# Lab5: Building Objects

In this lab, we will build upon the knowledge we learned from lab1 and lab2. 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 that you correctly allocate and free the memory once you do not need to use it.
- Work more with functions rather than writing everything in the main
- Next step: create client, interface and implementation. Now we are just separating everything into functions making it easier to move to the next step.
- Working with structs
- Working more with the string library in C
- Practicing makefiles.

**Lab details:**

In lab1 and lab2, you were required to read from a file that includes information about 55 books. You were asked to create two string arrays; one you used to store book name and the other one to store the author name. However, this is not an efficient technique to perform such an operation since the two arrays are used to store information about books. A better method is to create a struct that stores information about a single book. The struct will be:

```
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 struct book.
    3. The function will first dynamically allocate the pointer to hold 55 elements of type book struct
    4. Afterwards, the function will open the file
    5. Read information about each book from the file and store the information inside the struct book.
    6. Similar to lab2, 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 array of a struct book. The array was created using the **loadBookInfo** function
    2. Loop through the array and destroy the bookName and authorName strings inside each struct book entry
    3. Destroy the whole struct array (i.e., using the function free to free the memory)
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 a pointer to struct book to dynamically allocate the array of books.
    3. Loop through the array of struct book to make sure that the book's info is read correctly
    4. Call the function **destroyBookInfo** to destroy the struct book array
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.