Lab7: LinkedList

In this lab, we will build upon the knowledge we learned from previous labs. Additionally, from now on, we will compile our files by creating makefiles similar to the ones we learned about from the previous labs. You will also used the gdb in case you needed to debug your code.

The outcomes from the lab are:

- Dynamic memory allocation, making sure that you correctly allocate and free the memory once you do not need to use it.
- Separating programs into client, interface, and implementation files
- Working with linked-lists
- Practicing makefiles.

Lab details:

In lab5, we used the following struct to represent and store information about a single book. We were also still using arrays to store the 55 books information. In this lab, instead of using arrays, we will use linked lists

```
struct book
{
        char * bookName;
        char * authorName;
        //Add more info here if needed
};
```

Using the above struct, you will need to do the following:

1. Write down a function named **loadBookInfo** the function will do the following:
    1. Take one inputs: the file name
    2. Return a pointer to a linkedlist. This linkedlist holds the 55 books information.
    3. The code for the linked list is provided with the lab files
    4. The function will first use the linkedlistInit function to create an empty linkedlist
    5. Afterwards, the function will open the file
    6. The function will use AddNodeBack to add a node to the back of the linked list. Each time you add a new book info, you will use this function to add one new node to the back of the linkedlist. Each node will store information about a single book
    7. Read information about each book from the file and store the information inside the struct book.
    8. Similar to lab5, you will use use the following functions to effectively read from the file:
        1. fgets. It reads a whole line
        2. strtok. String token method
        3. strlen. String length method
        4. strcpy. String copy method.
        5. malloc. To allocate the memory for each string in the array.
2. Write down a function named **destroyBookInfo.** The function will do the following:
    1. Take an linked list of books. The linkedlist was created using the **loadBookInfo** function

    2. The function will call the linkedlistDestroy function that destroys a linkedlist. You will also need to create a function DestroyBookStruct to destroy a single book (i.e., destroy the two arrays inside the struct)

3. Write down your main to test the correctness of your functions, e.g.,:
    1. Ask the user for the name of the file
    2. Call **loadBookInfo** and give the function the file name and hold the return linkedlist
    3. Loop through the linkedlist to make sure that the book's info is read correctly
    4. Call the function **destroyBookInfo** to destroy the linkedlist
4. One possible way to make sure that your program creates and frees the memory correctly is to use the **valgrind** tool.

You will need to submit the following:

- README file explaining what the program does and how you compiled it, which command line options you used. Also, since you are allowed to work with a partner, write down your name, your partner name, what you did in the lab, and what the program does as a whole. You are going to submit your lab separately from your partner.
- Submit your program. The file must be fully documented.
    - All programs must include a comment section at the top of the program as outlined below:

```
/*********************************************************************
Program: Author: Date:
Time spent: Purpose:
<name of program>
<your name>
<date you finish the program>
<total amount of time spent on the project>
The purpose of this program is to blah blah blah
*********************************************************************/
```

- For the function, it must include a comment section giving information about the pre and post conditions

```
//Precondition:
//Postcondition:
```

- Submit the book file that includes the name of the books
- Submit the makefile used to compile the code
- If valgrind was used, submit also the report showing that you did not face any problems.
- The files must be compressed together and named using the following naming format: lab5-<firtinitialLastName>.zip. For example, lab4-jJohn.zip
- Submit the above compressed folder using the submit command to your lab TA account.