

Laboratorio 8 - Array e slice II

1 Qual è l'output?

Analizziamo l'output del seguente programma.

```
package main

import (
    "fmt"
    "strconv"
)

func main() {

    var numeroIntero int = 7

    fmt.Printf("Tipo: %T\n", numeroIntero)
    fmt.Printf("- formato default: %v\n", numeroIntero)
    fmt.Printf("- formato base 10: %d\n", numeroIntero)
    fmt.Printf("- formato base 10 (larghezza = 6): %6d\n", numeroIntero)
    fmt.Printf("- formato base 2: %b\n", numeroIntero)

    fmt.Println()

    var numeroReale float64 = 14.5674
    fmt.Printf("Tipo: %T\n", numeroReale)
    fmt.Printf("- formato default: %v\n", numeroReale)
    fmt.Printf("- formato con decimali: %f\n", numeroReale)
    fmt.Printf("- formato con numero fissato di decimali: %.2f\n", numeroReale)
    fmt.Printf("- formato esponenziale: %e\n", numeroReale)

    fmt.Println()

    var valoreLogico bool = false
    fmt.Printf("Tipo: %T\n", valoreLogico)
    fmt.Printf("- formato default: %v\n", valoreLogico)
    fmt.Printf("- formato true/false: %t\n", valoreLogico)

    fmt.Println()

    var carattere rune = 'A'
    fmt.Printf("Tipo: %T\n", carattere)
    fmt.Printf("- formato default: %v\n", carattere)
    fmt.Printf("- formato carattere: %c\n", carattere)
    fmt.Printf("- formato intero: %d\n", carattere)
    fmt.Printf("- formato unicode: %U\n", carattere)

    fmt.Println()

    var stringaTesto string = "Hello\tworld!"
    fmt.Printf("Tipo: %T\n", stringaTesto)
    fmt.Printf("- formato default: %v\n", stringaTesto)
    fmt.Printf("- formato stringa: %s\n", stringaTesto)

    fmt.Println()

    var posizioniDecimali = 2
    fmt.Printf("Printf con formato creato dinamicamente %." +
        strconv.Itoa(posizioniDecimali) + "f\n", 10.458)
}
```

2 Qual è l'output?

Analizziamo l'output del seguente programma.

```
package main

import "fmt"

func main() {
    a := [...]string{5:"E", 3:"C"}
    // equivalente a: var a [6]string = [6]string{"", "", "", "C", "", "E"}

    fmt.Printf("Tipo: %T - Valore: %v\n", a, a)

    for i := range a {
        fmt.Println("Indice", len(a)-1-i, " - Valore:", a[len(a)-1-i])
    }
    fmt.Println()

    for i:=len(a)-1; i>=0; i-- {
        fmt.Println("Indice", i, " - Valore:", a[i])
    }
    fmt.Println()

    for i:=0; i<len(a); i++ {
        fmt.Println("Indice", i, " - Valore:", a[i])
    }
    fmt.Println()
}
```

3 Qual è l'output?

Analizziamo l'output del seguente programma.

```
package main

import "fmt"

func main() {
    a := [...]int{1, 2, 3, 4, 5, 6, 7}

    fmt.Printf("a - %T: %v\n", a, a)

    sl1 := a[:] // slicing
    sl2 := sl1[1:3]

    fmt.Printf("len(sl1) = %d, cap(sl1) = %d\n", len(sl1), cap(sl1))
    fmt.Printf("sl1 - %T: %v\n", sl1, sl1)
    fmt.Printf("len(sl2) = %d, cap(sl2) = %d\n", len(sl2), cap(sl2))
    fmt.Printf("sl2 - %T: %v\n", sl2, sl2)

    sl2 = sl2[:len(sl2)+1] // reslicing
    fmt.Printf("\nsl2 - %T: %v\n", sl2, sl2)

    sl2 = sl2[:cap(sl2)]
    fmt.Printf("sl2 - %T: %v\n", sl2, sl2)

    elem, sl1 := sl1[0], sl1[1:]
    fmt.Printf("\nelem - %T: %v\n", elem, elem)
    fmt.Printf("sl1 - %T: %v\n", sl1, sl1)

    /* una slice s non può essere modificata per accedere ad elementi
       dell'array (a cui si riferisce) che precedono quello contenuto in
       s[0]; l'istruzione s = s[-1:] genera un errore */
}
```

4 Qual è l'output?

Qual è l'output di questo programma?

```
package main

import "fmt"

func main() {
    var a [6]int

    for i := range a {
        a[i] = i
    }

    var b []int
    b = a[:]

    for i := range b {
        b[i] = i * 2
    }

    fmt.Println(a)
}
```

