Final Project

For this project I decided to use vectors to favor from their indexing features. This helped me tremendously when implementing any function where I had to play the previous/next song/playlist.

My biggest takeaway from completing this project was learning how to create a project from scratch. I decided to use visual code this time but I had never done anything similar before on my own. I know we used VS code for our homework but the files were already provided to me. Creating my own files, learning how #include works and initializing my functions will be a great skill to have moving forward. I can now start creating my own personal projects which is something I look forward to.

While it was fun to create, I also faced many challenges. I think the biggest one was implementing my shuffle function where I had to rearrange all my playlists. I had no idea how to make this work but after researching for a while and many failed attempts to make it work, I found a template that seems to be working. Even though it took me a while to find, it is great practice for the future since part of being a computer science student is learning to solve problems on your own and utilizing resources available to you.

Walkthrough

I first started by creating three separate classes, song, playlist and jukebox. I created three files for each where I initialized them. These are the song.h, playlist.h and jukebox.h files. After setting that up, I moved on to creating my functions which are the following:

Insert();

```
#include "jukebox.h"

void Jukebox::insert(playlist& playlist){
    jukebox_playlist.push_back(playlist);
}
```

I was not asked to create an insert function but I thought it would be nice since we can use it to add a playlist into our jukebox. This is a very simple function that will push_back any playlist into our jukebox.

play();

```
//Play
void Jukebox::play(){
    cout<< "Now playing: "<<jukebox_playlist[current_playlist_index].content[current_song_index].get_song_name() <<
        " by " << jukebox_playlist[current_playlist_index].content[current_song_index].get_artist_name() << endl;
}
```

This is how I implemented my play function. It outputs “now playing: “ which will show the user which song and by which artist. I did this by indexing my jukebox_playlist at the current playlist index. I then used our content vector where the songs are stored and used my ‘get_song_name_ function to get the song name. There is then a text ‘by’ followed by the artist name which was done in the same way as above. This will be my most used function as it’ll be called after most of my other functions to display the song and artist information to the user.

SkipPreviousTrack(playlist playlist);

```
// previous song
void Jukebox::SkipPreviousTrack(playlist& playlist) {
    current_song_index = current_song_index - 1;
    play();

    }

}
```

Here I am skipping to the previous track. This was done by decrementing my current_song_index by 1. I then called our play function which will use this song index as well as the current playlist index to get out song name and artist.

SkipToNextTrack(playlist playlist);

```
//next song
void Jukebox::SkipToNextTrack(playlist& playlist){
    current_song_index++;
    play();

    }
}
'
```

This is a similar function to the previous one, the difference is that I am incrementing my song_index to access the element in front of it while then using my play() function to display this information.

pause();

```
//pause
void Jukebox::pause(){
    cout<<"paused"<<endl;
}
```

For pause I am just passing the ‘paused’ message to inform the user the playlist is paused. Since I am not passing actual music, I am “pretending” to pause.

skiptoFirstTrack();

```
//first song
void Jukebox::skiptoFirstTrack(){
    current_song_index = 0;
    play();

}
```

This function will skip to the first song of the playlist. I set the current_song_index to 0 and called my play() function to display the information.

SkipToFirstTrackPrev();

```
//first song, previous playlist
//Skip to first track of previous playlist, if it is currently playing the first track of a playlist
void Jukebox::SkipToFirstTrackPrev(){
    if(current_song_index == 0){
        current_playlist_index--;
        current_song_index = 0;
    }
    play();
}
```

This function is meant to skip to the first song of the previous playlist if we are currently playing the first song of a playlist. I created an if statement to test if we were on our first song. I then decremented our current_playlist_index to move to our previous playlist and set our current_song_index to 0 to access our first song. I called my play() function to display the rest of the song information;

SkipToFirstTrackNext();

```
//first song, next playlist
void Jukebox::SkipToFirstTrackNext(){
    current_playlist_index++;
    current_song_index = 0;
    }
    play();
}
```

This function is meant to skip to the first song of the next playlist. To achieve this, I incremented our current_playlist_index by one and set our current_song_index to 0. I called my play() function to display the rest of the information.

rearrange();

```
//move around playlists order
void Jukebox::rearrange(){
    //gets random value based on system time
    unsigned seed = chrono::system_clock::now().time_since_epoch().count();

    shuffle(jukebox_playlist.begin(),jukebox_playlist.end(), default_random_engine(seed));

    current_playlist_index = 0;
    current_song_index = 0;
    play();

}
```

This function is meant to randomly rearrange the order of our playlist. I had to do some research for this one since I initially did not know how this could work. I ended up utilizing a template I

found on geeksforgeeks that would get a random value based on my system time(hope this is okay!) and assigned this to my variable seed. I then used the standard function “shuffle” and passed our seed variable to generate a random order to our playlists.

