CS315
Homework 3
Ata Seren
21901575

Design Issue 1: Nested subprogram definitions

Go supports nested subprogram but not in a way similar to Python. Outer subprogram "nested()" is declared normally. However, I needed to declare inner subprogram anonymously, initialized as a variable "inner" and called the subprogram via this variable which are features that Go allows [1].

Code:

```
func nested() {
     fmt.Println("This is the outer part of nested
subprogram.")

    inner := func() {
        fmt.Println("This is the inner part of nested
subprogram.")
    }
     inner()
}

func main() {
     fmt.Println("Design Issue 1")
     nested() //This function is supposed to print 2 different
statement in the subprogram and in the inner subprogram.
     fmt.Println()
}
```

Result:

```
Design Issue 1
This is the outer part of nested subprogram.
This is the inner part of nested subprogram.
```

There is the nested subprogram declaration and calling of the subprogram in the main() above. First, the print statement at the beginning of the nested() runs. Then, inner anonymous subprogram is declared, initialized as a variable and called. It printed a statement too. Thanks to the Go feature of implementing anonymous subprograms with a variable, nested subprograms can be declared [2] [3].

Design Issue 2: Scope of local variables

In Go, variables declared out of a subprogram are global and variables declared in a subprogram are local. To show this, I declared a global variable "gVal". Value of this variable, 1975, has been printed by a subprogram in the main() subprogram. Also, it has been printed in main() subprogram separately. To show the local variable effect, I declared gVal with value 2010 locally this time, in the main() subprogram. As I expected, print statement has printed this new local value.

Code:

```
var gVal = 1975

func scope(){
     fmt.Println("Value of global variable gVal in scope()
subprogram = ", gVal)
}

func main() {

     fmt.Println("Design Issue 2")
     fmt.Println("Value of global variable gVal in main() = ",
gVal)
     scope()
     var gVal = 2021 //Local declaration of global variable
gVal
     fmt.Println("Value of global variable gVal after declared
and be local in main() = ", gVal)
     fmt.Println()
}
```

Result:

```
Design Issue 2
Value of global variable gVal in main() =  1975
Value of global variable gVal in scope() subprogram =  1975
Value of global variable gVal after declared and be local in
main() =  2021
```

As it can be seen in result, global value 1975 is accessed in scope() and print statement in main(). Since variables declared in a subprogram are local, locally declared gVal with value 2021 is valid for statements after this declaration that use gVal [4].

Design Issue 3: Parameter passing methods

Go supports pass-by-value and pass-by-reference (they are named as call-by-value and call-by reference in Go). For pass-by-value, I used passByValue1(a int) and passByValue2(a int) int subprograms. By using them and a variable, I tested the case of subprograms with pass-by-value take the value inside and can change its data locally but when they terminate, changes on the variables don't last. For pass-by-reference, I used passByReference(b *int) and a pointer initialized in terms of Go language rules and pointed to an int value. Like the languages that use pointers such as C++, changes on the referenced pointer happened in the subprogram has changed the value of the variable at the outside of the function.

Code:

```
func passByValue1(a int) {
    a = 10
}

func passByValue2(a int) int {
    return a
}

func passByReference(b *int) {
    *b = 13
}

func main() {

     fmt.Println("Design Issue  3")
     var x int = 5
     fmt.Println("Before passByValue1(x) x = ", x)
     passByValue1(x)
     fmt.Println("After passByValue1(x) x = ", x) //No change
     after passByValue1(x)
     fmt.Println("Result of passByValue2(x) = ",
     passByValue2(x)) //Directly returns the given value
     fmt.Println()

     var y1 int = 98
     var y *int = &y1
     fmt.Println("Before passByReference(y) y = ", *y)
     passByReference(y)
     fmt.Println("After passByReference(y) y = ", *y) //Changes
     the value via pointer
     fmt.Println()
}
```

Result:

```
Design Issue 3
Before passByValue1(x) x =  5
After passByValue1(x) x =  5
Result of passByValue2(x) =  5

Before passByReference(y) y =  98
After passByReference(y) y =  13
```

In passByValue1 and passByValue2 subprograms, it can be seen that actual parameter has been copied to formal parameter and processes happen on formal parameter. Therefore, actual parameter initialized in main() has not been changed [5] [6].

