UNIVERSITY OF TORONTO
Faculty of Arts and Science

term test #1, Version 1
CSC1481S
Date: Wednesday February 7, 6:10–7:00pm
Duration: 50 minutes
Instructor(s):
AbdulAziz Alhelali
Arnamoy Battacharyya
Danny Heap
No Aids Allowed

Name:
utorid:
U of T email:

Please read the following guidelines carefully!

- Please write your name, utorid, and student number on the front of this exam.
- This examination has 3 questions. There are a total of 9 pages, DOUBLE-SIDED.
- Answer questions clearly and completely.
- You will receive 20% of the marks for any question you leave blank or indicate “I cannot answer this question.”

Take a deep breath.
This is your chance to show us
How much you’ve learned.

We WANT to give you the credit
Good luck!
1. [10 marks] (≈ 25 minutes) Below we have an implementation of class Employee. On the following pages, implement two subclasses:

**SalariedEmployee** has an annual salary, which does not need to be in its string representation.

**HourlyEmployee** has an hourly rate, and works a fixed number of hours per month (both potentially different per employee), which do not need to be in its string representation.

Your implementations should provide a string representation of Employee objects that shows the employee’s name, phone, email, and monthly pay. You do not need to provide an \_eq\_ method.

You must write docstrings for each class and method with type signatures/annotations for parameters and public attributes given in the format of the example code below.

No examples (such as doctests) are required. Indicate which methods are overriding others with a brief comment in the docstring of the method.

class Employee:
    """ Represent an Employee’s information
    name - name
    phone - phone number
    email - email
    """
    name: str
    phone: int
    email: str
    def \_\_init\_\_(self, name: str, phone: int, email: str) \-> None:
        """ Initialize a new employee
        """
        self.name, self.phone, self.email = name, phone, email
    def \_\_str\_\_(self) \-> str:
        """ Return a string representation of the employee information.
        """
        return ("Name: {}
        Phone: {}
        Email: {}
        Monthly Pay: {}" .format(self.name, self.phone, self.email, self.get_monthly_payment()))
    def get_monthly_payment(self) \-> int:
        """ Return the monthly payment (in cents) of the employee.
        """
        raise Not Implemented Error

Solution

class SalariedEmployee(Employee):
    """ Represent an Salaried Employee information
    """
annual_salary - annual salary

annual_salary: int

def __init__(self, name: str, phone: int, email: str,
            annual_salary: int) -> None:
    """ Initialize a new Salaried Employee with annual salary (in cents)
    Extends Employee.__init__
    ""
    Employee.__init__(self, name, phone, email)
    self.annual_salary = annual_salary

def get_monthly_payment(self) -> int:
    """ Return the monthly payment (in cents) of the Salaried Employee.
    Overrides Employee.get_monthly_payment
    ""
    return self.annual_salary / 12

class HourlyEmployee(Employee):
    """ Represent an Hourly Employee information
    hourly_rate - payment rate per hour
    monthly_hours - hours worked per month
    ""
    hourly_rate: int
    monthly_hours: int

def __init__(self, name: str, phone: int, email: str,
             hourly_rate: int, monthly_hours: int) -> None:
    """ Initialize a new Hourly Employee with hourly rate (in cents)
    and fixed number of hours per month.
    Extends Employee.__init__
    ""
    Employee.__init__(self, name, phone, email)
    self.hourly_rate, self.monthly_hours = hourly_rate, monthly_hours
def get_monthly_payment(self) -> int:
    """Return the monthly payment (in cents) of the Hourly Employee."

