НИЖЕГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ
ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
им. Р. Е. Алексеева

Сравнительный анализ структур и классов в C#: производительность, память и целесообразность использования

Жучков Матвей Дмитриевич

Научный руководитель:

доц., к.т.н. О.В. Андреева

_____

(подпись)

Курсовая работа

Нижний Новгород, 2025

Федеральное государственное бюджетное образовательное учреждение высшего образования «Нижегородский государственный технический университет им. Р. Е. Алексеева»

Учебно-научный институт Радиоэлектроники и информационных технологий

Кафедра «Вычислительные системы и технологии»

АННОТАЦИЯ

Эта работа посвящена сравнительному анализу структур и классов в языке программирования C# с целью оценки их влияния на производительность и использование памяти. На основе теоретического анализа и практических экспериментов (копирование, передачу в методы и доступ к массивам) было определено, от чего зависит эффективность каждого типа данных.

Результаты показывают, что оптимальный выбор зависит от задачи: структуры хорошо подходят для массивов и небольших данных, так как они лежат близко друг к другу, и сборщику мусора приходится меньше работать, тогда как классы лучше подходят для крупных объектов, поскольку они более гибкие в работе (благодаря ссылкам и возможностям ООП).

Это исследование предоставляет разработчикам практические рекомендации по выбору между структурами и классами для эффективной оптимизации их приложений.

Ключевые слова: C#; структуры; классы; производительность; использование памяти; оптимизация; управление памятью; сборщик мусора.

ABSTRACT

This work presents a comparative analysis of structs and classes in C# to evaluate their impact on performance and memory utilization. Based on theoretical analysis and practical experiments (including copying operations, method parameter passing, and array access), we identify key factors influencing the efficiency of each data type.

The results demonstrate that the optimal choice is context-dependent. Structs (value types) prove effective when working with arrays and small data sets, benefiting from data locality and reduced pressure on the garbage collector. Conversely, classes (reference types) are generally preferable for larger objects, offering greater flexibility through reference semantics and Object-Oriented Programming (OOP) principles.

Ultimately, this study provides developers with practical recommendations for selecting between structs and classes to optimize C# application performance effectively.

Keywords: C#; structs; classes; performance; memory utilization; optimization; memory management; garbage collection.

СОДЕРЖАНИЕ

ПЕРЕЧЕНЬ СОКРАЩЕНИЙ И УСЛОВНЫХ ОБОЗНАЧЕНИЙ

API – Application Programming Interface – Интерфейс прикладного программирования

GC – Garbage Collector – Сборщик мусора

JIT – Just-In-Time – Компиляция "на лету" (во время выполнения программы)

LOH – Large Object Heap – Куча больших объектов (область памяти для крупных объектов в .NET)

OOP – Object-Oriented Programming – Объектно-ориентированное программирование

мкс – микросекунда (10^{-6} секунды)

нс – наносекунда (10^{-9} секунды)

ВВЕДЕНИЕ

Язык программирования C# является одним из самых популярных и широко используемых инструментов для разработки программного обеспечения, в частности, для создания Windows-приложений, веб-разработки, работы с базами данных. Структуры и классы в C# — это основные элементы для создания приложений, и они являются фундаментом объектно-ориентированного программирования. Между этими двумя типами данных есть несколько важных различий, которые влияют на то, как они ведут себя в памяти и как передаются.

Разбираться в различиях между структурами и классами важно, чтобы оптимизировать работу программ и повысить их производительность. Несмотря на схожесть этих типов данных, они могут сильно по-разному влиять на производительность и использование памяти, и это нужно учитывать при разработке приложений. Сейчас много внимания уделяют тому, насколько эффективно используются ресурсы, поэтому выбор между структурой и классом становится важным шагом при создании высокопроизводительных приложений.

Хотя теоретические различия между структурами и классами в целом понятны, на практике часто возникает вопрос: когда лучше использовать структуры, а когда — классы? Ответ здесь зависит от конкретных задач и того, какие требования есть к производительности и работе с памятью.

Цель этой работы — сравнить структуры и классы в C#, изучить их влияние на производительность, использование памяти и определить, когда какой тип лучше использовать в разных сценариях. Основное внимание будет уделяться практическому применению этих типов данных, выявлению их плюсов и минусов при решении реальных задач разработки.

Задачи работы — проанализировать теорию и практику использования структур и классов в C#. Сначала нужно изучить официальную документацию Microsoft и другие авторитетные источники, чтобы лучше понять различия между этими типами данных, а также определить типичные сценарии их использования. После изучения теории предстоит разработать набор тестов для оценки производительности и использования памяти при работе со структурами и классами. Эти тесты затем будут проведены для сбора данных.

После проведения тестов полученные результаты будут внимательно проанализированы, чтобы найти основные закономерности и зависимости. Собранные данные будут сравнены с результатами других исследований, чтобы проверить гипотезы. По результатам анализа будут подготовлены практические рекомендации для разработчиков по выбору между структурами и классами в разных ситуациях.

Теоретическая часть

В языке программирования C# структуры и классы являются основными типами данных, которые используются для представления и организации данных в программе. Несмотря на их внешнюю схожесть — оба типа позволяют хранить данные и определять их поведение — между ними существуют принципиальные различия. И эти различия влияют на выбор между структурой и классом в зависимости от задач, таких как управление памятью, производительность и сложность логики.