ToggleRepeat();

```
//autoplay
void Jukebox::ToggleRepeat(){
    jukebox_playlist[current_playlist_index] = jukebox_playlist[0];
    jukebox_playlist[current_playlist_index].content[current_song_index]= jukebox_playlist[current_playlist_index].content[0];
    play();

}
```

This function would sort of create a loop using our playlists. After the last song in the last playlist is done, it plays the first song in the first playlist. I decided to implement this as an autoplay function that can be called at the end of our last song in our last playlist to restart it.

Menu:

```
Jukebox my_jukebox;
cout<<"My_Jukebox App Menu" << '\n' << "Please select from the following options or Enter 0 to quit" << '\n' <<
"1: Play "<< '\n' << "2: Pause"<<'\n'<<"3: Play previous track" << '\n' << "4: Play next track"
<< '\n' <<  "5: Play 1st song of playlist" << '\n' <<
"6: Play 1st song of previous playlist" << '\n' << "7: play 1st song of next playlist" << '\n' <<
 "8: Random reassange pof playlists" << '\n' << "9: Autoplay(repeat) " << '\n' << endl;
int input;
cin>>input;

while(true){
if(input == 0){
    cout<<"Thank you for using My_Jukebox App"<<endl;

 }
else if(input == 1){
    cout<<"Playing" <<endl;
    //my_jukebox.play();
}
else if(input == 2){
    cout<<"Paused"<<endl;
    //my_jukebox.pause();
}
else if(input == 3){
    cout<<"Playing previous song"<<endl;
    //my_jukebox.SkipPreviousTrack(playlist);

}

else if(input == 4){
    cout<<"Playing next song"<<endl;
    //my_jukebox.SkipToNextTrack();
}
else if(input == 5){
```

My menu looks like this. I asked the user to make a selection with the option to exit the program at any time if they enter the number 0. Otherwise, there are a total of 9 options to choose from, each option representing one of our functions. Here is an example of how this might look if you want to exit:

```
  7: play 1st song of next playlist
  8: Random reassange pof playlists
  9: Autoplay(repeat)

  8
  Rearranging playlist order
● ledyurena@Ledys-MacBook-Air DS_final % g++ -std=c++11 -o jukebox main.cpp jukebox.cpp playlist.cpp song.cpp
  ./jukebox
  My_Jukebox App Menu
  Please select from the following options or Enter 0 to quit
  1: Play
  2: Pause
  3: Play previous track
  4: Play next track
  5: Play 1st song of playlist
  6: Play 1st song of previous playlist
  7: play 1st song of next playlist
  8: Random reassange pof playlists
  9: Autoplay(repeat)

  0
  Thank you for using My_Jukebox App
○ ledyurena@Ledys-MacBook-Air DS_final %
```

If you want to play next track:

```
  38        cout<<"Playing previous song"<<endl;
  PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS
  7: play 1st song of next playlist
  8: Random reassange pof playlists
  9: Autoplay(repeat)

  0
  Thank you for using My_Jukebox App
● ledyurena@Ledys-MacBook-Air DS_final % g++ -std=c++11 -o jukebox main.cpp jukebox.cpp playlist.cpp song.cpp
  ./jukebox
  My_Jukebox App Menu
  Please select from the following options or Enter 0 to quit
  1: Play
  2: Pause
  3: Play previous track
  4: Play next track
  5: Play 1st song of playlist
  6: Play 1st song of previous playlist
  7: play 1st song of next playlist
  8: Random reassange pof playlists
  9: Autoplay(repeat)

  4
  Playing next song
○ ledyurena@Ledys-MacBook-Air DS_final %
```

This is how I would assign each function to our menu (commented code):

```
while(true){
if(input == 0){
    cout<<"Thank you for using My_Jukebox App"<<endl;

 }
else if(input == 1){
    cout<<"Playing" <<endl;
    //my_jukebox.play();
}
else if(input == 2){
    cout<<"Paused"<<endl;
    //my_jukebox.pause();
}
else if(input == 3){
    cout<<"Playing previous song"<<endl;
    //my_jukebox.SkipPreviousTrack(playlist);

}

else if(input == 4){
    cout<<"Playing next song"<<endl;
    //my_jukebox.SkipToNextTrack();
}
else if(input == 5){
    cout<< "Playing 1st song"<<endl;
    //my_jukebox.skiptoFirstTrack();
}
else if(input == 6){
    cout<<"Playing 1st song of previous playlist"<<endl;
    //my_jukebox.SkipToFirstTrackPrev();
}
else if( input == 7){
    cout<<"Playing 1st song of next playlist"<<endl;
    //my_jukebox.SkipToFirstTrackNext();
}
else if(input == 8){
    cout<< "Rearranging playlist order"<<endl;
    //my_jukebox.rearrange();
}
```

Ln 37, Col 25 Spaces: 4 UTF-8 LF