Lab06 – Funzioni

Esempio 1 - Cosa stampa questo codice?

```
func test1(x int) (y int) {
  y = x * 10
  x, y = y, x
  return
}

func main() {
  var a, b int = 10, 5
  b = test1(a)
  fmt.Println(a, b)
}
```

Esempio 2 - Cosa stampa questo codice?

```
var a int = 10

func test1() int {
  a += 5
  return a
}

func test2(a int) int {
  a += 6
  return a
}
```

```go
func test3() int {
  return a + 7
}

func main() {
  var a, b, c int
  a, b, c = test1(), test2(a), test3()
  fmt.Println(a, b, c)
}
```

Esempio 3 - Trova e correggi gli errori

```go
func test(x int) y int, z int
{
  y := x + 1
  z := x + 2
  return
}

func main() {
  var a, b int
  a, b := test(10)
  fmt.Println(a, b)
}
```

Esercizio 1 – Numeri perfetti/amichevoli/primi gemelli sotto soglia

Si ricordi che:

- un numero naturale è **perfetto** se è uguale alla somma dei suoi divisori propri (per esempio, 6 = 1 + 2 + 3 è perfetto);
- due numeri naturali x e y, x < y, sono detti **amichevoli** se la somma dei divisori propri di ciascuno è uguale all'altro (i divisori propri di un numero sono tutti i divisori del numero - compreso l'1 – tranne il numero stesso); ad esempio (220, 284) è una coppia di amichevoli, essendo 284 = 1 + 2 + 4 + 5 + 10 + 11 + 20 + 22 + 44 + 55 + 110 (dove 1, 2, 4, 5, 10, 11, 20, 22, 44, 55, 110 sono i divisori di 220) e 220 = 1 + 2 + 4 + 71 + 142 (dove1, 2, 4, 71, 142 sono i divisori di 284); altri numeri amichevoli sono ad esempio le coppie 1184 e 1210, 2620 e 2924, 5020 e 5564, 6232 e 6368.
- un numero naturale è **primo** se è divisibile solo per se stesso o per 1; due primi p e q sono **gemelli** se p = q + 2.

Considerate il codice scritto per risolvere gli esercizi 10, 11, 12, 13 e 14 della lezione "Lab03 – Selezione/Iterazione", ristrutturatelo definendo funzioni in modo da riutilizzarlo per scrivere un programma che:

- riceva in input un numero intero strettamente positivo (che chiameremo `soglia`);
- legga un valore intero che specifica il tipo di computazione da effettuare:
  1: identificazione e stampa di numeri perfetti sotto soglia;
  2: identificazione e stampa di numeri amichevoli sotto soglia;
  3: identificazione e stampa di primi gemelli sotto soglia;
  (gestendo l'insertimento di un valore di scelta non compreso in [1, 3]);
- stampi a video l'output corrispondente.

  Per esempio:

  a) Se il valore di soglia fosse 1000 e venisse scelta l'opzione 1, l'output corrispondente sarebbe:

```
Numeri perfetti inferiori a 1000:
6
28
496
```

  b) Se il valore di soglia fosse 3000 e venisse scelta l'opzione 2, l'output corrispondente sarebbe:

```
Numeri amichevoli inferiori a 3000:
(220, 284)
(1184, 1210)
(2620, 2924)
```

  c) Se il valore di soglia fosse 15 e venisse scelta l'opzione 3, l'output corrispondente sarebbe:

```
Numeri primi gemelli inferiori a 15:
3, 5
5, 7
11, 13
```

Esercizio 2 - Garibaldi fu ferito

Scrivete un programma che legga una serie di righe in input. Dopo aver letto ogni riga, il programma la stampa sostituendo tutte le vocali con delle "`u`". Il programma si deve fermare quando viene inserita la riga vuota, stampando la stringa: "`Riga vuota, termino`".

Suggerimento

Utilizzate una funzione che accetti un carattere e lo restituisca trasformato opportunamente.

Tale funzione avrà segnatura:

```
func modificaCarattere(ch rune) rune
```

Esempio di funzionamento (in carattere grassetto le linee inserite dall'utente):

```
> go run garibaldi.go
Inserisci le righe:
mamma mia
mummu muu
bella aiuola
bullu uuuulu
Garibaldi fu ferito
Gurubuldu fu furutu

Riga vuota, termino.
```

Successivamente, modificate il programma in modo che invece di sostituire tutte le vocali con delle "`u`", le sostituisca con la *prima vocale* che compare in ogni riga. A tal fine modificate la funzione in modo che prenda come argomento anche la lettera da sostituire.