5 Qual è l'output?

Qual è l'output di questo programma?

```
package main

import "fmt"

func main() {
    var a []int
    a = []int{0, 1, 2, 3, 4, 5, 6}

    var b []int
    b = a[2:4]

    b[0] = a[0]
    b[len(b)-1] = a[0]

    fmt.Println(a)
}
```

6 Qual è l'output?

Qual è l'output di questo programma?

```
package main

import "fmt"

func main() {
    var a = [6]int{1, 2, 3, 4, 5, 6}
    stampa(a)
    modifica(a)
    stampa(a)
}

func stampa(a [6]int) {
    for _, v := range a {
```

```

        fmt.Print(v, " ")
    }
    fmt.Println()
}

func modifica(a [6]int) {
    for i := range a {
        a[i] *= 2
    }
}

```

7 Qual è l'output?

Qual è l'output di questo programma?

```

package main

import "fmt"

func main() {
    var a = [6]int{1, 2, 3, 4, 5, 6}
    stampa(a)
    a = modifica(a)
    stampa(a)
    modificaConPuntatore(&a)
    stampa(a)
}

func stampa(a [6]int) {
    for _, v := range a {
        fmt.Print(v, " ")
    }
    fmt.Println()
}

func modifica(a [6]int) [6]int{
    for i := range a {
        a[i] *= 2
    }
    return a
}

func modificaConPuntatore(a* [6]int) {
    for i := range (*a) {
        (*a)[i] *= 2
    }
}

```

8 Qual è l'output?

Qual è l'output di questo programma?

```

package main

import "fmt"

func main() {
    var a = [6]int{1, 2, 3, 4, 5, 6}
    stampa(a[:])

    b := a[:]
    b = modifica(b)
    stampa(b)
}

func stampa(sl []int) {
    for _, v := range sl {

```

```

        fmt.Print(v, " ")
    }
    fmt.Println()
}

func modifica(slIn []int) (slOut []int) {
    slOut = slIn
    for i := range slOut {
        slOut[i] *= 2
    }
    return
}

```

9 Qual è l'output?

Qual è l'output di questo programma?

```

package main

import "fmt"

func main() {
    b := []int{1,2,3,4,5}
    stampa(b)

    modifica(b)
    stampa(b)

    eliminaUltimoElemento(b)
    stampa(b)
}

func stampa(sl []int) {
    for _, v := range sl {
        fmt.Print(v, " ")
    }
    fmt.Println()
}

func modifica(sl []int) {
    for i := range sl {
        sl[i] *= 2
    }
}

func eliminaUltimoElemento(sl []int) {
    sl = sl[:len(sl)-1]
}

```

10 Qual è l'output?

Qual è l'output di questo programma?

```

package main

import (
    "fmt"
    "strconv"
)

func main() {

    sl := []int64{1, 2, 4, 5, 8, 15, 32}
    sl1 := converti(sl)
    sl2 := convertiPari(sl)
}

```

```

    fmt.Println(sl1)
    fmt.Println(sl2)
}

func converti(slIn []int64) []string {

    slOut := make([]string, len(slIn))

    for i := 0; i < len(slIn); i++ {
        slOut[i] = strconv.FormatInt(slIn[i], 2)
    }

    return slOut
}

func convertiPari(slIn []int64) []string {

    var slOut []string

    for i := 0; i < len(slIn); i++ {
        if slIn[i] % 2 == 0 {
            slOut = append(slOut, strconv.FormatInt(slIn[i], 2))
        }
    }

    return slOut
}

```

11 Qual è l'output?

Supponendo che l'utente inserisca da **riga di comando** i valori `1 a 5.6 ciao true`, cosa dovrebbe produrre in output il seguente programma?

```

package main

import (
    "os"
    "fmt"
)

func main() {
    fmt.Printf("TIPO os.Args: %T\n", os.Args)
    for i, v := range os.Args {
        fmt.Printf("os.Args[%d]: TIPO = %T - VALORE = %s\n", i, v, v)
    }
}

```

12 Filtra e moltiplica

Scrivere un programma che legga da **riga di comando** una sequenza di valori e stampi a video il risultato della moltiplicazione tra i valori che rappresentano numeri interi.

