Sorting and Inverting a Map

This lesson covers important details on sorting and inverting a map.

WE'LL COVER THE FOLLOWING

Sorting a map

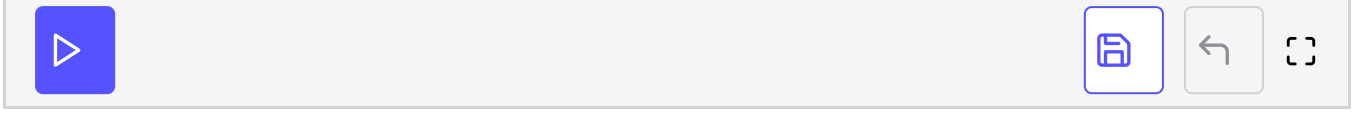
By default, a map is not sorted, not even on the value of its keys. If you want a sorted map, copy the keys (or values) to a slice, sort the slice (using the sort package), and print out the keys and/or values using the for-range on the slice. This is illustrated in the following program:

```
package main
import (
    "fmt"
    "sort"
)

var (
    barVal = map[string]int{"alpha": 34, "bravo": 56, "charlie": 23, "delta": 87,
    "echo": 56, "foxtrot": 12, "golf": 34, "hotel": 16, "indio": 87, "juliet": 65, "kilo":
    43, "lima": 98}
)

func main() {
    fmt.Println("unsorted:")
    for k, v := range barVal {
        fmt.Printf("key: %v, value: %v / ", k, v)   // read random keys
    }
    keys := make([]string, len(barVal))     // storing all keys in separate slice
    i := 0
    for k := range barVal {
        keys[i] = k
        i++
    }
    sort.Strings(keys)  // sorting the keys slice
    fmt.Println()
    fmt.Println("\nsorted:")
    for _, k := range keys {
```

```
        fmt.Printf("key: %v, value: %v / ", k, barVal[k])   // reading key from keys and valu
    }
}
```



Sorting a Map

In the code above, outside `main` at **line 4**, we import `sort` package to perform sorting, and at line 7, we make a map `barVal`. The declaration of `barVal` shows that its keys will be of *string* type and values associated with its keys will be of *int* type. The initialization is done at the same line. For example, the key `alpha` gets value **34**, `bravo` get **56**, `charlie` gets **23** and so on.

In `main`, we have a for-loop at **line 15**. This loop prints a key and the value associated with each key in every iteration, until all key-value pairs are not printed. The output of this for loop will be unsorted, as maps' keys are read randomly.

Now at **line 18**, we make a slice of *string* `keys` of length equal to `barVal`'s length (total number of keys in `barVel`). Then we make the for loop at **line 20** that reads keys from `barVel` in random order and stores them one by one in `keys` (at **line 21**). By the end of for loop, all keys will be present in the `keys` slice. There is a great chance that the keys from `barVal` will not be read in *sorted* (alphabetical) order and will be stored in the `keys` slice. So at **line 24**, we perform in-place sorting on the strings present in `keys` using: `sort.Strings(keys)`.

Again we have a for-loop at **line 27**. This loop prints a key and the value associated with each key in every iteration until all key-value pairs are not printed. It reads the key from `keys` and store it in `k`. Then read the value from `barVal` as: `barVal[k]`.The output of this for loop will be sorted, as `barVal`'s keys are read from `keys` slice (containing sorted strings).

Inverting a map

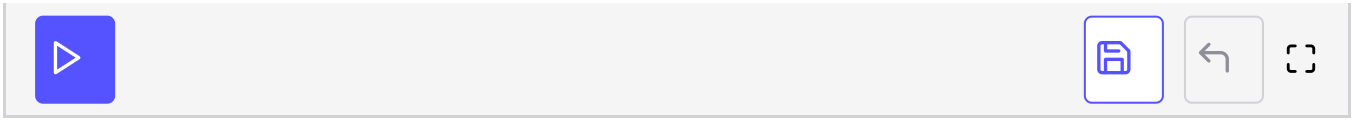
Inverting a map we mean switching the values and keys. If the value type of a map is acceptable as a key type and the map values are unique, this can be done easily. Look at the following program to see how a map can be inverted.

```
package main
```

```
import (
"fmt"
)

var (
barVal = map[string]int{"alpha": 34, "bravo": 56, "charlie": 23, "delta": 87,
"echo": 56, "foxtrot": 12, "golf": 34, "hotel": 16, "indio": 87, "juliet": 65, "kilo": 43,
"lima": 98}
)

func main() {
  invMap := make(map[int]string, len(barVal)) // interchanging types of keys and values
  for k, v := range barVal {
    invMap[v] = k   // key becomes value and value becomes key
  }
  fmt.Println("inverted:")
  for k, v := range invMap {
    fmt.Printf("key: %v, value: %v / ", k, v)
  }
}
```



In the above code, outside `main` at line **6**, we make a map `barVal`. The declaration of `barVal` shows that its keys will be of *string* type and values associated with its keys will be of *int* type. The initialization is done at the same line. For example, the key `alpha` gets value **34**, `bravo` get **56**, `charlie` gets **23** and so on.

In `main`, at line **13**, we make a map `invMap`. The declaration of `invMap` shows that its keys will be of *int* type and values associated with its keys will be of *string* type. The length of `invMap` is the same as `barVal`, and the map `invMap` is the inverse of `barVal`. That is why the type of keys and values are interchanged between them.

Then we have the for loop at **line 14**, where we read `barVal` key in `k` and value for each `k` in `v` for each iteration. To invert a map, the key becomes value and the value becomes key. This means that `invMap[v]=k` (see **line 15**) will serve the purpose. As `v` was the value associated to `k` key of `barVal`, to invert the map `k` should be the value associated with `v` key of `invMap`. Again we have a for-loop at **line 18**. This loop prints a key and the value associated with each key in every iteration until all key-value pairs are not printed.

Because keys must be unique, the code goes wrong when the original value items are not unique. In that case, no error occurs, but the inverted map will not contain all pairs from the original map. One solution is to carefully test for

uniqueness and using a multi-valued map. In this case, use a map of type **map[int][]string**.

Look at the following code, and see how uniqueness is tested.

```
package main
import (
"fmt"
)

var (
barVal = map[string]int{"alpha": 34, "bravo": 56, "charlie": 23, "delta": 87,
"echo": 56, "foxtrot": 12, "golf": 34, "hotel": 16, "indio": 87, "juliet": 65, "kilo": 43,
"lima": 98}
)

func main() {
  invMap := make(map[int][]string, len(barVal)) // interchanging types of keys and values
  for k,v := range barVal{
     var a [1]string // making an array to hold values for a key in invMap
     a[0]=k // placing key into array before passing directly into invMap
    _, isPresent := invMap[v] // checking existence
    if isPresent{ // if already present
      invMap[v] = append(invMap[v],",") // add comma before adding another value
    }
    invMap[v] = append(invMap[v],a[0])   // key becomes value and value becomes key
  }
  fmt.Println("inverted:")
  for k, v := range invMap {
    fmt.Printf("key: %v, value: %v / ", k, v)
  }
}
```

Now, you're familiar with sorting and inverting a map, there is a quiz in the next lesson for you to solve.