Exercises

Introduction

- Get started following the guide here: https://www.rust-lang.org/
- Follow tutorial here: https://doc.rust-lang.org/cargo/

Exercise 1

Write the body of the following `invert` function:

```rust
fn invert(s: &String) -> String {
}
```

`invert` shall return the inverted string

Write a main function with tests

Exercise 1b

Write the same function but inverting the string in place

```rust
fn invert(s: &mut String) {
}
```

Write associated tests.

Exercise 2

Write a function that converts an `i32` into a `String`, without using built-in facilities. Start by handling positive integers, then transition to all integers.

Write associated tests.

Exercise 3

Write a function that converts a `String` into an `i32`, with associated tests.

BONUS Exercise 4

Write a function that converts an `i32` into a `String` using Roman numerals.

Write the inverse function.

Exercise 6

Write a stack type, that implements a stack of integers with a fixed max size (using an array), with associated `pop`, `push`, and `peek` operations.

Exercise 7

Make the stack in exercise 6 be of unbounded size.

Exercise 7b [bonus]

Use traits and trait objects to hide the struct's contents.

Exercise 8

Implement an eval method for the `Expr` struct

```
#[derive(Debug)]
enum Operator {
    Plus, Minus, Divide, Multiply
}

#[derive(Debug)]
enum Expr {
    BinOp { l: Box<Expr>, op: Operator, r: Box<Expr> },
    IfExpr { cond: Box<Expr>, true_branch: Box<Expr>, false_branch: Box<Expr> },
    Literal(i32)
}
```