Esempio d'esecuzione:

```

$ go run prodotto.go 4 3
Il risultato della moltiplicazione tra i numeri interi è 12

$ go run prodotto.go 6
Il risultato della moltiplicazione tra i numeri interi è 6

$ go run prodotto.go 1 3 a 5 ciao 2
Il risultato della moltiplicazione tra i numeri interi è 30

```

13 Minimo, massimo e valor medio (1)

Scrivere un programma che legga da **riga di comando** una sequenza di valori interi e stampi a video il valore minimo, massimo e medio tra i valori letti.

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

- una funzione `Minimo(s1 []int) int` che riceve in input un valore `[]int` nel parametro `s1` e restituisce il minimo valore intero presente in `s1`.
- una funzione `Massimo(s1 []int) int` che riceve in input un valore `[]int` nel parametro `s1` e restituisce il massimo valore intero presente in `s1`.
- una funzione `Media(s1 []int) float64` che riceve in input un valore `[]int` nel parametro `s1` e restituisce un valore reale pari alla media aritmetica dei valori interi presenti in `s1`.

Suggerimento: Il numero di valori interi che il programma deve considerare è pari a `len(os.Args)-1`.

Nota: Per creare dinamicamente una slice, si utilizzi la funzione `make`.

Esempio d'esecuzione:

```
$ go run min_max_media.go 1 2 3 4
Minimo: 1
Massimo: 4
Valore medio: 2.50

$ go run min_max_media.go -1 10 6
Minimo: -1
Massimo: 10
Valore medio: 5.00

$ go run min_max_media.go -1 -2 -3
Minimo: -3
Massimo: -1
Valore medio: -2.00
```

14 Minimo, massimo e valor medio (2)

Scrivere un programma che legga da **riga di comando** una sequenza di valori e stampi a video il valore minimo, massimo e medio tra i valori letti che rappresentano numeri interi.

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

- una funzione `LeggiNumeri() (numeri []int)` che restituisce un valore `numeri` di tipo `[]int` in cui sono memorizzati i valori numerici interi specificati a **riga di comando**;
- una funzione `Minimo(s1 []int) int` che riceve in input un valore `[]int` nel parametro `s1` e restituisce il minimo valore intero presente in `s1`.
- una funzione `Massimo(s1 []int) int` che riceve in input un valore `[]int` nel parametro `s1` e restituisce il massimo valore intero presente in `s1`.
- una funzione `Media(s1 []int) float64` che riceve in input un valore `[]int` nel parametro `s1` e restituisce un valore reale pari alla media aritmetica dei valori interi presenti in `s1`.

Nota: Per creare dinamicamente una slice, si utilizzi la funzione `append`.

Esempio d'esecuzione:

```
$ go run min_max_media.go 1 ciao 2 pippo 3 4
Minimo: 1
Massimo: 4
Valore medio: 2.50

$ go run min_max_media.go -1 10 6 fine
Minimo: -1
```

```
Massimo: 10
Valore medio: 5.00

$ go run min_max_media.go tre -1 numeri: -2 -4
Minimo: -3
Massimo: -1
Valore medio: -2.33
```

15 Qual è l'output?

Analizziamo l'output del seguente programma.

```
package main

import (
    "fmt"
)

func main() {

    s1 := make([]int, 4, 8)

    for i := 0; i < len(s1); i++ {
        s1[i] = i+1
    }

    fmt.Println("1)\ns1:")
    stampa(s1)

    s2 := append(s1[2:], []int{10,20}...)
    fmt.Println("\n2)\ns2:")
    stampa(s2)
    fmt.Println("s1:")
    stampa(s1)

    s2[0] = -1
    s2 = append(s2, 100)
    fmt.Println("\n3)\ns2:")
    stampa(s2)
    fmt.Println("s1:")
    stampa(s1)

    s2 = append(s2, 1000, 2000)
    fmt.Println("\n4)\ns2:")
    stampa(s2)
    fmt.Println("s1:")
    stampa(s1)

    s2 = append(s2[:2], s2[len(s2)-2:]...)
    fmt.Println("\n5)\ns2:")
    stampa(s2)

}

func stampa(s []int) {
    var lunghezza int
    fmt.Println("a) ", s, len(s), cap(s))
    lunghezza = len(s)
    s = s[:cap(s)]
    fmt.Println("b) ", s, len(s), cap(s))
    s = s[:lunghezza]
}
```

16 Qual è l'output?

Qual è l'output del seguente programma?

```
package main

import (
    "fmt"
)

const Dimensione = 10

func main() {

    var a []int

    for i := 0; i < Dimensione; i++ {
        a = append([]int{i+1}, a...)
    }

    fmt.Println(a)

    b := make([]int, len(a))
    copy(b,a)
    fmt.Println(b)

    c := make([]int, Dimensione)
    copy(c[:], a[Dimensione/2:])
    fmt.Println(c)
}
```

