Homework 3

flow of control: if, if-else, switch, while, command line arguments

January 30, 2020

Abstract

The objective is to give you practice with the statements by which we control the flow of execution, and allow data to modify the flow of execution. We will also navigate through a graph data structure. User input will be collected and used.

The context of this problem is carrying out a hunt for treasure in a "house" that is represented by a graph. The program will accumulate treasure, and keep track of how much is found in what place. The program will accept user input to customize the search process.

Be sure to construct test cases for each function you build. Put thought into the test cases. The goal is to know that a function works, when it passes all of its test cases. One test case per function is usually not enough.

- Construct a sequence diagram for your project. You can draw it by hand and include a phone/photo, or draw it with a software tool.

- Use the test-driven development style for developing your code. Document this on a function by function basis. First the production function exists as a stub. Your test code will invoke this function, usually with multiple test cases, and a good test will be able to notice that the stub function is inadequate to pass all of the test cases. Take a screen shot of your production codee in its manifestation as a stub, including the console showing that the test function fails. (See Figure 2), This is called the "before" screen shot. After your production function has been developed sufficiently to pass all of the test cases, take another screenshot. (See Figure 3.) It is called the "after" screenshot. It should include the developed production code and the console message about passing the test. Depending upon how many functions are in your sequence diagram, there could be a large number of screen shots.

  Be sure to run your code every time you add a few lines of test or production code. (See Figures 4 and 5.) Do not allow the number of errors to get large.

- An explanation of a representation of a graph will be given here, because a graph data structure will be used to represent the layout of the house. The

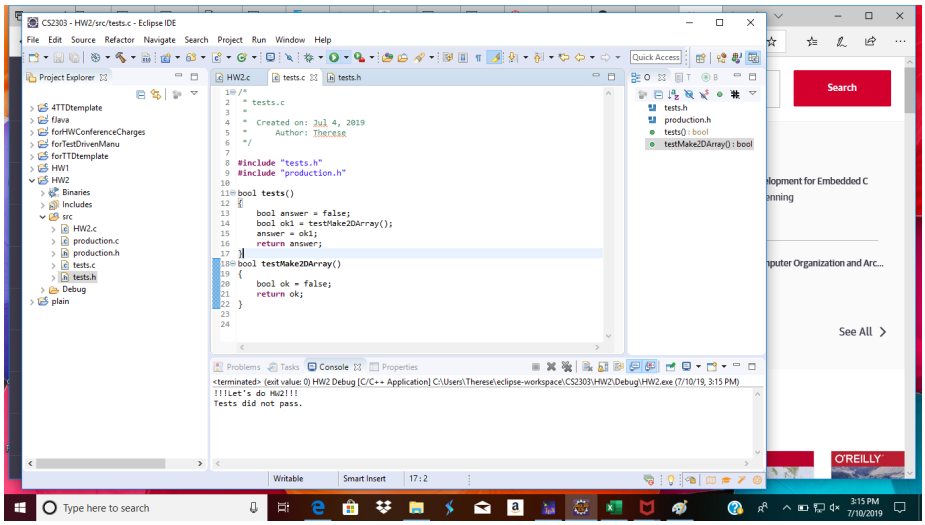

Figure 1: In the process of creating a new test.

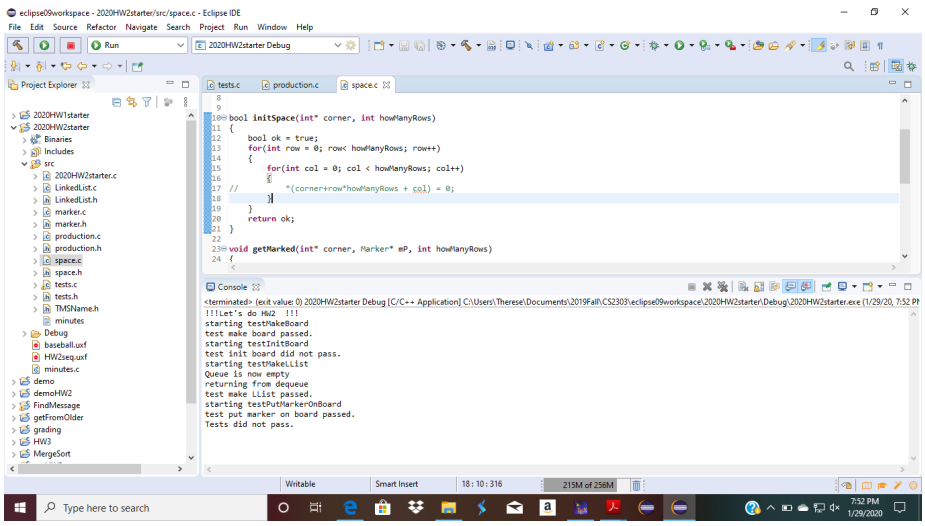

Figure 2: The test function for initialization fails, because the stub does not perform its work.

