

```

package main

import "fmt"

// 剪子布游戏
//
// 现在有两个人在玩石头剪子布游戏，请你判断最后谁赢了。
// 用 R 代表石头，S 代表剪子，P 代表布。
// 输入格式：
//
// 输入的第一行是一个整数 t ，表示测试样例的数目。
// 每组输入样例的第一行是一个整数 n ，表示游戏次数。
// 接下来 n 行，每行由两个字母组成，两个字母之间用一个空格分隔，这些字母只会是 R, S 或 P。
// 第一个字母表示 Player1 的选择，第二个字母表示 Player2 的选择。
// 输出格式：
//
// 对于每组输入样例，输出获胜方的名字（Player1 或 Player2），
// 如果平均，则输出 TIE。
// 输入样例：
//
// 3
// R P
// S R
// P P
// 输出样例：
//
// Player2
// Player2
// TIE
func main() {
    /*var m int
    fmt.Scanln(&m)
    var player1_win = []int{}
    var player2_win = []int{}
    player1_win = make([]int, m)
    player2_win = make([]int, m)

    for i := 0; i < m; i++ {
        var c1, c2 string
        fmt.Scanln(&c1, &c2)
        switch c1 + c2 {
        case "RS", "SP", "PR":
            player1_win[i]++

```

```

case "SR", "PS", "RP":
    player2_win[i]++

}

for i := 0; i < m; i++ {
    if player1_win[1] < player2_win[1] {
        fmt.Println("Player2")
    } else if player1_win[i] > player2_win[i] {
        fmt.Println("Player1")
    } else {
        fmt.Println("TIE")
    }
}

}

}

*/

var n int
fmt.Scanln(&n)

var player1_win = []int{}
var player2_win = []int{}
player1_win = make([]int, n)
player2_win = make([]int, n)

for i := 0; i < n; i++ {
    var player1_str string
    var player2_str string
    fmt.Scanln(&player1_str, &player2_str)

    switch {
    case player1_str == "R" && player2_str == "S":
        player1_win[i]++
    case player1_str == "S" && player2_str == "P":
        player1_win[i]++
    case player1_str == "P" && player2_str == "R":
        player1_win[i]++
    case player2_str == "R" && player1_str == "S":
        player2_win[i]++
    case player2_str == "S" && player1_str == "P":
        player2_win[i]++
    case player2_str == "P" && player1_str == "R":
        player2_win[i]++
    }
}

```

```

}

}

for i := 0; i < n; i++ {
if player1_win[i] > player2_win[i] {
fmt.Println("Player1")
} else if player2_win[i] > player1_win[i] {
fmt.Println("Player2")
} else {
fmt.Println("TIE")
}
}

}

```

```

package main

```

```

import "fmt"

```

```

func gcd(n, m int) int {
for m != 0 {
n, m = m, n%m
}
return n
}

func lcm(n, m int) int {
return m * n / gcd(n, m)
}

func main() {
var n int
fmt.Print("请输入第一个正整数 n: ")
fmt.Scanln(&n)
var m int
fmt.Print("请输入第二个正整数 m: ")
fmt.Scanln(&m)
fmt.Printf("最大公约数为: %d\n", gcd(n, m))
fmt.Printf("最小公倍数数为: %d\n", lcm(n, m))

}

```

```

//package main
//
//import "fmt"
//
//折半查找
//
//输入 n 值(1<n<1000)、n 个非降序排列的整数以及要查找的数 x，使用二分查找算法查找 x，输出 x 所在的下标 (0~n-1) 及比较次数。 若 x 不存在，输出 -1 和比较次数。
//输入格式：
//
//输入格式：
//第一行是数组的个数 n
//第二行开始是输入数组的元素
//最后一行是需要查找的元素
//输出格式：
//
//输出 x 所在的下标 (0~n-1) 及比较次数。若 x 不存在，输出-1 和比较次数。
//输入样例：
//
//4
//1
//2
//3
//4
//1
//输出样例：
//
//0
//2
//func main() {
// var n int
// fmt.Scan(&n)
// var slice []int = make([]int, n, 2*n)
// for i := 0; i < len(slice); i++ {
// fmt.Scan(&slice[i])
// }
// var x int
// fmt.Scan(&x)
// left, right := 0, n-1
// cnt := 0 // 统计比较次数
// for left <= right {
// mid := (left + right) / 2
// cnt++

```

```

// if slice[mid] == x {
// fmt.Printf("%d\n%d", mid, cnt)
// return
// } else if slice[mid] < x {
// left = mid + 1
// } else {
// right = mid - 1
// }
// }
// fmt.Printf("-1\n%d", cnt)

```

```

package main

```

```

import (
    "fmt"
)

```

```

func main() {
    var n, target, mid, cnt int
    fmt.Scan(&n)
    a := make([]int, n)
    for i := 0; i < n; i++ {
        fmt.Scan(&a[i])
    }
    fmt.Scan(&target)
    l, r := 0, n-1
    for l <= r {
        mid = (l + r) >> 1
        cnt++
        if a[mid] == target {
            fmt.Println(mid)
            fmt.Println(cnt)
            return
        } else if a[mid] < target {
            l = mid + 1
        } else {
            r = mid - 1
        }
    }
    fmt.Println("-1")
    fmt.Println(cnt)
}

