PROGRAM ONE / CSC 1310

VIDEO GAME LIBRARY

IMPORTANT DATES

Assignment Date: Tuesday, Sep 5, 2023
Due Date: Monday, Sep 25, 2023

DESCRIPTION OF PROGRAM – WHAT DOES THIS PROGRAM DO?

You are writing a program to keep track of a library of video games. Your program should allow users to load a list of video game data from a file, save video game data to a file, add a video game, remove a video game, display all video games in the library, and remove all video games.

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
- **VideoGame.h** – header file for a VideoGame class, which has data and functions describing a single video game.
- **VideoGame.cpp** – source file containing the function definitions required for the VideoGame class.
- **VideoGameLibrary.h** – header file for a VideoGameLibrary class, which has data and functions describing a video game library (multiple games)
- **VideoGameLibrary.cpp** – source file containing the function definitions required for the VideoGameLibrary class.
- **Program1.cpp** – this is the driver for your program which uses the VideoGameLibrary class to create a VideoGameLibrary object.
- **three_ games.txt** – text file **given to you** that contains data on three video games that you can use to test your program
- **games_list.txt** – text file **given to you** that contains data on twenty video games that you can use to test your program

SPECIFICATIONS

DOCUMENTATION

Documentation is part of your grade on all four of the major programs. Your program1.cpp file should have comments at the top that include your name, the course and section number, the due date, and the name of your partner if you're working in pairs. **If you do not include your partner's name your assignment may be flagged for cheating**.

Every class and function should have a high-level description of the purpose of that class or function, and there should be inline comments throughout the body of your program to explain how your program is operating. **You will have points taken off for not commenting your code**.

PROGRAM1.CPP (DRIVER)

Your program should ask the user "How many video games can your library hold?" Once it reads in this number, it should dynamically allocate a VideoGameLibrary object, sending this number as an argument.

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

```
What would you like to do?
1.  Load video games from file.
```

```
2.  Save video games to a file.
3.  Add a video game.
4.  Remove a video game.
5.  Display all video games.
6.  Remove ALL video games from this library and end program.
CHOOSE 1-6:
```

If the user chooses to load a video game from a file, then ask the name of the file and call the VideoGameLibrary's **loadVideo gamesFromFile** function, sending the filename as a string.

If the user chooses to save video games to a file, then ask the name of the file and call the VideoGameLibrary's **saveToFile** function, sending the filename as a string.

If the user chooses to add a video game, call the VideoGameLibrary's **addVideoGameToArray** function.

If the user chooses to remove a video game, call the VideoGameLibrary's **removeVideoGameFromArray** function.

If the user chooses to display all video games, call the VideoGameLibrary's **displayVideoGames** function.

If the user chooses to remove all video games from the library and end the program, then you should release (delete) the VideoGameLibrary object and end the program.

VIDEOGAMELIBRARY CLASS

ATTRIBUTES

- **videoGamesArray** – 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 Video game object.
  [Hint: you will need TWO stars to define this attribute.]
- **maxGames** – this is the maximum number of video games the library can hold and is the size of the video gamesArray.
- **numGames** – this is the current number of video games actually pointed to in the videoGamesArray.

FUNCTIONS

Function name: **resizeVideoGameArray**

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

Function name: **VideoGameLibrary** constructor

- Parameters: An integer containing the maximum size of the video game library
- Purpose: This function is automatically called when a VideoGameLibrary object is created and it creates a library of video games.
- Specifications: The function will dynamically allocate an array of pointers to VideoGame objects based on the maximum size and will also set the current number of video games to zero.

Function name: **~VideoGameLibrary** destructor

- Purpose: This function is automatically called when the VideoGame object is destroyed. This releases the dynamically created individual video games and then deletes the array.
- Specifications: Prints a message "VideoGameLibrary destructor: Released memory for each game in the video game array and the array itself."

Function name: **addVideoGameToArray**

- Parameters: none
- Returns: none (void)
- Purpose: This function should be called when you need to add a single video game to the video game library.
- Specifications: It should ask the user for the video game title (read in as c-string, then dynamically create a Text object), developer (read in as c-string, then dynamically create a Text object), publisher (read in as c-string, then dynamically create a Text object), and release year (integer). Then it should dynamically allocate a new **VideoGame** object, sending the video game data just acquired from the user as arguments to the **VideoGame** constructor. Then, this function should check to see if numGames is equal to maxGames. If it is equal, then call the **resizeVideoGameArray** function. Then, it should assign this new video game to the correct pointer in the videoGamesArray. Last, it should increment numGames.

