Lab03 – Selezione/Iterazione

NOTA BENE

Da questo laboratorio, lo scheletro dei vostri programmi deve essere

```
//Commento del Package/programma
package main
import(
            “fmt”
            // ...eventuali package da importare
)
var ... //eventuali variabili globali
func main() {
            …
            // CODICE DEL PROGRAMMA
}
```

Per i nomi potete usare lettere maiuscole, minuscole e numeri, purché il nome inizi con una lettera. Ricordatevi che il nome del file sorgente ***non deve necessariamente*** essere ***main***.

- Per leggere dati da tastiera potete usare la funzione Scan(…) del package `fmt`
- Per stampare, usate le funzioni `Println(…)`, che va a capo, `Print(...)`, che non va a capo, del package `fmt`

Esempio 1 - Cosa stampa il seguente codice?

```
func main() {
  var a = 10
  var b = 20

  a = a + b
  c := a + b

  fmt.Println(c)
}
```

Esempio 2 - Cosa stampa il seguente codice?

```
func main() {
  var (
     a = 10
     b = 20
     c = 30
  )

  if a > b {
     a = b
  } else {
     b = a
  }

  c = c + b + a

  fmt.Println(a, b, c)
}
```

Esempio 3 - Trova e correggi gli errori

```
func main() {
  var a
  a = 10

  var b int
  b := 20

  c = 30

  d := a + b + c

  fmt.Println((d)
}
```

Esercizio 1 – Indovina il numero

Create un file sorgente di nome `Indovina.go` contenente il seguente programma:

```
// L'utente deve inserire un numero da tastiera
// per cercare di indovinare il numero fissato nel codice
package main

import "fmt"

func main() {
	var daindovinare, x int = 10, 0
	fmt.Println("Inserisci un tentativo:")
	fmt.Scan(&x)
	if x == daindovinare {
		fmt.Println("Indovinato")
	} else {
		if x < daindovinare {
			fmt.Println("Troppo piccolo")
		} else {
			fmt.Println("Troppo grande")
		}
	}
}
```

Compilate il sorgente appena creato tramite il comando da terminale:

```
>go fmt indovina.go
```

Se il programma è formattato senza errori, usate il comando "`go run`" (vedete gli esempi di esecuzione sotto) per compilare ed eseguire il programma (se la compilazione ha successo).

ESEMPI DI ESECUZIONE:

```
>go run indovina.go
Inserisci un tentativo:
7
Troppo piccolo

>go run indovina.go
Inserisci un tentativo:
10
Indovinato

>go run indovina.go
Inserisci un tentativo:
15
Troppo grande
```

Provate diversi input sulla riga di comando e osservate il risultato.

DOMANDA: Che cosa succede a runtime se invece che delle cifre, l'utente inserisce dei caratteri che *non* sono un numero?

Esercizio 2 – Area di un poligono regolare

Scrivere un programma che chieda l'inserimento di due numeri che chiameremo **`n`** e **`l`** e calcoli l'area di un poligono regolare con **`n`** lati di lunghezza **`l`**.

Tale area si calcola come $A = \frac{n * l^2}{4 * \tan\left(\frac{\pi}{n}\right)}$ dove *tan(·)* è l'operatore che restituisce la tangente di un numero (usate `math.tan(x)` per calcolare la tangente di un valore double `x`). Il valore π è invece immagazzinato nella costante `Pi` del package `Math` (per richiamare tale costante scrivete `math.Pi`).

Per calcolare il valore da assegnare alla variabile `area` usando tale equazione, l'istruzione in codice `go` sarà:

```
area :=(n*math.Pow(l,2))/(4*math.Tan(math.Pi/n))
```

Esercizio 3 – Valore maggiore
(da fare solo se avete saltato la lezione Lab02)

Create un programma che legge due numeri interi inseriti da tastiera dall'utente e scrive qual è il più grande. Prevedete anche il caso in cui i due numeri siano uguali.

I numeri sono inseriti in due righe (quindi leggeteli tramite tramite le funzioni `Scanln` o `Scanf` del package `fmt`). Per stampare il numero maggiore utilizzate l'istruzione if, che ha la forma:

```
if  <condizione>  {
    ...
} else {
...
}
```

Il primo blocco viene eseguito se la condizione è vera, il secondo se è falsa. Come condizione potete utilizzare un confronto tra variabili intere, utilizzando gli operatori booleani:

`==` (uguale),

`!=` (diverso),

`<` (minore), `<=` (minore o uguale)

`>` (maggiore), `<=` (maggiore o uguale)

Esercizio 4 – Valore pari o dispari

Create un programma che legge un numero intero inserito da tastiera e scrive se il numero è pari o dispari (per stabilire se un numero è pari o dispari usate l'operatore di modulo).