```

```

package main

//求鞍点
//
//给定一个 n*n 矩阵 A。矩阵 A 的鞍点是一个位置 (i, j) ,
//在该位置上的元素是第 i 行上的最大数, 第 j 列上的最小数。
//一个矩阵 A 也可能没有鞍点。你的任务是找出 A 的鞍点。。
//输入格式:
//
//自行输入一个二维数组, 二维数组的长度才从键盘读入
//输出格式:
//
//对输入的矩阵, 如果找到鞍点, 就输出其下标。
//下标为两个数字, 第一个数字是行号, 第二个数字是列号, 均从 0 开始计数。
//
//如果找不到, 就输出 找不到鞍点
//输入样例:
//
//2 3 1 2 3 4 5 6
//输出样例:
//
//鞍点为 2(0,1)
import "fmt"

func findSaddlePoint(matrix [][]int) (int, [2]int) {
    n := len(matrix)
    m := len(matrix[0])
    maxInRow := make([]int, n) // 存储每行最大值的数组

    // 找出每行的最大值
    for i := 0; i < n; i++ {
        maxInRow[i] = matrix[i][0]
        for j := 1; j < m; j++ {
            if matrix[i][j] > maxInRow[i] {
                maxInRow[i] = matrix[i][j]
            }
        }
    }

    // 遍历每列, 找出每列的最小值, 与所在行的最大值比较, 如果相等就是鞍点
    for j := 0; j < m; j++ {
        minInCol := matrix[0][j]
        index := 0
        for i := 1; i < n; i++ {

```

```

    if matrix[i][j] < minInCol {
        minInCol = matrix[i][j]
        index = i
    }
}

if minInCol == maxInRow[index] {
    return minInCol, [2]int{index, j}
}
}

// 没有找到鞍点
return -1, [2]int{-1, -1}
}

func main() {
    var n, m int
    fmt.Scan(&n, &m)

    // 读取矩阵
    matrix := make([][]int, n)
    for i := 0; i < n; i++ {
        matrix[i] = make([]int, m)
        for j := 0; j < m; j++ {
            fmt.Scan(&matrix[i][j])
        }
    }

    // 查找鞍点
    value, index := findSaddlePoint(matrix)

    // 输出结果
    if value != -1 {
        fmt.Printf("鞍点为%d(%d,%d)", value, index[0], index[1])
    } else {
        fmt.Println("找不到鞍点")
    }
}

```

/*

输入格式:

测试数据有多组，处理到文件尾。每组测试输入一正整数 N ($1 \leq N \leq 1000000$)。

输出格式:

对于每组测试，输出占一行，如果输入的正整数是素数，则输出其排位，否则输出 0。

输入样例:

```
6
2
6
4
5
13
991703
```

输出样例:

```
1
0
0
3
6
77901
```

*/

```
package main
```

```
import (
    "fmt"
    "math"
)
```

```
func IsPrime(n int) bool {
    if n <= 1 {
        return false
    }
    // 判断 n 是否可以被 2~sqrt(n) 中的任意一个数整除
    for i := 2; i <= int(math.Sqrt(float64(n))); i++ {
        if n%i == 0 {
            return false
        }
    }
    return true
}
```



```

func PrimeSort(num int) int {
    count := 1
    if IsPrime(num) {
        if num == 2 {
            return count
        }
        for j := 3; j <= num; j += 2 {
            if IsPrime(j) {
                count++
            }
            if j == num {
                return count
            }
            break
        }
    }
    return 0
}

func main() {
    var (
        n int
        r []int
    )
    fmt.Scanf("%d\n", &n)
    arr := make([]int, n)
    for i := 0; i < n; i++ {
        fmt.Scanln(&arr[i])
    }
    for i := 0; i < len(arr); i++ {
        r = append(r, PrimeSort(arr[i]))
    }
    for i := 0; i < len(r); i++ {
        fmt.Println(r[i])
    }
}

```

/*

首先输入一个正整数 T，表示测试数据的组数，然后是 T 组测试数据。

每组测试数据先输入 1 个正整数 n ($1 \leq n \leq 100$)，表示学生总数。然后输入 n 行，每行包括 1 个不含空格的字符串 s (不超过 8 位) 和 1 个正整数 d，分别表示一个学生的学号和解题总数。

输出参考：

```
fmt.Printf("%-5d%-8s%d\n")
```

输出格式：

对于每组测试数据，输出最终排名信息，每行一个学生的信息：排名、学号、解题总数。每行数据之间留一个空格。注意，解题总数相同的学生其排名也相同。

输入样例：

```
1
4
0010 200
1000 110
0001 200
0100 225
```

输出样例：

```
1 0100 225
2 0001 200
2 0010 200
4 1000 110
```

*/

```
package main
```

```
import (
    "fmt"
    "sort"
)
```

```
type Student struct {
    Id      string
    Count   int
}
```

```
type ByCountThenId []Student
```

```
func (a ByCountThenId) Len() int           { return len(a) }
func (a ByCountThenId) Swap(i, j int)      { a[i], a[j] = a[j], a[i] }
func (a ByCountThenId) Less(i, j int) bool {
```

```

if a[i].Count == a[j].Count {
return a[i].Id < a[j].Id
}
return a[i].Count > a[j].Count
}

func main() {
var t int
fmt.Scan(&t)
for i := 0; i < t; i++ {
var n int
fmt.Scan(&n)
students := make([]Student, n)
for j := 0; j < n; j++ {
fmt.Scan(&students[j].Id, &students[j].Count)
}
sort.Sort(ByCountThenId(students))
for j := 0; j < n; j++ {
rank := j + 1
if j > 0 && students[j].Count == students[j-1].Count {
rank = j
}
fmt.Printf("%-5d%-8s%d\n", rank, students[j].Id, students[j].Count)
}
fmt.Println()
}
}

