

# Soluzioni degli esercizi su tipi di dato

Maggio 2022

## Esercizio 1

N.B.: il testo parla di parametri per riferimento, ma in realtà intende parametri per copia con meccanismi espliciti (i puntatori) usati per **simulare** quelli per riferimento (come in C).

Prima dell'invocazione della sottoprocedura, lo stato delle variabili locali al cliente è il seguente:

```
i: 1
x: 1, 2, 3, ...
```

I parametri sono così valutati:

- **x** punta all'inizio del vettore (proprio come la variabile **x** del cliente);
- **y** punta alla seconda cella del vettore (che al momento vale 2);
- **z** punta alla variabile **i** del cliente.

Se volessimo espandere in linea il corpo della sottoprocedura, potremmo quindi riscriverlo in questo modo:

```
x[i] = i + 1; // x: 1, 2, 3, ...
print(x[i]); // 2
x++; // x: 2, 3, ...
x[i] = i; // x: 2, 1, ...
print(x[i], i); // 1, 1
i = i + 1;
```

La stampa successiva all'invocazione fa ancora uso del puntatore **x** non incrementato. Complessivamente viene quindi stampato:

```
2
1, 1
1, 2
```

## Esercizio 2

Una possibile segnatura è `f: float -> int -> float`, e la corrispettiva assunzione è `int <: float`.

## Esercizio 3

Si consideri il linguaggio di programmazione C.

```
// Equivalenti strutturalmente ma non per nome
struct Book {
    char *name;
    int length;
};

struct Rope {
    char *name;
    int length;
}
```

int e float sono compatibili, ma non equivalenti.

## Esercizio 4

- I1: `List<A>` non è un sottotipo di `List<B>`
- I2: `List<A>` non è un sottotipo di `List<Top>`
- I3: `List<B>` non è un sottotipo di `List<A>`
- I5: non essendo A sottotipo di B, `List<A>` non è sottotipo di `List<? <: B>`

## Esercizio 5

La matrice A è memorizzata come un vettore linearizzato per righe (prima tutta la prima, poi tutta la seconda, ...). Ogni riga contiene 6 celle da 4 byte ciascuna, per un totale di 24 byte a riga. Lo scostamento della cella di indice bidimensionale (i, j) è quindi calcolabile come:

$$\text{offset}(i, j) = 4 * (6 * i + j)$$

E, in particolare:

$$\text{offset}(2, 3) = 4 * (6 * 2 + 3) = 60$$

## Esercizio 6

- `c = b`: B non è sottotipo di C;
- `c = sb.remove()`: B non è sottotipo di C;
- `sa = sb`: `Set[B]` non è sottotipo di `Set[A]`;
- `sb = sa`: `Set[A]` non è sottotipo di `Set[B]`;
- `sc = sb`: `Set[B]` non è sottotipo di `Set[C]`;
- `sa.add(sc)`: `Set[C]` non è sottotipo di A;
- `b = sb.add(b)`: `()` non è sottotipo di B;
- `c = sb.remove()`: B non è sottotipo di C.

## Esercizio 7

Poiché non diversamente specificato, assumiamo che non sia il caso che anche  $B :> A$  e consideriamo i campi come accedibili sia in lettura sia in scrittura. Anche se  $A :> B$ , quindi, non possiamo stabilire relazioni di sottotipaggio fra il tipo di  $a$  e quello di  $b$ .

- $a = b$ : il tipo di  $b$  non è sottotipo del tipo di  $a$ ;
- $b.id = a.id$ :  $A$  non è sottotipo di  $B$ ;
- $d[0] = b$ : il tipo di  $b$  non è sottotipo del tipo di  $d[0]$ ;
- $c = d$ : il tipo di  $d$  non è sottotipo del tipo di  $c$ ;
- $d = c$ : il tipo di  $c$  non è sottotipo del tipo di  $d$ ;
- $c[0] = d[0]$ : il tipo di  $d[0]$  non è sottotipo del tipo di  $c[0]$ ;
- $d[0] = c[0]$ : il tipo di  $c[0]$  non è sottotipo del tipo di  $d[0]$ ;
- $d[0] = b$ : il tipo di  $b$  non è sottotipo del tipo di  $d[0]$ .