Distributed Communication 1st practice

Li Jianhao
lijianhao288@hotmail.com

1 Basics

1.1 Go download and install

https://golang.org/doc/install

Run the command **go version** to check if the install is successful.

1.2 Hello world

Inside the folder you like, create a file named "hello.go". Type in the following code. Use command **go run hello.go** to run the program.

```
package main
import "fmt"
func main() {
    fmt.Println("Hello␣World")
}
```

Listing 1: Hello World

```
$ go run hello.go
Hello World
```

Notes: Only the package named "main" can be executed by the command **go run**. Inside the package named "main", there must be a function named "main" as the entry point of the program.

1.3 imports

```
import "fmt"
import "strconv"
```

Listing 2: Import ver.1

```
import (
    "fmt"
    "strconv"
)
```

Listing 3: Import ver.2

1.4 Constants. Operators

Syntax: const <Name> = <Value>

```
package main
import "fmt"

const University = "ELTE"
func main() {
    const threshold = 12
    fmt.Println(University+" Informatics")
    fmt.Println(threshold+1)
}
```

Listing 4: Constants declaration

```
ELTE Informatics
13
```

Other operators:

||, or. &&, and.

==, equal. !=, not equal.

<, less. <=, less or equal. >, greater. >=, greater or equal.

+, addition. -, subtraction. *, multiplication. /, division. %, modulus (get remainder).

1.5 Variables

Syntax: var <Name(s)> <Type>

```
package main
import "fmt"
var x, y, z bool              // package level variable
func main() {
    var i int                // function level variable
    fmt.Println(i, x, y, z)
}
```

Listing 5: Variables

```
0 false false false
```

Syntax: var <Name(s)> (<Type>) = <Value(s)>

```
package main
import "fmt"
var i = 1
var j int = 2
func main() {
    var x, y = true, "hello"
    fmt.Println(i, j, x, y)
}
```

Listing 6: Variables with initializers

```
1 2 true hello
```

Syntax: <Name(s)> := <Value(s)>

```
package main
import "fmt"
func main() {
    var i = 1
    j := 2       // short assignment
    fmt.Println(i, j)
}
```

Listing 7: Short variable declarations

```
1 2
```

1.6 Functions. If. Error

Syntax: func <Name>(<Parameters and their types>) (<Return types>) {<Function body> }
Syntax: if <condition> { } else { }

```
package main
import (
    "fmt"
    "errors"
)

func main() {
    fmt.Println(multiplicate(3, 4))

    p, e := multiplicateWithError(3,4,true)
    fmt.Println(p,e)
    p, e = multiplicateWithError(3,4,false)
    fmt.Println(p,e)
}

```

```
func multiplicate(x int, y int) int {
    return x * y
}

func multiplicateWithError (x int, y int, b bool) (int, error){
    if b {
        return 0, errors.New("special␣error")
    } else {
        return x*y, nil
    }
}
```

Listing 8: Function

```
12
0 special error
12 <nil>
```

1.7 Exported name. Package

The index of the packages of Go: https://golang.org/pkg/
The strconv package: https://golang.org/pkg/strconv/
The strings package: https://golang.org/pkg/strings/
Some of the function signature:

func Atoi(s string) (int, error)
func Itoa(i int) string
func Contains(s, substr string) bool

```
package main

import (
    "fmt"
    "strconv"
    "strings"
)

func main() {
    fmt.Println("strconv-------")
    fmt.Println(strconv.IntSize)
    s := strconv.Itoa(1)
    fmt.Println(s)
    i, err := strconv.Atoi("2")
    if err != nil {
        panic(err)
    }
    fmt.Println(i)

    fmt.Println("strings-------")
    fmt.Println("ToLower:␣"+strings.ToLower("APPLE"))
    fmt.Println("ToUpper:␣"+strings.ToUpper("apple"))
    fmt.Println(strings.Contains("apple","pp"))
    fmt.Println(strings.Count("apple","p"))
```

```
    fmt.Println(strings.Split("a,p,p,l,e",","))
}
```

Listing 9: Exported names

```
strconv-------
64
1
2
strings-------
ToLower: apple
ToUpper: APPLE
true
2
[a p p l e]
```

2 Practice

2.1 p1

Define a function named **inc** which increases its integer parameter by one and return the result.

In the main function, call the**inc** and pass it an argument 3. The result is stored in a variable named **result1**. Print out the **result1**.

2.2 p2

Define a function named **isEven** which checks whether its integer parameter is even. return the bool result.

In the main function, call the**isEven** and pass it an argument 3. The result is stored in a variable named **result2**. Print out the **result2**.

2.3 p3

Define a function named **devides** which takes two parameters, and checks whether the first integer parameter divides the second. return the bool result.

In the main function, call the**devides** and pass it arguments 3 and 9. The result is stored in a variable named **result3**. Print out the **result3**.

2.4 p4

Define a function named **area** which takes two parameters, and calculates the area of a rectangle using two parameters. return the int result.

In the main function, call the**are** and pass it arguments 3 and 4. The result is stored in a variable named **result4**. Print out the **result4**.

2.5 p5

Define a function named **stringAdd** which takes two string parameters. Convert them to int and return the sum of them. return the int result and the possible error.

Try to convert both strings to int. If the first convert or the second convert does not succeed (possible error not nil), return -1 and an error "Failed to convert". If the convert both success, return the sum and nil.

In the main function, call the**stringAdd** and pass it an arguments "3" and "4". The result is stored in a variable named **result5**. Print out the **result5** and the possible error. Call the**stringAdd** and pass it an arguments "3" and "a". The result is stored in a variable named **result6**. Print out the **result6** and the possible error.

2.6 p6

Define a function named **isUpper** which takes a string, and checks whether the characters of this string are all upper case. return the bool result.

The function should use the method **ToUpper** in the package strings.

In the main function, call the**isUpper** two times. First time pass it "apple". The second time passes "APPLE". print out the results.