Laboratorio 8 - Array, Slice, Riga di comando (2)

1) 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`.

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
```

2) 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]
```

3) 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:

```go
import (
   "math/rand"
   "time"
)
/* 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]
```

4) 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
```

5) 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.

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
```

6) 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
```