



НОВІТНІ МОВИ ПРОГРАМУВАННЯ

Робоча програма навчальної дисципліни (Силабус)

Реквізити навчальної дисципліни

Рівень вищої освіти	<i>Перший (бакалаврський)</i>
Галузь знань	<i>Інформаційні технології</i>
Спеціальність	<i>121 Інженерія програмного забезпечення</i>
Освітня програма	<i>Інженерія програмного забезпечення інтелектуальних кібер-фізичних систем в енергетиці</i>
Статус дисципліни	<i>Вибіркова</i>
Форма навчання	<i>Заочна</i>
Рік підготовки, семестр	<i>4-й курс, 1-й семестр</i>
Обсяг дисципліни	<i>4 кред. /120 год. (заочна форма: лекцій 6 год., лаб. 4 год., СРС 110 год.)</i>
Семестровий контроль/ контрольні заходи	<i>Залік, модульна контрольна робота</i>
Розклад занять	<i>http://schedule.kpi.ua/</i>
Мова викладання	<i>Українська/Англійська</i>
Інформація про керівника курсу / викладачів	<i>Лектор: д. е. н., професор Сігайов Андрій Олександрович Практичні / Семінарські: Лабораторні: Сігайов А. О.</i>
Розміщення курсу	<i>GitHub Classroom, eCampus</i>

Програма навчальної дисципліни

1 Опис навчальної дисципліни, її мета, предмет вивчення та результати навчання

Чому майбутньому фахівцю варто вчити саме цю дисципліну?

Це не одразу очевидно, але мова програмування Rust повністю ґрунтується на розширенні можливостей: незалежно від того, який код ви зараз пишете, Rust дає вам змогу досягти більшого, програмувати з упевненістю у ширшому спектрі доменів, ніж ви це робили раніше.

Візьмемо, наприклад, роботу на системному рівні, яка розглядає низькорівневі деталі управління пам'яттю, представлення даних та багатопотоковість. Традиційно ця сфера програмування вважається таємничою, доступною лише небагатьом обраним, які присвятили необхідні роки навчанню, щоб уникнути її сумнозвісних підводних каменів. І навіть ті, хто це практикує, роблять це з обережністю, щоб їх код не був відкритий для вразливостей, збоїв чи ушкоджень.

Rust руйнує ці бар'єри, усуваючи старі пастки та надаючи дружній, перевірений набір інструментів, які допоможуть вам на цьому шляху. Програмісти, яким потрібно "зануритися" в низькорівневе управління, можуть зробити це за допомогою Rust, не беручи на себе звичний ризик збоїв або дірок у безпеці та не вивчаючи тонкощів мінливого ланцюга інструментів. Ще краще, ця мова розроблена для того, щоб природно направити вас до надійного коду, ефективного з точки зору швидкості та використання пам'яті.

Програмісти, які вже працюють з низькорівневим кодом, можуть використовувати Rust для підняття своїх амбіцій. Наприклад, запровадження багатопотоковості в Rust є операцією з

відносно низьким ризиком: компілятор буде ловити для вас класичні помилки. І ви зможете займатися агресивнішою оптимізацією у своєму кодї з упевненістю, що ви випадково не внесете збої чи вразливості.

Але Rust не обмежується програмуванням систем низького рівня. Він є виразним та ергономічним настільки, що робить досить приємним написання застосунків командного рядку, веб-серверів та багато інших видів коду - прості приклади будуть надані протягом цього курсу. Робота з Rust дозволяє сформувати навички, які передаються з одного домену до іншого; Ви можете вивчити Rust, написавши вебзастосунок, а потім застосувати ці самі навички у розробці коду для Raspberry Pi.

Цей курс повністю охоплює потенціал Rust, надаючи його користувачам розширені можливості. Це дружній та доступний матеріал, призначений для того, щоб допомогти вам підвищити рівень не тільки своїх знань про Rust, але й ваших досягнень та впевненості в собі як у програмістї взагалі. Тож занурюйтесь, готуйтеся до навчання - і ласкаво просимо до спільноти Rust!

