Interaction with C and C++

Go lets its users write programs in combination with Go and C/C++. This lesson covers how a Go program can have functionalities of C and C++ imported in it.

Interacting with C

The **cgo** program provides the mechanism for **FFI**-support (**F**oreign **F**unction **I**nterface) to allow safe calling of C libraries from Go code. Here is the link to the primary cgo documentation. The cgo replaces the normal Go-compilers, and it outputs Go and C files that can be combined into a single Go package. It is good practice to combine the calls to C in a separate package. The following import is then necessary in your Go program:

```
import "C"
```

and usually also:

```
import "unsafe"
```

You can include C-libraries (or even valid C-code) by placing these statements as comments (with // or /* */) immediately above the `import "C"` line:

```
// #include <stdio.h>
// #include <stdlib.h>
import "C"
```

C is not a package from the standard library, and it is simply a special name interpreted by cgo as a reference to C's namespace. Within this namespace exist the C types denoted as `C.uint`, `C.long`, and so on, and functions like `C.random()` from `libc` can be called. Variables in the Go program have to be converted to the C type when used as parameters in C functions, and vice-versa. For example:

```
var i int
C.uint(i) // from Go int to C unsigned int
int(C.random()) // from C long (random() gives a long type number) to Go i
nt
```

In the above snippet, variable `i` is a Go type variable, which is converted to C type by `C.unit()` function. Then, in the next line, we create a random number of type C using `C.random()` function, which is then converted to Go type using `int()` function.

Strings do not exist as explicit type in C, which means to convert a Go string `s` to its C equivalent use:

```
C.CString(s)
```

The reverse is done with the function:

```
C.GoString(cs)
```

where `cs` is a C string. Memory allocations made by C code are not known to Go's memory manager, so they are not garbage collected. It is up to the developer to free the memory of C variables with `C.free`, as follows:

```
defer C.free(unsafe.Pointer(Cvariable))
```

This line best follows the line where `Cvariable` is created to make sure the memory release happens at the end of the scope.

Interacting with C++

SWIG (**S**implified **W**rapper and **I**nterface **G**enerator) (see documentation) support exists for calling C++ and C code from Go on Linux. Using SWIG is a bit more involved:

- Write the SWIG interface file for the library to be wrapped.

- SWIG will generate the C stub functions.

- These can then be called using cgo, doing so the Go files are automatically generated as well.

This interface handles overloading, multiple inheritance, and allows us to provide a Go implementation for a C++ abstract class.

That is all about interacting with C and C++. The next lesson includes some helpful resources to catch up with Golang.