Music Streaming Application

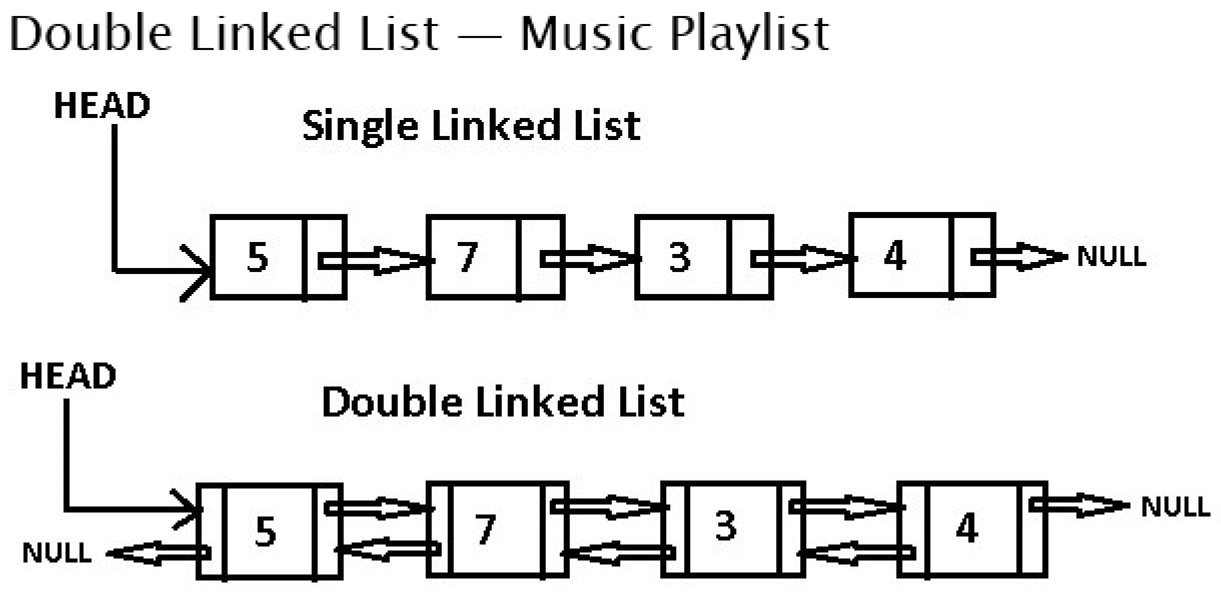
Implementing a Doubly Linked List

Project Due: 07/07/20 @ 2:30pm

1.0 Objective

This project is meant to help you develop skills at working with linked lists and writing entire programs from scratch. In this project you will only be provided with a project description and list of program requirements. No code will be provided at all. This means you will need to learn to design and implement an entire program on your own.

2.0 Project Description



2.1 Basic Program

You are going to create a modified double-linked list as the data structure for a playlist as part of a music streaming application. You will create an application that allows the user to select to skip backward / forward, play a track, append, and go to the beginning or the end.

You will be given an input file with music track titles from three different albums. These tracks are in random order. You are to read in the titles and place them into a data structure in the order in which you read them in. Another data file will contain a list of the music tracks in the order that they appeared on the album. You are to create a doubly linked list that allows the music tracks to be stepped through in the sequence that they appeared on the album.

A third data file will contain a list of commands that you are to execute for each of the albums. Based on what the music player is doing after reading in the next input you will have to indicate what song the system is currently playing.

Once you have completed this, you are to read in the user's playlist which consists of music tracks from all three albums in any order. You are to create a doubly linked list for the playlist that will allow the user to play songs off of their playlist. You are then to read in the list of commands and execute them for the playlist.

Note that a list in python makes this a trivial assignment, but you aren't here to learn python you are here to learn data structures therefore you may NOT create a new list to replicate an ordered album. The data must always reside in the original data structure and no 'shadow' data structures are allowed to implement the album.

The solution to this homework can be implemented using either Python lists or Python objects.

If you come up with a solution that makes this problem trivial, then you are doing something wrong! Talk to me.

