

**Федеральное государственное автономное образовательное
учреждение высшего образования
«Московский физико-технический институт
(национальный исследовательский университет)»**

УТВЕРЖДЕНО

**Директор физтех-школы
прикладной математики и
информатики**

А.М. Райгородский

	Рабочая программа дисциплины (модуля)
по дисциплине:	Программирование на Rust
по направлению:	Информатика и вычислительная техника
профиль подготовки:	Математическое моделирование и компьютерные технологии Физтех-школа Прикладной Математики и Информатики кафедра алгоритмов и технологий программирования
курс:	3
квалификация:	бакалавр

Семестр, формы промежуточной аттестации: 6 (весенний) - Экзамен

Аудиторных часов: 75 всего, в том числе:

лекции: 45 час.

семинары: 30 час.

лабораторные занятия: 0 час.

Самостоятельная работа: 120 час.

Подготовка к экзамену: 30 час.

Всего часов: 225, всего зач. ед.: 5

Количество контрольных работ, заданий: 2

Программу составили:

А.Д. Становой, учебный ассистент

В.В. Яковлев, канд. физ.-мат. наук, заведующий кафедрой

Программа обсуждена на заседании кафедры алгоритмов и технологий программирования 22.05.2023

Аннотация

В курсе изучается современное системное программирование на новом языке Rust. Это - естественное продолжение курсов Программирования на C++, Архитектуры Компьютерных и Операционных Систем, и Теории, и Практики Многопоточной Синхронизации, преподаваемых в МФТИ. Рассматривается история появления языка Rust, изучаются основы синтаксиса, стандартная библиотека, трейты, метапрограммирование, параллельные вычисления в языке, асинхронное программирование, тулинг вокруг языка, system safety на основе теории типов, примененных в языке для избежания memory unsafety, а также небезопасное подмножество языка. Активно производится сравнительный анализ с языками C++, Go, Python, Java.

В курс входит несколько десятков задач, простых и углубленных, призванных разобрать на практике разные аспекты данного языка. Также есть 3 проекта, предназначенных для написания цельного продукта на языке.

1. Цели и задачи

Цель дисциплины

- овладение студентами правил языка программирования Rust и приемами использования языка Rust в практике программирования.

Задачи дисциплины

- изучение базовых и продвинутых возможностей языка Rust;
- распространение его среди молодых разработчиков.

2. Перечень формируемых компетенций

Освоение дисциплины направлено на формирование следующих компетенций:

Код и наименование компетенции	Индикаторы достижения компетенции
УК-6 Способен управлять своим временем, выстраивать и реализовывать траекторию саморазвития на основе принципов образования в течение всей жизни	УК-6.1 Определяет приоритеты профессиональной деятельности и способы ее совершенствования на основе самооценки
	УК-6.2 Способен планировать самостоятельную деятельность в решении профессиональных задач; подвергать критическому анализу проделанную работу; находить и творчески использовать имеющийся опыт в соответствии с задачами саморазвития
ОПК-1 Способен применять фундаментальные знания, полученные в области физико-математических и (или) естественных наук, и использовать их в профессиональной деятельности	ОПК-1.1 Способен анализировать поставленную задачу, намечать пути ее решения
	ОПК-1.2 Способен строить математические модели, производить количественные расчеты и оценки
	ОПК-1.3 Способен определять границы применимости полученных результатов

3. Перечень планируемых результатов обучения по дисциплине (модулю)

В результате освоения дисциплины обучающиеся должны

знать:

- принцип исполнения программ на Rust.

уметь:

- реализовывать библиотеку общего назначения по заданным интерфейсам.

владеть:

- навыками работы с объектами и потоками и кругозором в выборе архитектурного решения поставленной задачи.

