# Laboratorio 12 - Ricorsione

## 1 Qual è l'output?

Analizziamo l'output del seguente programma.

```
package main

import (
    "fmt"
    "strings"
)

func main() {
    fmt.Printf("Funzione 'main' - Inizio\n")
    Conteggio(3)
    fmt.Printf("Funzione 'main' - Fine\n")
}

func Conteggio(n uint) {
    rientro := strings.Repeat("\t", int(3-n))
    fmt.Printf("%sFunzione 'Conteggio' - Inizio esecuzione per n = %d\n", rientro, n)
    if n == 0 {
        fmt.Printf("%sPartenza!\n", rientro)
    } else {
        fmt.Printf("%s%d\n", rientro, n)
        Conteggio(n - 1)
    }
    fmt.Printf("%sFunzione 'Conteggio' - Fine esecuzione per n = %d\n", rientro, n)
}
```

## 2 Qual è l'output?

Qual è l'output del seguente programma?

```
package main

import "fmt"

func main() {
    conteggio(10)
    fmt.Println("Partenza!")
    fmt.Println()
    conteggioAlternativo(10)
    fmt.Println("Partenza!")
```

```
}
func conteggio(n uint) {
    if n != 0 {
        fmt.Println(n)
        conteggio(n - 1)
    }
}
func conteggioAlternativo(n uint) {
    if n != 0 {
        conteggioAlternativo(n - 1)
        fmt.Println(n)
    }
}
```

## 3 Qual è l'output?

Analizziamo il funzionamento del seguente programma.

```
package main

import "fmt"

func main() {
    fmt.Println("La somma dei numeri da 0 a 10 è", Somma(10))
}

func Somma(n uint) uint {
    if n == 0 {
        return 0
    } else {
        return n + Somma(n-1)
        // ... equivalente a:
        // var somma_da_0_fino_a_n_meno_1 uint = Somma(n-1)
        // return n + somma_da_0_fino_a_n_meno_1
    }
}
```

## 4 Qual è l'output?

Qual è l'output del seguente programma? Quale proprietà viene testata?

```
package main

import "fmt"

func main() {
```

```
    if ProprietàSoddisfatta([]rune("1012982101")) {
        fmt.Println("La proprietà testata è soddisfatta.")
    } else {
        fmt.Println("La proprietà testata non è soddisfatta.")
    }

    if ProprietàSoddisfatta([]rune("1212982101")) {
        fmt.Println("La proprietà testata è soddisfatta.")
    } else {
        fmt.Println("La proprietà testata non è soddisfatta.")
    }

}

func ProprietàSoddisfatta(sl []rune) bool {
    fmt.Println(string(sl))
    if len(sl) <= 1 {
        return true
    } else {
        soddisfa := sl[0]==sl[len(sl)-1]
        if soddisfa {
            soddisfa = ProprietàSoddisfatta(sl[1:len(sl)-1])
        }
        return soddisfa
        // ... equivalente a:
        // soddisfa := sl[0] == sl[len(sl)-1]
        // return soddisfa && ProprietàSoddisfatta(sl[1:len(sl)-1])
    }
}
```

## 5 Somma

Scrivere un programma che legga da **riga di comando** una sequenza di numeri interi e stampi a video il risultato della somma dei numeri letti, calcolato utilizzando una funzione ricorsiva.

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

- una funzione ricorsiva `Somma(sl []int) (somma int)` che riceve in input un valore `[]int` nel parametro `sl` e restituisce un valore `int` pari alla somma dei valori presenti in `sl`.

**Esempio d'esecuzione:**

```
$ go run somma.go 1 4 2 7
Somma valori: 14

$ go run somma.go 10
Somma valori: 10

$ go run somma.go 10 4 24 17
```

```
Somma valori: 55

$ go run somma.go 1 ciao 5
Somma valori: 6
```

## 6 Righe invertite

Scrivere un programma che:

1. legga da **standard input** una stringa senza spazi;
2. inverta la stringa letta utilizzando una funzione ricorsiva e stampi a video la stringa ottenuta.

