Structures

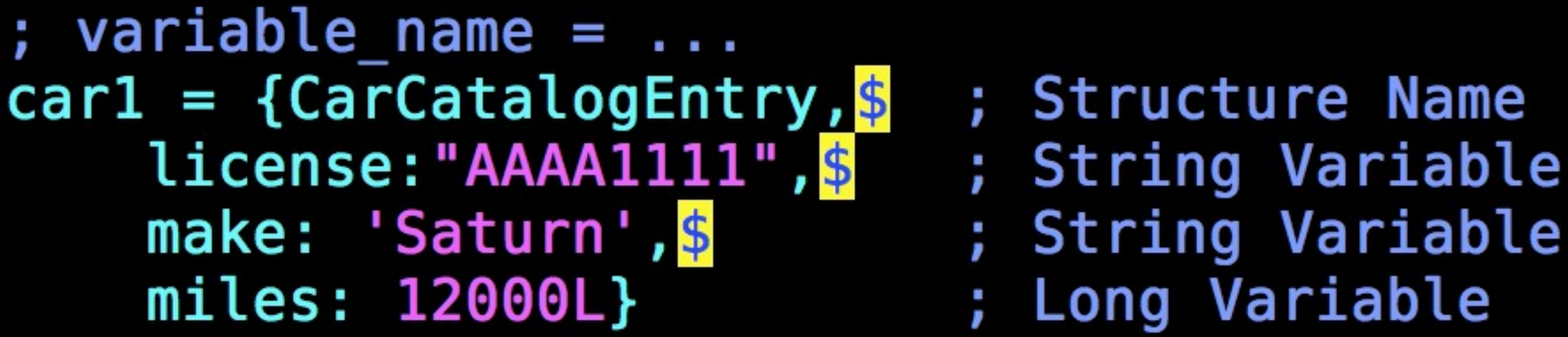
Structures

- are a means of organizing data
- they are hierarchical
  - things are ranked or ordered
- Similar to the way humans think about data

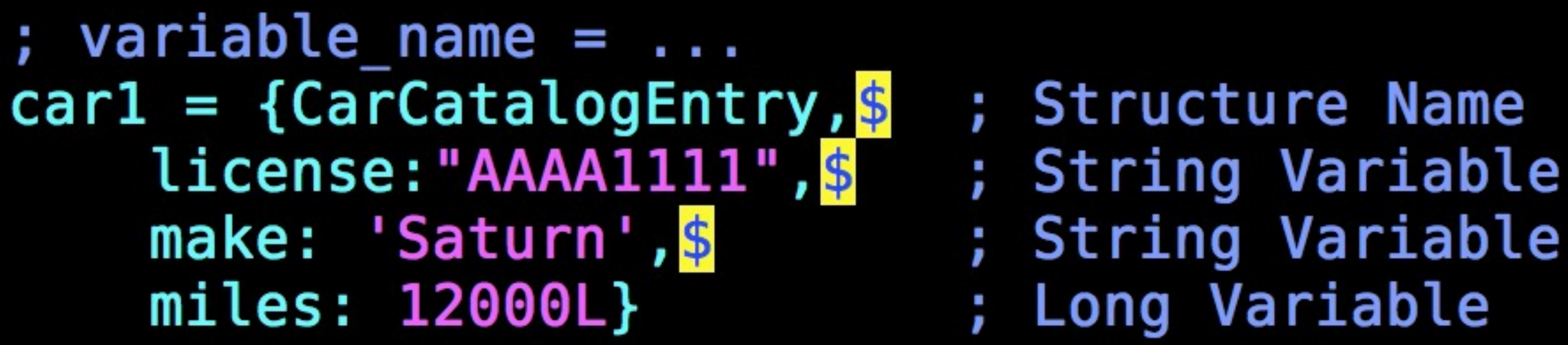
Example

- You see 5 cars
    - The cars each have a steering wheel, 4 tires, 4 doors, etc.
- If you were to think of cars as array-like, you would have 5 cars, 5 steering wheels, 20 tires, 20 doors, etc.
    - They would not be directly associated either; the tires aren’t on cars

Defining a Structure



```
; variable_name = ...
car1 = {CarCatalogEntry,$  ; Structure Name
    license:"AAAA1111",$   ; String Variable
    make: 'Saturn',$       ; String Variable
    miles: 12000L}         ; Long Variable
```



```
; variable_name = ...
car1 = {CarCatalogEntry,$   ; Structure Name
    license:"AAAA1111",$    ; String Variable
    make: 'Saturn',$        ; String Variable
    miles: 12000L}          ; Long Variable
```

- Does 3 things:

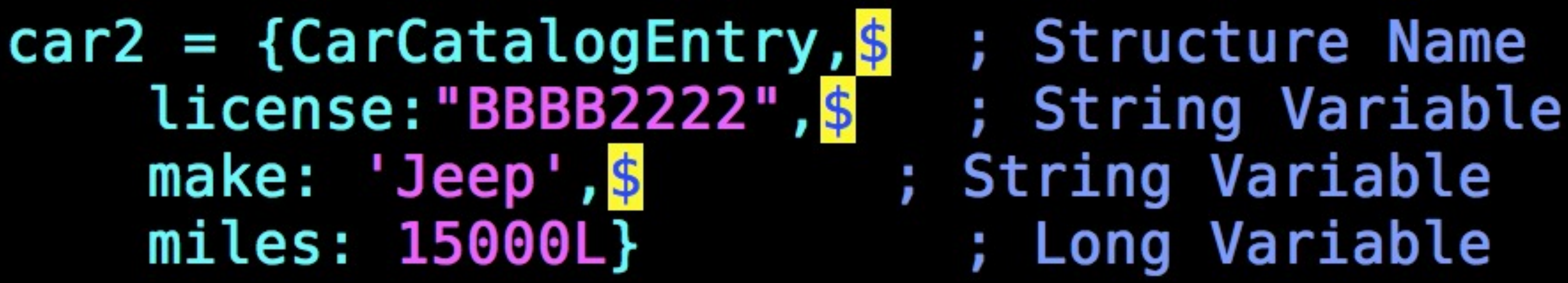
1. Defines the layout of the structure
    - It has `license`, `make`, and `miles`
2. Creates an *instance* of the structure, called `car1`
3. Sets the values of each field in the structure
    - variables that are parts of structs are called fields

Structured Data

- Like any data you would enter into forms
- A car has associated information, so does a person in a database, or a star

Making a catalog of cars

- Make new ones the same way:



```
car2 = {CarCatalogEntry,$   ; Structure Name
    license:"BBBB2222",$    ; String Variable
    make: 'Jeep',$        ; String Variable
    miles: 15000L}          ; Long Variable
```

What data type have you seen before that is similar to structure as I’ve presented them so far?

A) List

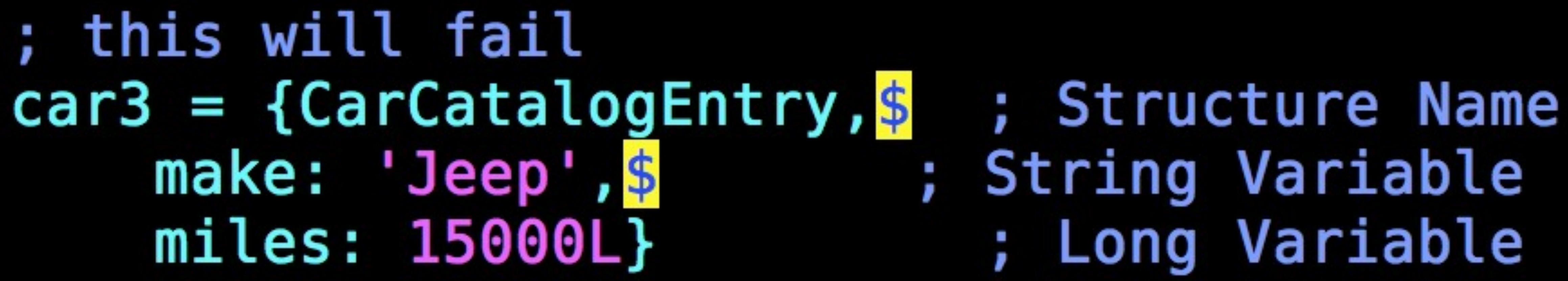
B) Hash

C) Array

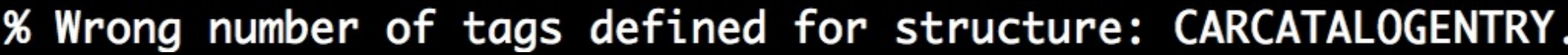
D) Foreach

E) None of the Above

What if you leave something out?



