implement this method by filling in the blanks in the partially complete code below. You must not call any other `LinkedList` methods. Do not add any additional lines of code or remove any.

```
def expand_all_tuples(self) -> None:
    """Mutate <self> so that every tuple has been "expanded".
    Expanding a tuple of length n results in n-1 new nodes, each containing
    one item from the original tuple, in the original order. The original node
    contains the first item from the original tuple.

    Preconditions:
    - each tuple in self has length > 0"""
    cur = _____________________________


    while _____________________________:  # traverse one node at a time


        if isinstance(_____________________________, tuple):  # check if we are at a tuple


            part = cur.item


            cur.item = part[0]


            part = part[1:]


            while len(part) > 0:  # while the tuple still has items to expand


                node = _______________________________  # create a new node object


                node.next, cur.next = _______________________________  # link new node in


                cur = _______________________________  # advance cur


                part = _______________________________  # the rest of the tuple to expand


        ___________________________
```

Use this page for rough work.
If you want the work on this page to be marked, indicate this clearly at the location of the original question.

Use this page for rough work.
If you want the work on this page to be marked, indicate this clearly at the location of the original question.

END OF EXAM Total Marks = 26