Esempio di funzionamento (in carattere grassetto le linee inserite dall'utente):

```
> go run garibaldiVariante.go
mamma mia
mamma maa
bella aiuola
belle eeeele
Garibaldi fu ferito
Garabalda fa farata

Riga vuota, termino.
```

Esercizio 3 - Alfabeto farfallino

I bimbi a volte usano uno speciale alfabeto, detto *alfabeto farfallino*. L'alfabeto farfallino consiste nel sostituire, a ciascuna vocale, una sequenza di tre lettere della forma vocale-f-vocale. Per esempio, alla vocale *a* viene sostituita la sequenza *afa*, alla vocale *e* la sequenza *efe* e così via. Notate che se una vocale V è maiuscola le tre lettere da sostituire ad essa dovranno essere maiuscole (ad esempio, A si trasforma in A-F-A) .
Scrivere un programma che legge un testo inserito su più righe dall'utente; al termine del testo il programma stampa la sua traduzione in alfabeto farfallino.

Suggerimento

Utilizzate una funzione del tipo:

```
func toFarfallino(ch rune) string
```

`toFarfallino` prende in input un carattere `ch` e restituisce una stringa che contiene la opportuna codifica del carattere in alfabeto farfallino. Ad esempio:

`toFarfallino('a')` deve restituire la stringa `"afa"`

`toFarfallino('B')` deve restituire la stringa "`B`"

`toFarfallino(',')` deve restituire la stringa "`,`"

`toFarfallino('L')` deve restituire la stringa "`L`"

Esempio di funzionamento:

Se il file `testo.txt` contenesse le seguenti linee:

```
Mamma mia
Aiuola                                    (Notate la A maiuscola)
farfAlla
```

utilizzando il comando di redirezione da file per lanciare il programma, l'output sarebbe il seguente:

```
> go run farfallino.go
Inserisci il testo:
Mafammafa mifiafa
AFAifiufuofolafa                          (la A maiuscola è stata soppiantata da AFA)
fafarfAFAllafa
```

Esercizio 4 - La strana sillabazione

Il professor Precisini, dell'*Accademia della crusca*, sostenendo che le regole di sillabazione della lingua italiana sono troppo complesse e piene di eccezioni, propone un nuovo e originale metodo di sillabazione. Il metodo consiste in questo: una sillaba è una sequenza massimale di caratteri consecutivi che rispettano l'ordine alfabetico. In tale ordine alfabetico, che segue l'ordine dei codici Unicode(US-ASCII) dei caratteri, le lettere **maiuscole sono minori di qualsiasi lettera minuscola**.

Per esempio, la parola *ambire* viene sillabata come *am-bir-e*; infatti, la lettera *a* precede la lettera *m*, e le lettere *b*, *i* e *r* rispettano anch'esse l'ordine.

Analogamente, la parola *sotterfugio* viene sillabata come *s-ott-er-fu-gio*.

Considerando invece la parola *MaRaChELla,* essa sillabata come *Ma-Ra-Ch-ELl-a*;

infatti, *M < a* poichè *M* è maiuscola,

a > R, *a > C* e *h > E* poichè *R, C, E* sono maiuscole,

e infine *L < l, l > a.*

Scrivete un programma che legge un testo scritto dall'utente e stampa la versione sillabata.

In prima istanza scrivete il programma supponendo che le lettere inserite siano solo maiuscole o solo minuscole.

Modificate poi il programma supponendo di considerare un testo con lettere sia maiuscole che minuscole (Se il vostro codice sfrutta l'ordinamento secondo lo standard Unicode, non dovrete fare modifiche!)

Esempio di funzionamento

Se il file `testo.txt` contenesse le seguenti linee:

```
amore Amore
medicina meDicO
MARACHELLA MaRaChELla
```

Lanciando il programma con il comando di redirezione da file:

```
> go run sillabazione.go < testo.txt
```

creerebbe il seguente output:

```
amor-e Amor-e
m-e-di-cin-a m-e-Di-c-O
M-AR-ACH-ELL-A Ma-Ra-Ch-ELl-a
```

Esercizio 5 - Il cifrario di Cesare

Giulio Cesare usava per le sue corrispondenze riservate un codice di sostituzione molto semplice, nel quale la lettera chiara veniva sostituita dalla lettera che la segue di tre posti nell'alfabeto: la lettera A è sostituita dalla D, la B dalla E… e così via fino alle ultime lettere che sono cifrate con le prime.

