COMP6991: Rust Programming

Mission

- To teach students to explore new perspectives in programming, particularly with respect to program safety, efficiency, and productivity.

Goals

- Develop a strong understanding of, and proficiency in, the Rust programming language.
- Understand Rust's primary goals (safety, efficiency, productivity) and their motives, considerations, and implementations.
- Manage Rust projects using the Rustup toolchain and Cargo package manager.
- Understand Rust's type system on both a fundamental and practical level.
- Learn to write powerful unit tests and documentation tests for all scales of Rust programs.
- Develop an appreciation in reasoning about software through static guarantees.
- Apply improved program design and reasoning knowledge learnt through Rust to other programming languages.
- Create well-considered, high performance, low cost abstractions to solve problems.

Principles

- Abstractions must be safe, powerful, and ergonomic.
- Libraries should provide a simple interface, yet maintain extensibility.
- Unsafe operations must be abstracted into safe interfaces.
- Code must be written for security, maintainability, and longevity.
- Software should be reliable and efficient, yet productive to develop.
- Software should never unexpectedly crash.

Assessment Structure

- Assignment 1: 20%
- Assignment 2: 25%
- Weekly Exercises: 20% (2.5% * 8 weeks)
- Exam: 35%

Plan

Week	Lectures	Tutorial / Weekly Exercises (20%)		Assignments
Wk0 *Course Intro*	• Introduction to the course • Goals and aims for the term • Expectations for students			
Wk1	• Course outline + introduction to Rust • Creating Rust projects using Rustup / Cargo • Variables, mutability, data types, functions • If, while, for, everything as an expression	• Tutorial: Course overview, introductions • Setting up Rust / Cargo / Rustup environments • Basic programming exercises to test environment / setup		
Wk2	• Common types (Option, Result) • Collections (vector, string, hashmap, iterator) • Structs (product types) and enums (sum types) • Pattern matching • Try trait / operator	• Tutorial: Reify previous week's lecture content • General programming exercises • Introduce new core types and concepts from previous week's lectures into weekly exercises • Ideally, most exercises can be automarked		
Wk3	• Ownership + move semantics • Borrowing (References) • Lifetimes • Smart pointers (Box, Rc/Arc, Cell/Refcell)			**Assignment 1 (20%)** • Stdin + Stdout application • Must be well formatted (rustfmt) • Must be idiomatic Rust (cargo clippy) • Must have basic testing • **Released Monday Week 3** • **Due Friday Week 6**
Wk4	• Modularising projects • Documenting code • Unit testing • Documentation testing			
Wk5	• Generics (and monomorphization) • Traits • Static vs dynamic dispatch	• Unit testing • Documentation + documentation testing • Manually marked with emailed feedback from tutors		
Wk6 *Flex Week*				
Wk7	• Closures • Function types (fn, FnOnce, FnMut, Fn) • Macros	• Programming with traits and challenging generics • Using dynamically dispatched types • Automarked		**Assignment 2 (25%)** • Multithreaded programming • Must be well formatted (rustfmt) • Must be idiomatic Rust (cargo clippy) • Must be thoroughly tested (unit-tests **and** doc-tests) • **Released Monday Week 7** • **Due Friday Week 10**
Wk8	• Fearless concurrency! • Threads • Sync primitives (Mutex, RwLock, mpsc, etc.) • Send / Sync traits	• Closures • Function traits • Writing macro_rules macros • Automarked		
Wk9	• Unsafe Rust • Safe abstractions over unsafe operations • Unsafe tooling (MIRI, sanitizers) • Basic FFI	• Writing concurrent code • Fixing broken concurrent code • Automarked		
Wk10	• Course wrap up • Current state and future of Rust • Exam details	• Tutorial: Unsafe Rust • Course wrap up	• No lab exercises in week 10 • Practice exam made available to students	

Course Details

Course Code	COMP6991
Course Title	Rust Programming
Convenor/Admin	*TBA*
Lecturer(s)	*TBA*
Contact for the course	cs6991@cse.unsw.edu.au
Classes	(2 x Lecture, 1 x Tutorial) / week
Consultations	Course forum, help sessions, course email
Units of Credit	6
Course Website	*TBA*
Handbook Entry	*TBA*

Prerequisites

- COMP2521
- As a 6-level course, it is suggested that students enroll in at least their 3rd year (or 2nd year as a very keen learner)

Projected Enrollments

- It is projected that approximately 50 students will enroll in the first couple of offerings
- It is further projected that this will raise to approximately 100-150 students in the long run

FAQ

- **What makes Rust a programming language worth teaching?**
  Rust is, in many ways, *not* a new programming language. Almost every defining feature of Rust has been heavily inspired by the historical success of other programming languages, each often from vastly different backgrounds. It is one of very few programming languages to sit in a fascinating intersection between academia and industry, being widely cited as a very *well-considered* language that avoids repeating the same mistakes of most mainstream languages.
  Importantly, Rust aggregates *many* powerful programming concepts into a single cohesive package that promotes excellence in software design, implementation, and reasoning.

  Although Rust draws strong influence from many "academic" programming languages (eg. Haskell, ML, etc.), its keen focus on industry has steered the language to remain approachable, while maintaining a general sense of familiarity to mainstream industry languages. As such, when compared to academic programming languages that promote similar benefits, Rust presents a unique opportunity to students – the chance to apply this knowledge in a real, high performance, industry relevant programming language. Students find this much more useful when trying to apply their improved software skills in other programming languages, which are much more likely to be mainstream than academic.