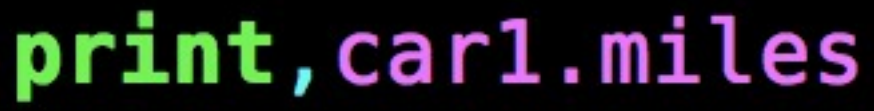
```
; this will fail
car3 = {CarCatalogEntry,$  ; Structure Name
    make: 'Jeep',$         ; String Variable
    miles: 15000L}           ; Long Variable
```



```
% Wrong number of tags defined for structure: CARCATALOGENTRY.
```

- You cannot change the layout of a structure once it’s been declared

Accessing Fields

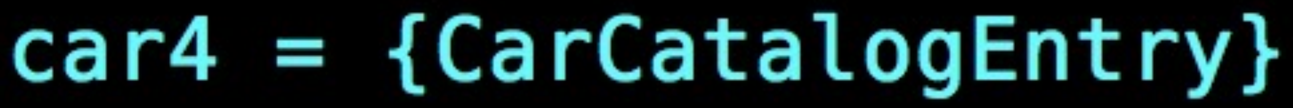


```
print,car1.miles
```

- Structures’ fields are accessed with a “dot”, so the above would be said “print car dot miles”

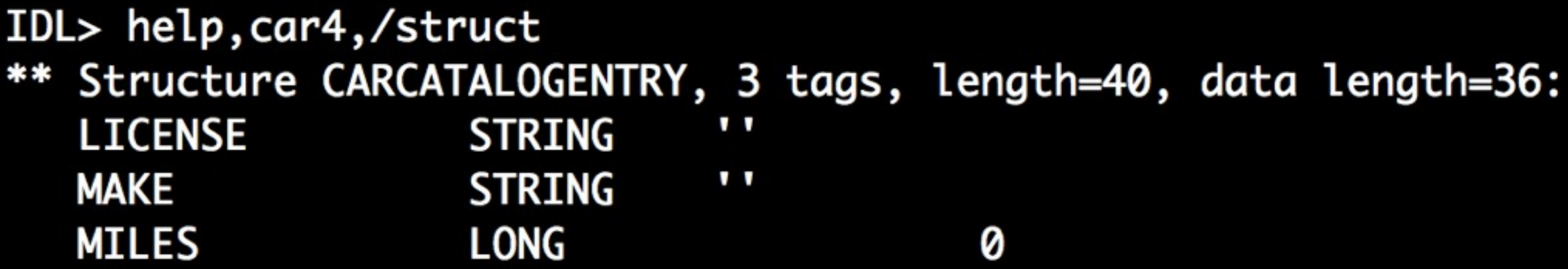
You can create blank entries

- It’s values will be blanks or zeros:



```
car4 = {CarCatalogEntry}
```

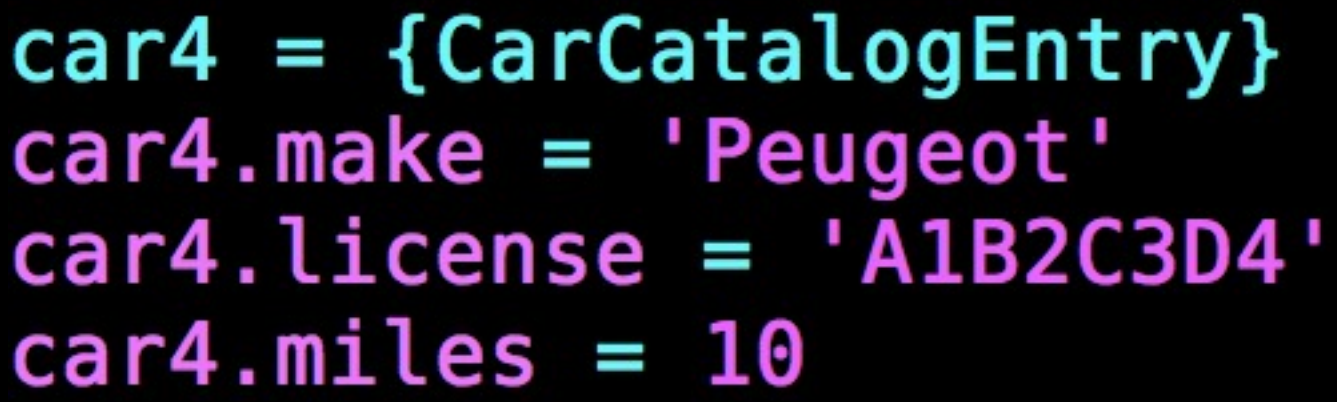
- You can check this with `help,/struct`:



```
IDL> help,car4,/struct
** Structure CARCATALOGENTRY, 3 tags, length=40, data length=36:
   LICENSE          STRING    ''
   MAKE             STRING    ''
   MILES            LONG                 0
```

Field Assignment

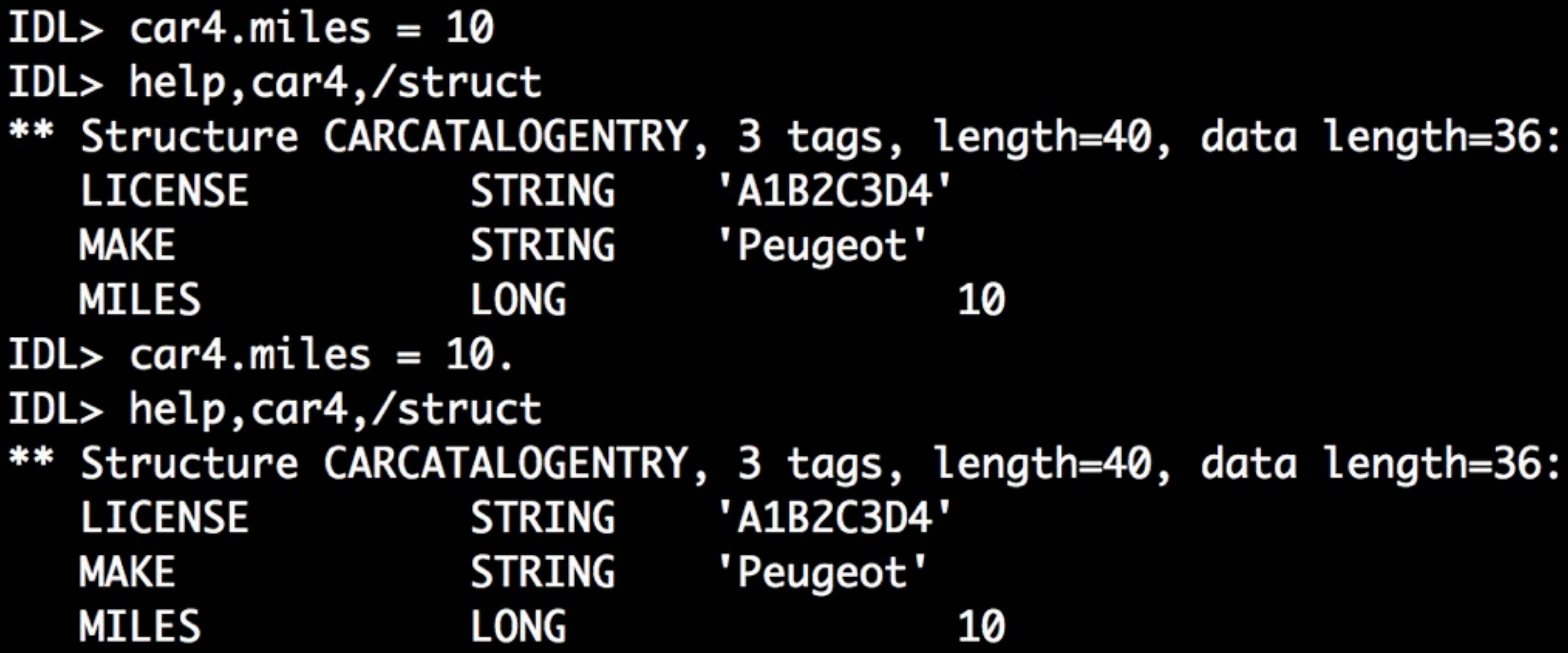
- Can now assign each individual field:



```
car4 = {CarCatalogEntry}
car4.make = 'Peugeot'
car4.license = 'A1B2C3D4'
car4.miles = 10
```

