Laboratorio 10 - Mappe, Bufio, Files

1 (Bufio) Lettura testo da stdin su più righe

```go
package main

import (
    "bufio"
    "fmt"
    "os"
)

func main() {

    fmt.Println("Inserisci testo (termina con CTRL+D):")

    testo := ""
    scanner := bufio.NewScanner(os.Stdin)
    for scanner.Scan() {
        testo += scanner.Text() + "\n"
    }

    fmt.Print("Testo letto:\n", testo)

}
```

2 (Mappe) Qual è l'output?

Analizziamo l'output del seguente programma.

```go
package main

import "fmt"

func main() {
    mappa := make(map[string]int)
    // equivalente a: mappa := map[string]int{}

    mappa["A"] = 10
    mappa["B"] -= 5
    mappa["D"] = mappa["E"] + 5
    if mappa["F"] == 0 {

        fmt.Printf("F è presente con valore %d\n", mappa["F"])

    } else {
```

```
        fmt.Print("F non è presente\n")

    }

    fmt.Println("Elementi in mappa:")

    for k := range mappa {

        fmt.Printf("Chiave: %s - Valore: %d\n", k, mappa[k])

    }
}
```

3 (Mappe) Qual è l'output?

Analizziamo l'output del seguente programma.

```
package main

import "fmt"

func main() {
    mappa := make(map[string]int)
    // equivalente a: mappa := map[string]int{}

    mappa["A"] = 10
    mappa["B"] -= 5
    mappa["D"] = mappa["E"] + 5

    if v, ok := mappa["C"]; ok {
        fmt.Printf("C è presente con valore %d\n", v)
    } else {
        fmt.Print("C non è presente\n")
    }
    if v, ok := mappa["B"]; ok {
        fmt.Printf("B è presente con valore %d\n", v)
    } else {
        fmt.Print("B non è presente\n")
    }

    delete(mappa, "B")

    mappa["A"] += 100

    fmt.Println("Elementi in mappa:")
```

```
    for k := range mappa {
        fmt.Printf("Chiave: %s - Valore: %d\n", k, mappa[k])
    }
}
```

4 (Mappe) Qual è l'output?

Analizziamo l'output del seguente programma.

```
package main

import (
    "fmt"
)

var (
    nominativi = map[string]string{"023314944": "Mario Rossi",
        "024158685": "Carlo Bianchi", "026424971": "Giuseppe Verdi",
        "0269001634": "Carlo Bianchi", "026691369": "Mario Rossi",
        "0248704925": "Carlo Bianchi", "023554756": "Giuseppe Verdi"}
)

func main() {
    numeriTelefonici := make(map[string]string)
    for k, v := range nominativi {
        numeriTelefonici[v] = k
    }
    for k, v := range numeriTelefonici {
        fmt.Printf("Nominativo: %v\nNumero telefonico: %v\n\n", k, v)
    }
}
```

5 (Mappe) Qual è l'output?

Analizziamo l'output del seguente programma.

```
package main

import (
    "fmt"
)

var (
    nominativi = map[string]string{"023314944": "Mario Rossi",
        "024158685": "Carlo Bianchi", "026424971": "Giuseppe Verdi",
        "0269001634": "Carlo Bianchi", "026691369": "Mario Rossi",
```

```
        "0248704925": "Carlo Bianchi", "023554756": "Giuseppe Verdi"}
)

func main() {
    numeriTelefonici := make(map[string][]string)
    for k, v := range nominativi {
        numeriTelefonici[v] = append(numeriTelefonici[v], k)
    }
    for k, v := range numeriTelefonici {
        fmt.Printf("Nominativo: %v\nNumero telefonico: %v\n\n", k, v)
    }
}
```

6 (File) Lettura file

Supponiamo che: * la directory corrente contenga un file denominato `punti.txt` contenente il seguente testo:

```
A 10.5 20
B 15 30
C 12.5 25.6
```

Il seguente programma dopo aver letto da riga di comando il nome del file (`punti.txt`), legge le righe del file di testo e le ristampa a video:

```
package main

import (
    "fmt"
    "io"
    "os"
)

func main() {
    f, err := os.Open(os.Args[1])
    if err != nil {
        fmt.Printf("Error while opening the file! %v\n", err)
        f.Close()
        return
    }

    for {
        var nome string
        var x, y float64
        _, err = fmt.Fscan(f, &nome, &x, &y)
        if err == io.EOF {
            f.Close()
```

```
            break
        }
        if err != nil {
            fmt.Printf("Error while reading the file! %v\n", err)
            f.Close()
            return
        }
        fmt.Printf("Punto %s = (%v, %v)\n", nome, x, y)
    }
}
```

