УДК 004.42

СРАВНИТЕЛЬНЫЙ АНАЛИЗ: КЛАССИЧЕСКИЕ И СОВРЕМЕННЫЕ ЯЗЫКИ ПРОГРАММИРОВАНИЯ

Башер Али Аль-Нами
кандидат технических наук, доцент
alnomibasheer@gmail.com
Диана Олеговна Быга
студент
diana20bg06@gmail.com
Владислав Дмитриевич Васильев
студент
vladislavvasilev53@gmail.com
Санкт-Петербургский государственный университет телекоммуникаций
им. проф. М.А. Бонч-Бруевича
г. Санкт-Петербург, Россия

Аннотация. В этой работе мы сравним два языка программирования, первый традиционный C, а другой современный - Rust. Мы сравним их, используя анализированные нами источники. Оценим преимущества и расскажем о недостатках.

Ключевые слова: C, Rust, программирование, синтаксис, безопасность, производительность, компиляция.

Введение. Первые ЭВМ появились почти век назад. За это время интерфейсы сильно видоизменялись и улучшались. Для облегчения работы в 1947–48 годах были спроектированы первые ассемблеры (трансляторы). Они транслировали текстовые файлы, написанные понятным для человека языком в двоичный код. В 1957 году появился первый высокоуровневый язык программирования – Fortran. Его широко использовали для научных и инженерных вычислений. Это следует из названия - FORmula TRANslator (формульный транслятор) [1].

В этой работе мы сравним 2 языка программирования: традиционный язык C и новый язык программирования Rust. Традиционные языки программирования — это языки, которые использовались продолжительное время и стали основой многих языков программирования. Такие языки обычно придерживаются строгого синтаксиса и структуры. Язык C идеально подходит под это описание, ведь вдохновляясь им, создавали такие языки как C++, который считается расширенной версией языка C, потому что в него добавили объектно-ориентированные возможности. Однако сейчас их можно назвать самодостаточными и развиваются независимо друг от друга. C# и Java во многом позаимствовали синтаксис у языка C, что сделало их более доступными для разработчиков, раннее работающих на C [2].

Язык C появился в 1969–1973 годах в компании Bell Labs программистом Деннисом Ритчем. Он представляет собой компилируемый язык со статической типизацией. Изначально он предназначался для написания операционной системы Unix, однако его стали использовать также для системного программирования, создания драйверов, антивирусов, различных утилит и т.д.

У языка C есть только стандартная библиотека, но и она невелика. Также в этом языке отсутствует множество функции, автоматическое управление памятью, объектно-ориентированное программирование, а также среды функционального программирования. У данного языка очень простая языковая база, из которой в стандартную библиотеку вынесены многие существенные возможности, вроде математических функций или функций работы с файлами.

Также он имеет высокую скорость выполнения, что является важным фактором для системного программирования. У него хорошая переносимость, что позволяет компилировать на различных платформах, число которых огромное количество [3].

Для того, чтобы приступить к сравнению нужно узнать еще о языке Rust. Этот язык программирования был создан компанией Mozilla программистом Грейдоном Хоар в 2006–2015 году для большой производительности и безопасности. Это современный язык программирования, который можно назвать кроссплатформенным. Также как и язык C, Rust является статически типизированным языком программирования. Такие приложения как Figma, Discord, GitHub, Amazon (AWS), Dropbox и многие другие частично или полностью написаны на языке Rust [4].

Так как Rust современный язык его синтаксис отличается от старых языков. Он предоставляет удобные инструменты, которые делают код более понятным и легким для поддержания. Также одной из его главной особенностью является безопасная работа с памятью, с помощью особой системы управления. Программисты могут передавать владение памятью между частями программы, но только таким образом, чтобы не возникало ситуации, когда несколько частей программы пытаются одновременно использовать один и тот же ресурс [2,5]. С помощью сборщика мусора его производительность быстрее, чем у большинства других языков. Несмотря на огромное количество плюсов и особенностей, Rust имеет и недостатки. Количество библиотек, в сравнении с другими современными языками программирования, такими как Java или Python, пока что ограничено. Время компиляции может быть дольше, чем у языков типа C, из-за особенностей компилятора и системы проверки типов. У языка Rust могут возникнуть проблемы с документацией и сторонними библиотеками.