- `car4.miles` has the *same type* as `car1.miles`: the assigned value is automatically promoted *or* demoted

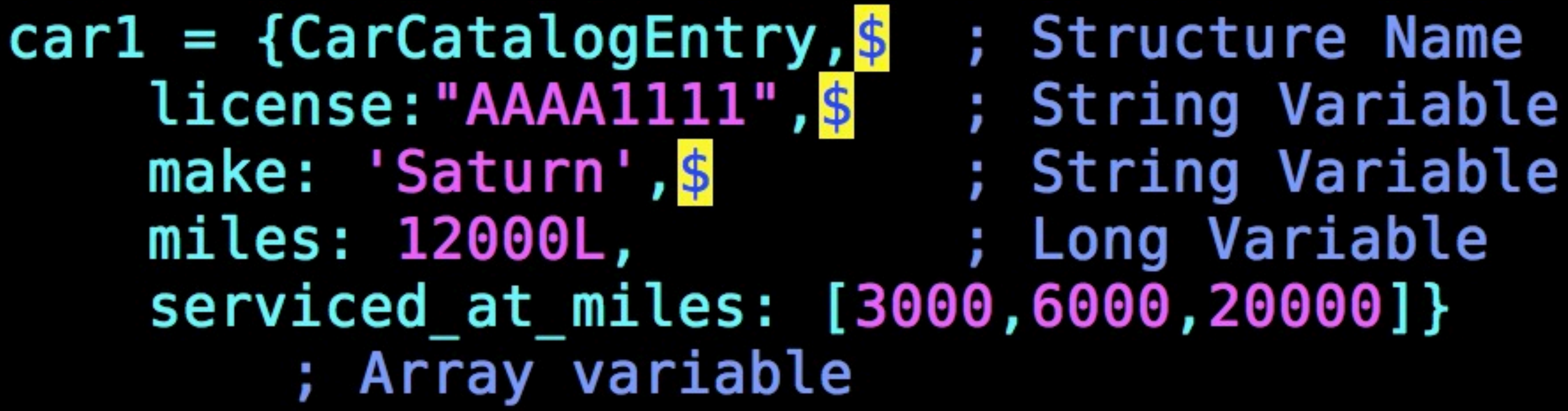
Field Promotion/ Demotion



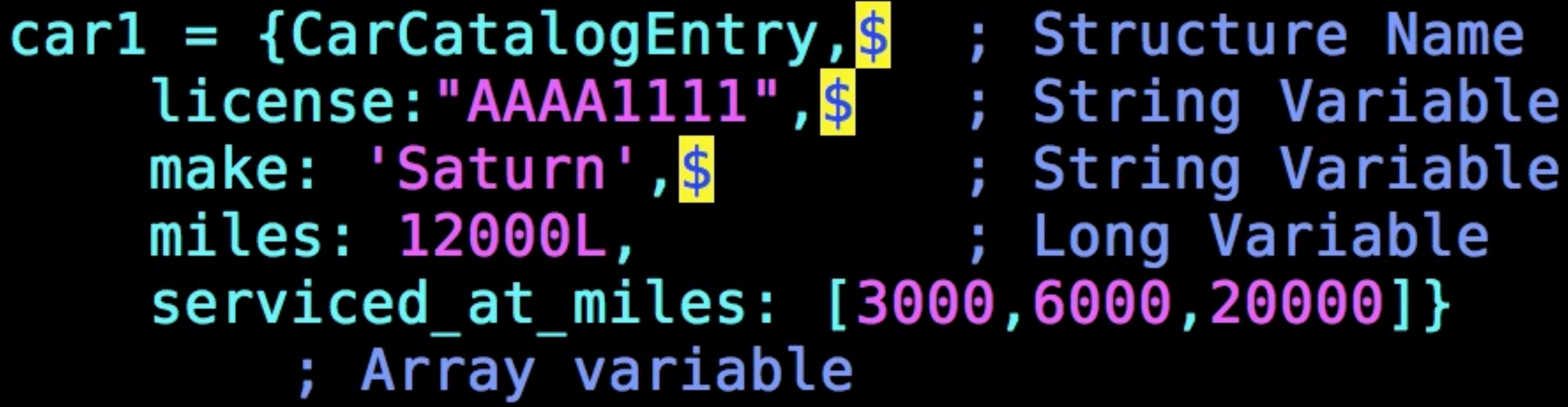
```
IDL> car4.miles = 10
IDL> help,car4,/struct
** Structure CARCATALOGENTRY, 3 tags, length=40, data length=36:
   LICENSE         STRING    'A1B2C3D4'
   MAKE            STRING    'Peugeot'
   MILES           LONG                10
IDL> car4.miles = 10.
IDL> help,car4,/struct
** Structure CARCATALOGENTRY, 3 tags, length=40, data length=36:
   LICENSE         STRING    'A1B2C3D4'
   MAKE            STRING    'Peugeot'
   MILES           LONG                10
```

Fields can be *any* type

- Including arrays, lists, hashes, etc.
- Let’s talk arrays first:



```
car1 = {CarCatalogEntry,$  ; Structure Name
    license:"AAAA1111",$   ; String Variable
    make: 'Saturn',$       ; String Variable
    miles: 12000L,         ; Long Variable
    serviced_at_miles: [3000,6000,20000]}
        ; Array variable
```



```
car1 = {CarCatalogEntry,$  ; Structure Name
    license:"AAAA1111",$   ; String Variable
    make: 'Saturn',$       ; String Variable
    miles: 12000L,         ; Long Variable
    serviced_at_miles: [3000,6000,20000]}
        ; Array variable
```

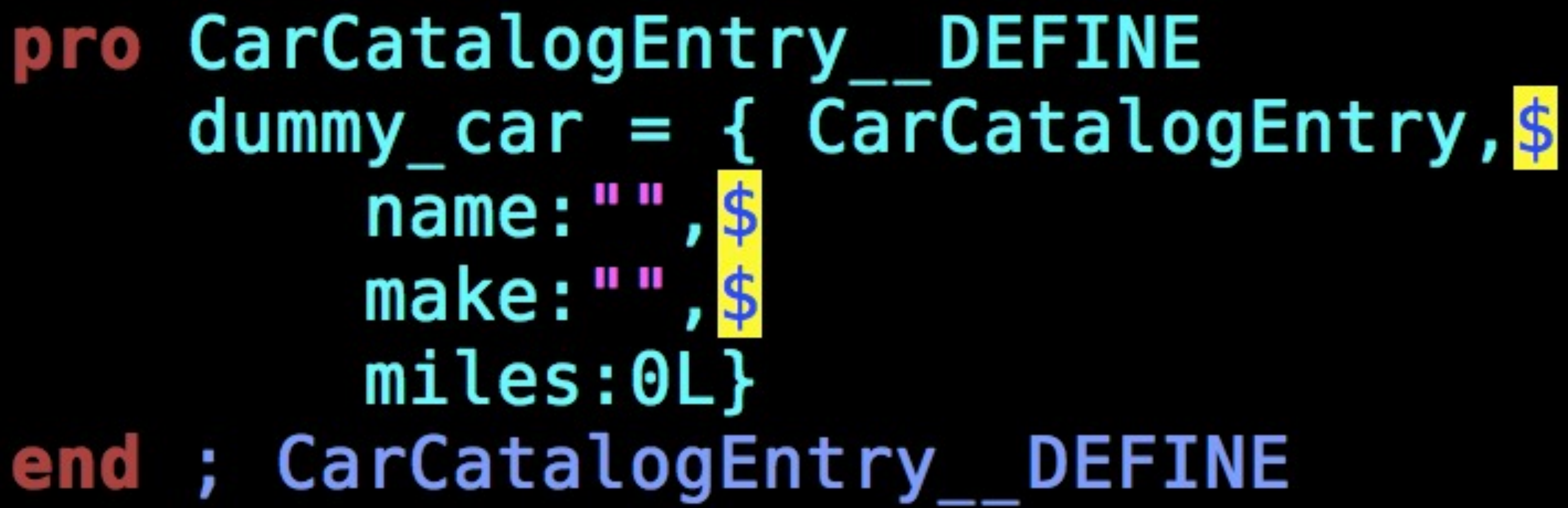
- `car.serviced_at_miles` has 3 entries and is an integer array
    - This can never be changed!
- But you can access each:
    - `print,car.serviced_at_miles[1]`
      `6000`