Мета дисципліни. Ознайомити студентів з сучасною мовою програмування Rust.

Предмет дисципліни. Огляд мови Rust, включаючи:

- володіння та запозичення, тривалість життя та риси;
- гарантована безпека програм;
- тестування, оброблення помилок та ефективний рефакторинг;
- розумні вказівники, багатопотоковість та зіставлення зі зразком;
- робота зі вбудованим менеджером пакетів Cargo для створення, тестування, документування коду та управління залежностями;
- просунуті засоби роботи з Unsafe Rust.

Очікувані результати навчання.

Фахові компетентності.

ФК 16. Володіти скриптовими та декларативними мовами програмування.

Програмні результати навчання.

ПРН 15. Мотивовано обирати мови програмування та технології розробки для розв'язання завдань створення і супроводження програмного забезпечення.

ПРН 32. Застосовувати на практиці фундаментальні концепції, парадигми та основні принципи функціонування мовних, інструментальних і обчислювальних засобів інженерії програмного забезпечення.

2 Пререквізити та постреквізити дисципліни (місце в структурно-логічній схемі навчання за відповідною освітньою програмою)

Дисципліна вивчається у сьомому семестрі. Пререквізитами є курси "Комп'ютерна дискретна математика", "Основи програмування" та "Архітектура системного програмного забезпечення". Постреквізитів у даного курсу на бакалаврському рівні немає.

3 Зміст навчальної дисципліни

1. Початок роботи з Rust. Програмування гри-відгадайки. Загальні концепції програмування.
2. Концепція володіння.

3. Конструкція *Struct* для структурування пов'язаних даних.
4. Перерахування та зіставлення зі зразком.
5. Управління проектами, що зростають, за допомогою пакетів, скриньок і модулів.
6. Спільні колекції.
7. Оброблення помилок.
8. Узагальнені типи, риси та тривалість життя.
9. Автоматизовані тести.
10. Проєкт введення-виведення: побудова програми командного рядку.
11. Засоби функціонального програмування: ітератори та замкнення.
12. Докладніше про *Cargo* та *Crates.io*.
13. Розумні вказівники.
14. Багатопотоковість без остраху.
15. Засоби об'єктноорієнтованого програмування у *Rust*.
16. Зівставлення зі зразком.
17. Просунути засоби.
18. Фінальний проєкт: побудова багатопотокового веб-сервера.

4 Навчальні матеріали та ресурси

Базова література:

Клабник, С., Николс, К. Программирование на *Rust*: СПб.: Пупер, 2021. 592 с. URL: <http://libgen.rs/book/index.php?md5=57FC451B6927457078601721158474B9>

Додаткова література:

1. McNamara, T. *S. Rust in Action: Shelter Island, NY: Manning Publications, 2021. 456 с.* URL: <http://libgen.rs/book/index.php?md5=38FD5C0062ED46D7574BE9987952D655>
2. *Learn Rust in Y Minutes.* URL: <https://learnxinyminutes.com/docs/rust/> , *Learn X in Y minutes: Scenic Programming Language Tours*, (дата звернення: 31.08.21)
3. *Amos. A half-hour to learn Rust.* URL: <https://fasterthanli.me/articles/a-half-hour-to-learn-rust> , *fasterthanli.me*, (дата звернення: 02.01.21)
4. *Biedert, R. Rust Language Cheat Sheet.* URL: <https://cheats.rs/> , *Rust Language Cheat Sheet*, (дата звернення: 25.04.21)
5. *Rust Design Patterns.* URL: <https://rust-unofficial.github.io/patterns/> , *Rust Design Patterns*, (дата звернення: 03.01.21)
6. *Nethercote, N. The Rust Performance Book.* URL: <https://nnethercote.github.io/perf-book/title-page.html> , *The Rust Performance Book*, (дата звернення: 25.04.21)
7. *Messier, R. Beginning Rust Programming: Indianapolis, IN: Wiley, 2021. 416 с.* URL: <http://libgen.rs/book/index.php?md5=0FB67E531BF1CB7B188BF84F817291AE>
8. *Wolverson, H. Hands-on Rust: Effective Learning through 2D Game Development and Play: Raleigh, NC: Pragmatic Bookshelf, 2021. 332 с.* URL: <http://libgen.rs/book/index.php?md5=9C11FE8FF7D64EE9E10BBC8817EE3882>

