Esercizio filtro

Scrivere un programma che legga da **riga di comando** una parola ed un carattere e, come mostrato nell'**Esempio di esecuzione**, stampi a video il numero di caratteri della parola uguali al carattere dato.

Si assuma che i valori letti da **riga di comando** siano specificati nel formato corretto.

Esempio d'esecuzione:

```
$ go run esercizio_filtro.go carcadè c
2

$ go run esercizio_filtro.go carcadè è
1

$ go run esercizio_filtro.go assassino s
4

$ go run esercizio_filtro.go assassino z
0
```

Test automatico:

L'esercizio filtro è considerato esatto **solo se** eseguendo il comando `go test esercizio_filtro.go esercizio_filtro_test.go` si ottiene un output simile al seguente:

```
$ go test esercizio_filtro.go esercizio_filtro_test.go

ok      command-line-arguments  0.002s
```

Invece, nel caso in cui l'output dovesse essere simile al seguente

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

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**.

Esercizio 1

Scrivere un programma che legga da **riga di comando** una sequenza di numeri reali di lunghezza arbitraria.

Come mostrato nell'**Esempio d'esecuzione**, il programma deve stampare a video:

- il carattere `">"` per ciascun numero maggiore di quello precedente (a meno di un valore `EPSILON`);
- il carattere `"<"` per ciascun numero minore di quello precedente (a meno un valore `EPSILON`);
- il carattere `"="` per ciascun numero uguale a quello precedente (a meno di un valore `EPSILON`).

Il valore `EPSILON` è il primo numero della sequenza inserita da **riga di comando** e non è da considerarsi ai fini della generazione della sequenza di caratteri da stampare a video.

Si ricordi che, dati due numeri reali `x` e `y`:

- `x` è maggiore di `y` a meno di un valore `EPSILON` se `x - y > EPSILON`;
- `x` è uguale a `y` a meno di un valore `EPSILON` se `|x - y| <= EPSILON`;
- `x` è minore di `y` a meno di un valore `EPSILON` se `x - y < -EPSILON`.

Si assuma che i valori letti da **riga di comando** siano specificati nel formato corretto.

Esempio d'esecuzione:

```
$ go run esercizio_1.go 0.01 5.4 5.3 5.6 7.0 6.999
<>>=

$ go run esercizio_1.go 2 3 4 5 6
===

$ go run esercizio_1.go 0.01 0.1 0 -0.1
<<

$ go run esercizio_1.go 0.01 -0.1 0 0.1
>>
```

Esercizio 2

Scrivere un programma che:

- legga da **standard input** una stringa `s` costituita da cifre decimali;
- stampi a schermo, dalla più lunga alla più corta, tutte le sottosequenze della stringa `s` nelle quali le cifre sono in ordine crescente (si considerino solamente sottosequenze di almeno 2 cifre).

Come mostrato nell'**Esempio d'esecuzione**, ciascuna sottosequenza deve essere stampata un'unica volta, riportando il numero di volte in cui la sottosequenza appare in `s`.

Se la stringa `s` letta da **standard input** non è costituita solamente da cifre decimali, il programma termina senza stampare nulla.

Oltre alla funzione `main()`, deve essere definita ed utilizzata almeno la funzione `Sottostringhe(s string) map[string]int`, che riceve in input un valore `string` nel parametro `s`, e restituisce un valore `map[string]int` in cui, per ogni sottosequenza di cifre ordinate (in senso crescente) presente in `s` di almeno 2 cifre, è memorizzato il numero di volte in cui la sottosequenza appare in `s`.

Esempio d'esecuzione:

```
$ go run esercizio_2.go
123456
output:
123456 1
12345 1
23456 1
1234 1
2345 1
3456 1
123 1
456 1
234 1
345 1
34 1
45 1
56 1
12 1
23 1

$ go run esercizio_2.go
654321
output:

$ go run esercizio_2.go
123121212
output:
123 1
23 1
12 4

$ go run esercizio_2.go
acc23

$ go run esercizio_2.go
01010101
output:
01 4
```

Esercizio 3

Un'utenza di telefonia mobile è identificata dal numero del telefono mobile e da un codice che identifica la scheda SIM che gli corrisponde.

Ne consegue che, in un dato istante temporale, ad un numero di telefono mobile possa corrispondere una ed una sola scheda SIM.

Comunque, nel corso del tempo, ad un numero di telefono mobile possono corrispondere diversi codici relativi a schede SIM.

L'utenza di telefonia mobile attiva è quella caratterizzata dal codice più recente relativo ad una scheda SIM.

Parte 1

Scrivere un programma che:

- legge da **standard input** una sequenza di righe di testo;
- termina 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 è nel formato

```
NUMERO_TELEFONO;CODICE_SIM
```

dove `NUMERO_TELEFONO` è il numero telefonico (valore composto da sole cifre) di una utenza, mentre `CODICE_SIM` è il codice che identifica la scheda SIM associata (valore composto da caratteri alfanumerici).

