MORE EFFECTIVE GO

SVN Update

Curt Clifton
Rose-Hulman Institute of Technology

REFACTORING FOR PARALLELISM

- Tech. Talk
  - 4:20–5:10, today
  - O169
- Grad School Talk
  - 5:15–6:00, today

MAKING THINGS

NEW(T)

- Allocates space for T
- Sets it to its *zero value*
- Returns a pointer to it

http://adamwaid.wordpress.com/

ZERO VALUE

- ints: *0*
- floats: *0.0*
- strings: ""
- maps, slices, channels: *nil*
- structs: each field allocated and initialized to its zero value

Try to arrange that the zeroed object can be used without further initialization

```
type SyncedBuffer struct {
    lock   sync.Mutex
    buffer bytes.Buffer
}
```

Q1,2

WRITE CONSTRUCTOR FUNCTION WHEN YOU MUST

```
func NewFile(fd int, name string) *File {
    if fd < 0 {
        return nil
    }
    f := File{fd, name, nil, 0}
    return &f
}
```

Taking the address of a local variable allocates a fresh copy on heap

Q3

MAKE REVIEW

- Only map, slice, or channel
- Returns an initialized value (not zero value)
- Returns actual value, not a pointer to it

MAKE RETURNS **REFERENCE TYPES**

Just means that the underlying data is aliased

- Assignment copies the reference
- The underlying value isn't copied

```
s := make([]int, 10)
t := s
```

Open GoEffective2/arraylist.go

Q4

MAPS TEST AND REMOVAL

```
type Color struct {
    r, b, g int
}

func main() {
    m := map[string]Color {
        "Red": Color{255,0,0},
        "Green": Color{0,255,0},
        "Blue": Color{0,0,255},
    }
    _, ok := m[“PeachPuff”]
    fmt.Println(ok)
    m["Red"] = Color{}, false
    fmt.Println(m)
}
```

http://www.visit-londoncity.com

Q5

INITIALIZATION

CONSTANTS

```
type ByteSize float64
const (
    _ = iota
    KB ByteSize = 1<<(10*iota)
    MB
    GB
    TB
    PB
    EB
    ZB
    YB
)
```

Increments with each use. Auto-resets

Blank constants get previous *expression*

CONSTANTS

```
type ByteSize float64
const (
    _ = iota
    KB ByteSize = 1<<(10*iota)
    MB ByteSize = 1<<(10*iota)
    GB ByteSize = 1<<(10*iota)
    TB ByteSize = 1<<(10*iota)
    PB ByteSize = 1<<(10*iota)
    EB ByteSize = 1<<(10*iota)
    ZB ByteSize = 1<<(10*iota)
    YB
)
```

Increments with each use. Auto-resets

Blank constants get previous *expression*

Q6

ARGS

PASS BY VALUE, REFERENCE VALUES

```
type FullName struct { first, last string }

// String doesn't mutate, so it's OK to pass value.
func (fn FullName) String() string {
    return fn.first + " " + fn.last
}

// SetLastBad tries to mutate, but it mutates it's own copy!
func (fn FullName) SetLastBad(l string) {
    fn.last = l
}

// SetLast mutates successfully because the receiver is passed as a pointer.
func (fn *FullName) SetLast(l string) {
    fn.last = l
}

func main() {
    n := FullName{"Randall", "Munroe"}
    fmt.Println(n)
    n.SetLastBad("Schwartz")
    fmt.Println(n)
    n.SetLast("Schwartz")
    fmt.Println(n)
}
```

```
Randall Munroe
Randall Munroe
Randall Schwartz
```

METHODS

- Receiver can be any named type
  except a pointer or interface
- Use conversions to select which method set to use…

Q7

```
type Sequence []int

// Methods required by sort.Interface.
func (s Sequence) Len() int {
    return len(s)
}
func (s Sequence) Less(i, j int) bool {
    return s[i] < s[j]
}
func (s Sequence) Swap(i, j int) {
    s[i], s[j] = s[j], s[i]
}

// Method for printing - sorts the elements before printing.
func (s Sequence) String() string {
    sort.Sort(s)
    str := "["
    for i, elem := range s {
        if i > 0 {
            str += " "
        }
        str += fmt.Sprint(elem)
    }
    return str + "]"
}
```

