

**CSC148 term test #1, L0101/L0301**

February 2017

Last name .....

First name .....

Utorid .....

Email .....

1. (10 pts) You are studying the Hogwarts school system of assigning students to houses, and how friendships are formed. You find that Professor McGonagall refers to all the children as “novices,” until they are sorted into their respective houses, at which point they become “students” of Hogwarts.

On the following pages, design a set of classes to replicate your knowledge gathered at Hogwarts, as follows:

- (a) A Novice is characterized by their name, and main attribute (like “bravery”, “ambition”, “hard-work”, and “wisdom”). The Novice class is implemented for you below (without `__str__` or `__eq__` methods). You do not need to add anything to the Novice class.

```
class Novice:
    """ A Novice child, before they get sorted into a house.

    ==== Attributes ====
    @param str name: the Novice child's name
    @param str main_attr: their main attribute,
        one of "bravery", "ambition", "wisdom", "hardwork"
    """

    def __init__(self, name, main_attr):
        """ Initializes a novice child.

        @param Novice self: this Novice child
        @param str name: a name string
        @param str main_attr: an attribute
        @rtype: None
        """

        self.name = name
        self.main_attr = main_attr
```

- (b) Once a Novice is sorted into their correct house, they become a Student in their respective house. A Student is characterized by a name, a main attribute, and the name of their assigned house. A Student `s1` can befriend another Student `s2`, and also check if another Student is already their friend.

On the following page, implement the class Student to reflect that characterization, keeping in mind the hints from the skeleton code and the main program in Part (c). You do not need to implement `__str__` or `__eq__`. You **do** need proper docstrings, including type contracts. You can use any of the docstring formats seen in the course.



- (c) `SortingHat` is a class that takes `Novices` and assigns them to the correct house. It assigns `Novices` to houses based on their main characteristic: “bravery” is typical for “Gryffindor”, “ambition” is typical for “Slytherin”, “hardwork” is typical for “Hufflepuff”, and “wisdom” is typical for “Ravenclaw”. A `SortingHat` also has the ability to find a `Student` based on their name. Implement the `SortingHat` class skeleton on the following two pages, keeping in mind the hints in the main program at the bottom of the next page.

```
# The main program
if __name__ == '__main__':
    novices = [Novice("Harry Potter", "bravery"), Novice("Hermione Granger", "bravery"),
               Novice("Luna Lovegood", "wisdom"), Novice("Draco Malfoy", 'ambition')]

    sohat = SortingHat(novices)
    harry = sohat.find_student("Harry Potter")
    hermione = sohat.find_student("Hermione Granger")
    harry.add_friend(hermione)
    hermione.add_friend(harry)
```

2. (8 pts) Function `myst` has no docstring! Read over its definition anyway.

```
def myst(x):
    if not isinstance(x, list):
        return 1
    else:
        return sum([myst(e) for e in x])
```

Now trace each of the function calls below in order, the way we traced them in lecture and lab. **Important:** Remember, if you have a recursive sub-call on a list of depth  $n$ , but you have already traced an example of a recursive call on a list of depth  $n$ , you **immediately** replace the sub-call by its value **without further expansion**. If you expand recursive calls further, you will lose marks.

(a) `myst("ab")`

(b) `myst(["ab", "cd"])`

(c) `myst(["ab", ["cd", "ef"]])`

(d) `myst(["ab", ["cd", ["ef"]], "gh"])`

3. (7 pts) Three empty Stacks are created and then loaded with some strings:

```
s1 = Stack()
s1.add("S")
s1.add("C")
s2 = Stack()
s2.add("T")
s2.add("K")
s3 = Stack()
s3.add("A")
```

In the space below, write the statements needed to load `s3` so that it contains `"S", "T", "A", "C", "K"`, in order, with `"S"` added last, so the code at the bottom of the page works as stated.

Use only Stack methods (`add`, `remove`, `size`, and `is_empty`), and only the strings already in the three Stacks. Example: In order to move `"A"` from `s3` to `s1`, do `s1.add(s3.remove())`.

You may not create other objects or cause exceptions by removing from an empty Stack in your solution.

```
result = ""
while not s3.isempty():
    result = result + s3.remove()
result == "STACK" # this should be True
```

This page is available for answers that don't fit elsewhere.