# Lab09 – Puntatori/Strutture

### Esercizio 1 - Analisi di codice – *int

```
func main() {

  var a, b, c int
  var ptr *int

  a, b, c = 10, 15, 20
  ptr = &a

  *ptr += 10
  a += 10

  ptr = &b
  *ptr += 10

  *ptr = c
  *ptr += 10

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

### Esercizio 2 - Analisi di codice – Passaggio di variabili di tipo int per indirizzo

```
func main() {

  var a, b = 15, 20

  f(&a, &b)

  fmt.Println(a, b)
```

```
}

func f(x, y *int) {
  *x, *y = *y, *x
}
```

### Esercizio 3 - Analisi di codice – Trova gli errori

```
func main() {

  var a, b int = 10, 20
  var ptr1, ptr2 *int

  ptr1 = a
  *ptr1 += 10
  *ptr2 = 100
  ptr2 = &b
  fmt.Println(a, b)
}
```

### Esercizio 4 - Analisi di codice – Strutture – Dichiarazione ed inizializzazione

```
type Persona struct {
  nome string
  cognome string
}

func main() {

  var p1, p2, p5 Persona

  p1 = Persona{"Rick", "Sanchez"}
  p2.nome = "Mario"
  p2.cognome = "Rossi"

  p3 := Persona{"Morty", "Smith"}
```

```
   p4 := Persona{cognome: "Bianchi", nome: "Paolo"}

   p6 := Persona{cognome: "Verdi"}

   fmt.Println("p1:",p1)
   fmt.Println("p5:",p5)

   fmt.Printf("Nome: %-10s - Cognome: %-10s\n", p1.nome, p1.cognome)
   fmt.Printf("%T\t%v\n", p3, p3)
   fmt.Printf("%#v\n", p4)
   fmt.Printf("%#v\n", p6)

 }
```

## Esercizio 5 - Analisi di codice – Passaggio di strutture per valore/indirizzo

```
type Persona struct {
  nome, cognome string
}

func main() {

  p := Persona{"Rick", "Sanchez"}

  fmt.Println(p)

  f(p)

  fmt.Println(p)

  g(&p)

  fmt.Println(p)
}
```

```
func f(p Persona) {
  p.nome, p.cognome = strings.ToUpper(p.nome),
strings.ToUpper(p.cognome)
}

func g(p *Persona) {
  p.nome, p.cognome = strings.ToUpper(p.nome),
strings.ToUpper(p.cognome)
}
```

## Esercizio 6 - Triangoli casuali

Utilizzando le funzionalità messe a disposizione dal `package triangolo`, scrivere un programma che:

- generi in maniera casuale 10 coppie di valori interi;
- consideri il triangolo corrispondente a ciascuna coppia di valori generati (il primo valore della coppia rappresenta la base del triangolo, il secondo valore rappresenta l'altezza del triangolo);
- stampi a video la descrizione del triangolo con area maggiore.

Nel programma non si può calcolare in maniera esplicita l'area del triangolo corrispondente ad una data coppia di valori interi. Ciascun valore intero deve essere generato nell'intervallo [1; 100].

*Esempio di funzionamento*

```
$ go run areaTriangolo.go
Le 10 coppie di valori casuali generati per base ed altezza sono:
  70 -   28
  55 -   87
  22 -   46
  14 -   28
  42 -   53
  27 -   27
  85 -   49
  28 -   64
  10 -   32
```

```
  87 -    2

Il triangolo con area maggiore è il seguente.

Triangolo con base e altezza rispettivamente pari a 55 e 87.
L'area del triangolo è pari a 2392.500
```

## Esercizio 7 - Analisi di codice – Slice come tipo/i restituito/i

```go
func main() {

  var a []int = make([]int,5)

  fmt.Println(a)

  f(a)

  fmt.Println(a)

  a = g(a)

  fmt.Println(a)
}

func f(s []int) {
  s[0] = 10
  s = append(s, 10)
  fmt.Println("s:",s)
}

func g(s []int) []int {
  s[0] = 20
  s = append(s, 20)
  fmt.Println("s:",s)
  return s
}
```

## Esercizio 8 - Analisi di codice – Slice di strutture – Dichiarazione ed inizializzazione

```go
type Persona struct {
  nome string
  cognome string
}

func main() {

  anagrafica := []Persona{
     {"Rick", "Sanchez"},
     {"Morty", "Smith"},
     {"Jerry", "Smith"},
     {"Summer", "Smith"},
     {"Beth", "Smith"}}

  fmt.Printf("Tipo anagrafica %T\n", anagrafica)
  fmt.Println()

  for _, p := range anagrafica {
     fmt.Println(p.nome, p.cognome)
  }
  fmt.Println()

  anagrafica[0].cognome = "Martinez"

  for _, p := range anagrafica {
     fmt.Println(p.nome, p.cognome)
  }
}
```

## Esercizio 9 - Rubrica

### PARTE I

Creare un programma che permetta la gestione di una rubrica:

1) Ogni contatto deve avere un nome e un cognome, un indirizzo e un numero di telefono
2) Ogni indirizzo deve contenere le seguenti informazioni: via, numero civico, CAP e città

Si ipotizzi che nella rubrica non possono esistere due contatti con lo stesso nome e lo stesso cognome.

Si devono implementare le seguenti funzionalità:

1) Una funzione per aggiungere contatti:
   Es: rubrica = aggiungiContatto(rubrica, "Mario", "Rossi", "02503111", "Via Festa del Perdono", 7, "20122", "Milano")
2) Una funzione per stampare i contatti presenti:
   Es: stampaRubrica(rubrica)
   Output:
   3 contatti in rubrica:
   [1] - Mario Rossi, Tel. 02503111, Via Festa del Perdono 11, 20122, Milano
   [2] - Anna Rossi, Tel. 02503111, Via Festa del Perdono 11, 20122, Milano
   [3] - Carlo Rossi, Tel. 02503111, Via Festa del Perdono 11, 20122, Milano
3) Una funzione per rimuovere contatti:
   Es: rubrica = eliminaContatto(rubrica, "Mario", "Rossi")
   stampaRubrica(rubrica)
   Output:
   2 contatti in rubrica:
   [1] - Anna Rossi, Tel. 02503111, Via Festa del Perdono 11, 20122, Milano
   [2] - Carlo Rossi, Tel. 02503111, Via Festa del Perdono 11, 20122, Milano
4) Una funzione per aggiornare l'indirizzo dei contatti:
   Es: rubrica = aggiornaContatto(rubrica, "Anna", "Rossi", "Via S. Sofia", 25, "20122", "Milano")
   stampaRubrica(rubrica)
   Output:
   2 contatti in rubrica:
   [1] - Anna Rossi, Tel. 02503111, Via S. Sofia 25, 20122, Milano
   [2] - Carlo Rossi, Tel. 02503111, Via Festa del Perdono 11, 20122, Milano

Per implementare la rubrica si utilizzi una slice di tipo `[]Persona`.

Deve essere fornito anche un codice all'interno del main che richiami le funzioni create in modo da testarne il corretto funzionamento.

## PARTE II

Creare un programma analogo al precedente implementando la rubrica mediante una mappa di tipo

`map[string]Persona` (ossia `rubrica := map[string]Persona{}`).

`rubrica["RossiAnna"]` indicizza nella mappa `rubrica` l'istanza di `Persona` associata ad Anna Rossi.