Сравнение синтаксиса. Синтаксисы на один взгляд совершенно разных языков программирования имеют некоторые сходства, так как оба ориентированы на системы. Мы сравним объявление переменных, типы

данных, указатели и ссылки, управление памятью, условия, циклы, обработку ошибок.

Таблица 1

Сравнительная таблица

Критерий сравнения	**C**	**Rust**
1.Объявление переменных	int x = 10;	let x =10; let mut y = 20;

2.Типы данных	int a = 10;
	float b = 5.5f;
	double c = 3.14;
	char d = 'A';
		let a: i32 = 10;
		let b: f32 = 5.5;
		let c: f64 = 3.14;
		let d: char = 'A';

3.Указатели и ссылки	int x = 10; int* p = &x;	let x = 10; let p = &x;

5.Условие	if (x > 10) {
	printf("X больше 10");
	} else {
	printf("X меньше или равно 10");
	}
		if x > 16 {
		println!("X больше 16");
		} else {
		println!("X меньше или равно 16");
		}

6.Циклы	for (int i = 0; i < 10; i++) {
	printf("%d", i);
	}
		for i in 0..10 {
		println!("{}", i);
		}

7.Обработка ошибок	if (некоторое ошибочное условие) {
	return -1; }
		fn divide(a: i32, b: i32) -> Result<i32, String> {
		if b == 0 {
		Err("Division by zero".to_string())
		} else {
		Ok(a / b)
		}
		}

В языке C все переменные по умолчанию изменяемы, однако в Rust, для того чтобы сделать переменную изменяемой нужно добавить ключевое слово mut.

В Rust данные типов явно отображаются через: (например, i32, f32), в то время как в C данные типов отображаются при объявлении переменных. Кроме того, в Rust для чисел с плавающей запятой по умолчанию используется f64, а для целых чисел — i32, но при необходимости они могут быть изменены на другие типы [6].

В указателях C — это явный элемент языка, и для работы с ними используется *и &. В Rust указатели представлены через ссылки (&), а управление памятью более безопасно благодаря системе заимствований.

Условие if схоже на двух языках, однако в Rust часто используются макросы println! для вывода на экран.

В Rust цикл for работает иначе, чем в C. Здесь используется диапазон 0 ... 10, который автоматически приводит к последовательностям чисел от 0 до 9. В C же необходимо явно вести начальное значение, условие и приращение.

В Rust для обработки ошибок используется тип Result или Option, который позволяет безопасно и наглядно обрабатывать ошибки, в отличие от C, где ошибки часто обрабатываются посредством возврата кода.

Производительность

Когда речь идет о сравнении языков программирования C и Rust, важно учитывать два ключевых аспекта: производительность и безопасность. Оба языка имеют свои преимущества и особенности, которые влияют на их использование в различных контекстах. Сейчас мы подробно рассмотрим только первый аспект [7].

Выбирая между C и Rust, разработчики часто оказываются перед компромиссом: скорость или безопасность? Оба языка мощные, но подходы у них разные. Давайте разберёмся, где каждый из них силён.

C — это классика, проверенная временем. Если нужна максимальная производительность и полный контроль над железом, C вне конкуренции. Он

даёт ручное управление памятью через указатели, минимум абстракций и почти нулевые накладные расходы. Именно поэтому на C пишут ядра ОС, драйверы, высоконагруженные сервисы вроде Redis или Nginx, и даже софт для микроконтроллеров [8].

Но за эту свободу приходится платить: C не защищает от ошибок. Переполнение буфера, висячие указатели, утечки памяти — всё это классические «болезни» C-программ. Да, опытный разработчик может писать безопасный код, но это требует дисциплины и тщательного тестирования. Более подробно мы рассмотрим это дальше.