```

```

//斐波那契数列
//
//输出所需长度的斐波那契数列
//输入格式:
//
//输入一个整数为数列的长度
//输出格式:
//
//输出对应长度的斐波那契数列, 每个数字用空格隔开
//输入样例:
//
//4
//输出样例:

```

```
//
//1 1 2 3
//package main
//
//import "fmt"
//
//func main() {
// var n int
// fmt.Scan(&n)
// result := make([]int, n)
// result[0] = 1
// result[1] = 1
// for i := 2; i < n; i++ {
// result[i] = result[i-1] + result[i-2]
// }
// for _, v := range result {
// fmt.Printf("%d ", v)
// }
//}
```

```
package main
```

```
import "fmt"
```

```
func main() {
var n, target, mid, cnt int
fmt.Scan(&n)
a := make([]int, n)
for i := 0; i < n; i++ {
fmt.Scan(&a[i])
}
fmt.Scan(&target)
var l, r int
l, r = 0, n-1
for l <= r {
mid = (r + l) / 2
cnt++
if a[mid] == target {
fmt.Println(mid)
fmt.Println(cnt)
return
} else if a[mid] < target {
l = mid + 1
```

```

} else {
r = mid - 1
}
}
}

//螺旋矩阵
//
//所谓“螺旋矩阵”，是指对任意给定的N，将1到N×N的数字从左上角第1
//格子开始，
//按顺时针螺旋方向顺序填入N×N的方阵里。本题要求构造这样的螺旋方阵。
//输入格式：
//
//输入在一行中给出一个正整数N（<10）。
//输出格式：
//
//输出N×N的螺旋方阵。每行N个数字，每个数字占4位。
//输入样例：
//
//5
//输出样例：
//
//1      2      3      4      5
//16     17     18     19     6
//15     24     25     20     7
//14     23     22     21     8
//13     12     11     10     9
//package main
//
//import "fmt"
//
//func main() {
// var n int
// fmt.Scan(&n) // 输入 n
// // 定义 n x n 的矩阵
// matrix := make([][]int, n)
// for i := range matrix {
// matrix[i] = make([]int, n)
// }
// num := 1 // 从 1 开始填充
// // 填充螺旋矩阵
//
// for l, r, t, b := 0, n-1, 0, n-1; l <= r && t <= b;

```

```

{
// // 从左往右填充
// for i := l; i <= r; i++ {
// matrix[t][i] = num
// num++
// }
// t++
// // 从上往下填充
// for i := t; i <= b; i++ {
// matrix[i][r] = num
// num++
// }
// r--
// // 从右往左填充
// for i := r; i >= l && t <= b; i-- {
// matrix[b][i] = num
// num++
// }
// b--
// // 从下往上填充
// for i := b; i >= t && l <= r; i-- {
// matrix[i][l] = num
// num++
// }
// l++
// }
// // 输出螺旋矩阵
// for _, row := range matrix {
// for _, v := range row {
// fmt.Printf("%4d", v)
// }
//     fmt.Println()
// }
//}

```

```
package main
```

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

```
//defer 使用
```

```
//  
//请编写一个程序，模拟一个简单的银行账户系统。该系统可以记录用户的账户  
//余额，  
//并提供存款和取款的功能。  
//  
//要求如下：  
//  
//程序需要定义一个 Account 结构体，包含以下字段：  
//  
//name: 表示账户名的字符串  
//balance: 表示账户余额的浮点数  
//Account 结构体需要提供以下方法：  
//  
//Deposit(amount float64): 存款方法，将传入的金额加到账户余额中。  
//Withdraw(amount float64): 取款方法，从账户余额中扣除传入的金额。如  
//果余额不足，则无法取款。  
//PrintBalance(): 打印当前账户余额。  
//使用 defer 关键字，在存款和取款方法中添加日志记录功能。在存款时打  
//印格式为 “存款:账户名 + 存款金额” 的日志，取款时打印格式为 “取款:  
//账户名 - 取款金额” 的日志。  
//  
//主函数中创建一个账户实例，并演示存款和取款的操作，以及打印账户余额。  
//  
//  
//请根据以上要求编写程序，并保证正确处理边界情况。  
//  
//输入输出示例：  
//（假设账户名为 “Alice”）  
//注意：  
//使用 fmt.Printf 函数打印日志和账户余额，保留两位小数，带回车。  
//需要使用 defer 关键字来确保日志的打印顺序。  
//在取款时，如果账户余额不足，需要打印错误信息：“账户余额不足，无法取  
//款”。  
//输入格式：  
//  
//输入：  
//存款金额: 100.0  
//取款金额: 50.0  
//输出格式：  
//  
//输出：  
//存款: Alice + 100.0  
//取款: Alice - 50.0  
//账户余额: 50.0
```

```

//输入样例:
//
//100.0
//50.0
//输出样例:
//
//存款: Alice + 100.0
//取款: Alice - 50.0
//账户余额: 50.0
/*
import (
    "fmt"
)

// 定义账户结构体
type Account struct {
    name      string
    balance   float64
}

// 存款方法, 打印存款日志
// 使用 defer 确保日志打印顺序
func (a *Account) Deposit(amount float64) {
    defer fmt.Printf("存款: %s + %.2f\n", a.name, amount)
    a.balance += amount
}

// 取款方法, 打印取款日志
// 使用 defer 确保日志打印顺序
func (a *Account) Withdraw(amount float64) {
    defer fmt.Printf("取款: %s - %.2f\n", a.name, amount)
    if a.balance < amount {
        fmt.Println("账户余额不足, 无法取款")
        return
    }
    a.balance -= amount
}

// 打印账户余额
func (a *Account) PrintBalance() {
    fmt.Printf("账户余额: %.2f\n", a.balance)
}

func main() {