4. Содержание дисциплины (модуля), структурированное по темам (разделам) с указанием отведенного на них количества академических часов и видов учебных занятий

4.1. Разделы дисциплины (модуля) и трудоемкости по видам учебных занятий

№	Тема (раздел) дисциплины	Трудоемкость по видам учебных занятий, включая самостоятельную работу, час.			
		Лекции	Семинары	Лаборат. работы	Самост. работа
1	Здравствуй, Rust! Обсуждение языка. Сравнение с C++. Основы языка.	2	2		9
2	Стандартная библиотека и коллекции.	2	2		9
3	Трейты. Функциональные возможности языка. Итераторы.	2	2		9
4	Таблица виртуальных методов. Управление памятью и формальные корни system safety.	4	2		9
5	Метапрограммирование в Rust. Написание идиоматичного кода.	4	4		9
6	Работа с файловой системой. Формальные корни system safety: исследования RustBelt.	4	4		9
7	Пакетный менеджер языка Rust: Cargo. Обработка ошибок.	4	2		9
8	Автоматические средства верификации и поддержки Rust кода.	4	2		9
9	Многопоточное программирование в Rust.	4	2		9
10	Асинхронный Rust и нетворкинг.	4	2		9
11	Unsafe Rust. Репрезентация типов в памяти.	5	2		9
12	Rust и взаимодействие с другими языками и операционной системой.	2	2		9
13	Техники ускорения Rust кода.	4	2		12
Итого часов		45	30		120
Подготовка к экзамену		30 час.			
Общая трудоёмкость		225 час., 5 зач.ед.			

4.2. Содержание дисциплины (модуля), структурированное по темам (разделам)

Семестр: 6 (Весенний)

1. Здравствуй, Rust! Обсуждение языка. Сравнение с C++. Основы языка.

- Почему нам вообще нужен Rust? Какие проблемы Rust решает? Где предполагается использовать Rust? Какие компании уже используют Rust? Где найти работу программистом на Rust?
- Что значат safe и unsafe. Что значат sound и unsound. Что Rust считает безопасным. Формальная модель RustBelt.
- Ключевые слова let и mut. Типы i8...i128, u8...u128, isize, usize, f32, f64, bool. Литералы. Shadowing.
- Ключевое слово as для примитивных кастов. Нетранзитивность кастов.
- Функции. Ключевое слово return. Выражения и условия.
- struct. Functional update. self и Self. Ключевое слово impl. Ассоциированные функции. Значение _.
- enum. size_of enum и дискриминант. std::cmp::Ordering. Сравнение с enum и union из C++.

- Синтаксис `if`, `while`. Именованный `break`. Синтаксический сахар `if/while` и `let`.
- `match`. Range matching. Множественные паттерны. Binding modes. Деструктуризация.
- Вывод типов.
- `T`, `&mut T`, `&T`.
- Введение в generics. Введение в `Vec` и его интерфейс. Слайсы `&[T]`.
- Напоминание о `std::reference_wrapper` из C++.
- `Unit` тип `()`.
- `panic!`, `unimplemented!`, `unreachable!`, `todo!`.
- `loop`. “Ненаселенный” тип `!`.
- Borrow checker: как не сдаться и выучить Rust. Механизм “заимствования”.
- Введение в экосистему Rust. Плагины VS Code: официальный и `rust-analyzer`.
- Написание простой программы на Rust: команда `cat`. `std::prelude`. Основы `cargo` и `rustc`.

2. Стандартная библиотека и коллекции.

- Compound типы `array` и `tuple`. Что изменяется, когда их размер большой.
- `Result` и его интерфейс.
- `Option`, его интерфейс и оптимизации компилятора.
- `VecDeque` и его интерфейс.
- `HashMap`, `HashSet` и их интерфейс. Асимптотики.
- `BTreeMap`, `BTreeSet` и их интерфейс. Почему B-дерево не может быть полной заменой `std::map`.
- `LinkedList`, `BinaryHeap` и их интерфейс.
- `String`. Случайный доступ. UTF-8. `&str` и напоминание о `string_view` из C++. `char` и значение понятия Unicode scalar value.
- Аллокации на куче: `Box`, `Rc`, `Cow` и их интерфейс. Упоминание `Arc`.
- Почему `Rc` немутабелен.
- Модуль `std::cell`. Interior mutability. `Cell`, `RefCell`. Reentrancy.
- `std::mem` модуль и его безопасные функции: `size_of`, `swap`, `replace`, `forget`, `drop`.
- Основы Drop checker. Drop flags. Стабильность порядка drop и причины. Порядок инициализации.
- Exotically sized types: `ZST`, `DST`, `Empty`. Контейнеры, особенно `Vec`, когда `T` - это `ZST`.
- `NonNull`, `NonZero`.
- `println!`, `println!`, `eprintln!`, `eprintln!`, `write!`, `writeln!` и блокирование IO.
- `BufReader` и `BufWriter`. Их интерфейс и уменьшение числа аллокаций.