Rust — ориентирован на высокую производительность, но с дополнительными механизмами для управления безопасностью. Его главный козырь — гарантии безопасности на этапе компиляции. При этом Rust не жертвует производительностью. Компилятор агрессивно оптимизирует код, и во многих бенчмарках Rust показывает результаты на уровне C, а иногда даже лучше. Например, Firefox (где Rust используется в движке Servo) демонстрирует, что безопасность не всегда означает замедление.

Языки программирования – это инструменты, а значит выбор между ними зависит от задач. Дальше мы проанализируем разницы в производительности используя открытые источники [9].

Согласно данным, в тестах, таких как числовые алгоритмы и обработка строк, C может быть быстрее, тогда как в задачах, связанных с бинарными деревьями, Rust показывает себя лучше, как указано в статье опубликованной 14 июня 2021 года, с обновлениями до 2025 года [5,6].

Исследование, проведённое 6 апреля 2025 года, подтверждает, что производительность зависит от оптимизаций компилятора и используемых библиотек[7]. Например, Rust может быть предпочтительнее в сценариях, требующих высокой безопасности памяти без сборщика мусора, благодаря своей системе владения и заимствования, что минимизирует накладные расходы на время выполнения. В таблице ниже представлены примеры сравнения производительности для типичных задач:

Таблица 2

Сравнение производительности

Тест	**Время C (с)**	**Время Rust (с)**
mandelbrot	1.0	1.05
binarytrees	2.0	1.8
http-server	0.5	0.4
json-serde	0.3	0.25
knucleotide	1.2	1.1
nbody	0.8	0.85
spectral-norm	1.5	1.4

Результат: В большинстве тестов Rust демонстрирует производительность, близкую к C или немного превосходящую её. Это указывает на то, что Rust способен конкурировать с C по скорости выполнения, а в некоторых случаях даже опережать его благодаря современным оптимизациям и особенностям языка.

Оба языка демонстрируют высокую производительность, и выбор между C и Rust зависит от конкретных потребностей проекта. Если приоритетом является скорость выполнения в вычислительных задачах, C может иметь небольшое преимущество. Однако Rust часто оказывается предпочтительнее благодаря комбинации производительности, безопасности и удобства разработки. Для точного сравнения рекомендую обратиться к актуальным данным на сайте и выбрать тесты, наиболее соответствующие вашим задачам.

Безопасность

Языки программирования C и Rust имеют принципиально разные подходы к безопасности, что особенно заметно в таких аспектах, как управление памятью, работа с указателями, многопоточность и проверка типов. Эти различия делают Rust более безопасным выбором для многих задач, тогда как C предоставляет больше контроля, но с повышенным риском ошибок. Рассмотрим ключевые отличия подробнее [2].

Управление памятью

В языке C управление памятью полностью лежит на разработчике: для выделения и освобождения памяти используются функции malloc и free. Это обеспечивает гибкость, но приводит к уязвимостям, таким как утечки памяти, повторное освобождение (double free) или использование памяти после её освобождения (use-after-free). Такие ошибки могут стать причиной сбоев программы или даже эксплойтов, требуя от программиста высокой дисциплины и тщательной отладки.

Rust, напротив, использует систему владения (ownership) и заимствования (borrowing), которая проверяется на этапе компиляции. Память автоматически освобождается, когда владелец ресурса выходит из области видимости, что устраняет необходимость ручного управления и сборщика мусора. Ошибки вроде use-after-free в Rust невозможны, так как компилятор не допустит некорректного доступа к памяти, снижая риск уязвимостей при сохранении высокой производительности [5,8].

Работа с указателями

В C указатели — мощный, но опасный инструмент. Они позволяют напрямую работать с памятью, однако ошибки, такие как разыменование нулевых указателей или выход за границы массива (buffer overflow), часто приводят к краху программы или становятся мишенью для атак. Эти проблемы требуют внимательного контроля со стороны разработчика [3].

Rust исключает классические нулевые указатели, заменяя их типами Option и Result, которые заставляют явно обрабатывать случаи отсутствия значения. Выход за границы массива также предотвращается: компилятор и проверки во время выполнения вызывают панику (panic) вместо неопределённого поведения, что делает такие ошибки менее разрушительными и более предсказуемыми.

