分布式计算实验-作业 1

1. 旋转切片（20 分）

描述：编写一个函数 Rotate(slice []int, k int) []int，将整数切片 slice 向右旋转 k 位，返回旋转后的切片。

要求：禁止使用额外的切片或辅助数组。

示例：Rotate([]int{1, 2, 3, 4, 5}, 2) // 返回 [4, 5, 1, 2, 3]

```go
package main
import "fmt"
// Rotate 旋转切片
func Rotate(slice []int, k int) []int {
    n := len(slice)
    k = k % n // 防止 k 超过 n
    // 反转整个切片
    reverse(slice, 0, n-1)
    // 反转前 k 个元素
    reverse(slice, 0, k-1)
    // 反转剩余的元素
    reverse(slice, k, n-1)

    return slice
}

// reverse 反转切片的一部分
func reverse(slice []int, start, end int) {
    for start < end {
        slice[start], slice[end] = slice[end], slice[start]
        start++
        end--
    }
}

func main() {
    arr := []int{1, 2, 3, 4, 5}
    k := 2
    result := Rotate(arr, k)
    fmt.Println(result)
}

```

```
haiden@haiden-virtual-machine:~/go$ go run 1.go
[4 5 1 2 3]
haiden@haiden-virtual-machine:~/go$
```

2. 统计单词出现次数（20 分）

描述：编写一个函数 WordCount(text string) map[string]int，接收一段文本 text，统计每个单词出现的次数，并返回统计结果。

要求：统计时不区分大小写，忽略标点符号。

示例：WordCount("Hello, world! Hello!") // 返回 map[string]int{"hello": 2, "world": 1}

```go
package main

import (
    "fmt"
    "regexp"
    "strings"
)

func WordCount(text string) map[string]int {
    // 使用正则表达式去除标点符号
    re := regexp.MustCompile(`[^\w\s]`)
    cleanedText := re.ReplaceAllString(text, "")

    // 全部转换为小写，避免区分大小写
    cleanedText = strings.ToLower(cleanedText)

    // 按空格分割字符串为单词
    words := strings.Fields(cleanedText)

    // 创建一个map来存储单词出现的次数
    wordCount := make(map[string]int)

    // 统计每个单词的出现次数
    for _, word := range words {
        wordCount[word]++
    }

    return wordCount
}

func main() {
    text := "Hello, world! Hello!"
    result := WordCount(text)
    fmt.Println(result) // 输出: map[hello:2 world:1]
}

```

```
haiden@haiden-virtual-machine:~/go$ go run 2.go
map[hello:2 world:1]
```

3. 查找键和值的最值 (20 分)

题目描述：实现一个函数 FindMaxKeyAndValue(m map[string]int) (string, int)，返回 map 中键对应值最大的键和其值。

要求：若存在多个键对应的值相同且为最大值，则返回其中字母序最小的键。

示例：FindMaxKeyAndValue(map[string]int{"a": 10, "b": 20, "c": 20}) // 返回 "b", 20

```go
package main

import (
    "fmt"
)

func FindMaxKeyAndValue(m map[string]int) (string, int) {
    var maxKey string
    maxValue := -1 // 初始值设置为-1，假设值均为非负数

    for key, value := range m {
        // 找到更大的值或相同值但字母序更小的键
        if value > maxValue || (value == maxValue && key < maxKey) {
            maxValue = value
            maxKey = key
        }
    }

    return maxKey, maxValue
}

func main() {
    resultKey, resultValue := FindMaxKeyAndValue(map[string]int{"a": 10, "b": 20, "c": 20})
    fmt.Printf("Max Key: %s, Max Value: %d\n", resultKey, resultValue)
}
```

```
haiden@haiden-virtual-machine:~/go$ go run 3.go
Max Key: b, Max Value: 20
```

4. Go 语言实现面向对象功能（40 分）

定义一个 Personal-Salary 接口， 它包括两个方法 OutputInfo() 与 OutputWage(), ，分别输出人员信息与实发工资。然后，定义 4 个结构体 Staff（员工）、Saleman（销售员），Manager（经理）和 SaleManager（销售经理），它们分别实现了 Personal-Salary 接口的 OutputInfo() 与 OutputWage()方法。

具体要求：定义 Staff 结构体，基于 Staff 采用结构体嵌套的方式实现 Saleman（销售员）结构体和 Manager（经理）结构体，再由 Saleman 和 Manager 实现结构体 SaleManager（销售经理）