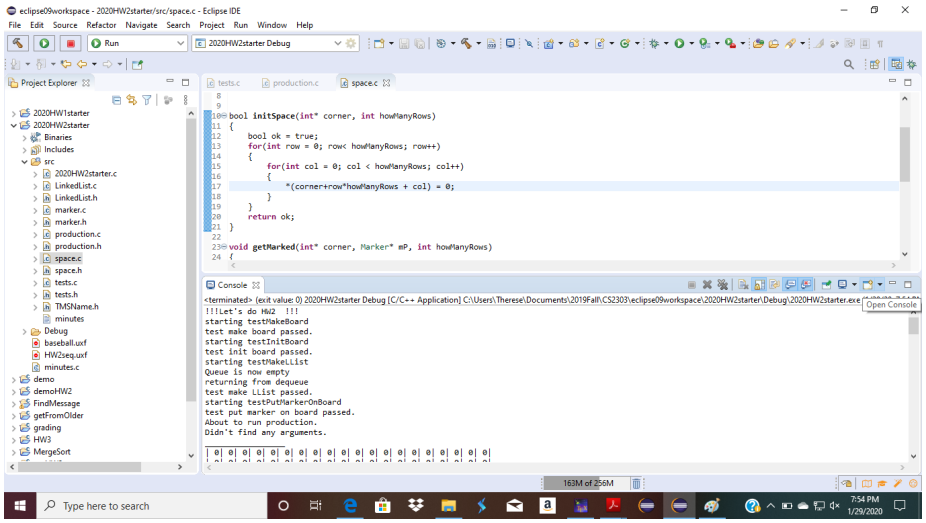

Figure 3: The test function passes, because the production function performs its job.

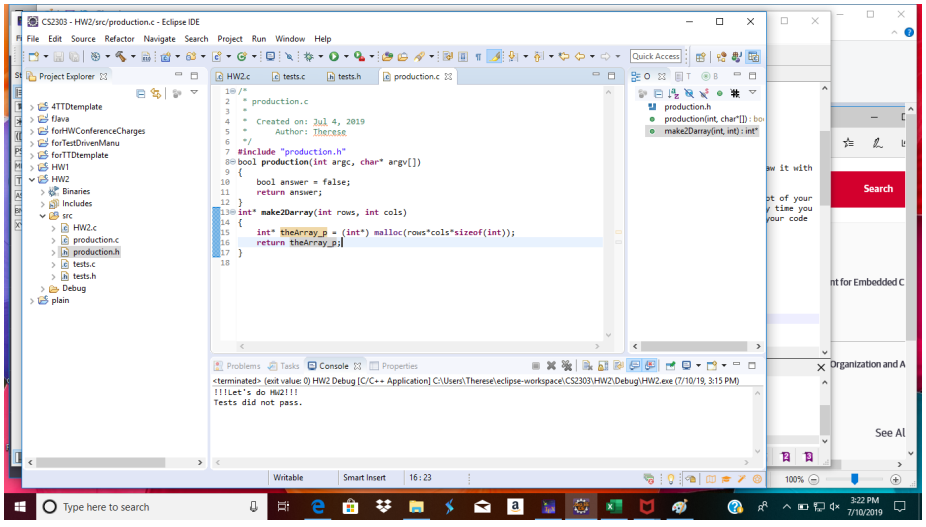

Figure 4: Starting some production code.

