Rust: A simple guide

Benjamin Lannon, James Bruska, David Josephs, Jacob Meite

November 30, 2015

Contents

1 Installation

To install Rust, you can find binaries for Linux, Mac OSX, or Windows at https://www.rust-lang.org/downloads.html or download the source from Github at https://github.com/rust-lang/rust. If you are on OSX, you can install Rust through Homebrew or if you are using Arch Linux, it is available in the community repositories of Pacman. This guide was made with the current version of the Rust compiler (rustc 1.4.0) and the current version of the Rust project manager (Cargo 0.6.0)

2 Hello World in Rust

Now that Rust is installed let's get working on a simple program. The easiest program to write is one which prints "Hello World" to the screen. The code will be saved as main.rs and is seen below.

```
fn main() {
        println!("Hello, world!");
}
```

to compile a Rust file, one can type *rustc main.rs* and an executable main (or main.exe on windows) will be generated.

3 Paradigms

Rust has the following paradigms, Imperative, Object-oriented, Functional, Procedural, Generic, Reflective, Concurrent.

4 Data Types

There are 10 types within Rust: primitive, textual, tuples vector, structure, enumerated, recursive, pointer, function, and object. There are also two things that support the typing. There are the type parameters and the self type.

The primitive types are made up of four main sub-types. There is the unit type (), the boolean type, the machine type, and the machine-dependent type. The unit type has a single unit value (). This can also be called nil. The boolean types evaluate to true and false. The machine types are split into three types. There are unsigned word types (u8, u16, u32, and u64), signed two's compliment word types (i8, i16, i32, and i64), and IEEE 754-2008 binary32 and binary64 floating-point types (f32 and f64). Finally the machine-dependent types are borken into integer (uint and int) and floating point (float [f32 or f64]) types. These are versions of the primitives that are machine specific.

The other types are consistant with the normal usage of the types.

The type parameters are similar to templates. They allow the use of a parameter without knowing what type will be used in it. The self types are a reference to the implementing item.

In Rust, data types are sectioned into kinds. The kinds are based on the properties of the components of the type. The types are freeze, send, 'static, drop, and default. Freeze make the item contain no mutable memory location. Send types (scalars, owning pointers, owned closures, and structural types) can be safely sent between tasks. All send types are also 'static. The 'static types do not have any extra pointers and makes sure that no unsafe opperations take place. The drop trait adds a destructor method called drop. This works with a top-down order. Only send types can also have a drop parameter. The default are types with destructors, closure environments, and other non-first-class types. These are not copyable and can only be accessed with pointers.

5 Immutability by Default

Rust has a primary focus, safety. In order to create a more secure environment Rust makes bindings immutable by default. In other word the item cannot change any of its values. The keyword mut allows the item to be mutable. This allow the compiler to catch an item that was changed when the programmer did not intend for the item to change. The programmer can then call an item mutable if it needs to be. This allows for a safer programming environment.

6 Rust Project Workflow: Cargo

Rust programs can be compiled by writing .rs files and using rustc directly, but the preferred workflow is to use Cargo. Cargo allows a developer to download a project off of something such as Github and immediately be able to start using the program.

Cargo has a fairly simple setup. The main file in every project is Cargo.toml which is a configuration file for a project. It is where one puts metadata such as authors, descriptions of the package, version number. It also includes the dependencies needed which will be downloaded from https://crates.io

An example Cargo.toml file is seen below:

```
[package]

name="Harry's Random Walk: Rust Edition"
version="0.1.0"
authors = [
        "Benjamin Lannon <lannonbr@clarkson.edu>",
```

```
        "Jacob Melite",
        "James Bruska",
        "David Joesephs"
]

[[bin]]
name="harrys-random-walk"

[dependencies]
piston = "0.16.0"
piston2d-graphics = "0.11.0"
piston_window = "0.30.0"
piston2d-opengl_graphics = "0.19.0"
image = "0.5.1"
vecmath = "0.2.0"
rand = "0.3.12"
```

Following, this, your source files are stored in the src/ directory and then all that needs to be done to run a Cargo project is either *cargo build* to build the executable, or *cargo run* to build and run the executable. Other major tools cargo offers include *cargo test* to run through tests in your code, and *cargo doc* to generate documentation.

7 Example: "Harry's Random Walk" Overview

Harry's Random Walk is good example to test out many features of the Rust programming language, including GUIs, random number generation, and the overall workflow of building Rust applications.

Attached in the HarrysRandomWalk/ directory is all of the required files to run the program. The main package used to create HRW is Piston [website: http://www.piston.rs/], a game engine that uses OpenGL as a graphical backend. Instead of downloading the engine manually and linking it together with my program, the only thing needed to do was include the packages needed in the Cargo.toml file and it would download the needed dependencies and nothing more.

8 Advantages to other languages

A key feature of Rust is how it handles memory allocation. In Rust, memory is freed automatically when it is no longer needed. However, a garbage collector is not used to do this. Instead, the compiler automatically determines when the program should deallocate memory, and inserts calls to do so itself. The compiler does this by allocating every variable on the stack by default, and deallocating memory when the variable goes out of scope. Any variable that needs to be allocated on the heap must be created via certain types, the simplest one being a Box. Example code is shown below:

```
fn main() {
        let x = Box::new(5);
}
```

9 Disadvantages to other languages

10 References

The Rust Programming Language Documentation: https://doc.rust-lang.org/stable/