**Program ONE / CSC1310**

redbox (a program to practice classes/objects)



# important dates

**Assignment Date**: Wednesday, August 28, 2019

**Due Date**: Wednesday, September 18, 2019 (by 11:59pm)

# Description of program – what does this program do?

You are writing a program for RedBox. Your RedBox program should allow users to load movie data from a file, save movies to a file, add a movie, remove a movie, display all movies, and remove all movies.

## Files you will submit in your zip file

This program contains multiple files as described below

* **Text.h** – header file for a Text class, which is your own version of the C++ String Class
* **Text.cpp** – source file containing function definitions required for the Text class
* **Movie.h** – header file for a Movie class, which has data and functions describing a single movie.
* **Movie.cpp** – source file containing the function definitions required for the Movie class.
* **MovieLibrary.h** – header file for a MovieLibrary class, which has data and functions describing a movie library (multiple movies)
* **MovieLibrary.cpp** – source file containing the function definitions required for the MovieLibrary class.
* **RedBox**.cpp – this is the driver for your program which uses the MovieLibrary class to create a MovieLibrary object.
* **three\_movies.txt** – text file **given to you** that contains data on three movies that you can use to test your program
* **crockett\_movie\_data.txt** – text file **given to you** that contains data on several movies that you can use to test your program

# specifications

## RedBox.cpp (driver)

Your program should ask the user “How many movies can this RedBox hold?” Once it reads in this number, it should dynamically allocate a MovieLibrary object, sending this number as an argument.

Then, the program should display a menu of six choices:

1. Load movies from a file.
2. Save movies to a file.
3. Add a movie.
4. Remove a movie.
5. Display all movies.
6. Remove ALL movies from this RedBox and end program.

If the user chooses to load a movie from a file, then ask the name of the file and call the MovieLibrary’s **loadMoviesFromFile** function, sending the filename as a string.

If the user chooses to save movies to a file, then ask the name of the fall and call the MovieLibrary’s **saveToFile** function, sending the filename as a string.

If the user chooses to add a movie, call the MovieLibrary’s **addMovieToArray** function.

If the user chooses to remove a movie, call the MovieLibrary’s **removeMovieFromArray** function.

If the user chooses to display all movies, call the MovieLibrary’s **displayMovies** function.

If the user chooses to remove all movies from the RedBox and end the program, then you should release (delete) the MovieLibrary object and end the program.

## MovieLibrary class

### Attributes

* moviesArray – a pointer to an array of pointers. Each pointer in the array should be able to point to (hold the memory address of) an individual Movie object. [Hint: you will need TWO stars to define this attribute. ]
* maxMovies – this is the maximum number of movies the RedBox can hold and is the size of the moviesArray.
* numMovies – this is the current number of movies actually pointed to in the moviesArray.

### Functions

Function name: **resizeMovieArray**

* Parameters: none
* Returns: none (void)
* Purpose: This function is called by **addMovieToArray** when the array size is not big enough to hold a new movie that needs to be added. The function makes the array twice as big as it currently is and then moves all the movie pointers to this new array.

Function name: **MovieLibrary** constructor

* Parameters: An integer containing the maximum size of the movie library
* Purpose: This function is automatically called when a MovieLibrary object is created and it creates a library of movies.
* Specification: The function will dynamically allocate an array of pointers to Movie objects based on the maximum size and will also set the current number of movies to zero.

Function name: **~MovieLibrary** destructor

* Purpose: This function is automatically called when the Movies object is destroyed. This releases the dynamically created individual movies and then deletes the array.

Function name: **addMovieToArray**

* Parameters: none
* Returns: none (void)
* Purpose: This function should be called when you need to add a single movie to the movie library.
* Specifications: It should ask the user for the movie title (read in as c-string, then dynamically create a Text object), movie year (integer), movie genre (read in as c-string, then dynamically create a Text object), movie rating (read in as c-string, then dynamically create a Text object), and IMDB star rating (a float between 0 and 10). Then it should dynamically allocate a new Movie object, sending the movie data just acquired from the user as arguments to the Movie constructor. Then, this function should check to see if numMovies is equal to maxMovies. If it is equal, then call the **resizeMovieArray** function. Then, it should assign this new movie to the correct pointer in the moviesArray. Last, it should increment numMovies.

Function name: **displayMovies**

* Parameters: none
* Returns: none (void)
* Purpose: This function should be called when the user wants to have all the movies in the library printed to the screen.
* Specifications: This function loops through the moviesArray and calls each Movie’s **printMovieDetails** function.

