Introducing the Rust Programming Language

Advanced Systems Programming (H) 2022-2023 – Laboratory Exercise 1
Dr Anna Lito Michala & Dr Colin Perkins, School of Computing Science, University of Glasgow

1 Introduction

The Advanced Systems Programming (H) course uses the Rust programming language (`https://rust-lang.org/`) to illustrate several topics in systems programming. You are expected to learn the basics of programming in Rust in a self-directed manner as part of this course. To assist this learning, this exercise introduces the Rust programming language and its tool set. **This is a formative exercise and is not assessed.**

The expected timeline for this laboratory exercise is that you complete the preliminaries, in Section 2, and ensure that you have access to a working version of the Rust compiler and Cargo build tool, by the middle of the timetabled lab session on Monday 9 January 2023. The remainder of the material, in Section 3, should be completed during the second half of the lab and requires teamwork.

2 Preliminaries

Ensure you have access to a recent version of the `rustc` compiler and the `cargo` package manager and build tool. These are pre-installed on the student Linux servers provided by the School (`stlinux01.dcs.gla.ac.uk` to `stlinux08.dcs.gla.ac.uk`), or you can download binaries from `https://rust-lang.org/` to install on your own system. When correctly installed, you should be able to run `rustc` and `cargo` and get output like the following, depending on the exact version you have installed (> is the command prompt):

```
>rustc --version
rustc 1.57.0 (f1edd0429 2021-11-29)
>cargo --version
cargo 1.57.0 (b2e52d7ca 2021-10-21)
>
```

These notes assume the Rust 2018 Edition (`rustc 1.31.0` or later) is available. Examples have been tested with `rustc 1.57.0`, but should work with any recent version of Rust and Cargo.

You create, compile, and run Rust application using `cargo`. The `cargo` tool is the Rust package manager and build tool. It downloads any necessary packages, invokes the compiler (`rustc`), and executes the resulting binary:

```
>cargo new --bin hello
 Created binary (application) `hello` package
>cd hello/
>cat src/main.rs
fn main() {
  println!("Hello, world!");
}
>cargo run
```

```
   Compiling hello v0.1.0 (/users/staff/csp/hello)
    Finished dev [unoptimized + debuginfo] target(s) in 5.38s
     Running `target/debug/hello`
Hello, world!
>
```

The resulting output file (in this case, `target/debug/hello`) is a stand-alone native executable file that can be directly run from the command prompt. The `cargo` tool produces debug builds by default, use `cargo run --release` to produce optimised release builds (smaller and *much* faster, but without debug symbols). The `cargo run` command compiles and runs an executable; the `cargo build` command can be used to compile, but not run, code.

Ensure you can compile and run the "Hello, world" application.

Inspect the `src/main.rs` and `Cargo.toml` files produced by `cargo` and make sure you understand their contents.

Read Chapter 1 of the online Rust book (`https://doc.rust-lang.org/book/ch01-00-getting-started.html`) for more details.

3 Formative Exercises

To demonstrate your understanding of the basics of programming in Rust, complete the following exercises. These exercises are not assessed, and you do not need to submit your solutions. For some of the labs these will be programming tasks. However, for other labs (like today) they will be reading and discussion tasks.

1. Organise yourselves into 8 groups (5 mins).
2. The lecturer will allocate each group one of the following languages to quickly review as a team: Idris, Oberon, OCaml, Objective C, Swift, Go, Ada, and Eiffel.
3. Use google or the library's resources to identify the main features of the language allocated (pros) as well as any well known issues and constraints (cons). Propose the best use case for your language (10 mins - individual).
4. Discuss your findings and pick 1 main benefit and 1 main drawback (10 mins - group).
5. Each group should populate their proposals on the lab whiteboard (5 mins - pick a representative from the group to do this).
6. The lab will end with an open discussion about the variety of language features and their ideal uses (5 mins).

Discuss your findings with the lecturer or lab demonstrators to make sure you are working towards the task and allow everyone in the group to speak and contribute. You can ask questions about the first part, setting up the environment. We will not use the environment you set up today, but we will be using it throughout the semester to complete your labs and Summative Assessments.

4 Next Steps

At this point you should have a very basic introduction of programming in Rust. We will dive into the specifics in Lecture 3 at which point we will start working on coding examples. **However, you are expected to learn the basics of programming in Rust in a self-directed manner**. The laboratory exercises and lectures will direct you to appropriate resources, but on onus is on you to study the material.

Lecture 3 will provide a basic understanding of the language. The later parts of the course will then consider the use of the Rust type system to help model the problem domain and help ensure correctness of the solution (Lecture 4), consider how it handles memory allocation and data ownership (Lectures 5 and 6), how it addresses the challenges of concurrency (Lecture 7), and how it supports asynchronous programming (Lecture 8). To prepare for this material, you should begin to review the remaining material in the online Rust book, and practice programming in Rust to ensure you start to have a good understanding of the language.