CS 162-010 Assignment #2 The Wizard Spellbook Catalog

Design Document due: Sunday, 4/17/2022, 11:59 p.m. (Canvas)

Assignment due: Sunday, 4/24/2022, 11:59 p.m. (TEACH)

Demo due: 5/6/2022 (without penalties)

Goals:

- Practice good software engineering design principles:
  - Design your solution before writing the program
  - Develop test cases before writing the program
- Review pointers, arrays, command line arguments, and structs
- Use file I/O to read from and write to files
- Practice file separation and create Makefile
- Use functions to modularize code to increase readability and reduce repeated code

---

Problem Statement:

The great wizard school of Pogwarts has been adding a lot of new spellbooks to their library lately. As a result of all the new literature, some mischievous students have been learning spells that are dangerous. In addition, it has become increasingly difficult for the staff to find specific spells they want to teach in class.

In order to alleviate these issues, the school administration has hired you to create a spellbook catalog program that will allow for access restriction and make searching for spellbooks easier.

To simplify your task, the school has provided you with their spellbook and wizard profile information. These come in the form of two textfiles: `wizards.txt` and `spellbooks.txt`. **(NOTE: The two filenames are provided on the command line and should not be hardcoded in your program!)**

The `spellbooks.txt` file contains a list of Pogwarts's spellbooks and their included spells. This file will give you the spellbook and spell information that your program will organize and display.

The `wizards.txt` file contains a listing of all wizards associated with Pogwarts. This file will give your program the information needed to control which spellbooks users can access.

Requirements:

1. **Command Line Arguments:**
   When starting the program, the user will provide two command line arguments. The first command line argument will be **the name of the file that holds the information about the wizards**. The

second command line argument will be **the name of the file that contains the information about spellbooks and included spells**. If the user does not provide two names of existing files, the program should print out an error message and quit. You can assume that the user always provides these two command line arguments in the correct order.

2. **Wizard Login:**
   Before the user can get information from the program, they must login. You must prompt the user for an ID and password. The ID of the must be all ints. If it isn't, you must re-prompt the user for an ID that is only ints. You will then prompt the user for the password. The password will be a string that could contain any number of characters and numbers. You do not need to do any error handling on the password. You will then need to check this information against the `wizard_info.txt` file contents to determine which wizard is logging in. If there is no match, you must re-prompt for both username and password again. If the user fails to login after three attempts, you must provide an error message and quit out of the program. **(Note: If the user provides an ID that is not all integers that should not count as an error towards the three attempts that they have to login)**. After a user logs in, you must display the corresponding name, ID, school position, and beard length.

3. **Sorting and Printing:**
   Once the user has logged in, they should be prompted with a list of different ways to display the sorted spellbook and spell information. Wizards with student status cannot view any spell that has an effect of **death** or **poison** and cannot view any spellbook with a spell that has an effect of **death** or **poison**. After the user has chosen an option, they should be asked if they want the information printed to the screen or written to a file. If they choose to write to a file, they should be prompted for a file name. If the file name already exists, the information should be **appended** to the file. If the file name does not exist, a file should be created, and the information should be written to the file. For your sorting, you CANNOT use the built-in sort function.

4. **Available Options (after successful login):**
   - **Sort spellbooks by number of pages**: If the user picks this option, the books must be sorted in *ascending* order by number of pages. Books with a spell that have an effect of **death** or **poison** should not be displayed when a student is logged in (i.e. hide those books from the results when a student is logged in). Once they are sorted, you should print/write to file the title of the book and the number of pages it has.

   - **Sort spells by effect**: There are five types of spells: fire, bubble, memory_loss, death, poison. Remember that students cannot see spells that have the type **death** or **poison**. The spells with bubbles as the effect should be listed first, followed by death, fire, memory_loss, and poison. Once they are sorted, you should print/write to file the spell name and its effect.

   - **Sort by average success rate of spells**: You must create a list of books sorted by the average success rate, in *descending* order, of all the spells contained within the book. Once calculated, you should print/write to file the title of each applicable book and the corresponding average success rate. Books with a spell that have an effect of **death** or **poison** should not be displayed when a student is logged in (i.e. hide those books from the results when a student is logged in).

   - **Quit**: The program will exit.

5. Your program should continue sorting and printing/writing until the user chooses to quit.

Required Structs:

The following structs are required in your program. They will help organize the information that will be read in (or derived) from the files. **You cannot modify, add, or take away any part of the struct**.

