Laboratorio 4 - Funzioni

1 Qual è l'output?

Qual è l'output del seguente programma?

```
package main

import "fmt"

func funzione() {
    fmt.Println("Hello world!")
}

func main() {
    funzione()
}
```

2 Qual è l'output?

Qual è l'output del seguente programma?

```
package main

import "fmt"

func funzione(x int) int {
    return x + 5
}

func main() {
    var x int = 10
    fmt.Println(funzione(x))
}
```

3 Qual è l'output?

Qual è l'output del seguente programma?

```
package main

import "fmt"
```

```
func funzione1(x int) int {
    return x * 10
}

func funzione2(x int) (y int) {
    y = x * 10
    return
}

func main() {
    fmt.Println(funzione1(5), funzione2(3))
}
```

4 Trova l'errore

Questo programma dovrebbe stampare `20 40` ma non genera l'output desiderato. Corregere e verificare che l'esecuzione sia conforme al comportamento atteso.

```
package main

import "fmt"

func test(x, y int) (int, int) {
    return 2 * x, 2 * y
}

func main() {
    var x, y int = 10, 20
    test(x, y)
    fmt.Println(x, y)
}
```

5 Trova l'errore

Questo programma contiene degli errori. Corregere gli errori ed eseguire il programma.

```
package main

import "fmt"

func test(x int) y int, z int
{
var y int = x + 1
var z int = x + 2
return
}

func main() {
```

```
    var a, b int
    a, b = test(10)
    fmt.Println(a, b)
}
```

6 Cerchio

Scrivere un programma che legga da **standard input** un numero reale `raggio` e stampi i valori di circonferenza e area di un cerchio di raggio `raggio`.

Oltre alla funzione `main()`, il programma deve definire e utilizzare le seguenti funzioni:

- `CalcolaArea(raggio float64) float64` che riceve in input il valore reale del raggio di un cerchio nel parametro `raggio` e restituisce il valore dell'area del cerchio associato;
- `CalcolaCirconferenza(raggio float64) float64` che riceve in input il valore reale del raggio di un cerchio nel parametro `raggio` e restituisce il valore della circonferenza del cerchio associato.

Esempio d'esecuzione:

```
$ go run cerchio.go
Inserisci il raggio del cerchio: 10.5
Area del cerchio: 346.36059005827474
Circonferenza del cerchio: 65.97344572538566

$ go run cerchio.go
Inserisci il raggio del cerchio: 3.6
Area del cerchio: 40.71504079052372
Circonferenza del cerchio: 22.61946710584651
```

7 Tabelline

Scrivere un programma che legga da **standard input** un numero intero e stampi la tabellina corrispondente solo se il numero inserito è compreso tra `1` e `9` (estremi inclusi).

Oltre alla funzione `main()`, il programma deve definire e utilizzare le seguenti funzioni:

- `Tabellina(numero int)` che riceve in input un valore intero nel parametro `numero` e, se `numero` è compreso tra `1` e `9` (estremi inclusi), stampa la tabellina associata, altrimenti non stampa nulla.

Esempio d'esecuzione:

```
$ go run tabellina.go
Inserisci un numero: 4
Tabellina del 4: 0 4 8 12 16 20 24 28 32 36 40
```

```
$ go run tabellina.go
Inserisci un numero: 6
Tabellina del 6: 0 6 12 18 24 30 36 42 48 54 60

$ go run tabellina.go
Inserisci un numero: 13

$ go run tabellina.go
Inserisci un numero: -1
```

8 Tabelline 2

*Estendete il programma che risolve l'esercizio **Tabelline** in modo tale che soddisfi la richiesta seguente.*

Scrivere un programma che legga da **standard input** un numero intero e stampi la tabellina corrispondente solo se il numero inserito è compreso tra `1` e `9` (estremi inclusi). Se il numero inserito non è valido (inferiore di `1` o superiore a `9`), il programma deve stampare `Numero non valido.`.

Oltre alla funzione `main()`, il programma deve definire e utilizzare le seguenti funzioni:

- `Tabellina(numero int) bool` che riceve in input un valore intero nel parametro `numero`. Se `numero` è compreso tra `1` e `9` (estremi inclusi), la funzione stampa la tabellina associata e restituisce un valore logico `true`. Altrimenti, la funzione non stampa nulla e restituisce un valore logico `false`.

