READY TO GO

Curt Clifton
Rose-Hulman Institute of Technology

WHY GO?

- Systems language
- OO, but no inheritance
- CSP-style concurrency

http://en.wikipedia.org/wiki/File:CAR_Hoare.jpg

HELLO, GO

Start in main.main()

```
package main

import myFmt "fmt"

func main() {
    myFmt.Printf("Hello, world. ☺\n")
}
```

Use *fmt* package, but call it *myFmt*

UTF-8 encoding is standard

Photo by drtchock - http://flic.kr/p/33FYcG

```
x := ((2 + 3)
      * 6)
```

hello.go:6: syntax error: unexpected semicolon …, expecting)
hello.go:7: syntax error: unexpected), expecting semicolon or …

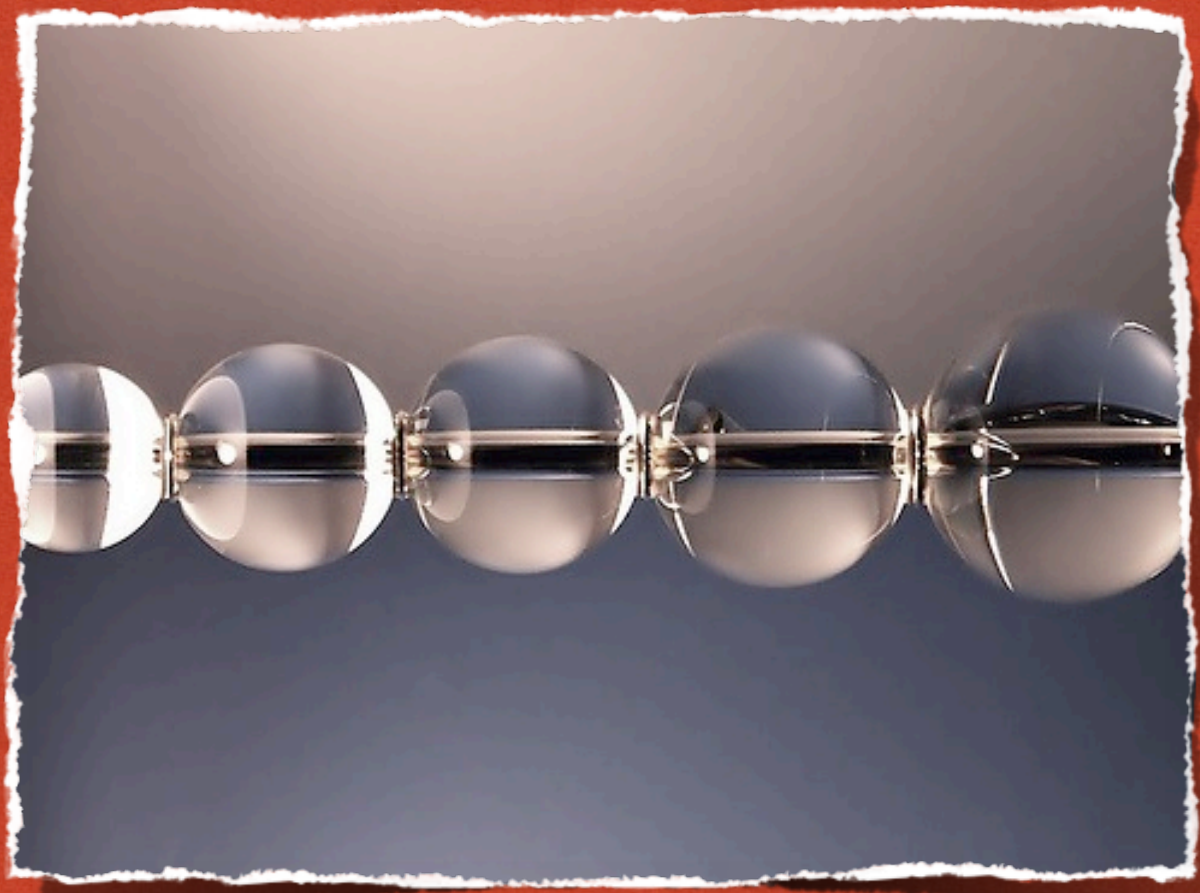


Photo by wdj(0) - http://flic.kr/p/Lcoi2

ECHO AND STRINGS

ARRAYS AND SLICES

- `var arrayOfInt [10]int`
- `var sliceOfInt []int`
- `sliceOfInt = arrayOfInt[2:5]`