Function name: **displayVideoGames**

- Parameters: none
- Returns: none (void)
- Purpose: This function should be called when the user wants to have all the video games in the library printed to the screen.
- Specifications: This function loops through the video gamesArray and calls each Video game's **printVideoGameDetails** function.

Function name: **displayVideoGameTitles**

- Parameters: none
- Returns: none (void)
- Purpose: This function should be called when you want to print only the video game titles out of the video game library (when a user wants to remove a video game from library).
- Specifications: This function should loop through the video gamesArray, retrieve the Video game's title by calling the Video game's **getVideoGameTitle** function, and then printing out the title by calling the Text's **displayText** function.

Function name: **loadVideoGamesFromFile**

- 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 video game data from a file and add the video games to the video game library. The file must have data in the following order: title, developer, publisher, year.
- Specifications: This function will use a loop to read the contents of the file until reaching the end of file. For each video game, 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 video game developer with a c-string and then dynamically allocate a Text to hold the publisher. Then it will read in the video game year. Then, it will dynamically create a new VideoGame object, sending the video game data just acquired from the user as arguments to the VideoGame constructor. Then, this function should check to see if numGames is equal to maxGames. If it is equal, then call the **resizeVideoGameArray** function. Then, it should assign this new video game to the correct pointer in the video gamesArray. Then, it should increment numGames. Then, it should print the title of the video game and say " was added successfully" This should happen for each video game read from the file. At the end of the function, it should print out how many video games were read from the file & added to the library.

[Note: The temporary c-string used to load in data should be able to hold a very large string – 10000 characters]

Function name: **removeVideoGameFromArray**

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

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 video game data from the video game library to a file. The data is printed in the following order (one piece of data per line): title, developer, publisher, year.
- Specifications: Open the file with the filename that was sent to this function. Then, use a loop to go through the video gameArray and call the Video game's **printVideoGameDetailsToFile** function, sending the file stream object to be printed to. Then, close the file and print a confirmation that all video games have been printed to the filename.

VIDEOGAME CLASS

ATTRIBUTES

- title – a pointer to a Text object, which will hold the title of the video game
- developer – a pointer to a Text object, indicating who made the video game
- publisher – a pointer to a Text object, indicating the publisher/distributor of the video game
- year – an integer representing the year the video game was released

Function name: **VideoGame** constructor

- Parameters:
  - A pointer to a Text variable, containing the title of the video game
  - A pointer to a Text variable, containing the developer of the video game
  - A pointer to a Text variable, containing the publisher of the video game
  - An integer containing the year the video game was released
- Purpose: This function should be called when all video game information is known and it will create a new video game with this information.
- Specifications: initialize all attributes to the arguments sent to this function.

Function name: **~VideoGame** destructor

- Purpose: This function is automatically called when a Video game object is destroyed. This function releases the dynamically allocated text arrays in the Video game.
- Specifications: Release the dynamically allocated space for the video game title, developer, and publisher.
- Prints a message: "VideoGame destructor: Released memory for VideoGame object."

Function name: **printVideoGameDetails**

- Parameters: none
- Returns: none (void)

- Purpose: This function should be called when the user wants to print ALL the video game information to the screen.
- Specifications: Print the title, developer, publisher, & year. Remember that in order to print the Text objects, you must call their **displayText** function.

Function name: **printVideoGameDetailsToFile**

- 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 video game information to the file.
- Specifications: Print the title, developer, publisher, & year to the file stream object 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: **getVideoGameTitle** (accessor function)

- Parameters: none
- Returns: a pointer to the Text object containing the video game 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 (using the strcpy function from the cstring library). 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
- Prints a message: “Text destructor: Released memory for textArray.”

Function Name: **displayText**

- Parameters: none
- Returns: none (void)
- Purpose: print the c-string (textArray) to the screen. (c++ will let you just cout a char array)

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

REFLECTION

Along with the program, You should also submit a reflection on the assignment between half a page and one page in length. Tell me about what went well and what didn't. What parts of your submission do you feel confident about, and which parts do you feel like you still don't have a good understanding of? If there are parts of the instructions you thought were unclear, this is a good opportunity to point that out to me so that I can improve my assignments in future semesters (I am very interested in your feedback!). If you worked with a partner on your assignment, please also write a brief evaluation of your partner and how fairly you think the work was distributed between the two of you.