Esercitazione 3

Esercizio 1

Implementare il buffer circolare descritto a questo link di exercism

https://exercism.org/tracks/rust/exercises/circular-buffer

Attenzione: per definizione il **buffer in cui memorizzare i valori T non va riallocato e modificato**, quindi non si possono usare le funzioni avanzate di Vec che permettono di fare push e pop di valori. Infatti lo scopo del circular buffer è proprio quello di avere un buffer dove si possono leggere e scrivere valori garantendo i tempi, senza penalità di riallocazione del buffer: non è solo imporre una dimensione massima al buffer!!!
Questo in Rust presenta un “problema”: quando si legge un valore dal buffer si deve liberare la cella, e il valore viene restituito con una move perché non è detto che T implementi il trait copy. Ma se si fa una move da un elemento del vettore che rimane nell’indice corrispondente? Rust non permette di lasciare una cella senza che ci sia un valore definito di cui qualcuno abbia l’ownership…

Il problema si può risolvere in due modi:

- sostituire un valore al posto del precedente, ma quale valore? C’è un valore di default di T? Vedere se esiste un tratto che può garantire tale condizione.
- se le celle possono avere o non avere un valore di tipo T c’è una Enum apposta che permette di gestire la situazione: quale?

Risolvere il problema con entrambe le strategie

Esercizio 2

Realizzare la **`struct MyCycle<I: Clone+Iterator>`** che implementa un iteratore circolare a partire da un iteratore base generico la cui sequenza dovrà essere ripetuta n volte (n=0 ⇒ infinite volte). Quando l’iteratore base finisce, MyCycle deve essere in grado di usare un clone dell’iteratore ricevuto come parametro iniziale e ricominciare dal principio una sequenza, andando avanti così fino al numero di volte specificato all'atto della creazione.

Poiché MyCycle è un iteratore a sua volta, deve implementare il tratto Iterator.

MyCycle può essere costruito così:

```
MyCycle::new(iter: I, repeat: usize);
```

Realizzare alcuni unit test, provando anche i seguenti casi:

- fornendo in ingresso un iteratore con zero elementi si deve ottenere un iteratore che restituisce zero elementi
- costruendo un MyCycle (con numero di ripetizioni finito pari a n1) a partire da un altro MyCycle (con numero di ripetizioni finito pari a n2), si deve ottenere una sequenza che contiene n1*n2 volte la sequenza originale
- concatenando (tramite .chain(...) due MyCycle, il primo basato su una sequenza di l1 elementi con numero di ripetizioni pari a n1, il secondo basato su una sequenza di l2 elementi con numero di ripetizioni pari a n2, si deve ottenere una sequenza di l1*n1+l2*n2 elementi
- facendo lo zip di due MyCycle si ottiene una sequenza formata dalle coppie ordinate ottenute dalle rispettive sequenze