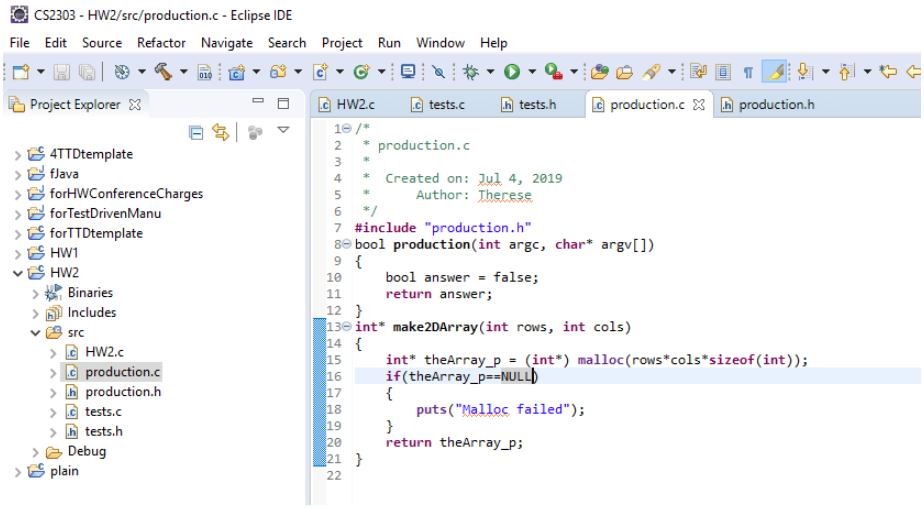

Figure 5: Adding more production code.

	room 0	room 1	room 2	room3
room 0	1	1	0	0
room 1	1	1	1	1
room 2	0	1	1	0
room 3	0	1	0	1

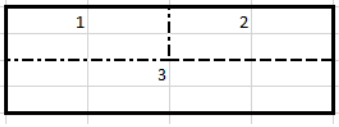

Figure 6: The matrix on the left represents the floor plan on the right. The front door opens into room 1. Room 1 connects to rooms 2 and to 3, but room three does not connect with room 2.

program must use this data structure to present to the user the choices of next rooms to search. The representation is called an adjacency matrix. It is a two dimensional matrix. The number of rows and the number of columns are both set to be the number of nodes in the graph. In our case, it is the number of rooms in the treasure house. The values in the matrix come from whether the room number that is the row number is directly connected to the room number that is the column number. Because these direct connections are bidirectional, the adjacency matrix is symmetric about its main diagonal. Moreover, the values in the main diagonal are all ones, because every room is directly reachable from itself. If the value in the matrix is not one, it is zero. You can use boolean or integer for the datatype of the array representing the adjacency matrix. We can add one more row and column to the graph to represent the front door, or, equivalently, the outside world if you prefer. Make functions for working with this data representation. For example, if the searcher is in room x, the searcher might wish to know which are the adjacent rooms.

- The searcher looks for treasure, and accumulates it. The goal is to maximize the treasure found. Functions supporting this include finding out how much treasure has been accumulated, and adding the amount of a newly found treasure.
- Whether a room has been searched yet, or not, is a useful idea. Make a data structure to record this information. Don't forget to initialize this so that no room has been searched. If you include the outside world as a room, it has already been searched.
- Print the rooms that have been searched, and the treasure subtotals, as they are accumulated.
- The program must enable the user to specify two limits: an amount of treasure and a number of rooms. If the subtotal of treasure is greater than or equal to the limit, the searcher must exit the house without picking up any more treasure. If the number of rooms searched equals the limit, the searcher must exit the house without picking up any more treasure. One or both of these two values can be entered on the command line. If fewer than two entries appear on the command line, the program must query the user for these values.

Things to do:

1. Either:
   (a) Make a C project from the Hello,World project.
   (b) Populate that project with tests.c, tests.h, production.c and production .h.

   or use the starter code.
2. Create the sequence diagram and include the electronic file (diagram, screenshot or photo). Make sure your name appears within the sequence diagram.
3. Place function prototypes for all of your functions from the sequence diagram into one or more .h files.
4. As you work on the assignment, collect a sequence of screen shots showing how your test code, and your production code are growing.
5. Be sure to build and run often; do not allow errors to build up.
6. Show the sequence of moves from room to room by listing them, and also show the subtotal of treasure through the house.
7. Receive the values entered on the command line. Ask for values not provided in the command line. In either case, check for reasonable values. Negative treasure or rooms implies no search should be conducted. Use these limit values in execution.

Grading

Criteria	Possible Poin
Project that looks like starter code	20
Sequence diagram that reflects the problem statement	20
Documentation of code development (those screenshots) that clearly follows test-driven style	20
Screenshot of rooms searched, visited and treasure collected subtotals, that correspond	20
Correct treatment of command line arguments	20
Total	100