Carl Kignsford, 02-201, Fall 2015

Loose Ends

Functions as values of variables

Sometimes it would be useful to pass a function as a parameter to another function. For example, suppose you were writing a protein folding program, but wanted to try out lots of diffrent energy functions:

```go
// crossings cost 10 units
func energy1(fold Fold) float64 { return ... }

// crossing costs infinite units
func energy2(fold Fold) float64 { return ... }

// find the lowest energy fold for protein `seq` using the given
// energyFunction
func findBestFold(seq string, energyFunction ... ) Fold
```

How could we write `findBestFold()` above? It turns out that this is easy in Go, since you can store functions in variables just like any other data:

```go
func findBestFold(seq string, energyFunction func(Fold)float64)) Fold {

    // you can call `energyFunction` like any other function
    energyFunction(myCurrentFold)
    //...
}

//...

findBestFold("HHHPPPPHPPHPHPH", energy1)
```

Function literals

You can even define a function on the fly when you need it:

```
var C int = 123
findBestFold("HPPPHHHPP", func(fold Fold)float64 {
        return C + 12*fold.numCrossings + fold.numBuriedH
    }
)
```

Notice that the function we are defining inline can access variables around it like `C`.

Notice also that this function we created doesn't have (and can't have) a name: it's an anonymous function.

Parallel Programs

Modern computers xhave mutiple "CPUs" that can each execute instructions at the same time (or "in parallel"). Each of these "CPUs" is often called a core. Even your iPhone has multiple cores (2 or 3).

The fastest computers today are fast not because each CPU is fast, but because they have a very large number of them. (The fastest computer currently has ~ 3 million cores).

Using parallelism is essential to quickly process large scientific data sets (partical physics measurments, astronomical observations, genomics, etc.). For example, the gene expression quantifiers developed by my group all make heavy use of multiple cores.

Go routines

Go has great support for writing programs that use several cores. The main way it does this is via the `go` keyword. If you have a statement that consists of a single function call `f(...)`, you can put the word `go` in front of it, and it will run that function in parallel with the rest of your program.

For example:

```go
package main

import (
    "fmt"
    "time"
    "math/rand"
)

func f(n int) {
    for i := 0; i < 10; i++ {
        fmt.Println("function", n, ":", i)
        amt := time.Duration(rand.Intn(250))
        time.Sleep(time.Millisecond * amt)
    }
}

func main() {
    for i := 0; i < 10; i++ {
        go f(i)
    }
    var input string
    fmt.Scanln(&input)
}
```

> What do you think happens if we remove the `fmt.Scanln(&input)` statement?

Each one of these things running in parallel is called a goroutine.

Communication between Goroutines

Sometimes, the different parallel parts of your program need to talk to each other. Go provides the ability to do this via `channels`.

Channels are a builtin type in Go. They are queues that can hold values of a particular type. You have to `make` a channel just like lists and maps:

```go
var I chan int = make(chan int) // a channel of ints
J := make(chan float64) // a channel of float64s
```

You can put `x` into a channel using the `c <- x` operator.

You can remove the next thing from the channel using the `<- c` operator.

```go
package main
import (
    "fmt"
    "time"
    "math/rand"
    "runtime"
)

func pinger(c chan string) {
    for  {
        c <- "ping"
    }
}

func ponger(c chan string) {
    for  {
        c <- "PONG"
        time.Sleep(time.Duration(rand.Intn(10)) * time.Second)
    }
}

func printer(c chan string) {
    for {
        fmt.Println(<- c)
        time.Sleep(time.Second * 1)
    }
}


func main() {
    runtime.GOMAXPROCS(10)

    var c chan string = make(chan string)
    go pinger(c)
    go ponger(c)
    go printer(c)
    var input string
    fmt.Scanln(&input)
}
```

> **Discussion questions.** Where could you use parallelism to speed up your homework assignments?
>
> What happens if more than 1 goroutine modifies a global variable or a `map` ?

Glossary

- goroutines: a function executing in parallel with the rest of your program, created with the `go` keyword.
- parallel programming: writing programs that simultaneously use more than 1 core.
- core: a unit of the CPU that can independently execute instructions.
- channel: a builtin data structure in Go that lets you communicate between goroutines. It acts like a queue.