Как указано в документации Microsoft: «Тип структуры представляет собой тип значения, который может инкапсулировать данные и связанные функции» [1]. Это означает, что данные структуры хранятся непосредственно в переменной, а не в виде ссылки на объект [3]. Структуры обычно используются для представления небольших и простых объектов, которые не требуют сложной логики. Поскольку структуры хранятся в стеке (если они не являются частью класса), они обеспечивают более высокую производительность при работе с небольшими объемами данных [4]. Однако структуры не поддерживают наследование, что ограничивает их гибкость, но делает их более предсказуемыми и эффективными.

Класс, согласно документации Microsoft, — «это структура данных, которая может содержать элементы данных (константы и поля), члены функции (методы, свойства, события, индексаторы, операторы, конструкторы экземпляров, методы завершения и статические конструкторы) и вложенные типы» [2]. Они

используются для создания объектов с более сложной логикой и длительным временем жизни. В отличии от структур, данные классов хранятся в куче, а переменные содержат ссылки на эти объекты [5]. Классы поддерживают наследование, полиморфизм и другие принципы объектно-ориентированного программирования, что делает их более гибкими и мощными инструментами для моделирования сложных систем. Однако это также приводит к дополнительным накладным расходам на управление памятью (например, работа GC) и производительность [6].

Для наглядного сравнения структур и классов в C# рассмотрим их основные характеристики:

Таблица 1 — Сравнение структур и классов

Характеристика	**Структуры**	**Классы**
Тип данных	Тип значения (*value type*)	Тип ссылки (*reference type*)
Место хранения	Обычно в стеке (*stack*)	Куча (*heap*)
Передача данных	По значению (копируются при передаче)	По ссылке (передаётся ссылка на объект)
Наследование	Не поддерживают	Поддерживают
Полиморфизм	Ограничен (через интерфейсы)	Поддерживается
Управление памятью	Память освобождается автоматически (при выходе из области видимости)	Память управляется GC

Конструкторы	Есть конструктор по умолчанию; можно объявлять свои с параметрами (явный без параметров - с версии *C#* 10)	Можно объявлять конструкторы с параметрами и без
Инициализация полей	Поля получают значения по умолчанию при создании	Поля нужно инициализировать в конструкторе (если нет значения по умолчанию)
Жизненный цикл	Завершается при выходе из области видимости	Завершается, когда объект удалит GC (нет ссылок)
Размер в памяти	Обычно меньше (нет дополнительных расходов на ссылку/заголовок)	Больше (данные в куче + ссылка + заголовок объекта)
Поддержка интерфейсов	Могут реализовывать интерфейсы	Могут реализовывать интерфейсы
Изменяемость	Изменяемые, но часто их делают неизменяемыми	Изменяемы, если поля не объявлены как *readonly*
Использование в коллекциях	При добавлении копируются (если коллекция для *value type*)	В коллекцию добавляется ссылка, объект можно менять
Производительность	Обычно быстрее для маленьких объектов (стек, нет нагрузки на GC)	Могут быть медленнее из-за работы с кучей и GC
Применение	Небольшие объемы данных, простые	Более сложные объекты с состоянием и поведением,

	объекты (координаты, цвета)	моделирование сущностей

Рисунок 1

Важным фактором производительности структур являются затраты на их копирование при передаче. Если структура велика (часто рекомендуют не превышать условный порог в 16 байт), эти затраты могут стать заметными. Стоит отметить, что современные версии *C#* предоставляют инструменты (*ref struct*, *Span<T>*) для более эффективного управления памятью при работе со структурами, что помогает обходить некоторые из этих ограничений [9].

Диаграмма ниже показывает, как выделяется память для структур в разных случаях:

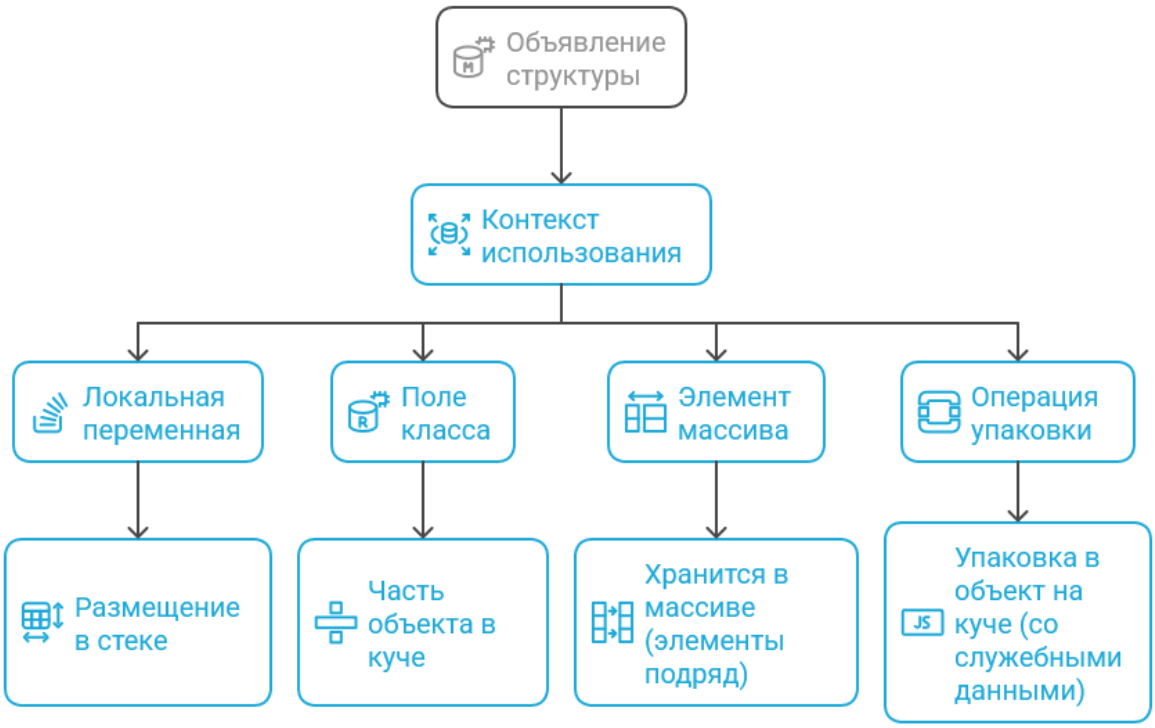

Рисунок 2: выделение памяти для структур в разных случаях

Классы поддерживают наследование и полиморфизм, что позволяет создавать на их основе сложные модели данных. Однако

объекты классов размещаются в куче (динамической памяти), что связано с дополнительными расходами на выделение памяти и работу GC. С одной стороны, это упрощает управление памятью для разработчика, но с другой — работа GC может вызывать небольшие «паузы» в работе приложения, что бывает критично для систем с высокими требованиями к производительности.

На следующей диаграмме показано типичное расположение в памяти для объекта (экземпляра) класса C#:

Рисунок 3: структура памяти экземпляра класса

Таблица 2 — Основные отличия структур и классов (память и производительность)

Характеристика	**Класс**	**Структура**
Место выделения памяти	Куча (h*eap*), иногда *Large Object Heap (LOH)*	В основном стек (*stack*), или как часть объекта/массива в куче
Дополнительные затраты памяти	Есть: заголовок объекта, указатель типа (примерно 12-24 байта)	Нет; размер = сумме размеров полей
Значение *null*	Может иметь значение *null*	Не может быть *null* (кроме обертки *Nullable<T>*)
При копировании	Копируется только ссылка (быстро)	Копируются все данные (может быть медленно для больших структур) [7]
Локальность в кэше	Хуже (объекты могут быть разбросаны по куче)	Хорошая (особенно в массивах, данные лежат рядом) [8]

Рисунок 4

Как видно из таблицы, классы требуют больше памяти из-за дополнительных расходов, в то время как структуры компактнее. Структуры могут дать выигрыш в скорости, особенно в ситуациях с хорошей локальностью данных в кэше и редким копированием.

Таким образом, из теоретического сравнения следует: не существует универсально лучшего типа между структурами и классами в C#. Выбор зависит от конкретных требований задачи. Если в приоритете минимальные расходы памяти и максимальное

быстродействие (особенно для небольших данных), стоит выбрать структуры. Если же нужна гибкость объектной модели, наследование и работа со сложными сущностями, правильным выбором будут классы.

Экспериментальная часть

В экспериментальной части работы сравнивается производительность и использование памяти структурами и классами в C#. Для точных измерений производительности нужен надежный инструмент. Для этого хорошо подходит библиотека ***BenchmarkDotNet*** [10].

BenchmarkDotNet — это популярная .NET библиотека для создания и запуска точных и надежных тестов производительности (бенчмарков). Библиотека имеет удобный API, который позволяет определять тесты, настраивать их запуск и собирать детальные метрики: время выполнения, использованную память, число выделений памяти (аллокаций) и другие.

Ключевые особенности BenchmarkDotNet:

- точность и надежность, достигаемые за счет использования продвинутых техник для минимизации влияния внешних факторов на результаты измерений (таких как «прогрев», JIT-компиляции и сборка мусора);
- гибкость и настраиваемость, позволяющие конфигурировать различные параметры выполнения тестов (например, количество

итераций, режимы компиляции, целевые платформы [x86, x64], версии среды выполнения .NET);

- генерация подробных отчетов о результатах тестирования, содержащих статистические данные, графики и диаграммы, которые способствуют анализу и сравнению производительности различных участков кода;

- поддержка измерения и анализа различных метрик производительности, включая не только время выполнения, но и такие важные показатели, как использование оперативной памяти, количество аллокаций и загрузка центрального процессора [11].

Рисунок 5: сравнение BenchmarkDotNet с другими библиотеками

BenchmarkDotNet выполняет несколько запусков тестов с последующим анализом, что позволяет уменьшить влияние случайных колебаний производительности. Библиотека автоматически компилирует код в релизной конфигурации, отключает JIT-оптимизации (при необходимости) и выполняет тесты в изолированной среде. Данные, полученные с помощью BenchmarkDotNet, важны для объективного сравнения структур и

классов. Анализ этих результатов позволит сделать выводы об эффективности каждого типа данных (с точки зрения производительности и потребления памяти) в разных сценариях использования.

Для экспериментов было подготовлено отдельное решение в среде *Visual Studio 2022*.

Эксперимент №1. Копирование структуры и класса

Цель: измерить и сравнить производительность операций копирования структур и классов различных размеров, чтобы оценить влияние типа данных на скорость копирования.