Redundant!

```
type Sequence []int

// Methods required by sort.Interface.
func (s Sequence) Len() int {
    return len(s)
}
func (s Sequence) Less(i, j int) bool {
    return s[i] < s[j]
}
func (s Sequence) Swap(i, j int) {
    s[i], s[j] = s[j], s[i]
}
```

Redundant!

```
// Method for printing - sorts the elements before printing.
func (s Sequence) String() string {
    sort.Sort(s)
    str := "["
    for i, elem := range s {
        if i > 0 {
            str += " "
        }
        str += fmt.Sprint(elem)
    }
    return str + "]"
}
```

return fmt.Sprint([]int(s))

Redundant!

```
type Sequence []int

// Methods required by sort.Interface.
func (s Sequence) Len() int {
    return len(s)
}
func (s Sequence) Less(i, j int) bool {
    return s[i] < s[j]
}
func (s Sequence) Swap(i, j int) {
    s[i], s[j] = s[j], s[i]
}
```

Redundant!

```
// Method for printing - sorts the elements before printing.
func (s Sequence) String() string {
    sort.Sort(s)
    str := "["
    for i, elem := range s {
        if i > 0 {
            str += " "
        }
        str += fmt.Sprint(elem)
    }
    return str + "]"
}
```

sort.IntArray(s).Sort()

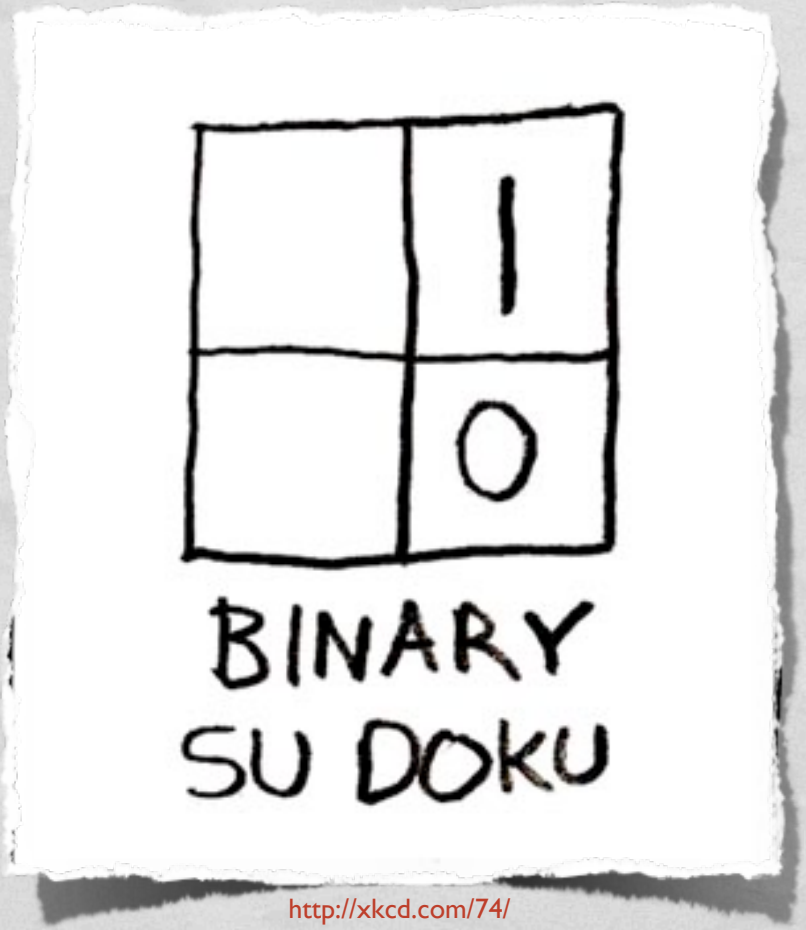
return fmt.Sprint([]int(s))

Redundant!

Q8

HW14: SU DOKU

- Due Thursday
- Pair program
- Can purchase partial solutions
- Work time in class tomorrow



http://xkcd.com/74/