PROBLEM 7-1 (10/10 points)

There are 2 coding problems on this page. Consider the following class definition:

```
class Frob(object):
    def __init__(self, name):
        self.name = name
        self.before = None
        self.after = None
    def setBefore(self, before):
        # example: a.setBefore(b) sets b before a
        self.before = before
    def setAfter(self, after):
        # example: a.setAfter(b) sets b after a
        self.after = after
    def getBefore(self):
        return self.before
    def getAfter(self):
        return self.after
    def myName(self):
        return self.name
```

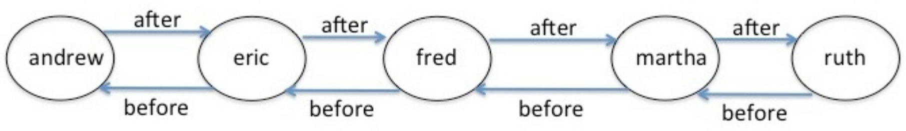
A `Frob` is an object that has a name, and two connections or **links**: a "before" and an "after" link that are intended to point to other instances of objects.

We can use `Frob`s to form a data structure called a **doubly linked list**. In a doubly linked list, each element has the property that if element A has a "before" link to element B, then element B has an "after" link to element A. We want to create a doubly linked collection of `Frob` instances with the property that all `Frob`s with names that are alphabetically before a specific `Frob`'s name appear ordered along the "before" link, and all `Frob`s with names that are alphabetically after a specific `Frob`'s name appear ordered along the "after" link. Here is an example:

```
eric = Frob('eric')
andrew = Frob('andrew')
ruth = Frob('ruth')
fred = Frob('fred')
martha = Frob('martha')

insert(eric, andrew)
insert(eric, ruth)
insert(eric, fred)
insert(ruth, martha)
```

And here is a diagram of the resulting data structure:



Note that if a `Frob` is inserted with the same name as a pre-existing `Frob`, both names should be inserted in the final data structure (the exact ordering of the two identical `Frob`s does not matter). So in the above example, if we were to next execute the line `insert(eric, Frob('martha'))`, we would expect the doubly linked list to have the elements in the following order: andrew - eric - fred - martha - martha - ruth.

Provide a definition for an insert function that will create an ordered doubly linked list. This function is defined outside of the class `Frob`, and takes two arguments: a `Frob` that is currently part of a doubly linked list, and a new `Frob`. The new `Frob` will not initially have any "before" or "after" links to other `Frob`s. The function should mutate the list to place the new Frob in the correct location, with the resulting doubly linked list having appropriate "before" and "after" links. Complete the following function definition:

```
def insert(atMe, newFrob):
    """
    atMe: a Frob that is part of a doubly linked list
    newFrob:  a Frob with no links
    This procedure appropriately inserts newFrob into the linked list that atMe is a part of.
    """
```

Please try the problem first without looking at the hints.

Hints

What 2 cases should you think about?

I'm still stuck!

```
    """
    prev, cur = None, atMe
    if cur.name < newFrob.name:
        while cur and cur.name < newFrob.name:
            prev, cur = cur, cur.getAfter()
    else:
        prev, cur = cur, cur.getAfter()
        while prev and prev.name > newFrob.name:
            cur, prev = prev, prev.getBefore()
    newFrob.setBefore(prev)
    newFrob.setAfter(cur)
    if prev:
        prev.setAfter(newFrob)
    if cur:
        cur.setBefore(newFrob)
```

Correct

Test results

Hide output

CORRECT

Test: 1A

Adding to the beginning: Adding 'allison' to the list ('gabby')

Output:

```
Test: insert(Frob("gabby"), Frob("allison"))
*** Walking the linked list forward: ***
allison
gabby
```

Test: 1B

Adding to the beginning: Adding 'allison' to the list ('gabby')

Output:

```
Test: insert(Frob("gabby"), Frob("allison"))
*** Walking the linked list backward: ***
gabby
allison
```

Test: 2A

Adding to the end: Adding 'zara' to the list ('gabby')

Output:

```
Test: insert(Frob("gabby"), Frob("zara"))
*** Walking the linked list forward: ***
gabby
zara
```

Test: 2B

Adding to the end: Adding 'zara' to the list ('gabby')

Output:

```
Test: insert(Frob("gabby"), Frob("zara"))
*** Walking the linked list backward: ***
zara
gabby
```

Test: 3A

Multiple names. test_list = Frob('abby')

Output:

```
Test: insert(test_list, Frob("xander"))
Test: insert(test_list, Frob("beto"))
*** Walking the linked list forward: ***
abby
beto
xander
```

Test: 3B

Multiple names. test_list = Frob('abby')

Output:

```
Test: insert(test_list, Frob("xander"))
Test: insert(test_list, Frob("beto"))
*** Walking the linked list backward: ***
xander
beto
abby
```

Test: 4

Equal names: Adding 'alvin' to the list ('alvin')

