Network Programming with Go

Operators

Arithmetic operators		
+	sum	integers, floats, complex values, strings
-	difference	integers, floats, complex values
*	product	integers, floats, complex values
/	quotient	integers, floats, complex values
%	remainder	integers
&	bitwise AND	integers
|	bitwise OR	integers
^	bitwise XOR	integers
&^	bit clear (AND NOT)	integers
<<	left shift	integer << unsigned integer
>>	right shift	integer >> unsigned integer
Comparison operators		
==	equal	
!=	not equal	
<	less	
<=	less or equal	
>	greater	
>=	greater or equal	
Logical operators		
&&	conditional AND	p && q is "if p then q else false"
||	conditional OR	p || q is "if p then true else q"
!	NOT	!p is "not p"

Variables

```
VarDecl = "var" ( VarSpec | "(" { VarSpec ";" } ")" ) .
VarSpec = IdentifierList ( Type [ "=" ExpressionList ] | "=" ExpressionList ) .
```

```
var i int
var U, V, W float64
var k = 0
var x, y float32 = -1, -2
var (
    i int
    u, v, s = 2.0, 3.0, "bar"
)
var re, im = complexSqrt(-1)
var _, found = entries[name] // map lookup; only interested in "found"
```

uint8	the set of all unsigned 8-bit integers (0 to 255)
uint16	the set of all unsigned 16-bit integers (0 to 65535)
uint32	the set of all unsigned 32-bit integers (0 to 4294967295)
uint64	the set of all unsigned 64-bit integers (0 to 18446744073709551615)
int8	the set of all signed 8-bit integers (-128 to 127)
int16	the set of all signed 16-bit integers (-32768 to 32767)
int32	the set of all signed 32-bit integers (-2147483648 to 2147483647)
int64	the set of all signed 64-bit integers (-9223372036854775808 to 9223372036854775807)
float32	the set of all IEEE-754 32-bit floating-point numbers
float64	the set of all IEEE-754 64-bit floating-point numbers
complex64	the set of all complex numbers with float32 real and imaginary parts
complex128	the set of all complex numbers with float64 real and imaginary parts
byte	alias for uint8
rune	alias for int32

Packages

Clause

```
// PackageClause = "package" PackageName .
// PackageName = identifier .

package math
```

Import declarations

```
//ImportDecl = "import" ( ImportSpec | "(" { ImportSpec ";" } ")" ) .
//ImportSpec = [ "." | PackageName ] ImportPath .
//ImportPath = string_lit .

package main - // Package declaration

// Multiple import statements
import "fmt"
import "time"
import "math"

// Factored import statements
import (
    "fmt"
    "time"
    "math"
)
```

Comments

```
// Line comments
/* General comments - closed */
```

Reserved Keywords

break • default • func • interface • select • case • defer • go • map • struct • chan • else • goto • package • switch • const • fallthrough • if • range • type • continue • for • import • return • var

Controls

If, Else

```
if x > 0 {
    return x
} else {
    return -x
}
```

Loop

```
// Only `for`, no `while`, no `until`
for i := 1; i < 10; i++ {
}
for ; i < 10; { // while - loop
}
for i < 10 { // omit semicolons if there is only a condition
}
for { // you can omit the condition ~ while (true)
}
```

Switch

```
switch tag {
    default: s3()
    case 0, 1, 2, 3: s1()
    case 4, 5, 6, 7: s2()
}
```

Arrays • Slices • Ranges

Arrays

```
[32]byte
[2*N] struct { x, y int32 }
[1000]*float64
[3][5]int
[2][2][2]float64 // same as [2]([2]([2]float64))
```

Slices

```
make([]T, length, capacity)

// the following two expressions are equivalent
make([]int, 50, 100)
new([100]int)[0:50]
```

Ranges

```
// RangeClause = [ ExpressionList "=" | IdentifierList ":=" ] "range" Expression .

var a [10]string
for i, s := range a {
    // type of i is int
    // type of s is string
    // s == a[i]
    g(i, s)
}
```

Functions

Structure

```
func function_name( [parameter list] ) [return_types]
{
    Function body
}
```

