Esercizio filtro

Scrivere un programma che legga da **riga di comando** due numeri interi `a` e `b` e stampi a video la somma dei numeri dispari compresi tra `a` e `b` (`a` e `b` esclusi).

Si assuma che la coppia di valori specificata a riga di comando sia nel formato corretto.

Esempio d'esecuzione:

```
$ go run esercizio_filtro.go 7 15
33

$ go run esercizio_filtro.go 3 5
0

$ go run esercizio_filtro.go 11 19
45

$ go run esercizio_filtro.go 1 10
24

$ go run esercizio_filtro.go -2 4
3
```

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 **standard input** una stringa di caratteri e controlli se (la stringa letta) può rappresentare una password ben definita.

Si assuma che, chiaramente, nessun carattere nella stringa può rappresentare un carattere di spaziatura, ossia un carattere il cui codice Unicode, passato come argomento alla funzione `func IsSpace(r rune) bool` del package `unicode`, fa restituire `true` alla funzione.

Una password è ben definita se, considerando la stringa di caratteri che la rappresenta, vengono soddisfatte le seguenti condizioni:

1. la stringa deve avere una lunghezza minima di 12 caratteri;
2. almeno 2 caratteri nella stringa devono rappresentare delle lettere minuscole;
3. almeno 2 caratteri nella stringa devono rappresentare delle lettere maiuscole;
4. almeno 3 caratteri nella stringa devono rappresentare delle cifre decimali;
5. almeno 4 caratteri nella stringa non devono rappresentare lettere o cifre decimali.

Un carattere rappresenta una lettera se il relativo codice Unicode, passato come argomento alla funzione `func IsLetter(r rune) bool` del package `unicode`, fa restituire `true` alla funzione.

Un carattere rappresenta una cifra decimale se il relativo codice Unicode, passato come argomento alla funzione `func IsDigit(r rune) bool` del package `unicode`, fa restituire `true` alla funzione.

Nel caso in cui la stringa letta rappresenti una password ben definita, il programma deve stampare:

```
La pw è ben definita!
```

In caso contrario, il programma deve stampare:

```
La pw non è definita correttamente:
```

ed uno o più dei seguenti messaggi opzionali:

```
- La pw deve avere una lunghezza minima di 12 caratteri
```

(da stampare solo se la condizione 1 non è stata soddisfatta dalla stringa letta)

```
- Almeno 2 caratteri della pw devono rappresentare delle lettere minuscole
```

(da stampare solo se la condizione 2 non è stata soddisfatta)

```
- Almeno 2 caratteri della pw devono rappresentare delle lettere maiuscole
```

(da stampare solo se la condizione 3 non è stata soddisfatta)

```
- Almeno 3 caratteri della pw devono rappresentare delle cifre decimali
```

(da stampare solo se la condizione 4 non è stata soddisfatta)

```
- Almeno 4 caratteri della pw non devono rappresentare lettere o cifre decimali
```

(da stampare solo se la condizione 5 non è stata soddisfatta)

Esempio d'esecuzione:

```
$ go run esercizio_1.go
$ go run esercizio_1.go
pa55Wordlunga
La pw non è definita correttamente:
- Almeno 2 caratteri della pw devono rappresentare delle lettere maiuscole
- Almeno 3 caratteri della pw devono rappresentare delle cifre decimali
- Almeno 4 caratteri della pw non devono rappresentare lettere o cifre decimali

$ go run esercizio_1.go
Qu35t@_E_Un@_Pa55w0rd
La pw è ben definita!

$ go run esercizio_1.go
pW5
La pw non è definita correttamente:
- La pw deve avere una lunghezza minima di 12 caratteri
- Almeno 2 caratteri della pw devono rappresentare delle lettere minuscole
- Almeno 2 caratteri della pw devono rappresentare delle lettere maiuscole
- Almeno 3 caratteri della pw devono rappresentare delle cifre decimali
- Almeno 4 caratteri della pw non devono rappresentare lettere o cifre decimali

$ go run esercizio_1.go
P@55WORD_LUNG@
La pw non è definita correttamente:
- Almeno 2 caratteri della pw devono rappresentare delle lettere minuscole
- Almeno 3 caratteri della pw devono rappresentare delle cifre decimali
- Almeno 4 caratteri della pw non devono rappresentare lettere o cifre decimali

$ go run esercizio_1.go
password
La pw non è definita correttamente:
- La pw deve avere una lunghezza minima di 12 caratteri
- Almeno 2 caratteri della pw devono rappresentare delle lettere maiuscole
- Almeno 3 caratteri della pw devono rappresentare delle cifre decimali
- Almeno 4 caratteri della pw non devono rappresentare lettere o cifre decimali
```

Esercizio 2

Un robot è in grado di muoversi nelle quattro direzioni nord, sud, est e ovest.

