Esercizio filtro

Scrivere un programma che legga da **riga di comando** una parola. Stampare a schermo la parola inserita in orizzontale ed in verticale in modo che le due parole si intersechino e condividano la lettera centrale. Sia in verticale che in orizzontale la parola deve essere stampata al contrario. Si assuma che la parola inserita abbia un numero dispari di caratteri.

Esempio d'esecuzione:

```
$ go run esercizio_filtro.go MANDARINO
    O
    N
    I
    R
ONIRADNAM
    D
    N
    A
    M

$ go run esercizio_filtro.go bignè
  è
  n
èngib
  i
  b

$ go run esercizio_filtro.go ƤⱤŌĢɭăƁ
   Ɓ
   ă
   ɭ
ƁăɭĢŌⱤƤ
   Ō
   Ɽ
   Ƥ
```

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 filtro` ottenendo il seguente output:

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

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 `PASS`, come nel seguente output:

```
$ go test
PASS
ok
```

Invece, nel caso in cui l'output dovesse restituire `FAIL`, come nel seguente esempio, significa che almeno un caso tra quelli riportati nell'esempio d'esecuzione non è stato eseguito in modo corretto, ed il filtro è considerato **errato**. In tal caso `go test` restituirà anche l'output atteso dal programma e l'output effettivo che il programma produce. Questo dato è utile per capire in cosa il risultato del programma differisce dall'output atteso.

```
$ go test
--- FAIL: TestFiltro (0.00s)
        esercizio_filtro_test.go:40:
                ...
```

Esercizio 1

Scrivere un programma che legga da **riga di comando** due stringhe rappresentanti due rette nella forma ax+by+c=0.

Esempio: `-5x+2y-3=0`

Si assuma che per ciascuna retta i coefficienti a,b,c siano interi e diversi da 0.

Verificare se le due rette sono parallele stampando in tal caso "le due rette sono parallele", altrimenti "le due rette non sono parallele".

Ricordiamo che due rette sono parallele se hanno il coefficiente angolare (m = -a/b) uguale tra loro.

Verificare l'uguaglianza del coefficiente angolare a meno di una soglia epsilon = 0.0001

Esempio d'esecuzione:

```
$ go run esercizio_1.go 5x-2y+3=0 -10x+4y-1=0
le due rette sono parallele

$ go run esercizio_1.go 2x+4y+5=0 5x-3y+2=0
le due rette non sono parallele

$ go run esercizio_1.go -1x+1000y+3=0 -1x+1001y-4=0
le due rette sono parallele
```

Esercizio 2

Scrivere un programma che legga da riga di comando una stringa costituita di sole cifre numeriche e stampi su standard output tutte le sottostringhe che siano **numeri primi** maggiori o uguali a 100. Ricordiamo che i primi sono numeri divisibili solo per se stessi o per 1.

I numeri devono essere stampati in ordine crescente, ed evitando ripetizioni. Devono inoltre essere stampati sulla stessa linea e separati da virgole (come riportato negli esempi di esecuzione).

In particolare, si implementi almeno la seguente funzione:

- `èPrimo(n int) bool` che, dato un numero intero `n`, restituisca `true` se il numero è primo, `false` altrimenti.

Esempio d'esecuzione:

```
$ go run esercizio_2.go 919103011
103, 191, 919, 3011, 9103, 10301, 19103011

$ go run esercizio_2.go 1234567890
4567, 23456789

$ go run esercizio_2.go 7300037
730003, 7300037

$ go run esercizio_2.go 248262

$ go run esercizio_2.go 8604
```

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:

```
id_prodotto;nome prodotto;variazione
```

dove `id_prodotto` è un codice alfanumerico che definisce un prodotto, `nome prodotto` è il nome del prodotto (stringa arbitraria con spazi) 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
- `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. Specificatamente:
    - se la quantità del prodotto dopo la variazione risulterebbe positiva, allora la modifica sarà applicata e l'operazione sarà valida
    - se la quantità del prodotto dopo la variazione risulterebbe 0, allora il prodotto dovrà essere eliminato dal Magazzino e l'operazione sarà valida
    - se la quantità del prodotto dopo la variazione risulterebbe 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, l'elenco finale dei nomi dei prodotti e le corrispondenti quantità in magazzino
- se una qualsiasi riga letta introduce un'operazione di modifica non valida, 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.

Esempio d'esecuzione:

```
$ go run esercizio_3.go < prod1.txt
Codice: 4765667, Prodotto: cuscino FJÄDERMOLN, Quantità: 31
Codice: 78549203, Prodotto: sedia ÖRFJÄLL, Quantità: 39
Codice: 60354989, Prodotto: tappeto STENLILLE, Quantità: 32
Codice: 49328402, Prodotto: base tv BESTÅ, Quantità: 26

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