Isaiah Dicristoforo

Contemporary Programming: Fall 2019

dicrisif@mail.uc.edu

Module 13 Assignment

Due: 11/27/2019

Thread Assignment Analysis

About My Program

This program models polling places where you would go to vote in an election. I created a class called polling location which contains…

private static int unitedStatesVoteTotal

<<This staticvariable stores the total number of votes cast in ***ALL*** polling places. Only one copy of this variable exists in each polling place object. It is not thread safe>>

private States locationState;

<<This enumeration type stores the state that this polling place is located. All states are available in this enumeration>>

private int locationVotes;

<<This is an instance variable which stores the number of votes cast in each location, not across the entire country, like the static variable above>>

private string locationName;

<<The name of the polling place location.>>

public void CastVote()

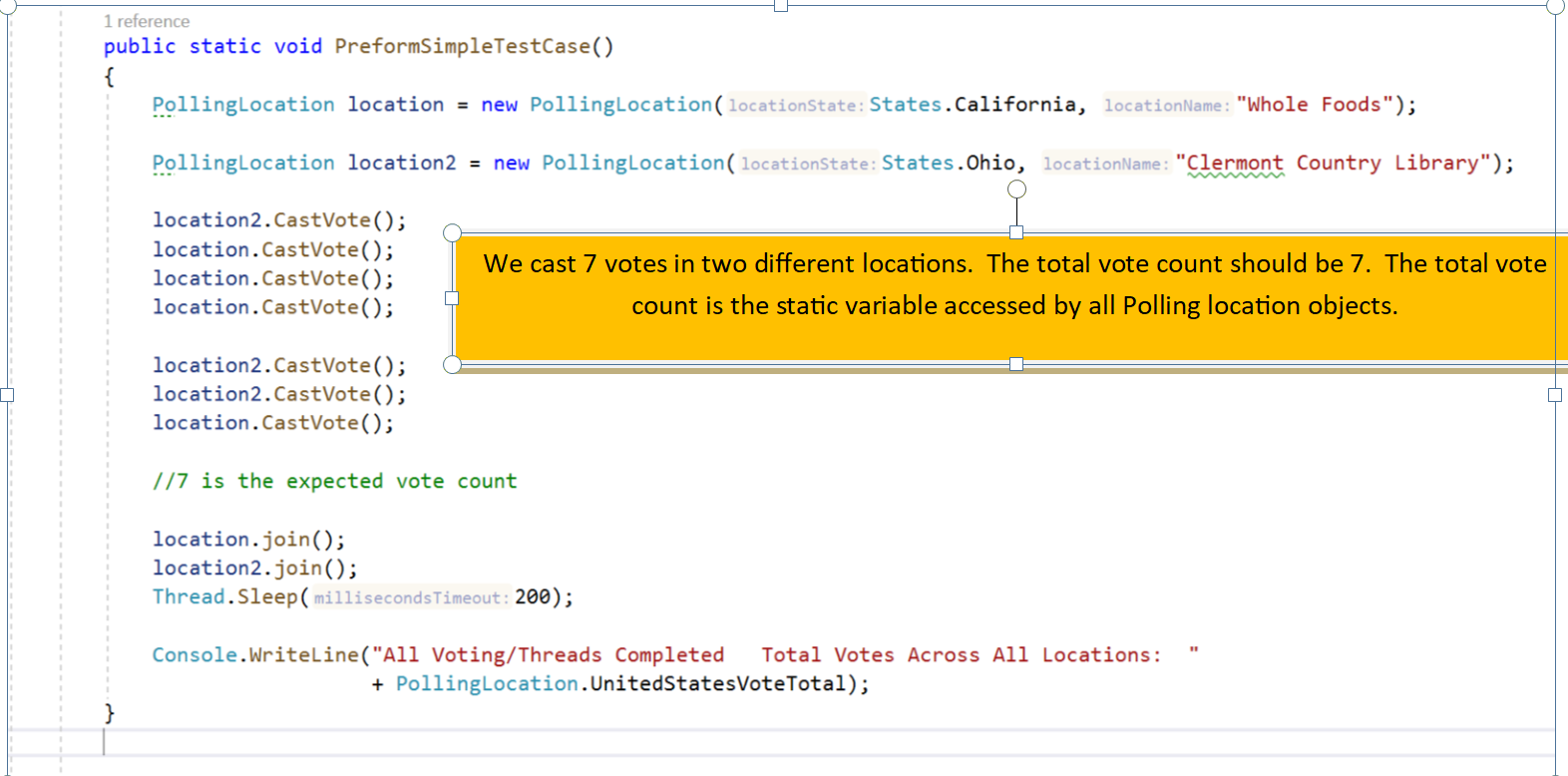
<<This public method instantiates, initializes, and starts a thread