Re-declaring structs

- Once a structure of some name has been declared, its layout can never be changed
- If you want to re-define a structure's layout, you must `.reset_session`

Defining Structures

- There is a “right way” to define a structure



```
pro CarCatalogEntry__DEFINE
    dummy_car = { CarCatalogEntry,$
        name:"",$
        make:"",$
        miles:0L}
end ; CarCatalogEntry__DEFINE
```

Defining Structures

- If there is no structure defined with a given name, IDL will go looking through your path for a procedure with the name `[structurename]__DEFINE`
- This is better because the structure is defined in one central location
  - reduces risk of typos by not duplicating code

Defining Structures

- *If there is no structure defined* with a given name, IDL will go looking through your path for a procedure with the name `[structurename]__DEFINE`
- This is better because the structure is defined in one central location
  - reduces risk of typos by not duplicating code

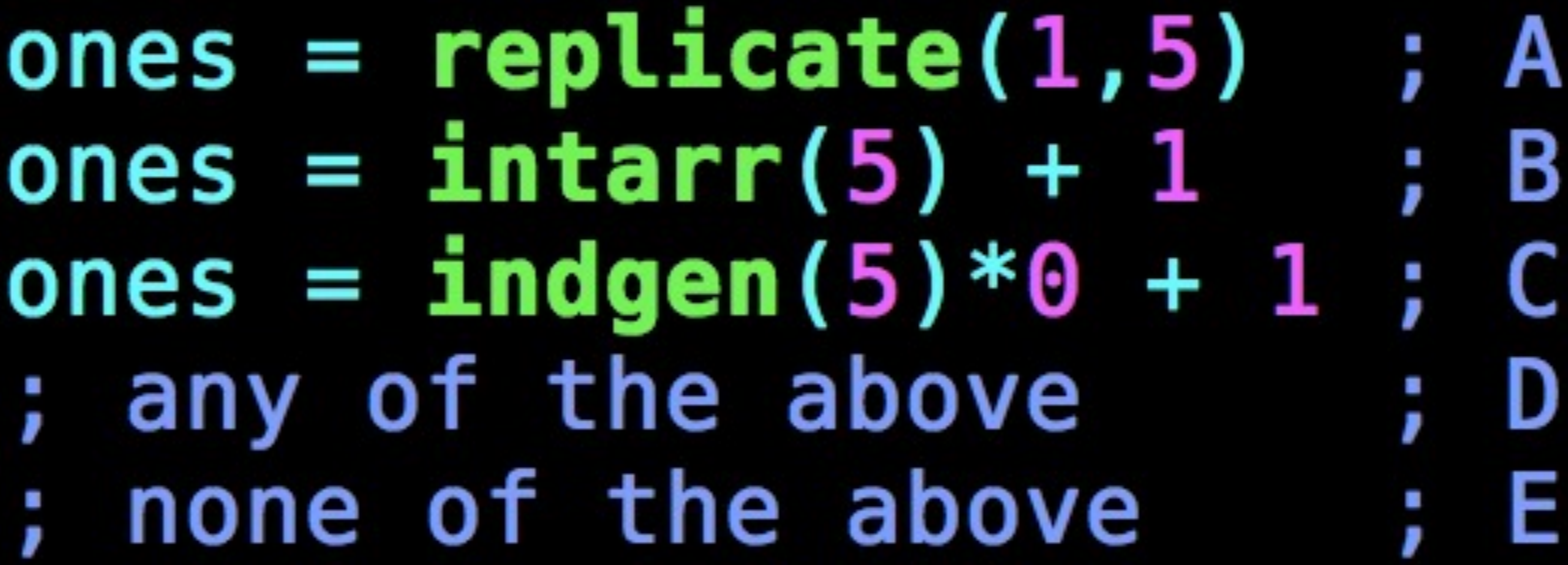
Define structures *only once*

- If you have a `StructureName__DEFINE` procedure, you should never define the same structure the other way (with `{}`)
    - The definitions using `{}` take precedence and can override your nice `__DEFINE` procedure, defeating the point and introducing risk of error

Arrays of Structures

- You can have arrays of structures!
- Since structures define a “type”, they can be treated similar to other structures
- But there’s no “structurearr” procedure
- Instead, we’ll use `replicate`

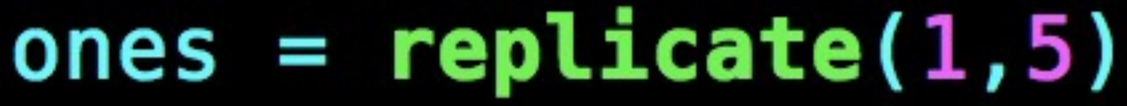
How do you make an array with 5 ones?



```
ones = replicate(1,5)   ; A
ones = intarr(5) + 1    ; B
ones = indgen(5)*0 + 1 ; C
; any of the above      ; D
; none of the above     ; E
```

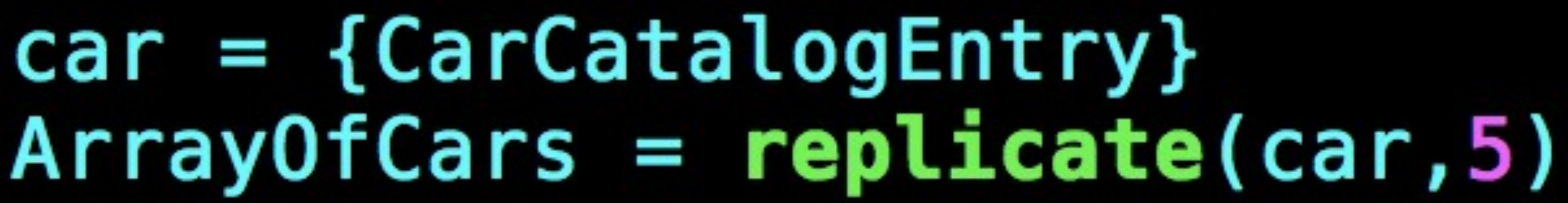
Structure Arrays

- If you want to make an array of length five with float 1’s, there are many options...



```
ones = replicate(1,5)
```

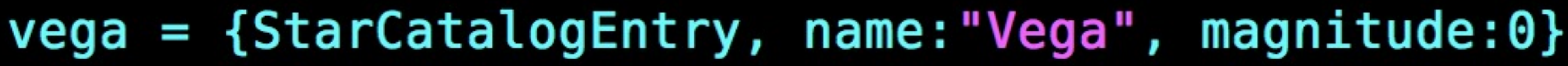
- For structures, you must use replicate, and the first argument is an *instance* of the structure



```
car = {CarCatalogEntry}
ArrayOfCars = replicate(car,5)
```

Type Vs Instance

- An instance is a particular occurrence of a type
- float, int, long, double, CarCatalogEntry are all types
- 1.0, 2, 3L, 5D, car1 are all instances of their respective types



```
vega = {StarCatalogEntry, name:"Vega", magnitude:0}
```

Is `vega` an instance or a type?

