Laboratorio 6 - Rune iterazione

1 Qual è l'output?

Qual è l'output del seguente programma?

```
package main

import "fmt"

func main() {
    var x int
    var y int = 'a'
    x = 'A'
    fmt.Print(x, " ")
    for x := 1; x < 10; x++ {
        fmt.Print(x+y, " ")
    }
    fmt.Println()
}
```

2 Carte

Sapendo che al codice Unicode 127153 (associato alla rappresentazione in bit Unicode/UTF-8 `'\U0001F0B1'`) corrisponde il simbolo "asso di cuori", e che i codici successivi corrispondono alle carte successive (2 di cuori, 3 di cuori, ...), scrivere un programma che stampi tutte le carte da gioco dall'asso di cuori al 10 di cuori.

Suggerimento: Un carattere è una variabile di tipo `rune`, il cui valore è un intero corrispondente al codice Unicode del carattere. Le istruzioni equivalenti `var c rune = 127163` e `var c rune = '\U0001F0B1'` servono a definire la varibile `c` di tipo `rune` ed inizializzarla al valore "asse di cuori". Per stampare la carta da gioco "asse di cuori" si può utilizzare l'istruzione `fmt.Print(string(c))` o `fmt.Printf("%c",c)`.

```
$ go run carte.go
Simbolo: 🂱 - Codice numerico in base 10: 127153
Simbolo: 🂲 - Codice numerico in base 10: 127154
Simbolo: 🂳 - Codice numerico in base 10: 127155
Simbolo: 🂴 - Codice numerico in base 10: 127156
Simbolo: 🂵 - Codice numerico in base 10: 127157
Simbolo: 🂶 - Codice numerico in base 10: 127158
Simbolo: 🂷 - Codice numerico in base 10: 127159
Simbolo: 🂸 - Codice numerico in base 10: 127160
Simbolo: 🂹 - Codice numerico in base 10: 127161
Simbolo: 🂺 - Codice numerico in base 10: 127162
```

3 Qual è l'output?

Analizzate l'output del seguente programma.

```
package main

import "fmt"

func main() {

    s := "Ciao René!"
    numerocaratteri := 0
    numeroiterazione := 0
```

```
    for i, c := range s {
        fmt.Print("Iterazione ", numeroiterazione, ": In posizione ", i,
            " inizia la sottosequenza di byte che codifica il carattere ",
            string(c), "\n")
        numerocaratteri++
        numeroiterazione++
    }
    fmt.Println("Numero iterazioni:", numeroiterazione)
    fmt.Println("Numero di byte utilizzati per rappresentare la stringa:", len(s))
    fmt.Println("Numero di caratteri che definiscono la stringa:", numerocaratteri)
}
```

Osservazioni:

In generale, una stringa è una sequenza di byte. Nelle esercitazioni di laboratorio assumeremo sempre, se non esplicitato altrimenti, che una stringa sia una sequenza di byte che rappresenta una sequenza di caratteri codificati secondo lo standard Unicode/UTF-8:

- Ogni carattere è rappresentato da una sequenza di bit definita dallo standard Unicode/UTF-8 la cui lunghezza varia da 1 a 4 byte (1 byte = 8 bit). Per i caratteri considerati nello standard US-ASCII, integrato nello standard Unicode, la sequenza di bit è lunga 1 byte. Per un carattere non considerato nello standard US-ASCII, la sequenza di bit può essere lunga da 2 a 4 byte (ad esempio, la sequenza di bit prevista dallo standard Unicode/UTF-8 per rappresentare il carattere `è` è lunga 2 byte).
- In generale, il numero di caratteri che definiscono una stringa `s` è minore o uguale al numero di byte utilizzati per rappresentare la stringa `s` (`len(s)`).
- Per lunghezza di una stringa `s` si intende il numero di byte utilizzati per rappresentare `s` (`len(s)`).

Ad ogni iterazione del ciclo definito dal costrutto `for range`, `i` (variabile di tipo `int`) indica la posizione in cui inizia la sottosequenza di byte che rappresenta il carattere `string(c)`. `c` è una variabile di tipo `rune`, il cui valore è un intero corrispondente al codice Unicode del carattere.

