Laboratorio 9 - Mappe

1 Qual è l'output?

Analizziamo l'output del seguente programma.

```
package main

import "fmt"

func main() {
    var m map[string]int
    if m == nil {
        fmt.Println("'nil' è lo zero-value di una variabile di tipo " +
            "'reference type' non inizializzata.")
    }
    //m["mamma"] = 5 /* panic: assignment to entry in nil map */
    m = make(map[string]int)
    for _, s := range []string{"questo", "è", "un", "test"} {
        m[s] = len([]rune(s))
    }

    for k, v := range m {
        fmt.Println(k, "->", v)
    }
    fmt.Println()

    for k := range m {
        fmt.Println(k, "->", m[k])
    }
}
```

2 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["F"] = 0
```

```
    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")
    }
    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])
    }
}
```

3 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)
    }
}
```

4 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)
    }
}
```

5 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: **
```

6 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:
    i. 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;
    ii. le lettere minuscole sono da considerarsi diverse dalle lettere maiuscole;

iii. 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 `*` ;

iv. 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.

```
package main

import (
    "fmt"
)

func main() {

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

    stati := []string{"Austria", "Francia", "Giappone", "Italia"}

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

Output:

```
Vienna
Parigi
Tokio
Roma
```

Il programma stampa a video i nomi delle capitali europee memorizzati in `capitali` in base all'ordine in cui i nomi degli stati corrispondenti sono memorizzati in `stati` .

- Le lettere associate alle barre possono essere ordinate in senso crescente utilizzando la seguente funzione.

```
func SortRunes(a []rune) {
    for i:=0;i<len(a)-1;i++{
        indiceMin := i
        for j := i + 1; j<len(a); j++ {
            if a[indiceMin] > a[j] {
                indiceMin = j
            }
        }
        a[i], a[indiceMin] = a[indiceMin], a[i]
    }
}
```

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: *
```

7 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:
  i. 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).
  ii. La lista di parole distinte presenti nel testo, riportando per ogni parola il relativo numero di occorrenze nel testo (cfr. **Esecuzione 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
```

8 Estratto conto

Scrivere un programma che, tramite **redirizione dello standard input**, legga un file di testo `operazioni.txt` memorizzato nella stessa directory in cui è memorizzato il programma.

La prima riga del file `operazioni.txt` è una stringa nel formato

```
IMPORTO
```

dove `IMPORTO` è un valore reale che specifica il saldo iniziale di un conto corrente.

Ogni riga del file `operazioni.txt` successiva alla prima è una stringa nel formato

```
DATA;TIPO;IMPORTO
```

e specifica un'operazione bancaria effettuata sul conto corrente:

- `DATA`: Una stringa senza spazi che codifica la data in cui è avvenuta l'operazione bancaria, specificata in uno dei seguenti possibili formati:
  i. aaaa/m/g
  ii. aaaa/m/gg
  iii. aaaa/mm/g
  iv. aaaa/mm/gg
  v. g/m/aaaa
  vi. gg/m/aaaa
  vii. g/mm/aaaa
  viii. gg/mm/aaaa
- `TIPO`: Un carattere che specifica la natura dell'operazione bancaria: `'V'` per versamento e `'P'` per prelievo.
- `IMPORTO`: Un valore reale maggiore di 0 che specifica l'ammontare dell'importo versato o prelevato.

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

1. Il saldo iniziale del conto corrente.
2. Per ogni giorno in cui è stata effettuata almeno un'operazione bancaria:
   i. la data del giorno considerato nel formato gg-mm-aaaa
   ii. la lista delle operazioni effettuate nel giorno considerato;
   iii. il saldo giornaliero: la differenza tra l'ammontare degli importi versati e l'ammontare degli importi prelevati con operazioni effettuate nel giorno considerato.
3. Il saldo finale del conto corrente.

In particolare, per quanto riguarda il punto 2., si noti che le informazioni relative ai giorni in cui è stata effettuata almeno un'operazione bancaria devono essere stampate considerando l'ordine cronologico dei giorni (dal giorno relativo alla data più antica al giorno relativo alla data più recente).

Si assuma che:

- ogni riga del file `operazioni.txt` sia nel formato corretto;
- i valori presenti in ogni riga del file `operazioni.txt` successiva alla prima specifichino correttamente un'operazione bancaria;
- il file `operazioni.txt` non sia vuoto.

Esempio d'esecuzione:

