Esercitazione 1

22/24 marzo 2023

Obiettivi generali

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

- Utilizzare il programma cargo, gestendo correttamente il riferimento a librerie esterne ed il processo di compilazione, e conoscere la struttura di un programma Rust
- Utilizzare correttamente i costrutti di controllo di flusso e l'istruzione match
- Comprendere il concetto possesso di un valore e distinguere, in ciascun segmento di codice, le variabili che posseggono un valore da quelle che non lo posseggono
- Manipolare stringhe, effettuare conversioni tra stringhe e array di byte e caratteri; conoscere la differenza tra byte e char nella gestione di stringhe e slice
- Comprendere la semantica del passaggio di parametri a funzioni
- Eseguire il testing delle funzioni, facendo riferimento a test pre-esistenti e sapendo scrivere test di unità
- Gestire gli argomenti passati attraverso la linea di comando

Esercizio 1

Con il termine “slug” si intende una stringa in formato leggibile in cui alcuni caratteri difficili da stampare vengono convertiti in altri facili da leggere.
Un possibile algoritmo è il seguente:

- tutti i caratteri accentati vengono convertiti nell’equivalente non accentato
- tutto viene convertito in minuscolo
- ogni carattere che non sia in [a-z][0-9] viene convertito in “-”
- due “-” consecutivi non sono ammessi, solo il primo viene tenuto
- un “-” finale non è ammesso a meno che non sia l’unico carattere nella stringa

L’obiettivo dell’esercizio è fare un funzione “slugify” che converta una stringa generica in uno slug.

Creare con cargo un nuovo package chiamato **slufigy** e, al suo interno, definire la funzione

```
fn slugify(s: &str) -> String {}
```

Per convertire le lettere accentate usare questa tabella di conversione, dove il carattere nella posizione i (come carattere) in SUBS_I corrisponde al carattere nella posizione i in SUBS_O:

```
const SUBS_I : &str =
"àáâäæãåāăąçćčđďèéêëēėęěğǵḧîïíīįìıİłḿñńňôöòóœøōõőṕŕřßśšşșťțûüùúūǔůűųẃẍ
ÿýžźż";
const SUBS_O: &str  =
"aaaaaaaaaacccddeeeeeeeeegghiiiiiiiilmnnnnoooooooooprrsssssttuuuuuuuu
uwxyyzzz";
```

Per la conversione definire una funzione che esegua la conversione con la tabella definita;

```
fn conv(c: char) -> char {}
```

Nel main leggere una sequenza di parole come argomento da command line, invocare la funzione e stampare il risultato.

Esempio:

```
cargo run -- “Questo sarà uno slug!”
```

Attenzione, quando si esegue la conversione delle stringhe accentate: si può usare il risultato di find() dentro SUBS_I come indice di SUBS_O?

Suggerimento: la lunghezza di un char è sempre uguale?

Scrivere anche i test case necessari per verificare il corretto comportamento. Coprire almeno i seguenti casi:

- stringa con più di una parola
- stringa con caratteri accentati
- stringa vuota
- stringa con più spazi e caratteri non validi
- stringa con solo caratteri non validi

Per il parsing degli argomenti di command line si suggerisce di inserire nel progetto la libreria clap

https://docs.rs/clap/latest/clap/

Esercizio 2

Se si è registrati su Exercism (https://www.execism.org) scaricare l'esercizio con il comando:

exercism download --exercise=robot-simulator --track=rust

Testo dell'esercizio a questo indirizzo:

https://exercism.org/tracks/rust/exercises/robot-simulator

Se non ci si vuole registrare su Exercism per questo esercizio e il successivo clonare il seguente repository git che contiene lo scheletro dei package da completare per la soluzione e i test da superare

https://github.com/exercism/rust.git

L'esercizio consiste nel costruire un simulatore di comandi per un Robot a partire dallo scheletro scaricato e poi lanciare i test inclusi con il comando "`cargo test`" per validare l'implementazione. (cargo test -- --include-ignored per i test segnati come "ignore")

Inoltre implementare anche la funzione main che legga da linea di comando:

- una sequenza di comandi come specificato nell'esercizio
- posizione iniziale e direzione

e stampi posizione e direzione finali

Esempio

```
cargo run -- -x 10 -y 10 -d N ALAAALAAR
```

stampa 7 9 W

Esercizio 3

Questo esercizio va svolto secondo le stesse modalità dell'esercizio 2
Testo (Minesweeper):

https://exercism.org/tracks/rust/exercises/minesweeper

Path del package nel repository:

`rust/exercises/practice/minesweeper/`

Funzione:

```
pub fn annotate(minefield: &[&str]) -> Vec<String> {
    unimplemented!("\nsolve):\n{:#?}\n", minefield);
}
```

Aggiungere un main che invochi *annotate* leggendo i parametri da command line. Es:

`cargo run -- --rows=3 --cols=3 "*  *  *  "`

(campo 3x3 con mine sulla prima colonna di sinistra)

Soluzione alternativa (da implementare dopo la prima):
Realizzare una seconda funzione che annoti il campo minato espresso internamente su una singola stringa (tutte le righe sono concatenate in una unica stringa come quando sono lette da command line) e lo faccia senza copiare mai il buffer della stringa

```
annotate2(minefield: String, rows: usize, cols: usize) -> String {}
```

Suggerimento: vedere il metodo `into_bytes()` di String
Perché non si potrebbe usare `as_bytes()`?