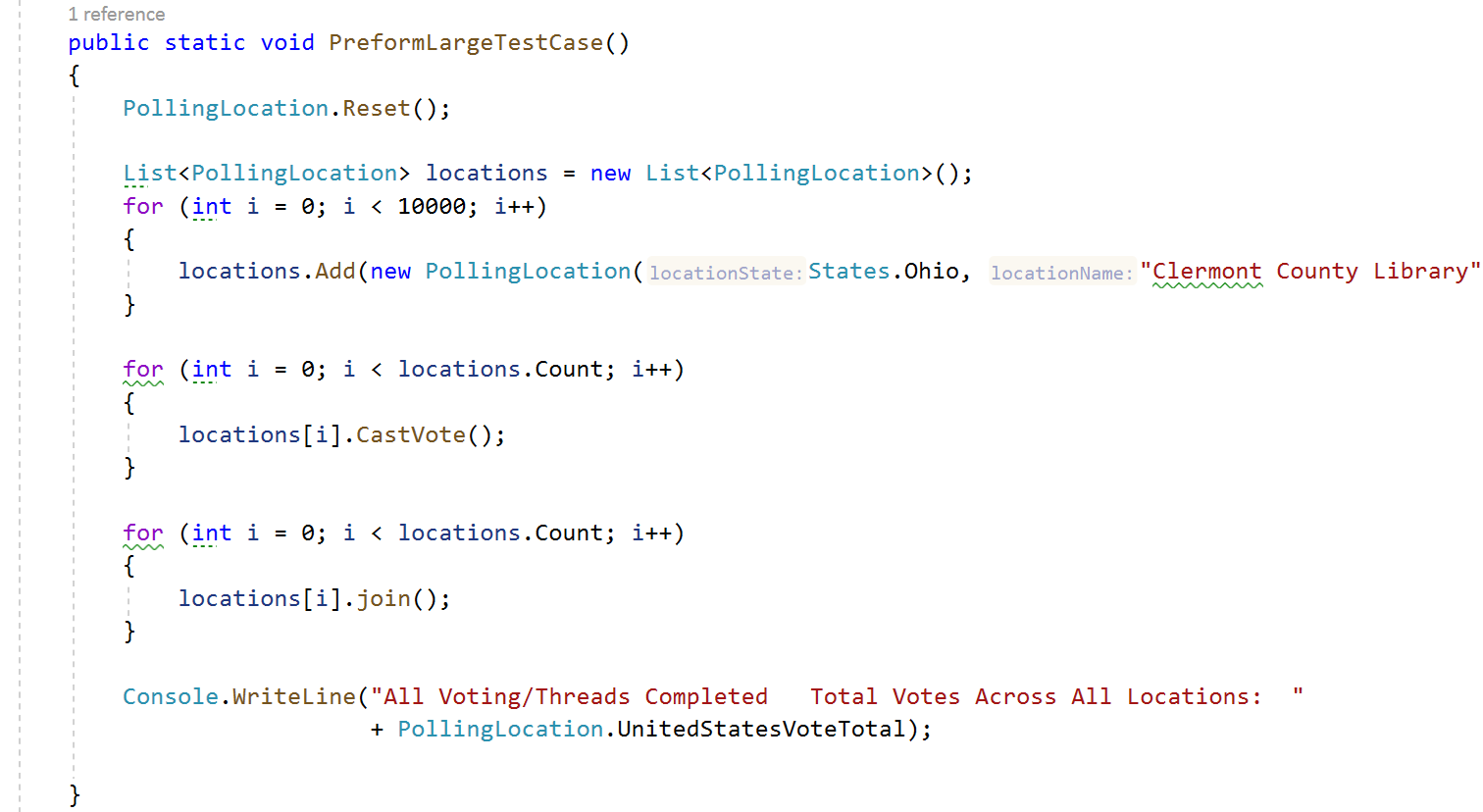
private void Vote()

<<This method is the entry point for our thread. It increases our static counter variable, as well as the instance variable>>

In my project I used a main method to test my threads. I also created an optional unit test project.