7 (File) Scrittura file

Il seguente script crea un file `tabellina.txt` all'interno della directory corrente contenente la tabellina del 10:

```
package main

import (
    "fmt"
    "os"
)

func main() {
    f, err := os.Create("tabellina.txt")
    if err != nil {
        fmt.Printf("Error while creating the file! %v\n", err)
        f.Close()
        return
    }

    for i:=1; i<=10; i++ {
        risultato := i * 10
        _, err = fmt.Fprintln(f, "10 x ", i, risultato)
        if err != nil {
            fmt.Printf("Error while writing the file! %v\n", err)
            f.Close()
            return
        }
    }
    f.Close()
}
```

Esercizi Pratici

1 (Mappe) Istogramma a barre orizzontali (1)

Scrivere un programma che: 1. legga da **standard input** un testo su più righe (alcune delle quali possono essere delle righe vuote (`""`)); 2. termini la lettura quando, premendo la combinazione di tasti `Ctrl+D`, viene inserito da **standard input** l'indicatore End-Of-File (EOF); 3. come mostrato nell'**Esempio di esecuzione**, stampi un istogramma a barre orizzontali per rappresentare il numero di occorrenze di ogni lettera presente nel testo letto: * una lettera è un carattere il cui codice Unicode, se passato come argomento alla funzione `func IsLetter(r rune) bool` del package `unicode`, fa restituire `true` alla funzione; * le lettere minuscole sono da considerarsi diverse dalle lettere maiuscole; * ogni barra viene rappresentata utilizzando il carattere asterisco (`*`); se il numero di occorrenze della lettera `e` è per esempio `9`, la barra corrispondente sarà formata da `9` caratteri `*`.

Oltre alla funzione `main()`, devono essere definite ed utilizzate almeno le seguenti funzioni: * una funzione `LeggiTesto() string` che legge da **standard input** un testo su più righe (alcune delle quali possono essere delle righe vuote (`""`)) e terminato dall'indicatore EOF, restituendo un valore `string` in cui è memorizzato il testo letto; * una funzione `Occorrenze(s string) map[rune]int` che riceve in input un valore `string` nel parametro `s` e restituisce un valore `map[rune]int` in cui, per ogni lettera presente in `s`, è memorizzato il numero di occorrenze della lettera in `s`.

Esempio d'esecuzione:

```
$ go run istogrammaV1.go
TESTO di prova
disposto su più righe!
Istogramma:
i: ****
a: *
h: *
d: **
r: **
g: *
e: *
p: ***
s: ***
t: *
u: *
o: ***
E: *
S: *
O: *
```

```
v: *
ù: *
T: **
```

2 (Mappe) Istogramma a barre orizzontali (2)

Scrivere un programma che: 1. legga da **standard input** un testo su più righe (alcune delle quali possono essere delle righe vuote (`""`)); 2. termini la lettura quando, premendo la combinazione di tasti `Ctrl+D`, viene inserito da **standard input** l'indicatore End-Of-File (EOF); 3. come mostrato nell'**Esempio di esecuzione**, stampi un istogramma a barre orizzontali per rappresentare il numero di occorrenze di ogni lettera presente nel testo letto: 1. una lettera è un carattere il cui codice Unicode, se passato come argomento alla funzione `func IsLetter(r rune) bool` del package `unicode`, fa restituire `true` alla funzione; 2. le lettere minuscole sono da considerarsi diverse dalle lettere maiuscole; 3. ogni barra viene rappresentata utilizzando il carattere asterisco (`*`); se il numero di occorrenze della lettera `e` è per esempio `9`, la barra corrispondente sarà formata da `9` caratteri `*`; 4. le barre devono essere stampate a partire da quella associata alla lettera con codice Unicode più piccolo fino a quella associata alla lettera con codice Unicode più grande.

Oltre alla funzione `main()`, devono essere definite ed utilizzate almeno le seguenti funzioni: * una funzione `StampaIstogramma(occorrenze map[rune]int)` che riceve in input un valore `map[rune]int` nel parametro `occorrenze`, in cui ad una data lettera è associato un dato numero di occorrenze, e stampa l'istogramma relativo alle lettere presenti come valori chiave in `occorrenze` secondo quanto descritto ai punti iii e iv.

