Program #1
CS 260 Data Structures

ALWAYS make a backup of your programs before using tar
Double check that the arguments specified with for tar are correct!
***Late penalties** will apply to all submissions incorrectly archived/emailed*

GENERAL INFORMATION about PROGRAMS in CS260:

Scope: For each project, we only have approximately 2 weeks to complete the write-ups and code. Therefore, it is critical that you focus on a limited scope, with an emphasis on the class(es) and data structures, rather than building a comprehensive application. Each program this term will have you create an Abstract Data Type (ADT), **which is what you will be primarily graded on**. *This means that we will primarily grade your projects based on the use of classes, member functions, arguments, data structures, pointers, efficiency, and how well the software demonstrates that it has been fully tested. The code needs to compile and run, and include a test program for the member functions provided.* **It is not appropriate to hard code in the test cases - all tests should be interactive with the user.** Your user interface must be clear for us to thoroughly test all features.

Class construct: Most programming assignments this term require only one class but you have the option to also implement classes to represent the underlying data as well. The use of structs is allowed.

Design is the First Step: The first step will be to design your software. This should include designing classes that are well structured and that efficiently use the data structure. **The use of recursion** should be included **at least once** in each assignment to better prepare us for future programs. Avoid multiple traversals whenever possible. *Take a look at the style sheet for instruction on the topics that your write-ups need to cover.*

Creating Test Plans: Part of the first step will be to develop a test plan. This includes planning out each of the cases that need to be implemented and subsequently tested. It is important to focus on thoroughly testing your class member functions.

BACKGROUND INFORMATION for Program #1:

This first program is an exercise in building, traversing, and destroying linear linked lists.

Wouldn't it be amazing to create your own "record label"? Shady Records was founded by Eminem in 1999 and has produced a lot of music as part of this label. Portland has a number of home grown (DIY) record labels. Good Cheer Records is about two friends who started their own label to build bridges between various music scenes around town. How great is that?

Programming Assignment: For the first programming assignment, we are going to create a program that helps a new record label (e.g., CS_Record_Label) to keep track of their artists and the most popular songs produced by that artist as part of this record label.

Your primary job will be to create an Abstract Data Type (ADT) for a CS_Record_Label program. The data member should be a head pointer to a LLL of all of the artists on this label. Each node will have a pointer to another linear linked list of the songs from that artist, organized by the most popular to the least. When songs change in popularity, your code will need to reorganize the data in the linear linked lists.

The **information** about an artist should include:

1. Name – the artist's name (e.g., Eminem)
2. Top story about this artist (e.g., " September 25 – Eminem royalties shares to be sold in an IPO")
3. Description – information about when the artist joined the label and the genre of music
4. A LLL of songs, where each song has a:
    a. Title (e.g., Lose Yourself)
    b. Length (e.g., 5.23)
    c. The number of views (e.g., 144,309,378)
    d. Number of likes (e.g., 654,000)

This ADT must have **public member** functions to perform the following:

1. Constructor - Construct an object and initialize the data members
2. Destructor - Release all dynamic memory and reset data members to their zero equivalent value
3. Add a new artist
4. Add a new song for the artist
5. Edit the number of views and likes for a song
6. Display all songs for an artist (in order of popularity)
7. Display all artists
8. Remove all songs with fewer than M views, where M is sent in as an argument

<u>Things you should know...as part of your program:</u>

__General Syntax Rules__

1) Do not use statically allocated arrays in your classes or structures. All memory must be dynamically allocated and kept to a minimum!
2) Global variables are not allowed in CS260; global constants are encouraged
3) **Do not use the String class! (use arrays of characters instead);** you may use the cstring library with strlen, strcpy, strcmp
4) Avoid using break or return from within the body of a loop
5) Avoid using while(1) type of syntax
6) Strive towards structured programming techniques
7) Remember that at least one function needs to be written using recursion!
8) For functions that have non-void return types, make sure to return a value through each path through the function.
9) NEVER pass class types by value (always by reference); NEVER return class types by value.
10) Use the iostream library for all I/O; do not use stdio.h for I/O
11) Use of external data files is required
12) MOST IMPORTANT:

    BACKUP your files before creating a TAR archive!!!!

__General rules for building ADTs__

1) The ADTs should be designed to "hide" the data structure as part of the private section of the class. This means your head pointer to a node should be a private data member in your ADT.

2) All data members in a class must be private
3) Public member functions should NEVER have a node pointer as an argument (Remember client programs should not be involved with the physical data structure when working with ADTs)

4) Never prompt and read from the user when inside a class member function; this should be done in the "client" program when developing ADTs

5) Never output error messages from a class member function; this should also be done in the "client" program when developing ADTs

6) Each public member function should have a way to communicate success or failure back

to the “client” program. This can be done with a returned value, and argument passed by reference, or exception handling.

7) Use modular design, separating the .h files from the .cpp files. Remember, .h files should contain the class header and any necessary prototypes. The .cpp files should contain function definitions. You must have at least 1 .h file and 2 .cpp files. **Never implement functions in your .h file! And, never "#include" .cpp files!**

8) NEVER archive your writeups. Your writeups need to be uploaded on D2L as separate documents.