Ownership e borrowing (Rust)

Rust gestisce la liberazione della memoria in modo automatico senza GC, e impedisce la creazione di dangling references con controlli statici.

Ownership

- ad ogni oggetto sull'heap è associato un (unico) proprietario;
- assegnamenti e passaggi di parametri (shallow copy del riferimento) trasferiscono la proprietà (*move*);
- utilizzare un oggetto di cui si è ceduta l'ownership è un errore statico;
- quando il proprietario esce dallo scope, l'oggetto viene liberato (*drop*).

Ci sono due modi per mantenere l'ownership dopo un assegnamento:

- alcuni tipi implementano il trait `Copy`, ovvero gli assegnamenti effettuano una deep copy e non trasferiscono ownership;
- i tipi che implementano `Clone` hanno a disposizione un metodo `.clone()` che crea un nuovo oggetto (deep copy) senza prendere l'ownership del'originale.

Le informazioni di ownership (e tipo) esistono solo a tempo di compilazione.

```
let x = String::from("hello");
let y = x;
println!("{}", x);
// error: borrow of moved value: ‘x‘
```

Lifetime

Parametro di tipo che descrive lo scope in cui i dati sono validi. In alcuni casi è necessario annotare il lifetime dei riferimenti restituiti da una funzione (generalmente in termini dei lifetime dei parametri).

Una chiamata esplicita a `std::mem::drop` termina in anticipo il lifetime visto che prende ownership del valore (no use after free).

Borrowing

Creazione di un riferimento esplicito che non è proprietario del valore a cui punta (operatore `&` e `&mut`).

- l'oggetto puntato non viene liberato quando il riferimento così ottenuto esce dallo scope;
- condivisione dell'accesso in sola lettura: non si possono effettuare modifiche all'oggetto finché il prestito è attivo (né attraverso il riferimento originale né con quello nuovo);
- i riferimenti sono sempre validi: il loro lifetime deve essere contenuto nel lifetime dell'oggetto da cui sono stati creati.

```
{
    let mut x = String::from("hello");

    {
        let y = &x;
        // x.push_str(" world"); -- errore
        println!("{}", y);

        // (in realtà qui potremmo modificare x perché il
        //  lifetime di y finisce alla riga in cui viene
        //  utilizzato l'ultima volta - NLL)

    } // drop y

    x.push_str(" world");

} // drop x
```