

Arrays

01

About

02

Creation & Access

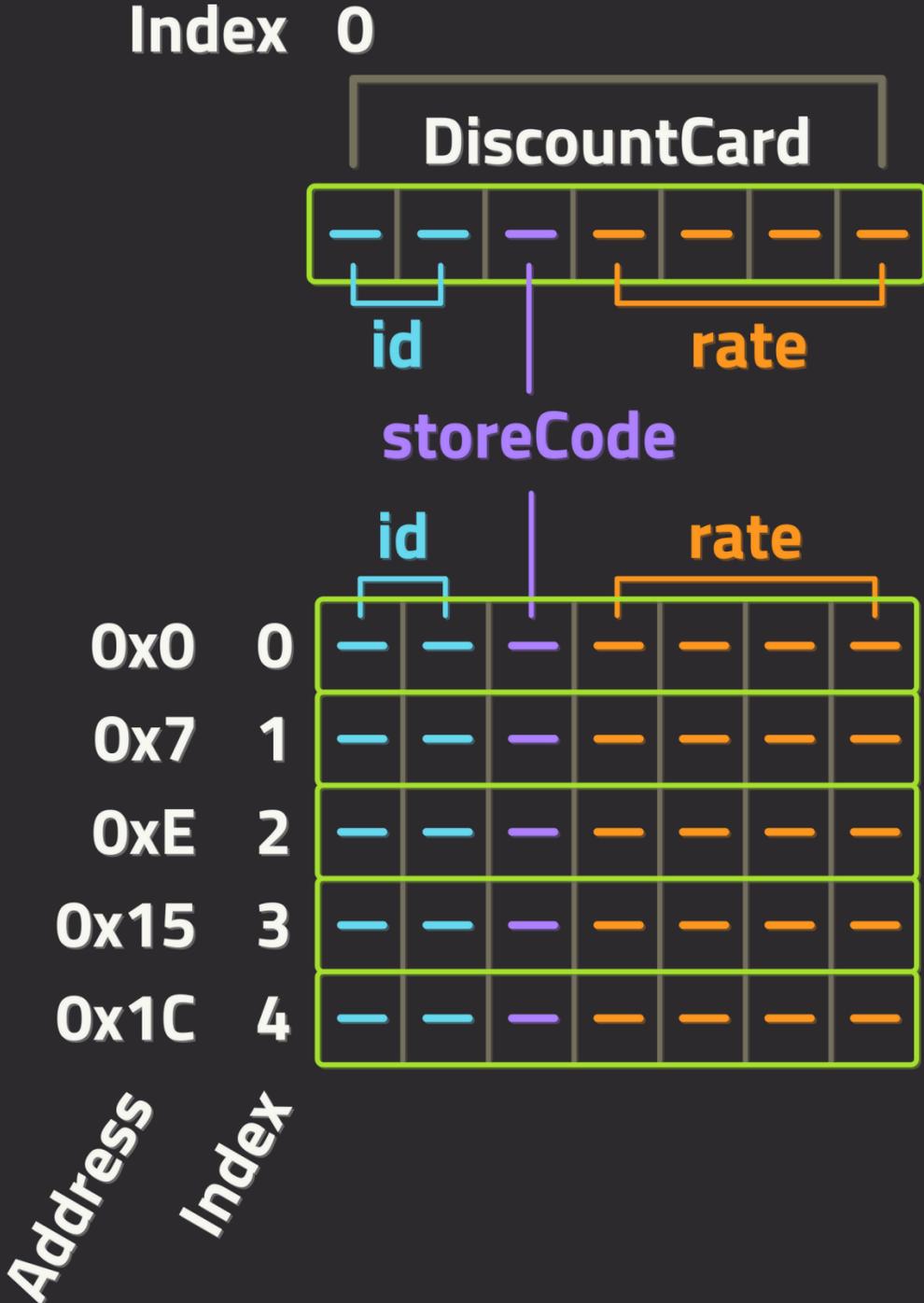
03

Iteration

Arrays

- | Arrays are a way to store multiple pieces of the same kind of data
 - | Data is stored consecutively in an "array" of data
 - | Each piece of data is called an **element**
- | To access items in the array, an **array index** is used
 - | The index starts at 0, meaning 0 **items** from the array start
- | Arrays are **fixed-size** and cannot be resized

Visualization



```
type DiscountCard struct {  
    id uint16  
    storeCode byte  
    rate float32  
}
```

Creating an Array

```
var myArray [3]int
```

```
myArray := [3]int{7, 8, 9}
```

```
myArray := [...]int{7, 8, 9}
```

```
myArray := [4]int{7, 8, 9}
```

- | Elements not addressed in array initialization will be set to default values

Accessing Array Elements

```
var myArray [3]int
```

```
myArray[0] = 7
```

```
myArray[1] = 8
```

```
myArray[2] = 9
```

```
item1 := myArray[0]
```

Iteration

- | Good practice to assign the element to a variable during iteration
- | Easier to read in large functions / nested loops

```
myArray := [...]int{7, 8, 9}
```

```
for i := 0; i < len(myArray); i++ {  
    item := myArray[i]  
    fmt.Println(item)  
}
```

Bounds

- | Attempting to access an element outside the bounds of an array will result in an error

Run Time Error

```
var myArray [3]int

for i := 0; i < 10; i++ {
    fmt.Println(myArray[i])
}
```

Compile Time Error

```
var myArray [3]int

myArray[0] = 7
myArray[1] = 8
myArray[2] = 9
myArray[3] = 10
```

Compile Time Error

```
item4 := myArray[4]
```

Recap

- | Arrays are **fixed-size** collections of **same-type** items
- | Arrays are accessed using an **array index**
 - | It is an error to use an index outside the bounds of an array
- | Array elements can be optionally set during array creation
 - | Elements not manually assigned a value will have a default
- | Use the **len()** function to iterate arrays in a **for** loop