Laboratorio 8 - 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?

Analizziamo l'output del seguente programma.

```
package main

import "fmt"

func main() {
    a := [...]int{1, 2, 3, 4, 5, 6, 7}

    fmt.Printf("a - %T: %v\n", a, a)

    sl1 := a[:] // slicing
    sl2 := sl1[1:3]

    fmt.Printf("len(sl1) = %d, cap(sl1) = %d\n", len(sl1), cap(sl1))
    fmt.Printf("sl1 - %T: %v\n", sl1, sl1)
    fmt.Printf("len(sl2) = %d, cap(sl2) = %d\n", len(sl2), cap(sl2))
    fmt.Printf("sl2 - %T: %v\n", sl2, sl2)

    sl2 = sl2[:len(sl2)+1]   // reslicing
    fmt.Printf("\nsl2 - %T: %v\n", sl2, sl2)

    sl2 = sl2[:cap(sl2)]
    fmt.Printf("sl2 - %T: %v\n", sl2, sl2)

    elem, sl1 := sl1[0], sl1[1:]
    fmt.Printf("\nelem - %T: %v\n", elem, elem)
    fmt.Printf("sl1 - %T: %v\n", sl1, sl1)

    /* una slice s non può essere modificata per accedere ad elementi
       dell'array (a cui si riferisce) che precedono quello contenuto in
       sl[0]; l'istruzione sl = sl[-1:] genera un errore */

}
```

(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?

```
package main

import (
    "fmt"
)

const Dimensione = 10
```

```go
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.

Suggerimento: Per creare dinamicamente una slice si utilizzi la funzione `make()`.

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 (2)

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 (1)

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

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

Suggerimento: Il numero di valori interi che il programma deve considerare è pari a `len(os.Args)-1`.

Nota: Per creare dinamicamente una slice, si utilizzi la funzione `make`.

Esempio d'esecuzione:

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

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

$ go run min_max_media.go -1 -2 -3
Minimo: -3
Massimo: -1
Valore medio: -2.00
```

5) Minimo, massimo e valor medio (2)

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`.

Nota: Per creare dinamicamente una slice, si utilizzi la funzione `append`.

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
```

6) 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`.

Nota: Per creare dinamicamente una slice, si utilizzi la funzione `make`.

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]
```

7) Somma di prodotti pari

Scrivere un programma che:

- legga da **riga di comando** una sequenza di numeri interi;
- stampi a video il risultato della somma dei prodotti pari associati alle coppie non ordinate di numeri che si possono definire a partire dai numeri letti (data la coppia non ordinata di numeri `(numero_1, numero_2)`, il valore del prodotto associato è `numero_1 * numero_2`).

Esempio: Se `10 1 31 4` è la sequenza letta, le coppie non ordinate di numeri che si possono definire a partire dai numeri letti sono: `(10, 1); (10, 31); (10, 4); (1, 31); (1, 4); (31, 4)`. Di queste, quelle il cui prodotto è pari sono: `(10, 1); (10, 31); (10, 4); (1, 4); (31, 4)`. La somma dei prodotti pari è `488`.

Oltre alla funzione `main()`, devono essere definite ed utilizzate almeno le seguenti funzioni: * una funzione `Calcola(sl []int) int` che riceve in input un valore `[]int` nel parametro `sl` e restituisce un valore di tipo `int` pari alla somma dei prodotti pari associati alle coppie non ordinate di numeri che si possono definire a partire dai numeri presenti in `sl`.

Nota: Per creare dinamicamente una slice, si utilizzi la funzione `make`.

Esempio d'esecuzione:

```
$ go run prodotti_pari.go 1 2 3 4 5 6
La somma è 152

$ go run prodotti_pari.go 1 2 3 4 5
La somma è 62
```

8) Filtra voti

Scrivere un programma che: * legga da **riga di comando** una sequenza di valori (i valori numerici interi che compaiono all'interno della sequenza rappresentano voti in una scala di valutazione tra 0 e 100; i valori numerici interi superiori a 60 corrispondono a voti sufficienti); * stampi a video le due sottosequenze di valori numerici interi che corrisponodo rispettivamente a voti insufficienti e sufficienti.

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 `FiltraVoti(voti []int) (sufficienti, insufficienti []int)` che riceve in input un valore `[]int` nel parametro `voti` e restituisce due valori di tipo `[]int`, `sufficienti` e

`insufficienti`, in cui sono memorizzati rispettivamente i voti sufficienti e insufficienti presenti in `voti`.

Esempio d'esecuzione:

```
$ go run filtro.go 80 75 60 55
Voti sufficienti: [80 75 60]
Voti insufficienti: [55]

$ go run filtro.go 100 98 59 40
Voti sufficienti: [100 98]
Voti insufficienti: [59 40]
```

9) Numeri casuali

Scrivere un programma che: 1) Legga da **riga di comando** un numero intero `soglia`; 2) Generi in modo casuale una sequenza di lunghezza arbitraria di numeri interi compresi nell'intervallo che va da 0 a 100, estremi inclusi. Il processo di generazione si interrompe quando viene generato un numero inferiore a `soglia`. 3) Stampi a video tutti i numeri generati. 4) Stampi a video tutti i numeri generati superiori a `soglia`.