A) instance

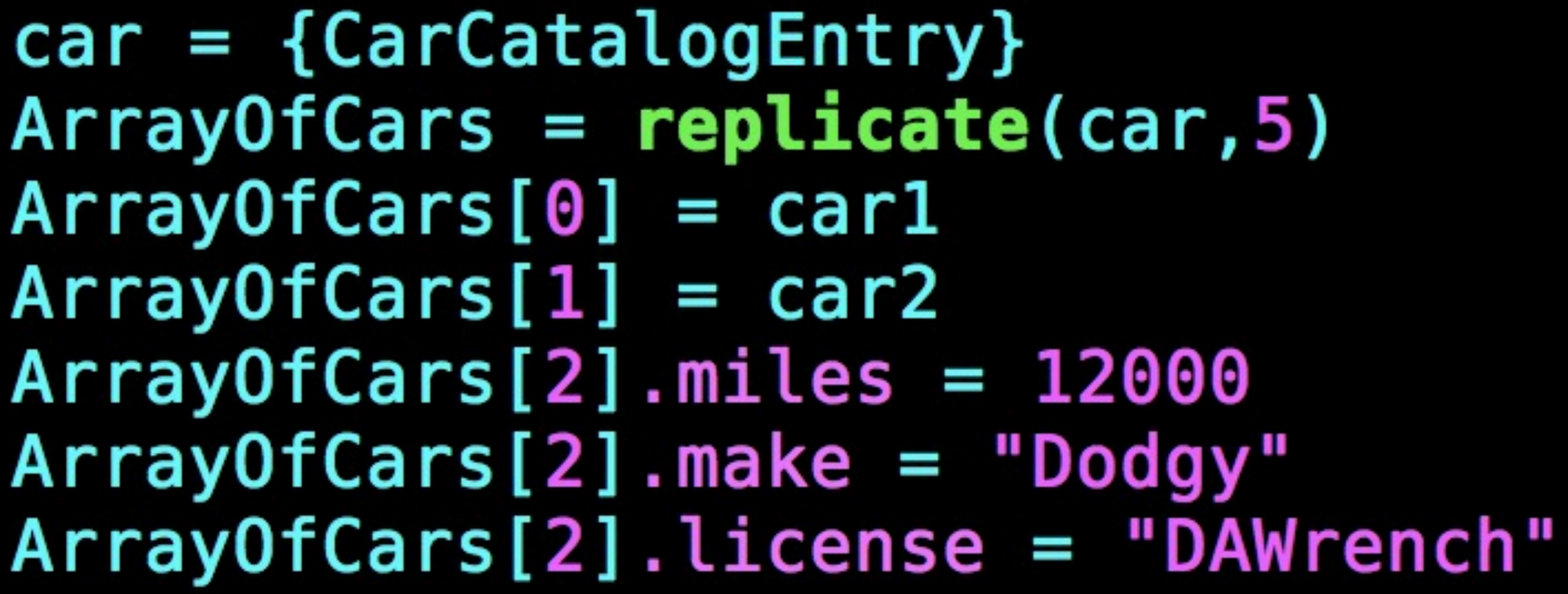
B) type

C) Neither

D) I don’t know

Structure Arrays

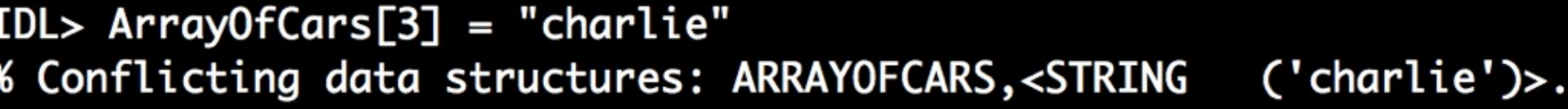
- Can access and assign like other arrays
  - but, promotion is not allowed as normal (the type of any field cannot be changed)



```
car = {CarCatalogEntry}
ArrayOfCars = replicate(car,5)
ArrayOfCars[0] = car1
ArrayOfCars[1] = car2
ArrayOfCars[2].miles = 12000
ArrayOfCars[2].make = "Dodgy"
ArrayOfCars[2].license = "DAWrench"
```

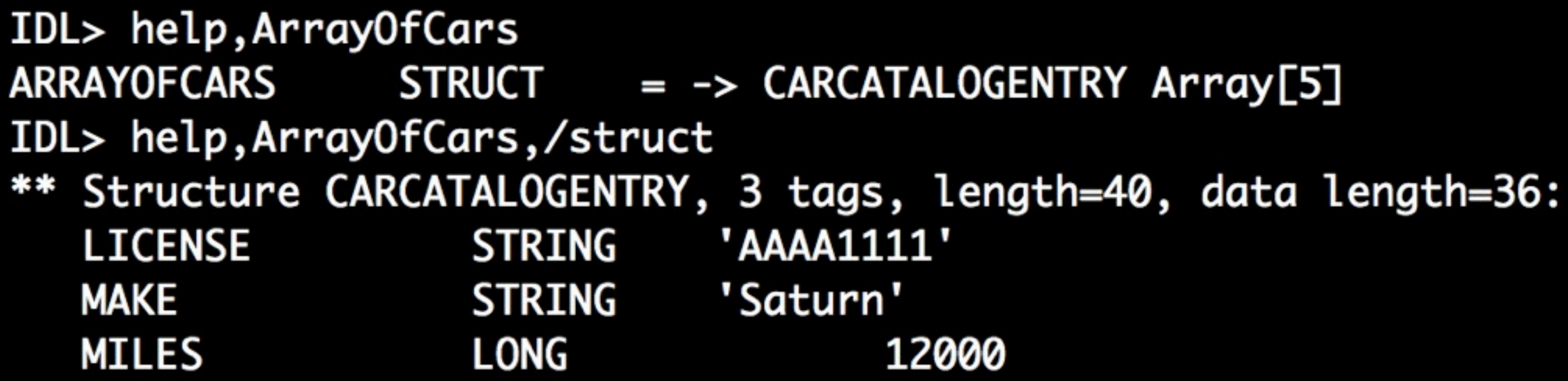
Structure Arrays

- Can’t assign elements of structure arrays to be anything but structs:



```
IDL> ArrayOfCars[3] = "charlie"
% Conflicting data structures: ARRAYOFCARS,<STRING   ('charlie')>.
```

The help...

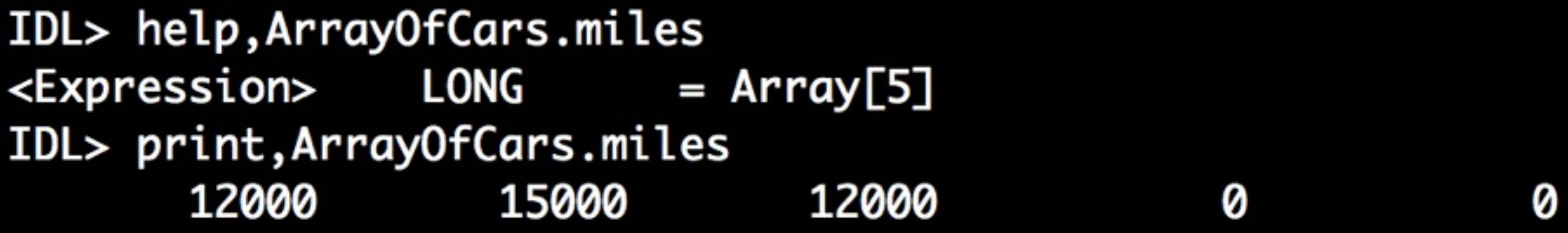


```
IDL> help,ArrayOfCars
ARRAYOFCARS     STRUCT    = -> CARCATALOGENTRY Array[5]
IDL> help,ArrayOfCars,/struct
** Structure CARCATALOGENTRY, 3 tags, length=40, data length=36:
   LICENSE          STRING    'AAAA1111'
   MAKE             STRING    'Saturn'
   MILES            LONG             12000
```

- Only gives # of entries or the first entry (depending on whether you use `/struct`)