```



```
// 创建账户实例
account := &Account{
name:      "Alice",
balance: 0,
}
```

```
// 存款并打印账户余额
var n,m float64
fmt.Scan(&n)
fmt.Scan(&m)
account.Deposit(n)
```

```
// 取款并打印账户余额
account.Withdraw(m)
account.PrintBalance()
}
```

```
*/
```

```
//今昔是何年
```

```
//
```

//天干地支，简称为干支，源自中国远古时代对天象的观测。十干是指阏逢、旃蒙、柔兆、强圉、著雍、屠维、上章、重光、玄默、昭阳。

//十二支是指困敦、赤奋若、摄提格、单阏、执徐、大荒落、敦牂、协洽、涓滩、作噩、阉茂、大渊献。

//简化后“甲、乙、丙、丁、戊、己、庚、辛、壬、癸”称为十天干，“子、丑、寅、卯、辰、巳、午、未、申、酉、戌、亥”称为十二地支。

//干支纪年一个周期的第1年为“甲子”，第2年为“乙丑”，第3年为“丙寅”，...，第11年为“甲戌”，

//第12年为“乙亥”，第13年为“丙子”，依此类推，60年一个周期；一个周期完了重复使用，周而复始。

//已知2022年为“壬寅”年，1984年为“甲子”年，请根据给定的年份，求对应年份的干支。

//输入格式：

```
//
```

//第一行包含一个整数 T ($1 \leq T \leq 1000$)，表示一共有 T 个样例，后面 T 行，每行包含一个整数 Y ($1 \leq Y \leq 5000$)，代表一个年份。

//输出格式：

```
//
```

//对于每个测试样例，输出相应的干支。

//输入样例：

```

//
//4
//1800
//1984
//2000
//2023
//输出样例:
//
//庚申
//甲子
//庚辰
//癸卯
func tiangandizhi(year int) string {
    tiangan := []string{"甲", "乙", "丙", "丁", "戊", "己", "庚",
        "辛", "壬", "癸"}
    dizhi := []string{"子", "丑", "寅", "卯", "辰", "巳", "午",
        "未", "申", "酉", "戌", "亥"}
    baseyear := 1984
    diff := year - baseyear
    if diff < 0 {
        diff = (60 - (-diff)%60) % 60 // 防止出现负数
    }
    return tiangan[diff%10] + dizhi[diff%12]
}

func main() {
    var n int
    fmt.Scan(&n)

    a := make([]int, n)

    for i := 0; i < n; i++ {
        fmt.Scan(&a[i])
    }

    for _, year := range a {
        fmt.Println(tiangandizhi(year))
    }
}

```

```

//搜索插入位置
//
//给定一个排序数组和一个目标值，在数组中找到目标值，并返回其索引。如果
//目标值不存在于数组中，返回它将会被按顺序插入的位置。
//输入格式：
//
//输入整数数存入数组。第一个数字代表数组长度    最后一个数字代表目标值，
//可参考下面输入代码
//👉var n int
//fmt.Scan(&n)
//nums := make([]int, n)
//for i := 0; i < n; i++ {
//fmt.Scan(&nums[i])
//}
//fmt.Scan(&target)
//
//nums 为无重复元素 的 升序 排列数组
//输出格式：
//
//输出插入位置下标。
//
//输入      4  1  3  5  6  5
//输出      2
//
//输入      4  1  3  5  6  7
//输出      4
//输入样例：
//
//4  1  3  5  6  2
//输出样例：
//
//1
//
/*
package main

import "fmt"

func searchInsert(nums []int, target int) int {
low, high := 0, len(nums)-1

for low <= high {
mid := low + (high-low)/2

```

```

if nums[mid] == target {
    return mid
} else if nums[mid] < target {
    low = mid + 1
} else {
    high = mid - 1
}
}

return low
}

```

```

func main() {
    var n, target int
    fmt.Scan(&n)

```

```

    nums := make([]int, n)
    for i := 0; i < n; i++ {
        fmt.Scan(&nums[i])
    }

```

```

    fmt.Scan(&target)

```

```

    insertIndex := searchInsert(nums, target)
    fmt.Println(insertIndex)
}

