Project 6

CIS 424

Peter Parianos

Brian Salvoro

Log In ID: peparian

Project Documentation

Introduction to Go Language:

Go Language (Golang) is a programming language designed by the popular company, Google. The language was first created by engineers Robert Griesemer, Rob Pike, Ken Thompson around 2007. Golang is an open source programming language meaning, its source codes are open to the public and anyone can contribute in improving, developing the language further [1]. Submission of new proposals, finding and fixing bugs, tweaking the language for more efficiency and speed, these are some of the possible things everyone can do to contribute towards the language. Golang was created for the purpose of removing the possible obstacles other languages have especially in terms of speed of compilation, safety against misinterpretations of implicit data types, and complications with the rising complexity of the implementation [1]. Before Golang, there has been frustrations in programming due to non-efficient compilations and executions and the lack in ease of programming, Go is a language which addresses these problems and provide solutions to provide a strong and robust coding environment for programmers [2]. As a modernized programming language, Golang supports multiple types of functions, some of these are Variadic Funcs, Named Funcs, Anonymous Funcs, First-Class Funcs and Higher-Order Funcs.

Higher-Order Functions:

What are Higher-Order Functions? Higher-Order Functions is a mathematical and computer science term which is a function that can do at least one of the following: have one or more functions as its own argument, and be able to return a functions as a result [3]. Multiple programming languages out there supports High-Order Functions such as JavaScript, Python, C#, and Go Language. One reason why Higher-Order Function exists is because of abstraction in programming. Abstraction is a key concept especially in Object Oriented Programming to which, its main goal is to handle complexities of various implementations by hiding unnecessary details from the user [4].

This allows the user to implement more complexity over an existing function or method over the given abstraction without the need to see or think about the hidden complexities. Abstraction is not only limited to the programming world, it can be also found in the real world. Take making coffee with a coffee machine as an analogy for abstraction. Making coffee requires various ingredients such as water and coffee. We also need to know how to operate the coffee machine and insert the necessary ingredients to start the operation. The coffee machine would represent the abstraction of the operations as it hides the complexities of making coffee, while our actions of operating the coffee machine would be the added complexity over the hidden one [4]. Going back to Object Oriented Programming, since all internal implementations are hidden, the programmer only needs to know which methods of the objects are available to call. There is no need to see and understand about the methods of these hidden implementations to have the expected result or answer.

It was discussed earlier that the Go Language was created to address the lack of ease in programming. Higher-Order Functions is an evidence of Golang providing solutions in making implementations simple, easy and readable by hiding the unnecessary complexities behind numerous operations and having less burden to programmers.

Example of Higher-Order Functions in Go Language:

An example using Go Language where Higher-Order Function is used to do simple calculations. This example was found as a video in YouTube, it explains how Higher-Order Functions work and perform. A channel named “Kishore” explains in his video, *Go Programming Language: First class and Higher order functions* [5]. Using the IDE, GoLand, we typed his example so we can observe how it works and to be able to provide explanations.

```
func calculate(a, b int, functionToApply func(c, d int) int) int {
    return functionToApply(a, b)
}
```

Using the keyword “func”, we have defined a function called *calculate*. Here, it looks like a regular function, but this is where we also see the use of Higher-Order Functions. Remember that Higher-Order Function at least do one of the following, have one or more functions as its own argument, and be able to return a functions as a result. In the function *calculate*, it has three arguments, the first two are “a” and “b” which are integer types, and the third argument is a function called *functionToApply* which also has it own arguments: “c” and “d” which are both integer values. The third “int” from the left means *functionToApply* is expected to return an integer value, while the last “int” means *calculate* also has to return an integer value. The whole function then returns *functionToApply* with parameters of a and b. Shown in this example that functions in Golang can indeed accept a function as an argument, hence demonstrating Higher-Order functionality.

```
func getSqFunction() func(a int) int {
    return func(a int) int {
        return a * a
    }
}
```

