Go

Tanner Wysocki

Mr. Woodring

CIS 343-02

04/21/18

Introduction

Go was born out of frustration with the lack of modern systems programming languages. Around the mid-2000's, the languages traditionally associated with systems languages like C and C++ were still being used in this domain despite becoming long in the tooth. Much had changed in the computing landscape since C/C++ was introduced, and there were many trends that the traditional language had not been able to keep up with. Specifically, header files made compilation slow and cumbersome for large project, dynamically typed languages like Python and JavaScript were on a meteoric rise, garbage collection tended to be absent at the system level, parallel computation was not well supported, and multicore systems were not well supported.

According to the official FAQ, Go was created in 2007 to address all of these issues. Created by Rob Pike (known for projects like Plan 9 and the acme text editor), Robert Griesemer, and Ken Thompson (known for creating the Unix operating system). They began on September 21, 2007 sketching out idea for the language over the course of a few days and went to work implementing these ideas over the course of the next year, and on November 10, 2009, the language became open source under the BSD license to allow for contribution from community members.

The design of the language was influenced by C in that it is a primarily imperative procedural language and it shares many similarities in syntax. It also was influenced by languages like Pascal for packages, and Newsqueak and Limbo for concurrency. According to the FAQ, languages were divided into three strengths: efficient compilation, efficient execution, and ease of use. Systems languages typically were

good for the first two but were harder to write for, and languages like Python were easy to use, but are unusable for lower level programming because they're slow to compile or slow at execution or both. They addressed address the ease of use issue through features that are found in dynamic languages with the safety of statically typed languages, and most importantly, they aimed for it to be fast.

Today we see Go used by several companies with it being ranked 19 on the TIOBE Index. Google is the most notable one as they are the developer of the language. Internally, they use Go for many projects such as their download server. Besides Google, there are a large number of companies out there today that use Go. Notable examples include Twitch, Twitter, StackExchange, GitHub, Walmart, and countless other companies. A big reason for the acceptance of Go among these companies is its ability to handle concurrent tasks well through Goroutines. Go also excels in areas such as big data or server management, making it tailor-made for the web.

Data Abstractions

Being a C based language, Go contains many of the basic primitives that most other languages contain (8,16,32,64 bit signed and unsigned integers, bytes, Booleans, and 32 or 64 bit floats). Unlike C, Go has strings built in, and chars are not "built in". Rather, chars are runes, which are an integer value that points to a Unicode character. Finally, it has support for bytes (which is an alias for unsigned 8-bit integers), and complex numbers that are either 64-bits or 128-bits in length.

As stated earlier, Go aims at combining the simplicity of languages like Python with the type safety of statically typed languages like Java. This makes Go a very reliable language in that the behavior of modifying data should always work as expected as the data type never changes.. To accomplish this, they have implemented a static type checking system that is capable of both implicit typing and explicit typing. Variables are bound to the type assigned to it, but can either be explicitly written with the type or be inferred by the data given to it, either by using the var keyword and providing no type keyword or by using the := operator. Take this code for instance:

```
var x int = 0 //Explicitly typed as an integer
var y = 10.7 //Implicitly typed as a float
z := "hello" //Implicitly typed as a string
x = "This will not compile." //Types are static, thus we cannot
assign a string to x.
```

Running this will not compile because we cannot change the type of x, and we get this error

```
prog.go:12:6: cannot use "This will not compile." (type string) as
type int in assignment
```

However, if we remove the last line and use the TypeOf method in the reflect library, we can see that it returns that x is an int, y is float64, and z is string. This implicit typing may hamper readability, but this makes the language much more writable.

Control Abstractions

To discuss the control abstractions of Go, we will first discuss the order of operations that exist within the language. Go's operator precedence is very similar to C,

but is not perfectly analogous. A major difference between Go from C or other languages is that prefix or postfix increment or decrementing is not actually an expression, but rather, it is a statement. Thus, ++ or -- fall nowhere on the hierarchy of operators. Consequently, this means that parenthesis have no effect on the operation. For instance, we will create two variables, x which will equal 42 and ptr_x which is a pointer to x.

Consider this snippet of code:

```
//Go code
(*ptr_x)++ //Val = 43 Address = 10414020
*ptr_x++ //Val = 44 Address = 10414020
//C code
(*ptr_x)++; //Val = 43 Address = e36c4a4c
*ptr_x++; //Val = -479442328 Address = e36c4a50
```

C has the increment operator at a higher precedence than the dereference operator, thus the pointer is moved to another memory location and the value of this location is dereferenced without using the parenthesis. Go, however, has the same result either way which is to increment the value held at the address. Otherwise, the order of precedence is pretty standard with regular mathematical expressions following standard order of operations, and bitwise operators also have high precedence.

Also very standard in the language is the selection constructs. This is provided by if/else statements or switch statements. A strange quirk of the conditionals, however, due to the way whitespace is handled is how if/else statements must be formatted. Else if and else clauses must be on the same line as the closing bracket of the previous condition, or the program will not compile.

Iteration is somewhat unique within Go. Go contains no keyword for while loops, everything is handled using the for keyword. To create a while loop, you just simply omit the assignment and iteration parts of the for loop and only put the conditional part

```
for x < 10{/*code*/}//While loop
for x := 0; x < 10; x++{/*code*/}//For loop
```

These are the only forms of loops Go support (aside from recursion). Thus, foreach and do-while are not supported.

Functions are all pass-by-value in Go. Pass-by-reference can be simulated much like C, but is not built into the language, nor does Go support reference types. In order to use values from a function, you must either assign the return value from the function to a variable, or pass the address of the variable you would like to modify to the function in order for any changes in a function to have an effect.

Scope is also very much standard fare for Go. The two basic levels of scope in Go are the global scope and local scope. Data declared in the global scope can be access anywhere in the program, and anything declared within a function can only be accessed within the function, and anything declared within a loop or a conditional can only be accessed within those sections of the program.

The package system used by Go, however, is fairly unique. As stated earlier, one of the goals of Go was to eliminate the clutter of header files. Header files do not exist in Go leaving only the source files. Go programs have “workspaces”, which are directories that contain three basic subdirectories: bin for binaries, pkg for packages, and src for

source code. To call on a library, you simply just point it to the directory that contains the source file. Any source file that is within the same directory does not need to be imported.

Finally, one very unique feature within Go is its exception handling. Go has exception handling; however, this is not the traditional try-catch-finally system you see with most languages that include exception handling. Rather, functions return multiple values as a pair with the actual return value, and an error value to return in the case of any illegal behavior. Thus, if the function returns an error code, it can be handled by the function that is calling it based upon the code it is given.

Bibliography

"Frequently Asked Questions (FAQ)." *Frequently Asked Questions (FAQ) - The Go Programming Language*, golang.org/doc/faq.

"Documentation." *Documentation - The Go Programming Language*, golang.org/doc/.

Golang. "Golang Users." *GitHub*, github.com/golang/go/wiki/GoUsers.

"C Operator Precedence." *C Operator Precedence - Cppreference.com*, en.cppreference.com/w/c/language/operator_precedence.

"TIOBE Index." *TIOBE Index* , www.tiobe.com/tiobe-index/.