Laboratorio 3 - Selezione e Iterazione

Selezione - 1 Qual è l'output?

```
package main

import "fmt"

func main() {
    var (
        a, b int = 10, 20
        c int = 30
    )
    if a > b {
        a = b
    } else {
        b = a
    }
    c = c + b + a
    fmt.Println(a, b, c)
}
```

Selezione - 2 Qual è l'output?

Supponendo che l'utente inserisca da **standard input** `5 8 6`

```
package main
import "fmt"

func main() {

    var a, b, c int
    var m int

    fmt.Scan(&a, &b, &c)

    if a < b {
        if a < c {
            m = a
        } else {
            m = c
        }
    } else {
        if b < c {
            m = b
        } else {
            m = c
```

```
        }
    }

    fmt.Println(m)

}
```

Selezione - 4 Qual è l'output?

Confronta i programmi che seguono. Cosa stampano nel caso in cui l'utente inserisca da **standard input** il valore `150`? E se invece inserisse `40`? I due programmi funzionano allo stesso modo?

```
package main

import "fmt"

func main() {

    var a int

    fmt.Scan(&a)

    if a < 100 {
        fmt.Println("a minore di 100")
    } else if a < 200 {
        fmt.Println("a compreso tra 100 e 200")
    } else {
        fmt.Println("a maggiore o uguale a 200")
    }

}
```

```
package main

import "fmt"

func main() {

    var a int

    fmt.Scan(&a)

    if a < 100 {
        fmt.Println("a minore di 100")
    }
    if a < 200 {
```

```
        fmt.Println("a compreso tra 100 e 200")
    } else {
        fmt.Println("a maggiore o uguale a 200")
    }

}
```

Selezione - Operatori logici

```
package main

import "fmt"

func main() {

    numero := 15

    // Controlla se il numero è tra 10 e 20 usando AND logico (&&)
    if numero >= 10 && numero <= 20 {
        fmt.Println("Il numero è compreso tra 10 e 20.")
    } else {
        fmt.Println("Il numero non è compreso tra 10 e 20.")
    }

    // Controlla se il numero è minore di 5 o maggiore di 50 usando OR logico (||)
    if numero < 5 || numero > 50 {
        fmt.Println("Il numero è minore di 5 o maggiore di 50.")
    } else {
        fmt.Println("Il numero è compreso tra 5 e 50.")
    }

    // Combinazione di operatori logici e selezione
    if numero >= 10 && numero <= 20 && numero%2 != 0 {
        fmt.Println("Il numero è dispari e compreso tra 10 e 20.")
    } else if numero >= 10 && numero <= 20 && numero%2 == 0 {
        fmt.Println("Il numero è pari e compreso tra 10 e 20.")
    } else {
        fmt.Println("Il numero non è compreso tra 10 e 20.")
    }
}
```

Iterazione - 5 Qual è l'output?

```
package main

import "fmt"
```

```go
func main() {
    // Il ciclo più semplice, con una singola condizione.
    // (simile al while degli altri linguaggi)
    i := 1
    for i <= 3 {
        fmt.Println(i)
        i = i + 1
    }

    // Un classico ciclo `for` inizializzazione/test/incremento.
    for j := 7; j <= 9; j++ {
        fmt.Println(j)
    }

    for {
        fmt.Println("loop")
        if 3 < 10 {
            break
        }
    }
}
```

Iterazione - 6 Qual è l'output?

Supponendo che l'utente inserisca da **standard input** `10`, qual è l'output di questo programma? E se inserisse `11`? Che cosa fa questo programma?

```go
package main

import "fmt"

func main() {
    var n int
    fmt.Print("Inserisci un numero: ")
    fmt.Scan(&n)

    var i int
    for i = 0; i <= n; i += 2 {
        fmt.Print(i, " ")
    }
    fmt.Println()
}
```

Iterazione - 7 Qual è l'output?

