Backend Programming

M. Yauri M. Attamimi

Upgrade your potential !

Foundation Class

- WEEK 1 (Session 1) -

About the Speaker

- 15 years (or so) in Software Development
- LOTS of project experience
- Java, NodeJS, Golang, Python
- Microservices Provocateur
- AI & Blockchain Enthusiast
- TDD and Clean Code Evangelist
- VP Engineering & Co-Founder of Automate.id

https://about.me/yauritux

github.com/yauritux
Twitter: @yauritux
https://dzone.com/users/366249/yauritux.html

My Professional Mission:

Guiding individuals and organizations to commercial success through the application of modern technologies

What will you get from this class ?

- Solid introduction to Go
- A practical guide /excellent starting point to mastering and having fun with Go as your programming language of choice
- Intended Audiences : Someone who’d like to know about Go (no prior knowledge)

Caveats

- ➔ Programming language is an evolving topic and changes rapidly
- ➔ Goal is to be reasonably comprehensive
- ➔ Enable you to fill in gaps yourself once you know what does it need to be a master

Introduction to Algorithms

What and Why?

Programs = Algorithms + Data Structures

- Algorithms
  - various ways to solve a given problem
- Data Structures
  - How to efficiently store, access, and manage data
  - Effects algorithm’s performance

Example Algorithms

- Two algorithms for computing the Factorial
- Which one is better ?

```go
func factorial(n int) int {
    if n <= 1 {
        return 1
    }
    return n * factorial(n-1)
}
```

```go
func factorial(n int) int {
    if n <= 1 {
        return 1
    } else {
        fact := 1
        for k:=2; k<=n; k++ {
            fact *= k
        }
        return fact
    }
}
```

Example of Famous Algorithms

- Bubble-Sort, Quick-Sort, Heap-Sort, etc (Sorting)
- Constructions of Euclid
- Newton's root finding
- Fast Fourier Transform (signal processing)
- Compression (Huffman, Lempel-Ziv, GIF, MPEG)
- DES, RSA Encryption (network security)
- Simplex algorithm for linear programming (optimization)
- Shortest Path Algorithms (Djikstra, Bellman-Ford)
- Error Correcting Codes (CDs, DVDs)
- TCP Congestion Control, IP Routing (computer networks)
- Pattern Matching (Genomics)
- Search Engines (www)

Role of Algorithms in Modern World

Enormous amount of data

- Network traffic (telecom billing, monitoring)
- Database transactions (Sales, Inventory)
- Scientific measurements (geology, biology)
- Sensor networks. RFID tags
  - Radio Frequency Identification (RFID) is a method of remotely storing and retrieving data using devices called RFID tags.
- etc

A Real World Problem

- Communication in the Internet
- Message (email, ftp) broken down into IP packets
- Sender/Receiver identified by IP address
- The packets are routed through the Internet by special computers called Routers
- Each packet is stamped with its destination address, but not the route
- Because the Internet topology and network load is constantly changing, routers must discover routes dynamically
- What should the Routing Table look like?

IP Prefixes and Routing

- Each router is really a switch: it receives packets at several input ports, and appropriately sends them out to output ports
- Thus, for each packet, the router needs to transfer the packet to that output port that gets it closer to its destination
- Should each router keep a table: IP Address x Output port?
- How big is this table?
- When a link or router fails, how much information would need to be modified?
- A router typically forward several million packets/sec!