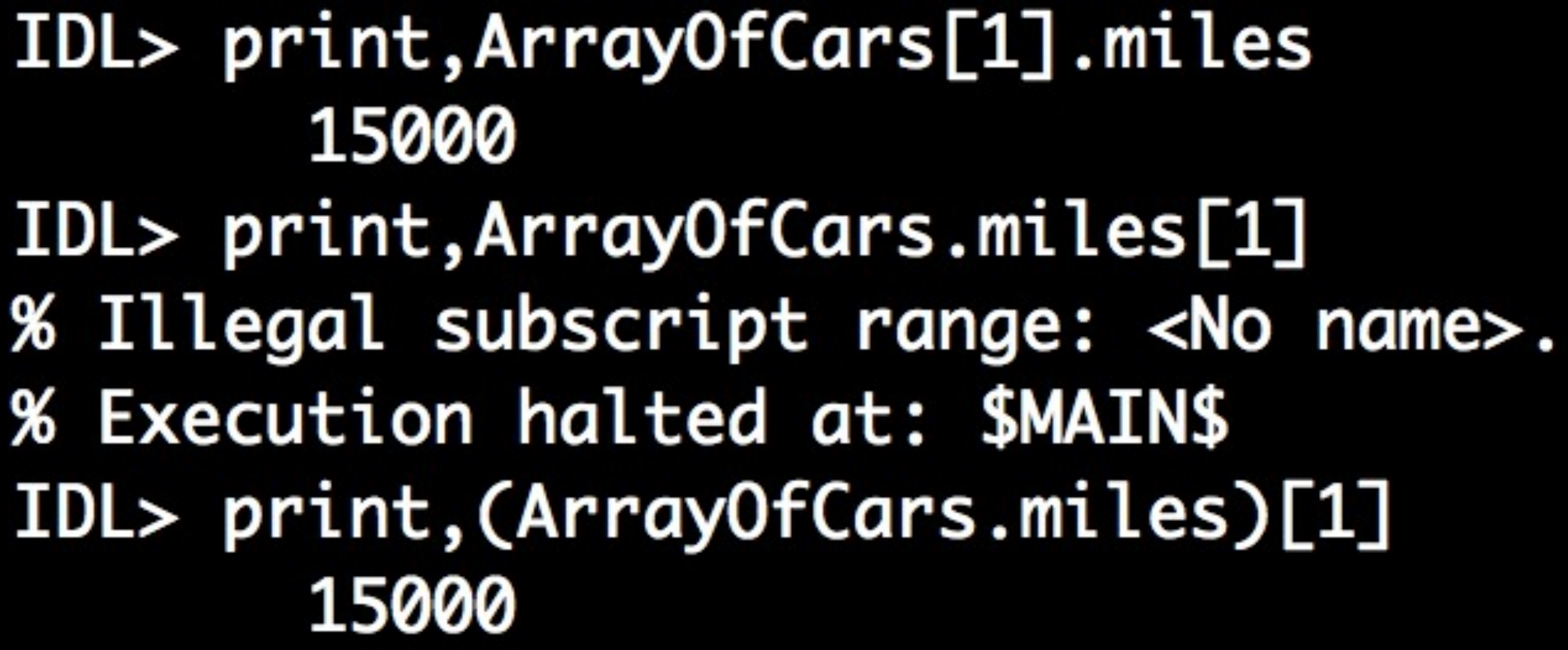
Array of Structures

- Now each “field” is its own array:.. kinda



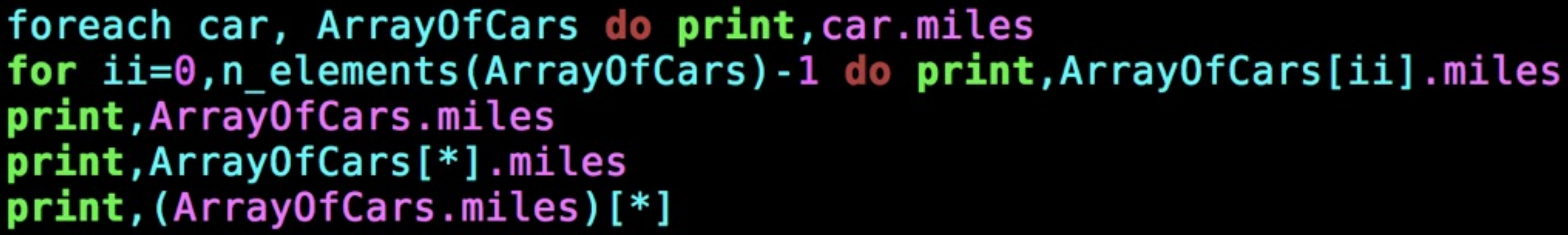
```
IDL> help,ArrayOfCars.miles
<Expression>    LONG      = Array[5]
IDL> print,ArrayOfCars.miles
       12000       15000       12000           0           0
```

- You actually have to trick IDL...



```
IDL> print,ArrayOfCars[1].miles
       15000
IDL> print,ArrayOfCars.miles[1]
% Illegal subscript range: <No name>.
% Execution halted at: $MAIN$
IDL> print,(ArrayOfCars.miles)[1]
       15000
```

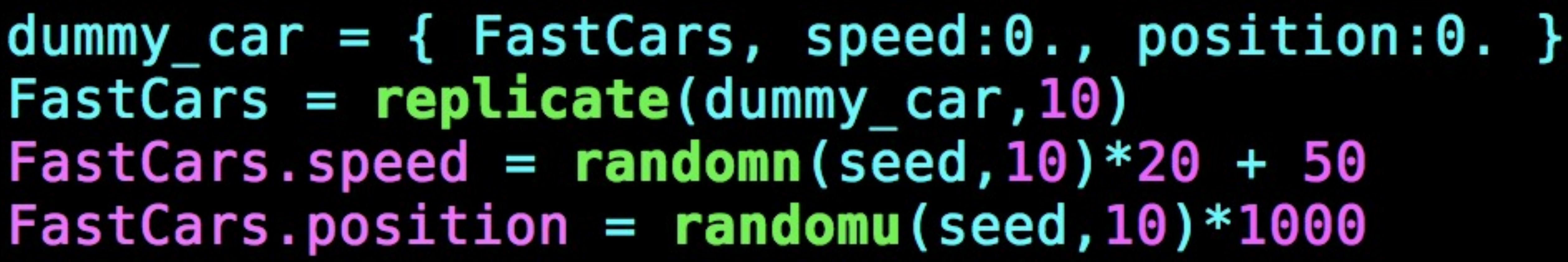
Some (nearly) equivalent ways to get data



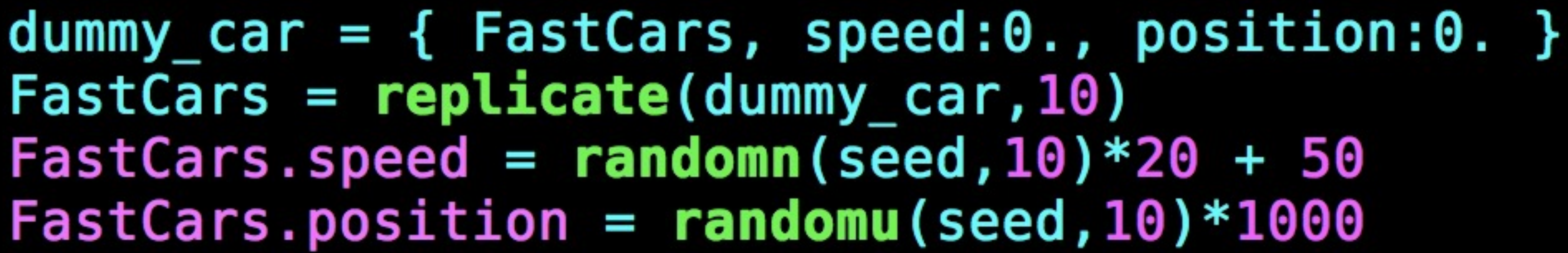
```
foreach car, ArrayOfCars do print,car.miles
for ii=0,n_elements(ArrayOfCars)-1 do print,ArrayOfCars[ii].miles
print,ArrayOfCars.miles
print,ArrayOfCars[*].miles
print,(ArrayOfCars.miles)[*]
```

Using Structure Arrays

- Say you have a list of cars, each having X positions on the highway and velocities V...