```

```

*/
//首先输入一个正整数 T，表示测试数据的组数，然后是 T 组测试数据。
//每组测试数据先输入 1 个正整数 n ( $1 \leq n \leq 100$ )，表示学生总数。
//然后输入 n 行，每行包括 1 个不含空格的字符串 s（不超过 8 位）和 1 个正整数
//d，分别表示一个学生的学号和解题总数。
//输出参考：
//fmt.Printf("%-5d%-8s%d\n")
//输出格式：
//
//对于每组测试数据，输出最终排名信息，每行一个学生的信息：排名、学号、
//解题总数。每行数据之间留一个空格。注意，解题总数相同的学生其排名也相同。
//输入样例：
//
//1

```

```

//4
//0010 200
//1000 110
//0001 200
//0100 225
//输出样例:
//
//1 0100 225
//2 0001 200
//2 0010 200
//4 1000 110
/*
package main

import (
    "fmt"
    "sort"
)

type student struct {
    id      string
    solved  int
    ranking int
}

func main() {
    var t int
    fmt.Scan(&t)
    for i := 0; i < t; i++ {
        var n int
        fmt.Scan(&n)
        stus := make([]student, n)
        for j := 0; j < n; j++ {
            fmt.Scan(&stus[j].id, &stus[j].solved)
        }
        sort.Slice(stus, func(i, j int) bool {
            if stus[i].solved != stus[j].solved {
                return stus[i].solved > stus[j].solved
            }
            return stus[i].id < stus[j].id
        })
        for j, s := range stus {
            if j == 0 {
                s.ranking = 1
            }
        }
    }
}

```

```

} else if s.solved == stus[j-1].solved {
s ranking = stus[j-1].ranking
} else {
s ranking = j + 1
}
stus[j] = s
}
for _, s := range stus {
fmt.Printf("%-5d%-8s%d\n", s ranking, s.id, s.solved)
}
fmt.Println()
}
}

*/

```

C 1.关于函数声明,下面语法错误的是。

A.func f(a,b int) (value int, err error)

B.func f(a int,b int) (value int, err error)

C.func f(a,b int) (value int, error)

D.func f(a int,b int) (int, error)

B 2.golang 中的引用类型不包括。

A.map

B.数组

C.channel

D.interface

D 3.关于 select 机制,下面说法不正确的是()。

A.select 用来处理异步 IO 问题

B.select 机制最大的一条限制就是每个 case 语句里必须是一个 IO 操作

C.golang 在语言级别支持 select 关键字

D.select 关键字的用法与 switch 语句非常类似,后面要带判断条件

D 4.以下哪个不属于 Go 的关键字。

A.break

B.func

C.struct

D.channel

B 5.关于变量的声明,以下错误的是。

A.var name = "fmt"

B.var name,sex int= "明明",19

C.var name string = "红红"

D.name := "明明"

B 6.下列关于数组声明中错误的是。

A.var arr0 [5]int = [5]int{1, 2, 3}

B.var arr1 = []int{1, 2, 3, 4, 5}

C.var arr2 = [...]int{1, 2, 3, 4, 5, 6}

D.var str = [5]string{3: "hello world", 4: "tom"}

D 7.下列有关数组声明的使用错误的是。

A.a := [2][3]int{{1, 2, 3}, {4, 5, 6}}

B.var str = [5]string{3: "hello world", 4: "tom"}

C.var arr0 [5][3]int

D.b := [...]int{{1, 1}, {2, 2}, {3, 3}}

B 8. b := 123 a := (b / 10) % 10.最后 a 的值为。

A.3

B.2

C.12

D.1

C 9.关于类型转化，下面语法正确的是（）。

A. type MyInt int var i int = 1 var j MyInt = i

B. type MyInt int var i int = 1 var j MyInt = (MyInt)i

C. type MyInt int var i int = 1 var j MyInt = MyInt(i)

D. type MyInt int var i int = 1 var j MyInt = i.(MyInt)

C 10.下面的程序的运行结果是（）。

```
func main() { if (true) { defer fmt.Printf("1") } else { defer fm  
t.Printf("2") } fmt.Printf("3") }
```

A.321

B.32

C.31

D.13

D

1. 从切片中删除一个元素，下面的算法实现正确的是（ ）。

A. `func (s *Slice)Remove(value interface{}) error { for i, v := range *s { if isEqual(value, v) { if i == len(*s) - 1 { *s = (*s)[:i] } else { *s = append((*s)[:i], (*s)[i + 2:]...) } return nil } } return ERR_ELEM_NT_EXIST }`

B. `func (s *Slice)Remove(value interface{}) error { for i, v := range *s { if isEqual(value, v) { *s = append((*s)[:i], (*s)[i + 1:]) return nil } } return ERR_ELEM_NT_EXIST }`

C. `func (s *Slice)Remove(value interface{}) error { for i, v := range *s { if isEqual(value, v) { delete(*s, v) return nil } } return ERR_ELEM_NT_EXIST }`

D. `func (s *Slice)Remove(value interface{}) error { for i, v := range *s { if isEqual(value, v) { *s = append((*s)[:i], (*s)[i + 1:]...) return nil } } return ERR_ELEM_NT_EXIST }`

A

2. 如果 Add 函数的调用代码为：。

`func main() { var a Integer = 1 var b Integer = 2 var i interface{} = a sum := i. (Integer). Add(b) fmt.Println(sum) }` 则 Add 函数定义正确的是（ ）

