Esercizio filtro

Scrivere un programma che riceva in input da riga di comando una stringa formata da caratteri arbitrari (nel sistema di codifica UNICODE) e la utilizzi per stampare su standard output un rombo come da esempi di esecuzione. Si assuma che la stringa in input contenga un numero dispari di caratteri.

Esempio d'esecuzione:

```
$ go run esercizio_filtro.go abc
 a
abc
 a

$ go run esercizio_filtro.go bignè
  b
 big
bignè
 big
  b

$ go run esercizio_filtro.go èòàéùù°
   è
  èòà
 èòàéù
èòàéùù°
 èòàéù
  èòà
   è
```

Test automatico

L'esercizio filtro è considerato esatto **solo se** rispetta le specifiche date e **solo se** passa il test automatico fornito

Inizializzazione del test automatico Al fine di poter eseguire il test automatico, è prima necessario eseguire **una volta sola** ed **esclusivamente nella cartella relativa a questo esercizio** il comando: `go mod init esercizio_filtro` ottenendo il seguente output:

```
$ go mod init esercizio_filtro
go: creating new go.mod: module esercizio_filtro
go: to add module requirements and sums:
        go mod tidy
```

E' necessario, successivamente, creare l'eseguibile relativo al codice con il comando:

```
$ go build esercizio_filtro.go
```

Esecuzione del test automatico Successivamente sarà possibile eseguire il test automatico utilizzando il comando `go test`. L'esercizio passa il test automatico se il comando `go test` restituisce, alla fine:

```
PASS
ok
```

Esercizio 1

Dati due vettori `a` e `b`, cioè, due sequenze di numeri di lunghezza `la` ed `lb` rispettivamente (rappresentate come slice di interi), una forma di prodotto vettoriale è quella che produce una matrice `c` di dimensione `la x lb` (rappresentata come slice di slice) il cui elemento `[i,j]` è pari ad `a[i] * b[j]`, dove `i` varia da `0` a `la-1` e `j` varia da `0` a `lb-1`.

Se ad esempio `a` è:

```
[1 2 3]
```

e `b` è

```
[1 2]
```

allora `c = a x b` è la matrice `3 x 2` uguale a

```
[1 2]
[2 4]
[3 6]
```

Scrivere un programma che legga da **standard input** due righe contenenti ciascuna una sequenza (di lunghezza arbitraria) di valori interi e produca in output la matrice risultante dal prodotto dei vettori (slice) corrispondenti alle due righe.

Per la stampa di ciascuna riga della matrice si usi la rappresentazione standard usata in Go per le slice.

Oltre alla funzione `main()` si definisca la funizione:

- `ProdottoVettoriale(a, b []int) [][]int` che dati due vettori `a` e `b` (in forma di slice) restituisca la matrice (slice di slice) corrispondente al prodotto `a x b`

Esempio d'esecuzione:

```
$ go run esercizio_1.go
1 2 3
1 2
[1 2]
```

```
[2 4]
[3 6]

$ go run esercizio_1.go
4 2 18 31
13 7 19 1 6
[52 28 76 4 24]
[26 14 38 2 12]
[234 126 342 18 108]
[403 217 589 31 186]
```

Test automatico

L'esercizio è considerato completamente esatto se rispetta le specifiche date e se passa il test automatico fornito.

Inizializzazione del test automatico Al fine di poter eseguire il test automatico, è prima necessario eseguire **una volta sola** ed **esclusivamente nella cartella relativa a questo esercizio** il comando: `go mod init esercizio_1` ottenendo il seguente output:

```
$ go mod init esercizio_1
go: creating new go.mod: module esercizio_1
go: to add module requirements and sums:
        go mod tidy
```

E' necessario, successivamente, creare l'eseguibile relativo al codice con il comando:

```
$ go build esercizio_1.go
```

Esecuzione del test automatico Successivamente sarà possibile eseguire il test automatico utilizzando il comando `go test`. L'esercizio passa il test automatico se il comando `go test` restituisce, alla fine:

```
PASS
ok
```

Esercizio 2

Scrivere un programma che legga da **riga di comando** due numeri naturali (interi maggiori di zero), rispettivamente `N` e `k`.

Il programma deve stampare a video tutti i numeri *abbondanti* ottenibili rimuovendo `k` cifre decimali *consecutive* da `N`. I numeri dovranno essere stampati in **ordine crescente**.

Definizione: Un numero abbondante è un numero naturale minore della somma dei suoi divisori interi (escludendo sé stesso). Per esempio, `12` è un numero abbondante poiché inferiore alla somma dei suoi divisori: `1+2+3+4+6=16`.

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

- una funzione `GeneraNumeri(N, k int) []int` che riceve in input due valori `int` nei parametri `N` e `k`, e restituisce un valore `[]int` in cui sono memorizzati tutti i numeri interi positivi ottenibili rimuovendo `k` cifre decimali consecutive da `N`;
- una funzione `FiltraNumeri(sl [] int) []int` che riceve in input un valore `[]int` nel parametro `sl` e restituisce un valore `[]int` in cui sono memorizzati tutti i numeri abbondanti presenti in `sl`.

Si assuma che:

- i valori letti da **riga di comando** siano specificati nel formato corretto;
- il numero di cifre decimali che definiscono `N` sia maggiore di `k`.

Esempio d'esecuzione:

```
$ go run esercizio_2.go 12834045496 7
1296
1496
5496

$ go run esercizio_2.go 12834045496 6
12496
12834
12896
15496
45496

$ go run esercizio_2.go 12834045496 2
128045496
128340456
128340496
128340496
128345496
128345496
134045496
```

Test automatico

L'esercizio è considerato completamente esatto se rispetta le specifiche date e se passa il test automatico fornito.

