Exploring language-based security with Rust

Maarten Everts, Jaco van de Pol, Freark van der Berg

Rules of the game:

1. Please work in the same group of 2 persons as the first assignment.
2. The work should be submitted on Blackboard under "Assignments".
3. The submission consists of 2 files: a "zip"[1] (all source code) and a "pdf" (report).[2]
4. Part A is an exercise (optional). The "core" parts (B,C) are obligatory.
5. Clearly indicate who participated in which part of the assignment.
6. The "pdf" should contain:
   - Your names and s-numbers, and roles in the various parts
   - A short description of your approach (max 1/2 A4 per part)
   - A short description of the results (max 1/2 A4 per part)
   - A clear reproducability statement (tools, versions, parameters, etc.)
7. The deadline for this second assignment is: **December 20, 23:59 CET**.
8. Submissions by e-mail instead of Blackboard, after the final deadline, or in formats different from zip+pdf will be ignored.

A: Guessing game (exercise: optional)

To get acquainted with the Rust language, the compiler and other supporting tools, we advice the following:

- Install the latest Rust toolchain using `rustup`. See `https://www.rust-lang.org/en-US/install.html` for more information.
- Configure an editor or IDE for Rust support. See `https://forge.rust-lang.org/ides.html` for an overview of plugins for IDEs and other (modern) editors.
- Follow the "guessing game" tutorial of the second edition of the Rust book: `https://doc.rust-lang.org/book/second-edition/ch02-00-guessing-game-tutorial.html`

[1] Just zip. No .rar, .7z, .tgz.
[2] So, *do not* put the pdf inside the zip file.

B: Implement a text-file analyzer (obligatory)

The goal of this part is to implement a simple stand-alone program that will allow you to work with the (zero-cost) abstractions provided by Rust and its standard library. Your task is to write a program that, given a filename of a text file, produces statistics (see below) for this text file.

Some guidelines and additional instructions:

- You can use `cargo` to create a template to start from:

  ```
  cargo new textstat --bin
  ```

- Allow the text file to read to be specified on the command line, that is:

  ```
  textstat file.txt
  ```

- Ignore case. "The", and "the" should be considered the same word.
- Ignore punctuation marks while looking for words, however do consider contractions (e.g., "I'm") to be one word. The text "It's a beautiful day." has the words "it's", "a", "beautiful", and "day".
- The statistics should include at least:
  - The total number of words.
  - The average word size.
  - A list of the number of words per word size (up to at least 10 characters)
  - A (sorted) list of the 10 most used words.
- Try to use the collections data structures of the Rust standard library[3] where possible (e.g., vectors, maps).
- Also consider using iterators and consumers with the collections data structures as they allow for very concise code.[4]

Additional resources to get you started:

- Rust by example, chapter on file I/O: `http://rustbyexample.com/std_misc/file.html`.
- The documentation for `std::fs::File`: `https://doc.rust-lang.org/std/fs/struct.File.html`
- The documentation for `std::io::BufReader`: `https://doc.rust-lang.org/1.8.0/std/io/struct.BufReader.html`
- The documentation on `std::string::String`: `https://doc.rust-lang.org/std/string/struct.String.html`.

[3] See `https://doc.rust-lang.org/std/collections/`.
[4] See `https://doc.rust-lang.org/book/first-edition/iterators.html`.

C: Reimplement the binary search tree in Rust (obligatory)

In the previous assignment we asked you to complete and secure a simple binary search tree program written in the C programming language. In this assignment you will reimplement this same program in the Rust programming language.

To get you started, we provide you with a skeleton program (see zip file) that includes a simple implementation of the command interpretation (that incidentally also illustrates the `enum` data type and the `match` language construction[5]). Some guidelines and additional instructions:

- Make sure this Rust implementation behaves exactly the same as the C program as we will also apply our automated test suit to the Rust version. One exception is that you no not have to implement the test (`t`) command (see below).
- Like in the C version, create a data structure (type) that implements the binary search tree. Call it `SortedContainer` and implement at least the following methods[6] for this data type:
  - `print`
  - `insert`
  - `contains`

  Implementing recursive data-structures (such as a binary search tree) is where the ownership concept gets tricky, see also the note below. Furthermore, you might find yourself working with the built-in `Option`[7] and `Box`[8] data types.
- (BONUS) Also implement the `erase` method. This is a bit harder than the other methods, but doable. For hints on how to battle with the borrow checker, check out this book on implementing linked lists: `http://cglab.ca/~abeinges/blah/too-many-lists/book/`
- Consider implementing the `Display` trait[9] for the data structures you create. Not only does this make implementing the `print` method much easier, it can also make debugging more convenient. Alternatively, for easier debugging you can also use the `#[derive(Debug)]` instruction.[10]
- Have look at what other functionality the `#derive` attribute[11] can provide and see how it can make things easier for you.

[5] See `https://doc.rust-lang.org/book/second-edition/ch06-00-enums.html`.
[6] See `https://doc.rust-lang.org/book/second-edition/ch05-03-method-syntax.html`.
[7] See `https://doc.rust-lang.org/std/option/`.
[8] See `https://doc.rust-lang.org/std/boxed/`.
[9] See `https://doc.rust-lang.org/std/fmt/trait.Display.html` and `https://rustbyexample.com/hello/print/print_display.html`.
[10] See `https://rustbyexample.com/trait/derive.html` for more information.
[11] See `https://rustbyexample.com/trait/derive.html`.

- Decide for yourself whether you want to implement `SortedContainer` in a separate module[12] or simply within the `main.rs` file.
- Instead of a command (`t`), please implement the tests in Rust's built-in test-framework, such that `cargo test` runs all the tests. See `https://doc.rust-lang.org/book/second-edition/ch11-01-writing-tests.html` for more information. Of course, feel free to write additional tests.
- As a final step, run `valgrind` on the the compiled binary to check whether there are any memory leaks or other memory-problems.

Do note that the Rust compiler requires you to be very precise when it comes to mutability and ownership. Try to follow the compiler's advice and see where it gets you. However you still might find yourself "fighting" the compiler. Consider the following resources when stuck:

- Rust documentation on references and borrowing:
  - First edition:
    `https://doc.rust-lang.org/book/references-and-borrowing.html`
  - Second edition:
    `https://doc.rust-lang.org/book/second-edition/ch04-00-understanding-ownership.html`
- A book on linked lists:
  `http://cglab.ca/~abeinges/blah/too-many-lists/book/`
- A discussion on iterating through a recursive data structure in rust:
  `https://www.reddit.com/r/rust/comments/464jge/lifetime_issues_turning_tail_recursion_into_a_loop/`

[12] See `https://doc.rust-lang.org/book/second-edition/ch07-00-modules.html`.