CS 161 Lab #8 – 2D Arrays and Structs

- Start by getting checked off for (up to) 3 points of work from Lab 7 in the first 10 minutes.
- Your goal is to get all points for Lab 8 checked off by a TA by the end of this lab, so that you do not need to do additional work outside of the lab.
- You will work in small groups to complete the programming assignments, with each person writing their own program. Include each partner's name in your file header.

Goals:

- Practice working with static and dynamic 2D arrays
- Practice passing and returning 2D arrays with functions
- Practice creating and using structs

(2 pts) A. Static 2D Arrays

1. Create a file called **lab8.cpp** and declare a **static** 2D array inside main() to store **the number of people living in each house in a city block**. The houses are laid out in a **4x3 grid.**
    a. Example from TA demo (you will need to change this):
    ```
    int tree_heights[2][3];
    ```
2. Write a **nested for loop** to set the number of people living in each house to a random number between 1 and 10.
    a. Remember to seed the random number generator once at the start of main().
    b. Example from TA demo (you will need to change this):
    ```
    for (int x = 0; x < 2; x++)
       for (int y = 0; y < 3; y++)
          tree_heights[x][y] = rand()%10 + 5; /* 5-14 feet */
    ```
3. Write a **nested for loop** to print out the populations that were generated, showing the **coordinates** of each house and the **number of people** in the house.

(2 pts) B. Dynamic 2D Arrays

1. Add code to read in two numbers from the user to define the layout of a new city block.
2. Declare a **dynamic** 2D array inside main() to store the number of people in this new city block.
    a. Example from TA demo (you will need to change this):
    ```
    int** your_forest = new int*[size_x];
    for (int x = 0; x < size_x; x++)
       your_forest[x] = new int[size_y];
    ```
3. Write a **nested for loop** to set the number of people living in each house to a random number between 1 and 10.
4. Write a **nested for loop** to print out the populations that were generated, showing the **coordinates** of each house and the **number of people** in the house.
5. Use **valgrind** to check your program for memory leaks.

```
$ valgrind lab8
```

Did you find any? If so, fix them now and re-run valgrind until you get no leaks.

(3 pts) C. 2D Arrays with Functions

1. Move your 2D array printing code into two functions: one for the static array and one for the dynamic array.
    a. Examples from TA demo (you will need to change these):
    ```
    void print_forest(int forest[][3], int sx) { ... }
    void print_forest(int** forest, int sx, int sy) { ... }
    ```
2. Compile, run, and test your program to ensure it still works the same way as in parts A and B.
3. Create a function to generates the dynamic city block, given dimensions.
    a. Example from TA demo (you will need to change it):
    ```
    int** create_forest(int nx, int ny) {
       int** f = new int*[nx];
       for (int x = 0; x < nx; x++)
          f[x] = new int[ny];
       return f;
    }
    ```
4. Replace your 2D dynamic array allocation in main() with a call to this function.
5. Compile, run, and test your program to ensure it still works the same way as in parts A and B.
6. Run valgrind to be safe! ☺

(3 pts) D. Structs

Let's create a new data type (**struct**) called `water_bottle` with **two member variables**:

```
struct water_bottle {
  string color;
  float volume;   /* in ounces */
};
```

Add this data type (struct) definition before the `main()` method. Don't forget the final semi-colon!

1. Inside `main()`, declare a **<u>static</u>** array that contains three water bottles.
2. Write a `for` loop to read in values from the user for the `color` and `volume` members of each water bottle. You do not need to check for valid user input.
    a. Example of accessing a member variable of an item in an array:
    ```
    my_arr[i].var = 3;
    ```
3. Write a `for` loop to find the **<u>index</u>** of the largest water bottle (by volume).
4. Output the **<u>index, color, and volume</u>** of the largest water bottle.
5. Test your program using the following example as a test case, plus tests you create.

Example output (user input is highlighted):

```
Enter color for water bottle 0: red
Enter length for water bottle 0: 28
Enter color for water bottle 1: green
Enter length for water bottle 1: 32
Enter color for water bottle 2: black
Enter length for water bottle 2: 16
The largest water bottle (index 1) has a volume of 32 ounces and is green.
```

E. Get your work checked off by a TA.

Submit your program on Canvas (can be done while you are waiting to be checked off).
All partners' names must be in the comment header to get credit.

1. Transfer your .cpp file from the ENGR servers to your local laptop.
2. Go to the Lab 8 assignment:
   https://canvas.oregonstate.edu/courses/1770357/assignments/7847930
3. Click "Submit Assignment".
4. Upload your .cpp file (lab8.cpp).
5. Click the **"Submit Assignment"** button.
6. You are done!

If you finish the lab early, this is a golden chance to work on your Assignment 5 (with TAs nearby to answer questions!).
Remember, the assignment you submit must be your work alone (no partners).