5 Методика опанування навчальної дисципліни (освітнього компонента)

1. Початок роботи з Rust. Програмування гри-відгадайки. Загальні концепції програмування.
 - 1.1. Встановлення Rust.
 - 1.1.1. Встановлення Rust у Linux та macOS.
 - 1.1.2. Встановлення у Windows.
 - 1.1.3. Оновлення та деінсталяція Rust.
 - 1.1.4. Усунення несправностей.
 - 1.1.5. Локальна документація.
 - 1.2. Програмування гри-відгадайки. Налаштування нового проєкту.
 - 1.3. Обробка здогадки.
 - 1.3.1. Зберігання значень за допомогою змінних.
 - 1.3.2. Обробка потенційного збою за допомогою типу Result.
 - 1.3.3. Друк значень за допомогою заповнювачів команди println!
 - 1.3.4. Тестування першої частини.
 - 1.4. Генерація таємного числа.
 - 1.4.1. Використання скриньки для отримання більшої функціональності.
 - 1.4.2. Генерація випадкового числа.
 - 1.5. Порівняння здогадки з таємним числом.
 - 1.6. Циклічна обробка здогадок.
 - 1.6.1. Вихід з гри після правильної здогадки.
 - 1.6.2. Обробка неприпустимого введення.
 - 1.7. Загальні концепції програмування. Змінні та мінливість.
 - 1.7.1. Розбіжності між змінними та константами.
 - 1.7.2. Затінення.
 - 1.8. Типи даних.
 - 1.8.1. Скалярні типи.
 - 1.8.2. Складені типи.
 - 1.9. Функції.
 - 1.9.1. Параметри функцій.
 - 1.9.2. Інструкції та вирази в тілах функцій.
 - 1.9.3. Функції, що повертають значення.
 - 1.10. Коментарі.
 - 1.11. Потік управління.
 - 1.11.1. Вирази if.
 - 1.11.2. Цикли.
2. Концепція володіння.
 - 2.1. Що таке володіння?
 - 2.1.1. Правила володіння.
 - 2.1.2. Область видимості змінної.
 - 2.1.3. Рядковий тип.
 - 2.1.4. Пам'ять та її виділення.
 - 2.1.5. Володіння та функції.
 - 2.1.6. Повернені значення та область видимості.
 - 2.2. Посилання та запозичення.
 - 2.2.1. Змінювані посилання.
 - 2.2.2. Посилання, що бовтаються.
 - 2.2.3. Правила посилань.
 - 2.3. Зрізовий тип.
 - 2.3.1. Рядкові зрізи.
 - 2.3.2. Інші зрізи.