Esercizio 5 – Somma, differenza, maggiore/minore, media, ... (primo ciclo "for")

Create un programma che legge da standard input (ovvero da tastiera) due numeri interi, che chiameremo **`x`** e **`y`**.

Letti i due numeri, il programma ne stampa:

a il maggiore tra **`x`** e **`y`** (chiamiamolo **`max(x,y)`**)

b il minore tra **`x`** e **`y`** (chiamiamolo **`min(x,y)`**)

c il risultato della somma **`x+y`**;

d il risultato della differenza **`max(x,y)-min(x,y)`**;

e il risultato della divisione **`x/y`** (ATTENZIONE! SI PUO' SEMPRE OTTENERE UN RISULTATO VALIDO?)

f il risultato del prodotto **`x*y`**

g il risultato dell'elevamento a potenza x^y.

Per calcolare questo risultato usate un ciclo `for` del tipo:

```
for i:=0; i<y; i++{
        ...
}
for i:=y; i>0; i--{
        ...
}
```

h il valor medio tra **x** e **y**

|----------|

Create poi un programma molto simile ma che legge da standard input due numeri REALI (ovvero di tipo `float64`) disposti su un'unica linea e separati da spazi, e quindi li utilizza per calcolare e stampare tutti i valori sopra elencati.

Esercizio 6 – Conversione secondi/minuti/ore/...

Scrivere un unico programma che:

- legga un valore intero che specifica il tipo di conversione da effettuare :

1: secondi (inseriti dall'utente) in ore
2: secondi inseriti dall'utente in minuti
3: minuti inseriti dall'utente in ore
4: minuti inseriti dall'utente in secondi
5: ore inserite dall'utente in secondi
6: ore inserite dall'utente in minuti
7: minuti inseriti dall'utente in giorni e ore
8: minuti inseriti dall'utente in anni e giorni.

- gestisca l'insertimento di un valore di scelta non compreso in [1, 8];
- legga un valore reale da convertire;
- stampi a video il valore converitito.

SUGGERIMENTI:

Il codice iniziale del vostro programma potrebbe essere:

```
package main

import "fmt"

func main() {

     var valore float64
     var scelta int
```

```
fmt.Print("Scegli la conversione:\n")
fmt.Print("1: secondi -> ore\n" +
fmt.Print("2: secondi -> minuti\n" +
fmt.Print("3: minuti -> ore\n" +
fmt.Print("4: minuti -> secondi\n" +
fmt.Print("5: ore -> secondi\n" +
fmt.Print("6: ore -> minuti\n" +
fmt.Print("7: minuti -> giorni e ore\n" +
fmt.Print("8: minuti -> anni e giorni\n" +

fmt.Scan(&scelta)

fmt.Print("Inserisci il valore da convertire: ")

fmt.Scan(&valore)

// ...
```

Implementate ora più costrutti if {...} else {...} nidificati in modo da eseguire l'opportuna conversione selezionata dall'utente e stampare quindi il valore convertito a video.

Esercizio 7 - Operazioni con n numeri

Scrivete un programma che legge un numero intero `n` inserito da standard input e quindi chiede all'utente di inserire `n` numeri interi. Dopo aver letto gli `n` numeri, il programma stampa:

a) la somma degli `n` numeri letti

b) il valore minimo letto

c) il valore massimo letto

d) il numero di interi strettamente positivi, strettamente negativi, nulli. Un numero è strettamente positivo se è maggiore di zero (non vale l'uguaglianza con zero).

SUGGERIMENTO

Per leggere i numeri usate un ciclo `for` che, per `n` volte, usa la funzione `Scan`.

Esempio di funzionamento (in grassetto il valore inserito dall'utente):

```
>go run nnumeri.go
9
Inserisci 9 numeri
1 -2 3 -4 5 -6 7 -8 9
```

```
somma = 5
valore minimo = -8
valore massimo = 9
interi > 0 = 5
interi < 0 = 4
interi = 0 = 0
```

Esercizio 8 - Medie

Data una sequenza di n numeri reali $x_1, x_2, \ldots, x_n$, sono dette rispettivamente:

$$media\ aritmetica = \frac{x_1 + x_2 + \ldots + x_n}{n}$$

$$media\ geometrica = \sqrt[n]{x_1 x_2 \ldots x_n}$$

$$media\ quadratica = \sqrt[2]{\frac{x_1^{\,2} + x_2^{\,2} + \ldots + x_n^{\,2}}{n}}$$

$$media\ armonica = \frac{n}{\frac{1}{x_1} + \frac{1}{x_2} + \ldots + \frac{1}{x_n}}$$

Scrivete un programma che chieda all'utente di inserire un numero intero n, e quindi legga una sequenza di n numeri reali strettamente positivi (ovvero >0).