17 Qual è l'output?

Qual è l'output del seguente programma?

```
package main

import "fmt"

func main() {

    //strFrom := "ABDEF"
    //strFrom[2] = 'C' /* error: "cannot assign to strFrom[2]" */

    /* (1) */
    strFrom1 := "ABÈEF"
    // len(sliceDiRune) è pari al numero di caratteri Unicode presenti in strFrom1
    sliceDiRune := []rune(strFrom1)
    sliceDiRune[2] = 'C'
    strTo1 := string(sliceDiRune)
    fmt.Println(strTo1)
    /* (1) - end */

    /* (2) */
    strFrom2 := "ABÈEF"
    //strTo2 := strFrom2[:2]+string('C')+strFrom2[3:]
    strTo2 := strFrom2[:2] + "C" + strFrom2[3:]
    fmt.Println(strTo2)
    /* (2) - end */

}
```

18 Date (1)

Scrivere un programma che:

- legge da **standard input** un testo su più righe;
- termina la lettura quando viene inserita da standard input una riga vuota (`" "`).

Ogni riga di testo è una stringa senza spazi che codifica una data in uno dei seguenti possibili formati:

1. `aaaa/m/g`
2. `aaaa/m/gg`
3. `aaaa/mm/g`
4. `aaaa/mm/gg`

Una volta terminata la fase di lettura, il programma deve stampare la codifica nel formato `aaaa-mm-gg` delle date lette.

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

- una funzione `LeggiTesto() []string` che legge da **standard input** un testo su più righe e terminato da una riga vuota (`" "`), restituendo un valore `[]string` in cui sono memorizzate le righe del testo letto;
- una funzione `Formatta(s string) string` che riceve in input un valore `string` nel parametro `s` in cui è codificata una data nel formato `aaaa/m/g`, `aaaa/m/gg`, `aaaa/mm/g` o `aaaa/mm/gg`, e restituisce un valore `string` in cui la data in input è codificata nel formato `aaaa-mm-gg`.

Esempio d'esecuzione:

```
$ cat test
2019/04/03
2019/6/4
2019/07/5
2019/4/05
$ go run date.go < test
2019-04-03
2019-06-04
2019-07-05
2019-04-05
```

19 Fattoriale

Definizione: Si definisce fattoriale di un numero intero positivo, il prodotto dei numeri interi positivi minori o uguali a tale numero. Il fattoriale di `k` è uguale a `1*2*3*...*(k-3)*(k-2)*(k-1)*k`.

Scrivere un programma che legga da **riga di comando** un numero intero `n` e stampi a video il fattoriale di tutti i numeri compresi tra `1` e `n` (estremi inclusi).

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

- una funzione `Fattoriali(n int) (f []int)` che riceve in input un valore `int` nel parametro `n` e restituisce il valore `f` di tipo `[]int` in cui in `f[0]` è memorizzato il fattoriale di `1`, in `f[1]` è memorizzato il fattoriale di `2`, ..., in `f[n-1]` è memorizzato il fattoriale di `n`.

Nota: Per creare dinamicamente una slice, si utilizzi la funzione `make`.

Esempio d'esecuzione:

```
$ go run fattoriale.go 2
Fattoriali: [1 2]

$ go run fattoriale.go 3
Fattoriali: [1 2 6]

$ go run fattoriale.go 10
Fattoriali: [1 2 6 24 120 720 5040 40320 362880 3628800]
```

20 Somma di prodotti

Scrivere un programma che legga da **riga di comando** una sequenza di numeri interi di lunghezza pari. Data la sequenza, il programma deve moltiplicare ciascun numero in una posizione di indice pari per il successivo numero in posizione di indice dispari e sommare i prodotti ottenuti.

Esempio: Se `10 2 3 4 5 6` è la sequenza letta, allora la somma calcolata deve essere `10*2 + 3*4 + 5*6 = 62`.

Il programma deve infine stampare a video il valore della somma calcolata.

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

- una funzione `Calcola(s1 []int) int` che riceve in input un valore `[]int` nel parametro `s1` e restituisce un valore di tipo `int` pari alla somma dei prodotti ottenuti moltiplicando ciascun numero in una posizione di indice pari di `s1` per il successivo numero in posizione di indice dispari.

Nota: Per creare dinamicamente una slice, si utilizzi la funzione `make`.

