CompSci 107 – Computer Science Fundamentals
S1 – 2015
Lab/Assignment 6 – Linked Lists (UnorderedList, OrderedList)

Assignment Due Date: 7pm May 8, 2015

Worth: This lab/assignment is worth 2.5% of your final grade. The work is marked out of 10. The programming exercises will be marked out of 9, with the remaining 1 mark from the code review process.

Topics covered: UnorderedList ADT, OrderedList ADT, implementations using a singly linked list

NOTE (VERY IMPORTANT): Download the "A6.zip" file from the assignments website:

https://www.cs.auckland.ac.nz/courses/compsci107s1c/labs/

Assignment Exercises PLEASE NOTE

The following exercises **must be developed on your computer (or on your USB, or on your university drive).** **Note that you may be asked to produce the code you developed.** Once you are happy that your answer for each question is correct, you will then need to insert the part of the answer required into CodeRunner.

Exercise 1 (3 marks).

In the `A6Ex01` folder you will find a partially defined UnorderedList class (containing the `init()` method, a `get_tail()` method and several 'magic' methods – these methods should not be changed in any way).

As well, you will find a `Node` class which has a single link to the next `Node` object in a chain of nodes, a `ListIterator` class and an application, `A6Ex01.py`.

The `UnorderedList` Abstract Data Type has the following operations:

add(item) adds a new item to the list. It needs the item and returns nothing. This method should add the item at the start of the unordered list.
append(item) adds a new item to the end of the list making it the last item in the collection. It needs the item and returns nothing. Assume the item is not already in the list.
size() returns the number of items in the list. It needs no parameters and returns an integer.
is_empty() tests to see whether the list is empty. It needs no parameters and returns a boolean value.
search(item) searches for the item in the list. It needs the item and returns a boolean value.
remove(item) removes the item from the list. It needs the item and modifies the list. Assume the item is present in the list.
pop() removes **and returns the last item in the list**. It needs nothing and returns an item. Assume the item is present in the list.
pop(pos) removes **and returns the item at position pos**. It needs the position and returns an item. Assume the item is present in the list.
You do not need to implement the following two List ADT methods: **index(item)**, **insert(pos, item)**.

Also add the following method to your class:

get(pos) returns the item at position pos. It needs the position and returns an item. Assume the item is present in the list.

Complete this singly linked list implementation of the `UnorderedList` ADT. Your implementation should have a pointer to the head of the list **and a pointer to the tail of the list** (see relevant slide in lecture 15). Your implementation should implement an iterator (use the `ListIterator` class provided). As well, your implementation should define the `__str__()` method for the class.

Run the `A6Ex01` application which should produce the following output:

```
small unordered list
1. 10 11 18 19
2. 19
3. 21
4. 13
5. [12, 14, 15, 16, 17, 18]
6. 15 is in the linked list
7. True False
8. 6
9. [12, 14, 15, 16, 17, 18, 18, 18]
10. [12 14 15 16 17 18 18 18]
```

```
bigger unordered list
1. 441
2. 28
3. 56
4. 58
5. 172
6. False True
7. 169
8. 119
```

Once you are happy that your class is correctly coded, insert the 11 methods of your `UnorderedList` class into CodeRunner, assignment 6 question 1.

Exercise 2 (1 mark).

Copy your `UnorderedList` implementation from Exercise 1 into the `A6Ex02` folder and add a `to_list()` method to your `UnorderedList` class. This method converts a linked list of items into a Python list of items, e.g., the following code:

```
sllist_of_names = UnorderedList()
sllist_of_nums.append("Jules")
sllist_of_nums.append("Jim")
sllist_of_nums.append("Jill")
little_list = sllist_of_nums.to_list()
print(little_list)
```

prints:

```
['Jules', 'Jim', 'Jill']
```

Run the `A6Ex02` application which should produce the following output:

```
['Preston', 'Tyler-James', 'Annabelle', 'Finn', 'Pippo', 'Betty',
'Giovanni', 'Kaden', 'Claudia', 'Daniela']
1. ['491', '490', '493', '497', '493', '483']
2. []
```

Once you are happy that your method is correctly coded, insert your `to_list()` method into CodeRunner, assignment 6 question 2.

Exercise 3 (2 marks). The `OrderedList` Abstract Data Type has the following operations:

add(item) adds a new item to the list. It needs the item in the correct position (this is an ordered list) and returns nothing. Assume the item is not already in the list.

size() returns the number of items in the list. It needs no parameters and returns an integer.

is_empty() tests to see whether the list is empty. It needs no parameters and returns a boolean value.

search(item) searches for the item in the list. It needs the item and returns a boolean value.

remove(item) removes the item from the list. It needs the item and modifies the list. Assume the item is present in the list.

pop() removes **and returns the last item in the list**. It needs nothing and returns an item. Assume the item is present in the list.

pop(pos) removes **and returns the item at position pos**. It needs the position and returns an item. Assume the item is present in the list.

Also add the following method to your class:

get(pos) returns the item at position pos. It needs the position and returns an item. Assume the item is present in the list.

Your implementation should also define **an iterator** and the **`__str__()`** method for the class.