Многопоточность

C не предоставляет встроенных средств для обеспечения безопасности в многопоточных программах. Разработчик сам управляет синхронизацией с помощью мьютексов и семафоров, что может привести к сложным ошибкам,

таким как data race (гонка данных) или deadlock (взаимная блокировка). Эти проблемы часто проявляются только на этапе тестирования.

Rust, напротив, гарантирует безопасность многопоточности на уровне языка. Его система типов не позволяет нескольким потокам одновременно изменять данные без явной синхронизации (например, через Mutex или RwLock). Это исключает data race на этапе компиляции, значительно упрощая создание надёжных многопоточных приложений [7,9].

Проверка типов

Статическая типизация в C относительно слабая: указатель можно привести к любому типу без строгих проверок, что может вызвать неопределённое поведение и трудно отслеживаемые ошибки.

Rust использует строгую систему типов, которая предотвращает некорректные преобразования. Система владения и времени жизни (lifetimes) дополнительно гарантирует, что ссылки всегда указывают на валидные данные, исключая ошибки, связанные с некорректным доступом к памяти. Это делает код более надёжным и предсказуемым.

Реальные примеры и статистика

Практическая разница в безопасности между C и Rust подтверждается статистикой. Например, Microsoft сообщает, что около 70% уязвимостей в их продуктах связаны с ошибками управления памятью, типичными для C и C++. В проекте Chromium использование Rust для новых компонентов сократило число уязвимостей, связанных с памятью, на 50% по сравнению с аналогичными частями на C++. Это показывает, что подход Rust эффективен на практике [1,6].

Сравнение экосистем

Экосистема языка программирования включает сообщество, доступные библиотеки, документацию и инструменты для разработки. Для C, языка с богатой историей, экосистема зрелая и обширная, охватывая области от операционных систем до встраиваемых систем. Исследования показывают, что C имеет множество библиотек, многие из которых входят в стандартную библиотеку или доступны через системные менеджеры пакетов, такие как apt или

yum. Сообщество C активно, с форумами вроде Stack Overflow и конференциями, такими как CppCon, что обеспечивает поддержку для разработчиков. Однако управление зависимостями может быть сложным из-за отсутствия единого стандарта.

Rust, напротив, имеет растущую экосистему, особенно популярную среди тех, кто интересуется системным программированием с акцентом на безопасность и производительность. Сообщество Rust известно своей инклюзивностью, а документация, включая официальную "Rust Book" [4], высоко ценится. Библиотеки в Rust представлены в виде "crates", и их легко найти через cargo, что делает экосистему более унифицированной и удобной для работы с зависимостями.

Менеджеры пакетов

Менеджер пакетов играет ключевую роль в управлении зависимостями и сборке проектов. Для C нет стандартного менеджера пакетов, что делает процесс управления зависимостями менее централизованным. Разработчики часто используют системные менеджеры, такие как apt или yum, или третьи стороны, такие как CMake [5]. Эти инструменты позволяют устанавливать библиотеки, но их использование может варьироваться в зависимости от платформы и проекта, что усложняет совместную работу.

В Rust ситуация иная: стандартным менеджером пакетов является cargo, который интегрирован в экосистему языка. Cargo не только управляет зависимостями, но и выполняет сборку, тестирование и публикацию пакетов (crates). Это упрощает процесс добавления библиотек, например, через указание зависимостей в файле cargo.toml, и обеспечивает единообразие. Исследования показывают, что большинство проектов Rust используют cargo, что делает его неотъемлемой частью разработки [9].

Заключение

В данной работы мы сравнили классический языка программирования C и современный - Rust, уделяя внимание их историческому развитию, особенностям синтаксиса, показателям производительности и аспектам безопасности. Оба

языка обладают своими уникальными преимуществами, делающими их востребованными в схожих областях разработки, однако их отличия иллюстрируют эволюцию подходов к разработке программного обеспечения.