2.2 Files Names As Program Parameters

You must take both files, in the correct order, as query string arguments. To make life easier for the TAs, include the following code in your program:

```
import sys


    if len(sys.argv) != 4:
        raise ValueError('Please provide three file names.')

    sys.argv[0] = sys.argv[0][0:len(sys.argv[0]) - sys.argv[0][::-1].find('/')]

    inputFile1 = sys.argv[0]+sys.argv[1]
    inputFile2 = sys.argv[0]+sys.argv[2]
    inputFile3 = sys.argv[0]+sys.argv[3]

    print("\nThe files that will be used for input are {0}, {1}, and
    {2}".format(sys.argv[1],sys.argv[2],sys.argv[3]))
```

To pass the three file name arguments to your python program, in PyCharm run your program (it will fail) and then go to “Run > Edit Configurations” and set the Parameters value to " HW2Songs.txt HW2Albums.txt HW2Commands.txt" (note no comma separating the file names) and save it.

2.3 Reading from a File

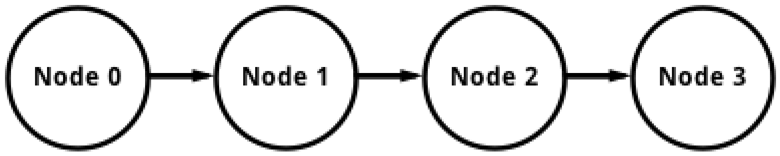
The album tracks will be in the input file in a random order. You are to read them in and place them into a data structure IN THE ORDER THAT YOU READ THEM IN. Once you have done this, you will then create a doubly linked list for this album that will allow the user to sequence through the titles in the proper order that the titles appeared on the album.

To read the music playlist and the playlist commands from a file ("HW2Songs.txt", "HW2Albums.txt" and "HW2Commands.txt") you must first use the Python *inputfile1 = open(filename,"r")* command. At this point you can start reading from the file using *variable = inputfile1.readline()*. Do be careful, if you program can't find the file that it is looking for it will crash.

2.4 What is a Linked List?

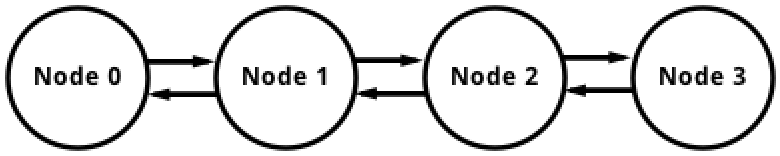
A linked list is a sequence data structure, which connects elements, called nodes, through links. Unlike an array data structure, a node in a linked list isn't necessarily positioned close to the previous element or the next element. Nodes in a linked list are connected by pointers. Linked lists are null terminating, which means whenever the pointer to the next node is null, the list has reached the end.

2.5 Single Linked List



In single linked list, element or nodes only link to the next element in the list. A single linked list is null terminating meaning when the pointer is null, the list is finished.

2.5 Double Linked List



In a double linked list, each node contains a reference to the previous node and the next node (as long as they aren't the head or tail node.) Each node has a value property.

2.6 Doubly Linked List Properties

The reason I chose a double linked list was that the structure suited the behavior of a music playlist.

- **Skip Back/Forward**
  Because each node in a double linked list has a pointer the previous and next node, it is easy to implement skip forward/backward functionality.

- **Play Next Track**
  The pointer to the next node also makes it quite easy to start the next track when a track is over.

- **Append**
  When you add a new track to a playlist, you tack it on to the end. In a linked list, adding a new element is constant time — O(1) operation. Note that as the songs are read in from the data file and added to the play list, this will be done as a sequence of calls to append.

- **Beginning/End**
  Finally, because a linked list has head and tail properties, this provides for an easy way to delineate the beginning and end of a playlist.