Suggerimenti: * Si consideri il seguente programma.

```go
package main

import (
        "fmt"
)

func main() {

        capitali := map[string]string{"Austria": "Vienna", "Italia": "Roma",
        "Giappone": "Tokio", "Francia": "Parigi"}

        for _, v := range capitali {
                fmt.Println(v)
        }
}
```

Output:

```
Vienna
Roma
Tokio
Parigi
```

Il programma stampa a video i nomi delle capitali europee memorizzati in `capitali` in ordine non alfabetico.

Modifichiamo il programma nel seguente modo:

```go
package main

import (
        "fmt"
        "sort"
)

func main() {

        capitali := map[string]string{"Francia": "Parigi", "Italia": "Roma",
        "Giappone": "Tokio", "Austria": "Vienna"}

        valori := []string{} //slice ausiliaria per fare l'ordinamento

        for _,k := range capitali {
                valori = append(valori, k)
        }

        sort.Strings(valori)

        for _,v := range(valori) {

                fmt.Println(v)

        }

}
```

Per ottenere il seguente output:

```
Parigi
Roma
Tokio
Vienna
```

- Le lettere associate alle barre possono quindi essere ordinate in senso crescente utilizzando una slice ausiliare sulle *chiavi* della mappa corrispondente. Si utilizzino quindi la funzione `sort.Strings`, la conversione da `[]runes`

a `string` per creare la slice ausiliare per fare il sorting, e la conversione da `string` a `[]runes` per stampare la mappa.

Esempio d'esecuzione:

```
$ go run istogramma.go
Ciao,
come stai?
Occorrenze:
C: *
a: **
c: *
e: *
i: **
m: *
o: **
s: *
t: *

$ cat test
Ciao,
come stai?
Tutto bene?
Spero di sì :-)
$ go run istogramma.go < test
Occorrenze:
C: *
S: *
T: *
a: **
b: *
c: *
d: *
e: ****
i: ***
m: *
n: *
o: ****
p: *
r: *
s: **
t: ***
u: *
ì: *
```

3 (Mappe) Ripetizioni

Scrivere un programma che: * legga da **standard input** un testo su più righe (alcune delle quali possono essere delle righe vuote (`""`)); * termini la lettura quando, premendo la combinazione di tasti `Ctrl+D`, viene inserito da **standard input** l'indicatore End-Of-File (EOF); * stampi a video le seguenti informazioni relative al testo letto: 1. Il numero di parole distinte presenti nel testo (una parola è una stringa interamente definita da caratteri il cui codice Unicode, se passato come argomento alla funzione `func IsLetter(r rune) bool`, fa restituire `true` alla funzione). 2. La lista di parole distinte presenti nel testo, riportando per ogni parola il relativo numero di occorrenze nel testo (cfr. **Esempio d'esecuzione**).

Oltre alle funzione `main()`, devono essere definite ed utilizzate almeno le seguenti funzioni: * una funzione `LeggiTesto() string` che legge da **standard input** un testo su più righe (alcune delle quali possono essere delle righe vuote (`""`)) e terminato dall'indicatore EOF, restituendo un valore `string` in cui è memorizzato il testo letto; * una funzione `SeparaParole(s string) []string` che riceve in input un valore `string` nel parametro `s` e restituisce un valore `[]string` in cui sono memorizzate tutte le parole presenti in `s`; * una funzione `ContaRipetizioni(sl []string) map[string]int` che riceve in input un valore `[]string` nel parametro `sl` e restituisce un valore `map[string]int` in cui, per ogni parola presente in `sl`, è memorizzato il numero di occorrenze della parola in `sl`.

Esempio d'esecuzione:

```
$ go run ripetizioni.go
Ciao come stai?
io sto bene, tu?
anche io sto bene, grazie
Parole distinte: 9
io: 2
sto: 2
tu: 1
anche: 1
come: 1
stai: 1
bene: 2
grazie: 1
Ciao: 1
```

4 (Mappe) Sottosequenze

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: 1. iniziano

e finiscono con lo stesso carattere; 2. 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. **Esempio d'esecuzione**).

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 sottosequenze.go a b b a b b a
a b b a -> Occorrenze: 2
a b b a b b a -> Occorrenze: 1
b b a b -> Occorrenze: 1
b b a b b -> Occorrenze: 1
b a b -> Occorrenze: 1
b a b b -> Occorrenze: 1

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

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

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

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

Siccome le mappe non sono strutture ordinate, il vostro output potrà presentare le sottosequenze in ordine diverso. L'importante è che ci siano tutte e con il

giusto numero di occorrenze.