Melis Atun

CS 315 - Homework III

Go Language

- Nested subprogram definitions

```
func(melis string) {

      fmt.Println("Hi", melis, "I'm a function.")

   }("Melis")



   //func(melis string) is assigned to the variable funcVar in order to be
called

   funcVar := func(melis string) {

      fmt.Println("Hi", melis, "I'm a function assigned to a variable.")

   }

   funcVar("Melis")



   //output: "Hi melis I'm a function."
```

- Results of execution: In the go language, functions must be assigned to a variable in order to be called. Therefore, I assigned func(melis string) to the variable funcVar and called it with a string (“Melis”).

- Scope of local variables

```
//local variables cannot be seen by the functions outside of the function that
they have been declared

   //global variables can be seen from everywhere

   var myAge int = 21

   var myFavNum int = 8



   fmt.Printf("My age is %d\n", myAge)

   fmt.Printf("My favorite number is %d\n", myFavNum)

   //output: My age is 21

   //My favorite number is 8
```

- Results of explanation: In the go language, just like other languages, local variables cannot be seen by the functions outside of the function that they have been declared. However, global functions can be seen from everywhere.

- Parameter passing methods
  - **pass by value**

```
type Person struct {

   firstName string

   surname   string

}



func changeValue(myMother Person) {

   myMother.firstName = "Cigdem"

}
```

```
  //pass by value example

   //although the function changeValue above changes the name of the
person, there is no change on the name

   //this is because the changeValue function only changes the copy of
the name, not the name itself

   person := Person{

       firstName: "Melis",

       surname:   "Atun",

   }

   changeValue(person)

   fmt.Println(person)

   //output: {Melis Atun}
```

- Results of explanation: In pass by value method of parameter passing methods, the changeValue function of the above example changes only the copy of the name, not the actual name itself. Therefore, output is the original name which is not changed by the changeValue function.

- **pass by pointer (reference)**

```
type Person2 struct {

   firstName string

   surname   string

}
```

```
func changePointer(myFather *Person2) {
   myFather.firstName = "Ata"
}

person2 := Person2{
       firstName: "Melis",
       surname:   "Atun",
   }

   changePointer(&person2)
   fmt.Println(person2)
   //output: {Ata Atun}
```

- Results of explanation: In pass by pointer (reference) method of parameter passing methods, the changePointer function of the above example changes the name because changePointer function is called with &person2, which a reference. Hence, person2 in calling the function and person2 as a parameter in the changePointer function are two different pointers pointing to the same struct, which is the Person2 struct. Therefore, output is the changed name, which is Ata Atun, not the original name Melis Atun.

- Keyword and default parameters

- **keywords:**

  Go does not support keywords in subprograms by design.

- **default parameters:**

  Go does not support default function arguments by design.

- Closures

```
//first declare a variable

melis := 0



//assign a function to the variable

myFunc := func() int {

    melis += 1

    return melis

}



fmt.Println(myFunc())

fmt.Println(myFunc())

//output: 1

//2
```

- Results of explanation: In the go language, a specifical feature is provided. This feature is anonymous functions and they can form closures. Closures are also anonymous functions which can reference variables declared outside of the function itself. This is actually similar to accessing global variables. In the above example, even though the variable “melis” is not a parameter of the

function, it can be called in main because of the closure feature of the go language.

Evaluation of Golang

First of all, I think that Go is a very simple and minimalistic language and it is very easy to understand, learn, and start coding with. It is also very readable and writable because there are not many complicated things to learn about it like other programming languages, especially because the syntax is extremely simple to understand. Also, compilation is very fast thanks to the simplicity of the language. Hence, I think that these things make Go language a usable, readable, and writable language. Moreover, the garbage collection feature of Go language is also a very useful feature because collecting garbage manually can be extremely agonizing (like in C++). Last of all, I want to mention something that drew my attention. While I was writing my program, I noticed that all of the errors occurred as a compile-time error and this surprised me because I realized that some of those errors would be run-time errors in other languages. Therefore, I think that this is also a very good feature of Go language.

On the other hand, there are also some bad sides of Go language as well because the language lacks lots of features by design. For example, since overloading the functions is not allowed in the language, more code must be written compared to other languages and this means that more errors can occur.

My Learning Strategy

First of all, without starting to write any code snippets, I did a general research on the Go programming language. This allowed me to have a general information about the language's syntax, how to pass parameters to functions, how to declare variables, etc. and made me familiar with the

programming language in general. After doing this brief research, I started to research the specific five topics that we have in this homework. I tried to write code snippets for each topic rather than writing a whole program and this allowed me to learn the rules of the Go programming language better without mixing things and being drowned in errors later on. I tried to look into every resource that I can found so that I can have a better understanding of the specific rule that will be the answer of a question. However, I also encountered some unreliable resources that lack necessary information and have redundant information and I eliminated them. Later on, after writing code snippets for every question and commenting them properly with good explanations, I brought them together in a whole program and put the program in Visual Studio Code.

Before doing anything with my program, I made some arrangements in order to be able to run Golang code in Visual Studio Code: I downloaded the Go extension and made sure that every command line tool in my Macbook is up to date. After organizing the code I wrote and saving the program with .go extension, I solved the errors that I have encountered before running my program. It is very good that Go always gives compile-time errors. Also the explanations of the errors are also very explanatory and therefore easy to solve. After solving the errors, I ran my program and made sure that it works properly with the correct outputs. Then, I commented my code properly so that the TA can understand my program better without any question mark in their head.

After being completely sure that my code works properly without any errors and it is a clean code with proper comments and everything, I closed the program after saving it. Then, I checked my pdf report of the homework because I made some little changes on the code snippets since I resolved the errors and added some stuff in order to test my program better. After I made sure that my pdf report is proper as well, I moved to writing this essay.

References

Krunal, “Golang variables scope: What is the scope of variables in go,” *AppDividend*, 23-Aug-2021. [Online]. Available: https://appdividend.com/2020/01/29/scope-of-variables-in-golang-go-variables-scope/. [Accessed: 19-Dec-2021].

Sharbeargle, “Nested functions,” *Nested Functions · GoLang Notes*. [Online]. Available: https://sharbeargle.gitbooks.io/golang-notes/content/nested-functions.html. [Accessed: 19-Dec-2021].

“Closures in Golang,” *GeeksforGeeks*, 23-Mar-2020. [Online]. Available: https://www.geeksforgeeks.org/closures-in-golang/. [Accessed: 19-Dec-2021].

S. P. (GetTemplate.com), “Yury Pitsishin on software design, Big Data and DevOps,” *Pass by pointer vs pass by value in Go*, 01-Feb-2016. [Online]. Available: https://goinbigdata.com/golang-pass-by-pointer-vs-pass-by-value/. [Accessed: 19-Dec-2021].

Go - shichao's notes. [Online]. Available: https://notes.shichao.io/golang/. [Accessed: 19-Dec-2021].

B. Ž. (bojanz), “Bojan živanović,” *Optional function parameters in Go · bojanz.github.io*, 29-May-2020. [Online]. Available: https://bojanz.github.io/optional-parameters-go/. [Accessed: 19-Dec-2021].