3. Трейты. Функциональные возможности языка. Итераторы.

- Трейты. Return type polymorphism. Автоматические трейты. Ключевое слово `where`. Extension traits.
- Основные трейты стандартной библиотеки и их возможности. `Default`, `Clone`, `Copy`. Почему они не генерируются по умолчанию? `Ord`, `PartialOrd`. `Eq`, `PartialEq`. `Hash`, `Hasher`. `Drop`, `ManuallyDrop` и идиома RAII. Почему нельзя полагаться на drop order. `Deref`, `DerefMut`, `Borrow`. `Index` and `IndexMut`.
- Модуль `std::ops`. Трейты `Add`, `Sub`, `Mul`, `Div`, `Rem`, `BitAnd`, `BitOr`, `BitXor`, `Shl`, `Shr` и их -Assign варианты. `Not`, `Neg`.
- Трейты `Debug` и `Display`. `Formatter`. Мотивация их дизайна. Трейт `ToString`.
- Трейты `FnOnce`, `Fn`, `FnMut`. Замыкания. Capture clause. Ключевое слово `move`. Variable rebinding в отдельной области видимости.
- Модуль `std::convert`. Трейты `From` и `Into`, `TryFrom` и `TryInto`, `AsRef` и `AsMut`. Функция `identity`.
- Ассоциированные типы и константы.
- Итераторы. Ленивость итераторов. Трейты `Iterator`, `IntoIterator`. Имплементации итераторов в `std`. Итераторы в рантайме.
- API итераторов: `map`, `filter`, `fold`, `flatten` и другие.
- Мотивация дизайна трейта `Iterator`.

- Инвалидация итераторов в C++ и Rust.
- Итераторы и векторизация. Как вернуть итератор и замыкание из функции: ключевое слово `impl`.
- Полезные функции из модуля `std::iter`: `from_fn`, `empty`, `once`, `repeat`, `repeat_with`.
- Трейты `FromIterator`, `ExactSizeIterator`, `DoubleEndedIterator`, `Index`, `IndexMut`.
- `collect`, `flatten` и их имплементация.

4. Таблица виртуальных методов. Управление памятью и формальные корни `system safety`.

- Таблица виртуальных методов. Толстый указатель. Ключевое слово `dyn`. Динамическая диспетчеризация. Динамическая диспетчеризация на стеке. `Dynamically sized types`.
- Модуль `std::any`. Трейт `Any`.
- `Type coercion` и `subtyping`. `Fully Qualified Syntax` и когда его нужно использовать.
- `Object Safety`. `Generics` в таблице виртуальных методов. `Hash` и инлайнинг.
- Как Rust управляет памятью: `aliasing` и правило “`Aliasing XOR Mutability`” (AXM).
- `Borrow checker`, `affine` система типов.
- `Lifetimes`. Именованные ссылки. `Lifetime elision`. `Reborrowing`.
- Неограниченный `'static lifetime`: почему он нам нужен и какое он имеет отношение к остальным лайфтаймам.
- `Higher-Rank Trait Bounds (HRTB)`. `Variance`.
- Ключевое слово `ref` и `match`. `Two phase borrows`.
- `Drop checker`. Связь между `PhantomData` и `variance inference`.
- Оператор точка и правила автоматического вывода типов.

5. Метапрограммирование в Rust. Написание идиоматичного кода.

- `Generics`. Мономорфизация. Статический и динамический полиморфизм.
- `Trait specialization`.
- Причина, по которой нет частичной специализации `generics`: отвратительные последствия `SFINAE`.
- Макросы. `macro_rules!`. Паттерны, `$crate`. Идентификаторы. Гигиеничность. Проблемы макросов. Внутренние макросы.
- Основы крейта `serde`.
- Атрибуты. `non_exhaustive`, `deprecated`. Макросы `env!`, `option_env!`, `stringify`.
- Условная компиляция и крейт `cfg-if`.
- Процедурные макросы. `derive`, `cfg`, `test`. `recursion_limit` атрибуты для макросов.
- Основы крейта `syn`.
- Метапрограммирование. Вычисление констант и ключевое слово `const`. `Const generics`. Генерация кода макросами.
- Паттерны проектирования: `command`, `interpreter`, `newtype idiom`, `strategy`, `visitor`, `builder`, `fold`.
- Антипаттерны проектирования: использование `deref polymorphism`.
- Советы написания идиоматичного кода.

