PSC 2022/23 (375AA, 9CFU)
Principles for Software Composition
Roberto Bruni
http://www.di.unipi.it/~bruni/

http://didawiki.di.unipi.it/doku.php/magistraleinformatica/psc/start

23 - Google Go
Google Go
concurrency oriented programming
Google Go

http://golang.org/
Go features

facilitate building reliable and efficient software

open source

compiled, garbage collected

functional and OO features

statically typed (light type system)

concurrent
Go principles

C, C++, Java:
too much typing (writing verbose code)
and too much typing (writing explicit types)
(and poor concurrency)

Python, JS:
no strict typing, no compiler issues
runtime errors that should be caught statically

Google Go:
compiled, static types, type inference
(and nice concurrency primitives)
Go project

designed by Ken Thompson, Rob Pike, Robert Griesemer

2007: started experimentation at Google
Nov 2009: first release (more than 250 contributors)
May 2012: version 1.0 (two yearly releases since 2013)
Feb 2023: version 1.20

C. Doxsey, Introducing Go (2016). Ch: 1-4, 6-7, 10
Go concurrency

any function can be executed in a separate lightweight thread

\[ \text{go } f(x) \]

goroutines run in the same address space
package \texttt{sync} provides basic synchronisation primitives
programmers are encouraged NOT TO USE THEM!

\textit{do not communicate by sharing memory}
\textit{instead, share memory by communicating}

use built-in high-level concurrency primitives:
channels and message passing
(inspired by process algebras)
Go channels

channels can be created and passed around

```go
var ch = make(chan int)
```

creates a channel for transmitting integers

```go
ch1 = ch
```

ch1 and ch refers to the same channel

```go
f(ch)
g(ch)
```

f and g share the channel ch
Directionality

channels are always created bidirectional

```go
var ch = make(chan int)
```

channel types can be annotated with directionality

```go
var rec <-chan int
```

`rec` can only be used to receive integers

```go
var snd chan<- int
```

`snd` can only be used to send integers

```go
rec = ch
snd = ch
```

are valid assignments

```go
rec = snd // invalid!
```
Go communication

- To send a value (like \(ch!2\)) \(ch \leftarrow 2\)
- To receive and store in \(x\) (like \(ch?x\)) \(x = \leftarrow ch\)
- To receive and throw the value away \(\leftarrow ch\)
- To close a channel (by the sender) \(\text{close}(ch)\)
- To check if a channel has been closed (by the receiver) \(x, ok = \leftarrow ch\) // either value, true or 0, false
Go sync communication
by default the communication is synchronous
BOTH send and receive are BLOCKING!
asynchronous channels can be created
by allocating a buffer of fixed size

```go
var ch = make(chan int, 100)
```
creates an asynchronous channel of size 100
receive on asynchronous channel is of course still blocking
send is blocking only if the buffer is full

no dedicated type for asynchronous channels:
buffering is a property of values not of types
Go communication
to choose between different options

```go
select {
  case x = <- ch1: { ... }
  case ch2 <- v: { ... }
  // both send and receive actions
  default: { ... }
}
```

the selection is made pseudo-randomly among enabled cases
if no case is enabled, the default option is applied
if no case is enabled, and no default option is given
the select blocks until (at least) one case is enabled
non-blocking receive

```plaintext
select {
    case x = <- ch: { ... }
    default: { ... }
}
```

receives on `x` from `ch`, if data available
otherwise proceeds
Name mobility

channels can be sent over channels (like in $\pi$-calculus)

```
var mob = make(chan chan chan int)
```

a channel for communicating channels

```
mob <- ch
```

send the channel ch over mob
// You can edit this code!
// Click here and start typing.
package main
import "fmt"
func main() {
    fmt.Println("Hello, 世界")
}

Hello, 世界

Program exited.
Name mobility: secrecy

\[ \text{Diagram showing nodes labeled } m, A, \text{ and } B \text{ connected by edges labeled } as \text{ and } bs. \]
Name mobility: secrecy

Diagram:

- Nodes: A, S, B
- Edges: m, ab, bs, as
- Secret agent represented by a silhouette

Legend:
- m: message
- ab: authentication
- bs: beacon signal
- as: access signal
Concurrent prime sieve

- Rec
- Filt
- Gen

- 3
- 5
- 7
- 11, ...

- 2
- 3, 5, 7, 9, ...

- Filt(2)
- Filt(3)

- ch
- ch1
- ch2

- Rec
- Gen

- ch
- ch1
- ch2

- 5, 7, 9, ...

- 3, 5, 7, 9, ...

- 2, 3, 5, 7, 9, ...