1. The **wizard** struct will be used when you read in entries from the `wizard.txt` file. There are three options for **position_title**: "Headmaster", "Teacher", or "Student".

```
struct wizard {
  string name;
  int id;
  string password;
  string position_title;
  float beard_length;
};
```

2. The **spellbook** struct will be used to hold information from the `spellbooks.txt` file. This struct holds information about a spellbook.

```
struct spellbook {
  string title;
  string author;
  int num_pages;
  int edition;
  int num_spells;
  float avg_success_rate;
  struct spell *s;
};
```

3. The **spell** struct will also be used to read in information from the `spellbooks.txt` file. This struct holds information about a spell. There are five options for effect: "fire", "poison", "bubbles", "death", or "memory_loss".

```
struct spell {
  string name;
  float success_rate;
  string effect;
};
```

Required Functions:

You must implement the following functions in your program. **You are not allowed to modify these required function declarations in any way**. *Note: it is acceptable if you choose to add additional functions (but you must still include the required functions).*

1. This function will dynamically allocate an array of spellbooks (of the requested size):

```
spellbook * create_spellbooks(int);
```

2. This function will fill a spellbook struct with information that is read in from `spellbooks.txt`. *Hint: "ifstream &" is a reference to a filestream object. You will need to create one and pass it into this function to read from the spellbooks.txt file.*

   ```
   void populate_spellbook_data(spellbook *, int, ifstream &);
   ```

3. This function will dynamically allocate an array of spells (of the requested size):

   ```
   spell * create_spells(int);
   ```

4. This function will fill a spell struct with information that is read in from `spellbooks.txt`.

   ```
   void populate_spell_data(spell *, int, ifstream &);
   ```

5. You need to implement a function that will delete all the memory that was dynamically allocated. You can choose the prototype. Possible examples include the following:

   ```
   void delete_info(wizard **, int, spellbook **, int);
   void delete_info(wizard **, spellbook **, int);
   ```

Required Input File Format:

1. Your program must accommodate the file formats as specified in this section. The `wizards.txt` file will have the following pattern:

   ```
   <number of wizards in file>
   <wizard name> <wizard ID#> <wizard password> <wizard position title> <wizard beard length>
   <wizard name> <wizard ID#> <wizard password> <wizard position title> <wizard beard length>
   <wizard name> <wizard ID#> <wizard password> <wizard position title> <wizard beard length>
   ...
   ```

   An example wizards.txt file is provided here.

2. The `spellbooks.txt` file has a slightly different structure. The file information will be provided in sets (corresponding to each spellbook). Each spellbook will be accompanied by a listing of the spells inside.

   The `spellbooks.txt` file will have the following pattern:

   ```
   <total number of spellbooks in file>
   <title of first spellbook> <author> <number of pages> <edition> <number of spells in book>
   <title of spell> <success rate> <effect>
   <title of spell> <success rate> <effect>
   <title of spell> <success rate> <effect>
   ...
   <title of second spellbook> <author> <number of pages> <edition> <number of spells in book>
   <title of spell> <success rate> <effect>
   <title of spell> <success rate> <effect>
   <title of spell> <success rate> <effect>
   <title of spell> <success rate> <effect>
   ...
   <title of third spellbook> <author> <number of pages> <edition> <number of spells in book>
   <title of spell> <success rate> <effect>
   <title of spell> <success rate> <effect>
   ```

An example spellbooks.txt file is provided here.

Required files:

You are expected to modularize your code into a header file (.h), an implementation file (.cpp), and a driver file (.cpp).

1. `catalog.h:` This header file contains all struct declarations and function prototypes
2. `catalog.cpp:` This implementation file contains all function definitions
3. `run_wizard.cpp:` This driver file contains the main function
4. A `Makefile` is required

Example Operation

The following snippet of text shows an example implementation of the spellbook program. Note that this example does not illustrate all required behavior. You must read this entire document to ensure that you meet all of the program requirements. (User inputs are highlighted)