6. Работа с файловой системой. Формальные корни `system safety`: исследования `RustBelt`.

- Работа с файловой системой с модулем `std::fs`. Сравнение дизайна Rust и Go.
- Статья `GhostCell`. Обсуждение силы системы типов и статических проверок в Rust.
- `Aliasing model`. Статья `stacked borrows`.

7. Пакетный менеджер языка Rust: `Cargo`. Обработка ошибок.

- `Cargo`. `Crates and modules`. `Compilation unit`. `What's in a crate`. `Coherence`.
- Структура `Cargo` пакета. `Cargo.lock`, `semantic versioning`. `Rustup`. `crates.io`. Типы библиотечных крейтов.
- `use`, `mod`, `pub`, `super`, `crate`. Где `pub` не работает.
- Релизный цикл Rust. Сырые идентификаторы. Мигрирование на другую версию.

- Обработка ошибок. Recoverable и unrecoverable ошибки. Паника и stack unwinding. Unwind safety. Result<T, E>. Оператор ? и устаревший try!. Лучшие практики обработки ошибок.
- Exception safety: минимальный и максимальный уровень.
- Управление паникой. catch_unwind, resume_unwind.
- Трейт Error и его проблемы.
- Основы крейтов anyhow и thiserror.

8. Автоматические средства верификации и поддержки Rust кода.

- Clippy и линты.
- Rust analyzer.
- MIR интерпретатор Miri.
- Rudra.
- Инструменты Dynamic Symbolic Execution (DSE): Rust verification tools (RVT), Cargo-KLEE.
- Модел чекеры: Rust Model Checker (RMC), SMACK
- Инструменты верификации: Haybale, Stateright.

9. Многопоточное программирование в Rust.

- Модель памяти Rust. Orderings. Connection of memory safety and absence of data races.
- Модуль std::thread.
- Поток. Thread builder. Область видимости и static. thread::scoped и его проблемы. Closure scope. паника в closure scope.
- Основы крейтов rayon и crossbeam.
- Трейты Send и Sync. Unsafe трейты. Ord и undefined behavior.
- std::atomic: Atomic и fence.
- Arc и пример undefined behavior в unsafe коде. Mutex и poisoning. RwLock. Lazy.

10. Асинхронный Rust и нетворкинг.

- Мотивация дизайна асинхронной стороны Rust.
- async и await.
- Stackless coroutines.
- Трейт Future. Создание экзекутора.
- Pin, Unpin и примеры их использования. PhantomPinned.
- Обсуждение устройства различных протоколов нетворкинга.
- Модель Open Systems Interconnect (OSI). Напоминание о Ethernet, IP, UDP and TCP.
- Нетворкинг в Rust. Модуль std::net. IpAddr. TcpListener, TcpStream и UdpSocket.
- Сохранение метрик в Rust.
- Основы крейтов tokio и loom.

11. Unsafe Rust. Репрезентация типов в памяти.

- Ключевое слово unsafe. Контракт между safe и unsafe кодом. Что умеет unsafe. Когда нам нужен unsafe. Проблема: безопасный mem::forget. mem::transmute. Ключевое слово static. UnsafeCell. Указатели: *const T, *mut T.
- MaybeUninit. Оптимизации компилятора, Container<MaybeUninit<T>>> и Container<T>.
- Unsafe трейты. Проблемы с safety у BTreeMap и трейта Ord.
- WTF-8. PathBuf и Path.
- CString и CStr. OsString и OsStr.
- Основы крейтов cbindgen и rust-bindgen.
- Drop checker. Атрибут may_dangle.
- Имплементация split_at_mut.
- Обсуждение имплементации Vec.
- Обсуждение имплементации Arc и Mutex.

- Модуль `std::alloc`. Функции `alloc`, `alloc_zeroed`, `dealloc`, `Layout`. `GlobalAlloc`.
- `realloc` в Rust и C++. Как `move` работает в Rust и C++. Когда ломается `move` в C++. Обсуждение крейта `moveit`. Копирование и клонирование в Rust.
- `Leaking`. Обсуждение крейта `once_cell`.
- Репрезентация типов в памяти. `Exotically sized type Extern`. `enum` в памяти. Гарантии `Option`. “Разметка” данных: `C`, `transparent`, `u*`, `i*`, `packed`, `align(n)`. Порядок полей.