Esempio d'esecuzione:

```
$ go run tabellina.go
Inserisci un numero: 4
Tabellina del 4: 0 4 8 12 16 20 24 28 32 36 40

$ go run tabellina.go
Inserisci un numero: 6
Tabellina del 6: 0 6 12 18 24 30 36 42 48 54 60

$ go run tabellina.go
Inserisci un numero: 13
Numero non valido.

$ go run tabellina.go
Inserisci un numero: -1
Numero non valido.
```

9 Tabelline 3

*Estendete il programma che risolve l'esercizio **Tabelline 2** in modo tale che soddisfi la richiesta seguente.*

Scrivere un programma che legga da **standard input** una sequenza di numeri interi compresi tra `1` e `9` (estremi inclusi) e stampi per ognuno di essi la tabellina corrispondente. Il programma si interrompe quando viene inserito in input un numero non valido (inferiore a `1` o superiore a `9`) stampando `Programma terminato.`.

Oltre alla funzione `main()`, il programma deve definire e utilizzare le seguenti funzioni:

- `Tabellina(numero int) bool` che riceve in input un valore intero nel parametro `numero`. Se `numero` è compreso tra `1` e `9` (estremi inclusi), la funzione stampa la tabellina associata e restituisce un valore logico `true`. Altrimenti, la funzione non stampa nulla e restituisce un valore logico `false`.

Esempio d'esecuzione:

```
$ go run tabellina.go
Inserisci un numero: 6
Tabellina del 6: 0 6 12 18 24 30 36 42 48 54 60
Inserisci un numero: 4
Tabellina del 4: 0 4 8 12 16 20 24 28 32 36 40
Inserisci un numero: 13
Programma terminato.
```

10 Radice quadrata

Scrivere un programma che legga da **standard input** un numero reale `n` e stampi a video il valore della radice quadrata di `n` solo se `n >= 0`. Se `n < 0` il programma deve stampare `Il valore inserito non appartiene al dominio della funzione`.

Suggerimento: Per calcolare la radice quadrata di un numero relae `n` potete usare la funzione `Sqrt` del package `math`:

```
radiceQuadrata = math.Sqrt(n)
```

Oltre alla funzione `main()`, il programma deve definire e utilizzare le seguenti funzioni:

- `RadiceQuadrata(numero float64) (float64, bool)` che riceve in input un valore reale nel parametro `numero`. Se `numero >= 0` la funzione restituisce il valore della radice quadrata di `numero` e un valore logico `true` come certificato della corretta esecuzione del calcolo. Altrimenti, la funzione restituisce un valore reale `0` e un valore logico `false`, per segnalare che non è stato possibile calcolare la radice quadrata di `numero` nel campo dei reali.

Esempio d'esecuzione:

```
$ go run radice.go
Inserisci un numero: 10
```

```
Radice quadrata: 3.1622776601683795

$ go run radice.go
Inserisci un numero: 4
Radice quadrata: 2

$ go run radice.go
Inserisci un numero: -1
Il valore inserito non appartiene al dominio della funzione
```

11 Scacchiera

Scrivere un programma che legga da **standard input** un numero intero e, come mostrato nell'**Esempio d'esecuzione**, stampi a video una schacchiera di dimensione `n x n` utilizzando i caratteri `*` (asterisco) e `+` (più).

Oltre alla funzione `main()`, il programma deve definire e utilizzare le seguenti funzioni:

- `StampaRigaInizioAsterisco(lunghezza int)` che riceve in input la lunghezza della riga da stampare nel parametro `lunghezza` e stampa in modo alternato i caratteri `*` e `+` (partendo dal carattere `*`);
- `StampaRigaInizioPiù(lunghezza int)` che riceve in input la lunghezza della riga da stampare nel parametro `lunghezza` e stampa in modo alternato i caratteri `+` e `*` (partendo dal carattere `+`);
- `StampaScacchiera(dimensione int)` che riceve in input la dimensione della scacchiera da stampare nel parametro `dimensione` e stampa la scacchiera utilizzando le funzioni `StampaRigaInizioAsterisco()` e `StampaRigaInizioPiù()`. Se `dimensione <= 0`, non stampa nulla.