    Overrides Employee.get_monthly_payment
    """
    return self.hourly_rate * self.monthly_hours
2. [6 marks] (≈ 10 minutes) Linked lists: Below is an implementation of classes LinkedListNode and LinkedList, which you’ve seen in lecture last week. At the bottom of the next page, write the body of method swap, which should swap the values of nodes in two linked lists without editing the front or back elements of the LinkedLists, or the next_ elements of the LinkedListNodes. Use only LinkedList methods implemented here, and do not use Python lists!

```python
from typing import Union, Any

class LinkedListException(Exception):
    pass

class LinkedListNode:
    """ Node to be used in linked list
    next_ - successor to this LinkedListNode
    value - data represented by this LinkedListNode
    """
    next_: Union["LinkedListNode", None]
    value: object

    def __init__(self, value: object, next_: Union["LinkedListNode", None] = None) -> None:
        """ Create LinkedListNode self with data value and successor next
        >>> LinkedListNode(5).value
        5
        >>> LinkedListNode(5).next_ is None
        True
        """
        self.value, self.next_ = value, next_

    def __str__(self) -> str:
        """ Return a user-friendly representation of this LinkedListNode.
        >>> n = LinkedListNode(5, LinkedListNode(7))
        >>> print(n)
        5 ->7 ->|
        """
        cur_node = self
        result = ''
        while cur_node is not None:
            result += '{} ->'.format(cur_node.value)
            cur_node = cur_node.next_
        return result + '|

class LinkedList:
    """ Collection of LinkedListNodes
    front - first node of this LinkedList
    back - last node of this LinkedList
    size - number of nodes in this LinkedList, >= 0
    """
    front: Union[LinkedListNode, None]
    back: Union[LinkedListNode, None]
    size: int
```

never, ever, write below this line...
```python
def __init__(self) -> None:
    """ Create an empty linked list. """
    self.front, self.back, self.size = None, None, 0

def prepend(self, value: object) -> None:
    """ Insert value before LinkedList self.front. """
    self.front = LinkedListNode(value, self.front)
    if self.back is None:
        self.back = self.front
    self.size += 1

def swap(self, other: 'LinkedList') -> None:
    """ Swaps the values of two Linked Lists, leaving node ids intact. Raise LinkedListException if lists are different sizes. """
    if self.size != other.size:
        raise LinkedListException("lists should have the same size")
    left_list, right_list = self.front, other.front
    while left_list is not None:
        left_list, right_list = right_list, left_list
```

### Solution

```python
if self.size != other.size:
    raise LinkedListException("lists should have the same size")
left_list, right_list = self.front, other.front
while left_list is not None:
```
left_list.value, right_list.value = right_list.value, left_list.value
left_list, right_list = left_list.next_, right_list.next_
Three empty Queues are created and then loaded with some strings:

```python
q1 = Queue()
q1.add("P")
q1.add("S")
q2 = Queue()
q2.add("H")
q2.add("A")
q3 = Queue()
q3.add("E")
```

Choose a sequence of commands from the table below to load q3 so that it contains "S", "H", "A", "P", "E", in order, with "E" added last. When you're done the code at the bottom of the page should run as stated.

You may not use any other Python expressions except those in the table. You may use some of the commands in the table more than once, some of them not all.

**Hint:** Try to draw what the queues contain to start with, and come up with the sequence of actions needed (in picture form, crossing out elements you remove) before writing any python code.

```
<table>
<thead>
<tr>
<th>q1.remove()</th>
<th>q1.add(q2.remove())</th>
<th>q1.add(q3.remove())</th>
</tr>
</thead>
<tbody>
<tr>
<td>q2.remove()</td>
<td>q2.add(q1.remove())</td>
<td>q2.add(q3.remove())</td>
</tr>
<tr>
<td>q3.remove()</td>
<td>q3.add(q1.remove())</td>
<td>q3.add(q2.remove())</td>
</tr>
</tbody>
</table>
```

```python
result = ""
while not q3.isempty():
    result = result + q3.remove()
result == "SHAPE"  # this should be True
```

**Solution**

```python
q2.add(q1.remove())
q2.add(q3.remove())
q3.add(q1.remove())
q3.add(q2.remove())
q3.add(q2.remove())
q3.add(q2.remove())
```