Solving Modern Programming Problems with Rust

Contents

If you have questions about the course or the current course outline, please reach out to the course email, listed below in the Course Details section.

Some items do not accurately reflect the course announcement (e.g. exercise deadlines and challenge/blog bonus marks). These will be updated before the first lecture.

Course Details

Course Code	COMP6991
Course Title	Solving Modern Programming Problems with Rust
Units of Credit	6
Course Contact	**<cs6991 at cse.unsw.edu.au>**
Lecturer	Zac Kologlu <z.kologlu at unsw.edu.au>
Admin	Shrey Somaiya <shrey.somaiya at unsw.edu.au>
Admin	Tom Kunc <t.kunc at unsw.edu.au>
Admin	Andrew Taylor <andrewt at unsw.edu.au>
Classes	**Lectures**: 1UGA: Mon 18:00—20:00; Tue 18:00—20:00; ... timetable for all classes
Course Website	https://dvorak.cse.unsw.edu.au/~cs6991/23T3/
Course Forum	https://dvorak.cse.unsw.edu.au/~cs6991/forum/
Handbook Entry	https://www.handbook.unsw.edu.au/undergraduate/courses/current/COMP6991/

Course Summary

This course aims to provide commentary and critique on the practice of programming, and the tooling used to program (primarily programming languages themselves). A variety of programming concepts across many programming languages are examined, including: syntax, typing, polymorphism, documentation, testing, meta-programming, concurrency, parallelism, safety, and more. The Rust programming language is used as a reference language to teach considerations behind these concepts. As a language commonly cited to be well-considered, it serves as a good foundation to help students understand where countless other languages may let them down. Lectures will compare and contrast Rust with other languages, but will also discuss where Rust can similarly let students down.

While proficiency in writing Rust programs is an important learning outcome of this course, of greater importance is the ability to write more robust programs in whichever language a student happens to be using at the time.

Per the advice of previous students, COMP6991 is a **difficult** course with a considerable workload. Please reach out to us either on the course forum or the course email if you're not sure whether COMP6991 is right for you.

Assumed Knowledge

As a 6000 level course, it is assumed that students are competent programmers that are able to engage in self-directed learning.

Before commencing this course, students should be able to ...

- write complex programs in the C programming language (COMP2521 equivalent)
- appreciate the fundamental complexity of programming; especially with respect to programs behaving unexpectedly
- understand snippets of code in unfamiliar languages when guided by a teacher

These are assumed to have been acquired in COMP2521 and other programming courses.

Learning Outcomes

After completing this course, students will be able to ...

1. Explain Rust's primary goals (safety, efficiency, productivity) and their motivations.
2. Develop programming solutions to substantial problems using Rust.
3. Produce effective program designs, and write unit tests and documentation tests to validate Rust programs.
4. Create well-considered, high performance, low cost abstractions to solve problems.
5. Reason about software through static guarantees to create robust interfaces and produce correct behaviour.
6. Manage Rust projects using the Rustup toolchain and Cargo package manager.

This course contributes to the development of the following graduate capabilities:

Graduate Capability	**Acquired in**
scholarship: understanding of their discipline in its interdisciplinary context	lectures, assignments
scholarship: capable of independent and collaborative enquiry	weekly exercises, workshops, assignments
scholarship: rigorous in their analysis, critique, and reflection	workshops
scholarship: able to apply their knowledge and skills to solving problems	workshops, weekly exercises, assignments
scholarship: ethical practitioners	all course-work, by doing it yourself
scholarship: capable of effective communication	assignments, workshops
scholarship: digitally literate	everywhere in CSE
leadership: enterprising, innovative and creative	assignments
leadership: collaborative team workers	workshops
professionalism: capable of operating within an agreed Code of Practice	all practical work

Teaching Strategies and Rationale

This course uses the standard set of practice-focussed teaching strategies employed by most CSE foundational courses:

- Lectures ... introduce concepts, show examples
- Workshops ... reinforce concepts and provide additional examples through practical work
- Weekly Exercises ... provide practical examples using various technologies
- Assignments ... allow you to solve larger practical problems

This course is taught in this format in order to demonstrate broad concepts in common programming languages during lectures, reinforce concepts with concrete examples in workshops, and allow students to apply their knowledge and skills to solving problems in weekly exercises and assignments.

