Video #8: Working with a Vector of Structures of Information About Stars

Many applications involve processing data about a collection of entities, where each entity has a set of properties. This video expands on an example we explored in an earlier lecture, where the entities are stars, and information about the stars is stored in a vector of structures. Imagine that we created a vector named stars, and each location of the vector contains a structure, and each structure has three fields: the name of the star, its constellation and its temperature. Because this is a vector, we refer to an individual structure using an index, for example, we'd refer to the second structure in the vector as stars(2). We can think of this expression as the name of this second structure. If we then want to refer to the content inside that structure, we use dot notation to refer to each field, so the name, constellation, and temperature of this particular star would be stars(2).name, stars(2).const, stars(2).temp.

I'm going to describe a few different functions for analyzing the data stored here, functions that perform tasks like printing all the information stored in this vector, printing the names of all the stars located in a particular constellation, printing the temperature of a star with a particular name and constellation, finding the hottest star overall, and finding the hottest star in a particular constellation.

To test these functions, we need some data to work with, so as a starting point, we'll create a vector of information about a set of 15 stars. This is done with a script called makeStars. At the top are the names of 15 stars, stored in a cell array, their corresponding constellations also stored in a cell array, and a vector of the 15 temperatures. These cell arrays and vectors are in synch, in the sense that the first name is in the first constellation and has the first temperature, and so on. I initialize the stars vector to be an empty vector and I have a loop with an index from 1 to 15, and a new structure is created for each index with our three fields and with values taken from the two cell arrays and vector. So this creates our data.

Going back to our picture, our first function will loop through the entire vector and print out all the information stored in each structure. I named this function printAllStarsInfo and it has a single input that's the vector of structures with all the star information. We have a for loop with an index i that goes from 1 up to the length of the input vector, and for each value of the index i, we display all the information in that structure with a single disp statement so it all appears on a single line, and we refer to the three fields of each structure as allStars(i).name, allStars(i).const (constellation), and the temperature is a number, so we need to use num2str to convert to a string that can be concatenated with all the other strings here. How do we call this function? I have a separate script file called testStars.m, and at the moment, I just have the first couple statement uncommented, so let's run and test <view output in Command Window>.

While I'm here, let me uncomment this section that tests a function printStarsInConstellation. This function has two inputs, the vector of information about the stars and a constellation. It prints all the stars located in that constellation, and we'd like to be able to handle the situation where the constellation isn't found in the vector. Let's look at the definition of this function.

There's a message at the beginning that just prints something like "stars in the constellation Ursa Major". We have a for loop like before that sets up an index from 1 to the length of the vector, and at each location, we use strcmp to check whether the constellation at this location is a match to the input constellation. If so, we print the name of the star. But what if the constellation that we provided as input is not found? We'd like to print a message saying so. To help with this task, I created a variable named found that I initialized to 0. The idea is, if we actually find the constellation, we can set found to 1 to indicate that. Later, when we're done with the loop, we can check the value of found. If it's still 0, the constellation was never found, so we print our message. <let's run these tests>

Next we'll look at this function printTemp. It obviously has three inputs, the vector of information about the stars, the name of a star, and a constellation. The function prints the temperature of this star in this constellation. Now we know that all stars have unique names, or at least I think this is true, but let's pretend that there are stars of the same name in multiple constellations, just like there are cities of the same name in different states, and look for this combination in our stars vector. And we may not have a valid combination. I don't think there's a star named Ellen, but when I asked Google if there's a star named Ellen, it of course came back with Ellen DeGeneres.

Before looking at the definition of the printTemp function, let's look back at our picture. We might be looking for a star that appears early in our vector (like Spica in Virgo), and maybe instead of 15, we have thousands of stars. As we're searching through the vector, if we find what we're looking for, it would be nice to stop the loop. We can do this with a for loop that has a break statement inside, or we can do this with a while loop. I'll show you both solutions. I put them both in the same function definition. The function has three inputs, so we need to set up the function header to have three input parameters that I called allStars, starName, constellation. Let's first look at the solution that uses the for loop and break, it's simpler. We'll set up the loop to go through the entire vector, and at each location, we can check with strcmp, whether the name stored here is a match to the input starName, and whether the constellation stored here is also a match to the input constellation. If both of these conditions are true, we'll print the temperature of this star, and break. But there's one more thing to handle again. What if the combination of star name and constellation that we provided as input are not a valid combination? We'd like to print a message saying that the star was not found. Again, I created a variable named found that I initialized to 0, and if we actually find the star, we set found to 1. When we're done with the loop, we check the value of found. If it's still 0, the star was never found, so we can print our message.

The solution using a while statement is different. Let's look at the picture again. The aim of the while loop will be to figure out the index where our star is located. We'll start with an index of 1, and if we don't find the star there, we increment the index. We repeat this process. While we haven't yet found our star, we keep incrementing the index. When we find the star, the loop will stop at the index where the star is found. We don't want to just keep incrementing our index indefinitely, eventually we'll hit the end of the vector, so we only want to keep

incrementing the index while we're still inside the bounds of the vector. Now let's look at the code. For the while loop, we need to create an index before we start, I'll call it i and initialize it to 1. Then inside our while statement, as long as the index is still in bounds, and we don't find the star name and constellation we're looking for, we keep going and increment i. If the star and constellation are found, the loop will stop and the value of i at that point will be the index of the structure where we found our star. If the star and constellation are not found in the vector, i will get to the end, where it's equal to the length(allStars), the star won't be found in this last location, and i will be incremented once more, to the length of the vector plus 1, and this condition (i <= length(allStars)) will no longer be true, so the loop will stop. After the loop stops, if i is within the bounds of the vector, this means the star was found and we can print its temperature. Otherwise star wasn't found and we can print this message.

That's a lot so far, but I have just one more kind of task I'd like to illustrate. This involves finding the hottest star. First we'll look at a function that returns the name and temperature of the hottest star overall <run>. Let me uncomment the statement that calls hottestStar and run the script. Notice that from the function call here, we can see that it will have one input, the vector of star information, and two outputs, the name and temperature of the hottest star. So we'll need to set up the function header to have one input parameter and two output parameters. For the overall strategy, let's look once more at the picture. The idea is to loop through the vector, and as we go, keep track of the hottest star we've seen so far, both name and temperature. We can initialize the hottest temperature to something like 0, then check the first location. Hottest we've seen so far, so update hottest temp to 4300 and keep track of name Ginan. Then move on to the next, is it hotter? No, so don't change our information about the hottest star. Then we move on to the next and it's hotter than the hottest thing we've seen so far, so we update hottest temp to be 22,400 and keep track of its name Spica. We keep moving on and updating as we find even hotter stars, and return this information when we're done looking at all the stars. So let's look at the code. Here's our function header with the input vector and two outputs. We initialize hottestTemp and hottestName, and loop through the vector from index 1 to length(allStars). If the temperature at the new location is greater than the hottest we've seen so far, we update the hottestTemp and hottestName.

Finally, let's qualify what we mean by the hottest star, and instead return the name and temp of the hottest star in a particular constellation (this is like finding the city with the largest population for a particular state). This is just like our previous function, with two modifications. As we loop through the vector, we only consider stars whose constellation matches the input constellation, so we just have an extra if statement around the statement that checks if we have a hotter star at this location. And the constellation may not be found, but in this case, don't need a separate variable to keep track of this, if the constellation was never found, hottestTemp would still be 0 and we can print the message. <run examples> So there's some sample functions that process data about a collection of entities that's stored in a vector of structures. In the assignment, you'll explore these kinds of tasks on your own, using data on US cities.