**Expected Results**



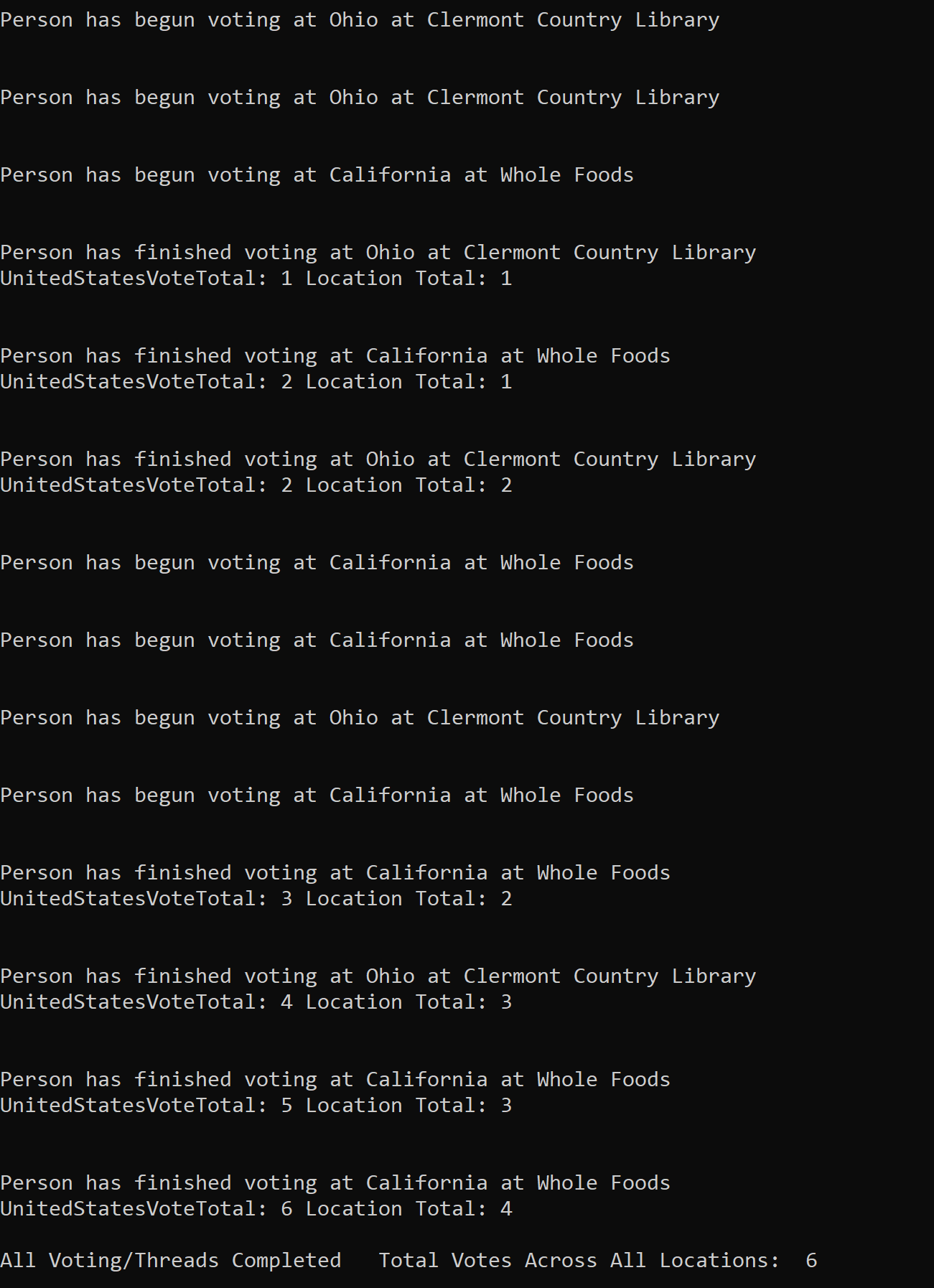


I create 10,000 polling locations. In the 2nd for loop I cast one vote at each location. I should have 10,000 votes cast across all locations.