3. Конструкція `Struct` для структурування пов'язаних даних.
 - 3.1. Визначення та інстанціювання `Struct`.
 - 3.1.1. Стикла ініціалізація полів: коли у змінних та полів однакові імена.
 - 3.1.2. Створення екземплярів з інших екземплярів за допомогою синтаксису `Struct Update`.
 - 3.1.3. Використання кортежів без іменованих полів для створення різних типів.
 - 3.1.4. `Unit`-подібні `Struct` без жодних полів.
 - 3.2. Приклад програми з використанням `Struct`.
 - 3.2.1. Рефакторинг за допомогою кортежів.
 - 3.2.2. Рефакторинг за допомогою `Struct`: додавання більшого сенсу.
 - 3.2.3. Додавання корисної функціональності за допомогою похідних рис.
 - 3.3. Синтаксис методів.
 - 3.3.1. Визначення методів.
 - 3.3.2. Методи з більшою кількістю параметрів.
 - 3.3.3. Пов'язані функції.
 - 3.3.4. Множинні блоки `impl`.
4. Перерахування та зіставлення зі зразком.
 - 4.1. Визначення перерахування.
 - 4.1.1. Значення `Enum`.
 - 4.1.2. `Option Enum` та його переваги перед значеннями `Null`.
 - 4.2. Вираз `match` як оператор потоку управління.
 - 4.2.1. Патерни, що прив'язуються до значень.
 - 4.2.2. Зіставлення з `Option<T>`.
 - 4.2.3. Збіги є вичерпними.
 - 4.2.4. Заповнювач `_`.
 - 4.3. Стислий потік управління за допомогою `if let`.
5. Управління проектами, що зростають, за допомогою пакетів, скриньок і модулів.
 - 5.1. Пакети та скриньки.
 - 5.2. Визначення модулів для управління областю видимості та приватністю.
 - 5.3. Маршрути для посилання на елемент у дереві модуля.
 - 5.3.1. Демонстрація шляхів за допомогою ключового слова `pub`.
 - 5.3.2. Початок відносних шляхів за допомогою ключового слова `super`.
 - 5.3.3. Позначення `Struct` і `Enum` як публічних.
 - 5.4. Введення маршрутів у область видимості за допомогою ключового слова `use`.
 - 5.4.1. Створення ідіоматичних шляхів `use`.
 - 5.4.2. Надання нових імен за допомогою ключового слова `as`.
 - 5.4.3. Реекспорт імен за допомогою ключового слова `pub`.
 - 5.4.4. Використання зовнішніх пакетів.
 - 5.4.5. Використання вкладених маршрутів для очищення великих списків `use`.
 - 5.4.6. Оператор `glob`.
 - 5.5. Розділення модулів у різні файли.
6. Спільні колекції.
 - 6.1. Зберігання списків значень за допомогою векторів.
 - 6.1.1. Створення нового вектора.
 - 6.1.2. Оновлення вектора.
 - 6.1.3. Відкидання вектора відкидає всі його елементи.
 - 6.1.4. Читання елементів вектора.
 - 6.1.5. Перебирання значень вектора.
 - 6.1.6. Використання `Enum` для зберігання кількох типів.
 - 6.2. Зберігання тексту у кодуванні UTF-8 за допомогою рядків.
 - 6.2.1. Що таке тип `String`?
 - 6.2.2. Створення нового рядку.

- 6.2.3. *Оновлення рядку.*
- 6.2.4. *Індексування рядків.*
- 6.2.5. *Нарізка рядків.*
- 6.2.6. *Методи перебирання рядків.*
- 6.2.7. *Рядки не такі вже й прості.*
- 6.3. *Зберігання ключів з пов'язаними значеннями у відображеннях Hash Maps.*
 - 6.3.1. *Створення нового відображення.*
 - 6.3.2. *Відображення та володіння.*
 - 6.3.3. *Доступ до значень у відображенні.*
 - 6.3.4. *Оновлення відображення.*
 - 6.3.5. *Хеш-функції.*
- 7. *Оброблення помилок.*
 - 7.1. *Невідновлювані помилки з командою panic!*
 - 7.1.1. *Використання зворотного трасування при виклиці panic!*
 - 7.2. *Відновлювані помилки з командою Result.*
 - 7.2.1. *Застосування команди match до різних помилок.*
 - 7.2.2. *Стислі команди обробки паніки у випадку помилки: unwrap та expect.*
 - 7.2.3. *Розповсюдження помилок.*
 - 7.3. *To panic! or Not to panic!*
 - 7.3.1. *Приклади, код прототипів і тести.*
 - 7.3.2. *Випадки, коли у вас більше інформації ніж у компілятора.*
 - 7.3.3. *Принципи обробки помилок.*
 - 7.3.4. *Створення налаштовуваних типів для перевірки допустимості.*
- 8. *Узагальнені типи, риси та тривалість життя.*
 - 8.1. *Видалення повторів шляхом витягування функцій.*
 - 8.2. *Узагальнені типи даних.*
 - 8.2.1. *У визначеннях функцій.*
 - 8.2.2. *У визначеннях Struct.*
 - 8.2.3. *У визначеннях перерахувань.*
 - 8.2.4. *У визначеннях методів.*
 - 8.2.5. *Продуктивність коду з використанням узагальнень.*
 - 8.3. *Риси: визначення спільної поведінки.*
 - 8.3.1. *Визначення риси.*
 - 8.3.2. *Реалізація риси в типі.*
 - 8.3.3. *Реалізації за умовчанням.*
 - 8.3.4. *Риси як параметри.*
 - 8.3.5. *Повернення типів, які реалізують риси.*
 - 8.3.6. *Виправлення функції largest за допомогою границь риси.*
 - 8.3.7. *Використання границь риси для умовної реалізації методів.*
 - 8.4. *Перевірка посилань за допомогою тривалості життя.*
 - 8.4.1. *Запобігання посиланням, що бовтаються, за допомогою тривалостей життя.*
 - 8.4.2. *Контролер запозичень.*
 - 8.4.3. *Узагальнені тривалості життя у функціях.*
 - 8.4.4. *Синтаксис анотацій тривалостей життя.*
 - 8.4.5. *Анотації тривалостей життя у сигнатурах функцій.*
 - 8.4.6. *Мислення у термінах тривалостей життя.*
 - 8.4.7. *Анотації тривалостей життя у визначеннях Struct.*
 - 8.4.8. *Пропуск тривалості життя.*
 - 8.4.9. *Анотації тривалостей життя у визначеннях методів.*
 - 8.4.10. *Статична тривалість життя.*
 - 8.5. *Параметри узагальненого типу, границі риси та тривалість життя разом.*

