

Esercitazione 2

6/8 aprile 2022

Obiettivi Generali

Al termine dell'esercitazione, se propriamente svolta, gli studenti saranno in grado di:


- Utilizzare tipi di dati complessi come struct e enum ed implementare i relativi metodi
- Definire ed implementare tratti
- Eseguire l'overload degli operatori
- Gestire valori di ritorno di funzione potenzialmente vuoti con il tipo Optional e eseguire il binding di variabili con enum
- Organizzare il codice in moduli e sottomoduli
- Familiarizzare con il pattern Builder
- Duplicare tipi complessi
- Comporre comandi tramite la command line

Esercizio 1

Scaricare lo scheletro di progetto "Binary Search" da Exercism e scrivere il codice necessario per passare i test.

<https://exercism.org/tracks/rust/exercises/binary-search>

Provare ad implementare la funzione search sia in modo iterativo che ricorsivo senza modificare la firma della funzione.

Attenzione alla gestione di Optional: come minimizzare il numero di test quando il valore da restituire è assente? 

Bonus rispetto all'esercizio base: scrivere un main che accetti come unico argomento il numero da cercare, legga dallo standard input i valori dell'array (uno per riga) e stampi l'indice in cui si trova il numero, -1 nel caso di nessun valore trovato.

Es:

```
cargo run -- 12 < file_convalori.txt
```

Esercizio 2

Scaricare lo scheletro di progetto Clock da Exercism e scrivere il codice necessario per passare i test

<https://exercism.org/tracks/rust/exercises/clock>

Bonus: fare l'overload degli operatori + e - in modo da sommare e sottrarre i minuti ad oggetti di tipo Clock in alternativa all'uso della funzione add_minutes(...). Deve essere possibile sommare/sottrarre tra loro due valori di tipo Clock o sommare/sottrarre ad un valore di tipo Clock un intero che indica il numero di minuti.

Esercizio 3

Scaricare lo scheletro di progetto DOT DSL da Exercism e scrivere il codice necessario per passare i test

<https://exercism.org/tracks/rust/exercises/dot-dsl>

Attenzione: il problema nel testo è ampiamente sottospecificato, occorre “interpretare” i test per capire quali strutture realizzare e con che metodi. Per risolvere l’esercizio non è sufficiente realizzare la struct Graph ma occorre implementare anche le struct aggiuntive che si deducono dai test. Mantenere la struttura dei moduli indicata.

Bonus: implementare il tratto Display per Graph in modo da scrivere il grafo generato sullo standard output secondo la sintassi di Graphviz (<https://graphviz.org/doc/info/lang.html>). In questo modo sarà possibile visualizzare il grafo definito con il comando dot (disponibile installando graphviz)

```
cargo run | dot -Tpng > grafo.png
```

Nota: “|” detto “pipe” collega lo standard output di un comando con lo standard input di un altro comando; le pipe, che vedremo più avanti nel corso sono uno strumento fondamentale per la comunicazione tra processi.