Più in generale si dice *codice di Cesare* un codice nel quale la lettera del messaggio chiaro viene spostata di un numero fisso *k* di posti, non necessariamente tre.

Scrivete un programma che chieda in input una stringa da cifrare e il numero *k* (la chiave di cifratura), e che emetta in output la stringa cifrata; il programma deve cifrare solo le lettere dell'alfabeto inglese, mantenendo minuscole le minuscole, e maiuscole le maiuscole, mentre deve lasciare inalterati gli altri simboli. Assumete il caso speciale di una stringa definita da una sequenza di caratteri che corrispondono a caratteri dell'alfabeto inglese.

Per provare se il programma funziona, cifrate un messaggio con una certa chiave *k* e poi applicate al risultato una nuova cifratura con chiave 26 − *k*: il risultato dovrebbe essere la stringa originale (cfr. "Suggerimento").

Esempio di funzionamento (in grassetto l'output inserito dll'utente)

```
> go run cesare.go
Frase da cifrare: mamma li Turchi
Chiave: 5
Frase cifrata: rfrrf qn Yzwhmn

> go run cesare.go
Frase da cifrare: rfrrf qn Yzwhmn
Chiave: 21
Frase cifrata: mamma li Turchi


> go run cesare.go
Frase da cifrare: A;B;C;d e f!
Chiave: 9
Frase cifrata: J;K;L;m n o!
```

Suggerimento

Dato un carattere `ch` (maiuscolo o minuscolo), per ottenere il carattere `chShift`, "spostato" circolarmente di *k* posti rispetto a `ch`, ricordando che ogni carattere corrisponde a un codice Unicode (US-ASCII) e che il totale dei caratteri alfabetici è 26, usate l'aritmetica modulare. Quindi:

nel caso `ch` sia maiuscolo, ottenete il carattere `chShift` con un'espressione del tipo:

```
chShift = ( ( ch - 'A' ) + k ) % 26 + 'A'
```

nel caso `ch` sia minuscolo, ottenete il carattere `chShift` con un'espressione del tipo:

```
chShift = ( ( ch - 'a' ) + k ) % 26 + 'a'
```

Esercizio 6 - Spaziate

Scrivete un programma che legga un testo formato da un numero variabile di righe e lo ristampi con spaziatura doppia, ovvero inserisca uno spazio tra ogni coppia di caratteri che non sono già spazi. Si faccia riferimento alla documentazione della funzione `unicode.IsSpace(r rune) bool` (comando: `go doc unicode.IsSpace`).

Esempio di funzionamento (in grassetto l'output inserito dall'utente)

```
> go run spaziate.go
Inserisci un testo:
Ma Ugo, non l'abbiamo mai fatto!
Uffa!
Testo a doppia spaziatura:
M a U g o , n o n l ' a b b i a m o m a i f a t t o !
U f f a !
```

Esercizio 7 - Massimo prefisso/suffisso comune

Scrivete un programma che legga due stringhe inserite dall'utente e ne stampi il massimo prefisso e il massimo suffisso comune. Il *massimo prefisso comune* di due stringhe $s = s_0 s_1 \cdots s_{m-1}$ e $t = t_0 t_1 \cdots t_{p-1}$ è la sequenza iniziale di **s** e ***t,*** ovvero è $pref(s,t) = s_0 s_1 \cdots s_j = t_0 t_1 \cdots t_j$ di massima lunghezza (cioè di massimo *j*). Notate che deve essere $s_0 = t_0, s_1 = t_1, \ldots, s_j = t_j$. Analogamente il massimo suffisso comune $suff(s,t)$ di due stringhe ***s*** e ***t,*** è la sottostringa finale che appartiene sia a ***s*** che a ***t,*** tale che $suff(s,t), = s_{m-j} s_{m-j+1} \cdots s_{m-1} = t_{p-j} t_{p-j+1} \cdots t_{p-1}$, ovvero $s_{m-j} = t_{p-j}, s_{m-j+1} = t_{m-j+1}, \ldots, s_{m-1} = t_{p-1}$. Attenzione: come vedete dagli esempi, non è detto che il suffissso e il prefissso comune esistano.

Esempio di funzionamento (in grassetto l'output inserito dll'utente)

```
> go run prefSuff.go
Prima stringa: mamma
Seconda stringa: madre
Massimo prefisso: 'ma'
Massimo suffisso: ''

Prima stringa: pippo
Seconda stringa: pappo
Massimo prefisso: 'p'
Massimo suffisso: 'ppo'

Prima stringa: foo
Seconda stringa: foo
Massimo prefisso: 'foo'
Massimo suffisso: 'foo'
```