Photo by libraryman - http://flic.kr/p/7VuSJ

Photo by reurinkjan - http://flic.kr/p/6PmeDF

HEAP ALLOCATION

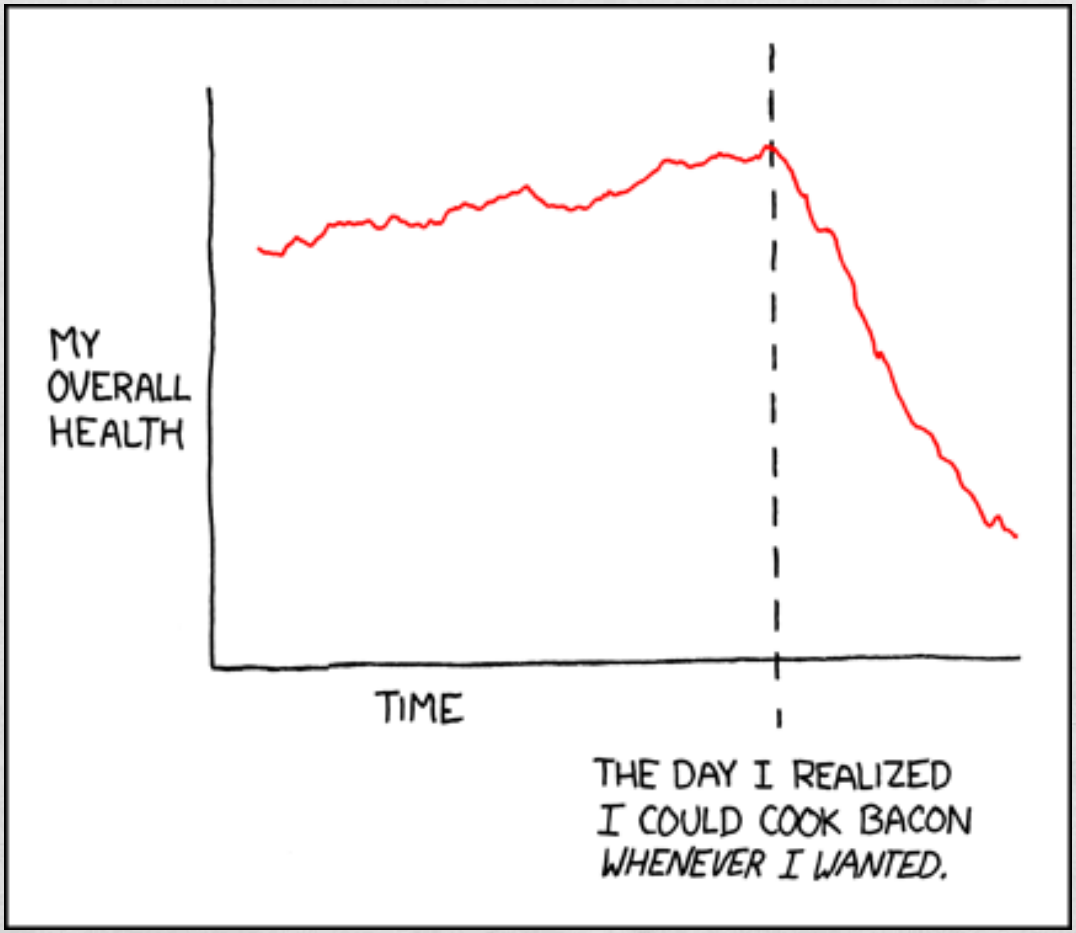
make FOR REFERENCE TYPES (SLICES, MAPS, AND CHANNELS)
new FOR VALUE TYPES (EVERYTHING ELSE)

Q2

CONSTANTS: ARBITRARY PRECISION

```
const hardEight = (1 << 100) >> 97
var a uint64 = 0
a := uint64(0)
i := 0x1234
var j int = 1e6
x := 1.5
i3div2 := 3/2
f3div2 := 3./2.
var k int = 3./2   // error!
```

STOVE OWNERSHIP



http://xkcd.com/418/

Although maybe it's just a phase, like freshman year of college when I realized I could just buy frosting in a can.