Описание: для проведения эксперимента будут созданы структура и класс с различным количеством полей (3, 10, 50 полей типа *int*). Будут реализованы и протестированы два типа копирования: поверхностное (при котором копируются только значения полей) и глубокое (копируются все связные объекты). Для каждого варианта будет проведен тест на время выполнения операций копирования. Полученные результаты будут сведены в таблицу и сопоставлены с ожидаемыми результатами.

Ожидаемые результаты: структуры будут копироваться быстрее классов, поскольку они передаются по значению без необходимости выделения дополнительной памяти. Поверхностное копирование классов, несмотря на работу с кучей, будет быстрее глубокого, которое потребует создания новых экземпляров вложенных объектов и увеличит нагрузку на память.

Код для этого эксперимента находится в сборке под названием «*Copying the Structure and Class*». В ней созданы отдельные реализации для структур и для классов с запланированным числом полей. Каждая реализация содержит конструктор, а также методы для выполнения поверхностного и глубокого копирования.

Также для тестирования и анализа производительности методов копирования используется вспомогательный класс «*CopyingBenchmarkEvaluator*» (в сборке «*BenchmarkLab*»).

Эксперимент №2. Передача структуры и класса в метод (с разными типами данных)

Цель: оценить, как механизм передачи данных (по значению для структур, по ссылке для классов) влияет на производительность методов, принимающих эти типы.

Описание: для эксперимента реализованы два метода: один принимает структуру, другой — класс. Используемые структура и класс содержат одинаковый набор полей: одно поле типа *int*, одно поле *string* и массив *byte[1024]*. В ходе теста методы будут вызываться многократно в цикле для того, чтобы измерить общее время выполнения. Чтобы избежать влияния кэша, в каждой итерации будут использоваться разные (случайные) входные данные.

Ожидаемые результаты: передача класса (по ссылке) будет происходить быстрее для больших структур, однако для небольших структур разница в производительности может оказаться незначительной.

Код для этого эксперимента находится в сборке «*Transfer of structure and class*». Она содержит два основных файла-класса: «*DataTypes*» (здесь определены структуры и классы для эксперимента с нужным числом полей) и «*ProcessingMethods*» (в нем реализованы методы, принимающие эти структуры и классы; методы также включают доступ к полям объектов для имитации их использования после передачи).

Также для тестирования и анализа производительности методов передачи используется вспомогательный класс «*TransferBenchmarkEvaluator*» (в сборке «*BenchmarkLab*»).

Эксперимент №3. Сравнение скорости доступа к массивам структур и классов

Цель: измерить и сравнить время доступа к элементам в массивах структур и массивах классов при разном порядке обхода (последовательном и случайном).

Описание: в этом эксперименте будет сравниваться скорость доступа к элементам двух больших массивов: один состоит из структур *PointStruct*, другой — из объектов класса *PointClass*. Структура и класс будут иметь одинаковые поля (*int X*, *int Y*).

Массивы будут заполняться случайными данными (для того, чтобы предотвратить оптимизацию компилятором). При создании массива классов каждый объект инициализируется отдельно через *new*.

Будут протестированы два варианта доступа к элементам (для обоих массивов):

- **последовательный доступ:** массив обходится по порядку индексов (0, 1, 2…), на каждом шаге считывается значение поля *X*;

- **случайный доступ:** используется заранее подготовленный массив случайных индексов. Доступ к элементам основного массива происходит по этим случайным индексам; также считывается значение поля.

Ожидаемые результаты: последовательный доступ к элементам массива будет быстрее случайного из-за лучшего использования кэша. При последовательном доступе структуры окажутся быстрее классов. Это связано с тем, что данные структур в массиве расположены подряд (хорошая локальность кэша) и доступ к ним прямой, без перехода по ссылке. При случайном доступе преимущество структур будет еще заметнее. Доступ к объектам классов потребует перехода по ссылке, что при случайных обращениях к памяти увеличивает вероятность промахов кэша. Крупные структуры могут потерять преимущество — .NET может размещать их в куче.

Код для этого эксперимента находится в сборке «*Comparison of structure and class speed*». Она содержит два основных файла-класса: «*DataModels*» (здесь только определение структуры и класса) и «*ArrayAccessExperimentCode*» (в нем содержится логика и данные для эксперимента).

Также для тестирования и анализа производительности методов передачи используется вспомогательный класс «*ComparisonBenchmarkEvaluator*» (в сборке «*BenchmarkLab*»).

Результаты и обсуждения

Эксперимент №1

Рассмотрим результаты эксперимента по копированию структур и классов с разным числом полей (3, 10 и 50). Ключевые метрики производительности и использования памяти представлены в таблице ниже (Таблица 3; полные данные измерений *см. в Приложении А*)

Таблица 3 — Результаты тестов копирования (Эксперимент №1)

Тип данных	**Количество полей**	**Тип копирования**	**Среднее время (нс)**	**Стандартное отклонение (нс)**	**Память (байт)**
Структура	3	Поверхностное	0.2304	0.1436	—
Структура	3	Глубокое	0.1818	0.1067	—
Структура	10	Поверхностное	8.3270	0.1619	—
Структура	10	Глубокое	10.7740	0.1897	—
Структура	50	Поверхностное	16.9653	0.5873	—
Структура	50	Глубокое	35.1520	1.3592	—
Класс	3	Поверхностное	0.1582	0.2805	—
Класс	3	Глубокое	2.1114	0.4924	33
Класс	10	Поверхностное	0.0878	0.1596	—
Класс	10	Глубокое	3.5093	0.6854	66
Класс	50	Поверхностное	0.1220	0.1544	—
Класс	50	Глубокое	24.6280	2.6618	229