4 Qual è l'output?

Qual è l'output del seguente programma?

```
package main

import "fmt"

func main() {

    s := "ciao, come va?"
    /* s è interamente definita da caratteri considerati nello standard US-ASCII */

    fmt.Println(string(s[0]) + string(s[len(s)-2]) + string(s[len(s)-1]))
}
```

5 Qual è l'output?

Qual è l'output del seguente programma?

```
package main

import "fmt"

func main() {

    s := "Ciao, come va?"
    // s è interamente definita da caratteri considerati nello standard US-ASCII

    for i := 0; i<len(s); i++ {
        fmt.Print(string(s[i]))
```

```
    }
    fmt.Println()

    s = "Ciao, come è andata?"
    // s non è interamente definita da caratteri considerati nello standard US-ASCII

    for i := 0; i<len(s); i++ {
        fmt.Print(string(s[i]))
    }
    fmt.Println()

}
```

Osservazioni:

- Data una stringa `s`, `s[i]` è il byte in posizione `i` nella sequenza di byte che rappresenta `s`. In generale, `s[i]` **non** codifica un carattere.
- In generale, per esaminare in sequenza i caratteri che definiscono una stringa si deve utilizzare il costrutto `for range`.

6 Qual è l'output?

Qual è l'output del seguente programma?

```
package main

import "fmt"

func main() {

    s := "ciao, come va?"
    /* s è interamente definita da caratteri considerati nello standard US-ASCII */

    fmt.Println(s[6:10] + string(s[len(s)-1]))
    fmt.Println(s[:5] + s[len(s)-4:])

}
```

7 Analisi lettere maiuscole/minuscole (1)

Scrivere un programma che legga da **standard input** una stringa senza spazi e, considerando **solamente** l'insieme delle lettere dell'alfabeto inglese, stampi

- il numero di lettere maiuscole;
- il numero di lettere minuscole;
- il numero di consonanti;
- il numero di vocali.

A tal fine definire le seguenti funzioni: 'èLetteraValida(l rune) bool', 'èMaiuscola(l rune) bool', 'èVocale(l rune) bool'.

Suggerimento: Le lettere dell'alfabeto inglese sono considerate nello standard US-ASCII (integrato nello standard Unicode). Per i caratteri considerati nello standard US-ASCII, il codice Unicode varia tra 0 e 127. In particolare, per le lettere dell'alfabeto inglese il codice Unicode varia negli intervalli:

- [65, 90] per le lettere MAIUSCOLE (con 'A'=65 e 'Z'=90)
- [97, 122] per le lettere minuscole (con 'a'=97 e 'z'=122)

Sia `c` una varibile di tipo `rune`, i seguenti blocchi di codici sono sintatticamente/semanticamente corretti:

```
if (c >='A' && c<='Z') || (c>='a' && c<='z'){
    fmt.Println(string(c), "è una lettera dell'alfabeto inglese!")
```

```
}else{
    fmt.Println(string(c), "non è una lettera dell'alfabeto inglese!")
}
// Si noti che la funzione unicode.IsLetter(c) del package unicode potrebbe
// restituire 'true' anche per caratteri che corrispondono a lettere ma che
// non appartenengono all'alfabeto inglese
```

Esempio d'esecuzione:

```
$ go run analisi.go
Ciaoà
Maiuscole: 1
Minuscole: 3
Vocali: 3
Consonanti: 1

$ go run analisi.go
Certo!Sto,bene!ìiiiìì
Maiuscole: 2
Minuscole: 10
Vocali: 5
Consonanti: 7

$ go run analisi.go
aaAA
Maiuscole: 2
Minuscole: 2
Vocali: 4
Consonanti: 0
```

8 Trasformazione lettere maiuscole/minuscole

Scrivere un programma che legga da **standard input** una stringa e, considerando l'insieme delle lettere dell'alfabeto inglese, ristampi a video la stringa due volte: la prima volta in maiuscolo e la seconda volta in minuscolo.

A tal fine definire due funzioni: 'inMaiuscolo(carattere rune) rune' e 'inMinuscolo(carattere rune) rune'

Suggerimento: Sia `c` una varibile di tipo `rune`, i seguenti blocchi di codici sono sintatticamente/semanticamente corretti:

```
if (c >='A' && c <= 'Z') {
    fmt.Println("L'equivalente lettera minuscola è:", string('a' + (c - 'A')))
}
// ... oppure, utilizzando il package "unicode"...
if (c >='A' && c <= 'Z') {
    fmt.Println("L'equivalente lettera minuscola è:", string(unicode.ToLower(c)))
}
```

Esempio d'esecuzione:

```
$ go run trasforma.go
TestoDiProva!!!
Testo maiuscolo: TESTODIPROVA!!!
Testo minuscolo: testodiprova!!!

$ go run trasforma.go
Testo_Di_PrOvA....
Testo maiuscolo: TESTO_DI_PROVA....
Testo minuscolo: testo_di_prova....
```

9 Generazione sottostringhe

Scrivere un programma che:

1. legga da **standard input** una stringa senza spazi ed interamente definita da lettere dell'alfabeto inglese;
2. stampi iterativamente la stringa ottenuta eliminando la prima e l'ultima lettera dalla stringa corrente.