Esempio d'esecuzione:

```
$ go run scacchiera.go
Inserisci la dimensione: 4
*+*+
+*+*
*+*+
+*+*

$ go run scacchiera.go
Inserisci la dimensione: 6
*+*+*+
+*+*+*
*+*+*+
+*+*+*
*+*+*+
+*+*+*

$ go run scacchiera.go
Inserisci la dimensione: -1
```

```
$ go run scacchiera.go
Inserisci la dimensione: 0
```

12 Numeri primi

Definizione: Un numero naturale è primo se è divisibile solo per se stesso e per `1`.

Scrivere un programma che legga da **standard input** un numero intero `soglia` e stampi tutti i numeri primi inferiori a `soglia`. Se `soglia <= 0` il programma deve stampare `La soglia inserita non è positiva`.

Oltre alla funzione `main()`, il programma deve definire e utilizzare le seguenti funzioni:

- una funzione `ÈPrimo(n int) bool` che riceve in input un valore intero nel parametro `n` e restituisce `true` se `n` è primo e `false` altrimenti;
- una funzione `NumeriPrimi(limite int)` che riceve in input un valore intero nel parametro `limite` e stampa tutti i numeri primi inferiori a `limite`; la funzione deve utilizzare la funzione `ÈPrimo()`.

Esempio d'esecuzione:

```
$ go run numeri_primi.go
Inserisci un numero: -3
La soglia inserita non è positiva

$ go run numeri_primi.go
Inserisci un numero: 5
Numeri primi inferiori a 5
2 3

$ go run numeri_primi.go
Inserisci un numero: 12
Numeri primi inferiori a 12
2 3 5 7 11
```

13 Numeri primi gemelli

Definizione: Due numeri primi `p` e `q` sono gemelli se `p = q + 2`.

Scrivere un programma che legga da **standard input** un numero intero `soglia` e stampi tutti i numeri primi gemelli tali che `p` sia inferiore a `soglia`. Se `soglia <= 0` il programma deve stampare `La soglia inserita non è positiva`.

Oltre alla funzione `main()`, il programma deve definire e utilizzare le seguenti funzioni:

- una funzione `ÈPrimo(n int) bool` che riceve in input un valore intero nel parametro `n` e restituisce `true` se `n` è primo e `false` altrimenti;
- una funzione `NumeriPrimiGemelli(limite int)` che riceve in input un valore intero nel parametro `limite` e stampa tutte le coppie di numeri primi gemelli tali che `p` sia inferiore a `limite` (cfr. Definizione); la funzione deve utilizzare la funzione `ÈPrimo()`.

Esempio d'esecuzione:

```
$ go run numeri_primi_gemelli.go
Inserisci un numero: -4
La soglia inserita non è positiva

$ go run numeri_primi_gemelli.go
Inserisci un numero: 10
Numeri primi gemelli inferiori a 10
(3,5) (5,7)

$ go run numeri_primi_gemelli.go
Inserisci un numero: 20
Numeri primi gemelli inferiori a 20
(3,5) (5,7) (11,13) (17,19)
```

14 Numeri perfetti

Definizione: Un numero naturale è perfetto se è uguale alla somma dei suoi divisori propri (per esempio, `6` è perfetto perché `6 = 1 + 2 + 3`).

Scrivere un programma che legga da **standard input** un numero intero `n` e stampi se `n` è perfetto oppure no.

Oltre alla funzione `main()`, il programma deve definire e utilizzare le seguenti funzioni:

- una funzione `SommaDivisoriPropri(n int) int` che riceve in input un valore intero nel parametro `n` e restituisce la somma dei divisori propri di `n`. Se `n` non ha divisori propri la funzione restituisce `0`;
- una funzione `ÈPerfetto(n int) bool` che riceve in input un valore intero nel parametro `n` e restituisce `true` se `n` è perfetto e `false` altrimenti; la funzione deve utilizzare la funzione `SommaDivisoriPropri()`.