In particolare, il robot accetta comandi che consistono in una coppia di valori `d p`, dove `d` è una stringa che indica la direzione e può assumere uno dei quattro valori `NORD`, `SUD`, `EST`, `OVEST`, mentre `p > 0` è un intero che indica il numero di passi che il robot deve compiere in quella direzione.

Scrivere un programma che:

- legga da **standard input** una sequenza di righe di testo;
- 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 consiste in una coppia di valori `d p` che descrivono un comando che il robot deve eseguire.

Dopo aver terminato la fase di lettura, come mostrato nell'**Esempio d'esecuzione**, il programma deve stampare a video:

- il numero totale di passi che deve compiere il robot in ognuna delle quattro direzioni (se il robot non si deve mai muovere in una certa direzione, l'output relativo a tale direzione non va stampato);
- la direzione in cui il robot deve compiere il maggior numero totale di passi (a parità di numero di passi, tra due direzioni deve essere selezionata quella che precede l'altra in ordine alfabetico);
- la sequenza di comandi inversi che si dovrebbe impartire al robot per farlo ritornare al punto di partenza lungo lo stesso percorso.

Inoltre, all'interno del programma:

1. deve essere definito il tipo `Comando`:

```
type Comando struct {
    direzione string
    passi int
}
```

2. oltre alla funzione main(), devono essere definite ed utilizzate almeno le seguenti funzioni:

- una funzione `LeggiComandi() []Comando` che legge da **standard input** un testo su più righe e terminato dall'indicatore EOF, restituendo un valore `[]Comando` in cui è memorizzata la sequenza di comandi specificati nelle righe di testo lette;
- una funzione `AnalizzaComandi(comandi []Comando) map[string]int` che riceve in input un valore `[]Comando` nel parametro `comandi` e restituisce un valore `map[string]int` in cui, per ogni direzione specificata per almeno un comando presente in `comandi`, è memorizzato il numero totale di passi che il robot deve compiere in quella direzione rispetto alla totalità dei comandi presenti in `comandi`.

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

Esempio d'esecuzione:

```
$ go run esercizio_2.go
OVEST 2
NORD 1
EST 5
SUD 4
EST 5
NORD 3
SUD 1
OVEST 3

Movimenti totali:
OVEST 5
NORD 4
EST 10
SUD 5
Direzione in cui il robot deve compiere il maggior numero totale di passi:
EST
Comandi inversi:
EST 3, NORD 1, SUD 3, OVEST 5, NORD 4, OVEST 5, SUD 1, EST 2

$ go run esercizio_2.go
EST 3
NORD 1
NORD 4
EST 2

Movimenti totali:
EST 5
NORD 5
Direzione in cui il robot deve compiere il maggior numero totale di passi:
EST
Comandi inversi:
OVEST 2, SUD 4, SUD 1, OVEST 3

$ go run esercizio_2.go
SUD 4
OVEST 3
SUD 2
NORD 6

Movimenti totali:
OVEST 3
NORD 6
SUD 6
Direzione in cui il robot deve compiere il maggior numero totale di passi:
NORD
Comandi inversi:
SUD 6, NORD 2, EST 3, NORD 4
```

Esercizio 3

Come illustrato nell'immagine di seguito riportata, il piano cartesiano è diviso in quattro quadranti: I, II, III e IV quadrante.

```
                ^
                |
   II quadrante |  I quadrante
          ------|------>
   III quadrante|  IV quadrante
                |
```

Sul piano cartesiano, ad ogni punto individuato da una coppia di numeri reali, chiamati rispettivamente ascissa e ordinata, può essere associata un'etichetta simbolica, generalmente una lettera maiuscola.

Scrivere un programma che:

- legga da **riga di comando** un valore reale `soglia`;
- legga da **standard input** una sequenza di righe di testo;
- termini la lettura quando, premendo la combinazione di tasti `Ctrl+D`, viene inserito da **standard input** l'indicatore End-Of-File (EOF).

