Using Structures in MATLAB

Table of Contents

Structures are one of the most usefull (and under-taught) tools in Matlab. They greatly simplify organization of your data both for easy analysis months later (after you've forgotten the details of your experiment) and for passing large sets of variables around different functions in your code. You'll see the crucial role they play in building GUIs in Matlab in a bit, but first let's go over what they are and how to use them for simple data organization.

Building a simple structure

Okay, let's say you're conducting an experiment on zombies. Somehow, inexplicably, every zombie apocolypse scenario came true simultaneously. You (and Matlab) have survived, and now you are at the CDC trying to track the different types of outbreaks. Right. This is happening. First, though, you catalog the survivors. You begin by making a structure.

```
survivors = struct()
```

```
survivors =

struct with no fields.
```

So far so good. We've made a structure, but it has no fields. A field is like a variable stored within the structure. As such, it can have any valid variable name and type. There are two ways to define these fields. The first is when we first make the structure. We pass the field name as a string, followed by the *value* we want that field to take. We're going to fill in our values later, so we'll create the fields as empty for now:

```
survivors = struct('Name',[],'Age',[])
```

```
survivors =

    Name: []
     Age: []
```

You can also add fields after creating a structure. I'll make this one empty as well.

```
survivors.Skills = []
```

```
survivors =

      Name: []
       Age: []
```

```
    Skills: []
```

Okay, now let's put some info in there. Let's store a string, a number, and a cell array.

```
survivors.Name = 'Jesse Eisenberg';
survivors.Age = 22;
survivors.Skills = {'Attention to rules';'Nutting up and shutting
 up';'Having a crush on Emma Stone'};
survivors
```

```
survivors = 

      Name: 'Jesse Eisenberg'
       Age: 22
    Skills: {3x1 cell}
```

We can look at that cell array with

```
survivors.Skills
```

```
ans = 

    'Attention to rules'
    'Nutting up and shutting up'
    'Having a crush on Emma Stone'
```

And we can access each element of the cell like we would normally

```
survivors.Skills{2}
```

```
ans =

Nutting up and shutting up
```

Now lets add some other survivors to our list:

```
survivors(2).Name = 'Hershel';
survivors(2).Age = 62;
survivors(2).Skills = {'Veterinary Medicine';'Farming';'Fatherly
 Advice'};
survivors(3).Name = 'Jim';
survivors(3).Skills = {'Shouting "Hello?" upon walking into literally
 every room'};
```

Note that each entry into the survivors shares the same fields, even if those fields don't all have values, or have values of different lengths

```
survivors
survivors(2)
survivors(3)


survivors =

1x3 struct array with fields:

    Name
    Age
    Skills


ans =

      Name: 'Hershel'
       Age: 62
    Skills: {3x1 cell}


ans =

      Name: 'Jim'
       Age: []
    Skills: {'Shouting "Hello?" upon walking into literally every
 room'}
```

Those fields don't even have to have the same *type* of value

```
survivors(4).Name = 'Deep Thought';
survivors(4).Skills = 42;
survivors(4)


ans =

      Name: 'Deep Thought'
       Age: []
    Skills: 42
```

We can also get all entries in a specific field

```
survivors.Name


ans =

Jesse Eisenberg


ans =

Hershel
```

```
ans =

Jim


ans =

Deep Thought
```

Nested Structures

Okay, now you've catalogued the sum total of the human race (yourself excluded) and now it's time to move on to the zombies themselves. This is a more complex endeavor since there are several different kinds of zombies, each of which has several characteristics we are interested in. We can actually have fields within a structure be structures themselves. It's like Inception. Zombie Inception. Roll with it.

```
zombies = struct('Viral',struct('Fast',struct(),'Slow',struct()),...
    'Radioactive',struct('Fast',struct(),'Slow',struct()),...
    'Total_Infections',7013595467)
```

```
zombies =

               Viral: [1x1 struct]
         Radioactive: [1x1 struct]
    Total_Infections: 7.0136e+09
```

(Note that `...` notation. It's very helpful if you have a long list of parameter-value pairs. You'll see it again when we build our GUI).

We can access those sub-structures just like normal structures.

```
zombies.Viral.Total_Infections = 5218364112;
zombies.Viral
zombies.Radioactive.Total_Infections = zombies.Total_Infections-
zombies.Viral.Total_Infections;
zombies.Radioactive
```

```
ans =

                Fast: [1x1 struct]
                Slow: [1x1 struct]
    Total_Infections: 5.2184e+09


ans =

                Fast: [1x1 struct]
                Slow: [1x1 struct]
    Total_Infections: 1.7952e+09
```

And, of course, each of those sub-structures can also have multiple entries

```
zombies.Viral.Fast(1).Prototype = '28 Days Later';
zombies.Viral.Fast(2).Prototype = 'I Am Legend';
zombies.Viral.Fast(2).Note = 'Technically vampires';

zombies.Viral.Fast
```

```

ans = 

1x2 struct array with fields:

    Prototype
    Note

```

```
zombies.Viral.Fast(2)
```

```

ans = 

    Prototype: 'I Am Legend'
         Note: 'Technically vampires'

```

There's also no need to fill elements in order

```
zombies.Viral.Fast(4).Prototype = 'Z Nation';
zombies.Viral.Fast(3)
```

```

ans = 

    Prototype: []
         Note: []

```

Dynamically addressing fields

Okay, here's one of the awesome things about structures structures. I can define a variable as a string, then use that variable to access a field in a structure

```
type = 'Viral';
zombies.(type).Fast
```

```

ans = 

1x4 struct array with fields:

    Prototype
    Note
```

```
speed = 'Fast';
zombies.(type).(speed)(2)


ans = 

    Prototype: 'I Am Legend'
         Note: 'Technically vampires'
```

This is particularly useful for using structures in loops. We can use this to get the same type of data from multiple branches of our structure tree.

```
zombies.Viral.Slow.Prototype = 'The Walking Dead';
zombies.Radioactive.Slow.Prototype = 'Night of the Living Dead';
zombies.Radioactive.Fast.Prototype = 'Left 4 Dead';
type = {'Viral','Radioactive'};
speed = {'Fast','Slow'};
for t = 1:length(type)
    for s = 1:length(speed)
        zombies.(type{t}).(speed{s})(1).Prototype
    end
end


ans =

28 Days Later


ans =

The Walking Dead


ans =

Left 4 Dead


ans =

Night of the Living Dead
```

Conclusion

In short, structures are a very convenient way to organize data in an easily-readible format. They are also a convenient way to output data from a function when you don't know how many outputs you'll have. For example, at one point I was writing a function to analyze fluorescence data from a plate reader. I wanted the function to only output the spectra of wells that had samples (not the empty wells), no matter how many wells the user used. So my function looked something like this:

```
function wells = AnalyzePlate(file)

wells.A(1) = ...

wells.A(2) = ...

wells.B(1) = ...
```

Well enough of zombies. Now I'll move on to putting a GUI together.

Published with MATLAB® R2015a