In passByReference subprogram, an int pointer has been used. This pointer has been initialized according to the Go language rules [7]. In the subprogram, processes happen on the address stored in the pointer and therefore, actual data has been changed via this pointer.

Design Issue 4: Keyword and default parameters

In Go, keyword and default parameters are not supported [8]. However, they can be imitated by using structs and "variadic functions". To imitate keyword parameters, I declared a struct that works in a similar way to structs in other languages and used it as a parameter in useKeywordParameters subprogram. In this way, I can access to the variables in struct in any order while using it in the subprogram call in the main(). To imitate default parameters, I declared a variadic function which is a function with a "..." parameter that allows function to be invoked with zero or more arguments for that parameter [10]. The parameter can be accessed by using array notation. In defaultParameters, I wrote a code that initializes a default value to the first element if no parameters have been entered [9].

Code:

```
type keywordParameters struct {
  p1, p2, p3 int
}

func useKeywordParameters(ps keywordParameters) int {
  return ps.p1 + ps.p2 + ps.p3
}


func defaultParameters(arr ...int){
     var def int = 8
     if len(arr) > 0 {
          def = arr[0]
     }

     fmt.Println("Result of defaultParameters = ", def)
}

func main() {
     fmt.Println("Design Issue  4")
     fmt.Printf("Result of subprogram with keyword parameters:
")
     fmt.Println(useKeywordParameters(keywordParameters{p3: 6,
p1: 12})) //No order is needed
     fmt.Println("Result of subprogram with default parameters:
")
     defaultParameters() // Prints 8 as default
     defaultParameters(3) // Prints given value
     fmt.Println()
}
```

Result:

```
Design Issue 4
Result of subprogram with keyword parameters: 18
Result of subprogram with default parameters:
```

```
Result of defaultParameters =  8
Result of defaultParameters =  3
```

In the struct keywordParameters, I declared 3 variables. By using this struct as a parameter in subprogram useKeywordParameters, I can use p1, p2 and p3 variables as keyword parameters. In the subprogram defaultParameters, "…" parameter allows subprogram to get any amount of variable, including no parameter. If there won't be an actual parameter, subprogram detects it and inside, a predetermined value will be assigned to the element or elements of "…" [11] [12].

Design Issue 5: Closures

Go language support anonymous functions that can be initialized with a parameter. Such function allows us to form a closure. [13] I declared a multiplier function that returns an anonymous function. I used this return value to initialize the anonymous function with a variable in main() subprogram [14] [15].

Code:

```
func multiplier(m1 int) func(m2 int) int{
     return func(m2 int) int{
          return m1*m2
     }
}

func main(){

     fmt.Println("Design Issue 5")
     var multiply = multiplier(9)
     fmt.Println(multiply(2))
     fmt.Println(multiply(3)) //Closure has been created and
used
}
```

Result:

```
Design Issue 5
18
27
```

A nested subprogram has been declared similar to one in Design Issue 1. However, this time inner subprogram can be called out of the subprogram. Actually, this is the way to run the subprogram. I multiplied 2 variables using this closure.

Evaluation of Golang in terms of readability and writability of subprogram syntax

For the evaluation of Golang in terms of readability and writability, I want to make comparison with Python because in my opinion, Python is one of the most readable and writable languages which I used in my other homeworks. In terms of whole readability and writability of Golang, implementations require few additional steps, such as var and type declarations. Also declaring and initializing a variable can be done with several methods and it decreases both readability and writability. In terms of readability and writability of subprogram syntax, I believe that declaring and calling named and anonymous subprograms increase the writability because it allows us to implement functionalities in an easy way. Return type declarations are different than Python but don't harm readability and writability. By not supporting keyword and default parameters and mandatory usage of anonymous declaration of subprograms for nested subprograms decrease both readability and writability of Golang. However, syntax is in this way and mentioned features are not supported to protect the reliability of Golang.