AWS Lambda Function Handler in Go

Based on AWS docs: https://docs.aws.amazon.com/lambda/latest/dg/golang-handler.html

```
package main

import (
    "fmt"
    "github.com/aws/aws-lambda-go/lambda"
)

type MyEvent struct {
    Name string 'json:"What is your name?"'
    Age int 'json:"How old are you?"'
}

type MyResponse struct {
    Message string 'json:"Answer:"'
}

func HandleLambdaEvent(event MyEvent) (MyResponse, error) {
    return MyResponse{Message: fmt.Sprintf("%s is %d years old!", event.Name,\
event.Age)}, nil
}

func main() {
    lambda.Start(HandleLambdaEvent)
}
```

AWS S3 Bucket List

```
package main

import (
    "fmt"

    "github.com/aws/aws-sdk-go/aws"
    "github.com/aws/aws-sdk-go/aws/session"
    "github.com/aws/aws-sdk-go/service/s3"
)

func main() {
    s3svc := s3.New(session.New())
    result, err := s3svc.ListBuckets(&s3.ListBucketsInput{})
    if err != nil {
        fmt.Println("Buckets list failed", err)
        return
    }

    fmt.Println("Buckets:")
    for _, bucket := range result.Buckets {
        fmt.Printf("%s : %s\n", aws.StringValue(bucket.Name),
        bucket.CreationDate)
    }
}
```

Test the TCP Client and Server

```
//Run your TCP server. From the directory containing the tcpS.go file, run the following command:

go run tcpS.go 1234

//The server will listen on port number 1234. You will not see any output as a result of this command.
//Open a second shell session to execute the TCP client and to interact with the TCP server. Run the following command:

go run tcpC.go 127.0.0.1:1234

//Note: If the TCP server is not running on the expected TCP port, you will get the following error message from tcpC.go:

dial tcp [::1]:1234: connect: connection refused

//You will see a >> prompt waiting for you to enter some text. Type in Hello! to receive a response from the TCP server:

Hello!

//You should see a similar output:

>> Hello!
->: 2019-05-23T19:43:21+03:00

//Send the STOP command to exit the TCP client and server:

STOP

//You should see a similar output in the client:

>> STOP
->: TCP client exiting...

//The output on the TCP server side will resemble the following:

-> Hello!
Exiting TCP server!
```

TCP Socket programming

Attribution: https://www.linode.com/docs/development/go/developing-udp-and-tcp-clients-and-servers-in-go/#create-the-tcp-client

Create the TCP Client

```
//create a file named tcpC.go

package main

import (
    "bufio"
    "fmt"
    "net"
    "os"
    "strings"
)

func main() {
    arguments := os.Args
    if len(arguments) == 1 {
        fmt.Println("Please provide host:port.")
        return
    }

    CONNECT := arguments[1]
    c, err := net.Dial("tcp", CONNECT)
    if err != nil {
        fmt.Println(err)
        return
    }

    for {
        reader := bufio.NewReader(os.Stdin)
        fmt.Print(">> ")
        text, _ := reader.ReadString('\n')
        fmt.Fprintf(c, text+"\n")

        message, _ := bufio.NewReader(c).ReadString('\n')
        fmt.Print("->: " + message)
        if strings.TrimSpace(string(text)) == "STOP" {
            fmt.Println("TCP client exiting...")
            return
        }
    }
}
```

Create the TCP Server

```
//create a file named tcpS.go

package main

import (
    "bufio"
    "fmt"
    "net"
    "os"
    "strings"
    "time"
    )

func main() {
    arguments := os.Args
    if len(arguments) == 1 {
        fmt.Println("Please provide port number")
        return
    }

    PORT := ":" + arguments[1]
    l, err := net.Listen("tcp", PORT)
    if err != nil {
        fmt.Println(err)
        return
    }
    defer l.Close()

    c, err := l.Accept()
    if err != nil {
        fmt.Println(err)
        return
    }

    for {
        netData, err := bufio.NewReader(c).ReadString('\n')
        if err != nil {
            fmt.Println(err)
            return
        }
        if strings.TrimSpace(string(netData)) == "STOP" {
            fmt.Println("Exiting TCP server!")
            return
        }

        fmt.Print("-> ", string(netData))
        t := time.Now()
        myTime := t.Format(time.RFC3339) + "\n"
        c.Write([]byte(myTime))
    }
}
```