Рисунок 6

Проанализировав результаты эксперимента №1, можно сделать следующие выводы:

Объекты с 3 полями:

- **поверхностное копирование:** структуры копировались дольше классов — в среднем 0.2304 нс против 0.1582 нс (~ 1.5 раза медленнее);
- **глубокое копирование:** структуры показали значительно лучшее время — 0.1818 нс против 2.1114 нс у классов (~ 11.6 раза быстрее);
- **сравнение внутри структур:** время поверхностного и глубокого копирования почти не отличалось (разница < 0.05 нс);
- **сравнение внутри классов:** поверхностное копирование было значительно быстрее глубокого (~ 13 раз);
- **память:** структуры — 0 байт, классы — 33 байта.

Объекты с 10 полями:

- **поверхностное копирование:** классы показали себя значительно быстрее структур — 0.0878 нс против 8.3270 нс (~ 95 раз);
- **глубокое копирование:** классы также были быстрее, но с меньшим отрывом — 3.5093 нс против 10.7740 нс у структур (~ 3 раза);
- **сравнение внутри структур:** глубокое копирование выполнялось медленнее поверхностного (~ 1.3 раза);
- **сравнение внутри классов:** поверхностное копирование осталось значительно быстрее глубокого (~ 40 раз);
- **память:** структуры — 0 байт, классы — 66 байт.

Объекты с 50 полями:

- **поверхностное копирование:** структуры выполняли копирование значительно медленнее классов — 16.9653 нс против 0.1220 нс (~ 139 раз);

- **глубокое копирование:** структуры показали лучшее время (24.6280 нс), оказавшись примерно в 1.4 раза быстрее классов (35.1520 нс);

- **сравнение внутри структур:** глубокое копирование заняло примерно в 2 раза больше времени, чем поверхностное;

- **сравнение внутри классов:** разница между поверхностным и глубоким копированием увеличилась — поверхностное было быстрее примерно в 200 раз;

- **память:** структуры — 0 байт, классы — 229 байт.

Проведенный эксперимент показывает, что производительность копирования зависит не только от типа данных (структура или класс), но также от числа полей и типа выполняемого копирования (поверхностное или глубокое).

Как и ожидалось, в некоторых случаях (например, при 3 полях и глубоком копировании) структуры продемонстрировали лучшую производительность. Однако с ростом числа полей и, особенно, при поверхностном копировании преимущество часто переходило к классам. По-видимому, незначительная стоимость копирования самой ссылки класса дает ему преимущество перед структурами, копирование которых становится все более ресурсоемким по мере добавления полей.

В простых случаях (при 3 полях и глубоком копировании) структуры действительно работали быстрее, как и ожидалось. Однако по мере усложнения объектов классы начали показывать лучшие результаты, особенно при поверхностном копировании.

Также подтвердилась разница в использовании памяти: классы всегда требовали выделения памяти в управляемой куче, в отличие от структур, которые не показывали выделения памяти в куче в проведенных тестах.

Обсуждение аномалий в результатах тестов

Результаты эксперимента выявили несколько интересных моментов, требующих обсуждения.

Во-первых, наблюдалась ситуация, которая отличалась от изначальных ожиданий для больших объектов. Для объектов с 3 полями структуры (при глубоком копировании) действительно показали лучшую производительность. Но для объектов с 10 и 50 полями классы часто оказывались быстрее, особенно при поверхностном копировании. Можно предположить, что внутренние механизмы копирования данных в среде .NET для объектов-классов могут быть оптимизированы эффективнее, чем простое побайтное копирование структур. При больших объемах данных выигрыш от таких оптимизаций, вероятно, перевешивает расходы на работу с управляемой кучей.

Во-вторых, была замечена повышенная вариативность (большее стандартное отклонение) времени выполнения при глубоком копировании классов, особенно с 50 полями. Такая

нестабильность результатов, вероятнее всего, связана со спецификой выделения памяти в управляемой куче и природой работы GC. Процесс выделения памяти под новые объекты и возможная активность GC вносят случайный фактор, отсутствующий при копировании структур, которые чаще размещаются в стеке.

Эксперимент №2

Рассмотрим результаты эксперимента по передаче в метод структур и классов для объектов разного размера (малого, среднего, крупного). Ключевые метрики производительности представлены в таблице ниже (Таблица 4; полные данные измерений *см. в Приложении Б*).

Таблица 4 — Результаты тестов по передаче (Эксперимент №2)

Тип объекта	**Состав полей**	**Среднее время вызова (нс)**	**Стандартное отклонение (нс)**	**Память (байт)**
Структура	*int*	7.103	0.351	—
Класс	*int*	9.396	0.374	33
Структура	*int, string*	565.830	19.712	773
Класс	*int, string*	581.301	9.668	803
Структура	*int, string, byte[1024]*	2 072.769	98.617	3395
Класс	*int, string, byte[1024]*	1 989.914	83.120	3441

Рисунок 7

Проанализировав результаты эксперимента №2, можно сделать следующие выводы:

Малый объект (поле *int*):

- **Время передачи:** структура - 7.103 нс; класс - 9.396 нс (структура быстрее на ~32%);
- **Память:** структура - 0 байт; класс - 33 байта.