My Learning Strategy

First of all, I visited the official website of Golang. I didn't check the documentation in a detailed way because from the past experiences with previous homeworks, I know that by using coding tutorial platforms such as W3Schools and GeeksForGeeks and coding forums such as StackOverflow, I learn the basics of a language in a better and easier way. This happens because of my learning strategy for almost every topic. I study the general points of a topic and then, I inspect the examples of the topic to see how the general points are implemented.

My first example is the one in the official online compiler of Golang, https://go.dev/play/. It didn't offer a lot, so I inspected the examples in GeeksForGeeks. Then, I read the relevant parts of our textbook about the mentioned design issues to refresh my knowledge about them. Then, I searched these design issues for Golang one by one. As the result of these searches, I mostly encountered 3 types of websites: forums, coding tutorial platforms and official documentations. First, I inspected websites to learn the mechanics of the examples for the design issues. Then, I inspected the forums to see more examples to have detailed knowledge about the issues. Finally, I wrote my code on my text editor and compiled it on the online compiler. While writing my code, I implemented the most basic version of the code that answers the design issue. Then, I added statements that will indicate that code gives a reliable answer for the design issue. I implemented the subprograms out of the main subprogram because subprogram declarations in main subprogram must be declared anonymously because of the syntax of Golang and I'm not familiar with using anonymous functions a lot. During my experiments, I tried to use different variables from each other to understand if my code works properly.

For the first issue, I mostly used forums because writing nested subprograms in Golang is different that Python and people explained the difference and syntax for Golang. These forums helped me a lot to understand logic behind the Golang's nested function.

For the second issue, I simply used coding tutorial platform because I know the scopes thanks to the lectures and Golang has few rules for the scopes in subprogram. So, it was easy to write codes that indicate scopes of local variables.

For the third issue, I used both forums and coding tutorial platforms. I already knew the pass-by-value and pass-by-reference that is supported by Golang. So, I simply showed the mechanism of pass-by-value with a simple code first. Then, I declared a pointer in terms of Golang syntax. I used this pointer in a subprogram and showed the mechanics of pass-by-reference.

For the fourth issue, I additionally used official documentation to learn what "variadic" functions are. I have seen this term in forum posts that is about how Golang doesn't support keyword and default parameters. I used this method to imitate these features in Golang code.

For the fifth issue, I studied the relevant topic in the textbook. Then, I inspected some code examples. In these examples, I learned another feature of anonymously declared functions.

REFERENCES:

[1] https://stackoverflow.com/questions/28252117/golang-nested-class-inside-function

[2] https://medium.com/swlh/how-to-write-a-nested-function-in-go-3338c56526f3

[3] https://sharbeargle.gitbooks.io/golang-notes/content/nested-functions.html

[4] https://stackoverflow.com/questions/2032149/optional-parameters-in-go

[5] https://www.geeksforgeeks.org/function-arguments-in-golang/

[6] https://stackoverflow.com/questions/47486606/please-explain-if-golang-types-pass-by-value

[7] https://www.geeksforgeeks.org/pointers-in-golang/

[8] https://stackoverflow.com/questions/2032149/optional-parameters-in-go

[9] https://forum.golangbridge.org/t/how-to-give-default-value-to-function-parameters/18099

[10] https://stackoverflow.com/questions/10128771/what-does-mean-when-next-to-a-parameter-in-a-go-function-declaration

[11] https://go.dev/ref/spec#Function_types

[12] https://stackoverflow.com/questions/23447217/go-named-arguments

[13] https://www.geeksforgeeks.org/closures-in-golang/

[14] https://go.dev/tour/moretypes/25

[15] https://gobyexample.com/closures