Esempio d'esecuzione:

```
$ go run numero_perfetto.go
Inserisci un numero: 0
0 non è perfetto

$ go run numero_perfetto.go
Inserisci un numero: 1
```

```
1 non è perfetto

$ go run numero_perfetto.go
Inserisci un numero: 6
6 è perfetto

$ go run numero_perfetto.go
Inserisci un numero: 28
28 è perfetto
```

15 Numeri abbondanti

Definizione: Un numero naturale è abbondante se è inferiore alla somma dei suoi divisori propri (per esempio, `12` è abbondante perché la somma dei suoi divisori propri (`1` , `2` , `3` , `4` , `6`) è `16`).

Scrivere un programma che legga da **standard input** un numero intero `soglia` e stampi tutti i numeri abbondanti inferiori a `soglia` . Se `soglia <= 0` il programma deve stampare `La soglia inserita non è positiva` .

Oltre alla funzione `main()` , il programma deve definire e utilizzare le seguenti funzioni:

- una funzione `SommaDivisoriPropri(n int) int` che riceve in input un valore intero nel parametro `n` e restituisce la somma dei divisori propri di `n` . Se `n` non ha divisori propri la funzione restituisce `0` ;
- una funzione `ÈAbbondante(n int) bool` che riceve in input un valore intero nel parametro `n` e restituisce `true` se `n` è abbondante, `false` altrimenti; la funzione deve utilizzare la funzione `SommaDivisoriPropri()` ;
- una funzione `NumeriAbbondanti(limite int)` che riceve in input un valore intero nel parametro `limite` e stampa tutti i numeri abbondanti inferiori a `limite` ; la funzione deve utilizzare la funzione `ÈAbbondante()` ;

Esempio d'esecuzione:

```
$ go run numeri_abbondanti.go
Inserisci un numero: 20
Numeri abbondanti: 12 18

$ go run numeri_abbondanti.go
Inserisci un numero: 30
Numeri abbondanti: 12 18 20 24

$ go run numeri_abbondanti.go
Inserisci un numero: -3
La soglia inserita non è positiva
```

16 Numeri amichevoli

Definizione: Due numeri naturali `x` e `y`, con `x < y`, sono detti amichevoli se la somma dei divisori propri di ciascuno è uguale all'altro; ad esempio `(220, 284)` è una coppia di amichevoli, essendo `284 = 1 + 2 + 4 + 5 + 10 + 11 + 20 + 22 + 44 + 55 + 110` (dove `1, 2, 4, 5, 10, 11, 20, 22, 44, 55, 110` sono i divisori di `220`) e `220 = 1 + 2 + 4 + 71 + 142` (dove `1, 2, 4, 71, 142` sono i divisori di `284`).

Scrivere un programma che legga da **standard input** un numero intero `soglia` e stampi tutte le coppie di numeri amichevoli tali che `y` sia inferiore a `soglia`. Se `soglia <= 0` il programma deve stampare `La soglia inserita non è positiva`.

Oltre alla funzione `main()`, il programma deve definire e utilizzare le seguenti funzioni:

- una funzione `SommaDivisoriPropri(n int) int` che riceve in input un valore intero nel parametro `n` e restituisce la somma dei divisori propri di `n`. Se `n` non ha divisori propri la funzione restituisce `0`;
- una funzione `SonoAmichevoli(n, m int) bool` che riceve in input due valori interi nei parametri `n` ed `m` e restituisce `true` se `n` ed `m` sono amichevoli e `false` altrimenti (utilizzando la funzione `SommaDivisoriPropri()`);
- una funzione `NumeriAmichevoli(limite int)` che riceve in input un valore intero nel parametro `limite` e stampa tutte le coppie di numeri amichevoli tali che `y` sia inferiore a `limite` (cfr. **Definizione**); la funzione deve utilizzare la funzione `SonoAmichevoli()`.

Esempio d'esecuzione:

```
$ go run numeri_amichevoli.go
Inserisci un numero: 300
Numeri amichevoli inferiori a 300
(220,284)

$ go run numeri_amichevoli.go
Inserisci un numero: 0
La soglia inserita non è positiva
```

17 Terna pitagorica

Definizione: Se `a`, `b` e `c` sono numeri naturali e `a² + b² = c²`, si dice che la terna di numeri `a`, `b` e `c` è una terna pitagorica.