Supponendo che l'utente inserisca da **standard input** `36`, qual è l'output di questo programma? E se inserisse `32`? Che cosa fa questo programma? Quali sono le differenze con il programma dell'esercizio precedente?

```
package main

import "fmt"

func main() {

    var n int

    fmt.Print("Inserisci un numero: ")
    fmt.Scan(&n)

    var i int
    for i = 1; i < n; i *= 2 {
        fmt.Print(i, " ")
    }

    fmt.Println()

}
```

Iterazione - 8 Trova gli errori

Questo programma è errato e non produce l'output descritto nel commento del package. Correggilo (3 errori: 1 di sintassi, 2 di logica del programma).

```
/*
Il programma stampa una sequenza di numeri

Dato un numero n inserito da tastiera, il programma stampa tutti i numeri
compresi nell'intervallo che va da 1 a n (estremi inclusi).
La sequenza è prodotta su un'unica riga di testo in cui ciascun numero è separato dal prece
*/
package main

import (
    "fmt"
)

func main() {

    var n int
```

```
    fmt.Print("Inserisci un numero: ")
    fmt.Scan(&n)

    var i int;
    for i = 1; i < n {
        fmt.Print(i)
        i++
    }

    fmt.Println()

}
```

9 Intero con segno

Scrivere un programma che legge da **standard input** un numero intero `n` (specificato senza segno se maggiore o uguale a 0) e stampi a video il numero con segno.

Esempio d'esecuzione:

```
$ go run interoconsegno.go
Inserisci numero: 5
+5

$ go run interoconsegno.go
Inserisci numero: 0
0

$ go run interoconsegno.go
Inserisci numero: -5
-5
```

10 Multiplo di 10

Scrivere un programma che legge da **standard input** un numero intero `n` e verifica se il numero è multiplo di 10.

Suggerimento: per verificare se un numero sia multiplo di 10 potete utilizzare l'operatore `%` che calcola il resto della divisione tra interi.

Esempio d'esecuzione:

```
$ go run multiplo10.go
Inserisci numero: 15
15 non è multiplo di 10
```

```
$ go run multiplo10.go
Inserisci numero: 20
20 è multiplo di 10
```

11 Intervallo

Scrivere un programma che legga da **standard input** un voto `v` da 0 a 100 e stampi:
* `Insufficiente` se il voto è inferiore a 60 (`v<60`) * `Sufficiente` se il voto è compreso tra 60 e 70 (`v>=60` e `v<70`) * `Buono` se il voto è compreso tra 70 e 80 (`v>=70` e `v<80`) * `Distinto` se il voto è comrpeso tra 80 e 90 (`v>=80` e `v<90`) * `Ottimo` se il voto è compreso tra 90 e 100 (`v>=90` e `v<=100`) * `Errore` se il voto è negativo o superiore a 100

Esempio d'esecuzione:

```
$ go run voto.go
Inserisci il voto: 75
Buono

$ go run voto.go
Inserisci il voto: 90
Ottimo

$ go run voto.go
Inserisci il voto: 110
Errore
```

12 Fizz Buzz

Scrivere un programma che legge da **standard input** un numero intero e stampa `"Fizz"` se il numero è multiplo di 3, `"Buzz"` se il numero è multiplo di 5, `"Fizz Buzz"` se è multiplo sia di 3 sia di 5, niente altrimenti.

Esempio d'esecuzione:

```
$ go run fizzbuzz.go
Inserisci un numero: 5
Buzz
$ go run fizzbuzz.go
Inserisci un numero: 4

$ go run fizzbuzz.go
Inserisci un numero: 15
Fizz Buzz
```

```
$ go run fizzbuzz.go
Inserisci un numero: 6
Fizz
```

13 Pari o dispari

Scrivere un programma che legge da **standard input** un intero `n` e stampa a video se il numero è pari o dispari.

Esempio d'esecuzione:

```
$ go run paridispari.go
Inserisci un numero: 10
10 è pari

$ go run paridispari.go
Inserisci un numero: 11
11 è dispari
```

14 Divisione

Scrivere un programma che legga da **standard input** due numeri interi `a` e `b` e calcoli il risultato della divisione `a/b`. Se `b` è uguale a 0, il programma stampa `Impossibile`.

Esempio d'esecuzione:

```
$ go run divisione.go
Inserisci due numeri:
5 2
Quoziente = 2.5

$ go run divisione.go
Inserisci due numeri:
5 0
Impossibile
```

15 Angoli di un triangolo

Scrivere un programma che legga da **standard input** le ampiezze di due angoli di un triangolo e stampi, se possibile, l'ampiezza del terzo angolo.

Suggerimento: ricordatevi che in un triangolo la somma delle ampiezze degli angoli interni è sempre 180°.

Esempio d'esecuzione:

```
$ go angolitriangolo.go
Inserire le ampiezze dei due angoli: 50 60
Ampiezza terzo angolo = 70°

$ go angolitriangolo.go
Inserire le ampiezze dei due angoli: 150 70
I due angoli non appartengono ad un triangolo
```

16 Conversioni

Scrivere un unico programma che: - legga da **standard input** 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

`gestendo l'insertimento di un valore di scelta non compreso tra 1 e 8;`

- legga da **standard input** un valore reale da convertire;
- stampi a video il valore convertito.