A. `type Integer int func (a Integer) Add(b Integer) Integer { return a + b }`

B. `type Integer int func (a Integer) Add(b *Integer) Integer { return a + *b }`

C. `type Integer int func (a *Integer) Add(b Integer) Integer { return *a + b }`

D. `type Integer int func (a *Integer) Add(b *Integer) Integer { return *a + *b }`

D

3. 关于无缓冲和有冲突的 channel，下面说法正确的是（ ）。

A. 无缓冲的 channel 是默认的缓冲为 1 的 channel

B. 无缓冲的 channel 和有缓冲的 channel 都是同步的

C. 无缓冲的 channel 和有缓冲的 channel 都是非同步的

D. 无缓冲的 channel 是同步的，而有缓冲的 channel 是非同步的

A

4. 关于 map，下面说法正确的是（ ）。

A. map 反序列化时 json.Unmarshal 的入参必须为 map 的地址

B. 在函数调用中传递 map，则子函数中对 map 元素的增加不会导致父函数中 map 的修改

C. 在函数调用中传递 map，则子函数中对 map 元素的修改不会导致父函数中 map 的修改

D. 不能使用内置函数 delete 删除 map 的元素

D 5. 以下标识符命名不正确的是。

A.maresh

B.a_123

C._crash

D.defer

C 6. 下面关于标识符的命名, 正确的是。

A.switch

B.a+b

C.move_name

D.3sa

A 7. 关于 Go 中切片的特性 下列说明正确的是：。

A. 一个空切片长度和容量都是 0，并且不存在底层数组

B. 每次调用 `append` 时都会创建一个新的切片值

C. 在容量增长的时候每次都将容量翻倍

D. 将切片赋值给其它变量，会创建一个新的切片值

C 8. 下面关于 Go 语言中 goroutine 的描述中，正确的是：。

A.goroutine 是由操作系统调度的线程

B.goroutine 的栈大小是固定的

C.goroutine 可以通过 `channel` 来同步和通信

D.goroutine 可以通过 `mutex` 来实现互斥访问

D 9. 下面关于 Go 语言中接口的描述中，正确的是：。

A. 接口类型可以被实例化

B. 接口类型可以包含私有成员

C. 实现接口时，必须显式声明实现了哪些接口

D.实现接口时，必须实现接口中所有的方法

B

10. 关于 Go 语言中的 defer 语句，以下描述正确的是：。

A.defer 语句必须在函数的最后一行执行

B.defer 语句可以用于关闭文件、释放资源等操作

C.defer 语句可以使用闭包访问外部变量

D.在 defer 语句中修改返回值不会影响函数的返回值

C

1. 下面关于 Go 语言中的 struct 的描述中，正确的是：。

A.struct 可以继承其他 struct

B.struct 的字段可以是函数类型

C.struct 的字段可以是切片类型

D.struct 的字段可以是接口类型

D

2. 下面关于 Go 语言中的错误处理的描述中，正确的是。

A.在函数中返回错误时，通常需要将错误信息输出到控制台

B.在函数中返回错误时，应该使用 panic

C.在函数中返回错误时，应该使用 log 来记录错误信息

D.在函数中返回错误时，应该将错误信息封装在 error 类型中返回

C

3. 下面关于 Go 语言中的数组和切片的描述中，正确的是：。

A.数组和切片都是引用类型

B.数组和切片都可以动态增长

C.数组的长度是固定的，切片的长度是可变的

D.数组和切片的元素类型必须相同

D

4. 设有说明语句如下；关于 c, d 的取值正确的是。

```
var a = false var b = true c := (a && b) && (!b) d := (a || b) && (!a)
```

A.true>true

B.true>false

C.false>false

D.false>true

D 5. 关于类型转化，下面语法不正确的是（ ）。

A.type MyInt int var i int = 1 var jMyInt = i

B.type MyIntint var i int= 1 var jMyInt = (MyInt)i

C.type MyIntint var i int= 1 var jMyInt = MyInt(i)

D.type MyIntint var i int= 1 var jMyInt = i.(MyInt)

A 6. 下面哪个函数可以将字符串转换为整数？。

A.strconv.Atoi()

B.strconv.Itoa() → int → string

C.strconv.ParseBool()

D. strconv.ParseFloat()

D 7. 下面哪个不是 Golang 的控制语句？。

A.if

B.switch

C.for

D.continue

A 8. 下面哪个语句可以创建一个新的 Goroutine？。

A.go myFunction()

B.newGoroutine(myFunction())

C.startGoroutine(myFunction())

D. createGoroutine(myFunction())

A

9. 下面哪个包提供了 Golang 中的并发功能？。

A.sync

B.io

C.time

D.fmt

A

10. Golang 中的包（package）是指什么？。

A.一组相关的变量和函数

B.一个可执行的程序

C.一组关联的数据结构

D.一段程序代码

C

1. 下面哪些场景不适合使用 Golang 的 channel 进行通信？。

A. 线程间通信

B.进程间通信

C.任务调度

D.事件驱动

B

2. 下列哪种关键字可以用于声明 Golang 中的常量？。

A.var

B.const

C.let

D.def

C

3. 下面哪个函数可以用于从标准输入读取一行字符串？。

A.fmt.Scanf()

B. fmt.Scan()

C.bufio.NewReader()

D.bufio.ReadLine()