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

- una funzione ricorsiva `InvertiStringa(s string) string` che riceve in input un valore `string` nel parametro `s` e restituisce un valore `string` in cui il primo carattere è l'ultimo che definisce `s`, il secondo carattere è il penultimo che definisce `s`, ... e l'ultimo carattere è il primo che definisce `s`.

**Esempio d'esecuzione:**

```
$ go run inverti_stringa.go
programmazione
enoizammargorp

$ go run inverti_stringa.go
laboratorio
oirotarobal
```

## 7 Righe

Scrivere un programma che:

- legga da **standard input** un testo su più righe (alcune delle quali possono essere delle righe vuote ("")), terminando la lettura quando, premendo la combinazione di tasti Ctrl+D, viene inserito da **standard input** l'indicatore End-Of-File (EOF);
- utilizzando una funzione ricorsiva, stampi a video le righe lette (dalla prima letta all'ultima letta).

Oltre alla funzione `main()` di seguito riportata, deve essere definita ed utilizzata la sola funzione ricorsiva `LeggiEStampa(scanner *bufio.Scanner)` che riceve in input, nel parametro `scanner`, un'instanza del tipo `bufio.Scanner` inizializzata al fine di permettere la lettura da **standard input**.

Per verificare il corretto funzionamento del programma, effettuare la redirezione dello standard output sul file `righe.txt`.

```
func main() {
    scanner := bufio.NewScanner(os.Stdin)
    LeggiEStampa(scanner)
}
```

**Esempio d'esecuzione:**

```
$ go run righe.go
Questa è una riga
Questa è una riga
Seconda riga in input
Seconda riga in input
Ogni riga viene stampata immediatamente
Ogni riga viene stampata immediatamente


Anche le righe vuote sono stampate
Anche le righe vuote sono stampate
```

## 8 Righe invertite

Scrivere un programma che:

- legga da **standard input** un testo su più righe (alcune delle quali possono essere delle righe vuote ("")), terminando la lettura quando, premendo la combinazione di tasti Ctrl+D, viene inserito da **standard input** l'indicatore End-Of-File (EOF);
- utilizzando una funzione ricorsiva, stampi a video le righe lette invertendone l'ordine (dall'ultima letta alla prima letta).

Oltre alla funzione `main()` di seguito riportata, deve essere definita ed utilizzata la sola funzione ricorsiva `LeggiEStampa(scanner *bufio.Scanner)` che riceve in input, nel parametro `scanner`, un'instanza del tipo `bufio.Scanner` inizializzata al fine di permettere la lettura da **standard input**.

Per verificare il corretto funzionamento del programma, effettuare la redirezione dello standard output sul file `righe_invertite.txt`.

```
func main() {
    scanner := bufio.NewScanner(os.Stdin)
    LeggiEStampa(scanner)
}
```

**Esempio d'esecuzione:**

```
$ go run inverti_righe.go
Questa è una riga
Seconda riga in input
Ogni riga viene stampata in un secondo momento

Anche le righe vuote sono stampate
Anche le righe vuote sono stampate

Ogni riga viene stampata in un secondo momento
Seconda riga in input
Questa è una riga
```

## 9 Testo invertito

Scrivere un programma che:

- legga da **standard input** un testo su più righe (alcune delle quali possono essere delle righe vuote ("")), terminando la lettura quando, premendo la combinazione di tasti Ctrl+D, viene inserito da **standard input** l'indicatore End-Of-File (EOF);
- ristampi il testo letto dall'ultimo carattere al primo.

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

- una funzione ricorsiva `InvertiStringa(s string) string` che riceve in input un valore `string` nel parametro `s` e restituisce un valore `string` in cui il primo carattere è l'ultimo che definisce `s`, il secondo carattere è il penultimo che definisce `s`, ... e l'ultimo carattere è il primo che definisce `s`;
- una funzione ricorsiva `LeggiEStampa(scanner *bufio.Scanner)` che riceve in input, nel parametro `scanner`, un'istanza del tipo `bufio.Scanner` inizializzata al fine di permettere la lettura da **standard input**.

