# Laboratorio 10 - Strutture e puntatori

## 1 Qual è l'output?

Qual è l'output del seguente programma?

```
package main

import "fmt"

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)
}
```

## 2 Qual è l'output?

Qual è l'output di questo programma?

```
package main

import "fmt"

func main() {
    var a, b = 15, 20
    f(&a, &b)
    fmt.Println(a, b)
}
func f(x, y *int) {
    *x, *y = *y, *x
}
```

## 3 Trova l'errore

Questo programma dovrebbe stampare `20 100` ma non genera l'output desiderato. Corregere e verificare che l'esecuzione del programma generi l'output atteso.

```
package main

import "fmt"

func main() {
    var a, b int = 10, 20
    var ptr *int
    ptr = a
    *ptr = *ptr + b
    fmt.Println(a, b)
}
```

## 4 Trova l'errore

Questo programma dovrebbe stampare `50` ma non genera l'output desiderato. Corregere e verificare che l'esecuzione del programma generi l'output atteso.

```
package main

import "fmt"

func main() {
    var ptr *int
    *ptr = 50
    fmt.Println(ptr)
}
```

## 5 Qual è l'output?

Qual è l'output di questo programma?

```
package main

import (
    "fmt"
    "strings"
)

type Persona struct {
    Nome, Cognome string
}

func main() {
```

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

    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)
}
```

## 6 Frazioni

Al fine di modellare l'entità matematica `frazione`, si definisca un nuovo tipo `Frazione` come una struttura avente due campi `numeratore` e `denominatore` di tipo `int`.

Implementare le funzioni:

- `NuovaFrazione(numeratore, denominatore int) *Frazione` che restituisce una nuova istanza del tipo `Frazione` inizializzata in base ai valori dei parametri `numeratore` e `denominatore`;
- `String(f Frazione) string` che riceve in input un'instanza del tipo `Frazione` nel parametro `f` e restituisce un valore `string` che corrisponde alla rappresentazione `string` di `f` nel formato `numeratore/denominatore`;

Verificare il corretto funzionamento delle funzioni utilizzando le unit di test allegate.

*Suggerimento:* per creare una stringa formattata potete utilizzare la funzione `fmt.Sprintf()` del package `fmt`. La funzione si comporta come la funzione `fmt.Printf()` ma, invece di stampare a video l'output, restituisce una `string` contenente l'output.

## 7 Riduzione di una frazione ai minimi termini

Si consideri il tipo `Frazione` dell'esercizio **5 Frazioni**.

Si modifichi il programma in modo tale che:

- legga da **riga di comando** due numeri interi `n` e `d`;
- utilizzi `n` e `d` per inizializzare una `Frazione` utilizzando `n` come numeratore e `d` come denominatore;
- riduca ai minimi termini la frazione;

- stampi a video, nel formato `numeratore/denominatore`, la frazione ridotta ai minimi termini.

Oltre alla funzione `main()` e alle funzioni implementate nell'esercizio **5 Frazioni**, devono essere definite ed utilizzate almeno le seguenti funzioni:

- una funzione `Riduci(f *Frazione)` che riceve in input un'instanza del tipo `Frazione` nel parametro `f` e, se necessario, modifica opportunamente il valore dei campi `f.numeratore` e `f.denominatore` affinché `f` rappresenti una frazione ridotta ai minimi termini.

Verificare il corretto funzionamento delle funzioni utilizzando le unit di test allegate.

**Esempio d'esecuzione:**

```
$ go run riduci.go 10 10
1/1

$ go run riduci.go 34 18
17/9

$ go run riduci.go 12 36
1/3
```

## 8 Moltiplicazione tra frazioni

Si consideri il tipo `Frazione` dell'esercizio **6 Riduzione di una frazione ai minimi termini**.

Si modifichi il programma in modo tale che:

- legga da **standard input** un testo su più righe;
- termini la lettura quando viene inserito da standard input l'indicatore EOF (`CTRL+D`).

In ogni riga sono specificati due numeri interi che rappresentano il numeratore e il denominatore di una frazione.

Una volta terminata la fase di lettura, il programma deve:

1. calcolare la frazione che si ottiene moltiplicando tra di loro le frazioni corrispondenti alle coppie di interi letti;
2. ridurre ai minimi termini la frazione calcolata al punto 1;
3. stampare a video, nel formato `numeratore/denominatore`, la frazione ridotta ai minimi termini ottenuta al punto 2.

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

- una funzione `LeggiFrazioni() []Frazione` che legge da **standard input** un testo su più righe e terminato da EOF (`CTRL+D`), restituendo un valore `[]Frazione` in cui sono memorizzate tutte le

istanze del tipo `Frazione` inizializzate in base ai numeri interi `numeratore` e `denominatore` specificati in ciascuna delle righe lette;

- una funzione `Moltiplica(f1, f2 Frazione) *Frazione` che riceve in input due istanze del tipo `Frazione` nei parametri `f1` e `f2` e restituisce una nuova istanza del tipo `*Frazione` i cui campi `numeratore` e `denominatore` sono inizializzati con i valori `f1.numeratore * f2.numeratore` e `f1.denominatore * f2.denominatore`;
- una funzione `MoltiplicaN(fN []Frazione) *Frazione` che riceve in input una slice di `Frazione` nel parametro `fN` e restituisce una nuova istanza del tipo `*Frazione` corrispondente alla frazione che si ottiene moltiplicando tra di loro le frazioni relative alle istanze del tipo `Frazione` presenti in `fN`; la funzione deve utilizzare la funzione `Moltiplica()`.

Verificare il corretto funzionamento delle funzioni utilizzando le unit di test allegate.

**Esempio d'esecuzione**:

```
$ go run prodotto.go
Inserisci numeratore e denominatore delle frazioni:
1 2
4 3
2 5
Prodotto: 4/15

$ go run prodotto.go
Inserisci numeratore e denominatore delle frazioni:
1 1
2 3
1 8
Prodotto: 1/12
```