Function name: **displayMovieTitles**

* Parameters: none
* Returns: none (void)
* Purpose: This function should be called when you want to print only the movie titles out of the movie library (when a user wants to remove a movie from RedBox).
* Specifications: This function should loop through the moviesArray, retrieve the Movie’s title by calling the Movie’s **getMovieTitle** function, and then printing out the title by calling the Text’s **displayText** function.

Function name: **loadMoviesFromFile**

* Parameters: A pointer to a character (c-string or string literal argument) containing the filename
* Returns: none (void)
* Purpose: This function should be called when the user wants to read movie data from a file and add the movies to the movie library. The file must have data in the following order: title, year, genre, rating, IMDB star rating.
* Specifications: This function will use a loop to read the contents of the file until reaching the end of file. For each movie, it will read the title in with a c-string and then dynamically allocate a Text to hold the title. Then it will read in the movie year. Then it will read in the movie genre with a c-string and then dynamically allocate a Text to hold the genre. Then it will read in the movie rating with a c-string and then dynamically allocate a Text to hold the rating. Then it will read in the IMDB star rating. Then, it will dynamically create a new Movie object, sending the movie data just acquired from the user as arguments to the Movie constructor. Then, this function should check to see if numMovies is equal to maxMovies. If it is equal, then call the **resizeMovieArray** function. Then, it should assign this new movie to the correct pointer in the moviesArray. Then, it should increment numMovies. Then, it should print the title of the movie and say “ was added to the movie library!” This should happen for each movie read from the file. At the end of the function, it should print out how many movies were read from the file & added to the library.

Function name: **removeMovieFromArray**

* Parameters: none
* Returns: none (void)
* Purpose: This function should be called when the user wants to remove one single movie from the movie library. The movie to be removed must is identified by the user and then removed.
* Specifications: This function should first make sure that the number of movies is at least 1. if not, it should print that there must always be one movie in the library and the function should end. Then, the function should call the **displayMovieTitles** function to print all the movie titles. Then, ask the user to choose a movie to remove between 1 & numMovies. Once the user identifies the movie, your program should print that the movie title has been successfully deleted. Then, release the dynamically allocated space for this movie and move all array elements in movieArray back 1 starting with this deleted movie’s element. Last, decrement numMovies.

Function name: **saveToFile**

* Parameters: A pointer to a character (c-string or string literal argument) containing the filename
* Returns: none (void)
* Purpose: This function should be called when the user wants to print all the movie data from the movie library to a file. The data is printed in the following order (one piece of data per line): title, year, genre, rating, IMDB star rating.
* Specifications: Use a loop to go through the movieArray and call the Movie’s **printMovieDetailsToFile** function, sending the name of the file to be printed to. Then, close the file and print a confirmation that all movies have been printed to the filename.

## movie class

### Attributes

* movieTitle – a pointer to a Text object, which will hold the title of the movie
* movieYear – an integer representing the year the movie was released
* movieGenre – a pointer to a Text object, which will hold the genre of the movie
* movieRating – a pointer to a Text object, which will hold the rating of the movie
* movieNumStars – a floating point number between 0 and 10 representing the number of stars in IMDB

### Functions

Function name: **Movie** constructor

* Parameters:
  + A pointer to a Text variable, containing the title of the movie
  + An integer containing the year the movie was released
  + A pointer to a Text variable, containing the genre of the movie
  + A pointer to a Text variable, containing the rating of the movie
  + A float containing the IMDB rating of the movie (out of 10 stars)
* Purpose: This function should be called when all movie information is known and it will create a new movie with this information.
* Specifications: initialize all attributes to the arguments sent to this function.

Function name: **~Movie** destructor

* Purpose: This function is automatically called when a Movie object is destroyed. This function releases the dynamically allocated text arrays in the Movie.
* Specifications: Release the dynamically allocated space for the movieTitle, movieGenre, and movieRating.

Function name: **printMovieDetails**

* Parameters: none
* Returns: none (void)
* Purpose: This function should be called when the user wants to print ALL the movie information to the screen.
* Specifications: Print the title, year, genre, rating & number of stars. Remember that in order to print the Text objects, you must call their **displayText** function.

Function name: **printMovieDetailsToFile**

* Parameters: a file stream object (sent by reference)
* Returns: none (void)
* Purpose: This function should be called when the user wants to print ALL the movie information to the file.
* Specifications: Print the title, year, genre, rating & number of stars to the filename that was sent to this function. In order to print the Text objects to the file, you must first retrieve the c-string attribute (calling the **getText** function) from this Text, and then you can print it to the file.

