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
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
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

$ 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
p@55Word_Lung@
La pw non è definita correttamente:
- Almeno 3 caratteri della pw devono rappresentare delle cifre decimali
- Almeno 4 caratteri della pw non devono rappresentare lettere o cifre decimali
```

Esercizio 2

Scrivere un programma che legga da **riga di comando** un intero `n` e stampi un pattern come di seguito riportato:

```
0000+0000
000    000
00      00
0        0
```

ovvero, due triangoli rovesciati (utilizzando il carattere `'0'` (lettera o maiuscola)), separati da un singolo carattere `'+'`. Per entrambi i triangoli, la larghezza e l'altezza hanno la stessa misura. In particolare, prendendo come unità di misura il carattere `'0'`, la larghezza e l'altezza dei triangoli misurano `n'0'` .

Esempio:

Per `n=4`, ogni triangolo deve essere largo `4` colonne e alto `4` righe, ossia largo `4'0'` e alto `4'0'`:

```
0000+0000
000    000
00      00
0        0
```

Si assuma che il valore specificato a riga di comando sia nel formato corretto e, in particolare, sia un intero maggiore o uguale a zero.

Esempio d'esecuzione:

```
$ go run esercizio_2.go 3
000+000
00   00
0     0

$ go run esercizio_2.go 2
00+00
0   0

$ go run esercizio_2.go 4
0000+0000
000   000
00     00
0       0

$ go run esercizio_2.go 1
0+0

$ go run esercizio_2.go 0

$ go run esercizio_2.go 5
00000+00000
0000   0000
000     000
00       00
0         0
```

Esercizio 3

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 tutti i segmenti che:

1. hanno una lunghezza maggiore del valore `soglia` letto da **riga di comando**;
2. non sono paralleli né all'asse delle ascisse né all'asse delle ordinate.

Se non esistono segmenti che soddisfano le condizioni 1 e 2, 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:

```
$ go run esercizio_3.go 5.3
A;0;0
B;0;10
C;10;0
D;3;3
E;3;4
F;6;4
Segmento con estremi A = (0.000000, 0.000000) e F = (6.000000, 4.000000).
Segmento con estremi B = (0.000000, 10.000000) e C = (10.000000, 0.000000).
Segmento con estremi B = (0.000000, 10.000000) e D = (3.000000, 3.000000).
Segmento con estremi B = (0.000000, 10.000000) e E = (3.000000, 4.000000).
Segmento con estremi B = (0.000000, 10.000000) e F = (6.000000, 4.000000).
Segmento con estremi C = (10.000000, 0.000000) e D = (3.000000, 3.000000).
Segmento con estremi C = (10.000000, 0.000000) e E = (3.000000, 4.000000).
Segmento con estremi C = (10.000000, 0.000000) e F = (6.000000, 4.000000).

$ go run esercizio_3.go 1.4
A;3;4
B;10;4
C;1;1
D;2;2
E;4;6
Segmento con estremi A = (3.000000, 4.000000) e C = (1.000000, 1.000000).
Segmento con estremi A = (3.000000, 4.000000) e D = (2.000000, 2.000000).
Segmento con estremi A = (3.000000, 4.000000) e E = (4.000000, 6.000000).
Segmento con estremi B = (10.000000, 4.000000) e C = (1.000000, 1.000000).
Segmento con estremi B = (10.000000, 4.000000) e D = (2.000000, 2.000000).
Segmento con estremi B = (10.000000, 4.000000) e E = (4.000000, 6.000000).
Segmento con estremi C = (1.000000, 1.000000) e D = (2.000000, 2.000000).
Segmento con estremi C = (1.000000, 1.000000) e E = (4.000000, 6.000000).
Segmento con estremi D = (2.000000, 2.000000) e E = (4.000000, 6.000000).
```

Esercizio 4

Scrivere un programma che:

- legga da **riga di comando** una sequenza `s` di valori che rappresentano caratteri appartenenti all'alfabeto inglese (e quindi codificati all'interno dello standard US-ASCII (integrato nello standard Unicode));
- stampi a video tutte le sottosequenze di caratteri presenti in `s` che:
  i. iniziano e finiscono con lo stesso carattere;
  ii. sono formate da almeno 3 caratteri.

Ciascuna sottosequenza deve essere stampata un'unica volta, riportando il relativo numero di occorrenze della sottosequenza in `s` (cfr. **Esecuzione d'esecuzione**).

Le sottosequenze devono essere stampate in ordine di lunghezza (dalla più lunga alla più corta).

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

Si noti che una sottosequenza può essere contenuta in un'altra sottosequenza più grande.

Si assuma che la sequenza di valori specificata a riga di comando sia nel formato corretto e includa almeno 3 caratteri.

Esempio d'esecuzione:

```
$ go run esercizio_4.go a b b a b b a
a b b a b b a -> Occorrenze: 1
b b a b b -> Occorrenze: 1
b a b b -> Occorrenze: 1
a b b a -> Occorrenze: 2
b b a b -> Occorrenze: 1
b a b -> Occorrenze: 1

$ go run esercizio_4.go a b c a c b a
a b c a c b a -> Occorrenze: 1
b c a c b -> Occorrenze: 1
a b c a -> Occorrenze: 1
a c b a -> Occorrenze: 1
c a c -> Occorrenze: 1

$ go run esercizio_4.go e a b c a c f
a b c a -> Occorrenze: 1
c a c -> Occorrenze: 1

$ go run esercizio_4.go e a b b c a b c b f
b b c a b c b -> Occorrenze: 1
b c a b c b -> Occorrenze: 1
a b b c a -> Occorrenze: 1
b b c a b -> Occorrenze: 1
c a b c -> Occorrenze: 1
b c a b -> Occorrenze: 1
b c b -> Occorrenze: 1

$ go run esercizio_4.go a b c c e
```