GOLANG YES!

A PREPRINT

Jiacheng Yang
Department of Computer Science
The University of Arizona
jiachengyang@email.arizona

Wenkai Zheng
Department of Computer Science
The University of Arizona
wenkaizheng@email.arizona

April 6, 2020

ABSTRACT

abstract Golang

***K*eywords** Go · Golang

1 Introduction

Introduction XXX

2 History

Golang was designed by Rob Pike, Ken Thompson, and Robert Griesemer in order to achieve fast execution speed and high efficiency of development. In comparison to popular compiled languages like C, C++, and Java, programs written in Golang are only slightly slower than the same programs written in C and C++ but faster than the ones written in Java, while it needs much less time to compile source code. Another feature that Golang offers is the ability to easily writing concurrency programs. The keyword "go" enables programmers to create a go-routine, which is similar to but costs less than a thread in Java or C++.

3 Control Structures

Golang has four kinds of control structures: if-else, switch, for-loop, and defer. If-else, switch, and for-loop work very similar in C. The keyword, defer, is designed for error handling and resource cleaning up.

3.1 Control Structures: if-else

If-else in Golang works just like in C with a minor difference that is variables can be defined in the same line of condition statement as the code below. This feature is useful when the condition statement and the body of if-else both need the same value returned by another function. Programmers can just declare a variable to receive it and use it in both places.

```
if x:= somefunction(); x%2 == 0 {
    fmt.Print(x)
    fmt.Println(" is even")
} else {
    fmt.Print(x)
    fmt.Println(" is odd")
}
```

This feature also benefits error handling. For example, when handling errors, since errors in Golang are returned by functions, callers can declare variables to receive the return values and then check errors in the condition statement. After that, errors can be handled in the body of if-else. This usage of if-else avoids declaring variables in the name-space outside of the if-else.

```
reader := bufio.NewReader(os.Stdin)
if x, err := reader.ReadString('\n'); err==nil {
    fmt.Println("read: "+ x)
} else {
    fmt.Println("fail")
}
```

3.2 Control Structures: switch

Switch in Golang is also slightly different from switch in C. One difference is that programmers no longer need to explicitly use break to break from cases; instead, programmers need to use the keyword, fallthrough, to explicitly execute the case below the current one. For example, if x in the code below is 2 or 3, the program will still print: "x is a prime less than 6."

```
switch x {
case 2:
    fallthrough
case 3:
    fallthrough
case 5:
    fmt.Println("x is a prime less than 6.")
}
```

The other way is to combine expected values into a single case as the code below.

```
switch x {
case 2, 3, 5, 7:
    fmt.Println("x is a prime less than 10")
default:
    fmt.Println("x is not a prime less than 10")
}
```

If a variable for comparison is not provided, switch in Golang then works like multiple if-else. Each case is evaluated from top to bottom, and the one evaluated to true will be executed.

```
switch {
case variable%2 == 0:
    fmt.Println("variable is divisible by 2")
case variable%3 == 0:
    fmt.Println("variable is divisible by 3")
case variable%5 == 0:
    fmt.Println("variable is divisible by 3")
default:
    fmt.Println("variable is not divisible by 2, 3 or 5")
}
```

3.3 Control Structures: for-loop

In Golang, iterations are implemented only by for-loops. This means there is no while and do-while loops. However, they can be easily and elegantly simulated by for-loops. A simple for-loop can be implemented as in C.

```
for i := 0; i < 10; i++ {
    sum += i
}
```

When a for-loop is not given initialization and change of variable, it works like a while loop, and programmers do no have to provide extra semi-colones as the code below.

```
for i < 10 {
     sum += i
     i++
}
```

For-loop in Golang can also work as for-loop in python. Programmers just need to use range keyword in a for-loop as the code below. Range keyword goes over an array or a string and assigns the index to the first variable and the value to the second variable defined in the for-loop. This feature allows programmers to easily iterate over an array or a string.

```
for i, char := range "abcdef"{
     fmt.Printf("Index %d is %s\n",i,string(char))
}
```

3.4 Control Structures: label and goto keyword

Like C, Golang supports labels and goto keyword. Programmers can define a label by a label name followed by a colon, and then it can be used by goto keyword. For instance, the code below will skip printing 2 and only print 1 and 3.

```
    fmt.Println("1")
    goto A
    fmt.Println("2")
A:
    fmt.Println("3")
```

Moreover, a loop can be label in order to break or continue a specific loop. For example, the code below will only print a = 0, a = 1, and then i = 2.

```
Loop1:
    for a:=0;a < 10; a++ {
        fmt.Println("a = "+ strconv.Itoa(a))
        for i := 1; i < 4; i++ {
            fmt.Println("i = "+ strconv.Itoa(i))
            if i%2 == 0 {
                break Loop1
            }
        }
}
```

3.5 Control Structures: defer

Golang has a unique keyword, defer. As mentioned at the beginning of this section, defer is designed to deal with error handling and resource cleaning up. Programmers can defer a call of a function immediately after the return of the current function. For example, the code snippet will first print zero and then one.

```
func foo() {
     defer func() { fmt.Println("one") }()
     fmt.Println("zero")
}
```

When there are multiple defer's, it follows the rule of FILO (first in last out). So, the code snippet below will print two after one.

```
func deferPrintln() {
     defer func() { fmt.Println("two") }()
     defer func() { fmt.Println("one") }()
}
```

Because a deferred function will always be executed regardless of errors, programmers can close resources, such as IO-stream, in a deferred function as the code below.

```
func writeTo(fileName string) {
     f, _ := os.Create(fileName)
     defer func() {f.Close()}()
     //write file
}
```

3.6 Control Structures: error handling

4 Data Types

Data Types xxx

5 Subprograms

Subprograms xxx

6 Summary

Summary xxx

References

References xxx