Laboratorio 9 - Array e slice II - Complementi

1 Qual è l'output?

Qual è l'output del seguente programma?

```
package main

import (
    "fmt"
)

func main() {

    var a [3][2]int = [3][2]int{{1, 2}, {10, 20}, {100, 200}}
    //a := [3][2]int{{1, 2}, {10, 20}, {100, 200}}
    //a := [...][2]int{{1, 2}, {10, 20}, {100, 200}}

    fmt.Println("a:", a)
    /* a è un array bi-dimensionale di tipo [3][2]int
        con lunghezza pari a 3 */
    fmt.Printf("Tipo di a: %T\n", a)

    fmt.Println()

    for i, r := range a {
        fmt.Printf("a[%d]: %v\n", i, r)
    }
    for i := 0; i < len(a); i++ {
        /* a[i], con 0 <= i < len(a), è di tipo [2]int */
        fmt.Printf("Tipo di a[%d]: %T\n", i, a[i])
    }

    fmt.Println()

    for i := 0; i < len(a); i++ {
        for j := 0; j < len(a[i]); j++ {
            fmt.Printf("%d ", a[i][j])
        }
        fmt.Println()
    }
    for i := 0; i < len(a); i++ {
        for j := 0; j < len(a[i]); j++ {
            /* a[i][j], con 0 <= i < len(a) e con 0 <= j < len(a[i]),
                è di tipo int */
            fmt.Printf("a[%d][%d] è il valore alla riga %d e " +
                "colonna %d dell'array bi-dimensionale a: %d\n", i, j, i, j, a[i][j])
        }
    }

}
```

2 Qual è l'output?

Qual è l'output del seguente programma?

```
package main

import (
    "fmt"
```

```
    "math/rand"
    "time"
)

func main() {

    var slc [][]int
    fmt.Printf("Tipo di slc: %T\n", slc)

    if slc == nil {
        fmt.Println("'nil' è lo zero-value di una variabile di tipo " +
            "'reference type' non inizializzata.")
    }
    fmt.Printf("slc: %v %d %d\n", slc, len(slc), cap(slc))
    fmt.Println()

    n := 4
    slc = CreaSliceBidimensionale(n)

    InizializzaSliceBidimensionale(slc)

    fmt.Println("slc:")
    for _, r := range slc {
        fmt.Printf("%v\n", r)
    }

    slc_soprasoglia := FiltraSliceBidimensionale(slc, 2)

    fmt.Println("slc_soprasoglia:")
    for _, r := range slc_soprasoglia {
        fmt.Printf("%v\n", r)
    }

}

/* 'CreaSliceBidimensionale' riceve in input un valore int nel parametro l
e restituisce un valore s di tipo [][]int (una slice bi-dimensionale
di interi) con lunghezza/capacità pari a l in cui s[i], con 0 <= i < l,
è un valore di tipo []int (una slice di interi) con lunghezza/capacità pari a l
*/
func CreaSliceBidimensionale(l int) [][]int {
    var s [][]int
    /* s è una slice bi-dimensionale di tipo [][]int*/
    fmt.Printf("Tipo di s: %T\n", s)
    if s == nil {
        // 'nil' è lo zero-value di una variabile di tipo
        // 'reference type' non inizializzata
        fmt.Printf("s == nil \t=>\t")
        fmt.Printf("s: %v %d %d\n", s, len(s), cap(s))
    }
    s = make([][]int, l)
    /* s è una slice bi-dimensionale di tipo [][]int con lunghezza/capacità
        pari a l */
    fmt.Printf("s: %v %d %d\n\n", s, len(s), cap(s))

    for i := 0; i < l; i++ {
        /* s[i], con 0 <= i < dim, è di tipo []int */
        fmt.Printf("Tipo di s[%d]: %T\n", i, s[i])
        if s[i] == nil {
            // 'nil' è lo zero-value di una variabile di tipo
            // 'reference type' non inizializzata
            fmt.Printf("s[%d] == nil \t=>\t", i)
            fmt.Printf("s[%d]: %v %d %d\n", i, s[i], len(s[i]), cap(s[i]))
        }
        s[i] = make([]int, l)
        /* s[i] è una slice con lunghezza/capacità pari a l */
        fmt.Printf("s[%d]: %v %d %d\n\n", i, s[i], len(s[i]), cap(s[i]))
    }
    return s
}
```

```go
/* 'InizializzaSliceBidimensionale' riceve in input un valore [][]int nel
parametro s ed inizializza s[i][j] (con 0 <= i < len(s) e 0 <= j <len(s[i]))
con un valore intero estratto in maniera casuale nell'insieme {0,1}
ossia
'InizializzaSliceBidimensionale' inizializza una slice bi-dimensionale di
interi con valori interi estratti in maniera casuale nell'insieme {0,1} */
func InizializzaSliceBidimensionale(s [][]int) {

    rand.Seed(int64(time.Now().Nanosecond()))

    for i := 0; i < len(s); i++ {
        for j := 0; j < len(s[i]); j++ {
            s[i][j] = rand.Intn(2)
        }
    }

}

/* 'FiltraSliceBidimensionale' riceve in input un valore [][]int ed un valore
int nei parametri s e soglia, rispettivamente, e restituisce un
valore [][]int definito dai valori s[i] di s (con 0 <= i < len(s)) che
includono valori interi la cui somma è maggiore o uguale a soglia */
func FiltraSliceBidimensionale(s [][]int, soglia int) [][]int {

    var s_soprasoglia [][]int

    for i := 0; i < len(s); i++ {
        somma := 0
        for j := 0; j < len(s[i]); j++ {
            somma += s[i][j]
        }
        if somma >= soglia {
            s_soprasoglia = append(s_soprasoglia, s[i])
        }
    }

    return s_soprasoglia
}
```

