Building a Linked Data Structure

In this lab, you will build a data structure suitable for storing information about a music CD collection. In doing so, you will employ the two main techniques for representing a sequential collection (contiguous and linked).

1. Open `IDLE` and use it to create a new file called `cds.py`. Put some comments at the top identifying your team members.

2. Create a class definition for `MusicCD`. A `MusicCD` object has attributes for artist(string), album(string), and tracks(list). Artist and title are supplied when the MusicCD is created. The list of tracks is initially empty. Put this main() program in to test your constructor:

```
def main():
    c1 = MusicCD("Pink Floyd", "The Final Cut")
    assert c1.artist == "Pink Floyd"
    assert c1.album == "The Final Cut"
    assert c1.tracks == []
```

3. Add two more CDs of your own choosing to your testing code. Store them in the variables `c2` and `c3`.

4. Add an `addtrack` method to the `MusicCD` class. A track consists of a title(string) and a duration(int, in seconds). Use the method to add a couple tracks to each of your CDs in your testing code.

```
c1.addtrack("The Post War Dream", 182)
c1.addtrack("Your Possible Pasts", 262)
...
assert c1.tracks == [("The Post War Dream", 182),("Your Possible Pasts", 262)]
```

5. Wait for all groups to catch up, then listen to your instructor for the next step. While you are waiting, you can create more CDs in your test code and/or add the following functionality.

   (a) Write a `duration`() method that returns to total running time of a CD (in seconds).

```
assert c1.duration() == 444
```

   (b) Write an `info`() method that returns a three line string:

```
assert c1.info() == "Album: The Final Cut\nArtist: Pink Floyd\nDuration: 444"
```

6. Link your CDs into a collection. Add an instance variable called `link` to `MusicCD`. Set it to `None` in the constructor. Then modify your `main` so that you have a single variable, `collection`, that contains a chain of all your CDs together. Here's a snippet:

```
collection = c1
c1.link = c2
c2.link = c3
...
assert collection.album == "The Final Cut"
assert collection.link.album == <whatever it should be>
assert collection.link.link.album == <whatever it should be>
```

7. Write a function (not a class method) `printCDs` that takes collection as a parameter and loops through the collection to print out information on all of the CDs. It should not matter how many albums are in the collection. Note: If you have not implemented the `info`() method from part 5, then just print out the album title. Here's the algorithm:

```
set local variable current to collection
while current is a MusicCD (i.e. it is not None):
    print out current's info
    set current to the next CD in the collection
```

8. Use the insight from part 7 to create a `CDCollection` class. A `CDCollection` has a single instance variable, `first`, that stores the first CD in the collection.

   (a) Write a constructor that sets `first` to `None`

   (b) Write an `addCD` method that takes a `MusicCD` as a parameter and adds it to the beginning of the collection. Here's an algorithm:

   ```
   set the newCD's link to what first is now
   set first to the newCD
   ```

   (c) Make your `printCDs` function into a class method inside `CDCollection`.

   (d) Add test code in `main`() to create a `CDCollection`, add your CDs to the collection and then test printing them out. Note: they will print in the reverse order that you added them (why?).