Ogni riga del testo è una stringa nel formato:

```
etichetta;x;y
```

La tripla di valori separati dal carattere `;` specifica un punto sul piano cartesiano:

1. *etichetta*: una stringa che specifica l'etichetta simbolica associata al punto (ad es.: "A", "B", ...)
2. *x*: un valore reale che specifica l'ascissa del punto;
3. *y*: un valore reale che specifica l'ordinata del punto.

Ogni coppia di punti descrive un segmento che ha per estremi i punti stessi.

Si ipotizzi che vengano inserite da **standard input** le seguenti di righe di testo:

```
A;10.0;2.0
B;11.5;3.0
C;8.0;1.0
```

La coppia di punti:

```
A;10.0;2.0
B;11.5;3.0
```

specifica il segmento `AB`.

La coppia di punti:

```
A;10.0;2.0
C;8.0;1.0
```

specifica il segmento `AC`.

La coppia di punti:

```
B;11.5;3.0
C;8.0;1.0
```

specifica il segmento `BC`.

La lunghezza di ciascun segmento è pari alla distanza euclidea tra gli estremi del segmento.

Per esempio, la lunghezza del primo segmento, quello con estremi il punto `A` ed il punto `B`, è pari alla distanza euclidea tra i punti `A` e `B`: $((x_A-x_B)^2 + (y_A-y_B)^2)^{1/2}$.

Una volta terminata la fase di lettura, il programma deve stampare a video (come mostrato nell'**Esempio di esecuzione**) la descrizione di ogni segmento definibile a partire dai punti specificati nelle righe di testo lette da **standard input** tale che:

1. il segmento non sia parallelo né all'asse delle ascisse né all'asse delle ordinate.
2. i due estremi del segmento giacciano nello stesso quadrante del piano cartesiano;
3. il segmento abbia una lunghezza minore del valore `soglia` letto da **riga di comando**;

Se non esistono segmenti che soddisfano le condizioni 1, 2 e 3, il programma non deve stampare nulla.

Si assuma che:

- il valore reale specificato a **riga di comando** sia nel formato corretto;
- le righe di testo lette da **standard input** siano nel formato corretto;
- la tripla di valori presente in ogni riga specifichi correttamente un punto sul piano cartesiano;
- vengano lette da **standard input** almeno 2 righe di testo.

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

- una funzione `Distanza(p1, p2 Punto) float64` che riceve in input due instanze del tipo `Punto` nei parametri `p1` e `p2` e restituisce un valore `float64` pari alla distanza euclidea tra i punti rappresentati da `p1` e `p2`;
- una funzione `StringPunto(p Punto) string` che riceve in input un'instanza del tipo `Punto` nel parametro `p` e restituisce un valore `string` che corrisponde alla rappresentazione `string` di `p` nel formato `ETICHETTA = (X, Y)`, dove `ETICHETTA` è il valore `string` che specifica l'etichetta simbolica di `p`, mentre `X` ed `Y` sono i valori `float64` che specificano rispettivamente l'ascissa e l'ordinata di `p`;
- una funzione `StringSegmento(s Segmento) string` che riceve in input un'instanza del tipo `Segmento` nel parametro `s` e restituisce un valore `string` che corrisponde alla rappresentazione `string` di `s` nel formato `Segmento con estremi ESTREMO_1 e ESTREMO_2.`, dove `ESTREMO_1` ed `ESTREMO_2` sono le rappresentazioni `string` delle istanze del tipo `Punto` che rappresentano gli estremi di `s`.

Esempio d'esecuzione:

```
$ cat punti1.txt
A;1;1
B;1;10
C;10;3
D;-3;3
E;-3;-4
F;6;-4
G;-4;-10

$ go run esercizio_3.go 10 < punti1.txt
Segmento con estremi A = (1.00, 1.00) e C = (10.00, 3.00).
Segmento con estremi E = (-3.00, -4.00) e G = (-4.00, -10.00).

$ go run esercizio_3.go 4.5 < punti1.txt

$ cat punti2.txt
A;3;4
B;10;-4
C;1;1
D;-2;-2
E;-4;6
F;-10;-3

$ go run esercizio_3.go 13 < punti2.txt
Segmento con estremi A = (3.00, 4.00) e C = (1.00, 1.00).
Segmento con estremi D = (-2.00, -2.00) e F = (-10.00, -3.00).

$ go run esercizio_3.go 3.8 < punti2.txt
Segmento con estremi A = (3.00, 4.00) e C = (1.00, 1.00).
```