Per verificare il corretto funzionamento del programma, effettuare la redirezione dello standard output sul file `testo_invertito.txt`.

**Esempio d'esecuzione:**

```
$ go run inverti_testo.go
Questa è una riga
Seconda riga in input

Anche le righe vuote contano
onatnoc etouv ehgir el ehcnA

tupni ni agir adnoceS
agir anu è atseuQ
```

# 10 Prodotto scalare

Scrivere un programma che:

1. Legga da **standard input** un numero intero `n`.
2. Chieda all'utente di inserire, sempre da **standard input**, due sequenze di `n` numeri interi:

   $a_0$ $a_1$ $a_2$ ... $a_n$

   $b_0$ $b_1$ $b_2$ ... $b_n$
3. Stampi a video il risultato dell'espressione

   $a_0 * b_0 + a_1 * b_1 + ... + a_n * b_n$

   calcolato utilizzando una funzione ricorsiva.

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

- una funzione ricorsiva `Moltiplica(sl1, sl2 []int) int` che riceve in input due valori `[]int`, rispettivamente nei parametri `sl1` e `sl2`, e restituisce un valore `int` pari a `sl1[0]*sl2[0] + sl1[1]*sl2[1] + ... + sl1[n-1]*sl2[n-1]`, dove `n` è pari a `len(sl1)` e `len(sl2)`.

**Esempio d'esecuzione:**

```
$ go run prodotto_scalare.go
4
1 4 2 7
2 3 8 9
Prodotto = 93
```

# 11 Divisione tra interi

Scrivere un programma che:

- legga da **riga di comando** due valori interi `dividendo` e `divisore`;
- stampi a video il *quoziente* ed il *resto* della divisione tra interi `dividendo:divisore`, calcolati utilizzando una funzione ricorsiva.

*Suggerimento:* Il risultato della divisione tra interi `dividendo:divisore` può essere calcolato sottraendo ripetutamente il `divisore` al `dividendo` fino a che il primo non diventa maggiore del secondo. Il risultato della divisione tra interi `9:2` può essere per esempio calcolato come segue:

```
9 - 2 = 7 (il 2 sta nel 9 una volta)
7 - 2 = 5 (il 2 sta nel 9 due volte)
```

```
5 - 2 = 3 (il 2 sta nel 9 tre volte)
3 - 2 = 1 (il 2 sta nel 9 quattro volte)
```

Essendo 1 minore di 2, la procedura di calcolo si interrompe. Il quoziente ed il resto della divisione `9:2` sono rispettivamente pari a `4` e `1`.

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

- una funzione ricorsiva `Dividi(dividendo, divisore int) (q, r int)` che riceve in input due valori `int`, rispettivamente nei parametri `dividendo` e `divisore`, e restituisce due valori `int` nelle variabili `q` ed `r` che rappresentano rispettivamente il quoziente ed il resto della divisione tra interi `dividendo:divisore`.

**Esempio d'esecuzione:**

```
$ go run divisione.go 15 2
Quoziente = 7, Resto = 1

$ go run divisione.go 12 3
Quoziente = 4, Resto = 0

$ go run divisione.go 9 4
Quoziente = 2, Resto = 1
```

## 12 Frazione continua

**Definizione**: Sia $a_0$ un numero intero qualsiasi e $a_1$, $a_2$, ... , $a_n$ numeri interi positivi. La notazione [$a_0$, $a_1$, ... , $a_n$] indica la frazione continua

$$a_0 + \cfrac{1}{a_1 + \cfrac{1}{a_2 + \cfrac{1}{a_3 + \dots}}}$$

Ad esempio:

$$[-1, 5, 2, 4] = -1 + \cfrac{1}{5 + \cfrac{1}{2 + \frac{1}{4}}}$$

e allo stesso modo:

$$[-7, 3, 5, 7, 9] = -7 + \cfrac{1}{3 + \cfrac{1}{5 + \cfrac{1}{7 + \frac{1}{9}}}}$$

Scrivere un programma che:

- legga da **riga di comando** una sequenza di numeri interi $a_0$ $a_1$ $a_2$ ... $a_n$ (con $a_i > 0$ per 1 <= i <= n);
- stampi a video il valore reale corrispondente alla frazione continua $[a_0, a_1, \dots, a_n]$, calcolato utilizzando una funzione ricorsiva.