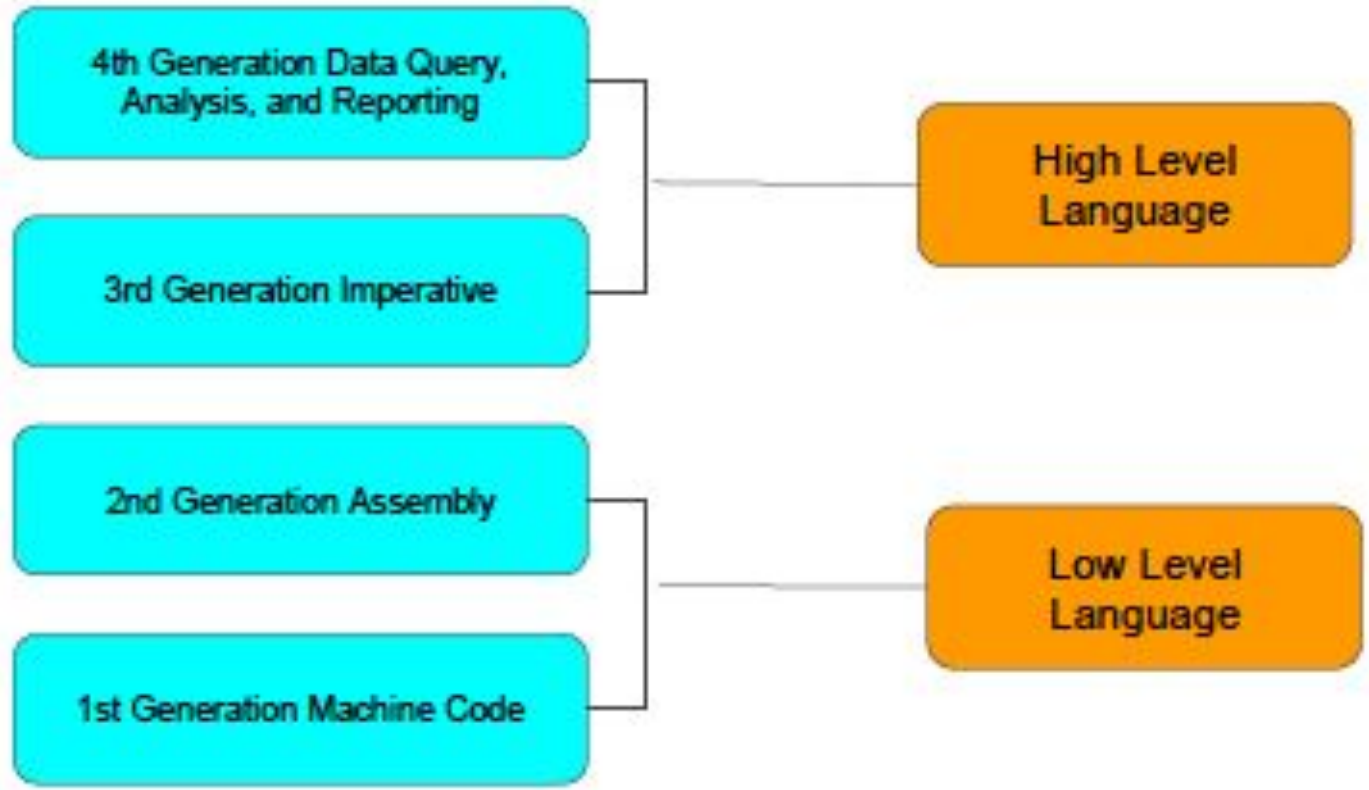
Data Structures

- The IP Packet Forwarding is a Data Structure problem!
- Efficiency, Scalability is very important
- Similarly, how does Google find the documents matching your query so fast?
- Uses sophisticated algorithms to create index structures, which are just data structures
- Algorithms and Data Structures are ubiquitous

Other Real-world Problems

- Correlation between time spent at a website and purchase amount?
- Did source “s” send a packet in last “s” seconds?
- Send an alarm if any international arrival matches a profile in the database
- etc

Programming Classifications


4th Generation Data Query, Analysis, and Reporting
3rd Generation Imperative
2nd Generation Assembly
1st Generation Machine Code
High Level Language
Low Level Language

Low-Level Languages:

Machine-Code and Assembly-Code

High-Level Languages:

A programming language that allows programs to be written in English keywords and is platform independent

Introduction to Go

Quick Introduction

- Created by Google's Engineer ⇒ Invented by Robert Griesemer, Rob Pike, and Ken Thompson in 2007 (public in '09)
- Born out of a need for ease of programming combined with type safety and portability
- A language for the multi-core processors
- Other goals:
  - Easy to learn
  - Multi-platform language
  - Easy concurrency via Channels and Goroutines
  - Low latency garbage collection
  - Fast compilation

Who is Ken Thompson?

- Kenneth Lane **Thompson** (born February 4, 1943), commonly referred to as **ken** in hacker circles, is an American pioneer of computer science.
- Having worked at Bell Labs for most of his career, **Thompson** designed and implemented the original Unix operating system.