The next example above also demonstrates Higher-Order functionality. A function named *getSqFunction* is defined. We saw that this function has an empty parentheses indicating no arguments has been passed, but could return a function with an argument “a” which is an integer type. The returned object/value is specified after the name of the function is stated. The return statement of *getSqFunction* should return the function which takes a single integer and returns the integer. Recall that one of the things Higher-Order Functions should at least do is for functions to be able to return a functions as a result.

```go
func higherOrderFunctions() {
    multiplicationFunction := func(a, b int) int {
        return a * b
    }
    divisionFunction := func(a, b int) int {
        return a / b
    }
    fmt.Println( a...: "Product of 10 and 20: ", calculate( a: 10,  b: 20, multiplicationFunction))
    fmt.Println( a...: "Quotient 10 over 20: ", calculate( a: 10,  b: 20, divisionFunction))

    sqFunction := getSqFunction()
    fmt.Println( a...: "Square of 10: ", sqFunction( a: 10))
}
func main(){
    higherOrderFunctions()
}
```

Next is a regular function called *highOrderFunction* which will be calling the functions with Higher-Order functionalities to perform some operations. Some topics which has not been discussed yet in this report are the “First-Class Functions” and” Anonymous Functions.” First-Class Functions are functions that are treated as any variables while Anonymous Functions are functions that are not in the program file and does not have an identifier or function name. From above, we can see both functions when we construct the variables *multiplicationFunction* and *divisionFunction*. Both are assigned to a function with no identifier. Highlighted, we see the function calls of those Higher-Order Functions defined earlier and from the main, we call *higherOrderFunction* to run the program. Below is the result:

```
go build exampleProj6.go
<4 go setup calls>
Product of 10 and 20:  200
Quotient 10 over 20:  0
Square of 10:  100

Process finished with exit code 0
```

Programming Project Implementation:

```
1  package main
2
3  import "fmt"
4
5  func square(a int) int {
6    return a * a
7  }
8  func Map(f func(int) int, arr []int) []int {
9    length := len(arr)
10   for i := 0; i < length; i++ {
11     arr[i] = f(arr[i])
12   }
13   return arr
14 }
15
16 func main() {
17   arrayin := []int{1, 2, 3, 4, 5}
18   Map(square, arrayin)
19   fmt.Println(arrayin)
20   arrayout := Map(func(a int) int { return a + a }, arrayin)
21   fmt.Println(arrayout)
22 }
```

In this project sample, we made a simple program that has three different methods created: *square*, *Map*, and *main()*. The package declaration on line that would determine what the program will call the function main first. This also tells where the executable package first. As we know, to create a binary executable file, we need our program to be a part of main package and it must have main function which is the entry point of execution [6]. On line 5, function *square* will be used to be a parameter to the function *Map*, which is a function that uses a function as a parameter, i.e., Higher-Order function. This is function *f*, on line 8.

Map takes two parameters: a function named *f*, and an array named *arr* which will be used as the input array as well as returned to the caller. On line 10, we loop through *arr* C-style, which is acceptable syntax in Go. We used the same address location where the value is stored, then rewrite it with the output from the function *f*. This overwrites the previous values stored in *arrayin* that is declared in line 17. It allows us to save memory

in the long run if the program needed to conserve space, so we can just use “println()” on *arrayin* to show the changes made. In order to print, a package name *fmt* was imported on line 3 which includes everything needed to print to the console.
In order to demonstrate anonymous function in Go, we anonymously declared a function in the caller parameter of Map() used on line 20. The syntax is to use the key-word “func” with parameters declared, and the body of the function is created in between curly braces. We simply return the doubled value of the parameter *a.*

One of the reasons to use Higher-order functions include that there is another level of abstraction that allows us to not yet define the function that is applied to the array. It allows us to abstract over actions, not just values. Hypothetically, if the program was implemented without higher-order functionality, we would have to create a new function for each operation such as square and then add. One of the advantages of abstraction using Higher-Order Functions is that its makes shorter and easier to interpret by the programmer, as long as we are aware of how a function works [7].

Debugging and Testing Results

Some of the errors that occured when creating this project is when declaring a parameter, you need to specify what the parameter type will be. Also you need to specify what the type will be when you returned something from the function.

The programming environment that was used when implementing this project is Atom with go-plus package installed within. This allows easy intelliSense that helps autocomplete the code and help you program faster. Another programing environment we used was the IDE, GoLand. GoLand is published by JetBrains, it is a great IDE and it also supports intelliSense. In order to build and run our Go code I installed a built in terminal and used the command “go run <name_of_file>” which compiled and ran the program. Another option for compilation is the “go build <name_of_file>” option which would only create an executable binary. You will need to run this by using bash “.\”.

