Google Go!
Seminar aus Informatik

Martin Aigner 0621270
Alexander Baumgartner 0620345

Department of Computer Science
University of Salzburg

May 28, 2010
1 Basic Types
2 Arrays and Strings
3 Slices
4 Maps
5 New and Make
Basic Types

i := 1234

\[ \begin{array}{c|c}
1234 & \text{int} \\
\end{array} \]

j := int32(1)

\[ \begin{array}{c|c}
1 & \text{int32} \\
\end{array} \]

f := float32(3.14)

\[ \begin{array}{c|c}
3.14 & \text{float32} \\
\end{array} \]

b := [3]byte{"a","b","c"}

\[ \begin{array}{c|c|c|c|c}
\text{a} & \text{b} & \text{c} & \text{[3]byte} \\
\end{array} \]

primes := [3]int{2,3,5}

\[ \begin{array}{c|c|c|c|c|c|c|c|c}
2 & 3 & 5 & \text{[3]int32} \\
\end{array} \]

**Figure:** Memory Layout of basic types
type Point struct { X, Y int }

p := Point{10, 20}

pp := &Point{10, 20}

Figure: Memory Layout of structs
Struct Type

type Rect1 struct { Min, Max Point }
type Rect2 struct { Min, Max *Point }

r1 := Rect1{Point{10, 20}, Point{50, 60}}

r2 := Rect2{&Point{10, 20}, &Point{50, 60}}

Figure: Memory Layout of composite structs
s := "hello"

![Memory Layout of a string]

Figure: Memory Layout of a string
Slices

x := []int{2,3,5,7,11}

y := x[1:3]

Figure: Slicing an array of integers
Maps are...

... built-in data structures to associate values of different types. Keys can be any type for which the equality operator is defined.

- integers
- floats
- strings
- pointer
- interfaces (if the dynamic type supports equality)
Example

```go
//composite literal construction
var timeZone = map[string]int {
    "UTC": 0*60*60,
    "EST": -5*60*60,
    // and so on
}

//accessing map values
offset := timeZone["EST"]

//checking 0 v.s. non-existanve
var seconds int
var ok bool
seconds, ok = timeZone[tz] //comma ok idiom
```
New and Make

New
- `new(T)` returns a *T, a pointer to zeroed storage
- ready to use
- works transitively

Make
- `make(T, args)` returns a value of type T, not a pointer
- used for slices, maps and channels only
- initialized complex datastructure
Examples for New

Figure: Allocation with new
Examples for Make

```
new([]int)

*[][]int

nil 0 0 [][]int
ptr len cap

make([][]int, 0)

[][]int

[]0][]int

make([][]int, 2, 5)

[]2][]int

0 0 0 0 0 0 [][]int
ptr len cap

Figure: Allocation with make
```
Concurrency

6 Share by communicating

7 Goroutines

8 Channels

9 Parallelization

10 Example
Share by communicating

Slogan
Do not communicate by sharing memory; instead, share memory by communicating

- Shared values are passed around on channels
- Only one goroutine has access to the value at any given time
- Using channels to control access makes it easier to write clear, correct programs
- It can also be seen as a type-safe generalization of Unix pipes

For reference counts there is no need to put a mutex around the integer variable
Goroutines are...

...functions executing in parallel with other goroutines in the same address space

- Prefix a function or method call with the `go` keyword to run the call in a new goroutine
- Hides many of the complexities of thread creation and management
- Goroutines are multiplexed onto multiple OS threads
- When the call completes, the goroutine exits, silently

```go
func main() {
    go expensiveComputation(x, y, z)
    anotherExpensiveComputation(a, b, c)
}
```
Channels combine...

...communication with synchronization

- Shared values are passed around on channels
- Like maps, channels are a reference type and are allocated with make
- Channels can be buffered
- With a channel you can make one goroutine wait for another
  - Receivers always block until there is data to receive
  - If the channel is unbuffered, the sender blocks until the receiver has received the value
  - If the channel has a buffer, the sender blocks if the buffer is full

```go
ci := make(chan int) // unbuffered channel of integers
cs := make(chan *os.File, 100) // buffered channel of pointers to Files
```
Channels combine

...communication with synchronization

c := make(chan int) // Allocate a channel.
// Start the sort in a goroutine; when it completes, signal on the channel.
go func() {
    list.Sort()
    c <- 1 // Send a signal; value does not matter.
}()
doSomethingForAWhile()
<- c // Wait for sort to finish; discard sent value.
Parallelization

If the calculation can be broken into separate pieces,...
...it can be parallelized, with a channel to signal when each piece completes.

- Current compiler implementations will not parallelize code by default
- Environment variable GOMAXPROCS sets the number of cores to use
- Or call runtime.GOMAXPROCS(NCPU) from your code

A good example for parallelization is a request-broker. We handle a defined number of requests in parallel and block incoming requests if the maximum number is reached.
Semaphore using a channel I (Code)

```go
var sem = make(chan int, MaxOutstanding)
func handle(r *Request) {
    sem <- 1; // Wait for active queue to drain.
    process(r); // May take a long time.
    <-sem; // Done; enable next request to run.
}
func Serve(queue chan *Request) {
    for {
        req := <-queue;
        go handle(req); // Don’t wait for handle to finish.
    }
}
```
Semaphore using a channel II (Figure)

```
Serve
for { 
    req := <-queue 
go handle(req); 
}

Thread 1
sem <- 1
critical section <- sem;

Thread 2
sem <- 1
critical section <- sem;

Thread 3
sem <- 1
critical section <- sem;

Thread 4
sem <- 1
critical section <- sem;
```

```
make (chan int, 3)
```

1
2
3
4

Blocked waiting for slot
The End

Thank You!
References