Movie Manager C++ Project

Ali Savari

230208874

In this project I have created a small Movie manager with C++.

The code performs simple tasks and the operations of listing, adding and deleting a movie inside a Movie Manager.

The folder contains 3 files, movie.h , movie-manager.h, and main.cpp

The movie manager is a simple data structure to help manage the movies inside a single containing class.

I have started by creating a simple data structure class for Movie, a movie normally contains the following data, title, genre and release year.

```
class Movie
{
public:
  int id;
  string title;
  string genre;
  int release_year;
};
```

This file is called `movie.h` and will be used by `movie-manager.h`

MovieManager

```
#include "movie.h"
```

Inlcuding the Movie Class source file

```
class MovieManager
{
public:
  vector<Movie> movies;
```

The vector<Movies> movies is an advanced version of the old classic C++ array. Its a part of the standard C++ library and it comes built-in with the language itself, it helps with using arrays, it has dynamic sizing (so you don't need to worry about index) and it handles memory allocation by itself. It contains a lot of methods like push_back() and find() to help developers write cleaner and less code. Here i just defined that I need the variable movies to be of type vector of type Movie.

Adding

```
void add(string title, string genre, int release_year)
  {
    Movie m;
    m.id = this->movies.size() + 1;
    m.title = title;
    m.genre = genre;
    m.release_year = release_year;
    this->movies.push_back(m);
  }
```

As the code itself is showing, Im just taking 3 arguments and creating a simple Movie object by passing the values. For ID of the movie I handled it with relative to the size of the movie list. So the method this->movies.size() will return the size of the list. Finally I add the newly created movie into the movie list by using the push_back() vector method.

Deleting

```
void del (int id) {
    int i = 0;
    for(const auto &m: this->movies){
      if(m.id == id){
        cout << "Movie " << m.title << " Deleted!" << endl;
        this->movies.erase(this->movies.begin() + i);
        break;
      }
      i++;
    }
  }
```

This function’s purpose is to delete a movie based on a given ID. I'm taking an int argument and by using a for loop I'm searching for the specific movie. And again, by using the vector methods it's much easier to delete an element inside a list. The auto keyword here helps the compiler to deduce the type of variable.

Searching

```
void search(const string text)
  {
    vector<Movie> foundMovies;
    for (const auto &movie : this->movies)
    {
      size_t found = movie.title.find(text);
      if (found != string::npos)
      {
        foundMovies.push_back(movie);
      }
    }
    if (foundMovies.size() > 0)
    {
      cout << "Found Movies : " << endl;
      this->print_movies(foundMovies);
    }
    else
    {
      cout << "No Movies Found with this name!" << endl;
    }
  }
```

In this method Im trying to search through the list of movies in order to find the specified movie. This method takes string text and by using the vector method .find() I can search inside a given string, for Example “The” can yield 2 results in our code. Thats why I have created a secondary vector to store the result and pass it to another method for printing. If there’s no results simply print message and exit.

Printing

```
void print_movies(vector<Movie> &printList)
  {
    for (const auto &movie : printList)
    {
      cout << "---" << endl;
      cout << "ID: " << movie.id << endl;
      cout << "Title: " << movie.title << endl;
      cout << "Genre: " << movie.genre << endl;
      cout << "Release Year: " << movie.release_year << endl;
      cout << "" << endl;
    }
  }

  void list_movies()
  {
    cout << "Here's the list of available movies" << endl;
    cout << "Movies listed in DB: " << this->movies.size() << endl;
    this->print_movies(this->movies);
  }
```

The 2 functions are simply printing the list of the movies in a readable fashion

Main.cpp

```
#include "movie-manager.h"
```

Including the MovieManager Class source file

```
void ignore_newline(){
  cin.ignore(numeric_limits<streamsize>::max(), '\n');
}

bool get_valid_int(int &num)
{
  cin >> num;
  if (cin.fail())
  {
    cin.clear();
    ignore_newline();
    return false;
  }
  return true;
}
```

I have added 2 custom util functions in order to fix the bug of cin input in C++. When you take in string input in C++, whenever you input a space, the stream closes and the code execution continues. Thats why the function ignore_newline() helps in reading a whole line of strings including white spaces. The method get_valid_int() helps in the issue of automatic input type conversion by C++ cin.

```
int main()
{
  cout << endl
       << ".:: Welcome to Movie Manager! ::." << endl
       << endl;
  MovieManager manager;
```

```cpp
  int choice;
  int id_del;
  string search_name;
  string add_name;
  string add_genre;
  int add_release_year;

  manager.create_examples();
  print_menu();



  while (choice != 0)
  {
    while (!get_valid_int(choice))
    {
      cout << "Invalid input! Please enter a number: ";
    }

    switch (choice)
    {
    case 0:
      cout << "GoodBye!" << endl;
      exit(0);
    case 1:
      manager.list_movies();
      print_menu();
      break;
    case 2:
      cout << "please type the movie name : ";
      cin >> search_name;
      manager.search(search_name);
      print_menu();
      break;
    case 3:
      cout << "Add a new movie to the Database:" << endl;
      cout << "Movie title: ";
      ignore_newline();
      getline(cin, add_name);
      cout << "Movie genre: ";
      getline(cin, add_genre);
      cout << "Movie release year: ";
      cin >> add_release_year;
      cout << endl;
      manager.add(add_name, add_genre, add_release_year);
      cout << "Added new movie:  " << add_name << endl;
      print_menu();
      break;
    case 4:
      cout << "Please input the ID of the movie you wish to delete" << endl;
      cout << "ID: ";
      cin >> id_del;
      if(id_del > 0) {
        manager.del(id_del);
      } else {
        cout << "Please input a valid ID!";
      }
      print_menu();
      break;
    default:
      cout << "Wrong Choice please select carefully!" << endl;
      print_menu();
      break;
    }
  }

  return 0;
}
```

As you can see, here in my main function, I have defined and initialized some variables for the code to work, I have created my movie manager object and by using the method create_examples() i have generate some sample data.

The main functionality is the while loop and the switch statement, in the first while loop im specifying that the code runs until the entered choice by the user is 0, in that case it will exit. (you can also see that for safety reasons i also exit the whole process by calling exit function)

The second while loop ensures that the user only enters valid integers. And nothing else. By using the help of get_valid_int() function.

The switch statement easily handles all the cases for the App's functionality. For each number there's a specific task. Im using the movie manager object which i have created by using the MovieManager class. I process the input, then pass the operations to the manager methods. For example 1 is listing the movies. I simply call

```
manager.list_movies();
```

Or for example deleting

```
manager.del(id_del);
```

And finally if no choice has been given :

```
default:
      cout << "Wrong Choice please select carefully!" << endl;
      print_menu();
      break;
```

The code is self-explanatory, and the Video shows a working example of the application.

Thank you, Professor Durdu.