Because I was using Windows PowerShell as my command line interface, this also worked for me.

OUTPUT:

```
PS C:\Users\peter\OneDrive - Cleveland State University\Documents\School\CIS424 - Comparative Program Languages\Homeworks\Project6> go run Map.go
[1 4 9 16 25]
[2 8 18 32 50]
```

Conclusion:

In conclusion, Go language is very suitable for large scale programs that require precise memory management. GoLang also allows for open-source editing on itself, which is helpful for finding bugs and also appeals to the programmer using it. GoLang supports Higher-Order functionality, which is a function that can at least do one of the following: have one or more functions as its own argument, and be able to return a functions as a result. This also allows for abstraction, which can make it easier to understand, interpret, and better implemented. Higher-Order functions allows programmers to add another level of abstracts on the actions or operations. Our examples show a couple of use cases, and the use of higher-order and anonymous functions made our program simpler and implemented in less lines of code. Our opinion on GoLang is that is it a good language to learn, because of it being open-source and its flexibility. We would suggest GoLang to programmers because it removes the possibility of having errors due to implicit declaration of variables due to its strong-type characteristic.

References

1. I. Gumus, “About Go Language - An Overview – Learn Go Programming,” *Learn Go Programming*, 20-Sep-2017. [Online]. Available: https://blog.learngoprogramming.com/about-go-language-an-overview-f0bee143597c. [Accessed: 02-Dec-2018].
2. “Frequently Asked Questions (FAQ),” *The Go Project - The Go Programming Language*. [Online]. Available: https://golang.org/doc/faq#history. [Accessed: 04-Dec-2018].
3. N. Perez , “Higher order functions, what are they? – Hacker Noon,” *Hacker Noon*, 11-Dec-2015. [Online]. Available: https://hackernoon.com/higher-order-functions-what-are-they-be74111659e8. [Accessed: 02-Dec-2018].
4. T. Janssen, “OOP Concept for Beginners: What is Abstraction?,” *Stackify*, 21-Nov-2017. [Online]. Available: https://stackify.com/oop-concept-abstraction/. [Accessed: 04-Dec-2018].
5. Nagakishore Sidde, “5. Go Programming Language : First class and Higher order functions,” *YouTube*, 23-Apr-2017. [Online]. Available: https://www.youtube.com/watch?v=GNUOwhOw3nA. [Accessed: 02-Dec-2018].
6. U. Hiwarale, “Everything you need to know about Packages in Go,” *Medium*, 20-Jul-2018. .
7. M. Haverbeke, *Eloquent javascript: a modern introduction to programming,* 3rd ed. S.l.: OReilly Media, 2018.

Biography:

Peter Parianos is a Senior that is studying Computer Science at Cleveland State University (CSU) after transferring from Lakeland Community College in January 2017. Programming and computers has been a huge part of his life from when he was 14. Some of the first programming languages used was Java, Visual Basic, and C Arduino. Later on during his undergraduate years he explored other languages such as Python, C#, ML, and Go. Programming caught his attention when he was playing a favorite video game and he wanted to disassemble and customize the game. He found the IDE that the original developers used when creating the game. This allowed him from a young age to freely mess with software and create ideas. Nowadays, programming is done in a real-world settings, with projects that have a more prominent impact. But this does not mean he doesnt have side projects. Some of the side projects include learning and starting blockchain technology and a Blockchain-based business which helped explain what and how it works.

Brian Salvoro is a senior Computer Science major attending at Cleveland State University (CSU). He has been attending Cleveland State University for one and half years. He first attended Cuyahoga Community College for the first couple of years of his collegiate education. He graduated from Cuyahoga Community College in Summer 2017 and achieved his Associates in Science which was a requirement for transfering to CSU. Since his attendance at CSU, he has taken classes such as Systems Programming, Operating System, Software Engineering, Language Processors, etc. Brian has introductory experiences in multiple programming languages such as C, JAVA, C#, Python through his homeworks and projects at school. He does a bits of programming out of school time mostly with the involvement with his hobby of playing games. He created mini scripts to reinforce his love for gaming. He is currently living in Parma Heights, Ohio, he is an only child, and would be the first person in the family to achieve a computer science degree when he graduates. Brian is expected to graduate and achieve his Bachelors of Science in Computer Science in the fall of 2019.