Esempio d'esecuzione:

```
$ go run conversioni.go
Scegli la conversione:
1) secondi -> ore
2) secondi -> minuti
3) minuti -> ore
4) minuti -> secondi
5) ore -> secondi
6) ore -> minuti
7) minuti -> giorni e ore
8) minuti -> anni e giorni
: 8
Inserisci il valore da convertire: 7200
7200 minuti corrispondono a 0 anni e 5 giorni

$ go run conversioni.go
Scegli la conversione:
1) secondi -> ore
2) secondi -> minuti
3) minuti -> ore
```

```
4) minuti -> secondi
5) ore -> secondi
6) ore -> minuti
7) minuti -> giorni e ore
8) minuti -> anni e giorni
: 1
Inserisci il valore da convertire: 3618
3618 secondi corrispondono a 1.005 ore

$ go run conversioni.go
Scegli la conversione:
1) secondi -> ore
2) secondi -> minuti
3) minuti -> ore
4) minuti -> secondi
5) ore -> secondi
6) ore -> minuti
7) minuti -> giorni e ore
8) minuti -> anni e giorni
: 9
Scelta errata
```

17 Retta

Scrivere un programma che legga da **standard input** 4 valori a virgola mobile: - i primi due valori sono il coefficiente angolare `m` e il termine noto `q` di una retta `r: y = m*x + q` - il terzo e il quarto valore sono le coordinate `px` e `py` di un punto `P(px,py)`

Il programma deve determinare se il punto `P` sta sopra o sotto la retta od appartiene ad essa, e stampare a video il relativo messaggio.

Suggerimento: un punto appartiene ad una retta se sostituendo le sue coordinate nell'equazione della retta l'uguaglianza è verificata. Un punto sta sopra una retta se sostituendo il valore dell'ascissa nell'equazione della retta si ottiene `y < py`.

```
$ go run retta.go
Inserisci m e q: 1 0
Inserisci x e y: 5 5
Il punto appartiene alla retta

$ go run retta.go
Inserisci m e q: 1 1
Inserisci x e y: 5 5
Il punto sta sotto la retta
```

18 Tabellina

Scrivere un programma che, dopo aver richiesto all'utente di inserire da **standard input** un numero intero `n`, stampi a video la corrispondente tabellina (moltiplicando `n` per i numeri naturali da 1 a 10) come mostrato nell'**Esempio d'esecuzione**.

Esempio d'esecuzione:

```
$ 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
```

19 Somma di numeri in intervallo

Scrivere un programma che legga da **standard input** due numeri interi `a` e `b` e stampi a video la somma dei numeri pari compresi tra `a` e `b` (estremi esclusi).

Esempio d'esecuzione:

```
$ go run sommaintervallo.go
1 9
Somma = 20
```

20 Operazioni con un numero variabile di valori

Scrivere un programma che, dopo aver letto da **standard input** un numero intero `n`, chiede all'utente di inserire `n` numeri interi (sempre da **standard input**).

Dopo aver letto gli `n` numeri interi, il programma deve stampare: * la somma degli `n` numeri letti; * il minimo tra i numeri letti; * il massimo tra i numeri letti; * il numero di interi letti strettamente positivi (maggiori di 0), strettamente negativi (minori di 0), e nulli.

Esempio d'esecuzione:

```
$ 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
```

21 Media

Scrivere un programma che legga da **standard input** una sequenza di numeri reali strettamente positivi (un numero è strettamente positivo se è maggiore di 0; se un numero è minore o uguale 0 non è strettamente positivo). La lettura termina quando viene letto un numero minore o uguale a 0.

Il programma deve stampare a video il risultato della media aritmetica dei valori inseriti.

Esempio d'esecuzione:

```
$ go run medie.go
Inserisci una sequenza di numeri (interrompi con numero<=0): 4 6 8 0
Media aritmetica: 6

$ go run medie.go
Inserisci una sequenza di numeri (interrompi con numero<=0): 3 5 2 6 1 -1
Media aritmetica: 3.4
```

22 Indovina il numero

Scrivere un programma che:

1) Genera in modo casuale un numero intero compreso nell'intervallo che va da 1 a 100, estremi inclusi (sia `numeroGenerato` la variabile intera in cui viene memorizzato il numero generato, come indicato nella consegna deve valere 1<= `numeroGenerato` <= 100).

