Laboratorio 7 - Array, Slice, Riga di comando

(Array) Qual è l'output?

Qual è l'output di questo programma?

```
package main

import "fmt"

func main() {
    b := [3]rune{'a', 'b', 'c'}

    for i := range b {
        fmt.Println(i)
    }

}
```

(Array) Qual è l'output?

Qual è l'output di questo programma?

```
package main

import "fmt"

func main() {
    var a [5]string

    a[1] = "hello"
    a[4] = "world"

    for i := range a {
        fmt.Print(a[i])
    }

}
```

(Array) Qual è l'output?

Qual è l'output di questo programma?

```
package main

import "fmt"

func main() {
```

```
    var a [6]int

    for i := range a {
        a[i] = i
    }

    for _, v := range a {
        v *= 2
    }

    fmt.Println(a)
}
```

(Array) Qual è l'output?

Qual è l'output di questo programma?

```
package main

import "fmt"

func main() {
    var a = [6]int{1, 2, 3, 4, 5, 6}
    fmt.Println(a)
    modifica(a)
    fmt.Println(a)
}

func modifica(a [6]int) {
    for i := range a {
        a[i] *= 2
    }
}
```

(Slice) Qual è l'output?

Qual è l'output di questo programma?

```
package main

import "fmt"

func main() {

    var n int = 5

    var s []int
```

```
    s = make([]int, n)

    for i := 0; i < n; i++ {
        s[i] = i
    }

    fmt.Println(s)
}
```

(Slice) Qual è l'output?

Qual è l'output di questo programma?

```
package main

import "fmt"

func main() {
    var a [6]int

    for i := range a {
        a[i] = i
    }

    var b []int
    b = a[:]

    for i := range b {
        b[i] = i * 2
    }

    fmt.Println(a)
}
```

(Slice) Qual è l'output?

Qual è l'output di questo programma?

```
package main

import "fmt"

func main() {
    var a []int
    a = []int{0, 1, 2, 3, 4, 5, 6}

    var b []int
```

```
    b = a[2:4]

    b[0] = a[0]
    b[len(b)-1] = a[0]

    fmt.Println(a)
}
```

(Slice) Qual è l'output?

Qual è l'output di questo programma?

```
package main

import "fmt"

func main() {
    b := []int{1, 2, 3, 4, 5}
    stampa(b)

    modifica(b)
    stampa(b)

    eliminaUltimoElemento(b)
    stampa(b)
}

func stampa(sl []int) {
    for _, v := range sl {
        fmt.Print(v, " ")
    }
    fmt.Println()
}
func modifica(sl []int) {
    for i := range sl {
        sl[i] *= 2
    }
}

func eliminaUltimoElemento(sl []int) {
    sl = sl[:len(sl)-1]
}
```

(Slice) Qual è l'output?

Qual è l'output del seguente programma?

```go
package main

import (
    "fmt"
)

const Dimensione = 10

func main() {

    var a []int

    for i := 0; i < Dimensione; i++ {
        a = append([]int{i + 1}, a...)
    }

    fmt.Println(a)

    b := make([]int, Dimensione)

    copy(b, a[Dimensione/2:])

    fmt.Println(b)

}
```

(Riga di comando) Qual è l'output?

Supponendo che l'utente inserisca da **riga di comando** i valori `1 a 5.6 ciao true`, cosa dovrebbe produrre in output il seguente programma?

```go
package main

import (
    "os"
    "fmt"
)

func main() {
    fmt.Printf("TIPO os.Args: %T\n", os.Args)
    for i, v := range os.Args {
        fmt.Printf("os.Args[%d]: TIPO = %T - VALORE = %s\n", i, v, v)
    }
}
```

Esercizi Pratici

0) Occorrenze

Definire una funzione `Occorrenze(s string) [26]int` che data una stringa, restituisca un array con le occorrenze delle 26 lettere dell'alfabeto contenute nella stringa (non facendo distinzione tra minuscole e maiuscole)

Suggerimento: E' possibile trasformare una stringa in minuscolo utilizzando la funzione `strings.ToLower(s)`.

Esempio d'esecuzione:

```
$ go run esercizio_0.go Ciaocomestai?

a 2 b 0 c 2 d 0 e 1 f 0 g 0 h 0 i 2 j 0 k 0 l 0 m 1 n 0 o 2 p 0 q 0 r 0 s
```

1) Stampa in ordine inverso

Scrivere un programma che, dopo aver letto da **standard input** un numero intero `n`, chiede all'utente di inserire `n` numeri interi (sempre da **standard input**).

Il programma deve stampare gli `n` numeri interi in ordine inverso rispetto a quello di inserimento.

Esempio d'esecuzione:

```
$ go run stampa_rovescio.go
9
Inserisci 9 numeri:
1 -12 3 -4 5 -6 7 -7 9
Numeri in ordine inverso:
9 -7 7 -6 5 -4 3 -12 1

$ go run stampa_rovescio.go
5
Inserisci 5 numeri:
1 2 3 4 5
Numeri in ordine inverso:
5 4 3 2 1
```

2) Controlla sequenza

Scrivere un programma che legga da **riga di comando** una sequenza di valori intervallati da caratteri di spaziatura.

