LAB #9 – Structs vs. Classes

1. Create a Class:

In the last lab we made a movie_record struct, and then we made an array of movie_record structs. In this lab we’ll change the previous lab so that it uses a class, and we’ll make an array of movie objects. You will write a definition for a class named Movie to be used to model a movie at a redbox rental stand. Before you go further with this programming exercise, pair with your neighbor, and write down the attributes and behavior you expect from a movie from the point of view of the redbox.

You should have thought of some of the attributes we used in lab #8, e.g. title, copies, rating, description, and genre. You can certainly add more, e.g. price, type, etc. By default, these member variables are private.

Now, what functionality did you associate with a movie? You should have operations such as renting a movie, which will decrement the number of copies. You will need to set the member variables to some value, e.g. set_title(), set_rating(), set_genre, etc. In addition, you will need to get information too, e.g. get_title(), get_rating(), get_genre(), etc. How about displaying information about the movie, i.e. show_movie_info(), which displays all the information about a movie?

Do you now have a successful Movie class? Show your lab TA your Movie class to get 5 points for the lab!!

2. Create an Object/Use the Class:

Now, pair again with your neighbor and discuss how you might implement a redbox that gets populated with 5 movies, and then presents the user with a menu displaying all the titles of the movies in the box and asking the user if he/she wants to see more information about a movie or rent a movie and which movie they want to select.

Now, let’s implement a redbox that uses the Movie class. Since a redbox has a collection of movies, you will probably want an array of movies, which really isn’t the best choice because it is hard to grow and shrink an array (later, you can read about vectors in Chap. 7.3). However, this is just a lab exercise. You can create an array of 5 movies with a macro, and make sure you can populate the array with movie information to simulate adding movies to the redbox, i.e. add_movie_info().

For example:
int main() {
    Movie redbox[NUM_MOVIES];
    add_movie_info(redbox);
    ...
}

void add_movie_info(Movie redbox[]) {
    string title;
    for(int i=0; i<NUM_MOVIES; i++)
        read_title_info(redbox[i]);
    ...
}

void read_title_info(Movie &m) {
    string title;
    cout << "Enter the Title: ";
    getline(cin, title);
    m.set_title(title);
}

You should have also thought about being able to provide a menu for the purchaser, menu(). This menu() function will display the movies titles for the purchaser and ask the user if he/she wants to rent or see more information about a movie, as well as which movie he/she wants to select. Depending on the choice you need to call the right member function of the Movie class.

Are you now successfully using the Movie class? Show your lab TA your array/vector of Movie objects to get 5 points for the lab!!!

Extended Learning (Practice w/ Structs):
In this lab you will create a two dimensional array of structs that contain the multiplication table and division of the indices. The program needs to read the number of rows and columns from the user as command line arguments. You do not have to do error checking, but you will have to convert the string to a number.

    rows=atoi(argv[1]);  cols=atoi(argv[2]);

For example, if you run your program with these command line arguments, ./prog 5 5, then your program should create a 5 by 5 matrix of structs and assign the multiplication table to the mult variable in the struct and the division of the indices to the div variable in the struct. The mult variable is an integer and the div variable needs to be a float (or double).

    struct mult_div_values {
        int mult;
        float div;
    };

Your program needs to be well modularized with functions, including main, with 15 or less lines of code. This means you will have a function that creates the matrix of structs given the m x n dimensions, `mult_div_values** create_table(int m, int n)`. You need to have functions that set the multiplication values and division values, `void set_mult_values(mult_div_values **table, int m, int n)` and `void set_div_values(mult_div_values **table, int m, int n)`. Then, call functions to print the tables.

**Example:** `./prog 5 5`

Multiplication Table:

```
1   2   3   4   5
2   4   6   8  10
3   6   9  12  15
4   8  12  16  20
5  10  15  20  25
```

Division Table:

```
1.00  0.50  0.33  0.25  0.20
2.00  1.00  0.67  0.50  0.40
3.00  1.50  1.00  0.75  0.60
4.00  2.00  1.33  1.00  0.80
5.00  2.50  1.67  1.25  1.00
```