Function name: **getMovieTitle** (accessor function)

* Parameters: none
* Returns: a pointer to the Text object containing the movie title

## Text class

### Attributes

* textArray – a pointer to a constant character array
* textLength – an integer representing the number of characters stored in the textArray

### Functions

Function Name: **Text** (constructor)

* Parameters: Send a pointer to a constant character array or a string literal to this function
* Purpose: called automatically when Text object is created, dynamically allocates a character array which contains the character array passed to the function.
* Specifications: dynamically allocate a new character string the size of the string passed to this function plus one (for the null terminator). Then, copy the text sent as an argument to this constructor to the new dynamically allocated c-string. Then, set the textArray attribute to this newly created c-string.

Function Name: **~Text** (destructor)

* Purpose: release dynamically allocated memory for the c-string in the Text object
* Specifications: release the memory for the c-string pointed to by textArray

Function Name: **displayText**

* Parameters: none
* Returns: none (void)
* Purpose: print the c-string (textArray) to the screen

Function Name: **getText** (accessor function)

* Parameters: none
* Returns: pointer to a constant character array

Function Name: **getLength** (accessor function)

* Parameters: none
* Returns: the length of the string

# Sample Output

How many movies can this RedBox hold?

2

What would you like to do?

1. Load movies from file.

2. Save movies to a file.

3. Add a movie.

4. Remove a movie.

5. Display all movies.

6. Remove ALL movies from this RedBox and end program.

CHOOSE 1-6: 1

What is the name of the file? (example.txt): three\_movies.txt

Interstellar was added to the movie library!

The Green Mile was added to the movie library!

Resizing movieArray from 2 to 4.

Back to the Future was added to the movie library!

3 movies were read from the file and added to your movie library.

What would you like to do?

1. Load movies from file.

2. Save movies to a file.

3. Add a movie.

4. Remove a movie.

5. Display all movies.

6. Remove ALL movies from this RedBox and end program.

CHOOSE 1-6: 5

----------MOVIE 1----------

Movie Title: Interstellar

Year Released: 2014

Genre: Adventure, Drama, Sci-Fi

Rating: PG-13

Number of Stars: 8.6

----------MOVIE 2----------

Movie Title: The Green Mile

Year Released: 1999

Genre: Crime, Drama, Fantasy

Rating: R

Number of Stars: 8.5

----------MOVIE 3----------

Movie Title: Back to the Future

Year Released: 1985

Genre: Adventure, Comedy, Sci-Fi

Rating: PG

Number of Stars: 8.5

What would you like to do?

1. Load movies from file.

2. Save movies to a file.

3. Add a movie.

4. Remove a movie.

5. Display all movies.

6. Remove ALL movies from this RedBox and end program.

CHOOSE 1-6: 3

MOVIE TITLE: Up

MOVIE YEAR: 2014

MOVIE GENRE: Family

MOVIE RATING: PG

STAR RATING (out of 10): 8.3

What would you like to do?

1. Load movies from file.

2. Save movies to a file.

3. Add a movie.

4. Remove a movie.

5. Display all movies.

6. Remove ALL movies from this RedBox and end program.

CHOOSE 1-6: 4

Choose from the following movies to remove:

MOVIE 1: Interstellar

MOVIE 2: The Green Mile

MOVIE 3: Back to the Future

MOVIE 4: Up

Choose a movie to remove between 1 & 4: 2

The movie "The Green Mile" has been successfully deleted.

What would you like to do?

1. Load movies from file.

2. Save movies to a file.

3. Add a movie.

4. Remove a movie.

5. Display all movies.

6. Remove ALL movies from this RedBox and end program.

CHOOSE 1-6: 5

----------MOVIE 1----------

Movie Title: Interstellar

Year Released: 2014

Genre: Adventure, Drama, Sci-Fi

Rating: PG-13

Number of Stars: 8.6

----------MOVIE 2----------

Movie Title: Back to the Future

Year Released: 1985

Genre: Adventure, Comedy, Sci-Fi

Rating: PG

Number of Stars: 8.5

----------MOVIE 3----------

Movie Title: Up

Year Released: 2014

Genre: Family

Rating: PG

Number of Stars: 8.3

What would you like to do?

1. Load movies from file.

2. Save movies to a file.

3. Add a movie.

4. Remove a movie.

5. Display all movies.

6. Remove ALL movies from this RedBox and end program.

CHOOSE 1-6: 6

GOODBYE!

# VS Code tasks.json

My tasks.json in VS Code looks like this:

{

"version": "2.0.0",

"tasks": [

{

"label": "build RedBox",

"type": "shell",

"command": "g++",

"args": [

"-std=c++11", //use the C++ language dialect

"-Wall", //enables all warnings to be printed during compile

"-g", //compile with debugging flags

"-o", //output file name should be ...

"RedBox", //output file name

"RedBox.cpp",

"MovieLibrary.cpp",

"Movie.cpp",

"Text.cpp"

],

"group": {

"kind": "build",

"isDefault": true

}

}

]

}

