Structs

02-601

Organizing Contacts

Exercise: Say that you would like to organize a set of contacts. Each contact has:

- unique identifier
- first name
- last name
- phone number
- email address
- zip code
- (etc.)

How should we organize the data into a *single* data structure?

Frequent Words: Map of Maps

Key	Value
Bacterium A	
Bacterium B	•
Bacterium C	•
Bacterium D	•
Bacterium E	•
Bacterium F	•
Bacterium G	

Key	Value
"ATGCACGCT"	8
"GGACGTACG"	1
"GTACGACAG"	2
"ATAAATTGC"	6
"GATACCAGA"	2

Key	Value
"GTACGACGA"	1
"AACATACGG"	3
"GATACACAC"	7
"CTACCAGTA"	2
"TATCATCGG"	4

Storing Phone Contacts: Map of Maps?

Key	Value
dwatson	
rpetty	•
jcole	•
dearnhardt	•
mjordan	

Key	Value
firstName	“Doc”
lastName	“Watson”
phone	9835401
zipCode	27421
email	dwatson@cmu.edu

Key	Value
firstName	“Michael”
lastName	“Jordan”
phone	3219840
zipCode	28037
email	mjordan@cmu.edu

Storing Phone Contacts: Map of Maps?

Key	Value
dwatson	
rpetty	•
jcole	•
dearnhardt	•
mjordan	

Key	Value
firstName	"Doc"
lastName	"Watson"
phone	9835401
zipCode	27421
email	dwatson@cmu.edu

Key	Value
firstName	"Michael"
lastName	"Jordan"
phone	3219840
zipCode	28037
email	mjordan@cmu.edu

Think: Is this reasonable?

A Better Data Structure

Because every contact has the *same* properties (of *different* types), we should create a Contact **object**.

In Go, this is called a **struct**.

```
type Contact struct {
   firstName string
   lastName string
   phone []int
   email string
   zipCode [5]int
}
```

firstName, lastName, etc. are called **fields**.

A Better Data Structure

This is a generalization of what we saw before, when we defined a game board as equivalent to a [][]int.

```
type GameBoard [][]int
```

Declaring a Struct Variable

Declaring a struct variable is the same as declaring another variable.

```
var me Contact
```

Declaring a Struct Variable

Declaring a struct variable is the same as declaring another variable.

```
var me Contact
```

Accessing the fields of an object can be done with "objectName.fieldName"

```
me.firstName = "Phillip"
me.zipCode = [5]int{1, 5, 2, 1, 3}
// etc.
```

Initializing Struct Fields

Initially, all Contact fields are null. Any slices need to be “made” or else we will have a runtime error.

```
var you Contact
fmt.Println(you.firstName) // = “”
fmt.Println(you.Phone) // = []
fmt.Println(you.zipCode) // = [0 0 0 0 0]
```

Shortcut Declarations

Rather than set fields one at a time, we can do it all at once using a **struct literal**.

The name of the struct type

A field name

The value for the field

```
you := Contact{firstName: "Anna",
   lastName: "Johnson",
   phone: []int{4,1,2,3,4,5,9,8,7,6},
   email: "ajohnson@cmu.edu",
   zipCode: [5]int{1,5,2,1,3}, //need comma!
   }
```

Structs as Function Input/Output

Taking structs as a function argument:

```
func PrintContact(c Contact) {
  // insert code to print contact fields
}
```

Structs as Function Input/Output

Taking structs as a function argument:

```
func PrintContact(c Contact) {
   // insert code to print contact fields
}
```

Returning a struct as function output:

```
func CreateContact(name string) Contact {
   // create a new contact from name
}
```

Returning to Our Original Question

Exercise: Say that you would like to organize a set of contacts. Each contact has:

- unique identifier
- first name
- last name
- phone number
- email address
- zip code
- (etc.)

How should we organize the data into a *single* data structure?

Answer: Map Whose Values are Contacts!

Key	Value
`dwatson`	Contact1
`rpetty`	Contact2
`jcole`	Contact3
`dearnhardt`	Contact4
`mjordan`	Contact5

`firstName`	“Doc”
`lastName`	“Watson”
`phone`	9835401
`zipCode`	27421
`email`	dwatson@cmu.edu

`firstName`	“Michael”
`lastName`	“Jordan”
`phone`	3219840
`zipCode`	28037
`email`	mjordan@cmu.edu

Answer: Map Whose Values are Contacts!

Key	Value
dwatson	Contact1
rpetty	Contact2
jcole	Contact3
dearnhardt	Contact4
mjordan	Contact5

firstName	"Doc"
lastName	"Watson"
phone	9835401
zipCode	27421
email	dwatson@cmu.edu

firstName	"Michael"
lastName	"Jordan"
phone	3219840
zipCode	28037
email	mjordan@cmu.edu

```
var people map[string]Contact
```

Answer: Map Whose Values are Contacts!