A 4. 关于 Go 语言切片 (slice) 的描述, 哪个选项是正确的? 。

A.切片是数组的引用类型 ✓

B.切片的长度是固定的 ✗

C.切片不能增加或减少元素 ✗

D.切片的容量不能改变 ✗

C 5. 关于 Go 语言接口 (interface) 的描述, 哪个选项是不正确的? 。

A.接口可以嵌套其他接口

B.接口只能包含方法定义, 不能包含变量 ✓

C.接口可以实现具体的方法 ✗

D.接口不能被实例化 ✓

D 6. 下面哪个语句可以用来从一个通道中读取数据, 同时判断通道是否已经关闭? 。

A.val := <-ch

B.<-ch

C.close(ch)

D.val, ok := <-ch

A 7. 下面程序会输出什么? 。

```
package main import ( "fmt" ) func main() { ch := make(chan int) go func() { for i := 0; i < 10; i++ { ch <- i } close(ch) }() for v := range ch { fmt.Println(v) } }
```

A.0 1 2 3 4 5 6 7 8 9

B.1 2 3 4 5 6 7 8 9 10

C.0 1 2 3 4 5 6 7 8 9 10

D.程序会死锁

C 8. 下面哪个关键字可以用来定义一个匿名字段? 。

- A.field
- B.type
- C.struct
- D.anonymous

D 9. 下面哪个方法可以用来创建一个新的缓冲区？。

- A.bufio.NewWriter()
- B.bufio.NewReader()
- C.bufio.NewScanner()
- D.bufio.NewBuffer()

D 10. 下面哪个函数可以用来在通道上进行非阻塞的读写操作？。

- A.close()
- B.len()
- C.cap()
- D.select()

C 1. 下面程序输出什么。

```
package main import ( "fmt" "time" ) func main() { ch := make(chan int, 1) go func() {  
time.Sleep(time.Second * 2) v := <-ch fmt.Println("Received value:", v) }() v := 42 fmt.  
Println("Sending value:", v) ch <- v time.Sleep(time.Second * 3) }
```

- A.Sending value: 42
- B.Received value: 42
- C.Sending value: 42 Received value: 42
- D.程序会一直阻塞

C 2. 下面程序输出什么？。

```
package main import ( "fmt" "sync" ) func main() { var wg sync.WaitGroup wg.Add(2) ch :  
= make(chan int) go func() { defer wg.Done() for i := 0; i < 3; i++ { v := <-ch fmt.Pri  
ntln("Received value:", v) } }() go func() { defer wg.Done() for i := 0; i < 3; i++ { f  
mt.Println("Sending value:", i) ch <- i } }() wg.Wait() close(ch) }
```


A. Sending value: 0 1 2 3 4 Received value: 0 1 2 3 4

B. Received value: 0 1 2 3 4 Sending value: 0 1 2 3 4

C. Sending value: 0 Received value: 0 Sending value: 1 Sending value: 2 Received value: 1 Received value: 2

D.输出结果不确定

A B

3. 以下哪个选项可以正确地使用 `errors` 包创建一个错误？。

A.`err := errors.New("This is an error.")`

B.`err := errors.Error("This is an error.")`

C.`err := errors.Create("This is an error.")`

D.`err := errors.Err("This is an error.")`

B

4. 对于局部变量整型切片 `x` 的赋值，下面定义不正确的是？。

A.`x := []int{ 1, 2, 3, 4, 5, 6, }`

B.`x := []int{ 1, 2, 3, 4, 5, 6 }`

C.`x := []int{ 1, 2, 3, 4, 5, 6}`

D.`x := []int{1, 2, 3, 4, 5, 6,}`

C

5. 关于 `channel` 的特性，下面说法正确的是。

A.给一个 `nil channel` 发送数据，造成永远阻塞

B.从一个 `nil channel` 接收数据，造成永远阻塞

C.给一个已经关闭的 `channel` 发送数据，不会引起 `panic`

D.从一个已经关闭的 `channel` 接收数据，如果缓冲区中为空，则返回一个零值

D

6. 关于 `select` 机制，下面说法正确的是。

A.`select` 机制用来处理异步 IO 问题

B.`select` 机制最大的一条限制就是每个 `case` 语句里必须是一个 IO 操作

C.golang 在语言级别支持 `select` 关键字

D.`select` 关键字的用法与 `switch` 语句非常类似，后面要带判断条件

A

7. 请选出代码正确的输出结果。

```
type Person struct { name string age int } func main() { p1 := Person{"Alice", 20} p2 := Person{name: "Bob", age: 30} p3 := Person{name: "Charlie"} fmt.Println(p1.name, p1.age) fmt.Println(p2.name, p2.age) fmt.Println(p3.name, p3.age) }
```

A.Alice 20 Bob 30 Charlie 0

B.Alice 20 Bob 30 Charlie

C.Alice 20 Bob 30 empty 0

D.编译错误

A

8. 下面代码输出的正确结果是。

```
type Rectangle struct { width, height float64 } func (r Rectangle) Area() float64 { return r.width * r.height } func (r Rectangle) Perimeter() float64 { return 2 * (r.width + r.height) } func main() { rect := Rectangle{width: 10, height: 5} fmt.Println("Area:", rect.Area()) fmt.Println("Perimeter:", rect.Perimeter()) }
```