Язык C остаётся эталоном для традиционных языков, благодаря своей простоте, высокой скорости работы и переносимости. Однако, отсутствие автоматического управления памятью и встроенных механизмов защиты делает C уязвимым для таких ошибок, как утечка памяти и переполнение буфера, что требует от разработчиков высокой степени ответственности и дисциплинированности. Rust же представляет современный подход, совмещающий производительность и безопасность. Его система владения и заимствования исключает множество традиционных уязвимостей, таких как use-after-free и гонки данных, что подтверждается анализом, проведённым выше. При этом тесты производительности демонстрируют, что Rust не уступает C, а иногда и превосходит его благодаря оптимизациям компилятора.

Список источников:

1. Аль-Нами Б.А., Баскова А.А., Васенькова А.А. Автоматизированные информационные технологии // Актуальные проблемы инфотелекоммуникаций в науке и образовании (АПИНО 2023). Сборник научных статей XII Международной научно-технической и научно-методической конференции. В 4-х томах. Под ред. С.И. Макаренко; сост. В.С. Елагин, Е.А. Аникевич. Санкт-Петербург. 2023. С. 50-52.

2. Аль-Нами Б.А., Губин Ю.М. Вредоносное программное обеспечение // Актуальные проблемы инфотелекоммуникаций в науке и образовании (АПИНО 2023). Сборник научных статей XII Международной научно-технической и научно-методической конференции. В 4-х томах. Под ред. С.И. Макаренко; сост. В.С. Елагин, Е.А. Аникевич. Санкт-Петербург. 2023. С. 117-120.

3. Аль-Нами Б.А., Загретдинов А.Э. Анализ методов шифрования // Актуальные проблемы инфотелекоммуникаций в науке и образовании (АПИНО

2023). Сборник научных статей XII Международной научно-технической и научно-методической конференции. В 4-х томах. Под ред. С.И. Макаренко; сост. В.С. Елагин, Е.А. Аникевич. Санкт-Петербург. 2023. С. 133-135.

4. Аль-Нами Б.А., Каримбаев Т.Т., Меньшова И.Э. Сравнение эффективности различных методов оптимизации // Актуальные проблемы инфотелекоммуникаций в науке и образовании (АПИНО 2023). Сборник научных статей XII Международной научно-технической и научно-методической конференции. В 4-х томах. Под ред. С.И. Макаренко; сост. В.С. Елагин, Е.А. Аникевич. Санкт-Петербург. 2023. С. 149-154.

5. Erolin J. Rust Vs C++ Performance: When Speed Matters // BairesDev – URL: https://www.bairesdev.com/blog/when-speed-matters-comparing-rust-and-c/ (дата обращения: 21.04.2025).

6. Официальный сайт языка Rust – URL: https://www.rust-lang.org/ (дата обращения: 21.04.2025).

7. CMake: Мощная система сборки программного обеспечения // CMake – URL: https://cmake.org/ (дата обращения: 21.04.2025).

8. Клабник С., Николс К., Язык программирования Rust: учебное пособие // Пер.: Н.Н.Кузнецов; ред.: Н.Н.Кузнецов, М.: Символ-Плюс. 2023. 560с.

9. C VS Rust benchmarks // Benchmarks – URL: https://programming-language-benchmarks.vercel.app/c-vs-rust (дата обращения: 21.04.2025).

UDC 004.42

COMPARATIVE ANALYSIS: CLASSICAL AND MODERN PROGRAMMING LANGUAGES

Basher Ali Al-Nami
candidate of technical sciences, associate professor
alnomibasheer@gmail.com
Diana O. Byga
student
diana20bg06@gmail.com
Vladislav V. Vasiliev
student
vladislavvasilev53@gmail.com
The Bonch-Bruevich Saint Petersburg State University of Telecommunications
Saint-Petersburg, Russia

Annotation. In this paper, we will compare two programming languages, the first is traditional C, and the other is modern - Rust. We will compare them using the sources we have analyzed. Let's evaluate the advantages and tell you about the disadvantages.

Keywords: C, Rust, programming, syntax, security, performance, compilation.

Статья поступила в редакцию 10.05.2025; одобрена после рецензирования 20.06.2025; принята к публикации 30.06.2025.
The article was submitted 10.05.2025; approved after reviewing 20.06.2025; accepted for publication 30.06.2025.