Project 0 Design Document

Bill Yao (b29yao)

1. Overview of classes

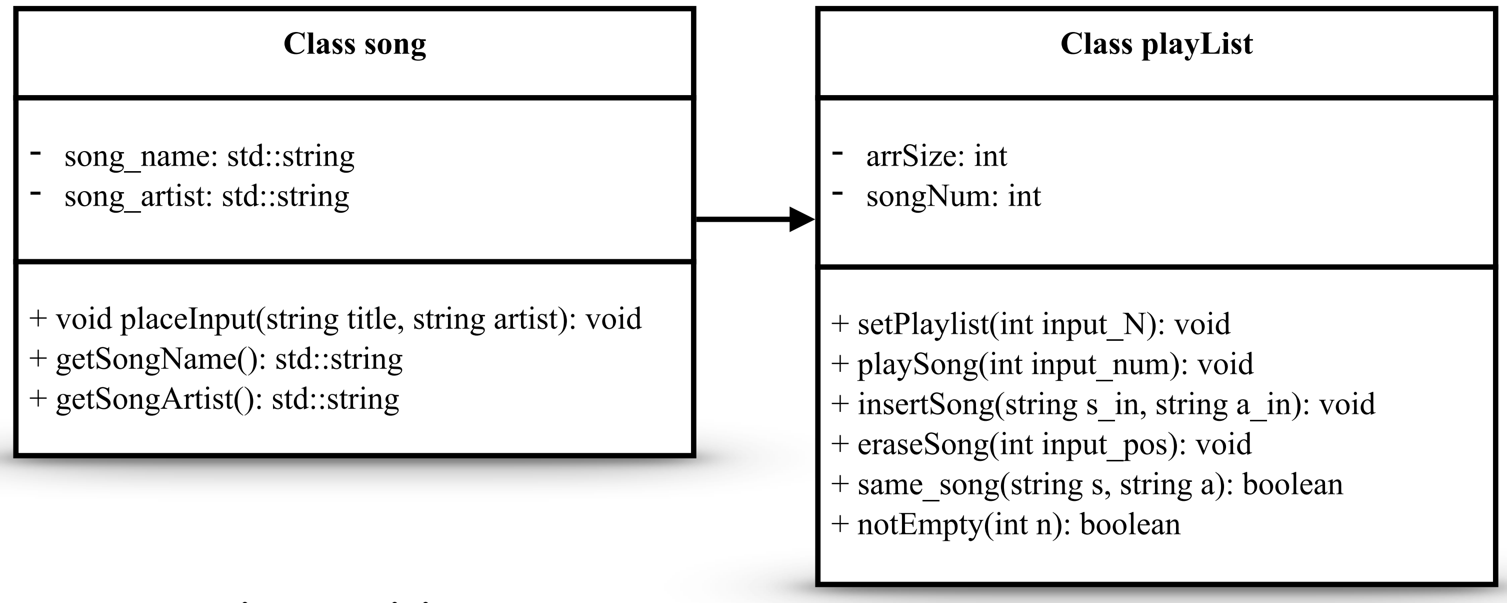
In this project, I have designed two classes to implement the functionality of the program.

1). The first class "song" stores the title and artist of the input song separately into two member variables "song_name" and "song_artist", also it possesses three member functions to update and get the stored song data. Objects will be created based on this class and saved inside the array.

2). The second class "playlist" contains the major member functions and stores the song object into a dynamically allocated "songList" array. It has two major private member variables, "arrSize" and "songNum", which we utilize to store the input and simultaneously cost the testing. Two helper functions are used to validate the inputs and check the conditions, while four other functions are used to set, add songs, play songs, and erase songs at different index positions within the array.

3). Class playlist can be considered as a container of the object from the Class song and each entry into the array would take the input data from the users.

2. UML class diagram



Class song
- song_name: std::string
- song_artist: std::string
+ void placeInput(string title, string artist): void
+ getSongName(): std::string
+ getSongArtist(): std::string
Class playList
- arrSize: int
- songNum: int
+ setPlaylist(int input_N): void
+ playSong(int input_num): void
+ insertSong(string s_in, string a_in): void
+ eraseSong(int input_pos): void
+ same_song(string s, string a): boolean
+ notEmpty(int n): boolean

3. Design Decisions

For Class song, I decided to use a default constructor, since no variables need to hold a default value and objects don't need to be implemented in the constructor. However, I do need to clear up the member variable in the destructor. There are no operators that we used to override. When setting the song name and artist, we did not pass by reference on any parameters since they will potentially experience add and erase functions and we don't need to use reference or set it to constant.

For Class playList, we did dynamically allocate a default array with size 0 in the constructor. Also, we do need to delete the array to free up the memory location in the destructor and set variables to a default value. In this class, we decided to create two helper functions "notEmpty" and "same_song", since these two conditions need to be examined throughout the other four functions. Therefore we use the helper function to reduce the amount of tedious repetitive code in other functions.

4. Testing Strategy

When testing the entire program, we can divide it into multiple simple testing tasks that would internally back up each other. So we set up the test cases for each member function and make sure they all work properly, then we wrap them up for overall testing.

Setting: To test the functionality of setting a playlist, we can simply substitute in a command like "m 3" to see if the "success" message is displayed in the terminal. Also, we might want to use the play function to check the number of the entry in the array, in order to reassure we set the correct size of the playlist.

Play: To test the functionality of playing a song from the list, we would enter commands like "p 1" to check if the song is correctly played. Also, we might want to enter some entry number that contains a null value or exceeds the capacity of the array, which should generate the error message "can not play" in the terminal. By choosing several scenarios, we can make sure the play function works

Insert: To validate the insertion, we need three major restriction inputs that contain "Muskrat Love;Captain and Tennille", "My Heart Will Go On" by any artist, and the same song from the existing playlist. Make sure all these cases would compute error messages, as well as a full playlist which can no longer take in any song to the list. Other than that, we can use "play" function to make sure every inserted song is placed at the end of the current playlist.

Erase: Make sure the erase entry is not null and does not go beyond the array capacity. Then we just execute erase command at different entries throughout the playlist and check the deletion with "play" function.

5. Performance considerations

playSong(int input_num): O(1), because the index is given through the parameter and it can access the data stored in the array with a unit of time.

insertSong(string s_in, string a_in): O(n), because we need to iterate through the array to make sure no duplications and restricted inputs, therefore we require a linear amount of time.

eraseSong(int input_pos): O(n), because we need to move every song behind the deleted position up by one index.