# Distributed Communication 3rd practice

Li Jianhao
lijianhao288@hotmail.com

# 1 Basics

## 1.1 Slice(continue)

Syntax: <SliceName>[x]
Syntax: <SliceName>[a:b]
Syntax: <SliceName>[x] = <Value>
Syntax: <SliceName> = append(<SliceName1>, <SliceName2>...)

Review (append elements):
Syntax: <SliceName> = append(<SliceName>, <NewElement(s)>)

```
package main
import "fmt"
func main() {
    animals := []string{
        "0_dog",
        "1_cat",
        "2_bird",
        "3_lion",
        "4_panda",
        "5_tiger",
        "6_wolf",
    }

    fmt.Println(animals[3])

    fmt.Println(animals[:3])

    fmt.Println(animals[3:])

    fmt.Println(animals[2:4])

    animals[3] = "SSS"
    fmt.Println(animals)

    result := append(animals[:2], animals[5:]...)
    fmt.Println(result)
}
```

Listing 1: Slice

```
3_lion
[0_dog 1_cat 2_bird]
[3_lion 4_panda 5_tiger 6_wolf]
[2_bird 3_lion]
[0_dog 1_cat 2_bird SSS 4_panda 5_tiger 6_wolf]
[0_dog 1_cat 5_tiger 6_wolf]
```

## 1.2 Struct

Syntax:

type $<StructName>$ struct {
$<FieldName1><FieldType1>$
$<FieldName2><FieldType2>$
$<...>$
$<FieldNameN><FieldTypeN>$
}

Syntax: $<TypedValueName> := <StructName>\{<FieldValue(s)>\}$

Syntax: $<TypedValueName>.<FieldName2>$

Syntax: $<TypedValueName>.<FieldName2> = <Value>$

```
package main
import "fmt"

type student struct {
    name   string
    id string
}

func main() {
    adam := student{"Adam",  "abcdef"}
    fmt.Println(adam)
    fmt.Println(adam.name)

    adam.id = "nnnnn"
    fmt.Println(adam)
}
```

Listing 2: Struct

```
{Adam abcdef}
Adam
{Adam nnnnn}
```

## 1.3 Pointer

Syntax: *< $Type$ >
Syntax: *< $Pointer$ >
Syntax: &< $Value$ >

```
package main
import "fmt"
type student struct {
    name   string
    id string
}
func main() {
    adam := student{"Adam", "abcdef"}
    fmt.Println(adam)

    modifyStudentName(&adam, "Levi")
    fmt.Println(adam)

    animals := []string{
        "dog",
        "lion",
        "panda",
    }
    fmt.Println(animals)

    modifyFirstElement(animals, "cat")
    fmt.Println(animals)
}

func modifyStudentName(pointerToStudent *student, newName string) {
    (*pointerToStudent).name = newName
}

func modifyFirstElement(animals []string, newFirstElement string) {
    animals[0] = newFirstElement
}
```

Listing 3: Pointer

```
{Adam abcdef}
{Levi abcdef}
[dog lion panda]
[cat lion panda]
```

## 1.4 Method

Syntax: func (<ReceiverName> <ReceiverType>) <Name>(<Parameters and their types>) (<Return types>) {<Function body> }
Syntax: < $TypedValueName$ >.< $method$ >()
Syntax: type < $TypeName$ > < $Type'Type$ >

```
package main
import "fmt"

type student struct {
    name   string
    id string
}

type neptun string

func main() {
    adam := student{"Adam", "abcdef"}
    fmt.Println(adam.getNameR())
    fmt.Println(getNameA(adam))

    var aNeptun neptun = "asdf"
    aNeptun.printId()
}

func (s student) getNameR() string {
    return s.name
}

func getNameA(s student) string {
    return s.name
}

func (i neptun) printId(){
    fmt.Println(i)
}
```

Listing 4: Receiver

```
Adam
Adam
asdf
```

## 1.5 Shorthand

https://go.dev/ref/spec#Calls

"If x is addressable and &x's method set contains m, x.m() is shorthand for (&x).m()"

https://go.dev/ref/spec#Selectors

"As an exception, if the type of x is a defined pointer type and (*x).f is a valid selector expression denoting a field (but not a method), x.f is shorthand for (*x).f."

```
package main