9. Автоматизовані тести.

9.1. Як писати тести.

- 9.1.1. *The Anatomy of a Test Function.*
- 9.1.2. *Перевірка результатів макрокомандою assert!*
- 9.1.3. *Перевірка рівності макрокомандами assert_eq! і assert_ne!*
- 9.1.4. *Додавання повідомлень про помилки.*
- 9.1.5. *Перевірка на паніку за допомогою should_panic.*
- 9.1.6. *Використання типу Result<T, E> у тестах.*

9.2. Контроль виконання тестів.

- 9.2.1. *Паралельне або послідовне виконання тестів.*
- 9.2.2. *Показ результатів функції.*
- 9.2.3. *Виконання підмножини тестів за ім'ям.*
- 9.2.4. *Ігнорування декількох тестів, якщо вони явно не викликаються.*

9.3. Організація тестів.

- 9.3.1. *Модульні тести.*
- 9.3.2. *Інтеграційні тести.*

10. Проєкт введення-виведення: побудова програми командного рядка.

10.1. Приймання аргументів командного рядка.

- 10.1.1. *Читання значень аргументів.*
- 10.1.2. *Збереження значень аргументів у змінних.*

10.2. Читання файлу.

10.3. Рефакторинг з метою покращення модульності та обробки помилок.

- 10.3.1. *Розділення піклування у бінарних проєктах.*
- 10.3.2. *Виправлення обробки помилок.*
- 10.3.3. *Витягання логіки з таіп.*
- 10.3.4. *Розбиття коду на бібліотечну скриньку.*

10.4. Розвиток функціональності бібліотеки за допомогою розробки на основі тестів.

- 10.4.1. *Написання провального тесту.*
- 10.4.2. *Написання коду для проходження тесту.*

10.5. Робота зі змінними середовища.

- 10.5.1. *Написання провального тесту для функції search, нечутливої до реєстру.*
- 10.5.2. *Реалізація функції search_case_insensitive.*

10.6. Запис повідомлень про помилки у Standard Error замість Standard Output.

- 10.6.1. *Перевірка, куди записуються помилки.*
- 10.6.2. *Друк повідомлень про помилки у Standard Error.*

11. Засоби функціонального програмування: ітератори та замкнення.

11.1. Замкнення: анонімні функції, що можуть захоплювати середовище.