2.7 Statistics

Another major part of this project is recording statistics. A summary of all the statistics you must keep are presented in the following table:

Statistic	**Description**
Songs	This is the total number of songs read by your tester program.
Skip Forward	Number of times the Skip Forward command is entered via the music streaming application
Skip Backward	Number of times the Skip Backward command is entered via the music streaming application
Play Next Track	Number of times the Play Next Track command is entered via the music streaming application
Append	Number of times the Append command is entered via the music streaming application
Beginning	Number of times the Beginning command is entered via the music streaming application
End	Number of times the End command is entered via the music streaming application
Time	You need to record the total time of your program. This time should include the time to read the songs words, process the music player commands, and record all the statistics. It is suggested that you start a timer as one of the first things you do in the program and stop it immediately before printing out the

	statistics. The following is a little sample code that shows how you can time events.
	
	`import time`
	
	`start = time.time()`
	`...`
	`end = time.time()`
	`print(end - start)`

While the program is running, every time that it moves to a new song or starts to play a new song, the program should display:

```
Now At:  xxx
Now Playing:  xxx
```

Where `xxx` is replaced with the name of the song.

When the program is finished reading the command text file, it should print out a list of statistics. The output should look exactly like the following:

```
**********************
***** Statistics *****
**********************
Total Songs Read: xxx
Number of Skip Forward Commands: xxx
Number of Skip Backward Commands: xxx
Number of Play Next Track Commands: xxx
Number of Append Commands: xxx
Number of Beginning Forward Commands: xxx
Number of End Forward Commands: xxx
Total Time of Program: xxxxx milliseconds
```

The x's should be replaced with actual numbers.

3.0 Program Design Tips

The number one rule about writing a program from scratch is not to write a single line of code until you have developed a good design. Developing this good design is probably the hardest thing to learn how to do in programming. Here are a few of my suggestions on how you should go about designing and writing this project.

1. Read the program through once to get an idea of what is expected. Then, read it again. The second time through you may want to take some notes. At this point, you should have a thorough understanding of what is expected of you. If you need to read it a third time, do so.

2. Do a very rough design. This basically means figuring out what you are going to want to create and what the purpose of each part of the program will be. It is completely up to you; however, be sure you know why you need each part and what its purpose will be for.

3. Once you have a list of your parts, go through each part and determine what functions it will need and what each of these functions will do. This should include information about what parameters each function will take and what type of value it will return.

4. Then go through each function and write psuedo-code describing how the function will flow. This is a very critical step because if it is done properly, your code writing will be much simpler. A good piece of advice at this stage is when you are examining a function, assume all the other functions already work - even if you have not written the pseudo-code for a function. If you did Step 3 properly, you will know how each function should work, what arguments it will take, and what values it will return.

5. The very last step is to go ahead and write your project up. Do not write all the code at once. Write functions one or two at a time, test them and make sure they work as you expect and then go on to the next function or two. It is always best to start with the lowest level functions first and work your way up. In other words, if you have some function that does not call any other of your functions, then that is the one you want to start with. You can then test it and make sure it works (this often requires a little creativity). Once you have all of these functions written, you can then move up the chain to functions that only call these lowest level functions. Once you finish all of these, move up another level. Continue to do this until all of your functions are written.

One of the things you will notice about this design is that Step 4 calls for a top-down approach (describe all of the higher level functions assuming the lower level ones are done and then move down to the next level). Whereas, Step 5 calls for a bottom-up approach (write and test all of the low level functions before moving on to the higher level functions). I think if you follow this approach, you should find your programs (for this or any other class) get written much more quickly and with far fewer bugs.

4.0 Handing in Your Project

Be sure to comment your code well so that it can be easily browsed to see what is going on.

Code that is excessively difficult to read will be penalized.

Turn in a single file called "COP4530–HW#2.py" to the assignment on Canvas.

5.0 Grading

The project is due at 2:30pm on July 7, 2020. The project will be graded out of 100 points and be graded on the following criteria:

- Non-trivial program compilation
- Correct Output
- Performance (running time)

We will grade your code by running it against a series of test files and checking the output of the program against a known, correct implementation. Any differences in your output versus that of the correct implementation will result in a point deduction. Without a doubt, much more in depth tests will be run than the initial one provided. This means you should write some of your own test files and check for any errors.

We will also examine parts of your code to make sure that you are implementing the data structures as required. If you are getting the correct results but not implementing the proper data structure, your score will suffer severely. You must implement the proper data structures – it is a data structures course after all!