Esempio d'esecuzione:

```
$ go run somma_prodotto.go 1 2 3 4 5 6
La somma è 44

$ go run somma_prodotto.go 7 3 1 8
La somma è 29
```

21 Somma di prodotti pari

Scrivere un programma che:

- legga da **riga di comando** una sequenza di numeri interi;
- stampi a video il risultato della somma dei prodotti pari associati alle coppie non ordinate di numeri che si possono definire a partire dai numeri letti (data la coppia non ordinata di numeri `(numero_1, numero_2)`, il valore del prodotto associato è `numero_1 * numero_2`).

Esempio: Se `10 1 31 4` è la sequenza letta, le coppie non ordinate di numeri che si possono definire a partire dai numeri letti sono: `(10, 1)`; `(10, 31)`; `(10, 4)`; `(1, 31)`; `(1, 4)`; `(31, 4)`. Di queste, quelle il cui prodotto è pari sono: `(10, 1)`; `(10, 31)`; `(10, 4)`; `(1, 4)`; `(31, 4)`. La somma dei prodotti pari è `488`.

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

- una funzione `Calcola(s1 []int) int` che riceve in input un valore `[]int` nel parametro `s1` e restituisce un valore di tipo `int` pari alla somma dei prodotti pari associati alle coppie non ordinate di numeri che si possono definire a partire dai numeri presenti in `s1`.

Nota: Per creare dinamicamente una slice, si utilizzi la funzione `make`.

Esempio d'esecuzione:

```
$ go run prodotti_pari.go 1 2 3 4 5 6
La somma è 152

$ go run prodotti_pari.go 1 2 3 4 5
La somma è 62
```

22 Filtra voti

Scrivere un programma che:

- legga da **riga di comando** una sequenza di valori (i valori numerici interi che compaiono all'interno della

sequenza rappresentano voti in una scala di valutazione tra 0 e 100; i valori numerici interi superiori a 60 corrispondono a voti sufficienti);

- stampi a video le due sottosequenze di valori numerici interi che corrispondono rispettivamente a voti insufficienti e sufficienti.

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

- una funzione `LeggiNumeri()` (`numeri []int`) che restituisce un valore `numeri` di tipo `[]int` in cui sono memorizzati i valori numerici interi specificati a **riga di comando**;
- una funzione `FiltraVoti(voti []int)` (`sufficienti, insufficienti []int`) che riceve in input un valore `[]int` nel parametro `voti` e restituisce due valori di tipo `[]int`, `sufficienti` e `insufficienti`, in cui sono memorizzati rispettivamente i voti sufficienti e insufficienti presenti in `voti`.

Esempio d'esecuzione:

```
$ go run filtro.go 80 75 60 55
Voti sufficienti: [80 75 60]
Voti insufficienti: [55]

$ go run filtro.go 100 98 59 40
Voti sufficienti: [100 98]
Voti insufficienti: [59 40]
```

23 Numeri casuali

Scrivere un programma che: 1) Legga da **riga di comando** un numero intero `soglia`; 2) Generi in modo casuale una sequenza di lunghezza arbitraria di numeri interi compresi nell'intervallo che va da 0 a 100, estremi inclusi. Il processo di generazione si interrompe quando viene generato un numero inferiore a `soglia`. 3) Stampi a video tutti i numeri generati. 4) Stampi a video tutti i numeri generati superiori a `soglia`.

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

- una funzione `Genera(soglia int) []int` che riceve in input un valore `int` nel parametro `soglia` e restituisce un valore di tipo `[]int` in cui è memorizzata una sequenza di lunghezza arbitraria di numeri interi, generata in base alle specifiche di cui al punto 2.

Suggerimento: per generare in modo casuale un numero intero, potete utilizzare le funzioni dei package `math/rand` e `time` 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) */
numeroGenerato := rand.Intn(100)
```

Esempio d'esecuzione:

```
$ go run numeri_random.go 20
Valori generati [21 72 44 64 30 13]
Valori sopra soglia: [21 72 44 64 30]
```

24 Filtra testo

Scrivere un programma che:

- legga da **standard input** un testo su più righe (alcune delle quali possono essere delle righe vuote (`" "`));

- termini la lettura quando, premendo la combinazione di tasti `Ctrl+D`, viene inserito da **standard input** l'indicatore End-Of-File (EOF);
- ristampi le righe del testo letto in cui compaiono cifre.

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

- una funzione `LeggiTesto() []string` che legge da **standard input** un testo su più righe (alcune delle quali possono essere delle righe vuote (`" "`)) e terminato dall'indicatore EOF, restituendo un valore `[]string` in cui sono memorizzate le stringhe corrispondenti alle righe del testo letto;
- una funzione `ContieneCifre(s string) bool` che riceve in input un valore `string` nel parametro `s` e restituisce `true` se almeno un carattere in `s` è una cifra, `false` altrimenti;
- una funzione `FiltraTesto(sl []string) []string` che riceve in input un valore `[]string` nel parametro `sl` e restituisce un valore `[]string` in cui sono memorizzate le stringhe di `sl` in cui compaiono cifre; la funzione deve utilizzare la funzione `ContieneCifre()`.

Suggerimento: per sapere se un carattere è una cifra si può far riferimento alla documentazione della funzione `unicode.IsDigit` del package `unicode`.

```
$ go run filtra_testo.go
riga senza cifre
riga con 1 cifra
2a riga con 2 cifre
nessuna cifra
Testo filtrato:
riga con 1 cifra
2a riga con 2 cifre
```

25 Mazzo di carte

Scrivere un programma che:



























1. Legga da **riga di comando** un numero intero `n` tale che `0 < n < 10`.
2. Inizializzi una stringa che rappresenti un mazzo di carte formato dalle sole carte di cuori. i) Le carte di cuori corrispondono ai caratteri con codice Unicode compreso nell'intervallo che va da `'\U0001F0B1'` a `'\U0001F0BA'`, estremi inclusi. ii) Le carte del mazzo non sono mischiate: la prima carta del mazzo è l'asse di cuori; la seconda carta del mazzo è il due di cuori;... l'ultima carta del mazzo è il dieci di cuori.
3. Simuli l'estrazione casuale (ed in sequenza) di `n` carte dal mazzo, stampando a video le carte estratte e quelle rimaste nel mazzo dopo ogni estrazione.

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

- una funzione `LeggiNumero() int` che legge da **riga di comando** un numero intero e ne restituisce il valore;
- una funzione `GeneraMazzo() string` che restituisce un valore `string` che rappresenta un mazzo di carte formato dalle sole carte di cuori;
- una funzione `EstraiCarta(mazzo string) (cartaEstratta rune, mazzoResiduo string)` che riceve in input un valore `string` nel parametro `mazzo` e restituisce un valore di tipo `rune` nella variabile `cartaEstratta` ed un valore di tipo `string` nella variabile `mazzoResiduo`, che rappresentano rispettivamente la carta estratta in modo casuale dal mazzo di carte rappresentato da `mazzo` ed il mazzo residuo di carte dopo l'estrazione;
- una funzione `EstraiCarte(mazzo string, n int)` che riceve in input un valore `string` nel parametro `mazzo` ed un valore `int` nel parametro `n` e simula l'estrazione casuale (ed in sequenza) di `n` carte dal mazzo di carte rappresentato da `mazzo`, stampando a video le carte estratte e quelle rimaste nel mazzo dopo ogni estrazione; la funzione deve utilizzare la funzione `EstraiCarta()`.

Esempio d'esecuzione:

```
$ go run carte.go 6
Estratta la carta ♠ - Carte rimaste nel mazzo: ♠♠♠♠♠♠
Estratta la carta ♠ - Carte rimaste nel mazzo: ♠♠♠♠♠♠
```

Estratta la carta  - Carte rimaste nel mazzo:       
Estratta la carta  - Carte rimaste nel mazzo:      
Estratta la carta  - Carte rimaste nel mazzo:     
Estratta la carta  - Carte rimaste nel mazzo:    

26 Date (2)

Scrivere un programma che:

- legga da **standard input** un testo su più righe;
- termini la lettura quando viene inserita da standard input una riga vuota ("").

Ogni riga di testo è una stringa senza spazi che codifica una data in uno dei seguenti possibili formati:

1. aaaa/m/g
2. aaaa/m/gg
3. aaaa/mm/g
4. aaaa/mm/gg
5. g/m/aaaa
6. gg/m/aaaa
7. g/mm/aaaa
8. gg/mm/aaaa

Una volta terminata la fase di lettura, il programma deve stampare la codifica nel formato aaaa-mm-gg delle date lette. In particolare, le date devono essere stampate in ordine cronologico (dalla più antica alla più recente).

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

- una funzione `daInvertire(s string) bool` che riceve in input un valore `string` nel parametro `s` in cui è codificata una data nel formato **aaaa/m/g**, **aaaa/m/gg**, **aaaa/mm/g**, **aaaa/mm/gg**, **g/m/aaaa**, **gg/m/aaaa**, **g/mm/aaaa**, o **gg/mm/aaaa** e restituisce `true` se in `s` è codificata una data nel formato **g/m/aaaa**, **gg/m/aaaa**, **g/mm/aaaa**, o **gg/mm/aaaa**, `false` altrimenti.
- una funzione `Inverti(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`.

Suggerimento: Una volta codificate nel formato **aaaa-mm-gg**, le date possono essere ordinate cronologicamente utilizzando la funzione `sort.Strings(a []string)` del package `sort`.

Esempio d'esecuzione:

```
$ cat test
2019/04/03
2019/6/4
2019/07/5
2019/4/05
5/5/2019
05/4/2019
7/05/2019
07/09/2019
07/09/2018

$ go run date.go < test
2018-09-07
2019-04-03
2019-04-05
2019-04-05
2019-05-05
2019-05-07
2019-06-04
2019-07-05
2019-09-07
```

27 Somma unici

Scrivere un programma che legga da **riga di comando** una sequenza di valori e stampi a video la somma dei valori letti che rappresentano numeri interi e che compaiono nella sequenza una sola volta.

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

- una funzione `LeggiNumeri()` (`numeri []int`) che restituisce un valore `numeri` di tipo `[]int` in cui sono memorizzati i valori numerici interi specificati a **riga di comando**;
- una funzione `Occorrenze(numeri []int, n int) int` che riceve in input un valore `[]int` nel parametro `numeri` e restituisce un valore `int` pari al numero di occorrenze di `n` in `numeri`.

Esempio:

Supponendo di leggere da **riga di comando** la sequenza `1 2 a 4 ciao 3 2 1 5`, il programma deve stampare `12`, ovvero la somma dei numeri `4`, `3` e `5`. Se la sequenza fosse `4 3 5 non_conto 4 2 2 3 2`, l'output sarebbe invece `5`.

Esempio d'esecuzione:

```
$ go run somma_unici.go 1 2 % 4 3 2 1 5
12

$ go run somma_unici.go 4 3 5 4 2 2 3 2
5

$ go run somma_unici.go 1 2 sarà zero 1 2
0

$ go run somma_unici.go 1 2 3 2 2 2
4

$ go run somma_unici.go che 10 4 7 12 4 12 sfortuna
17
```

28 Controlla sequenza (1)

Scrivere un programma che legga da **standard input** una stringa di caratteri in cui i caratteri corrispondenti a cifre decimali rappresentano dei numeri interi minori di 10.

All'interno della stringa letta, i caratteri corrispondenti a cifre decimali sono intervallati tra loro da sottostringhe formate da caratteri arbitrari.

Il programma deve considerare la sequenza di numeri interi nascosta all'interno della stringa letta da **standard input**.

Esempio:

All'interno della stringa di caratteri

```
&4&$4%mammas!6mia6cosa1succede0
```

la sequenza di numeri interi nascosta è:

```
4 4 5 6 6 1 0
```

Il programma deve controllare che i numeri presenti all'interno della sequenza siano ordinati in senso decrescente, cioè dal più grande al più piccolo.

Nel caso in cui i numeri interi siano ordinati in senso decrescente, il programma deve stampare a video il messaggio `Sequenza nascosta ordinata.`. In caso contrario, il programma deve stampare a video il messaggio `Sequenza nascosta non ordinata.`

Si assuma che:

- all'interno della stringa di caratteri letta da **standard input** non siano presenti caratteri consecutivi corrispondenti a cifre decimali;
- all'interno della stringa di caratteri letta da **standard input** non sia presente nessun carattere di spaziatura, ossia un carattere il cui codice Unicode, passato come argomento alla funzione `func IsSpace(r rune) bool` del package `unicode`, fa restituire `true` alla funzione.

Esempio d'esecuzione:

```
$ go run controlla_sequenza.go
abc8dèf6asa2sd0
Sequenza nascosta ordinata.

$ go run controlla_sequenza.go
Èbc8d8e5a#4$d1e
Sequenza nascosta ordinata.

