Laboratorio 11 - Simulazione

Esercizio filtro

Definizione: La codifica compressa di una stringa di caratteri è definita da una sequenza di coppie di valori `ch n`, dove `ch` rappresenta un carattere, mentre `n>0` è il numero di volte che il carattere `ch` viene ripetuto.

> *Esempio:* Alla sequenza di valori `e 3 a 4`, corrisponde la stringa `eeeaaaa`, ovvero il carattere `'e'`, ripetuto 3 volte, seguito dal carattere `'a'`, ripetuto 4 volte.

Scrivere un programma che legga da **riga di comando** la codifica compressa di una stringa di caratteri e stampi a video la stringa non compressa.

Si assuma che ogni carattere dato in input appartenenga all'alfabeto inglese e sia quindi codificato all'interno dello standard US-ASCII (integrato nello standard Unicode).

Si assuma inoltre che la sequenza di valori specificata a riga di comando sia nel formato corretto e includa almeno una coppia `ch n`.

Esempi d'esecuzione:

```
$ go run esercizio_filtro.go e 3 r 4 Y 3
eeerrrrYYY

$ go run esercizio_filtro.go R 3 A 1 B 10
RRRABBBBBBBBBB

$ go run esercizio_filtro.go W 2
WW

$ go run esercizio_filtro.go r 3 P 4
rrrPPPP
```

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 i casi testati e riportati nell'esempio d'esecuzione non sono stati eseguiti in modo corretto, ed il filtro è considerato **errato**.

Esercizio 1

Scrivere un programma che legga da **riga di comando** una sequenza di stringhe di caratteri (parole di senso compiuto rispetto alla lingua italiana).

La **prima parola** specificata nella sequenza letta è in **posizione 0** (nella sequenza letta), la **seconda parola** specificata nella sequenza letta è in **posizione 1**, la **terza parola** specificata nella sequenza letta è in **posizione 2**, etc.

Il programma deve ristampare a video la sequenza di parole lette come mostrato nell'**Esempio d'esecuzione**:

- ogni parola viene ristampata alternando caratteri in maiuscolo a caratteri in minisculo;
- le parole specificate in posizione pari nella sequenza letta vengono ristampate a video incominciando con un carattere maiuscolo, mentre quelle in posizione dispari incominciano con un carattere minuscolo.

Si assuma inoltre che la sequenza di valori specificata a riga di comando sia nel formato corretto e includa almeno una stringa.

Oltre alla funzione `main()` il programma deve includere le seguenti funzioni:

- `TrasformaParola(parola string, posizione int) (parolaTrasformata string)` che riceve in input un valore `string` ed un valore `int` nei parametri `parola` e `posizione`, e restituisce nella variabile `parolaTrasformata` un valore `string` in cui i caratteri di `parola` compaiono alternativamente in maiuscolo e in minuscolo: se `posizione` è pari, il primo carattere in `parolaTrasformata` è maiuscolo, altrimenti il primo carattere in `parolaTrasformata` è minuscolo,

Esempio d'esecuzione:

```
$ go run esercizio_1.go ciao mondo
CiAo mOnDo

$ go run esercizio_1.go esame di programmazione
EsAmE dI PrOgRaMmAzIoNe
```

```
$ go run esercizio_1.go gennaio febbraio marzo aprile maggio giugno luglio agosto
GeNnAiO fEbBrAiO MaRzO aPrIlE MaGgIo gIuGnO LuGlIo aGoStO
```

Esercizio 2

Scrivere un programma che:

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

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

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

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

Si noti che una sottostringa può essere contenuta in un'altra sottostringa più lunga.

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

Esempio d'esecuzione:

```
$ go run esercizio_2.go abbabba
abbabba -> Occorrenze: 1
bbabb -> Occorrenze: 1
babb -> Occorrenze: 1
abba -> Occorrenze: 2
bbab -> Occorrenze: 1
bab -> Occorrenze: 1

$ go run esercizio_2.go abcacba
abcacba -> Occorrenze: 1
bcacb -> Occorrenze: 1
abca -> Occorrenze: 1
acba -> Occorrenze: 1
cac -> Occorrenze: 1

$ go run esercizio_2.go eabcacf
abca -> Occorrenze: 1
cac -> Occorrenze: 1
```

```
$ go run esercizio_2.go eabbcabcbf
bbcabcb -> Occorrenze: 1
bcabcb -> Occorrenze: 1
abbca -> Occorrenze: 1
bbcab -> Occorrenze: 1
cabc -> Occorrenze: 1
bcab -> Occorrenze: 1
bcb -> Occorrenze: 1

$ go run esercizio_2.go abcce
```

Esercizio 3

Parte 1

Scrivere un programma che legga da **standard input** una sequenza di righe di testo, ognuna delle quali descrive un punto sul piano cartesiano.

