LAB #8 – Arrays vs. Structs

Understanding Your Mistakes:
For 3 points of this lab, you must correct three errors you have made in your assignments and/or labs. First, show the assignment/lab mistake to a TA, as well as discuss how you would fix the error. After you make the correction, show your work to a TA, and repeat this for all three mistakes for full credit. Depending on the complexity of the error, it may be easier to show the TA multiple mistakes at one time to reduce commotion😊

Show the TA your 3 mistakes/fixes for 3 points of this lab!!!

Understanding Pointers:
As a part of this lab, you have to draw a picture and explain each part of a pointer. For example, when you create some pointer, <type> *pointer;, what do the three pieces of a pointer access, and how would you use each?

&pointer
pointer
*pointer

You must be able to draw a picture and explain how you would use each of the three pieces of a pointer for 2 points of this lab!!!

Practice Structs:
In this lab, you will practice using structs in your program. We use structs as containers to hold information of mixed types, and many times this includes records of information in the real world, such as information about movies for renting. If you had to write a program for a redbox or Netflix, then you would have thousands of records of information about movies. How do you carry around this information in your program? Do you make an array for each piece of information? For example:

```c
#define NUM_MOVIES 100000
int main() {  
    string title[NUM_MOVIES];  
    int copies[NUM_MOVIES];  
    string rating[NUM_MOVIES];  
    string description[NUM_MOVIES];  
    string genre[NUM_MOVIES];  
...
```

Think about keeping up with what movie is in which slot of the arrays. For instance, all the information in all the elements must match, and if you sort the movies by title/genre
or add/delete movies from your inventory, then you always have to keep track of what you do and apply the same action to all other arrays to keep them aligned, right? Wouldn’t this be easier to make a movie record of this information in a struct, and make one array of movie record structs? Your job is to do just this. **Make a movie_record struct, and then make an array of movie_record structs.**

After you setup your array of movie records, then write a function, `set_movies()`, that allows you to set the information for each movie, i.e. the title, rating, description, copies, and genre. You only have to do this for 5 movies, i.e. #define NUM_MOVIES 5.

Now, write another function, `which_movies_to_view()`, that asks the user which movies he/she wants to view. You can ask the user if he/she want to see all movies or by genre, and write a function, `get_movies()`, to display the titles of movies depending on the user’s choice. Hint: This might be a good time to use function overloading.

Lastly, let the user choose to see more information about a movie or rent a movie. Write a function, `get_movie_info()`, that displays all the information about a specific movie for a user, and write another function, `rent_movie()`, that rents the movie to the user by decrementing the number of copies for a specific movie.

**Show your program to a lab TA for the other 5 points of this lab.**

**Extended Learning:**
Instead of creating a pointer to a pointer to a pointer to a pointer that points to an integer, i.e. int ****p;, create a linked list of integers using structs that contain an integer and a pointer to another struct. This is what we call a linked list in computer science, and we use this data structure often. The struct that you will create is called a node:

```c
struct node {
    int num;
    node *next;
};
```

Now, create a list of integers that are chained together using the struct node, but you will need to make a “head” that points to the beginning location of the first node in your list of integers, i.e. node *head = new node; Set the num value to 5, and then create 4 more nodes with integers set to 4, 3, 2, and 1 (NOTE: use a tmp to save your pointer value!). Now print the integers in order from 1 to 5. Refer to Chapter 17 for help.