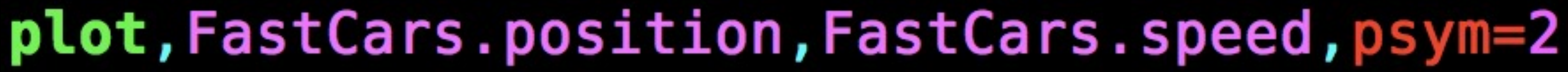


```
dummy_car = { FastCars, speed:0., position:0. }
FastCars = replicate(dummy_car,10)
FastCars.speed = randomn(seed,10)*20 + 50
FastCars.position = randomu(seed,10)*1000
```

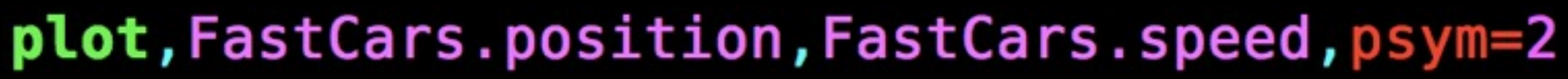


```
dummy_car = { FastCars, speed:0., position:0. }
FastCars = replicate(dummy_car,10)
FastCars.speed = randomn(seed,10)*20 + 50
FastCars.position = randomu(seed,10)*1000
```

- Clearly, you want to plot these.

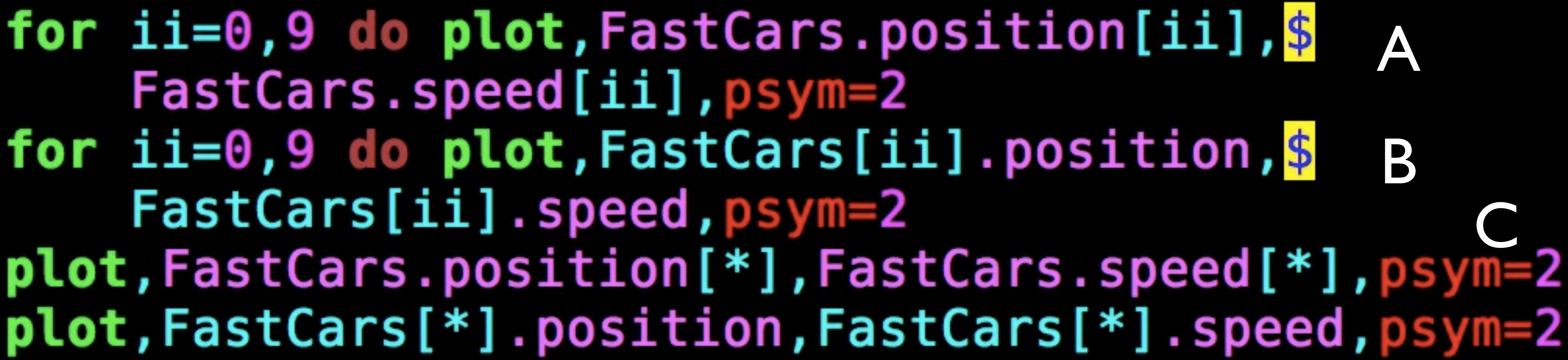


```
plot,FastCars.position,FastCars.speed,psym=2
```



```
plot,FastCars.position,FastCars.speed,psym=2
```

Which of the following is equivalent?



```
for ii=0,9 do plot,FastCars.position[ii],$
    FastCars.speed[ii],psym=2
for ii=0,9 do plot,FastCars[ii].position,$
    FastCars[ii].speed,psym=2
plot,FastCars.position[*],FastCars.speed[*],psym=2
plot,FastCars[*].position,FastCars[*].speed,psym=2
```

A

B

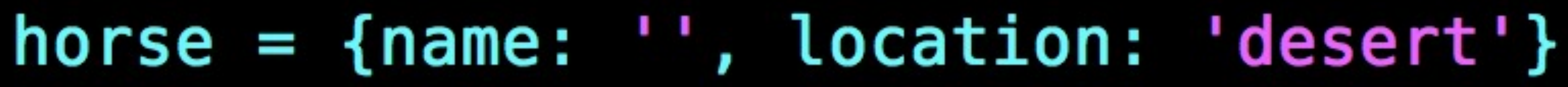
C

D

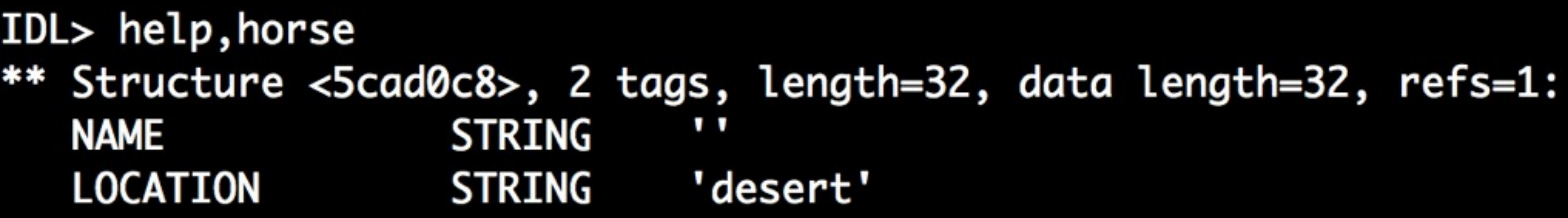
E. None of the above

Anonymous Structures

- You can create a structure with no name



```
horse = {name: '', location: 'desert'}
```



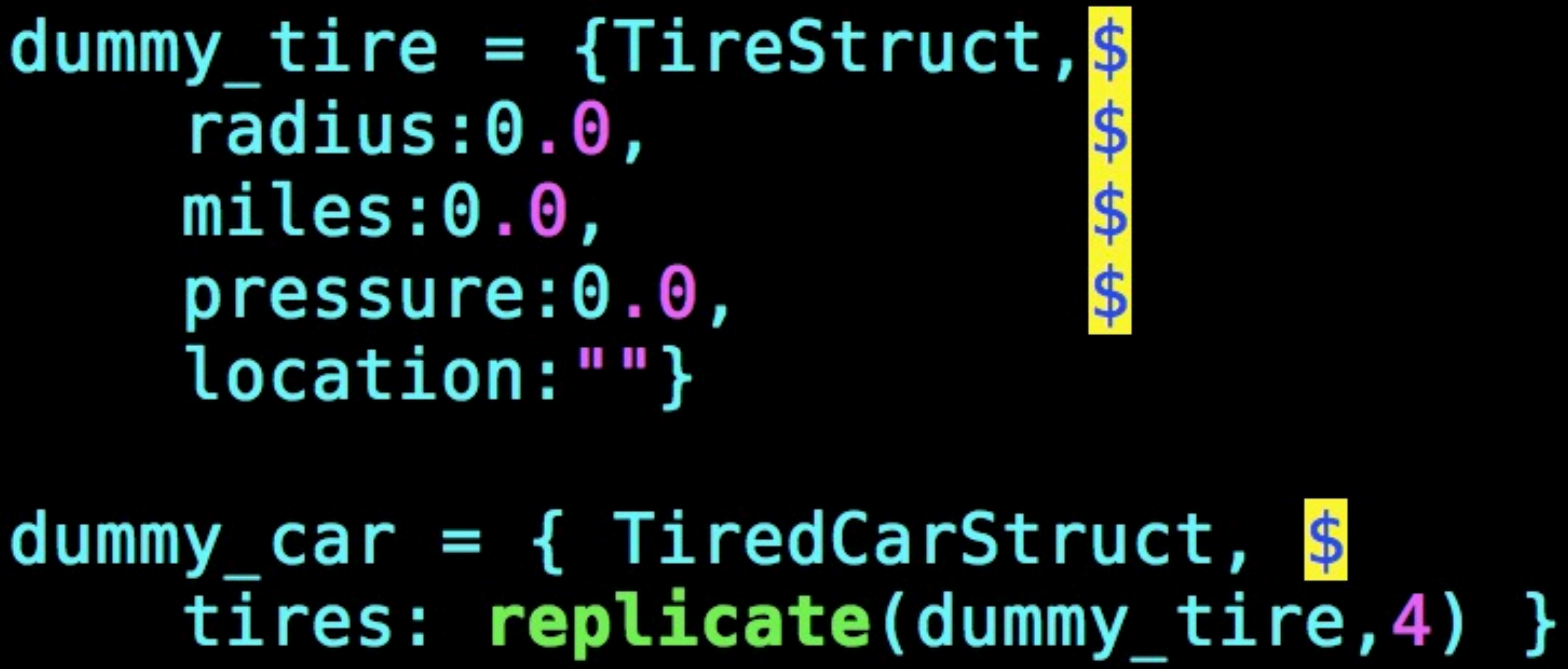
```
IDL> help,horse
** Structure <5cad0c8>, 2 tags, length=32, data length=32, refs=1:
   NAME            STRING    ''
   LOCATION        STRING    'desert'
```