Ogni riga di testo è una stringa che specifica l'etichetta del punto (ad es.: `A`, `B`, ...), l'ascissa e l'ordinata del punto nel seguente formato:

```
etichetta;x;y
```

> *Esempio*: 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
> ```
>
> tali righe specificano 3 punti: A(10, 2), B(11.5, 3) e C(8, 1).

Definire la struttura `Punto` per memorizzare l'`etichetta`, l'`ascissa` e l'`ordinata` di un punto sul piano cartesiano.

Implementare una funzione `NuovoTragitto() (tragitto []Punto)` che:

1. legge da **standard input** una sequenza di righe di testo nel formato *etichetta;x;y*, terminando la lettura quando viene letto l'indicatore End-Of-File (EOF);
2. restituisce un valore `[]Punto` nella variabile `tragitto` in cui è memorizzata la sequenza di istanze del tipo `Punto` inizializzate con i valori letti da **standard input**. L'ordine dei punti all'interno della slice `tragitto` deve rispettare l'ordine in cui i corrispondenti valori sono stati letti da **standard input**.

Si assuma che:

- 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.

Parte 2

I punti letti da **standard input** nella parte `Parte 1` dell'esercizio definiscono un *tragitto*. Ogni coppia di punti consecutivi definisce una *tratta* del tragitto.

La lunghezza di ciascuna tratta del tragitto è pari alla distanza euclidea tra i due punti che definiscono la tratta. Per esempio, la lunghezza della tratta `AB`, è pari alla distanza euclidea tra i punti `A` e `B`: $((x_A-x_B)^2 + (y_A-y_B)^2)^{1/2}$.

La lunghezza del tragitto è data dalla somma delle lunghezze delle sue tratte.

Una volta terminata la fase di lettura (**Parte 1**), il programma deve:

- stampare a video la lunghezza totale del tragitto;
- stampare a video la rappresentazione `string` del primo punto del tragitto che si incontra dopo aver percorso più della metà della lunghezza del tragitto.

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

- `Distanza(p1, p2 Punto) float64` che riceve in input due punti nei parametri `p1` e `p2` e restituisce un valore `float64` pari alla distanza euclidea tra i punti rappresentati da `p1` e `p2`;
- `String(p Punto) string` che riceve in input un punto nel parametro `p` e restituisce un valore `string` che corrisponde alla rappresentazione `string` di `p` nel formato `etichetta = (x, y)`;
- `Lunghezza(tragitto []Punto) float64` che riceve in input una slice di punti nel parametro `tragitto` e restituisce un valore `float64` pari alla lunghezza del tragitto rappresentato da `tragitto`.

Esempio d'esecuzione:

```
$ cat punti1
A;10.0;2.0
B;11.5;3.0
C;8.0;1.0
D;3;4
E;1;0
F;-1;5
$ go run esercizio_3.go < punti1
Lunghezza percorso: 21.522
Punto oltre metà: D = (3.0, 4.0)
```

```
$ cat punti2
A;0;0
B;4;0
C;4;4
D;0;4
E;0;0
$ go run esercizio_3.go < punti2
Lunghezza percorso: 16.000
Punto oltre metà: D = (0.0, 4.0)
```

Esercizio 4

Scrivere un programma che legga da **riga di comando** due numeri interi `n` e `d`.

Il programma deve calcolare e stampare il più piccolo numero ottenibile rimuovendo `d` cifre decimali da `n`.

> *Esempio*: Si ipotizzi che i valori di `n` e `d` siano rispettivamente 4567 e 2.
>
> 45 è il più piccolo numero ottenibile rimuovendo 2 cifre da 4567. Gli altri numeri ottenibili rimuovendo 2 cifre da 4567 sono 46, 47, 56, 57 e 67. Si noti che non è possibile considerare permutazioni delle cifre di `n`: rimuovendo 2 cifre da 4567, non è possibile, ad esempio, ottenere 76.

Si assuma che i valori letti da **riga di comando** siano nel formato corretto e che il numero di cifre di `n` sia maggiore di `d`.

Esempio d'esecuzione:

```
$ go run esercizio_4.go 9452 2
numero migliore: 42

$ go run esercizio_4.go 1324 3
numero migliore: 1

$ go run esercizio_4.go 4612 1
numero migliore: 412

$ go run esercizio_4.go 4213 3
numero migliore: 1
```