Exercises

Introduction

- Get Alire here: https://ada-lang.io/ or here https://alire.ada.dev/
- Check the "Creating a new crate" section here https://alire.ada.dev/docs/ for how to create a new project
- If you want to use an IDE with that, checkout VS Code with the Ada extension by AdaCore
- Then run `alr config --set editor.cmd "code ."` and `alr edit`, which will open VS Code on your project, fully setup.

Exercise 1 [arrays]

Write the body of the following `Invert` function:

```
function Invert (S : String) return String
   ...
begin
   ...
end Invert;
```

`Invert` shall return the inverted string, with the same bounds as `S`.

You can put this function inside a main procedure that will serve as a test procedure:

```
--  test_invert.adb
with Ada.Text_IO; use Ada.Text_IO;

procedure Test_Invert is
   function Invert (S : String) return String
      ...
   begin
      ...
   end Invert;
begin
  ...  --  Put some tests here
end Test_Invert;
```

Exercise 1b [arrays]

Write the same subprogram, but as a procedure, that inverts the string in-place:

```
procedure Invert (S : in out String)
   ...
begin
```

```
   ...
end Invert;
```

Write associated tests.

Exercise 2 [arrays]

Write a function that converts an `Integer` into a `String`, without using the `'Image` attribute. Start by handling positive integers, then transition to all integers.

Write associated tests.

Exercise 3 [arrays]

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

BONUS Exercise 4 [arrays]

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

Write the inverse function.

Exercise 5 [aggregates]

Write a program that shows a rectangle triangle with `*` of size `N*N`, using array aggregates. For example, for `N=4`:

```
   *
  **
 ***
****
```

Exercise 6 [packages]

Write a package (spec and body) that implements a singleton `Integer` stack (no type). The stack has a fixed maximum size of `512`.

The public subprograms are:

```
procedure Pop (V : out Integer);
procedure Push (V : Integer);
```

Exercise 7 [packages, types, privacy]

Add a `Stack` private type to the previous package, to allow having several distinct stacks in the same program.

Exercise 8 [privacy, heap allocation]

Modify the implementation of the previous package to use a linked list instead of an array.

OR

Modify the implementation of the previous package so that the programmer can specify the size of the array when declaring the stack.

OR

Modify the implementation of the previous package so that it uses an array that is automatically resized when the programmer goes beyond the initial max size.

Exercise 9 [generics]

Make the previous package generic on the type of elements.