```

```
import "fmt"

func main() {
    adam := student{"Adam", "abcdef"}
    fmt.Println(adam)
    (&adam).setName1("AAAA")
    fmt.Println(adam)
    adam.setName2("BBBB")
    fmt.Println(adam)
}

type student struct {
    name   string
    id string
}

func (pointerToStudent *student) setName1(newName string) {
    (*pointerToStudent).name = newName
}

func (pointerToStudent *student) setName2(newName string) {
    pointerToStudent.name = newName
}
```

Listing 5: Pointer

```
{Adam abcdef}
{AAAA abcdef}
{BBBB abcdef}
```

## 1.6 Map

Syntax: $< MapName >:=$ make(map[$< KeyType >$]$< ValueType >$)
Syntax: $< MapName >[< Key >] = < Value >$
Syntax: delete ($< MapName >, < Key >$)
Syntax: $< ValueName >, < IsKeyExist > := < MapName >[< Key >]$
Syntax:

for <KeyName>, <ValueName> := range <MapName> {
}

Review (for range loop of slice):
for <IndexName>, <ElementName> := range <SliceName> {
}

```
package main

import "fmt"
```

```
                                                                    4
func main() {                                                       5
    neptunMap := make(map[string]string)                            6
                                                                    7
    neptunMap["AABBCC"] = "Adam"                                    8
    neptunMap["CCBBAA"] = "Ben"                                     9
    neptunMap["BBAACC"] = "Ada"                                     10
    fmt.Println(neptunMap)                                          11
    fmt.Println(len(neptunMap))                                     12
                                                                    13
    neptunMap["BBAACC"]  = "CCC"                                    14
    fmt.Println(neptunMap)                                          15
                                                                    16
    delete(neptunMap, "BBAACC")                                     17
    fmt.Println(neptunMap)                                          18
                                                                    19
    v1, ok1 := neptunMap["CCBBAA"]                                  20
    fmt.Println(v1, ok1)                                            21
    v2, ok2 := neptunMap["AAAAAA"]                                  22
    fmt.Println(v2, ok2)                                            23
                                                                    24
    for k,v :=range neptunMap {                                     25
        fmt.Println(k,v)                                            26
    }                                                               27
                                                                    28
    for _,v :=range neptunMap {                                     29
        fmt.Println(v)                                              30
    }                                                               31
}                                                                   32
```

Listing 6: Map

```
map[AABBCC:Adam BBAACC:Ada CCBBAA:Ben]                              1
3                                                                   2
map[AABBCC:Adam BBAACC:CCC CCBBAA:Ben]                              3
map[AABBCC:Adam CCBBAA:Ben]                                         4
Ben true                                                            5
 false                                                              6
AABBCC Adam                                                         7
CCBBAA Ben                                                          8
Ben                                                                 9
Adam                                                                10
```

# 2 Practice

## 2.1 p1

Create a struct **animal**. Its fields: specie (type string), name (type string).
In the main function, create its typed value **dog1** with field value "dog" and "One". Print out the dog1. Print out dog1's name. Modify dog1's name to "Two". Print out dog1 again.

## 2.2 p2

Create a method **setName** of animal, which takes the pointer to the animal as the receiver (the receiver name is ap). It takes a string n as the parameter. In the function, we change the animal's name to n (Change the original value).
In the main function, use **setName** to change dog1's name to "Three". Print out the dog1.

## 2.3 p3

Create a method **move** of animal, which takes the animal as the receiver (the receiver name is a). It prints out the a' specie, name, and a string "move". Like "dog One move".

## 2.4 p4

Create a slice of animal. Its name is **animalSlice**. It has five initial elements: animal "dog" "One", animal "dog" "Two", animal "cat" "Three", animal "cat" "Four", animal "bird" "Five".
Print out the slice.
Print out **animalSlice**' third elements.
Pirnt out **animalSlice**' second elements till the fourth elements.

## 2.5 p5

Create a map **animalMap** which maps the string to animal. It has elements: "A" -> animal "dog" "One", "B" -> animal "dog" "Two", "C" -> animal "cat" "Three", "D" -> animal "bird" "Four".
Print out the **animalMap**. Print out the length of the **animalMap**. Modify the element with key "C" to animal "bird" "Three". Print out the **animalMap** again.

## 2.6 p6

Try to get the mapped value of the key "B" in the**animalMap**. If it is existing, print out the "B" and its mapped value. And delete this element from the**animalMap**. Print out the **animalMap** again.