Il primo valore che definisce la sequenza (da sinistra verso destra) è in posizione 0, il secondo in posizione 1, etc.

La sequenza è valida se: 1. Tutti i valori letti rappresentano dei numeri interi. 2. Ciascun numero che appare in una posizione *dispari* all'interno della sequenza è *minore* del numero che lo precede. 3. Fatta eccezione per il numero che appare in posizione 0, ciascun numero che appare in una posizione *pari* all'interno della sequenza è *maggiore* del numero che lo precede.

Nel caso in cui la sequenza letta sia valida, il programma deve stampare:

`Sequenza valida.`

In caso contrario, il programma deve stampare:

`Valore in posizione POSIZIONE non valido.`

dove `POSIZIONE` è la posizione del primo valore che invalida la sequenza.

Ad esempio, se la sequenza di valori letta da **riga di comando** fosse:

`5 4 9 abc 6`

il programma deve stampare:

`Valore in posizione 3 non valido.`

Si assuma che la sequenza di valori letta da **riga di comando** sia definita da almeno un valore.

Esempio d'esecuzione:

```
$ go run controlla_sequenza.go mamma mia!
Valore in posizione 0 non valido.

$ go run controlla_sequenza.go 5 4 9 2 6
Sequenza valida.

$ go run controlla_sequenza.go 5 5 9 2 6
Valore in posizione 1 non valido.

$ go run controlla_sequenza.go 5 4 9 -2 -6
Valore in posizione 4 non valido.
```

3) Filtra e moltiplica

Scrivere un programma che legga da **riga di comando** una sequenza di valori e stampi a video il risultato della moltiplicazione tra i valori che rappresentano numeri interi.

Esempio d'esecuzione:

```
$ go run prodotto.go 4 3
Il risultato della moltiplicazione tra i numeri interi è 12
```

```
$ go run prodotto.go 6
Il risultato della moltiplicazione tra i numeri interi è 6

$ go run prodotto.go 1 3 a 5 ciao 2
Il risultato della moltiplicazione tra i numeri interi è 30
```

4) Minimo, massimo e valor medio

Scrivere un programma che legga da **riga di comando** una sequenza di valori e stampi a video il valore minimo, massimo e medio tra i valori letti che rappresentano numeri interi.

Oltre alla funzione `main()`, devono essere definite ed utilizzate almeno le seguenti funzioni: * una funzione `LeggiNumeri() (numeri []int)` che restituisce un valore `numeri` di tipo `[]int` in cui sono memorizzati i valori numerici interi specificati a **riga di comando**; * una funzione `Minimo(sl []int) int` che riceve in input un valore `[]int` nel parametro `sl` e restituisce il minimo valore intero presente in `sl`. * una funzione `Massimo(sl []int) int` che riceve in input un valore `[]int` nel parametro `sl` e restituisce il massimo valore intero presente in `sl`. * una funzione `Media(sl []int) float64` che riceve in input un valore `[]int` nel parametro `sl` e restituisce un valore reale pari alla media aritmetica dei valori interi presenti in `sl`.

Esempio d'esecuzione:

```
$ go run min_max_media.go 1 ciao 2 pippo 3 4
Minimo: 1
Massimo: 4
Valore medio: 2.50

$ go run min_max_media.go -1 10 6 fine
Minimo: -1
Massimo: 10
Valore medio: 5.00

$ go run min_max_media.go tre -1 numeri: -2 -4
Minimo: -3
Massimo: -1
Valore medio: -2.33
```

5) Fattoriale

Definizione: Si definisce fattoriale di un numero intero positivo, il prodotto dei numeri interi positivi minori o uguali a tale numero. Il fattoriale di `k` è uguale a `1*2*3*...*(k-3)*(k-2)*(k-1)*k`.

Scrivere un programma che legga da **riga di comando** un numero intero `n` e stampi a video il fattoriale di tutti i numeri compresi tra `1` e `n` (estremi inclusi).

Oltre alla funzione `main()`, devono essere definite ed utilizzate almeno le seguenti funzioni: * una funzione `Fattoriali(n int) (f []int)` che riceve in input un valore `int` nel parametro `n` e restituisce il valore `f` di tipo `[]int` in cui in `f[0]` è memorizzato il fattoriale di `1`, in `f[1]` è memorizzato il fattoriale di `2`, ..., in `f[n-1]` è memorizzato il fattoriale di `n`.

Esempio d'esecuzione:

```
$ go run fattoriale.go 2
Fattoriali: [1 2]

$ go run fattoriale.go 3
Fattoriali: [1 2 6]

$ go run fattoriale.go 10
Fattoriali: [1 2 6 24 120 720 5040 40320 362880 3628800]
```