- 11.1.1. *Створення абстракції поведінки за допомогою замкнень.*
- 11.1.2. *Виведення типу та анотація замкнення.*
- 11.1.3. *Збереження замкнень за допомогою узагальнених параметрів та рис Fn.*
- 11.1.4. *Обмеження реалізації Cacher.*
- 11.1.5. *Захоплення середовища за допомогою замкнень.*

11.2. Обробка серії елементів за допомогою ітераторів.

- 11.2.1. *Риса Iterator і метод next.*
- 11.2.2. *Методи, що споживають ітератор.*
- 11.2.3. *Методи, що продукують інші ітератори.*
- 11.2.4. *Використання замкнень, які захоплюють своє середовище.*
- 11.2.5. *Створення власних ітераторів за допомогою риси Iterator.*

11.3. Покращення нашого проєкту введення-виведення.

- 11.3.1. *Видалення методу clone за допомогою ітератора.*
- 11.3.2. *Написання чистішого коду за допомогою адаптерів-ітераторів.*

11.4. Порівняння продуктивності: цикли проти ітераторів.

12. Докладніше про Cargo та Crates.io.

12.1. Налаштування збирань за допомогою профілів релізів.

12.2. Публікація скриньки на Crates.io.

12.2.1. Як зробити корисні коментарі для документації.

12.2.2. Експорт зручного публічного API за допомогою `pub`.

12.2.3. Налаштування облікового запису Crates.io.

12.2.4. Додавання метаданих до нової скриньки.

12.2.5. Публікація на Crates.io.

12.2.6. Публікація нової версії скриньки, що існує.

12.2.7. Видалення версій з Crates.io за допомогою `cargo yanks`.

12.3. Робочі простори Cargo.

12.3.1. Створення робочого простору.

12.3.2. Створення другої скриньки у робочому просторі.

12.4. Встановлення бінарних файлів з Crates.io за допомогою `cargo install`.

12.5. Розширення Cargo власними командами.

13. Розумні вказівники.

13.1. Використання `Vox<T>` для указання на дані в купі.

13.1.1. Використання `Vox<T>` для зберігання даних у купі.

13.1.2. Використання рекурсивних типів за допомогою `Vox`.

13.2. Поводження з розумними вказівниками як зі звичайними посиланнями за допомогою `Deref`.

13.2.1. Слідування за вказівником до значення за допомогою оператора розіменування.

13.2.2. Використання `Vox<T>` як посилання.

13.2.3. Визначення нашого власного розумного вказівника.

13.2.4. Поводження з типом як з посиланням шляхом реалізації риси `Deref`.

13.2.5. Неявне примусове приведення типів застосуванням `Deref` до функцій та методів.

13.2.6. Як примусове приведення типів за допомогою `Deref` взаємодіє з мінливістю.

13.3. Виконання коду при очищенні за допомогою риси `Drop`.

13.3.1. Дострокове відкидання значення за допомогою `std::mem::drop`.

13.4. `Rc<T>`, розумний вказівник для підрахунку посилань.

13.4.1. Застосування `Rc<T>` для спільного використання даних.

13.4.2. Клонування `Rc<T>` збільшує лічильник посилань.

13.5. `RefCell<T>` і патерн внутрішньої мінливості.

13.5.1. Дотримання правил запозичення під час ран-тайму за допомогою `RefCell<T>`.

13.5.2. Внутрішня мінливість: змінюване запозичення незмінного значення.

13.5.3. Наявність кількох власників змінюваних даних через поєднання `Rc<T>` і `RefCell<T>`.

13.6. Цикли у переходах за посиланнями призводять до витоку пам'яті.

13.6.1. Створення циклів у переходах за посиланнями.

13.6.2. Запобігання циклам у переходах за посиланнями: перетворення `Rc<T>` на `Weak<T>`.

14. Багатопотоковість без остраху.

14.1. Одночасне виконання коду за допомогою потоків.

14.1.1. Створення нового потоку за допомогою `spawn`.

14.1.2. Очікування завершення роботи всіх потоків за допомогою дескрипторів `join`.

14.1.3. Використання замкнення `move` з потоками.

14.2. Використання передачі повідомлення для пересилання даних між потоками.

14.2.1. Канали та передавання володіння.