```
$ cat operazioni_1.txt
12.5
2019/06/04;P;34
2020/01/05;V;45.34
2019/6/04;P;10
2019/6/24;P;23
2019/06/04;P;456.10
2019/06/4;V;26.1
2019/06/8;P;74.23
2020/1/05;V;178.30
2020/01/7;V;194
2020/02/05;V;56

$ go run estratto_conto.go < operazioni_1.txt
SALDO INIZIALE:                 12.50

DATA: 04-06-2019
Prelievo:                       34.00
Prelievo:                       10.00
Prelievo:                      456.10
Versamento:                     26.10
                           ____________
SALDO GIORNALIERO:            -474.00
                           ============

DATA: 08-06-2019
Prelievo:                       74.23
                           ____________
SALDO GIORNALIERO:             -74.23
                           ============

DATA: 24-06-2019
Prelievo:                       23.00
                           ____________
SALDO GIORNALIERO:             -23.00
                           ============

DATA: 05-01-2020
Versamento:                     45.34
```

```
Versamento:                   178.30
                           ___________
SALDO GIORNALIERO:            223.64
                           ===========

DATA: 07-01-2020
Versamento:                   194.00
                           ___________
SALDO GIORNALIERO:            194.00
                           ===========

DATA: 05-02-2020
Versamento:                    56.00
                           ___________
SALDO GIORNALIERO:             56.00
                           ===========

SALDO FINALE:                 -85.09
                           ===========

$ cat operazioni_2.txt
1255.55
2019/10/2;V;130.11
2019/10/3;P;10.2
2019/11/01;V;560.9
2019/12/01;P;126
2019/10/02;P;30
2019/10/03;V;110.49
2019/11/1;P;560
2019/12/1;V;120.11

$ go run estratto_conto.go < operazioni_2.txt
SALDO INIZIALE:              1255.55

DATA: 02-10-2019
Versamento:                   130.11
Prelievo:                      30.00
                           ___________
SALDO GIORNALIERO:            100.11
                           ===========

DATA: 03-10-2019
Prelievo:                      10.20
Versamento:                   110.49
                           ___________
SALDO GIORNALIERO:            100.29
                           ===========

DATA: 01-11-2019
Versamento:                   560.90
Prelievo:                     560.00
                           ___________
SALDO GIORNALIERO:              0.90
                           ===========
```

```
DATA: 01-12-2019
Prelievo:                     126.00
Versamento:                   120.11
                           ___________
SALDO GIORNALIERO:             -5.89
                           ===========

SALDO FINALE:                1450.96
                           ===========
```

9 Sottosequenze (1)

Scrivere un programma che legga da **riga di comando** una sequenza di numeri interi e stampi tutte le sottosequenze che iniziano e finiscono con lo stesso numero. Ciascuna sottosequenza deve essere stampata su una riga diversa.

Esempio:

Se la sequenza di input è `1 2 3 -14 2 5`, l'unica sottosequenza è `2 3 -14 2`. Se la sequenza di input è `1 2 1 2 3`, abbiamo 2 sottosequenze: `1 2 1` e `2 1 2`.

Si consideri che:

- se a **riga di comando** non viene specificata alcuna sequenza, il programma non deve stampare nulla;
- una sottosequenza può essere contenuta in una sottosequenza più grande;
- ogni sottosequenza deve comparire una sola volta tra quelle stampate a video;
- le sottosequenze devono essere stampate in ordine di lunghezza (dalla più corta alla più lunga).

Esempio d'esecuzione:

```
$ go run sottosequenze.go 1 2 3 4 2 5
2 3 4 2

$ go run sottosequenze.go 1 2 1 2 3
1 2 1
2 1 2

$ go run sottosequenze.go 1 2 -45 2 1
2 -45 2
1 2 -45 2 1

$ go run sottosequenze.go

$ go run sottosequenze.go 10 3 12 10 4 5 13
10 3 12 10

$ go run sottosequenze.go 1 2 5 5 2 3 1
5 5
```

```
2 5 5 2
1 2 5 5 2 3 1

$ go run sottosequenze.go 1 3 2 2 3 2 2 3 1
2 2
2 3 2
2 3 2 2
3 2 2 3
2 2 3 2
2 2 3 2 2
3 2 2 3 2 2 3
1 3 2 2 3 2 2 3 1
```

10 Sottosequenze (2)

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 sottosequenze.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 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
a b c a -> Occorrenze: 1
a c b a -> Occorrenze: 1
c a c -> Occorrenze: 1

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

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