Inizializzazione del test automatico Al fine di poter eseguire il test automatico, è prima necessario eseguire **una volta sola** ed **esclusivamente nella cartella relativa a questo esercizio** il comando: `go mod init esercizio_2` ottenendo il seguente output:

```
$ go mod init esercizio_2
go: creating new go.mod: module esercizio_2
go: to add module requirements and sums:
        go mod tidy
```

E' necessario, successivamente, creare l'eseguibile relativo al codice con il comando:

```
$ go build esercizio_2.go
```

Esecuzione del test automatico Successivamente sarà possibile eseguire il test automatico utilizzando il comando `go test`. L'esercizio passa il test automatico se il comando `go test` restituisce, alla fine:

```
PASS
ok
```

Esercizio 3

Scrivere un programma che legga da standard input una sequenza di righe di testo e termini la lettura quando, premendo la combinazione di tasti Ctrl+D, viene inserito da standard input l'indicatore End-Of-File (EOF).

Ogni riga di testo letta rappresenta una variazione di un prodotto in un magazzino nel seguente formato:

```
nome_prodotto;id_prodotto;variazione
```

dove `nome_prodotto` è il nome del prodotto (stringa arbitraria), `id_prodotto` è un codice alfanumerico che definisce un prodotto, e `variazione` è un numero intero che specifica la quantità del prodotto: aggiunta nel magazzino se positivo, o sottratta dal magazzino se negativo.

Si definiscano i seguenti tipi:

- `Prodotto`: contenente i parametri `Nome` e `Codice`, nei quali saranno memorizzati i rispettivi dati del prodotto.
- `Magazzino`: che dovrà associare ciascun `Prodotto` inserito con la quantità presente nel magazzino

Il programma deve implementare almeno le seguenti funzioni:

- `NuovoMagazzino() Magazzino` che dovrà inizializzare e restituire una nuova variabile Magazzino vuota

- `StringProdotto(p Prodotto) string` che riceve un prodotto e restituisce una stringa formattata che lo descrive; Esempio:

  `Codice: 4765667, Prodotto: cuscino FJÄDERMOLN`

- `CreaProdotto(nome string, codice string) Prodotto` che dati gli estremi di un prodotto, crea e restituisce una nuova variabile Prodotto
- `NumeroProdotti(m Magazzino) int` che dato il magazzino restituisce il numero di prodotti in esso contenuto (0 se il magazzino è vuoto)
- `Quantità(m Magazzino, p Prodotto) int` che dati il magazzino ed un prodotto restituisce il numero di occorrenze del prodotto in magazzino
- `ModificaProdotto(m Magazzino, p Prodotto, variazione int) (Magazzino, bool)` che dovrà modificare la quantità del prodotto `p` nel magazzino `m`, applicando la variazione specificata, e restituire il magazzino così modificato. Inoltre dovrà essere restituito un valore booleano che specifica se l'operazione è valida. Nello specifico:

  ```
  * se il prodotto non è presente in magazzino e la variazione
    risultasse positiva, allora il prodotto viene inserito in
    magazzino
  * se la quantità del prodotto dopo la variazione risultasse
    positiva, allora la modifica sarà applicata e l'operazione
    sarà valida
  * se la quantità del prodotto dopo la variazione risultasse 0,
    allora il prodotto dovrà essere eliminato dal Magazzino e
    l'operazione sarà valida
  * se la quantità del prodotto dopo la variazione risultasse
    negativa, allora la modifica NON dovrà essere applicata.
    La funzione dovrà restituire il Magazzino NON modificato e
    l'operazione NON sarà valida.
  ```

Il programma deve stampare su standard output:

- se tutte le operazioni di modifica sono valide, il numero di prodotti presenti in magazzino e l'elenco finale dei nomi dei prodotti con le corrispondenti quantità in magazzino *ordinate alfabeticamente in modo crescente secondo il codice prodotto.*
- altrimenti, non appena viene letta una riga che introduce un'operazione di modifica non valida, UNICAMENTE: il numero della riga alla quale si è verificato l'errore, il nome del prodotto, la quantità del prodotto in magazzino e la variazione richiesta che ha causato l'errore.

Tutte le funzioni richieste dovranno essere utilizzate nel programma.

Esempio d'esecuzione:

```
$ go run esercizio_3.go < prod1.txt
```

```
Il magazzino contiene 4 prodotti.
Codice: 4765667, Prodotto: cuscino FJÄDERMOLN, Quantità: 31
Codice: 49328402, Prodotto: base tv BESTÅ, Quantità: 26
Codice: 60354989, Prodotto: tappeto STENLILLE, Quantità: 32
Codice: 78549203, Prodotto: sedia ÖRFJÄLL, Quantità: 39

$ go run esercizio_3.go < prod2.txt
Errore alla riga 19: Codice: 49328402, Prodotto: base tv BESTÅ - disponibilità 33,
richiesta -73
```

Test automatico

L'esercizio è considerato completamente esatto se rispetta le specifiche date e se passa il test automatico fornito.

Inizializzazione del test automatico Al fine di poter eseguire il test automatico, è prima necessario eseguire **una volta sola** ed **esclusivamente nella cartella relativa a questo esercizio** il comando: `go mod init esercizio_3` ottenendo il seguente output:

```
$ go mod init esercizio_3
go: creating new go.mod: module esercizio_3
go: to add module requirements and sums:
        go mod tidy
```

E' necessario, successivamente, creare l'eseguibile relativo al codice con il comando:

```
$ go build esercizio_3.go
```

Esecuzione del test automatico Successivamente sarà possibile eseguire il test automatico utilizzando il comando `go test`. L'esercizio passa il test automatico se il comando `go test` restituisce, alla fine:

```
PASS
ok
```