Scrivere un programma che legga da **standard input** un intero `soglia>0` e stampi a video tutte le terne pitagorighe tali che `a<soglia`, `b<soglia` e `c<soglia`.

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

- `ÈTernaPitagoriga(a int, b int, c int) bool` che riceve in input tre valori interi nei parametri `a`, `b` e `c`, e restituisce `true` se `c²` è uguale a `a² + b²` e `false` altrimenti;

- `TernePitagoriche(soglia int)` che riceve in input un valore intero nel parametro `soglia` e stampa tutte le terne pitagoriche inferiori a `soglia`; la funzione deve utilizzare la funzione `ÈTernaPitagoriga()`.

Esempio d'esecuzione:

```
$ go run terna_pitagorica.go
Inserisci la soglia: 10
Terne pitagoriche:
(3, 4, 5)
(4, 3, 5)

$ go run terna_pitagorica.go
Inserisci la soglia: 20
Terne pitagoriche:
(3, 4, 5)
(4, 3, 5)
(5, 12, 13)
(6, 8, 10)
(8, 6, 10)
(8, 15, 17)
(9, 12, 15)
(12, 5, 13)
(12, 9, 15)
(15, 8, 17)
```

18 Scala

Scrivere un programma che legga da **standard input** un numero intero `n` e, come mostrato nell'**Esempio d'esecuzione**, stampi a video una scala utilizzando il carattere `*` (asterisco):

- La scala è formata da `n` gradini.
- Ciascun gradino è profondo 3 caratteri e alto 2.
- Il gradino più in alto deve essere stampato senza alcuna rientranza verso destra; considerando poi i successivi gradini dall'alto verso il basso, ciascuno di essi rientra (è traslato) verso destra rispetto al precedente di due caratteri ` ` (spazio) .

Se `n` è negativo o nullo, anziché stampare la scala il programma deve stampare il messaggio d'errore `Dimensione non sufficiente`.

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

- `StampaGradino(gradino int)` che riceve in input un valore intero nel parametro `gradino` e stampa a video un singolo gradino della scala, opportunamente traslato verso destra. Se `gradino < 0` non stampa nulla. Se `gradino = 0` non stampa la rientranza. Se `gradino > 0` stampa il gradino traslato di `gradino * 2` spazi verso destra;

- `StampaScala(gradini int)` che riceve in input un valore intero nel parametro `gradini` e stampa una scala composta da `gradini` gradini.

Esempio d'esecuzione:

```
$ go run scala.go
Inserisci il numero dei gradini: 3
***
  *
  ***
    *
    ***
      *

$ go run scala.go
Inserisci il numero dei gradini: 0
Dimensione non sufficiente

$ go run scala.go
Inserisci il numero dei gradini: 2
***
  *
  ***
    *
```

19 Scala 2

Scrivere un programma che legga da **standard input** un numero intero `n` e, come mostrato nell'**Esempio d'esecuzione**, stampi a video una scala utilizzando il carattere `'*'` (asterisco):

- La scala è formata da `n` gradini.
- Ciascun gradino è profondo 3 caratteri e alto 2.
- Il gradino più in basso deve essere stampato senza alcuna rientranza verso destra; considerando poi i successivi gradini dal basso verso l'alto, ciascun di essi rientra (è traslato) verso destra rispetto al precedente di due caratteri `' '` (spazio) .

Se `n` è negativo o nullo, anziché stampare la scala il programma deve stampare un messaggio d'errore.

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

- `StampaGradino(gradino int)` che riceve in input un valore intero nel parametro `gradino` e stampa a video un singolo gradino della scala, opportunamente traslato verso destra. Se `gradino < 0` non stampa nulla. Se `gradino = 0` non stampa la rientranza. Se `gradino > 0` stampa il gradino traslato di `gradino * 2` spazi verso destra;
- `StampaScala(gradini int)` che riceve in input un intero nel parametro `gradini` e stampa una scala composta da `gradini` gradini.

Esempio d'esecuzione:

```
$ go run scala.go
Inserisci il numero dei gradini: 3
    ***
    *
  ***
  *
***
*

$ go run scala.go
Inserisci il numero dei gradini: 0
Dimensione non sufficiente

$ go run scala.go
Inserisci il numero dei gradini: 2
  ***
  *
***
*
```