**Actual Results**

**Results of My Simple Test Cases (Before The Lock)**

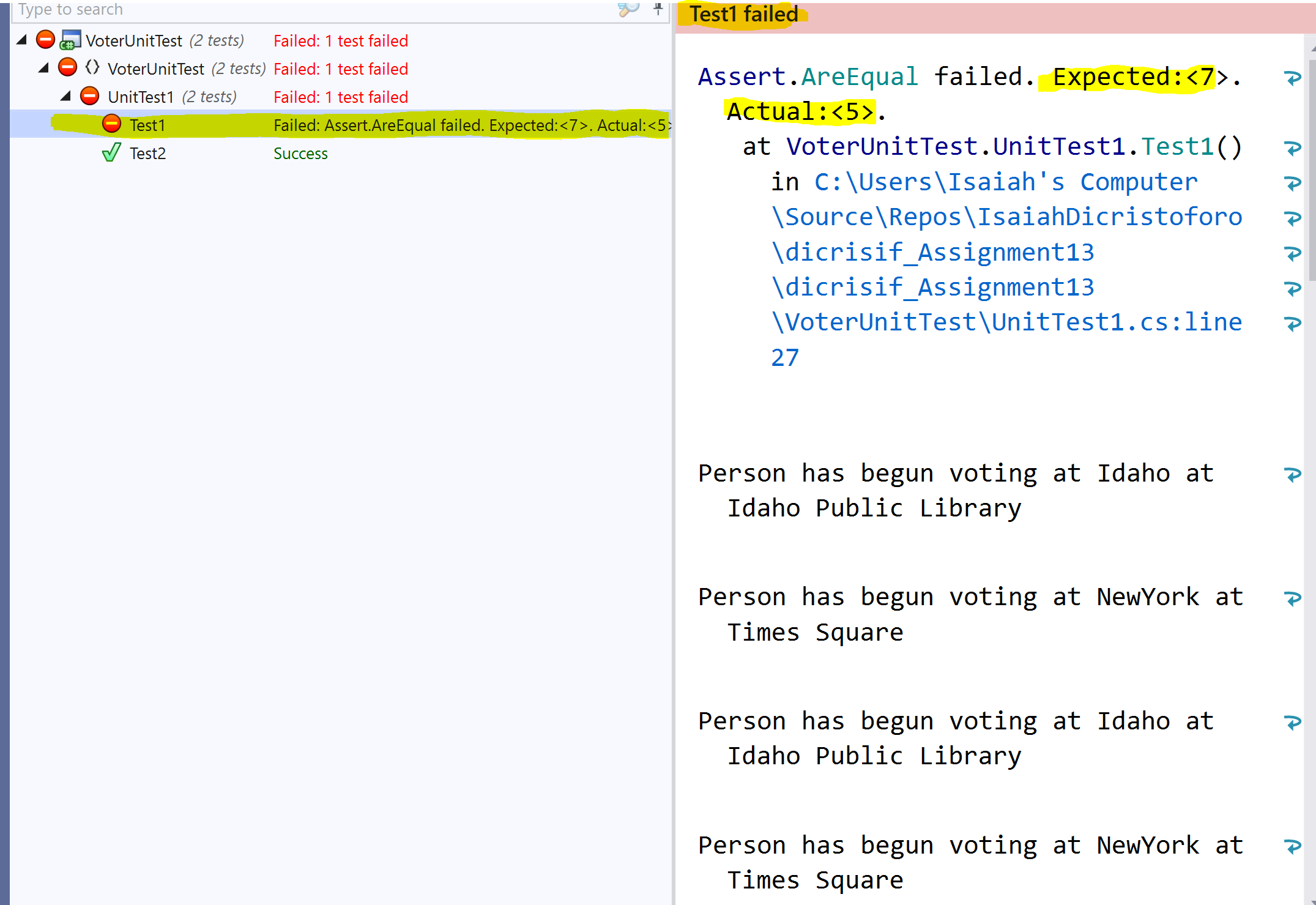
Most of the times I ran my simple test case, 7 votes were cast in total as expected. However, every so often, my data would get corrupted as shown below. In the case below, two of my threads got access to my static variable at the same time and incremented it at the same time. While the static variable should have increased by two, it only increased by one.



Results….Notice how we have 7 people finish voting, but only 6 votes cast at the end of our program. Our static variable has been corrupted.

Here we have 2 people finish voting. The static variable, UnitedStatesVoteTotal, should not be 2 both times. Both threads incremented this variable at the same time, corrupting our data.