Esempio d'esecuzione:

```
$ go run sottostringhe.go
Parola in input: Prova
Sottostringa 1: Prova
Sottostringa 2: rov
Sottostringa 3: o

$ go run sottostringhe.go
Parola in input: Faro
Sottostringa 1: Faro
Sottostringa 2: ar
```

10 Spaziatura caratteri

Scrivere un programma che:

1. legga da **standard input** una stringa senza spazi ed interamente definita da lettere dell'alfabeto inglese;
2. stampi la stessa stringa in modo tale che ogni lettera sia separata da quella successiva da uno spazio.

Esempio d'esecuzione:

```
$ go run spazia.go
Inserisci una stringa di testo: CiaoMondo!
C i a o M o n d o !
```

11 Analisi lettere maiuscole/minuscole (2)

Scrivere un programma che legga da **standard input** una stringa senza spazi e, considerando l'insieme delle lettere dell'alfabeto inglese, stampi

- il numero di vocali maiuscole;
- il numero di vocali minuscole;
- il numero di consonanti maiuscole;
- il numero di consonanti minuscole.

A tal fine definire le seguenti funzioni: 'èLetteraValida(l rune) bool', 'èMaiuscola(l rune) bool', 'èVocale(l rune) bool'.

Alternativamente, è possibile anche utilizzare le funzioni 'unicode.IsLetter' e 'unicode.IsUpper' del package 'unicode' al posto di 'èLetteraValida' e 'èMaiuscola' rispettivamente. Che differenze ci sono?

Esempio d'esecuzione:

```
$ go run analisi.go
Ciao
Vocali maiuscole: 0
Consonanti maiuscole: 1
Vocali minuscole: 3
Consonanti minuscole: 0

$ go run analisi.go
Certo!Sto,bene
Vocali maiuscole: 0
Consonanti maiuscole: 2
Vocali minuscole: 5
Consonanti minuscole: 5
```

```
$ go run analisi.go
aaAA
Vocali maiuscole: 2
Consonanti maiuscole: 0
Vocali minuscole: 2
Consonanti minuscole: 0
```

12 Rombo

Scrivere un programma che legga da **standard input** un numero intero `n` e, come mostrato nell'**Esempio di esecuzione**, stampi a video un rombo con diagonale maggiore e diagonale minore di lunghezza `2*n+1` utilizzando il carattere `*` (asterisco).

A tal fine definire due stringhe: 'stringaSpazi' e 'stringaAsterischi' contenenti il massimo numero dei caratteri necessari e utilizzare le loro sottostringhe per la stampa.

Se `n` è negativo o nullo, anziché stampare il rombo il programma deve stampare un messaggio d'errore.

Esempio d'esecuzione:

```
$ go run rombo.go
3
   *
  ***
 *****
*******
 *****
  ***
   *

$ go run rombo.go
0
Dimensione non sufficiente
```

13 Stringa alternata

Scrivere un programma che legga da **standard input** due stringhe senza spazi `s1` e `s2` e stampi a video la stringa creata alternando i caratteri delle stringhe `s1` e `s2`.

A tal fine utilizzare una funzione 'StringheAlternate(s1, s2 string) (risultato string)'

Esempio: Se `"ciao!"` e `"MONDO"` sono le stringhe lette, allora la stringa stampata video deve essere `"cMiOaNoD!O"`.

Si assuma che le stringhe lette siano interamente definite da caratteri considerati nello standard US-ASCII.

Se le stringhe lette non sono definite dallo stesso numero di caratteri, si deve utilizzare il carattere '-' come carattere di riempimento:

Esempio: Se `"ciao"` e `"mondo!"`, sono le stringhe lette, allora la stringa stampata video deve essere `"cmioanod-o-!"`.

Esempio d'esecuzione

```
$ go run stringa_alternata.go
ciao
mondo
cmioanod-o

$ go run stringa_alternata.go
ciaone
mondo
```

```
cmioanodnoe-

$ go run stringa_alternata.go
esame
go
egsoa-m-e-
```

14 Parola palindroma

Definizione: Una parola è palindroma se può essere letta normalmente, da sinistra verso destra, sia viceversa, cioè da destra verso sinistra.

Scrivere un programma che:

- legga da **standard input** una stringa senza spazi;
- stampi a video il messaggio `Palindroma` nel caso in cui la stringa letta sia palindroma e `Non palindroma` altrimenti.

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

- una funzione `ÈPalindroma(s string) bool` che riceve in input un valore `string` nel parametro `n` e restituisce `true` se `s` è palindroma e `false` altrimenti.

Esempio d'esecuzione:

```
$ go run palindroma.go
anna
Palindroma

$ go run palindroma.go
anni
Non palindroma

$ go run palindroma.go
osso
Palindroma

$ go run palindroma.go
èssè
Palindroma

$ go run palindroma.go
èsse
Non palindroma
```