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; 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

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

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

La funzione deve convertire in maiuscolo il primo carattere di ogni parola che compone il testo `s`, ignorando eventuali altri caratteri maiuscoli al suo interno.
Le parole sono separate da uno o più spazi.

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

Esempio:

```
cargo run -- “questa è una   frase”
```

È possibile ipotizzare di trasformare in maiuscolo i caratteri *in place*, senza copiare la stringa? Motivare la risposta.
Suggerimento: in alcune lingue non c’è una corrispondenza 1:1 come numero di codepoint tra maiuscole e minuscole. Ad esempio in tedesco la maiuscola di ß può essere sia SS che ß.

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

- stringa con più di una parola
- stringa con una sola parola senza spazi
- stringa con caratteri accentati all’inizio di parola
- stringa vuota
- stringa con più spazi

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=luhn --track=rust
```

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
```

Il testo dell'esercizio è a questo indirizzo e spiega come validare un numero di carta di credito secondo l'algoritmo di Luhn

```
https://exercism.org/tracks/rust/exercises/luhn
```

Il package scheletro si trova nella sottocartella

```
rust/exercises/practice/luhn/
```

Occorre implementare la seguente funzione nel file lib.rs

```
pub fn is_valid(code: &str) -> bool {
    unimplemented!("Is the Luhn checksum for {} valid?", code);
}
```

e poi lanciare i test con `cargo test` per validare l'implementazione

Inoltre implementare anche la funzione main che legga il numero di carta di credito come argomento da command line, sempre utilizzando clap.
Prima di invocare `is_valid` occorre validare che il numero letto sia specificato in modo corretto, ovvero 4 gruppi di 4 cifre.

Esempio

```
cargo run --  "1234 1234 1234 1234"
```

stampa "valid" o "not valid" in base all'algoritmo;

invece

```
cargo run --  "1234 1234 1234 abcd"
```

stampa "invalid format".

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 "*  *  *  "` ** - 2 spazi vuoti - * 2 spazi vuoti - * 2 spazi vuoti*

(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()`?