12. Rust и взаимодействие с другими языками и операционной системой.

- Секции `.data`, `.rodata`, `.bss` и `.text`. Структура кучи и стека. Переполнение буфера.
- Использование C/C++ кода из Rust. Использование Rust кода из C/C++.
- Состояние Rust ABI.
- Советы для написания FFI.
- Обработка сигналов.
- Вызовы методов ядра.
- Состояние Rust в ядре Linux.

13. Техники ускорения Rust кода.

- Модуль `core::arch` и SIMD. `core::intrinsics` и его использование в стандартной библиотеке.
- Модуль `std::hint`.
- Link-time Optimization (LTO).
- Profile-guided Optimization (PGO).
- Профилирование Rust кода.
- Инлайнинг кода. Причины и примеры.
- `dyn` против `impl` когда оба могут быть использованы.
- Заметка о `HashSet` и `HashMap`. `Siphash` и альтернативы.
- Удаление и уменьшение числа аллокаций и реаллокаций. Профилирование `malloc` и `free`.
- Проблема коротких векторов. Обсуждение крейтов `smallvec` и `arrayvec`.
- Ускорение компиляции и линковки больших проектов на Rust.

5. Описание материально-технической базы, необходимой для осуществления образовательного процесса по дисциплине (модулю)

Компьютер и мультимедийное оборудование (проектор, звуковая система).

6. Перечень рекомендуемой литературы

Основная литература

1. Введение в программирование , учебное пособие / И. Ю. Баженова, В. А. Сухомлин. — Москва, ИНТУИТ, 2016.— URL: <https://e.lanbook.com/book/100695> (дата обращения: 30.12.2020). - Полный текст (Режим доступа : из сети МФТИ / Удаленный доступ)

Дополнительная литература

7. Перечень ресурсов информационно-телекоммуникационной сети "Интернет", необходимых для освоения дисциплины (модуля)

Не используются

8. Перечень информационных технологий, используемых при осуществлении образовательного процесса по дисциплине (модулю), включая перечень необходимого программного обеспечения и информационных справочных систем (при необходимости)

Система GitLab для автоматического тестирования домашней работы.

9. Методические указания для обучающихся по освоению дисциплины (модуля)

- Материалы лекций в репозитории курса.
- Задачи курса.
- Студенту будет предложено зайти в Telegram канал и чат, где он сможет оперативно получать дополнительные материалы: блоги, презентации с конференций, статьи, ответы на свои вопросы.

ОЦЕНОЧНЫЕ МАТЕРИАЛЫ ПО ДИСЦИПЛИНЕ (МОДУЛЮ)

по направлению:	Информатика и вычислительная техника
профиль подготовки:	Математическое моделирование и компьютерные технологии Физтех-школа Прикладной Математики и Информатики кафедра алгоритмов и технологий программирования
курс:	3
квалификация:	бакалавр

Семестр, формы промежуточной аттестации: 6 (весенний) - Экзамен

Разработчики:

А.Д. Становой, учебный ассистент

В.В. Яковлев, канд. физ.-мат. наук, заведующий кафедрой

1. Компетенции, формируемые в процессе изучения дисциплины

Код и наименование компетенции	Индикаторы достижения компетенции
УК-6 Способен управлять своим временем, выстраивать и реализовывать траекторию саморазвития на основе принципов образования в течение всей жизни	УК-6.1 Определяет приоритеты профессиональной деятельности и способы ее совершенствования на основе самооценки
	УК-6.2 Способен планировать самостоятельную деятельность в решении профессиональных задач; подвергать критическому анализу проделанную работу; находить и творчески использовать имеющийся опыт в соответствии с задачами саморазвития
ОПК-1 Способен применять фундаментальные знания, полученные в области физико-математических и (или) естественных наук, и использовать их в профессиональной деятельности	ОПК-1.1 Способен анализировать поставленную задачу, намечать пути ее решения
	ОПК-1.2 Способен строить математические модели, производить количественные расчеты и оценки
	ОПК-1.3 Способен определять границы применимости полученных результатов

2. Показатели оценивания компетенций

В результате изучения дисциплины «Программирование на Rust» обучающийся должен:

знать:

- принцип исполнения программ на Rust.

