Putting a pointer in a struct

Edit your `carstruct__define.pro` in `gvim`. Add a new field, `serviced_at_miles`, which is a `ptr_new([])` object.

Now, interactively, make two `carstruct` instances. `car1 = {carstruct}` etc.

Set the mileage of the cars to new arrays with values `[5000,10000,15000]` and `[3000,6000,9000,12000]`. You'll need to use `ptr_new` to do this, e.g.:
`arptr = ptr_new([1,2,3,4,5])`

Then, to show what you've done, print out the values of `car1.serviced_at_miles` and `car2.serviced_at_miles`. Remember, to get the *values* instead of things that look like `<PtrHeapVar2>`, you need to use `*`, e.g.: `print,*arptr`.

Finally, change the value of `serviced_at_miles` for `car1` to be `[5000,10000,15000,20000,25000]`. You *do not* need to use `ptr_new` for this, instead you can make the change directly, e.g.:
`*arptr=[1,2,3,4,5,6]`

System Variables (global) & Unit Tests

Write a procedure `define_mks_units.pro` to define MKS units.

For now, use at least the following constants:
`AU, G, kmpers, year, parsec`

Use the `defsysv` procedure to make a system variable `!mks_units` that is a structure with the appropriately defined constants. This means that you'll make a structure such that, if you wanted to get `G`, you would:
`print,!mks_units.G`

Help for `defsysv` can be found at http://www.exelisvis.com/docs/DEFSYSV.html and http://idlastro.gsfc.nasa.gov/idl_html_help/DEFSYSV.html.

This tutorial is intended to give you practice with *testing* code. We covered this example in Lecture 17 (see page 33 and later), so it may be helpful to refer back to that document.

Edit the file `tutorial18_testing/test_mks_units.pro`. It contains lots of comments explaining what it does. Read through those notes to try to gain some understanding of what's going on. There's more discussion below if you're still confused.

The code *should* run right now, but it will give you a bunch of failures. Your goal is to turn these failures into PASSes.

Remember to `git commit -a` and `git push` your code.

```
IDL> test_mks_units
             Tag Name              Command                Value
                   au          X Failed X          X Failed X
             kmpersec          X Failed X          X Failed X
               parsec          X Failed X          X Failed X
                 year          X Failed X          X Failed X
```

Note that the line:
`cmd = 'OK = [something]'+ tag +' eq '+string(val)`
is intended to create commands to be executed with the `execute` function. Try this example:

```
cmd = "checkvar = 1+1 eq 2"
status = execute(cmd,1,1)
```

Then see what each variable is set to: `help,cmd,status,checkvar`

What if we have an array and want to index it in the `cmd`?

```
x = ['a','b','d','c']
results = intarr(4)
for ii=0,3 do results[ii] = execute("print,x["+string(ii)+"]")
print,results
```

The output of `execute` is `1` if it succeeds.

What happens if you try to run an invalid command? `print,execute('fail')`

For this tutorial, you need to change the `[something]` in the code you've been given and add more key/value pairs to test in the hash defined in that code (for example, `'G'`).

You should make one small change at a time and run the code after each change. Better not to break the whole thing in multiple places!