$ go run controlla_sequenza.go
a°c9d8e6aà7sd1e
Sequenza nascosta non ordinata.
```

29 Controlla sequenza (2)

Scrivere un programma che legga da **riga di comando** una sequenza di valori intervallati da caratteri di spaziatura.

Il primo valore che definisce la sequenza (da sinistra verso destra) è in posizione 0, il secondo in posizione 1, etc.

La sequenza è valida se:

1. Tutti i valori letti rappresentano dei numeri interi.
2. Ciascun numero che appare in una posizione *dispari* all'interno della sequenza è *minore* del numero che lo precede.
3. Fatta eccezione per il numero che appare in posizione 0, ciascun numero che appare in una posizione *pari* all'interno della sequenza è *maggiore* del numero che lo precede.

Nel caso in cui la sequenza letta sia valida, il programma deve stampare:

```
Sequenza valida.
```

In caso contrario, il programma deve stampare:

```
Valore in posizione POSIZIONE non valido.
```

dove `POSIZIONE` è la posizione del primo valore che invalida la sequenza.

Ad esempio, se la sequenza di valori letta da **riga di comando** fosse:

```
5 4 9 abc 6
```

il programma deve stampare:

```
Valore in posizione 3 non valido.
```

Si assuma che la sequenza di valori letta da **riga di comando** sia definita da almeno un valore.

Esempio d'esecuzione:

```
$ go run controlla_sequenza.go mamma mia!
Valore in posizione 0 non valido.

$ go run controlla_sequenza.go 5 4 9 2 6
```


Sequenza valida.

```
$ go run controlla_sequenza.go 5 5 9 2 6  
Valore in posizione 1 non valido.
```

```
$ go run controlla_sequenza.go 5 4 9 -2 -6  
Valore in posizione 4 non valido.
```

30 Primi

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

Scrivere un programma che legga da **riga di comando** un numero intero `numero` e stampi tutti i numeri *primi* ottenibili rimuovendo al più `3` cifre consecutive tra quelle che definiscono `numero`.

In particolare, i numeri primi devono essere stampate in ordine crescente (cioè dal più piccolo al più grande).

Ad esempio, se il numero intero letto da **riga di comando** fosse:

```
5899
```

i numeri ottenibili rimuovendo al più `3` cifre consecutive tra quelle che definiscono `5899` sarebbero:

```
5899  
899  
599  
589  
589  
99  
59  
58  
9  
5
```

Si assuma che il valore specificato a riga di comando sia nel formato corretto e, in particolare, sia un intero maggiore o uguale a 1000.

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

- una funzione `ÈPrimo(n int) bool` che riceve in input un valore (di tipo) `int` nel parametro `n` e restituisce `true` se `n` è primo (i.e., se il valore di `n` rappresenta un numero primo) e `false` altrimenti.

Suggerimento: I numeri primi possono essere ordinati in senso crescente utilizzando la funzione `sort.Ints(a []int)` del package `sort`.

Esempio d'esecuzione:

```
$ go run primi.go 5899  
5  
59  
599  
  
$ go run primi.go 5894457  
4457  
5857  
5897  
594457  
  
$ go run primi.go 10113  
13  
13  
101  
103  
113  
113
```

```
113
1013
1013

$ go run primi.go 2468
2
```

31 Divisori comuni

Definizione: I divisori propri di un numero naturale (un numero intero positivo) sono tutti i suoi divisori, tranne il numero stesso.

Definizione: Un numero naturale (un numero intero positivo) 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 **riga di comando** tre numeri interi positivi, rispettivamente `N` , `DIVISORIMIN` , e `DIVISORIMAX` ;
- stampi a video tutte le coppie di interi positivi `a` e `b` , con `a <= N` e `b <= N` , tali che:
 - i. `a` e `b` abbiano in comune un numero di divisori propri maggiore o uguale a `DIVISORIMIN` e minore o uguale a `DIVISORIMAX` ;
 - ii. almeno uno tra `a` e `b` sia un numero perfetto.

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

- una funzione `DivisoriPropri(n int) []int` che riceve in input un valore `int` nel parametro `n` e restituisce un valore `[]int` in cui sono memorizzati tutti i divisori propri di `n` .

Si assuma che:

- i valori letti da **riga di comando** siano specificati nel formato corretto;
- `N > MAX > MIN > 0` .

Esempio d'esecuzione:

```
$ go run divisori.go 50 3 4
6 6
6 12
6 18
6 24
6 30
6 36
6 42
6 48
8 28
12 28
14 28
16 28
20 28
24 28
28 32
28 36
28 40
28 42
28 44
28 48

$ go run divisori.go 20 1 2
2 6
3 6
4 6
5 6
6 7
6 8
```

```
6 9
6 10
6 11
6 13
6 14
6 15
6 16
6 17
6 19
6 20
```

```
$ go run divisor.go 20 2 3
```

```
4 6
6 6
6 8
6 9
6 10
6 12
6 14
6 15
6 16
6 18
6 20
```