Lectures

Lectures will be used to present the theory and practice of the programming concepts covered in this course. The lectures will include practical demonstrations of various programming languages. Lecture notes will be available on the course web pages before each lecture.

All lectures will be held in-person in Colombo Theatre A (K-B16-LG03). During lectures, discussion and questions are always welcome!

I will attempt to have watchable lecture recordings, and possibly live-streaming if we're very lucky. As a backup, last term's lecture recordings will also be made available.

Workshops

From week 1, you will also be expected to participate in a 2-hour workshop to work on larger practical examples in a classroom setting, usually working on a substantial program design as a class, before implementing smaller individual pieces in smaller "breakout" groups. The workshop will conclude by integrating all of the components together back into the original program design, testing, followed by detailed discussion on both program design both in the large and in the small. There are no marks for workshop participation. Most workshops are run in-person in J17 labs. If you have enrolled in such a class, please ensure you follow UNSW's Safe Return to Campus guidance. The remaining workshops are held in CATS rooms. These will be BYOD (Bring Your Own Device) workshops, where you should bring a laptop to class for your practical programming. If you don't have a personal laptop, you can hire one from the K17 CSE building. There is one "online" workshop -- this is for students who wish to do *self-directed* workshops, and subsequently ask their questions on the course forum. If you do not intend to join us in workshops this term, please select the "online" class so students who do intend to join are able to.

Weekly Exercises

Each week (excluding weeks 6 and 10), there will be one or more exercises to work on. These exercises will usually examine pratical examples relevant to previous lectures.

Completed exercises need to be submitted. You must submit exercises before the deadline using **give** to obtain a mark for a weekly exercise. The deadline to receive marks for a weekly exercise is the final day of term, i.e. **Friday Week 10**. Weekly exercises are released each Monday. The recommended due date is 20:00 on the Wednesday of the following week, however you **will** have until the end of term to submit for marks. The recommended due date gives you at least 9 days to complete each set of exercises, and we strongly recommend trying to follow it in order to keep up with the cadence of the course.

The weekly exercises for each week are worth in total 2.5 marks. All 8 of your weekly exercise marks will be summed to give you a mark out of 20.

Assignments

There are two assessable programming assignments. Assignments give you the chance to practice what you have learnt on relatively large problems (compared to the generally smaller weekly exercises). Assignments are a **very important** part of this course, therefore it is essential that you attempt them yourself.

The assignments are **indivdual** assessment tasks.

- Assignment 1, on Program design; due Week 7; worth 20%
- Assignment 2, on Concurrent programming; due Week 10; worth 25%

Late assignments submissions will be penalised. The penalty will follow the standard UNSW late penalty: a per-day mark reduction equal to 5% of the max assessment mark.

Final Exam

There will be an online exam which students completely remotely (from home). This will be centrally timetabled, and appear in your UNSW exam timetable.

It will contain a mixture of: implementation tasks (which will require you to write Rust programs), and “theory” questions (which require analysis and written answers). During this exam you will need to execute, debug and test your answers to implementation tasks which will be similar to those encountered in weekly exercises.

Student Conduct and Academic Integrity

Student Conduct

The **Student Code of Conduct** (Information, Policy) sets out what the University expects from students as members of the UNSW community. As well as the learning, teaching and research environment, the University aims to provide an environment that enables students to achieve their full potential and to provide an experience consistent with the University's values and guiding principles. A condition of enrolment is that students *inform themselves* of the University's rules and policies affecting them, and conduct themselves accordingly.

Students have the responsibility to observe standards of equity and respect in dealing with every member of the University community. This applies to all activities on UNSW premises and all external activities related to study and research. This includes behaviour in person as well as behaviour on social media; for example, in Facebook groups set up for the purpose of discussing UNSW courses or course work. Behaviour that is considered in breach of the Student Code Policy as discriminatory, sexually inappropriate, bullying, harassing, invading another's privacy, or causing any person to fear for their personal safety is serious misconduct, and can lead to severe penalties, including suspension or exclusion.

If you have any concerns, you may raise them with your lecturer, or approach the School Ethics Officer, Grievance Officer, or one of the student representatives.

Academic Integrity

UNSW has an ongoing commitment to fostering a culture of learning informed by academic integrity. All UNSW staff and students have a responsibility to adhere to this principle of academic integrity.