In the `A6Ex03` folder you will find a partially defined `OrderedList` class (containing the `init()` method, a `get_tail()` method and some 'magic' methods – these methods should not be changed in any way). As well, there is a `Node` class, a `ListIterator` class and an application, `A6Ex03.py`.

Complete this singly linked list implementation of the `OrderedList` ADT. Your implementation should have a pointer to the head of the list **and a pointer to the tail of the list** (see the relevant slides in lecture 16).

To test your completed implementation, run the `A6Ex03` application. The output produced should be:

```
small ordered list
1. Currently list is: [10, 13, 16, 19, 22, 25, 28, 31, 34, 37, 40]
2. 10 13 34 37
3. 40
4. 45
5. Currently list is: [10, 13, 16, 19, 22, 25, 28, 31, 34, 37, 40, 45]
6. Currently list is: [16, 19, 25, 31, 34, 37]
7. 19
8. Currently list is: [16, 25, 31, 34, 37]
9. 25 is in the ordered linked list
10. True True False True
11. 5
12. Currently list is: [16, 25, 33, 34, 37]
13. Currently list is: [16, 25, 33, 36, 37]
14. [16 25 33 36 37]
```

```
bigger ordered list
1. 10 49 171 435 493 497
2. 497
3. 562
4. 161
5. 463
6. True False
7. 170
8. 49
```

Once you are happy that your class is correctly coded, insert the 10 methods of your `OrderedList` class into CodeRunner, assignment 6 question 3.

Exercise 4 (2 marks).

Copy your `UnorderedList` implementation from Exercise 1 into the `A6Ex04` folder and add a `slice()` method to your `UnorderedList` class. This method returns a slice of the unordered list (an `UnorderedList` object), e.g., the following code:

```
sllist_of_nums = UnorderedList()
sllist_of_nums.append("34")
sllist_of_nums.append("21")
sllist_of_nums.append("6")
a_slice = sllist_of_nums.slice(0, 2)
print(sllist_of_nums)
print(a_slice)
print(sllist_of_nums.slice(2, -1))
print(sllist_of_nums.slice(-1, 2))
```

prints:

```
[34, 21, 6]
[34, 21]
[6, 21, 34]
[]
```

This method has two parameters, a start index and an end index. If the start index is less than the end index, the method returns a list of the elements from the first index to the end index not including the end element. If the start index is greater than the end index, the method returns a list of the elements from the end index down to the start index not including the start element. Note that a slice from some index down to -1 includes the element at position 0 and that a slice from a position to the end of the list includes the last element of the list. If one or both of the indices are not valid then the method returns an empty UnorderedList object.

Note: you should NOT use any Python lists in your answer to this question.

To test your completed code, run the `A6Ex04` application which should produce the following output:

```
slicing an unordered list
1. Currently list is: [44, 7, 68, 24, 29, 91, 52, 39, 63, 38, 70]
```

```
2. slice(2, 6): [68, 24, 29, 91]
3. slice(6, 2): [52, 91, 29, 24]
4. slice(7, 10): [39, 63, 38]
5. slice(10, 7): [70, 38, 63]
6. slice(7, -2): []
7. slice(-2, -5): []
8. slice(7, 8): [39]
9. slice(3, 2): [24]
10. slice(3, 0): [24, 68, 7]
11. slice(3, 3): []
12. slice(4, -1): [29, 24, 68, 7, 44]
13. slice(6, -1): [52, 91, 29, 24, 68, 7, 44]
14. slice(-1, 6): []
15. type(...slice(3, 7)): <class 'UnorderedSLList.UnorderedList'>
```

Once you are happy that your method is correctly coded, insert your `slice()` method into CodeRunner, assignment 6 question 4.

Exercise 5 (1 mark).

A `Node` class which has a link to the next `Node` object in a chain of nodes and a link to the previous `Node` object in a chain of nodes is defined as follows:

```
class Node:
    def __init__(self, init_data):
        self.data = init_data
        self.prev = None
        self.next = None
    def get_data(self):
        return self.data
    def get_prev(self):
        return self.prev
    def get_next(self):
        return self.next
    def set_data(self, new_data):
        self.data = new_data
    def set_next(self, new_next):
        self.next = new_next
    def set_prev(self, new_prev):
        self.prev = new_prev
    def __str__(self):
        return str(self.data)
```

In CodeRunner, assignment 6 question 5 show the output of the following section of code. Your output should be a list of elements, e.g., [...]

```
n1 = Node("A")
n2 = Node("B")
n3 = Node("C")
n4 = Node("D")
n3.set_next(n2)
n2.set_next(n4)
n1.set_next(Node("E"))
n4.set_next(n1.get_next())
n1.set_prev(n4)
a_node = n1.get_next()
a_node.set_prev(n2)
n2.set_prev(n3)
n4.set_prev(Node("F"))

temp = n2.get_data()
n2.set_data(n3.get_data())
n3.set_data(temp)

visited = []

a_node = n3
while a_node != None:
      visited.append(str(a_node))
      a_node = a_node.get_next()

visited.append("--")

a_node = n1
while a_node != None:
      visited.append(str(a_node))
      a_node = a_node.get_prev()

print(visited)
```

Exercise 6 (1 mark). Participation in the code review of the programming exercises.