

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                                1
import "fmt"                                2
func main() {                                3
    animals := []string{                     4
        "0_dog",                             5
        "1_cat",                             6
        "2_bird",                           7
        "3_lion",                           8
        "4_panda",                          9
        "5_tiger",                         10
        "6_wolf",                          11
    }                                         12
                                           13
    fmt.Println(animals[3])                  14
                                           15
    fmt.Println(animals[:3])                 16
                                           17
    fmt.Println(animals[3:])                 18
                                           19
    fmt.Println(animals[2:4])                20
                                           21
    animals[3] = "SSS"                      22
    fmt.Println(animals)                    23
                                           24
    result := append(animals[:2], animals[5:]...) 25
    fmt.Println(result)                     26
}                                             27
```

Listing 1: Slice

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

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	1
import "fmt"	2
	3
type student struct {	4
name string	5
id string	6
}	7
	8
func main() {	9
adam := student{"Adam", "abcdef"}	10
fmt.Println(adam)	11
fmt.Println(adam.name)	12
	13
adam.id = "nnnnn"	14
fmt.Println(adam)	15
}	16

Listing 2: Struct

{Adam abcdef}	1
Adam	2
{Adam nnnnn}	3

1.3 Pointer

Syntax: $*\langle Type \rangle$

Syntax: $*\langle Pointer \rangle$

Syntax: $\&\langle Value \rangle$

```
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 ($\langle ReceiverName \rangle$ $\langle ReceiverType \rangle$) $\langle Name \rangle$ ($\langle Parameters$
and their types>) ($\langle Return types \rangle$) { $\langle Function body \rangle$ }

Syntax: $\langle TypedValueName \rangle.\langle method \rangle$

Syntax: type $\langle TypeName \rangle$ $\langle Type'Type \rangle$

package main	1
import "fmt"	2
	3
type student struct {	4
name string	5
id string	6
}	7
	8
type neptun string	9
	10
func main() {	11
adam := student{"Adam", "abcdef"}	12
fmt.Println(adam.getNameR())	13
fmt.Println(getNameA(adam))	14
	15
var aNeptun neptun = "asdf"	16
aNeptun.printId()	17
}	18
	19
func (s student) getNameR() string {	20
return s.name	21
}	22
	23
func getNameA(s student) string {	24
return s.name	25
}	26
	27
func (i neptun) printId(){	28
fmt.Println(i)	29
}	30

Listing 4: Receiver

Adam	1
Adam	2
asdf	3

1.5 Map

Syntax: $\langle MapName \rangle := \text{make}(\text{map}[\langle KeyType \rangle] \langle ValueType \rangle)$

Syntax: $\langle MapName \rangle[\langle Key \rangle] = \langle Value \rangle$

Syntax: $\text{delete}(\langle MapName \rangle, \langle Key \rangle)$

Syntax: $\langle ValueName \rangle, \langle IsKeyExist \rangle := \langle MapName \rangle[\langle Key \rangle]$

Syntax:

for $\langle KeyName \rangle, \langle ValueName \rangle := \text{range } \langle MapName \rangle$ {
}

Review (for range loop of slice):

```
for <IndexName>, <ElementName> := range <SliceName> {  
    }
```

```
package main 1  
  
import "fmt" 2  
3  
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 5: Map

```
map[AABBCC:Adam BBAACC:Ada CCBBAABen] 1  
3 2  
map[AABBCC:Adam BBAACC:CCC CCBBAABen] 3  
map[AABBCC:Adam CCBBAABen] 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.

Print 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.