A.Area: 50 Perimeter: 30

B.Area: 30 Perimeter: 50

C.编译错误

D.Area: 30.0 Perimeter: 50.0

D

9. 下面程序正确的输出结果是。

```
type ListNode struct { Val int Next *ListNode } func fn(node *ListNode) { node.Val = node.Next.Val node.Next = node.Next.Next } func main() { head := &ListNode{Val: 1, Next: &ListNode{Val: 2, Next: &ListNode{Val: 3, Next: nil}}} fn(head.Next) fmt.Println(head.Val, head.Next.Val) }
```

A.1 2

B.2 3

C.3 2

D.1 3

C

10. 下面代码输出什么。

```
func main() { a1 := []int{10} a2 := a1[1:] fmt.Println(a2) }
```

A.编译失败

B.panic: runtime error: index out of range [1] with length 1

C.[]

D.其他

B

1. 下面正确的答案是。

```
func main() { var x int = 10 var y *int = &x *y = 20 fmt.Println(x) }
```

A.10

B.20

C.运行错误

D.编译错误

B

2. 下面答案正确的一项是。

```
func main() { nums := []int{1, 2, 3, 4, 5} var funcs []func() for _, num := range nums { funcs = append(funcs, func() { fmt.Println(num) }) } for _, f := range funcs { f() } }
```

A.1 2 3 4 5

B.5 5 5 5 5

C.编译错误

D.1 1 1 1 1

D

3. 关于协程，下面说法正确是。

A.通过共享内存来实现通信

B.线程比协程更轻量级

C.协程不存在死锁问题

D.通过 channel 来进行协程间的通信

D

4. 关于 channel，下面语法错误的是。

A.var ch chan int ✓

B.ch := make(chan int) ✓

C.<- ch

D.ch <-

D 5. 有关协程的说法错误的是。

A.协程和线程都可以实现程序的并发执行

B.通过 `channel` 来进行协程间的通信

C.只需要在函数调用前添加 `go` 关键字即可实现 `go` 的协程，创建并发任务

D.协程比线程耗费资源更多一些

D 6. Go Modules 的主要作用是什么？。

A.管理项目的依赖关系

B.安装和更新依赖包

C.管理项目版本号

D.所有以上都是

B 7. 在 Go Modules 中，如何初始化一个新的模块？。

A.使用 `go get` 命令

B.使用 `go mod init` 命令

C.使用 `go mod tidy` 命令

D.使用 `go mod vendor` 命令

A 8. Go 语言和 Java 的编译方式有何不同？。

A.Go 语言需要先编译成二进制文件再运行

B. Java 需要先编译成二进制文件再运行

C.Go 语言可以直接运行源代码

D. Java 可以直接运行源代码

A 9. Gin 框架中如何实现文件上传？。

A.使用 `context.SaveUploadedFile()` 函数实现

B.使用 `context.PostForm()` 函数实现

C.使用 `context.File()` 函数实现

D.以上都不是

A

10. Gin 框架支持哪种类型的路由？。

A.RESTful 路由

B. RPC 路由

C.WebSocket 路由

D.全局路由

C

1. Gin 框架中如何实现中间件？。

A.使用 `context.Next()` 函数实现

B.使用 `context.Abort()` 函数实现

C. 使用 `gin.Use()` 函数注册中间件

D.以上都可以

A

2. 在 Go 语言中，如何向切片中添加元素？。

A.使用 `append()` 函数

B.使用 `push()` 函数

C. 使用 `insert()` 函数

D.以上都不是

D

3. 在 Go 语言中，如何向数组中添加元素？。

A.使用 `append()` 函数

B.使用 `push()` 函数

C.使用 `insert()` 函数

D.无法做到

A

4. 在 Go 语言中，如何获取切片的长度和容量？。

- A.使用 `len()` 函数获取切片的长度，使用 `cap()` 函数获取切片的容量
- B.使用 `length()` 函数获取切片的长度，使用 `capacity()` 函数获取切片的容量
- C.使用 `size()` 函数获取切片的长度和容量
- D.以上都不是

B

5. 在 Go 语言中，如果一个无缓冲 channel 中没有数据可读，会发生什么？。

- A.发送操作会被阻塞
- B.接收操作会被阻塞
- C.以上都会被阻塞
- D.不会被阻塞

A

6. 在 Go 语言中，如何创建一个有缓冲的 channel？。

- A.`make(chan T, n)`
- B.`make(chan T, 0)`
- C.以上都可以
- D.都不可以

D

7. 在 Go 语言中，如何使用互斥锁？。

- A.使用 `sync.Mutex` 类型
- B.使用 `mutex()` 函数
- C.使用 `lock()` 和 `unlock()` 方法
- D.以上都是

A

8. 在 Go 语言中，如何使用 `select` 语句？。

- A.用于等待多个 channel 中的数据到达
- B.用于等待单个 channel 中的数据到达
- C.用于等待定时器到期
- D.以上都是

A

9. 在 Go 语言中，如何使用 panic 和 recover 实现异常处理？。

- A.使用 defer 语句和 panic() 函数触发 panic，使用 recover() 函数捕获 panic
- B.使用 try-catch 块实现异常处理
- C.使用 goto 语句实现异常处理
- D.以上都不是

A

10. 在 Go 语言中，如何使用反射？。

- A.使用 reflect 包中的类型和值函数
- B.使用反射语句 ref(x)
- C.使用反射类型反射函数
- D.以上都不是