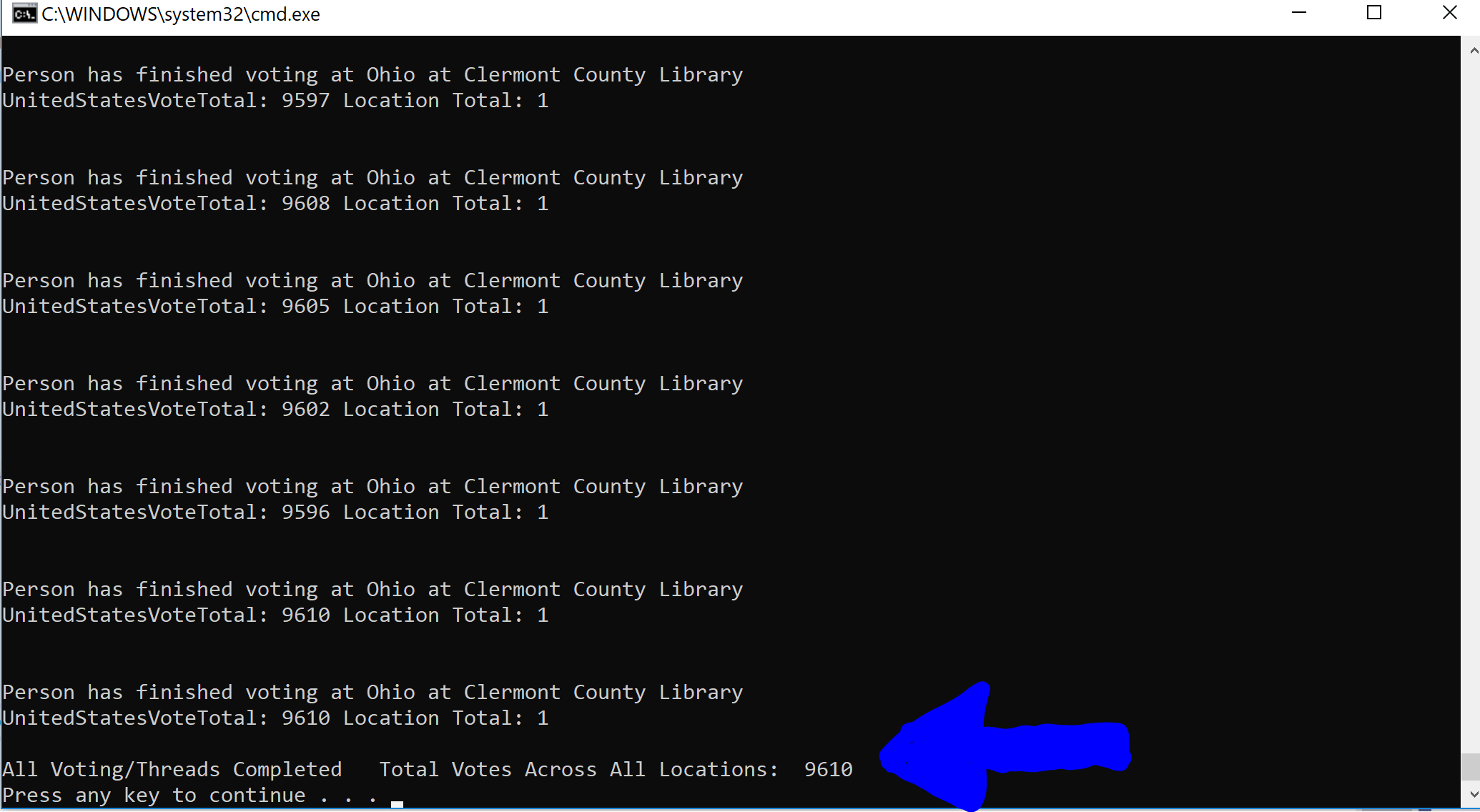
I also did simple test case in a unit test project. My static variable was corrupted again, as it was incremented at the same time by different threads. Similar to the test case in my main, the vote total should be 7.



**Results of My Advanced Test Cases (Before The Lock)**

The previous test cased were more basic. I only created a few test PollingLocation objects and cast only a few votes. While I got my program to break, I had to run it a few times to get inaccurate results. Below, I implemented some test cases where I hammered my static variable per the assignment requirements. This time, it was easily corrupted.

Here, I created 10,000 polling locations, and stored them in an array list. I cast cast one vote at each location, spawning 10,000 threads. Finally, I joined those threads together. As with the previous test cases, our static variable was corrupted due to multiple threads accessing it, and incrementing it at the same time.