Средний объект (поля *int*, *string*):

- **Время передачи:** структура - 565.830 нс; класс - 581.301 нс (разница незначительна, <3%);
- **Память:** структура - 773 байта; класс - 803 байта (сопоставимо, выделяется из-за поля *string*).

Крупный объект (поля *int*, *string* и *byte[1024]*):

- **Время передачи:** класс - 1989.914 нс; структура - 2072.769 нс (класс быстрее ~на 4%);
- **Память:** структура - 3395 байт; класс - 3441 байт (сопоставимо, выделяется из-за *string* и массива *byte[]*).

Обсуждение аномалий в результатах тестов

Интересное наблюдение, касаемое стабильности времени выполнения при передаче объектов среднего размера. Несмотря на близкие средние значения времени, стандартное отклонение для передачи структуры оказалось заметно выше (19.712 нс), чем для класса (9.668 нс).

Такой больший разброс времени для структур можно объяснить самим процессом копирования по значению. При передаче структуры копируются все её поля, и время этой операции может сильнее зависеть от текущего состояния кэша процессора и

шины памяти, чем копирование единственной ссылки на объект класса. Поэтому стабильность времени при передаче структуры ниже по сравнению с передачей класса.

Эксперимент 3

Рассмотрим результаты эксперимента по скорости доступа к элементам массивов структур и классов для размеров 10 000 и 100 000 элементов. Ключевые метрики производительности представлены в таблице ниже (Таблица 5; полные данные измерений *см. в Приложении В*).

Таблица 5 — Результаты тестов скорости доступа (Эксперимент №3)

Тип объекта	**Паттерн доступа**	**Размер массива (кол-во элементов)**	**Среднее время (мкс)**	**Стандартное отклонение (мкс)**	**Память (байт)**
Структура	Последовательный	10 000	4.7450	0.2030	—
Структура	Случайный	10 000	5.5690	0.3335	—
Класс	Последовательный	10 000	5.1050	0.1644	—
Класс	Случайный	10 000	6.3990	0.2533	—
Структура	Последовательный	100 000	47.2760	0.5560	—
Структура	Случайный	100 000	56.1780	0.9763	—
Класс	Последовательный	100 000	67.4810	1.6526	—
Класс	Случайный	100 000	178.5370	3.0347	—

Рисунок 8

Проанализировав результаты эксперимента №3, можно сделать следующие выводы:

При последовательном доступе структуры показали стабильное преимущество в скорости над классами. Для массива из 10 000 элементов время составило 4.7 мкс для структур против 5.1 мкс для классов; для 100 000 элементов — 47.3 мкс против 67.5 мкс соответственно. Причина этого преимущества — в непрерывном размещении данных структур в массиве, что положительно сказывается на локальности кэша. При увеличении размера массива в 10 раз время последовательного доступа для обоих типов также увеличивалось примерно в 10 раз (линейная зависимость).

Картина при случайном доступе была иной. Структуры по-прежнему оставались быстрее классов: 5.6 мкс против 6.4 мкс для 10 000 элементов и 56.2 мкс против 178.5 мкс для 100 000 элементов. Однако если для структур рост времени при увеличении размера массива оставался линейным (~10 раз), то для классов он был значительно более резким — почти в 28 раз. Такое сильное падение производительности классов при случайном доступе, вероятно, объясняется необходимостью перехода по ссылке к каждому объекту, что в условиях непредсказуемых обращений к памяти часто приводит к промахам кэша.

Как и ожидалось, последовательный доступ к элементам массива был стабильно быстрее случайного (независимо от типа элемента и размера массива) благодаря лучшей предсказуемости обращений для кэша процессора.

Обсуждение особенностей в результатах тестов

В результатах эксперимента заслуживает внимания один специфический момент. При размере массива в 10 000 элементов случайный доступ к структурам занял немного больше времени (5.569 мкс), чем последовательный доступ к классам (5.105 мкс), хотя обычно ожидается обратное.

Возможное объяснение этого в том, что накладные расходы на получение случайного индекса из вспомогательного массива (для организации случайного доступа) в данном конкретном случае превысили выгоду от прямого доступа к структуре в памяти. Иными словами, стоимость вычисления индекса оказалась чуть выше, чем стоимость перехода по ссылке для класса в условиях очень эффективного кэширования при последовательном доступе.

Важно отметить, что это наблюдалось только для массива меньшего размера (10 000 элементов). При увеличении массива до 100 000 элементов соотношение уже соответствовало общим ожиданиям: случайный доступ к структурам был быстрее, чем последовательный доступ к классам.

ЗАКЛЮЧЕНИЕ

Эта работа посвящена анализу проблемы выбора между структурами и классами в C#, обусловленной ростом требований к производительности современных приложений и нехваткой четких рекомендаций по эффективному использованию памяти для обычных операций, принимая во внимание большое разнообразие самих задач и данных. Получены следующие результаты:

1. Удалось получить сравнительные данные производительности для структур и классов применительно к объектам с разным количеством полей через стандартные тесты (библиотека BenchmarkDotNet) [10, 11];
2. Измерены различия в скорости: подтверждено, что копирование структуры замедляется с ростом числа полей [7], в противовес почти постоянному времени копирования ссылок класса; также установлено преимущество структуры при последовательной обработке массивов [8];
3. Выделены сценарии, где структура выглядит предпочтительнее (работа с массивами [8], частые операции с небольшими объектами для экономии памяти [4, 7] и уменьшения пауз GC [6]) и где лучше подходит класс (сложная логика с наследованием [2], передача крупных объектов [5], необходимость изменять объект после создания);
4. Анализ вскрыл и некоторые особенности: оказалось, классы могут опережать структуры при копировании больших

объемов (эффект оптимизации .NET), а стабильность времени их работы ниже из-за влияния управляемой памяти и GC [6].