Per ogni valore inserito x_i il programma controlla che esso sia STRETTAMENTE positivo, *ovvero* $x_i > 0$; se ciò non accade il valore non viene considerato. Altrimenti il programma utilizza anche x_i per il calcolo delle 4 medie. Per svolgere l'esercizio usate `math.Sqrt(a)` e `math.Pow(a, b)` del package `math` che permettono di calcolare la radice quadrata $\sqrt{a}$ e la potenza a^b, essendo `a` e `b` due valori reali (`float64`).

SUGGERIMENTO

Per leggere la sequenza di numeri reali dovete usare un ciclo che usa la funzione `fmt.Scan(&x)`, dove `x` è una variabile di tipo `float64`. Ad ogni iterazione del ciclo potete:

- controllare che l'input appena letto sia strettamente positivo;
- se il valore è maggiore di zero aggiornare i valori che vi servono per calcolare le varie medie utilizzando il nuovo valore letto; se il valore è minore o uguale a zero non si fa nulla.

Per esempio, per poter calcolare la media aritmetica vi servirà una variabile (che indicheremo con `numLetti` in questo suggerimento) in cui tenere il *numero* di valori letti fino a quel momento e una variabile (detta "accumulatore", e indicata con `somma` in questo suggerimento) contenente la *somma* dei valori letti fino a quel momento (di che tipo devono essere queste due variabili?). Ogni volta che viene letto un nuovo valore si incrementa di uno la variabile `numLetti` e si incrementa la variabile accumulatrice `somma` del valore letto (entrambe le variabili **devono** essere inizializzate a 0 prima del ciclo). State attenti a non incrementare né `numLetti` né `somma` quando leggete un valore nullo o negativo.

Esempio di esecuzione (in grassetto i numeri inseriti dall'utente)

```
>go run medie.go
Inserisci un numero: 3
2
4
5
Media aritmetica: 3.666667
Media geometrica: 3.419952
Media quadratica: 3.872983
Media armonica: 3.157895
```

Notate che, dal momento che i valori negativi non vanno considerati, avrei ottenuto lo stesso output se i valori inseriti fossero stati:

```
>go run medie.go
Inserisci un numero: 10
-3
-7
-4
2
-600
-1000
-3000
4
5
-1
Media aritmetica: 3.666667
Media geometrica: 3.419952
```

```
Media quadratica: 3.872983
Media armonica: 3.157895
```

Esercizio 9 - Tabelline

Scrivete un programma che, dopo aver richiesto in input da tastiera un numero intero `n`, stampa la corrispondente tabellina, moltiplicando `n` per i numeri interi da 1 a 10, come indicato nel seguente esempio di esecuzione:

Esempio di funzionamento (in grassetto il valore inserito dall'utente):

```
>go run tabelline.go
Inserisci un numero: 9
1 x 9 = 9
2 x 9 = 18
3 x 9 = 27
4 x 9 = 36
5 x 9 = 45
6 x 9 = 54
7 x 9 = 63
8 x 9 = 72
9 x 9 = 81
10 x 9 = 90
```

Esercizio 10 – Divisori

Implementate un programma che legge un intero inserito da standard input e ne stampa i **divisori propri**.

Tenete presente che i **divisori propri** di un numero sono tutti i suoi divisori, escluso il numero stesso. Ad esempio, i **divisori** del numero `12` sono: `1, 2, 3, 4, 6, 12`

quindi i **divisori propri** di `12` sono: `1, 2, 3, 4, 6`

SUGGERIMENTO:

Per controllare se un numero `num` è divisibile per un numero `div` basta controllare che `num%div==0` dove l'operatore "%" indica l'operatore modulo (resto della divisione di `num` con `div`). Ovvero:

`8%2=0, 9%2=1, 11%3=2, 36%3=0, 39%5=4,…`

Esempi di funzionamento (in grassetto il valore inserito dall'utente):

```
>go run divisori.go
Inserisci numero: 15
```

```
divisori di 15: 1 3 5
>go run divisori.go
Inserisci numero: 286
divisori di 286: 1 2 11 13 22 26 143
```

Domanda: dovete per forza scorrere tutti gli interi tra 1 e n-1 e vedere se essi sono divisori di n, o potete limitarvi a scorrere gli interi minori o uguali n/2? Perché?