Why Google Started Go

- No major system language has emerged in over a decade
- Computers are enormously quicker but software development is not faster
- C tradition header file based dependency management is much cleaner and faster in compilation
- Dynamically typed languages such as Python, JS are much easier to use
- Fundamental concepts such as garbage collection and parallel computation were not supported well in popular system languages

Why Google Started Go

- None of the system languages were providing following 3 aspects in one:
  - Efficient compilation
  - Efficient execution
  - Ease of programming
- Go is an attempt to combine the ease of programming of an interpreted, dynamically typed language with the efficiency and safety of a statically typed, compiled language

Go Ancestors

- Mostly in the C family (basic syntax)
- Input from Python, JS
- Significant input from Pascal/Modula/Oberon family (declarations, packages)
- Ideas from CSP style languages Newsqueak, Limbo (concurrency)

In every respect, it’s a new language built by thinking what programmers do and how to make programming more efficient and more fun

Interesting Caveats

- Unused variables are a compilation error
- Unused import directives also a compilation error
- Go focuses to solve engineering problem than language design
  - No Classes ; structs instead of classes
  - No overloading
  - No inheritance
- Not object-oriented, but has support for interfaces, more like functional programming
- No exception handling, uses multi-value returns
- Provides Garbage Collection
- CSP style concurrent programming

The Reason for Go

Goals:

- Eliminate slowness
- Eliminate clumsiness
- Improve effectiveness
- Maintain (even improve) scale

Go was designed by and for people who write, read, debug, and maintain large software systems at Google since development at Google can be slow, often clumsy.

In short: Go aims to help software engineering (focus on software engineering rather than talk about language design)

What is Go good for?

"Large programs written by many developers, growing over time to support networked services in the cloud: in short, server software"

- Rob Pike

The Design of Go

from Software Engineering Perspective

Dependencies in Go

Dependencies are defined (syntactically) in the language.

Explicit, clear, computable.

```
import "encoding/json"
```

Unused dependencies cause error at compile time.

Efficient: dependencies traversed once per source file.

Hoisting Dependencies

Consider:
A imports B imports C but A does not directly import C.
The object code for B includes all the information about C needed to import B.
Therefore in A the line

```
import "B"
```

does not required the compiler to read C when compiling A.
Also, the object files are designed so the “export” information comes first; compiler doing import does not need to read whole file.

Exponentially less data read than with `#include` files.

With Go in Google, about 40x fanout (recall C++ was 2000x)
Plus in C++ it’s general code that must be parsed; in Go it’s just export data.

No Circular Import

Circular imports are illegal in Go.

The big picture in a nutshell:

- Occasional minor pain,
- but great reduction in annoyance overall
- structural typing makes it less important than with type hierarchies
- keeps us honest!

Forces clear demarcation between packages.

Simplifies compilation, linking, initialization.

Syntax

Go has a clean syntax; not super-small, just clean.

- Only 25 keywords
- Straightforward to parsed (no type-specific context required)
- Easy to predict, reason about.

Declarations

Uses Pascal/Modula-style syntax: name before type, more type keywords.

```
var fn func([]int) int
type T struct { a, b int }
```

not

```
int (*fn)(int[]);
struct T { int a, b; }
```

Easier to parse—no symbol table needed. Tools become easier to write.

One nice effect: can drop `var` and derive type of variable from expression:

```
var buf *bytes.Buffer = bytes.NewBuffer(x) // explicit
buf := bytes.NewBuffer(x)                  // derived
```

For more information:

golang.org/s/decl-syntax

No Default Arguments

Go does not support default function arguments.

Why not ?

- Too easy to throw in defaulted args to fix design problems.
- Encourage too many args.
- Too hard to understand the effect of the fn for different combination of args.

Extra verbosity may happen but that encourages extra thought about names.

Related: Go has easy-to-use, type-safe support for variadic functions.

Nothing's Perfect

Why Go Is Not Good

Now you may say "But why is Go not good? This is just a list of complaints; you can complain about any language!". This is true; no language is perfect. However, I hope my complaints reveal a little bit about how

- Go doesn't really do anything new.
- Go isn't well-designed from the ground up.
- Go is a regression from other modern programming languages.

Three Months of Go (from a Haskeller's perspective)

25 Aug, 2016

Other than that, I will probably never choose to use Go for anything ever again, unless I'm being paid for it. Go is just too different to how I think: when I approach a programming problem, I first think about the types and abstractions that will be useful; I think about statically enforcing behaviour; and I don't worry about the cost of intermediary data structures, because that price is almost never paid in full.