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

- una funzione ricorsiva `ValoreFrazione(sl []int) float64` che riceve in input un valore `[]int` nel parametro `sl` e restituisce il valore `float64` corrispondente alla frazione continua `[sl[0], sl[1], ... , sl[n-1]]`, dove `n` è pari a `len(sl)`.

**Esempio d'esecuzione:**

```
$ go run frazioneContinua.go -1 2 2
Frazione continua: -0.6

$ go run frazioneContinua.go -1 2
Frazione continua: -0.5

$ go run frazioneContinua.go -1 4 6
Frazione continua: -0.76

$ go run frazioneContinua.go 2 4 6
Frazione continua: 2.24

$ go run frazioneContinua.go -1 4 6 5
Frazione continua: -0.7596899224806202
```

## 13 Sottoinsiemi

Scrivere un programma che legga da **standard input** una stringa senza spazi interamente definita da lettere dell'alfabeto inglese. Ogni lettera è un elemento dell'insieme rappresentato dalla stringa (ogni lettera può comparire al più una volta nella stringa).

Una volta terminata la fase di lettura, il programma deve stampare a video tutti i possibili sottoinsiemi dell'insieme rappresentato dalla stringa (insieme vuoto escluso).

Oltre alla funzione `main()` di seguito riportata, deve essere definita ed utilizzata la sola funzione ricorsiva `func Sottoinsiemi(insiemeResiduo string) []string` che riceve un valore `string` nel parametro `insiemeResiduo`, e restituisce un valore `[]string` in cui ogni elemento è uno dei possibili sottoinsiemi dell'insieme rappresentato dalla stringa `insiemeResiduo`.

```
func main() {

    insiemeResiduo := ""
```

```
    fmt.Scan(&insiemeResiduo)

    sottoinsiemi := Sottoinsiemi(insiemeResiduo)
    fmt.Printf("%d sottoinsiemi: %v\n", len(sottoinsiemi), sottoinsiemi)

}
```

**Esempio d'esecuzione:**

```
$ go run sottoinsiemi.go
ABC
7 sottoinsiemi: [C B BC A AC AB ABC]

$ go run sottoinsiemi.go
ciao
15 sottoinsiemi: [o a ao i io ia iao c co ca cao ci cio cia ciao]
```

## 14 Permutazioni

Scrivere un programma che legga da **standard input** una stringa senza spazi interamente definita da lettere dell'alfabeto inglese. Ogni lettera è un elemento dell'insieme rappresentato dalla stringa (ogni lettera può comparire al più una volta nella stringa).

Una volta terminata la fase di lettura, il programma deve stampare a video tutte le possibili permutazioni semplici degli elementi dell'insieme rappresentato dalla stringa.

Oltre alla funzione `main()` di seguito riportata, deve essere definita ed utilizzata la sola funzione ricorsiva `func Permutazioni(insiemeResiduo string) []string` che riceve un valore `string` nel parametro `insiemeResiduo`, e restituisce un valore `[]string` in cui ogni elemento è una delle possibili permutazioni degli elementi dell'insieme rappresentato dalla stringa `insiemeResiduo`.

```
func main() {

    insiemeResiduo := ""

    fmt.Scan(&insiemeResiduo)

    permutazioni := Permutazioni(insiemeResiduo)
    fmt.Printf("%d sottoinsiemi: %v\n", len(permutazioni), permutazioni)

}
```

**Esempio d'esecuzione:**

```
$ go run permutazioni.go
ABC
```

```
6 permutazioni: [ABC ACB BAC BCA CAB CBA]

$ go run permutazioni.go
ciao
24 permutazioni: [ciao cioa caio caoi coia coai icao icoa iaco iaoc ioca ioac acio acoi aico
aioc aoci aoic ocia ocai oica oiac oaci oaic]
```