Dopo aver sviluppato la corretta implementazione, create una copia del programma e modificatela in modo che essa legga il numero e stampi:

1) i divisori del numero
2) il numero dei divisori del numero
3) la somma dei divisori del numero

Esempio di funzionamento (in grassetto il valore inserito dall'utente):

```
>go run divisoriVariazione.go
Inserisci numero: 15
divisori di 15: 1 3 5
somma dei 3 divisori = 9
>go run divisori Variazione.go
Inserisci numero: 286
divisori di 286: 1 2 11 13 22 26 143
somma dei 7 divisori = 218
>go run divisori Variazione.go
Inserisci numero: 138769
divisori di 138769: 1 151 919
somma dei 3 divisori = 1071
```

Esercizio 11 – Numeri Primi

Implementate un programma che legge un intero maggiore di 0 inserito da standard input e dice se il numero è primo. Un numero intero è primo se è divisibile solo per se stesso o per 1.

Per controllare che un numero n sia primo dovete quindi scorrere tutti gli interi minori di n e controllare che il numero n non sia divisibile per alcuno di essi.

Esempi di funzionamento (in grassetto il valore inserito dall'utente):

```
>Inserisci un numero: 12
Composto
>Inserisci un numero: 13
Primo
```

Esercizio 12 – Numeri perfetti

Un numero è perfetto se è uguale alla somma dei suoi divisori propri. Per esempio, `6 = 1 + 2 + 3` è perfetto. Scrivete un programma che, dato un intero `n` inserito da standard input scriva se esso è perfetto.

Esempi di funzionamento (in grassetto il valore inserito dall'utente):

```
$ go run perfetti.go
Inserisci un numero: 10
10 non è un numero perfetto
$ go run es9.go
Inserisci un numero: 6
6 è un numero perfetto
```

Esercizio 13 – Numeri Amichevoli

Due numeri interi `(x, y)` sono detti *amichevoli* se la somma dei divisori propri di uno è uguale all'altro (fra i divisori è compreso l'1 ed è escluso il numero stesso), e viceversa. Ad esempio `(220,284)` è una coppia di amichevoli, essendo

`284 = 1 + 2 + 4 + 5 + 10 + 11 + 20 + 22 + 44 + 55 + 110` (che sono i divisori di `220`)

`220 = 1 + 2 + 4 + 71 + 142` (che sono i divisori di `284`).

Scrivete un programma che, chieda in ingresso all'utente due numeri interi, che chiameremo `n1` e `n2`; il programma deve quindi controllare e comunicare all'utente se `n1` e `n2` sono amichevoli.

SUGGERIMENTO

Cercate i divisori di `n1` (con un ciclo), e sommateli man mano che li trovate (con un accumulatore…). Se la somma dei divisori di `n1` è diversa da `n2`, allora avete già finito poiché `n1` e `n2` non sono amichevoli; ALTRIMENTI, procedete cercando i divisori di `n2` e quindi controllando se anche la somma dei divisori di `n2` sia uguale a `n1`.

Esempio di funzionamento richiesto (in grassetto i numeri inseriti dall'utente):

```
>go run numeriAmichevoli.go
Inserire primo intero:
```

```
220
Inserire secondo intero:
284
Divisori di 220: 1 2 4 5 10 11 20 22 44 55 110.
Somma = 284
Divisori di 284: 1 2 4 71 142
Somma = 220
220 e 284 sono numeri amichevoli

>go run numeriAmichevoli.go
Inserire primo intero:
3
Inserire secondo intero:
7
Divisori di 3: 1
Somma 1
3 e 7 non sono numeri amichevoli
```

Esercizio 14 – Primi Gemelli

Due primi p e q sono *gemelli* se `p = q + 2`. Scrivete un programma che legge da standard input due numeri e controlla se sono primi gemelli.

Esempio di funzionamento richiesto (in grassetto i numeri inseriti dall'utente):

```
> go run primiGemelli.go
Inserisci un numero: 3
Inserisci un numero: 5
3 e 5 sono primi gemelli

> go run primiGemelli.go
Inserisci un numero: 7
Inserisci un numero: 11
7 e 11 non sono primi gemelli
```