I/O

```
package file

import (
    "os"
    "syscall"
)

type File struct {
    fd      int         // file descriptor number
    name    string      // file name at Open time
}
```

Initial cap makes File public, otherwise package protected

```
package file

import (
    "os"
    "syscall"
)

type File struct {
    fd      int          // file descriptor number
    name    string       // file name at Open time
}

func newFile(fd int, name string) *File {
    if fd < 0 {
        return nil
    }
    return &File{fd, name}
}
```

Equivalently:

```
n := new(File)
n.fd = fd
n.name = name
return n
```

```
type File struct {
    fd      int         // file descriptor number
    name    string      // file name at Open time
}

func newFile(fd int, name string)  *File {
    if fd < 0 {
        return nil
    }
    return &File{fd, name}
}

var (
    Stdin = newFile(0, "/dev/stdin")
    Stdout = newFile(1, "/dev/stdout")
    Stderr = newFile(2, "/dev/stderr")
)
```

```
func newFile(fd int, name string) *File {
    if fd < 0 {
        return nil
    }
    return &File{fd, name}
}

var (
    Stdin = newFile(0, "/dev/stdin")
    Stdout = newFile(1, "/dev/stdout")
    Stderr = newFile(2, "/dev/stderr")
)

func Open(name string, mode int, perm uint32) (file *File, err os.Error) {
    r, e := syscall.Open(name, mode, perm)
    if e != 0 {
        err = os.Errno(e)
    }
    return newFile(r, name), err
}
```

os library standardizes error handling

```
    if e != 0 {
        err = os.Errno(e)
    }
    return newFile(r, name), err
}

func (file *File) Close() os.Error {
    if file == nil {
        return os.EINVAL
    }
    e := syscall.Close(file.fd)
    file.fd = -1  // keeps us from closing the real fd again
    if e != 0 {
        return os.Errno(e)
    }
    return nil
}

func (file *File) Read(b []byte) (ret int, err os.Error) {
    ...
}

func (file *File) Write(b []byte) (ret int, err os.Error) {
    ...
}
```

```
type File struct {
    fd      int
    name    string
}
```

```
func (file *File) Close() os.Error {
    if file == nil {
        return os.EINVAL
    }
    e := syscall.Close(file.fd)
    file.fd = -1    // keeps us from closing the real fd again
    if e != 0 {
        return os.Errno(e)
    }
    return nil
}

func (file *File) Read(b []byte) (ret int, err os.Error) {
    ...
}

func (file *File) Write(b []byte) (ret int, err os.Error) {
    ...
}

func (file *File) String() string {
    return file.name
}
```

```
package file

import (
    "os"
    "syscall"
)

type File struct { ... }

func newFile(fd int,

var (
    Stdin = newFile(0
    Stdout = newFile(
    Stderr = newFile(
)

func Open(name string, mode int, perm uint32) (file *File, err os.Error) { ... }

func (file *File) Close() os.Error { ... }
func (file *File) Read(b []byte) (ret int, err os.Error) { ... }
func (file *File) Write(b []byte) (ret int, err os.Error) { ... }
func (file *File) String() string { ... }
```

```
package main

import (
    "./file"
    "fmt"
    "os"
)

func main() {
    hello := []byte("hello, world\n")
    file.Stdout.Write(hello)
    file, err := file.Open("/does/not/exist", 0, 0)
    if file == nil {
        fmt.Printf("can't open file; err=%s\n", err.String())
        os.Exit(1)
    }
}
```