  Rust's increasingly growing industry backing and adoption will also ensure the longevity of the language, and will likely allow students to put their education into *direct* practice in the near future as programming in Rust continues to become more popular in industry.

- **Why is this course being titled as a Rust programming course, rather than naming a concept?**
  Although this course is simply titled "Rust programming", its volume of theoretical content should not be underestimated. This course heavily scrutinizes both software design and implementation, the fundamental concepts behind them, and their motivations.

  Proficiency in the Rust programming language is certainly *a* learning outcome of this course, but more important is the fundamentals that come along with that education. Fortunately, the vast majority of the theoretical knowledge in this course can be *assessed* through practical activities, so most demonstrable work will be in the form of Rust programs. With this all considered, "Rust programming" seems to be the least confusing / vague course title for students, recalling that the course's title does not represent the ceiling of course content.

- **Why is *now* (upd. June 2021) a good time to start offering this course?**
  Rust is now in a critical period of maturity, becoming increasingly popular month-on-month, with industry adoption growing faster than ever. Since its official 1.0 release just over 6 years ago (May 2015), it has been voted the "Most loved programming language" every single year in Stack Overflow's annual developer survey, and is projected to remain first place this year continuing. Earlier this year, the formation of the Rust Foundation was officially announced, backed by industry leaders Amazon AWS, Facebook, Google, Huawei, Microsoft, and Mozilla – most of which have even been forming programs solely to drive Rust's adoption internally at each respective company. Rust is projected to be the first major programming language to be introduced into the Linux kernel since C (excl. assembly), and recently Google has officially announced support for Rust in the Android source code as an alternative to C/C++.
  This sudden influx of adoption is for good reason, and students are certainly going to become increasingly "Rust-curious" as time passes. Now is an excellent time to set a solid foundation, and begin iterating on the education of Rust, similarly to multiple other leading tertiary schools (eg. Stanford, UPenn, Tsinghua Uni, etc.).

- **Why weekly programming exercises?**
  Weekly programming exercises allow students to slowly trickle new concepts into practice, acutely understanding their fundamental operations in a small, limited context. This will help alleviate students becoming locked into previous mental models when approaching the larger assignment projects. Consequently, students will already feel familiar with Rust's tools and various "Rust-isms" prior to attempting to integrate them into their assignments – overall resulting in a higher quality of education.

- **Why 2 assignments instead of 3?**
  Rust's type features, abstractions, and safety guarantees can sometimes be difficult to apply or reason about in smaller software. As project size increases, the requirement for effective abstraction, and the scope for potential problems, safety concerns, and correctness issues arising, also increases (potentially at an even faster rate). This will help challenge students to apply and demonstrate their newfound knowledge more effectively. Although this does result in a lesser exposure to unique projects, this decision is in the best interests of the mission and goals of the course. Furthermore, it is quite difficult to empathetically schedule 3 large programming assignments into a 10 week term, while being mindful of flexibility week – multiple 3-assignment courses (eg. COMP6080, COMP6771) tend to lean on the first assignment as somewhat of a "getting started" project, which this course plans to achieve through the weekly exercises instead.

- **Why a relatively low-weighting final exam (35%)?**
  Most of the goals of this course revolve around designing, orchestrating and implementing medium-large sized Rust projects, in order to fully appreciate the safety, efficiency, and productivity that the Rust programming language offers. While this course does involve strong components of theory (especially in software design), this is quite effectively assessed through the practical design components of the course. The final exam is envisioned to be an approximate 30/70 split of theory/practical exercises. The theory component will be utilised to assess any theory that wasn't possible to assess practically, and the practical component will challenge students' productivity improvements they have developed throughout the term in a time-restricted context.

- **Why are there weekly exercises, but no lab sessions?**
  With a large portion of teaching now residing online, lab sessions have begun to endure a minor identity crisis. In essence, online lab sessions have turned into glorified TLB-enrollment-restricted help sessions. This is undoubtedly less efficient than simply taking the budgeted lab session hours, and rescheduling them into completely open consultation hours (i.e. help sessions), promoting asynchronous education. It is also considered that supplementing additional tutorial time (for example, to a 2 hour session), may be of greater interest to the student body. This would help account for the much less efficient communication online, and face-to-face classes could possibly translate this additional hour into a small practical session. This would strike a convenient balance between the strengths of both online, and face-to-face teaching methodologies.

- **Why not introduce COMP3141 as a prerequisite to this course?**
  COMP3141 shares some similar material to this course, especially regarding software correctness and design. Although this would help students progress more easily through early course materials, there likely exists a large subset of students who are interested in learning effective Rust programming, but aren't inclined to take COMP3141. This prerequisite can definitely be reconsidered if COMP3141 is introduced into the Computer Science core program in future.

- **Why does the week 4 content seem unrelated to adjacent weeks?**
  The first three weeks of content are incredibly necessary to attain a fundamental ability to create Rust programs. The first two weeks teach Rust's most primitive tools and essential standard library types, and the third week develops an important mindset regarding memory management and rules entirely unique to Rust. However,

since the first large programming assignment is released during week 3, understanding how to structure, maintain, and test large Rust projects is incredibly relevant for students, so it is taught as soon as it plausibly could be.

Proposal Author

- Zac Kologlu, with Andrew Taylor

With contributions:

- Shrey Somaiya, advice and perspective in course development
- Jashank Jeremy, advice and perspective in course development
- Tom Kunc, advice and perspective in course development
- Curtis Millar, advice and perspective in course development
- Hayden Smith, for COMP6080 proposal templates, aspects of COMP6771 course outline / structure