- You don’t have to worry about conflicts, but you also don’t have a new type

Hierarchy

- Cars have tires
  - Tires have properties:
    - Location
    - # of miles driven
    - Radius
    - Pressure

Structs within Structs

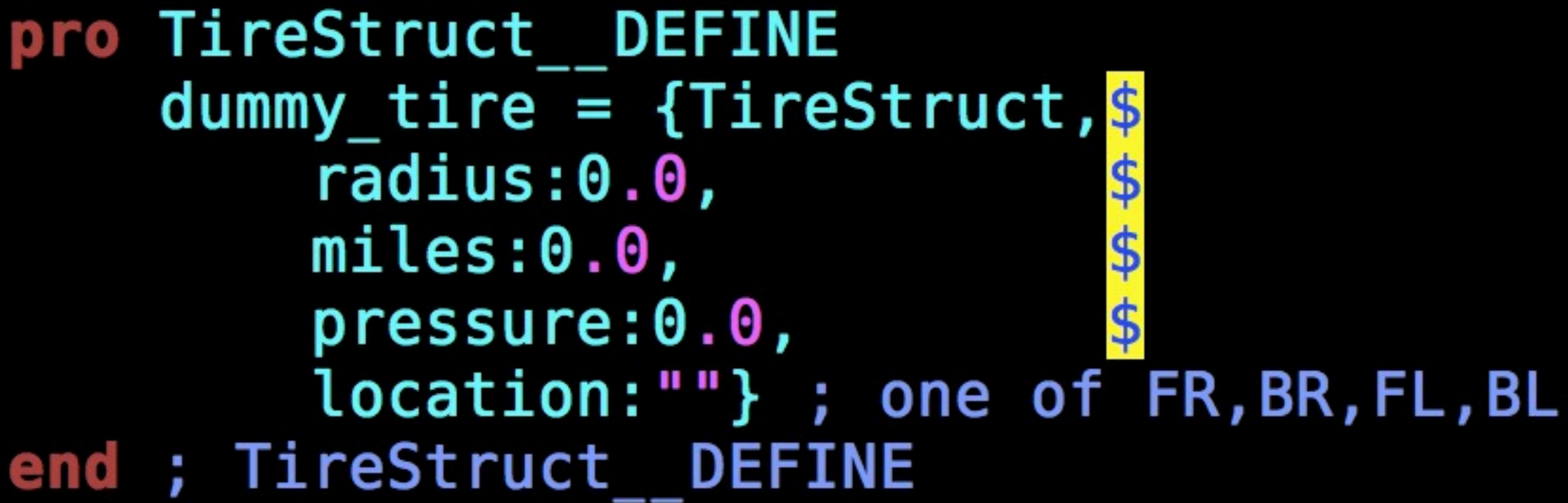


```
dummy_tire = {TireStruct,$
    radius:0.0,          $
    miles:0.0,           $
    pressure:0.0,        $
    location:""}

dummy_car = { TiredCarStruct, $
    tires: replicate(dummy_tire,4) }
```

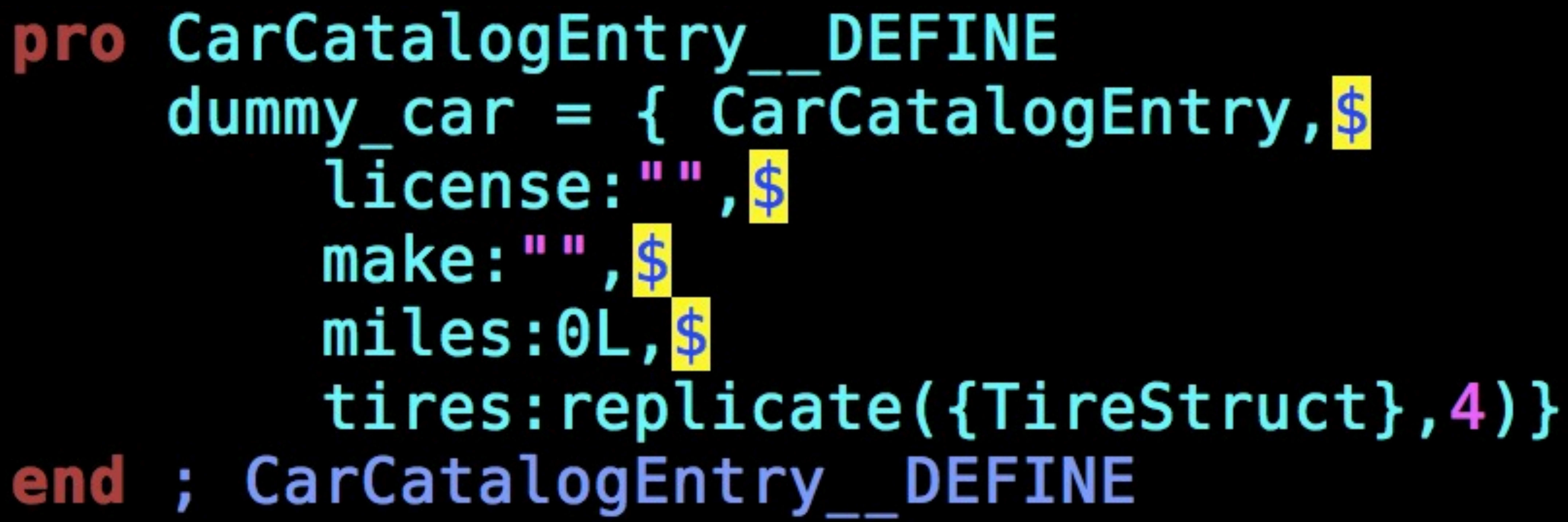
- Now all `TiredCarStruct` instances have 4 tires!

Building a hierarchy using your __DEFINES



```
pro TireStruct__DEFINE
    dummy_tire = {TireStruct,$
        radius:0.0,          $
        miles:0.0,           $
        pressure:0.0,        $
        location:""} ; one of FR,BR,FL,BL
end ; TireStruct__DEFINE
```

Building __DEFINE cont’d



```
pro CarCatalogEntry__DEFINE
    dummy_car = { CarCatalogEntry,$
        license:"",$
        make:"",$
        miles:0L,$
        tires:replicate({TireStruct},4)}
end ; CarCatalogEntry__DEFINE
```

READING

- Section 15.2, pages 17 - 19 in particular
- it’s important for the homework

Arrays vs Structures

- There was a time before structures
  - (it preceded the birth of anyone in this room)
- If you had a list of cars, you’d have to separate its properties:
  - array of makes
  - array of mileages
  - 4xN array of tire radii, pressures, locations...

Arrays vs Structures

- Many “parallel” arrays is an ugly way to represent data
    - it’s easy to get mixed up - what happens if one of the arrays gets truncated? Suddenly they’re all wrong.
    - It’s often harder to read

Aside: Celestial Coordinates

- Right Ascension
  - measured in hours
  - corresponds to longitude
- Declination
  - measured in degrees
  - corresponds to latitude