‘Go’ Forth and Code

Jonathan Gertig

CSC 415: Programming Languages

Dr. Lyle

2013
‘Go’ dogs ‘Go’ or A Brief History of Go

6 years ago Robert Griesemer, Rob Pike and Ken Thompson conceptualized the language of “Go.” They worked on the language part time only until January of 2008 when Ken Thompson worked on a production level compiler that generated C code. In 2009 Go became open source. This very November marks the 4th anniversary of the Go code becoming open source.

Go is a general-purpose language designed with systems programming in mind. It is strongly typed and garbage-collected and has explicit support for concurrent programming. Programs are constructed from packages, whose properties allow efficient management of dependencies. The existing implementations use a traditional compile/link model to generate executable binaries.

Go has a very strong community and has done very well compared to other languages in their infancy. Some of the increase is directly correlated to the increase in the computer science field itself none then less, Go as a language has done very well. In their recent Blog post on the anniversary of Go they showed the increase in golang as a google search term in the last four years.
Digging with a ‘Go’pher or A View on Go’s Scope and Names

Go was built with strong global variables but like many C based languages you can change the scope of the variable you are working with. Go uses a hierarchical scoping mechanic that allows for many scopes from global to local and sub-local. You can force change the scope of a variable by using the “:=” operand forcing a value declaration of the variable, instead of “=” which would just update the higher scope level’s as an assignment statement.

Go is strongly tied to its parent language of C and continues to use goto statements with in its code. Go unlike languages like PL/I keeps goto labels as local scope only and cannot be passed to a function as a parameter. This keeps some of the spaghetti like tendencies of goto statements at bay.

Go does allow for named variables these it calls identifiers. Identifiers must begin with a Unicode uppercase character in their name. Like languages like java Go has a construct like classes that are called packages identifiers can be exported out of a package as long as they are in the package block or are field names or method names for the package.

Go has a few predeclared identifiers. These include all the type names, the constants true, false, and iota as well as a select few system level functions those being append, cap, close, complex, copy, delete, imag, len, make, new, panic, print, println, real and recover. My favorite being panic! I will talk more on panic later in error handling.
Go’s constants are what they seem, immutable values stored to identifiers. The universal constants are true, false, and iota. Iota is very special universal identifier to GO. It only works in constant declarations. Iota is also dependent on its scope so in a list of constants it will increment in value as it goes(Miek). An example of this is:

```go
const a = iota

const (  
    John = iota
    Luke = iota
    Mark = iota
    People’Going = iota
)
```

In the example above ‘a’ will have a value of 0 then ‘John’ will be 0 but ‘Luke’ will return 1 and so forth till ‘PeopleGoing’ will be 3.

**Data Types**

Go has many data types and a strong propensity for numbers. Go has Boolean values true and false as well as strings. Strings in Go are UTF-8 encoded byte arrays of data which allows for some fast manipulation of string data. Go unlike most languages uses immutable strings meaning once declared they are unable to be changed (Miek). Go has many numeric data types from
8-bit up to 64-bit as unsigned and signed. This allows for some low level control of how much data your program is actually using. This gives you a range of 0 to 18446744073709551615 for unsigned integers and -9223372036854775808 to 9223372036854775807 for signed integers.

Go uses the IEEE-754 standard for its floating point numbers offering a 32-bit and a 64-bit version. Go also has a complex number’s class that are 64-bit and 128-bit. They use the 32-bit and 64-bit floating point numbers to represent the real and imaginary parts of the complex number.

Go aliases a few of its numeric types for those that are not that concerned with how accurately they use space and for better understanding of parts. Go has a byte type which is directly connected to uint8 and is used as the base type in many other types in Go. The rune is an alias of int32 and can be used to find the UTF-8 value of a character. Just using unit as the type will dynamically choose between uint32 or uint64. Which is also true for int as it is dynamically chosen between int32 or int64.

Go like C and C++ does allow for direct address access of variables with pointers. Arrays are not often seen in Go programs because the size of an array is part of its type, which limits its expressive power. Arrays have their place—they are a good representation of a transformation matrix for instance—but their most common purpose in Go is to hold storage for a slice (Pike).

Slices are native to Go. Slices are not arrays themselves but are like pointers to specific ranges with in an array. For example:
var slice = buffer[100:150]

This will give you pointer access to the data in the 100 to 150\textsuperscript{th} exclusive location of the buffer. You can re-slice so in affect truncate by:

    slice = slice[1:len(slice)-1]

Go has a type called a channel. A channel is a type used for communication between threads. In Go, the \texttt{chan} keyword is used to define a channel. The \texttt{make} keyword is used to create it, along with the type of data that the channel can hold. \texttt{ie:}

    my_channel := make(chan int)

You can then use arrow notation to direct the flow data.

    my_channel <- 5
    //within some other goroutine - to take a value off the channel
    var my_recv_value int
    my_recv_value = <- my_channel

Channels can only store one value at a time which is very important it makes routines synchronous, because it requires the once a value is in the channel it has to be removed before more data can be put in it(VJ).