For me :

1. It all depends on "Man behind the Gun"
2. Use the right tool for the right job

Popular Products using Go

- Docker, the world famous containerization platform is developed using Go
- Kubernetes, was also developed by Google using Go
- Dropbox has migrated their performance critical components from Python to Go.

Companies Using Go (Other than Google)

Reasons:

- Compilation speed.
- Process large amount of data in a matter of seconds.
- Easy built-in concurrency support for developers (CSP over Traditional Lock) → https://golang.org/doc/faq#csp

Setup and Installation

Setup and Installation

- Generally: download from https://golang.org/dl/
- Untar/unzip the file ; alternatively, install by using package manager such as brew (Mac/OSX), apt (Debian/Ubuntu), etc
- Set the following environment variables:
  - ~~GOROOT ⇒ installation directory of Go~~
  - GOPATH
  - ~~PATH ⇒ $GOROOT/bin:$GOPATH:$PATH~~

Go Workspace

- Is a physical location on disk where you will load and work with Go code.
- Go workspace can’t be the same location where Go is installed.
- Create a new set of folders to represent the Go workspace. There is a special folder that must exist inside Go workspace named src. All the code you load and work on must exist inside the src folder.
- The src folder represents the start of Go workspace.
- GOPATH is an environment that points to your Go workspace.
- Occasionally, you’ll also need to set GOBIN

Go Modules

- Since version of 1.11+, Go supports modules which means you don’t need to set GOPATH anymore. (it’s still experimental feature on Go 1.11, and the final release is on version 1.12)
- Go modules in a nutshell is a built-in dependency versioning and dependency management feature for Go.
- Check : https://blog.golang.org/using-go-modules

Greeting Application

Say Hello in Go

```go
package main

import "fmt"

func main() {
    fmt.Println("Hi Gopher, welcome aboard")
}
```

Import & Factored Import

Anatomy of Import Declaration

ImportDeclaration = “import” ImportSpec
ImportSpec = [“.” | “_” | Identifier] ImportPath

- Identifier is any valid identifier which will be used in qualified identifiers
- ImportPath is string literal (raw or interpreted)

Examples:

```
import . “fmt”
import _ “io”
import log “github.com/g2lab/advlog
import m “math”
```

Factored Import Declaration

```
import (
   “fmt”
   “strings”
   log “github.com/g2lab/advlog”
)
```

Anatomy of .go File

```
// description...
package main // package clause

// zero or more import declarations
import (
    "fmt"
    "strings"
)

import "strconv"

// top-level declarations

func main() {
    fmt.Println(strings.Repeat(strconv.FormatInt(15, 16), 5))
}
```

Go Variable Declarations

Declare Variables

```
var num int
num = 5
var proj string
proj = “G2Lab”
```

```
var (
    num int = 5
        proj string = “G2Lab”
    ) // factored declaration
```

or

```
var num int = 5
var proj string = “G2Lab”
```

or

```
var num = 5 // type inference (int)
var proj = “G2Lab” // type inference (string)
```

or

```
num := 5 // short-hand syntax
proj := “G2Lab // short-hand syntax
```

Basic Data Types, Casting

Basic Data Types in Go

```
bool
string

Numeric types:

uint         either 32 or 64 bits
int          same size as uint
uintptr      an unsigned integer large enough to store the uninterpreted bits of
             a pointer value
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 (represents a Unicode code point)
```

Type Conversion (Casting)

- The expression T(v) converts the value v to the type T. Some numeric conversions:

```
var i int = 42
var f float64 = float64(i)
var u uint = uint(f)
```

- Go assignment between items of different type requires an explicit conversion which mean that you manually need to convert types if you are passing a variable to a function expecting another type.

Basic I/O

Basic Console I/O

INPUT

```
reader := bufio.NewReader(os.Stdin)
text, _ := reader.ReadString('\n')
ch, _, err := reader.ReadRune()

scanner := bufio.NewScanner(os.Stdin)
scanner.Scan()
scanner.Text()
```

OUTPUT

```
fmt.Println
fmt.Printf
```

Check out

https://exercism.io

SUMMARY

You have studied about the role of algorithms and data structures in programming, how Go can be used to solve your engineering problem, how to install and prepare Go workspace for your local development, Go modules, basic syntax, data types, and some

basic console I/O operations

Thank You