（1） Staff 的成员变量：编号(num)、姓名(name)、出勤率(rateOfAttend)、基本工资(basicSal)和奖金(prize)。

（2） Saleman 在 staff 基础上还包含：销售员提成比例(deductRate)和个人销售额(personAmount)。

（3） Manager 中在 staff 基础上还包含：经理提成比例(totalDeductRate)和总销售额(totalAmount)。

（4） SaleManager 包含 Saleman 和 Manager 的所有成员变量。

各类人员的 salary 计算公式如下：

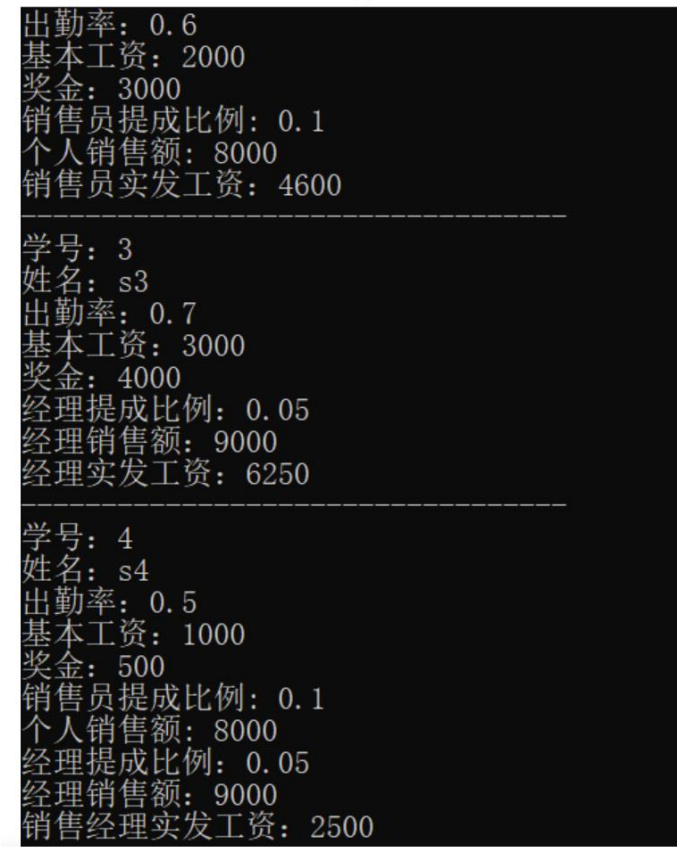
staff 实发工资 = 基本工资 + 奖金 * 出勤率

Saleman 实发工资 = 基本工资 + 奖金 * 出勤率 + 个人销售额 * 销售员提成比例

Manager 实发工资 = 基本工资 + 奖金 * 出勤率 + 总销售额 * 经理提成比例

SaleManager 实发工资 =基本工资 + 奖金 * 出勤率 + 个人销售额 * 销售员提成比例+ 总销售额 * 经理提成比例

请按照下面的数据情况分别输出对应的人员信息与实发工资。



```go
package main

import (
	"fmt"
)

// 定义 PersonalSalary 接口，包含 OutputInfo() 和 OutputWage() 方法
type PersonalSalary interface {
	OutputInfo()
	OutputWage()
}

// Staff 结构体，包含员工的基本信息
type Staff struct {
	num         string  // 编号
	name        string  // 姓名
	rateOfAttend float64 // 出勤率
	basicSal    float64 // 基本工资
	prize       float64 // 奖金
}

// Saleman 结构体，嵌套 Staff，并添加销售员的额外字段
type Saleman struct {
	Staff
	deductRate  float64 // 销售员提成比例
	personAmount float64 // 个人销售额
}

// Manager 结构体，嵌套 Staff，并添加经理的额外字段
type Manager struct {
	Staff
	totalDeductRate float64 // 经理提成比例
	totalAmount     float64 // 总销售额
}

// SaleManager 结构体，同时包含 Saleman 和 Manager 的字段
type SaleManager struct {
	Saleman
	Manager
}

// Staff 的 OutputInfo 方法实现，输出员工的基本信息
func (s Staff) OutputInfo() {
	fmt.Printf("编号: %s\n姓名: %s\n出勤率: %.1f\n基本工资: %.1f\n奖金: %.1f\n",
		s.num, s.name, s.rateOfAttend, s.basicSal, s.prize)
```

```go
}

// Staff 的 OutputWage 方法实现，计算并输出员工的实发工资
func (s Staff) OutputWage() {
        salary := s.basicSal + s.prize * s.rateOfAttend
        fmt.Printf("员工实发工资: %.1f\n", salary)
}

// Saleman 的 OutputInfo 方法实现，输出销售员的基本信息
func (s Saleman) OutputInfo() {
        s.Staff.OutputInfo()
        fmt.Printf("销售员提成比例: %.2f\n个人销售额: %.1f\n", s.deductRate, s.personAmount)
}

// Saleman 的 OutputWage 方法实现，计算并输出销售员的实发工资
func (s Saleman) OutputWage() {
        salary := s.basicSal + s.prize * s.rateOfAttend + s.personAmount * s.deductRate
        fmt.Printf("销售员实发工资: %.1f\n", salary)
}

// Manager 的 OutputInfo 方法实现，输出经理的基本信息
func (m Manager) OutputInfo() {
        m.Staff.OutputInfo()
        fmt.Printf("经理提成比例: %.2f\n经理总销售额: %.1f\n", m.totalDeductRate, m.totalAmount)