Oltre alla funzione `main()`, devono essere definite ed utilizzate almeno le seguenti funzioni: * una funzione `Genera(soglia int) []int` che riceve in input un valore `int` nel parametro `soglia` e restituisce un valore di tipo `[]int` in cui è memorizzata una sequenza di lunghezza arbitraria di numeri interi, generata in base alle specifiche di cui al punto 2.

Suggerimento: per generare in modo casuale un numero intero, potete utilizzare le funzioni dei package `math/rand` e `time` come mostrato nel seguente frammento di codice:

```
/* inizializzazione del generatore di numeri casuali */
rand.Seed(int64(time.Now().Nanosecond()))
/* generazione di un numero casuale compreso nell'intervallo
   che va da 0 a 99 (estremi inclusi) */
numeroGenerato := rand.Intn(100)
```

Esempio d'esecuzione:

```
$ go run numeri_random.go 20
Valori generati [21 72 44 64 30 13]
Valori sopra soglia: [21 72 44 64 30]
```

10) Somma unici

Scrivere un programma che legga da **riga di comando** una sequenza di valori e stampi a video la somma dei valori letti che rappresentano numeri interi eche

compaiono nella sequenza una sola volta.

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 `Occorrenze(numeri []int, n int) int` che riceve in input un valore `[]int` nel parametro `numeri` e restituisce un valore `int` pari al numero di occorrenze di `n` in `numeri`.

Esempio: Supponendo di leggere da **riga di comando** la sequenza `1 2 a 4 ciao 3 2 1 5`, il programma deve stampare `12`, ovvero la somma dei numeri `4`, `3` e `5`. Se la sequenza fosse `4 3 5 non_conto 4 2 2 3 2`, l'output sarebbe invece `5`.

Esempio d'esecuzione:

```
$ go run somma_unici.go 1 2 % 4 3 2 1 5
12

$ go run somma_unici.go 4 3 5 4 2 2 3 2
5

$ go run somma_unici.go 1 2 sarà zero 1 2
0

$ go run somma_unici.go 1 2 3 2 2 2
4

$ go run somma_unici.go che 10 4 7 12 4 12 sfortuna
17
```

11) Somma di prodotti

Scrivere un programma che legga da **riga di comando** una sequenza di numeri interi di lunghezza pari. Data la sequenza, il programma deve moltiplicare ciascun numero in una posizione di indice pari per il successivo numero in posizione di indice dispari e sommare i prodotti ottenuti.

Esempio: Se `10 2 3 4 5 6` è la sequenza letta, allora la somma calcolata deve essere `10*2 + 3*4 + 5*6 = 62`.

Il programma deve infine stampare a video il valore della somma calcolata.

Oltre alla funzione `main()`, devono essere definite ed utilizzate almeno le seguenti funzioni: * una funzione `Calcola(sl []int) int` che riceve in input un valore `[]int` nel parametro `sl` e restituisce un valore di tipo `int` pari alla somma dei prodotti ottenuti moltiplicando ciascun numero in una posizione di indice pari di `sl` per il successivo numero in posizione di indice dispari.

Nota: Per creare dinamicamente una slice, si utilizzi la funzione `make`.

Esempio d'esecuzione:

```
$ go run somma_prodotto.go 1 2 3 4 5 6
La somma è 44

$ go run somma_prodotto.go 7 3 1 8
La somma è 29
```

12) Primi

Definizione: Un numero naturale è primo se è divisibile solo per se stesso e per 1.

Scrivere un programma che legga da **riga di comando** un numero intero `numero` e stampi tutti i numeri *primi* ottenibili rimuovendo al più `3` cifre consecutive tra quelle che definiscono `numero`.

In particolare, i numeri primi devono essere stampate in ordine crescente (cioè dal più piccolo al più grande).

Ad esempio, se il numero intero letto da **riga di comando** fosse:

```
5899
```

i numeri ottenibili rimuovendo al più `3` cifre consecutive tra quelle che definiscono `5899` sarebbero:

```
5
9
58
59
99
589
589
599
899
5899
```

Si assuma che il valore specificato a riga di comando sia nel formato corretto e, in particolare, sia un intero maggiore o uguale a 1000.

Oltre alle funzione `main()`, devono essere definite ed utilizzate almeno le seguenti funzioni: * una funzione `ÈPrimo(n int) bool` che riceve in input un valore (di tipo) `int` nel parametro `n` e restituisce `true` se `n` è primo (i.e., se il valore di `n` rappresenta un numero primo) e `false` altrimenti.

Suggerimento: I numeri primi possono essere ordinati in senso crescente utilizzando la funzione `sort.Ints(a []int)` del package `sort`.

Esempio d'esecuzione:

```
$ go run primi.go 5899
5
59
599

$ go run primi.go 5894457
4457
5857
5897
594457

$ go run primi.go 10113
13
13
101
103
113
113
113
1013
1013

$ go run primi.go 2468
2
```