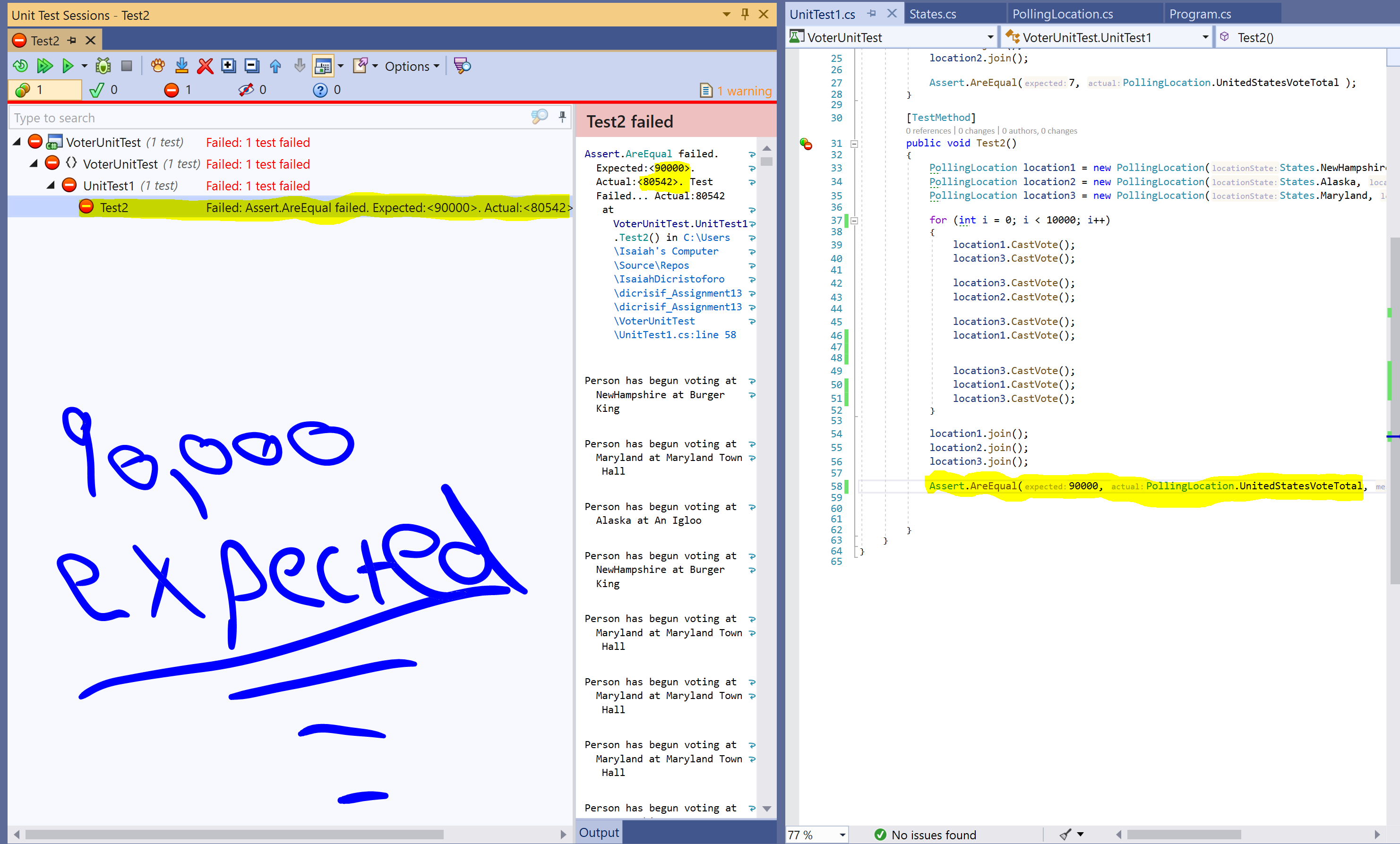


The two cases directly to the left show two people voting, yet no change in the static variable, *UnitedStatesVoteTotal*. Our variable is corrupted again.

