# Program 1

**Due** Apr 11, 2021 by 11:59pm **Points** 90 **Submitting** a file upload **File Types** tar
**Available** until Apr 16, 2021 at 11:59pm

This assignment was locked Apr 16, 2021 at 11:59pm.

# The Magician Spellbook Catalog

## Problem Statement

The great magic university of Fakebills has been adding a lot of new spellbooks to their library lately. The school administration has hired you to create a spellbook catalog program that will make searching for spellbooks easier.
To simplify your task, the school has provided you with their spellbook information. These come in the form of a text file that file contains a list of Fakebill's spellbooks and their included spells, all of the information that your program will display.

## Requirements

### Command Line Argument:

When starting the program, the user will provide one command line argument. The 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 the name of an existing file the program should print out an error message and quit.

### Sorting and Printing:

Once the program begins, the user should be prompted with a list of different ways to display the spellbook and spell information. 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 pted 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.

Available Options:

- Sort spellbooks by number of pages: If the user picks this option the books must be sorted in ascending order by number of pages. 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. The spells with bubble as the effect should be listed first, followed by memory_loss, fire, poison, and death. 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 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.
- Quit: The program will exit.

Your program should continue sorting and printing/writing until the user chooses to quit. For the sorting functionality, you can write your own sorting function, or consider using C++'s built in sort function.

## 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.

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;
};
```

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", "bubble", "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). Note2: You must write the dynamic memory allocation functionality yourself (i.e. no using vectors).*

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

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

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 get_spellbook_data(spellbook*, int, ifstream &);
```

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

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

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

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

You need to implement a function that will delete all the memory that was dynamically allocated. You can choose the prototype. A possible example prototype includes the following:

```
void delete_spellbook_data(spellbook*, int);
```

## Required Input File Format

Your program must accommodate the file formats as specified in this section. The spellbooks.txt file has the following 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, and an example can be found **here (https://canvas.oregonstate.edu/courses/1810790/files/85706758/download?download_frd=1)** :

```
<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>
```

## 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.

```
$ ./catalog_prog spellbooks.txt

Which option would you like to choose?
Sort spellbooks by number of pages (Press 1):
Group spells by their effect (Press 2):
```

```
Sort spellbooks by average success rate (Press 3):
Quit (Press 4):
1
How would you like the information displayed?
Print to screen (Press 1)
Print to file (Press 2)
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?
Sort spellbooks by number of pages (Press 1):
Group spells by their effect (Press 2):
Sort spellbooks by average success rate (Press 3):
Quit (Press 4):
3
How would you like the information displayed?
Print to screen (Press 1)
Print to file (Press 2)
1
<Note: This example output is only intended to illustrate the program operation. Your values will be differen
t.>
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?
Sort spellbooks by number of pages (Press 1):
Group spells by their effect (Press 2):
Sort spellbooks by average success rate (Press 3):
Quit (Press 4):
3
How would you like the information displayed?
Print to screen (Press 1)
Print to file (Press 2)
2
Please provide desired filename: success_rate_list.txt
Appended requested information to file.

Which option would you like to choose?
Sort spellbooks by number of pages (Press 1):
Group spells by their effect (Press 2):
Sort spellbooks by average success rate (Press 3):
Quit (Press 4):
2
How would you like the information displayed?
Print to screen (Press 1)
Print to file (Press 2)
1
bubble Bubble_Beam
bubble Exploratory_Maneuver
memory_loss Space_Out
memory_loss Preliminary_Black_out
memory_loss Wacky_Dreams
fire Blasto_Strike
fire Beginning_Blast
poison Cthulhu_Venom
```

```
death Deathrock
death Nightmare
death Deadly_Details

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

## 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: catalog.cpp
** Author:
** Date:
** Description:
** Input:
** Output:
******************************************************/
```

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 tarred archive that contains your .h, .cpp, and Makefile files. This tar file will be submitted to Canvas. In order to create the tar file, use the following command:

```
tar –cvf assign1.tar catalog.h catalog.cpp prog.cpp Makefile
```

▶ 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.

## Grading

You are required to **meet with a TA** within two weeks of the assignment due date to demo and receive a grade. You can schedule a demo with a TA online using the link provided on the TAs page. You must use your OSU email to schedule (for FERPA reasons), or your appointment will be cancelled.

**Programming assignments that do not compile and run on the OSU ENGR servers will receive a grade of zero, with no exceptions.**

## Magician Spellbook Catalog

| Criteria | Ratings | Pts |
|---|---|---|
| Program Header / Good indentation & Use of whitespace / Function Documentation<br>At a minimum, header should contain author's name (2pts) and a description of the program. (2pts)<br>Is code easy for the TA to read? Conditional blocks of code should always be indented. (5pts) Every function contains it's own initial block comment (or multiple lines of comments prior to the function definition) that provides the reader with an explanation of the function's purpose (5pts). -1 pt for each function that is missing a header (up to 5 point penalty). | | 12 pts |
| All functions designed in a modular fashion / No global variables<br>-5 pts if there are any global variables used in the code. If functions are exceptionally long (greater than about 20 lines of actual code) this is a potential indication of poor modularity. -3 pts for each function that the TA concludes is poorly modularized. | | 10 pts |
| Command Line Functionality<br>(3 pts) Program displays error and terminates gracefully if exactly one command line parameter is not provided. (4pts) Program displays error and terminates gracefully if a non-existent file is specified. | | 7 pts |
| Struct Usage<br>Code defines/uses the spellbook and spell structs exactly as specified. -3 pts for any incorrect struct. It's okay if the code uses additional structs for other purposes. | | 6 pts |
| Memory Cleanup<br>The Valgrind LEAK SUMMARY should report: "definitely lost: 0 bytes in 0 blocks". Each "new" operation should have a corresponding delete operation. | | 3 pts |
| Program Functionality<br>Program does not crash during testing (e.g. segmentation fault or similar abrupt halts) | | 3 pts |
| Function prototypes<br>The following function prototypes are used (exactly as shown) and included in a .h file. -2pts for any function that doesn't match.<br>llbook * create_spellbooks(int); void get_spellbook_data(spellbook *, int, ifstream &); spell * te_spells(int); void get_spell_data(spell *, int, ifstream &); | | 8 pts |
| Correct TAR file<br>Program was submitted as a single TAR file that contains a working Makefile, an application .cpp file, a header .h file, and an implementation .cpp file. Exact filenames do not matter for the source code. | | 4 pts |
| Dynamic arrays of structs<br>Dynamically allocated arrays of structs are used for the spellbooks. | | 10 pts |

| Criteria | Ratings | Pts |
|---|---|---|
| Sort spellbooks by # pages<br>Program sorts and displays the spellbooks by number of pages. | | 6 pts |
| Group spells by effect<br>Program displays spells in groups (bubble, memory_loss, fire, poison, death) | | 6 pts |
| Display spellbooks (sorted by average success rate)<br>Program sorts spellbooks by the average success rate of the spells contained within. The sorted list is displayed on screen. | | 6 pts |
| Program writes requested output to file<br>The user is able to request that sorted output be redirected to a user-specified filename. The correct information is appended to that particular file. | | 6 pts |
| Program repeats<br>Program repeats until the user chooses to exit. | | 3 pts |
| | | Total Points: 90 |