Plagiarism is defined as using the words or ideas of others and presenting them as your own. Plagiarism undermines academic integrity, and is not tolerated at UNSW. Instances of plagiarism are treated by UNSW and CSE as acts of academic misconduct, which carry penalties as severe as being excluded from further study at UNSW. There are several on-line resources to help you understand what plagiarism is and how it is dealt with at UNSW.

- Plagiarism and Academic Integrity
- UNSW Plagiarism Procedure

Make sure that you read and understand these. Ignorance is not accepted as an excuse for plagiarism. In particular, at CSE you are responsible for ensuring that your assignment files are not accessible by anyone but you by setting correct permissions in your CSE home directory and for any code repositories you may use. Note also that plagiarism includes paying or asking another person to do a piece of work for you, and then submitting it as your own work.

The pages below describe the policies and procedures in more detail:

- Student Code Policy
- Student Misconduct Procedure
- Plagiarism Policy Statement
- Plagiarism Procedure

You should also read the following page which describes your rights and responsibilities in the CSE context:

- Essential Advice for CSE Students

Assessment

Item	Topics	Due	Marks	LOs
Weekly Exercises	all topics	Week 1-5 + 7-9	20	2-6
Assignment 1	Program design	Week 7	20	1-6
Assignment 2	Concurrent programming	Week 10	25	1-6
Final Exam	all topics	exam period	35	1-3,5

Your final mark for this course will be computed using the above assessments as follows:

CourseWorkMark	=	ExerciseMark + Ass1Mark + Ass2Mark	out of 65
ExamMark			out of 35
FinalMark	=	CourseWorkMark + ExamMark	out of 100
FinalGrade	=		
	FL, if FinalMark < 50/100		
	PS, if 50/100 ≤ FinalMark < 65/100		
	CR, if 65/100 ≤ FinalMark < 75/100		
	DN, if 75/100 ≤ FinalMark < 85/100		
	HD, if FinalMark ≥ 85/100		

Course Schedule

The following is a rough schedule of when topics will be covered. This may change slightly over the session if topics take more or less time to cover.

The lectures will introduce topics on Monday and Tuesday. Workshops and weekly exercises will follow those same topics on Wednesday through Friday.

Week	Topics	Assigns
0	Course intro, toolchain setup	-
1	Rust intro, algebraic types	-
2	Collections, error handling	-
3	Borrowing, lifetimes	-
4	Documentation, testing, modularity	-

Week	Topics	Assigns
5	Polymorphism	-
6	Flexibility week	-
7	Functions, meta-programming	assign 1 due
8	Concurrency, parallelism	-
9	Unsafe, community crates	-
10	Wrap up, exam overview *(No exercises)*	assign 2 due

Topic Notes

Course intro, toolchain setup

- Introduction to the course
- Goals and aims for the term
- Expectations for students
- Course outline
- Creating Rust projects using Rustup / Cargo

Rust intro, algebraic types

- Variables, mutability, data types, functions
- If, while, for, everything as an expression
- Structs (product types) and enums (sum types)
- Common types (Option, Result)
- Pattern matching
- Ownership + move semantics (basic)

Collections, error handling

- Collections (vector, string, hashmap, iterator)
- Simple derive macros (Copy, Clone, Debug, etc.)
- Error handling (Result, panic)
- Try trait / operator

Borrowing, lifetimes

- Borrowing (References)
- Lifetimes
- Smart pointers (Box, Rc/Arc, Cell/Refcell)

Documentation, testing, modularity

- Modularising projects
- Documenting code
- Unit testing
- Documentation testing

Polymorphism

- Generics type parameters (parametric polymorphism)
- Monomorphization
- Traits (ad-hoc polymorphism)
- Static vs dynamic dispatch

Functions, meta-programming

- Closures
- Function types (fn, FnOnce, FnMut, Fn)
- Macros

Concurrency, parallelism

- Fearless concurrency!
- Threads
- Sync primitives (Mutex, RwLock, mpsc, etc.)
- Send / Sync traits
- Rayon

Unsafe, community crates

- Unsafe Rust
- Safe abstractions over unsafe operations
- Unsafe tooling (MIRI, sanitizers)
- Community crates

Wrap up

- Course wrap up
- More on community crates
- Current state and future of Rust
- Exam details

Resources for Students

There is no single book that covers all of the material in this course at the same level of detail as we are. The lecture notes should provide sufficient detail to introduce topics, and you will then study them in further depth in the workshops, exercises and assignments.