3 Tavola pitagorica

Scrivere un programma che legga da **riga di comando** un numero intero `n` e, come mostrato nell'**Esempio di esecuzione**, stampi a video la corrispondente tavola pitagorica `n x n`.

Oltre alla funzione `main()`, devono essere definite ed utilizzate almeno le seguenti funzioni:

- una funzione `CreaTavolaPitagorica(n int) [][]int` che riceve in input un valore `int` nel parametro `n` e restituisce un valore di tipo `[][]int` in cui sono memorizzati i valori di una tavola pitagorica `n x n`;
- una funzione `StampaTavolaPitagorica(s [][]int)` che riceve in input un valore di tipo `[][]int` nel parametro `s` e, come mostrato nell'**Esempio di esecuzione**, stampa la tavola pitagorica corrispondete ai valori memorizzati `s`.

Esempio d'esecuzione:

```
$ go run tavola_pitagorica.go 5
   1    2    3    4    5
   2    4    6    8   10
   3    6    9   12   15
   4    8   12   16   20
   5   10   15   20   25

$ go run tavola_pitagorica.go 10
   1    2    3    4    5    6    7    8    9   10
   2    4    6    8   10   12   14   16   18   20
   3    6    9   12   15   18   21   24   27   30
```

```
   4    8   12   16   20   24   28   32   36   40
   5   10   15   20   25   30   35   40   45   50
   6   12   18   24   30   36   42   48   54   60
   7   14   21   28   35   42   49   56   63   70
   8   16   24   32   40   48   56   64   72   80
   9   18   27   36   45   54   63   72   81   90
  10   20   30   40   50   60   70   80   90  100
```

4 Coppie

Scrivere un programma che:

- legga da **riga di comando** un numero intero `n`;
- utilizzi le funzioni `CreaSliceBidimensionale(l int) [][]int` e `InizializzaSliceBidimensionale([][]int)` descritte nell'Esercizio 2 (Laboratorio 9 - Array e slice III) per inizializzare una variabile `s` di tipo `[][]int` con lunghezza/capacità pari a `n` in cui ogni elemento `s[i]`, con `0 <= i < l`, è un valore di tipo `[]int` con lunghezza/capacità pari a `n`;
- stampi a video tutte le coppie di indici `(i, j)`, con `0 <= i < l` e `0 <= j < l`, tali che `s[i][j]` è uguale a `1`.

Oltre alle funzioni `main()`, `CreaSliceBidimensionale()`, e `InizializzaSliceBidimensionale()` devono essere definite ed utilizzate almeno le seguenti funzioni:

- una funzione `Coppie(s [][]int) (coppie [][]int)` che riceve in input un valore `[][]int` nel parametro `s` e restituisce il valore di tipo `[][]int` nella variabile `coppie` in cui sono memorizzate tutte le coppie di indici `(i, j)`, con `0 <= i < l` e `0 <= j < l`, tali che `s[i][j]` è uguale a `1` (`coppie[k]`, con `0 <= k < len(coppie)`, è un valore di tipo `[]int` di lunghezza `2`).

Esempio d'esecuzione:

```
$ go run coppie.go 4
[1 1]
[1 3]
[2 1]
[2 2]
[3 2]
[3 3]
```