уметь:

- реализовывать библиотеку общего назначения по заданным интерфейсам.

владеть:

- навыками работы с объектами и потоками и кругозором в выборе архитектурного решения поставленной задачи.

3. Перечень типовых (примерных) вопросов, заданий, тем для подготовки к текущему контролю

1. Система типов.
2. Числовые типы.
3. Коллекции.
4. Строковые типы.
5. Константы.
6. Управление памятью.
7. Синтаксис языка.
8. Объектная система.
9. Особенности программирования.
10. Сравнение с другими языками.

4. Перечень типовых (примерных) вопросов и тем для проведения промежуточной аттестации обучающихся

1. Реализовать граф объектов на сырых указателях.
2. В файле `src/lib.rs` реализуйте функцию `add`, которая складывает два числа.
3. Напишите несколько функций для работы с итераторами:
`count()` - создаёт итератор, который возвращает по порядку числа от 0 до `u64::MAX`. При вызове `.next()` после того, как вернулся `u64::MAX`, можно паниковать.
4. Напишите несколько функций для работы с итераторами:
`cycle(into_iter)` - создаёт бесконечный итератор, который возвращает зацикленно последовательность

- элементов, полученную из `into_iter`. Заметьте, что нижележащий итератор должен истощаться лениво,
- а не сразу при вызове `cycle`. `Item` обязан быть `Clone`.
5. Напишите несколько функций для работы с итераторами:
- `extract(into_iter, index)` - возвращает пару, первый элемент которой - `Option<Item>`, где `Item` - элемент итератора под номером `index` (начиная с 0), если таковой есть. Второй элемент пары - итератор, который должен возвращать элементы исходного итератора в той же последовательности, но без извлечённого элемента. Заметьте, что исходный итератор должен быть истощён не более, чем на `index + 1` элемент.
6. Напишите несколько функций для работы с итераторами:
- `tee(into_iter)` - возвращает пару из двух независимых итераторов, каждый из которых возвращает ровно ту же последовательность элементов, что исходный итератор. Исходный итератор должен при этом истощаться лениво: из него всегда должно быть извлечено кол-во элементов, равное максимуму из кол-ва извлечённых элементов у двух итераторов, которые `tee` вернул. `Item` обязан быть `Clone`.
7. Напишите несколько функций для работы с итераторами:
- `group_by(into_iter, f)` - объединяет все идущие подряд элементы, для которых `f` возвращает одинаковое значение, в группы. `group_by` возвращает итератор по парам, где первый элемент - это `f(item)`, а второй элемент - это вектор из идущих подряд элементов, таких, что `f(item)` для каждого из этих элементов равен первому элементу пары.
8. Напишите библиотеку ORM (Object-relational mapping).
9. Реализуйте интерпретатор игрушечного языка программирования `Polka`.
10. Напишите распараллеленный `grep`.

Критерии оценивания

отлично

- 10 Полностью и вовремя решены все задачи без ошибок. Продемонстрирован грамотный подход к решению задач, реализованы оптимальные алгоритмы, код оформлен в едином удобочитаемом стиле
- 9 Полностью и вовремя решены все задачи без ошибок. Продемонстрирован грамотный подход к решению задач, реализованы оптимальные алгоритмы
- 8 Полностью и вовремя решены все задачи без ошибок. Продемонстрирован грамотный подход к решению задач

хорошо

- 7 Полностью решены все задачи. Допущены несущественные ошибки.
- 6 Полностью решено большинство задач. В некоторых задачах допущены и не исправлены ошибки, либо некоторые задачи решены частично.
- 5 Полностью решено две трети задач. В некоторых задачах допущены и не исправлены ошибки, либо некоторые задачи решены частично.

удовлетворительно

- 4 Полностью решено более половины задач. В остальных задачах допущены и не исправлены ошибки, либо некоторые задачи решены частично.
- 3 Полностью решено более половины задач.

неудовлетворительно

- 2 Решено менее половины задач.
- 1 Не решено ни одной задачи.

5. Методические материалы, определяющие процедуры оценивания знаний, умений, навыков и (или) опыта деятельности

Во время проведения экзамена обучающиеся могут пользоваться программой дисциплины,