Rust Evaluation

Riley Fiske – December 4th, 2023

***Reason for Choosing Rust***

When selecting a language for the last assignment, I wanted to select a language I had heard about a fair amount, would be fun to learn, and could potentially help me in the future as a programmer. That begs the question then, as a statistics guy, why Rust? This summer, I was having dinner with a fellow intern, who I would later call one of my best friends from Germany, while I was in Munich, and since it was still early in the summer, I was still getting to know him. I knew that he was an electrical engineering intern working at Hochschule München, but that was about it. I asked him about his work and later on in the conversation, after we talked about my work in programming, he discussed a bit of programming that he had done. This involved using Rust on a Raspberry Pi to interface with something in his house. He asked if I had ever used Rust before and I hadn’t; I had heard of it and seen some ways people used it on the internet, but my understanding of it was next to none.

With this in mind, when I was given the choice of any programming language to do an overview of, Rust was an early contender in my mind and after seeing what it was, how to set it up, and what it could do, I decided to give it a go. For my project, I developed a simple text adventure game that utilizes Rust’s version of objects, known as structs, traits, and implementations, as well as creating functions, manipulating text, building and extracting from arrays, and many more things.

***Rust Features***

Rust was a very fun language to interact with for my last assignment as it is reminiscent of languages I learned earlier in my programming career while having interesting twists along the way that in many ways reminds me of many elements of languages we worked on throughout the semester. Firstly, like Java, Rust programs run from the main method, however the syntax for this is much less clunky than Java’s, simply being “fun main() {}”. When declaring variables, you use the keyword “let”, similarly to how in mathematical proofs, you would say “let” before declaring a variable to use later (example, let x be an element of the positive integers).

Rust also allows the programmer to specify if the variable should be able to change value or not using mutability. Java has something similar with finality of variables, where you declare that it shall never change, but this is slightly more common in Rust since variables get borrowed, moved, or copied quite frequently in functions. This was an issue I frequently ran into as I was trying to pass my objects into functions and I needed them to be mutable, but that then introduces copies of variables needing to be made by the compiler that result in errors when the same variable gets back referenced. It felt a little like Whack-A-Mole, trying to figure out what should be mutable and what shouldn’t be, but in the end, I got my structs to work in the way I intended.

In the same vein of variables, when creating instances of structs that have attributes, the programmer must specify the name and value for each of the attributes, unlike other languages where the programmer must know the order and the name of each attribute is not required. I think this makes the struct declarations more readable and easier to tweak.

Overall, Rust was a blast to learn, and I can definitely see myself using it in the future for projects. It’s likeness to Java made it simpler to pick up than if I had no prior experience, and the exposure to other formatting styles for just about anything when it comes to programming languages has opened my eyes and made it easier to adapt to small syntactical differences on the fly and not be stuck in a root, only knowing one way to program.