Ciascuna riga di testo descrive quindi un'utenza di telefonia mobile. Le utenze di telefonia mobile sono specificate in ordine cronologico: la prima riga descrive l'utenza di telefonia mobile meno recente, l'ultima riga descrive l'utenza di telefonia mobile più recente.

Si assuma che le righe di testo lette da **standard input** siano nel formato corretto.

Definire la struttura `Utenza` per memorizzare il numero telefonico e il codice della SIM associata.

Implementare le funzioni:

- `LeggiUtenze() (utenze []Utenza)` che:
  i. legge da **standard input** una sequenza di righe di testo, terminando la lettura quando viene letto l'indicatore End-Of-File (EOF);
  ii. restituisce un valore `[]Utenza` nella variabile `utenze` in cui è memorizzata la sequenza di istanze del tipo `Utenza` inizializzate con i valori letti da **standard input**.

Parte 2

Il registro telefonico di un operatore è la lista di tutte le utenze telefoniche passate (non attive) e presenti (attive). Il registro contiene, per ogni numero di telefono, l'elenco di utenze associate in ordine cronologico. Definire il tipo di dato `RegistroTelefonico` per memorizzare le utenze telefoniche lette nella **Parte 1** dell'esercizio.

Implementare le funzioni:

- `InizializzaRegistro() (registro RegistroTelefonico)` che inizializza un'istanza di tipo `RegistroTelefonico` (un registro telefonico vuoto) e la restituisce all'interno della variabile `registro`;
- `AggiungiUtenza(registro RegistroTelefonico, utenza Utenza) (registroAggiornato RegistroTelefonico)` che riceve in input un'istanza di tipo `RegistroTelefonico` nella variabile `registro` e un'istanza di tipo `Utenza` nella variabile `utenza`. La funzione aggiunge l'utenza telefonica `utenza` al registro `registro` e restituisce il registro telefonico aggiornato nella variabile `registroAggiornato`.
- `Identifica(registro RegistroTelefonico, telefono string) (codiceSIM string)` che riceve in input un'istanza di tipo `RegistroTelefonico` nella variabile `registro` e un valore di tipo `string` nella variabile `telefono` (un numero di telefono mobile). La funzione restituisce il codice della SIM (presente in `registro`) corrispondente a `telefono` nella variabile `codiceSIM`. Se in `registro` sono presenti più codici SIM corrispondenti a `telefono`, viene restituito il codice più recente, ovvero quello che nella sequenza di input è letto per ultimo. Se in `registro` non è presente alcun codice SIM corrispondente a `telefono`, viene restituito il valore `""`.

Parte 3

Un operatore vuole avviare un'indagine statistica.

Implementare la funzione:

`NumeroUtenze(registro RegistroTelefonico, telefono string) (n int)` che riceve in input un'istanza di tipo `RegistroTelefonico` nella variabile `registro` e un valore di tipo `string` nella variabile `telefono` (un numero di telefono mobile). La funzione restituisce nella variabile `n` il numero di utenze caratterizzate dal numero di telefono `telefono` presenti in `registro`.

Il programma deve stampare:

- il numero medio di cambi di SIM effettuati da utenti con numero di telefono mobile che inizia con 338;
- il numero massimo di cambi di SIM effettuati da utenti con numero di telefono mobile che inizia con 338;
- la lista dei numeri di telefono che iniziano per 338 che non ha mai effettuato un cambio SIM.

Si noti che il numero di cambi relativi ad un dato numero di telefono mobile è pari al numero di utenze presenti nel registro per quel numero meno uno.

Esempio d'esecuzione:

```
$ more registro1.txt
338145067;73198a2ec2bc83aff9d63c8d3ea41098
339445267;28812f5027d291b1edc054de7f657fa1
347345067;f385a984f6f32c1f188f0e6876c62f50
348145367;5daf1afa90fa043914e3df328bbe850f
338445367;8e70d09a082b888d0aeee4dd07bba310
339045167;afabbbbe686c62201f2146fe66b6f372
...

$ go run esercizio_3.go < registro1.txt
Media cambio SIM: 0.76
Numero massimo cambi SIM: 7
Numeri di telefono senza cambi SIM:
338145067
338045367
338145567
338245267
338045167
338545267
338345267
338245367

$ more registro2.txt
338545567;73198a2ec2bc83aff9d63c8d3ea41098
339445167;28812f5027d291b1edc054de7f657fa1
347545067;f385a984f6f32c1f188f0e6876c62f50
348245267;5daf1afa90fa043914e3df328bbe850f
338145367;8e70d09a082b888d0aeee4dd07bba310
339545167;afabbbbe686c62201f2146fe66b6f372
...

$ go run esercizio_3.go < registro2.txt
Media cambio SIM: 0.77
Numero massimo cambi SIM: 5
Numeri di telefono senza cambi SIM:
338045567
338445367
338445267
338145267
338345367
338245367
338545467
```