2) Chiede iterativamente all'utente di inserire da **standard input** un numero intero; ad ogni iterazione il programma controlla se il numero inserito è uguale al numero memorizzato in `numeroGenerato`:

- se sono uguali, il programma termina;
- se sono diversi, il programma fornisce un suggerimento specificando se il numero inserito è più alto o più basso di quello memorizzato in `numeroGenerato`.

L'output stampato a video dal programma deve essere quello riportato nell'**Esempio d'esecuzione** (eccezion fatta per i numeri inseriti dall'utente).

Suggerimento: per generare in modo casuale un numero intero, potete utilizzare le funzioni del package `math/rand` come mostrato nel seguente frammento di codice:

```
/* inizializzazione del generatore di numeri casuali */
rand.Seed(int64(time.Now().Nanosecond()))
/* generazione di un numero casuale compreso nell'intervallo
   che va da 0 a 99 (estremi inclusi) */
var numeroGenerato int = rand.Intn(100)
```

Esempio d'esecuzione:

```
$ go run indovina.go
Tentativo n° 1: 50
Troppo basso! Riprova!
Tentativo n° 2: 75
Troppo alto! Riprova!
Tentativo n° 3: 63
Troppo basso! Riprova!
Tentativo n° 4: 68
Troppo alto! Riprova!
Tentativo n° 5: 66
Hai indovinato in 5 tentativi!
```

23 Fizz Buzz

Scrivere un programma che legga da **standard input** un intero `n`. Il programma deve stampare a video la sequenza di numeri che vanno da 1 a `n` come mostrato nell'**Esempio d'esecuzione**. In particolare: * ogni numero divisibile per 3 deve essere rimpiazzato dalla parola `"Fizz"`; * ogni numero divisibile per 5 deve essere rimpiazzato dalla parola `"Buzz"`; * ogni numero divisibile sia per 3 sia per 5 deve essere sostituito da `"Fizz Buzz"`.

Esempio d'esecuzione:

```
$ go run fizzbuzz.go
6
1 2 Fizz 4 Buzz Fizz

$ go run fizzbuzz.go
20
1 2 Fizz 4 Buzz Fizz 7 8 Fizz Buzz 11 Fizz 13 14 FizzBuzz 16 17 Fizz 19 Buzz
```

24 Divisori

Scrivere un programma che legga da **standard input** un numero intero `n` e stampi a video i **divisori propri** del numero letto, ovvero 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.

Esempio d'esecuzione:

```
$ go run divisori.go
Inserisci numero: 6
Divisori di 6: 1 2 3

$ go run divisori.go
Inserisci numero: 10
Divisori di 10: 1 2 5
```

25 Operazioni

Create un programma che legge da **standard input** due numeri interi, che chiameremo `x` e `y`. Letti i due numeri, il programma stampa: * il maggiore tra `x` e `y` * il minore tra `x` e `y` * il risultato della somma tra `x` e `y` * il risultato della differenza tra il maggiore e il minore * il risultato della divisione `x/y` * il risultato del prodotto `x*y` * il valore medio tra `x` e `y` * il risultato di `x` elevato alla `y` (utilizzando sia un ciclo `for` sia la funzione `math.Pow`)

Esempio d'esecuzione:

```
$ go run operazioni.go
Inserisci due numeri interi:
2 4
Maggiore: 4
Minore: 2
Somma: 6
Differenza: 2
Prodotto: 8
Divisione: 0.5
Valore medio: 3
Potenza (ciclo for): 16
Potenza (math.Pow): 16
```

26 Sequenza crescente/decrescente

Il programma legge da tastiera una serie di numeri > -1 e dopo ogni lettura stampa "crescente" se il nuovo valore è maggiore o uguale al precedente e "dcrescente" altrimenti. Il programma si ferma quando il numero in input è <= -1. Utilizzare fmt.Scan per la lettura.

```
$ go run sequenza.go
inserisci una sequenza di numeri:
```

```
1 3 2 5 6 3 1 -1

crescente
crescente
decrescente
crescente
crescente
decrescente
decrescente
```

27 Fibonacci

Create un programma che legge da **standard input** un intero positivo `n` e stampi i primi `n` elementi della successione di Fibonacci. La successione di Fibonacci (detta anche successione aurea) è una successione di numeri interi in cui ciascun numero è la somma dei due precedenti.

```
$ go run fibonacci.go
Inserisci un numero intero:
4

1 1 2 3

$ go run fibonacci.go
Inserisci un numero intero:
5

1 1 2 3 5

$ go run fibonacci.go
Inserisci un numero intero:
12

1 1 2 3 5 8 13 21 34 55 89 144
```