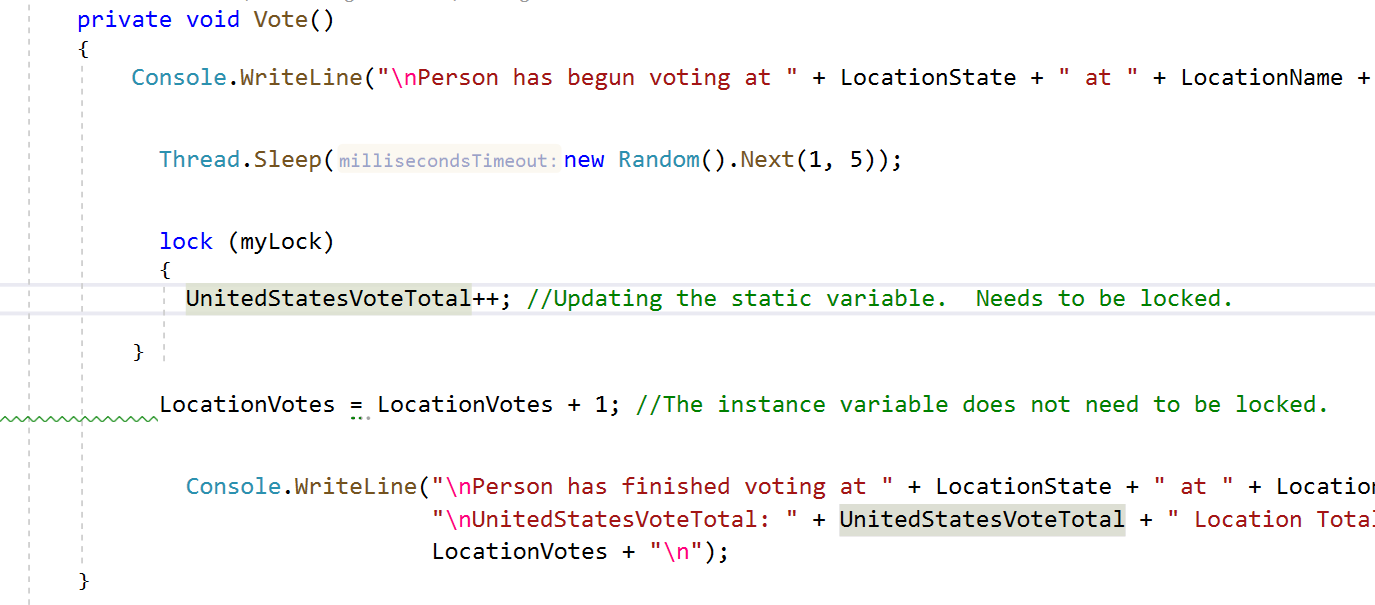
Result, 9610 votes

Expected, 10000 votes

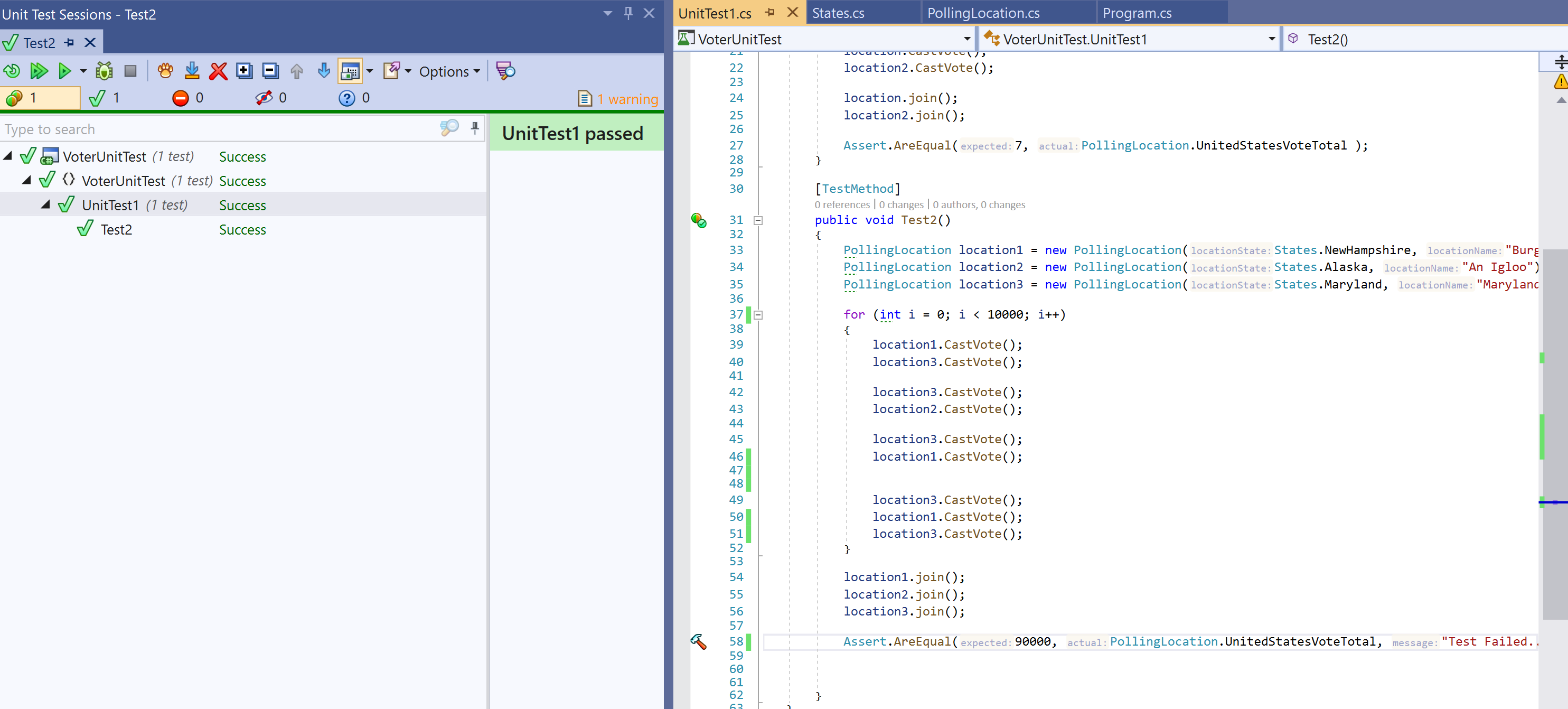
I also did a massive unit test (pictured below), where I created three polling locations, and cast 90,000 votes. The static vote-counter was only incremented 80,542 times. My data was severely corrupted, and my unit test failed. Once again, this was due to my static variable being accessed by multiple threads at the same time.



**Results When I Locked My Critical Code**

I put a lock on my critical code to ensure that my critical code could *only* be accessed by one thread at a time. I do not want to put a lock around all of my multi-threaded code, because that would defeat the purpose of a multithreaded program. As you can see, my print statements and instance variable calculations did not need to be locked.

After implementing the lock, I immediately tested the previous methods without making any changes. Both unit tests were successful, as well as the tests in my main method. Only one thread had access to my static variable at a time.