The last data type is the Error and it is self-explanatory. In languages like Cobal and Ada there are types for the Error. You can store a string used to describe in some way the error.

\texttt{'Go' For it or A View of Control Structures}

We touched on Label types before but a Labeled statement is the control structure for it. Labeled statements match with break and goto statements. Labels end in a semicolon and have a colon separator. Expression statements are the base for almost all other methods or statements in the Go language. Expression statements are how
you call a function such as `len("param")` that is an expression statement. Go uses send statements to push a value into a channel with arrow notation ie: `ch <- b`. Go like most modern languages has `IncDec` statements which are the unary `++` and – statements.

Go has very interesting grammar for its if statements it look like this:

```plaintext
IfStmt = "if" [ SimpleStmt ";" ] Expression Block [ "else" ( IfStmt | Block ) ].
```

That `SimpleStmt` segment allows for an expression that is run before the if statement is checked. This is a Go specific statement, I look but could not find the reason the developers decided to implement this. Go's switch statement also includes the

`SimpleStmt` in its grammar:

```plaintext
ExprSwitchStmt = "switch" [ SimpleStmt ";" ] [ Expression ] "{" { ExprCaseClause } "}".
ExprCaseClause = ExprSwitchCase ";" StatementList .
ExprSwitchCase = "case" ExpressionList | "default".
```

This allows you to run something like:

```go
switch x := f(); { // missing switch expression means "true"
    case x < 0: return -x
    default: return x
}
```

You can use a falltrough statement in a switch which moves on to the next case.

```
Select statements in Go are similar to which statements but is used for channels.
```

For all the cases in the "select" statement, the channel expressions are evaluated in top-to-bottom order, along with any expressions that appear on the right hand side of
send statements. A channel may be nil, which is equivalent to that case not being present in the select statement except, if a send, its expression is still evaluated. If any of the resulting operations can proceed, one of those is chosen and the corresponding communication and statements are evaluated. Otherwise, if there is a default case, that executes; if there is no default case, the statement blocks until one of the communications can complete. There can be at most one default case and it may appear anywhere in the "select" statement. If there are no cases with non-nil channels, the statement blocks forever. Even if the statement blocks, the channel and send expressions are evaluated only once, upon entering the select statement (golang).

Unfortunately Go has no set way to run a sub program except for calling a function locally or globally. Go has a really nice statement called Defer. Defer calls a function and waits for the surrounding scope to finish or for the program to panic.

**Shall we ‘Go’ or A Review of Abstract Types and Methods**

Go’s base abstract type like C++ is a Struct. Structs allow for any number of uniquely named expressions or functions. Go allows for functions but unlike most languages it is really nice and allows you to run a function without giving an input that it requires. If you do it will default to a nil value. Go includes a map type for key value store of a single value type.

‘Go’bjects or Objects in Go
Although Go has types and methods and allows an object-oriented style of programming, there is no type hierarchy. The concept of “interface” in Go provides a different approach that we believe is easy to use and in some ways more general. There are also ways to embed types in other types to provide something analogous—but not identical—to subclassing. Moreover, methods in Go are more general than in C++ or Java: they can be defined for any sort of data, even built-in types such as plain, “unboxed” integers. They are not restricted to structs (classes). The lack of type hierarchy makes “objects” in Go feel much more lightweight than in languages such as C++ or Java and therefore feels like they are not really objects (Golang).

**How to ‘Go’ or Go concurrency**

“Do not communicate by sharing memory; instead, share memory by communicating.”(Golang)

Go has explicit support for concurrent programming. The main building blocks for Go concurrency are goroutines and channels. You run a goroutine by prefixing a function with ‘go’.

```go
go list.Sort()
```

They're called *goroutines* because the existing terms—threads, coroutines, processes, and so on—convey inaccurate connotations. A goroutine has a simple model: it is a function executing concurrently with other goroutines in the same address space. It is lightweight, costing little more than the allocation of stack space. And the
stacks start small, so they are cheap, and grow by allocating (and freeing) heap storage as required (Effective Go). Goroutines use channels to pass around shared values. Only one goroutine can have access to a value by default. Sets up for a synchronized routine setup. A common misconception is that goroutines run in parallel but they do not they run concretely not in parallel.

‘Go’ Time or Exception Handling and Event Handling in Go

Go has implicate support for errors and error handling because you can return a data object of type Error. IE:

```go
define Read(f *File, b []byte) (n int, err error)
```

This function will return the int n and an error err after it reads the file and fills the byte array. You can also deal with a system error by panicking! When you panic any functions that are on the stack will run first then the function in the panic expression then the program will crash(Golang). The only implicate event handling that Go has is for http events. There are packages that have been written to add more event handling into Go.

My ‘Go’

Go is not the most readable language because it has a lot of similarities with C. But it would not be hard to learn to write the language as it is a simplified version of C.
The cost to implement GO in a company environment I cannot claim to know but the language itself is free to use because it has been released as an open source solution. The only real cost is the time to learn Go. Go is not hard to learn for anyone who has worked with C or C++.
Bibliography

http://golang.org/
All the things!

Go book

http://golang.org/doc/effective_go.html
How to write effective Go