}

// Manager 的 OutputWage 方法实现，计算并输出经理的实发工资
func (m Manager) OutputWage() {
        salary := m.basicSal + m.prize * m.rateOfAttend + m.totalAmount * m.totalDeductRate
        fmt.Printf("经理实发工资: %.1f\n", salary)
}

// SaleManager 的 OutputInfo 方法实现，输出销售经理的全部信息
func (sm SaleManager) OutputInfo() {
        sm.Saleman.Staff.OutputInfo() // 显式调用 Saleman 的 Staff
        fmt.Printf("销售员提成比例: %.2f\n个人销售额: %.1f\n", sm.Saleman.deductRate, sm.Saleman.personAmount)
        fmt.Printf("经理提成比例: %.2f\n经理总销售额: %.1f\n", sm.Manager.totalDeductRate, sm.Manager.totalAmount)
}

// SaleManager 的 OutputWage 方法实现，计算并输出销售经理的实发工资
func (sm SaleManager) OutputWage() {
        // 使用 Saleman 和 Manager 的字段计算工资
        salary := sm.Saleman.basicSal + sm.Saleman.prize * sm.Saleman.rateOfAttend +
                sm.Saleman.personAmount * sm.Saleman.deductRate +
                sm.Manager.totalAmount * sm.Manager.totalDeductRate
        fmt.Printf("销售经理实发工资: %.1f\n", salary)
}

func main() {
        // 从图片中获取数据
        staff1 := Staff{"1", "s1", 0.6, 2000, 3000}
        staff2 := Staff{"2", "s2", 0.7, 3000, 4000}
        staff3 := Staff{"3", "s3", 0.5, 1000, 500}

        saleman1 := Saleman{staff1, 0.1, 8000}
        manager1 := Manager{staff2, 0.05, 9000}
        saleManager1 := SaleManager{Saleman: Saleman{staff3, 0.1, 8000}, Manager: Manager{staff3, 0.05, 9000}}

        // 输出信息和工资
        saleman1.OutputInfo()
        saleman1.OutputWage()
        fmt.Println("------------------------------------")
        manager1.OutputInfo()
        manager1.OutputWage()
        fmt.Println("------------------------------------")
        saleManager1.OutputInfo()
        saleManager1.OutputWage()
}
```

```
haiden@haiden-virtual-machine:~/go$ go run 4.go
编号: 1
姓名: s1
出勤率: 0.6
基本工资: 2000.0
奖金: 3000.0
销售员提成比例: 0.10
个人销售额: 8000.0
销售员实发工资: 4600.0
----------------------------------
编号: 2
姓名: s2
出勤率: 0.7
基本工资: 3000.0
奖金: 4000.0
经理提成比例: 0.05
经理总销售额: 9000.0
经理实发工资: 6250.0
----------------------------------
编号: 3
姓名: s3
出勤率: 0.5
基本工资: 1000.0
奖金: 500.0
销售员提成比例: 0.10
个人销售额: 8000.0
经理提成比例: 0.05
经理总销售额: 9000.0
销售经理实发工资: 2500.0
haiden@haiden-virtual-machine:~/go$
```