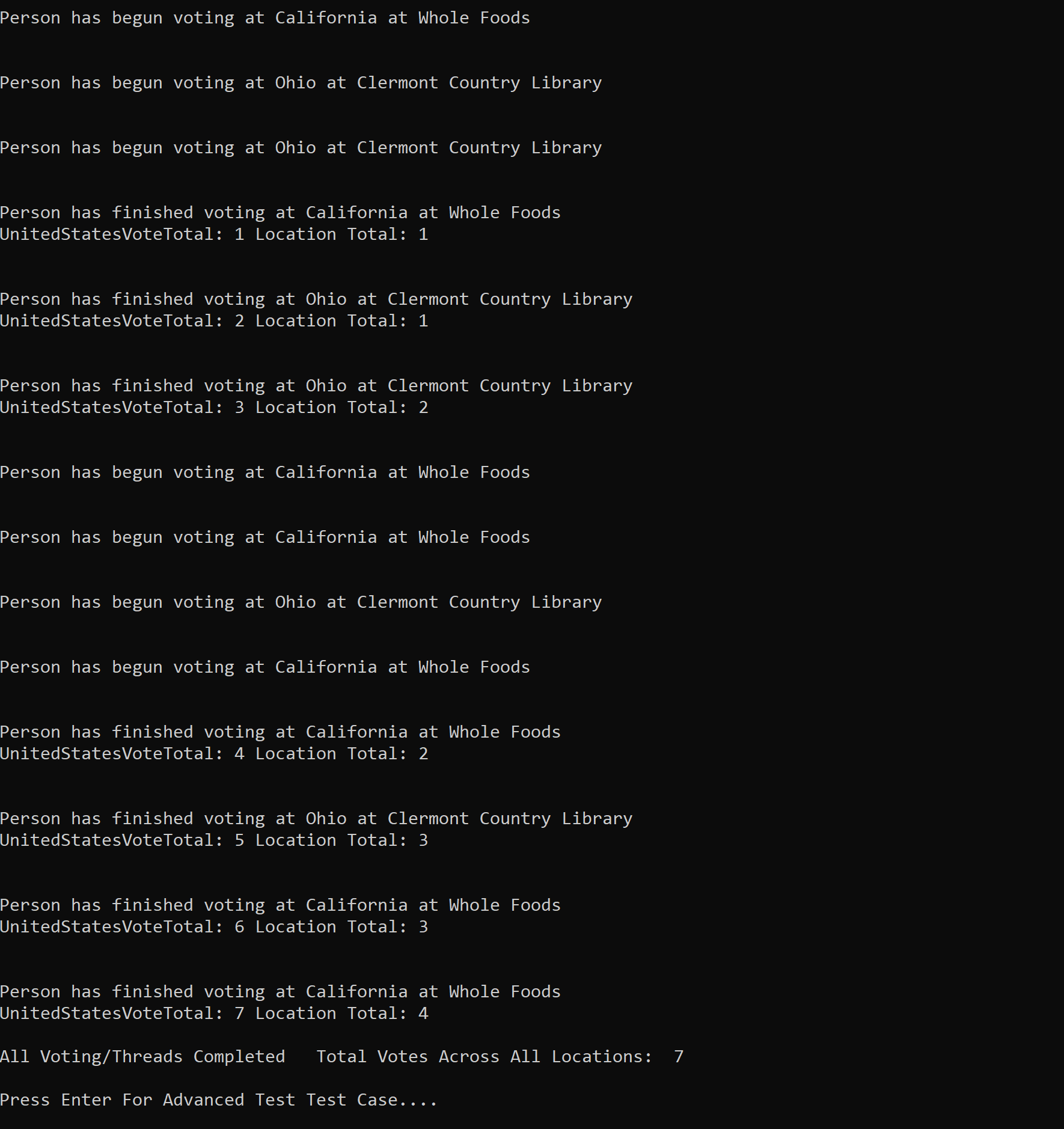


Both Unit Tests Succeed…

Test One: 7

Test Two: 90,000

More Analysis + Results Below



As shown above, my simple test was successful, as 7 votes were successfully cast. Without the lock, this number varied from 5-7. Now it correctly stayed at 7, the expected result, considering we spawned 7 threads who casted one vote each. Note that while is our end result is correct, the order of the result statements in the console is not necessarily the order the threads were executed.

7 Votes should be cast each time!

Similarly, my advanced test case in my main proved successful, as 10000 votes were successfully cast. The lock was successful, and multiple threads did not increment it at the same time. We spawned 10,000 threads in this test case, which each incremented the static variable by one, so 10,000 was the expected result! Before the lock, this number was varying somewhere in the 9,000 range each time the program was run.