Полученные результаты помогут разработчикам делать более обоснованный выбор между структурами и классами при разработке приложений, способствуя улучшению общей производительности ПО и эффективности работы с памятью в среде .NET.

СПИСОК ЛИТЕРАТУРЫ

1. Типы структур (справочник по C#) [Электронный ресурс] // Microsoft Learn. – URL: https://learn.microsoft.com/ru-ru/dotnet/csharp/language-reference/builtin-types/struct (дата обращения: 03.05.2025). – Текст : электронный.
2. Classes - C# language specification [Электронный ресурс] // Microsoft Learn. – URL: https://learn.microsoft.com/ru-ru/dotnet/csharp/language-reference/language-specification/classes (дата обращения: 03.05.2025). – Текст : электронный.
3. Value types - C# reference [Электронный ресурс] // Microsoft Learn. – 2024. – URL: https://learn.microsoft.com/en-us/dotnet/csharp/language-reference/builtin-types/value-types (дата обращения: 03.05.2025). – Текст : электронный.
4. Struct vs Class in C#: Choosing the Right Data Type [Электронный ресурс] // ByteHide : [сайт]. – 2023. – URL: https://www.bytehide.com/blog/struct-vs-class-csharp (дата обращения: 03.05.2025). – Текст : электронный.
5. Reference types - C# reference [Электронный ресурс] // Microsoft Learn. – 2024. – URL: https://learn.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/reference-types (дата обращения: 03.05.2025). – Текст : электронный.
6. Advanced Memory Management Techniques in C# [Электронный ресурс] // TechOnDiapers : [сайт]. – 2024. – URL: https://tech-on-diapers.hashnode.dev/advanced-memory-management-techniques-in-c (дата обращения: 03.05.2025). – Текст : электронный.
7. Beginners Guide To C# Struct vs Class (With Code Examples) [Электронный ресурс] // Zero To Mastery : [сайт]. – 2025. – URL: https://zerotomastery.io/blog/c-sharp-struct-vs-class/ (дата обращения: 03.05.2025). – Текст : электронный.
8. Structs Versus Classes In C# [Электронный ресурс] // MDFT Academy : [сайт]. – 2024. – URL: https://www.mdft.academy/blog/whats-faster-in-csharp-a-struct-or-a-class (дата обращения: 03.05.2025). – Текст : электронный.

9. ref struct types - C# reference [Электронный ресурс] // Microsoft Learn. – 2025. – URL: https://learn.microsoft.com/en-us/dotnet/csharp/language-reference/builtin-types/ref-struct (дата обращения: 03.05.2025). – Текст : электронный.
10. Getting started [Электронный ресурс] // BenchmarkDotNet : [сайт]. – URL: https://benchmarkdotnet.org/articles/guides/getting-started.html (дата обращения: 03.05.2025). – Текст : электронный.
11. How to Use BenchmarkDotNet: 6 Simple Performance-Boosting Tips to Get Started [Электронный ресурс] // CodeProject : [сайт]. – 2024. – URL: https://www.codeproject.com/Articles/5378617/How-to-Use-BenchmarkDotNet-6-Simple-Performance-Bo (дата обращения: 03.05.2025). – Текст : электронный.

ПРИЛОЖЕНИЕ А. РЕЗУЛЬТАТЫ ИЗМЕРЕНИЯ ВРЕМЕНИ ВЫПОЛНЕНИЯ ОПЕРАЦИЙ КОПИРОВАНИЯ СТРУКТУР И КЛАССОВ

```
// * Summary *

BenchmarkDotNet v0.14.0, Windows 11 (10.0.26100.3775)
12th Gen Intel Core i5-12450H, 1 CPU, 12 logical and 8 physical cores
  [Host]  : .NET Framework 4.8.1 (4.8.9300.0), X86 LegacyJIT
  LongRun : .NET Framework 4.8.1 (4.8.9300.0), X64 RyuJIT VectorSize=256

Job=LongRun  Jit=RyuJit  Platform=X64
InvocationCount=500  IterationCount=100  LaunchCount=3
UnrollFactor=1  WarmupCount=100

| Method            | Categories | Mean       | Error     | StdDev    | Median     | Rank | Allocated |
|------------------ |----------- |-----------:|----------:|----------:|-----------:|-----:|----------:|
| Class_10_Shallow  | 10_Fields  |  0.0878 ns | 0.0318 ns | 0.1596 ns |  0.0000 ns |    1 |         - |
| Class_10_Deep     | 10_Fields  |  3.5093 ns | 0.1340 ns | 0.6854 ns |  3.4000 ns |    2 |      66 B |
| Struct_10_Shallow | 10_Fields  |  8.3270 ns | 0.0319 ns | 0.1619 ns |  8.4000 ns |    3 |         - |
| Struct_10_Deep    | 10_Fields  | 10.7740 ns | 0.0376 ns | 0.1897 ns | 10.8000 ns |    4 |         - |
|                   |            |            |           |           |            |      |           |
| Class_3_Shallow   | 3_Fields   |  0.1582 ns | 0.0565 ns | 0.2805 ns |  0.0000 ns |    1 |         - |
| Struct_3_Deep     | 3_Fields   |  0.1818 ns | 0.0218 ns | 0.1067 ns |  0.2000 ns |    2 |         - |
| Struct_3_Shallow  | 3_Fields   |  0.2304 ns | 0.0277 ns | 0.1436 ns |  0.2000 ns |    2 |         - |
| Class_3_Deep      | 3_Fields   |  2.1114 ns | 0.0977 ns | 0.4924 ns |  2.0000 ns |    3 |      33 B |
|                   |            |            |           |           |            |      |           |
| Class_50_Shallow  | 50_Fields  |  0.1220 ns | 0.0311 ns | 0.1544 ns |  0.0000 ns |    1 |         - |
| Struct_50_Shallow | 50_Fields  | 16.9653 ns | 0.1180 ns | 0.5873 ns | 17.0000 ns |    2 |         - |
| Class_50_Deep     | 50_Fields  | 24.6280 ns | 0.5142 ns | 2.6618 ns | 23.7000 ns |    3 |     229 B |
| Struct_50_Deep    | 50_Fields  | 35.1520 ns | 0.2737 ns | 1.3592 ns | 35.7000 ns |    4 |         - |
```