- 14.2.2. Відправлення декількох значень і очікування приймача.
- 14.2.3. Створення декількох виробників шляхом клонування передавача.
- 14.3. Конкурентність спільного стану.
 - 14.3.1. Використання м'ютексів для забезпечення доступу до даних з одного потоку за один раз.
 - 14.3.2. Схожість між `RefCell<T>/Rc<T>` і `Mutex<T>/Arc<T>`.
- 14.4. Розширювана конкурентність за допомогою рис `Sync` і `Send`.
 - 14.4.1. Дозвіл передавання володіння між потоками за допомогою `Send`.
 - 14.4.2. Дозвіл доступу з декількох потоків за допомогою `Sync`.
 - 14.4.3. Ручна реалізація `Send` і `Sync` є небезпечною.
- 15. Засоби об'єктноорієнтованого програмування у Rust.
 - 15.1. Характеристики об'єктноорієнтованих мов.
 - 15.1.1. Об'єкти містять дані та поведінку.
 - 15.1.2. Інкапсуляція, що приховує деталі реалізації.
 - 15.1.3. Успадкування як система типів і як спільне використання коду.
 - 15.2. Використання об'єктів з рисами, які дозволяють значення різних типів.
 - 15.2.1. Визначення риси `Trait` для спільної поведінки.
 - 15.2.2. Реалізація риси `Trait`.
 - 15.2.3. Об'єкти з рисами виконують динамічну диспетчеризацію.
 - 15.2.4. Об'єктна безпека вимагається для об'єктів з рисами.
 - 15.3. Реалізація об'єктноорієнтованого патерну проектування.
 - 15.3.1. Визначення поста і створення нового екземпляру в стані чернетки.
 - 15.3.2. Зберігання тексту поста.
 - 15.3.3. Створюємо пусту чернетку.
 - 15.3.4. Запит на перевірку поста змінює його стан.
 - 15.3.5. Додавання методу `approve`, який змінює поведінку методу `content`.
 - 15.3.6. Компроміси патерну переходів між станами.
- 16. Зіставлення зі зразком.
 - 16.1. Де можна використовувати патерни зіставлення зі зразком.
 - 16.1.1. Гілки `match`.
 - 16.1.2. Умовні вирази `if let`.
 - 16.1.3. Умовні цикли `while let`.
 - 16.1.4. Цикли `for`.
 - 16.1.5. Інструкції `let`.
 - 16.1.6. Параметри функцій.
 - 16.2. Спростовність: можливість незбігу патерну.
 - 16.3. Синтаксис патернів.
 - 16.3.1. Зіставлення літералів.
 - 16.3.2. Зіставлення іменованих змінних.
 - 16.3.3. Декілька патернів.
 - 16.3.4. Зіставлення інтервальних значень за допомогою синтаксису ...
 - 16.3.5. Деструктурування об'єктів для виділення значень окремих частин.
 - 16.3.6. Ігнорування значень у патерні.
 - 16.3.7. Додаткові умови з обмежувачами збігу.
 - 16.3.8. Прив'язки `@`.
- 17. Просунути засоби.
 - 17.1. Небезпечний Rust.
 - 17.1.1. Небезпечні надздібності.
 - 17.1.2. Розіменування "сирого" вказівника.
 - 17.1.3. Виклик небезпечної функції або метода.
 - 17.1.4. Звернення до мінливої статичної змінної або її модифікація.
 - 17.1.5. Реалізація небезпечної риси.