<Note: This example output is only intended to illustrate the program operation. Your values will be different.>

```
$ ./catalog_prog wizards.txt spellbooks.txt
Please enter your id: 123456
Please enter your password: gr#ywiz
Incorrect id or password
Please enter your id: 123456
Please enter your password: gr3ywiz

Welcome Gandalf
ID: 123456
Status: Teacher
Beard Length: 9001

Which option would you like to choose?
1. Sort spellbooks by number of pages
2. Group spells by their effect
3. Sort spellbooks by average success rate
4. Quit
Your Choice: 1

How would you like the information displayed?
1. Print to screen (Press 1)
2. Print to file (Press 2)
Your Choice: 1

Spells_for_Dummies 303
Wacky_Witch_Handbook 1344
Necronomicon 1890
Forbidden_Tome 1938
Enchiridion 2090
The_Uses_of_Excalibur 3322
Charming_Charms 4460
Dorian 50000
```

```
Which option would you like to choose?
1. Sort spellbooks by number of pages
2. Group spells by their effect
3. Sort spellbooks by average success rate
4. Quit
Your Choice: 3

How would you like the information displayed?
1. Print to screen (Press 1)
2. Print to file (Press 2)
Your Choice: 1

Wacky_Witch_Handbook 90.05
The_Uses_of_Excalibur 87.9
Forbidden_Tome 76.8
Necronomicon 72.34
Enchiridion 51.2
Dorian 48.64
Charming_Charms 37.8
Spells_for_Dummies 29.74

Which option would you like to choose?
1. Sort spellbooks by number of pages
2. Group spells by their effect
3. Sort spellbooks by average success rate
4. Quit
Your Choice: 3

How would you like the information displayed?
1. Print to screen (Press 1)
2. Print to file (Press 2)
Your Choice: 2

Please provide desired filename: success_rate_list.txt
Appended requested information to file.

Which option would you like to choose?
1. Sort spellbooks by number of pages
2. Group spells by their effect
3. Sort spellbooks by average success rate
4. Quit
Your Choice: 2

How would you like the information displayed?
1. Print to screen (Press 1)
2. Print to file (Press 2)
Your Choice: 1

bubble Bubble_Beam
bubble Exploratory_Maneuver
death Deathrock
death Nightmare
death Deadly_Details
fire Blasto_Strike
fire Beginning_Blast
memory_loss Space_Out
memory_loss Preliminary_Black_out
memory_loss Wacky_Dreams
poison Cthulhu_Venom


Which option would you like to choose?
1. Sort spellbooks by number of pages
2. Group spells by their effect
3. Sort spellbooks by average success rate
4. Quit
Your Choice: 4

Bye!
```

Programming Style/Comments

In your implementation, make sure that you include a program header. Also ensure that you use proper indentation/spacing and include comments! Below is an example header to include. Make sure you review the style guidelines for this class , and begin trying to follow them, i.e. don't align everything on the left or put everything on one line!

```
/******************************************************
** Program: wizard_catalog.cpp
** Author: Your Name
** Date: 09/30/2021
** Description:
** Input:
** Output:
******************************************************/
```

Design Document – Due Sunday 4/17/2022, 11:59pm on Canvas
Refer to the Example Design Document – Example_Design_Doc.pdf

Understanding the Problem/Problem Analysis:

- What are the user inputs, program outputs, etc.?
- What assumptions are you making?
- What are all the tasks and subtasks in this problem?

Program Design:

- What does the overall big picture of each function look like? (Flowchart or pseudocode)
    - What variables do you need to create, when you read input from the user?
    - How to read from a file, and store information into your program?
    - How to write information from your program to a file?
    - What are the decisions that need to be made in this program?
    - What tasks are repeated?

    - How would you modularize the program, how many functions are you going to create, and what are they?
- What kind of bad input are you going to handle?

Based on your answers above, list the **specific steps or provide a flowchart** of what is needed to create. Be very explicit!!!

Program Testing:
Create a test plan with the test cases (bad, good, and edge cases). What do you hope to be the expected results?

- What are the good, bad, and edge cases for ALL input in the program? Make sure to provide enough of each and for all different inputs you get from the user.

Electronically submit your Design Doc (.pdf file!!!) by the design due date, on Canvas.

Program Code – Due Sunday,4/24/2022, 11:59pm on TEACH

Additional Implementation Requirements:

- Your user interface must provide clear instructions for the user and information about the data being presented
- Use of dynamic array is required.
- Use of command line arguments is required.
- Use of structs is required
- Your program must catch all required errors and recover from them.
- You are not allowed to use libraries that are not introduced in class.
- Your program should be properly decomposed into tasks and subtasks using functions. To help you with this, use the following:
    - Make each function do one thing and one thing only.
    - No more than 15 lines inside the curly braces of any function, including main(). Whitespace, variable declarations, single curly braces, vertical spacing, comments, and function headers do not count.
    - Functions over 15 lines need justification in comments.
    - Do not put multiple statements into one line.
- No global variables allowed (those declared outside of many or any other function, global constants are allowed).
- No goto function allowed
- You must not have any memory leaks
- You program should not have any runtime error, e.g. segmentation fault
- Make sure you follow the style guidelines, have a program header and function headers with appropriate comments, and be consistent.

Compile and Submit your assignment

When you compile your code, it is acceptable to use C++11 functionality in your program. In order to support this, change your `Makefile` to include the proper flag.
For example, consider the following approach (note the inclusion of **`-std=c++11`**):

```
g++ -std=c++11 <other flags and parameters>
```

In order to submit your homework assignment, you must create a **zip file** that contains your `.h, .cpp,` and `Makefile` files. This tar file will be submitted to TEACH . In order to create the tar file, use the following command:

```
zip assign1.zip catalog.h catalog.cpp run_wizard.cpp Makefile
```

Note that you are expected to modularize your code into a header file (.h), an implementation file (.cpp), and a driver file (.cpp).

You do not need to submit the txt files. The TA's will have their own for grading.

Remember to sign up with a TA to demo your assignment. The deadline of demoing this assignment without penalties is 5/6/2022.