ПРИЛОЖЕНИЕ Б. РЕЗУЛЬТАТЫ ИЗМЕРЕНИЯ ВРЕМЕНИ ВЫПОЛНЕНИЯ ОПЕРАЦИИ ПЕРЕДАЧИ В МЕТОД СТРУКТУР И КЛАССОВ

```
// * Summary *

BenchmarkDotNet v0.14.0, Windows 11 (10.0.26100.3775)
12th Gen Intel Core i5-12450H, 1 CPU, 12 logical and 8 physical cores
  [Host]  : .NET Framework 4.8.1 (4.8.9300.0), X86 LegacyJIT
  LongRun : .NET Framework 4.8.1 (4.8.9300.0), X64 RyuJIT VectorSize=256

Job=LongRun  Jit=RyuJit  Platform=X64
InvocationCount=500  IterationCount=100  LaunchCount=3
UnrollFactor=1  WarmupCount=100

| Method       | Categories | Mean         | Error      | StdDev     | Median       | Rank | Allocated |
|------------- |----------- |-------------:|-----------:|-----------:|-------------:|-----:|----------:|
| LargeClass   | Large      | 1,989.914 ns | 16.7983 ns | 83.1201 ns | 2,031.800 ns |    1 |    3441 B |
| LargeStruct  | Large      | 2,072.769 ns | 19.5646 ns | 98.6172 ns | 2,050.600 ns |    1 |    3395 B |
|              |            |              |            |            |              |      |           |
| MediumStruct | Medium     |   565.830 ns |  3.9913 ns | 19.7124 ns |   574.200 ns |    1 |     773 B |
| MediumClass  | Medium     |   581.301 ns |  1.9467 ns |  9.6689 ns |   577.400 ns |    1 |     803 B |
|              |            |              |            |            |              |      |           |
| SmallStruct  | Small      |     7.103 ns |  0.0682 ns |  0.3510 ns |     7.200 ns |    1 |         - |
| SmallClass   | Small      |     9.396 ns |  0.0748 ns |  0.3743 ns |     9.400 ns |    2 |      33 B |
```

ПРИЛОЖЕНИЕ В. РЕЗУЛЬТАТЫ СКОРОСТИ ДОСТУПА К ЭЛЕМЕНТАМ МАССИВОВ СТРУКТУР И КЛАССОВ

```
BenchmarkDotNet v0.14.0, Windows 11 (10.0.26100.3775)
12th Gen Intel Core i5-12450H, 1 CPU, 12 logical and 8 physical cores
  [Host]  : .NET Framework 4.8.1 (4.8.9300.0), X86 LegacyJIT
  LongRun : .NET Framework 4.8.1 (4.8.9300.0), X64 RyuJIT VectorSize=256

Job=LongRun  Jit=RyuJit  Platform=X64
InvocationCount=500  IterationCount=100  LaunchCount=3
UnrollFactor=1  WarmupCount=100
```

Method	Categories	ArraySize	Mean	Error	StdDev	Median	Rank	Allocated
RandomAccess_Structs	Random	10000	5.569 us	0.0664 us	0.3335 us	5.405 us	1	-
RandomAccess_Classes	Random	10000	6.399 us	0.0495 us	0.2533 us	6.287 us	2	-
RandomAccess_Structs	Random	100000	56.178 us	0.1923 us	0.9763 us	56.141 us	3	-
RandomAccess_Classes	Random	100000	178.537 us	0.5843 us	3.0347 us	178.917 us	4	-
SequentialAccess_Structs	Sequential	10000	4.745 us	0.0410 us	0.2030 us	4.671 us	1	-
SequentialAccess_Classes	Sequential	10000	5.105 us	0.0327 us	0.1644 us	5.019 us	2	-
SequentialAccess_Structs	Sequential	100000	47.276 us	0.1087 us	0.5560 us	47.268 us	3	-
SequentialAccess_Classes	Sequential	100000	67.481 us	0.3187 us	1.6526 us	67.734 us	4	-