Output:

```
Test: insert(Frob("alvin"), Frob("alvin"))
*** Walking the linked list forward: ***
alvin
alvin
*** Walking the linked list backward: ***
alvin
alvin
```

Test: 5

Multiple names. test_list = Frob('allison')

Output:

```
Test: insert(test_list, Frob("lyla"))
Test: insert(test_list, Frob("christina"))
Test: insert(test_list, Frob("ben"))
*** Walking the linked list forward: ***
allison
ben
christina
lyla
*** Walking the linked list backward: ***
lyla
christina
ben
allison
```

Test: 6

Multiple names. test_list = Frob('zsa zsa')
a = sm.Frob('ashley')
m = sm.Frob('marcella')
v = sm.Frob('victor')

Output:

```
Test: insert(test_list, m)
Test: insert(m, a)
Test: insert(a, v)
*** Walking the linked list forward: ***
ashley
marcella
victor
zsa zsa
*** Walking the linked list backward: ***
zsa zsa
victor
marcella
ashley
```

Test: 7

Multiple names. test_list = Frob('mark')
c = Frob('craig')

Output:

```
Test: insert(test_list, Frob("sam"))
Test: insert(test_list, Frob("nick"))
Test: insert(test_list, c)
Test: insert(c, Frob("xanthi"))
Test: insert(test_list, Frob("jayne"))
Test: insert(c, Frob("martha"))
*** Walking the linked list forward: ***
craig
jayne
mark
martha
nick
sam
xanthi
*** Walking the linked list backward: ***
xanthi
sam
nick
martha
mark
jayne
craig
```

`Test: 8`

Multiple names. test_list = Frob('leonid')

a = Frob('amara')

j1 = Frob('jennifer')

j2 = Frob('jennifer')

s = Frob('scott')

`Output:`

```
Test: insert(test_list, s)
Test: insert(s, j1)
Test: insert(s, j2)
Test: insert(j1, a)
*** Walking the linked list forward: ***
amara
jennifer
jennifer
leonid
scott
*** Walking the linked list backward: ***
scott
leonid
jennifer
jennifer
amara
```

`Test: 9`

Multiple names. test_list = Frob('eric')

`Output:`

```
Test: insert(test_list, Frob("eric"))
Test: insert(test_list, Frob("chris"))
Test: insert(test_list, Frob("john"))
Test: insert(test_list, Frob("john"))
Test: insert(test_list, Frob("chris"))
Test: insert(test_list, Frob("eric"))
Test: insert(test_list, Frob("john"))
Test: insert(test_list, Frob("chris"))
*** Walking the linked list forward: ***
chris
chris
chris
eric
eric
eric
john
john
john
*** Walking the linked list backward: ***
john
john
john
eric
eric
eric
chris
chris
chris
```

Hide output

Check Save *You have used 2 of 10 submissions*

PROBLEM 7-2 (10/10 points)

Now assume that you have a working insert procedure. Starting with any `Frob` in a doubly linked list, we would like to find the "front" `Frob`, i.e., the one whose name is closest to the beginning of the alphabet. Write a **recursive** function called `findFront` to do this. `findFront` should take as an argument any `Frob` that is part of a doubly linked list.

```python
def findFront(start):
    """
    start: a Frob that is part of a doubly linked list
    returns: the Frob at the beginning of the linked list
    """
    # Your Code Here
    return findFront(start.getBefore()) if start.getBefore() else start

```

Correct

Test results

Hide output

CORRECT

```
Test: 1
```

p = Frob('percival')

Output:

```
findFront(p)
percival
```

```
Test: 2
```

p = Frob('percival')
r = Frob('rupert')
insert(p, r)

Output:

```
findFront(p)
percival
findFront(r)
percival
```

```
Test: 3
```

s = Frob('sterling')
r = Frob('rupert')
insert(s, r)

Output:

```
findFront(s)
rupert
findFront(r)
rupert
```

```
Test: 4
```

Multiple names. test_list = Frob('zsa zsa')
a = sm.Frob('ashley')
m = sm.Frob('marcella')
v = sm.Frob('victor')

Output:

```
insert(test_list, m)
insert(m, a)
insert(a, v)
findFront(v)
ashley
findFront(m)
ashley
```

```
Test: 5
```

```
Multiple names. test_list = Frob('leonid')
a = Frob('amara')
j1 = Frob('jennifer')
j2 = Frob('jennifer')
s = Frob('scott')
```

Output:

```
insert(test_list, s)
insert(s, j1)
insert(s, j2)
insert(j1, a)
findFront(a)
amara
findFront(j1)
amara
findFront(j2)
amara
findFront(s)
amara
findFront(test_list)
amara
```

Hide output

Note: In programming there are many ways to solve a problem. For your code to check correctly here, though, you must write your recursive function such that you make a recursive call directly to the function `findFront`. Thank you for understanding.

Check Save *You have used 1 of 10 submissions*