Key	Value
dwatson	Contact1
rpetty	Contact2
jcole	Contact3
dearnhardt	Contact4
mjordan	Contact5

firstName	"Doc"
lastName	"Watson"
phone	9835401
zipCode	27421
email	dwatson@cmu.edu

firstName	"Michael"
lastName	"Jordan"
phone	3219840
zipCode	28037
email	mjordan@cmu.edu

```
var people map[string]Contact
people["dwatson"].firstName = "Doc" // ERROR!
```

Workaround for “Go Issue 3117”

```
people := make(map[string]Contact)

var tmp Contact
tmp.firstName = “Doc”
// set rest of fields…

people[“dwatson”] = tmp
```

We Have Already Been Working with Structs!

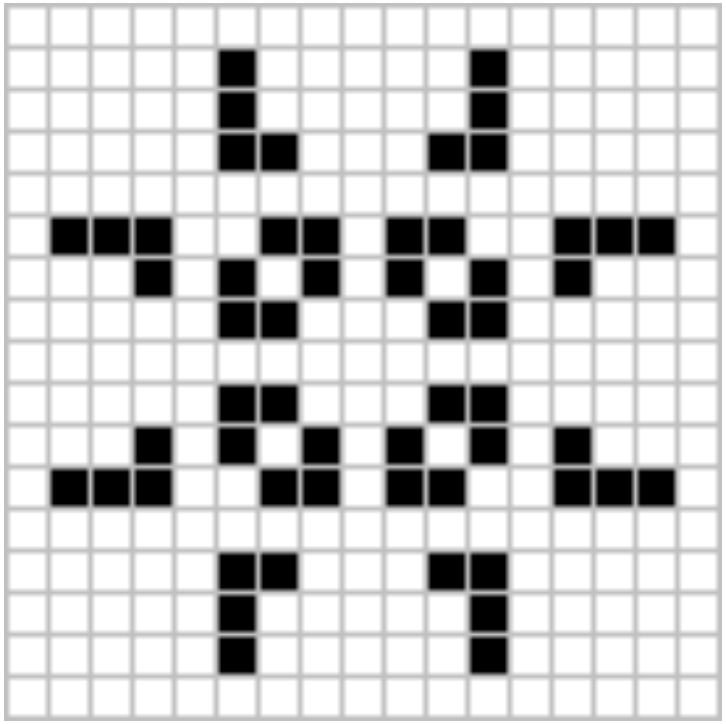
Don't worry about * for now

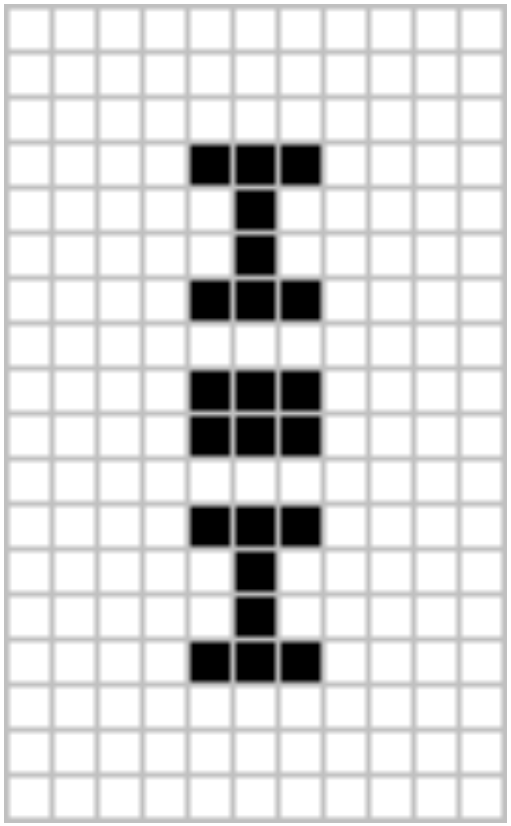
```
type Canvas struct {
   gc      *draw2d.ImageGraphicContext
   img     image.Image
   width   int
   height  int
}
```

An object that represents the image

The Game of Life Will Not Die

Last time: we hacked the Game of Life ...





... but how can we *see* what we have done?

This is Just a Slice of Images…

How to Create an Animated GIF

```
1. Given a GOL board, create a canvas c.
2. Place c.img into a []image.Image slice.
3. Use someone else's package to convert a
   []image.Image into an animated GIF.
4. Enjoy!
```

```go
type Canvas struct {
   gc      *draw2d.ImageGraphicContext
   img     image.Image
   width   int
   height  int
}
```

Animated GIF Code for GOL: See Piazza