- 17.1.6. Коли використовувати небезпечний код.
- 17.2. Просунуті риси.
 - 17.2.1. Специфікація типів-заповнювачів у визначеннях рис за допомогою пов'язаних типів.
 - 17.2.2. Параметри узагальненого типу за умовчанням та перезавантаження операторів.
 - 17.2.3. Повний синтаксис для усунення неоднозначності: виклик методів з однаковим іменем.
 - 17.2.4. Використання супер-рис, які вимагають функціональності однієї риси всередині іншої.
 - 17.2.5. Використання патерну Newtype для реалізації зовнішніх рис у зовнішніх типах.
- 17.3. Просунуті типи.
 - 17.3.1. Використання патерну Newtype задля безпеки типів і абстракції.
 - 17.3.2. Створення синонімів типів за допомогою псевдонімів типів.
 - 17.3.3. Тип Never, який ніколи не повертається.
 - 17.3.4. Динамічно змінювані типи та риса Sized.
- 17.4. Просунуті функції та замкнення.
 - 17.4.1. Вказівники функцій.
 - 17.4.2. Замкнення, що повертаються.
- 17.5. Макроси.
 - 17.5.1. Відмінність між макросами та функціями.
 - 17.5.2. Декларативні макроси за допомогою macro_rules! для загального метапрограмування.
 - 17.5.3. Процедурні макроси для генерації коду з атрибутів.
 - 17.5.4. Як написати власний налаштовуваний макрос derive.
 - 17.5.5. Макроси, подібні до атрибутів.
 - 17.5.6. Макроси, подібні до функцій.
- 18. Фінальний проєкт: побудова багатопотокового вебсервера.
 - 18.1. Побудова однопотокового сервера.
 - 18.1.1. Прослуховування TCP-з'єднання.
 - 18.1.2. Читання запиту.
 - 18.1.3. Детальніший погляд на HTTP-запит.
 - 18.1.4. Написання відповіді.
 - 18.1.5. Повернення реального HTML.
 - 18.1.6. Перевірка запиту та вибіркова відповідь.
 - 18.1.7. Невеличкий рефакторинг.
 - 18.2. Перетворення нашого однопотокового сервера на багатопотоковий сервер.
 - 18.2.1. Моделювання повільної відповіді у поточній реалізації сервера.
 - 18.2.2. Покращення пропускну здатності за допомогою пула потоків.
 - 18.3. Коректне відключення та очищення.
 - 18.3.1. Реалізація риси Drop для об'єкта ThreadPool.
 - 18.3.2. Подача потокам сигналу про припинення прослуховування завдань.

6 Самостійна робота студента/аспіранта

Студент витратить 3-4 година на тиждень на самостійну роботу з матеріалом курсу.

7 Політика навчальної дисципліни (освітнього компонента)

Студенти отримують бали за правильне та вчасне виконання лабораторних робіт. Загальний рейтинг (кількість балів) складається з: 1) лабораторних робіт (у формі практичних завдань з програмування) 60%, 2) заліку 40%.

Наразі в курсі наявні три лабораторні роботи, кожне оцінюється до 20 балів. Студент повинен здати правильно виконану лабораторну роботу протягом двох тижнів з дня видачі завдання для отримання повної кількості балів, в іншому випадку застосовуються штрафні бали не більше 40% від загальної кількості за лабораторну роботу.

8 Види контролю та рейтингова система оцінювання результатів навчання (PCO)

Поточний контроль: МКР, лабораторні роботи

Календарний контроль: заочною формою навчання не передбачений.

Семестровий контроль: залік

Умови допуску до семестрового контролю: зарахування усіх лабораторних робіт, семестровий рейтинг не менше 40 балів.

Таблиця відповідності рейтингових балів оцінкам за університетською шкалою:

Кількість балів	Оцінка
100-95	Відмінно
94-85	Дуже добре
84-75	Добре
74-65	Задовільно
64-60	Достатньо
Менше 60	Незадовільно
Не виконані умови допуску	Не допущено

9 Додаткова інформація з дисципліни (освітнього компонента)

- перелік питань, які виносяться на семестровий контроль (див. додаток до силабусу).

Робочу програму навчальної дисципліни (силабус):

Складено Професор кафедри інженерії програмного забезпечення в енергетиці, д.е.н., професор А. О. Сігайов

Ухвалено кафедрою інженерії програмного забезпечення в енергетиці (протокол № 28 від 15 травня 2023 р.)

Погоджено Методичною комісією Навчально-наукового інституту атомної і теплової енергетики КПІ ім. Ігоря Сікорського (протокол № 9 від 26 травня 2023 р.)