There are also many online resources available, and we will provide links to the most useful ones. Some are listed below. If you find others, please post links on the course forum.

The following is a Recommended Reading for this course:

- *The Rust Programming Language*, by Steve Klabnik and Carol Nichols, with contributions from the Rust Community

The book is accessible through the web as HTML, or a paperback / ebook copy can be ordered from No Starch Press.

Some suggestions for other books that cover at least some of the topics in this course:

- *Rust in Action: Systems programming concepts and techniques*, by Tim McNamara
- *Rust for Rustaceans: Idiomatic Programming for Experienced Developers*, by Jon Gjengset; complex reading for more intermediate Rust programmers

Documentation for the various systems used in the course is linked from the course website.

Course Evaluation and Development

Every term, student feedback is requested in a survey using UNSW's myExperience online survey system where the feedback will be used to make improvements to the course.

Students are also encouraged to provide informal feedback during the session, and to let course staff know of any problems as soon as they arise. Suggestions will be listened to openly, positively, constructively, and thankfully, and every reasonable effort will be made to address them.

23T1 was our second offering of COMP6991. Our myExperience feedback was again largely positive, and you can read the report for yourself here! We intend to address the following feedback this term:

- Weekly exercises due date
  * Weekly exercises now have a _recommended submission date_, as opposed to a strict _due date_. * The official due date of all weekly exercises is the end of term: Friday week 10. * Weekly exercises will be marked on a continuous basis. You _are_ allowed (and encouraged) to submit, view your marking, and resubmit many times with no penalty throughout the term.
- The course forum's tagging will be more specific; e.g. the "Assignment" tag will be split into "Assignment 1" and "Assignment 2".
- We are aiming for assignment feedback **sooner**.
  - With approximately the same student headcount as 23T1, we have increased our tutor capacity by approximately 50%.
  - This should mean less marking load per tutor (on average), and therefore a faster marking turnaround on assignments.
- Weekly exercise content will continue to improve.
  - New easier questions to help you get started.
  - New challenging questions to push your knowledge.
  - Challenge exercises will be worth bonus marks, as opposed to core marks within that week's exercises.
  - Extension content for those who want to play with ideas outside the scope of the course (e.g. WebAssembly)
- Lecture / course notes.
  - We will be developing hand-made course notes for your revision.
  - For many students, this may suffice to replace notes you would otherwise take yourself.
- Help sessions.
  - We will be attempting to introduce help sessions to COMP6991!
  - Previously we have had staff capacity issues, but we believe it may be possible for the first time this term.
- Guided assignment introductions.
  - We plan to generate more content to help students get started on the assignments.
  - This may include bonus lectures, pre-recorded content, or live group getting-started sessions.
- VCS assignment submission / marking.
  - We plan to **experiment** with using GitLab for assignment submission and marking.
  - We cannot _commit_ (ha) to this one just yet, but GitLab seems like a better interface for both student submissions and tutor marking. We'll investigate this one further!

Previously (23T1)...

22T3 was the first ever offering of COMP6991. myExperience feedback was largely positive, with some concessions noted regarding various growing pains that will not be run into again. We intend to address the remaining feedback this term:

- **To receive full marks in weekly exercises, you must make zero mistakes.**
  We will be offering students the chance to submit "blog posts" that will grant marks made from lost weekly exercise marks, increasing your mark leeway.
- **Some workshops were too complex, particularly for the allotted time.**
  We will be reviewing all workshops and modifying where is fit to reduce and/or restructure the workload.
- **Assignment specifications were unclear in places; expected behaviour sometimes ambiguous.**
  We will be updating both assignment specs and reference solutions to make intended behaviour clearer and more exact.
- **FFI / week 10 bonus topic (async) weren't the most interesting, content could be revisited?**
  We will be experimenting with demonstrating more industry crates (libraries) instead of FFI / async during weeks 9.5 and 10. This also addresses some feedback that desired a larger focus on interesting Rust crates!

Previously (22T3)...

COMP6991 has not previously run; we eagerly await your feedback!

We will be making a consistent effort throughout 22T3 to gather student feedback during the delivery of the course.

COMP6991 23T3: Solving Modern Programming Problems with Rust is brought to you by
the School of Computer Science and Engineering
at the University of New South Wales, Sydney.
For all enquiries, please email the class account at cs6991@cse.unsw.edu.au

CRICOS Provider 00098G

Login as tutor