Р Е Ф Е Р А Т

На тему:

Язык программирования **C#**

Группа ИУ5 31Б

Студент: Керимова Жанна Руслановна

2025 г.

Содержание

Введение

C# — это типизированный, объектно-ориентированный, простой и в то же время мощный язык программирования, который позволяет разработчикам создавать многофункциональные приложения. Разработан в 1998—2001 годах группой инженеров под руководством Андерса Хейлсберга в компании Microsoft как основной язык разработки приложений для платформы Microsoft .NET (программной платформы от компании Microsoft, предназначенной для создания обычных программ и вебприложений).

Символ # (октоторп) в названии языка печатается на клавиатуре как Shift+3, что символизирует третью реализацию C. С другой стороны # можно интерпретировать и как две пары плюсов ++;++, намекающие на новый шаг в развитии языка по сравнению с C++ (подобно шагу от C к C++), и как музыкальный символ диез, вместе с буквой C, составляющий в английском языке название ноты до-диез (англ. C sharp). Октоторп # часто называют «шарпом» (от англ, sharp) из-за его схожести с диезом, отсюда и название языка — «Си шарп».

C# относится к семье языков с C-подобным синтаксисом, из них его синтаксис наиболее близок к C++ и Java.

C# — это фактически гибрид разных языков. Переняв многое от своих предшественников — языков C++, Java, Delphi, Модула и Smalltalk - и опираясь на практику их использования, C# синтаксически не менее (если не более) чист, чем Java, так же прост, как Visual Basic, и обладает практически той же мощью и гибкостью, что и C++.

Актуальность: Язык программирования C# является одним из наиболее популярных за счет простоты его изучения.

1. История возникновения

Рождение C#.

Язык *C#* появился на свет в июне 2000 г., в результате кропотливой работы большой группы разработчиков компании Microsoft, возглавляемой Андерсом Хейлсбергом (Anders Hejlsberg). Этот человек известен как автор одного из первых компилируемых языков программирования для персональных компьютеров IBM -- Turbo Pascal. Наверное, на территории бывшего Советского Союза многие разработчики со стажем, да и просто люди, обучавшиеся в той или иной форме программированию в вузах, испытали на себе очарование и удобство использования этого продукта. Кроме того, во время работы в корпорации Borland Андерс Хейлсберг прославился созданием интегрированной среды Delphi (он руководил этим проектом вплоть до выхода версии 4.0).

Для новых задач - новый язык программирования!

Появление языка *C#* и инициативы *.NET* отнюдь не случайно пришлось на начало лета 2000 г. Именно к этому моменту компания Microsoft подготовила промышленные версии новых компонентных технологий и решений в области обмена сообщениями и данными, а также создания Internet-приложений (COM+, ASP+, ADO+, SOAP, Biztalk Framework). Несомненно, лучшим способом продвижения этих новинок является создание инструментария для разработчиков с их полноценной поддержкой. В этом и заключается одна из главных задач нового языка *C#*.

Кроме того, Microsoft не могла больше расширять все те же инструменты и языки разработки, делая их все более и более сложными для удовлетворения конфликтующих между собой требований поддержки

современного оборудования и обеспечения обратной совместимости с теми продуктами, которые были созданы в начале 1990-х гг. во время первого появления Windows. Наступает момент, когда необходимо начать с чистого листа для того, чтобы создать простой, но имеющий сложную структуру набор языков, сред и средств разработки, которые позволят разработчику легко создавать современные программные продукты

C# и .NET

C# и .NET и являются той самой отправной точкой. Если говорить упрощенно, то *.NET* представляет собой новую платформу, новый API для программирования в Windows, а *C#* есть новый язык, созданный с нуля, для работы с этой платформой, а также для извлечения всех выгод из прогресса сред разработки и нашего понимания принципов объектно-ориентированного программирования в течение последних 20 лет.

Необходимо отметить, что обратная совместимость не потеряна. Существующие программы будут выполняться, а платформа *.NET* была спроектирована таким образом, чтобы она могла работать с имеющимся программным обеспечением. Связь между компонентами в Windows сейчас почти целиком осуществляется при помощи COM. С учетом этого *.NET* обладает способностью создавать оболочки (wrappers) вокруг существующих компонентов COM, так что компоненты *.NET* могут свободно общаться с ними.

C# это мощь C++ и простота Visual Basic

Авторы *C#* стремились создать язык, сочетающий простоту и выразительность современных объектно-ориентированных языков (вроде Java) с богатством возможностей и мощью C++. По словам Андерса Хейлсберга, *C#* позаимствовал большинство своих синтаксических конструкций из C++. В частности, в нем присутствуют такие удобные типы данных, как структуры и перечисления (другой потомок C++ -- Java -- лишен

этих элементов, что создает определенные неудобства при программировании). Синтаксические конструкции *C#* унаследованы не только от C++, но и от Visual Basic. Например, в *C#*, как и в Visual Basic, используются свойства классов. Как C++, *C#* позволяет производить перегрузку операторов для созданных вами типов, Java не поддерживает ни ту, ни другую возможность). *C#* это фактически гибрид разных языков. При этом *C#* синтаксически не менее (если не более) чист, чем Java, и так же прост, как Visual Basic, но обладает практически той же мощью и гибкостью, что и C++.

2. Общее описание языка

Последнее время C и C++ являются наиболее используемыми языками для разработки коммерческих и бизнес приложений. Эти языки устраивают многих разработчиков, но в действительности не обеспечивают должной продуктивности разработки. К примеру, процесс написания приложения на C++ зачастую занимает значительно больше времени, чем разработка эквивалентного приложения, скажем, на Visual Basic. Сейчас существуют языки, увеличивающие продуктивность разработки за счет потери в гибкости, которая так привычна и необходима программистам на C/C++. Подобные решения являются весьма неудобными для разработчиков и зачастую предлагают значительно меньшие возможности. Эти языки также не ориентированы на взаимодействие с появляющимися сегодня системами и очень часто они не соответствуют существующей практике программирования для Web.

Многие разработчики хотели бы использовать современный язык, который позволял бы писать, читать и сопровождать программы с простотой Visual Basic и в то же время давал мощь и гибкость C++, обеспечивал доступ ко всем функциональным возможностям системы, взаимодействовал бы с

существующими программами и легко работал с возникающими Web стандартами.

Учитывая все подобные пожелания, Microsoft разработала новый язык - C#. В него входит много полезных особенностей - простота, объектная ориентированность, типовая защищенность, "сборка мусора", поддержка совместимости версий и многое другое. Данные возможности позволяют быстро и легко разрабатывать приложения, особенно COM+ приложения и Web сервисы.

При создании C#, его авторы учитывали достижения многих других языков программирования: C++, C, Java, SmallTalk, Delphi, Visual Basic и т.д. Надо заметить, что по причине того, что C# разрабатывался с чистого листа, у его авторов была возможность (которой они явно воспользовались), оставить в прошлом все неудобные и неприятные особенности (существующие, как правило,для обратной совместимости), любого из предшествующих ему языков. В результате получился действительно простой, удобный и современный язык, по мощности не уступающий C++, но существенно повышающий продуктивность разработок.

Очень часто можно проследить такую связь - чем более язык защищен и

устойчив к ошибкам, тем меньше производительность программ, написанных на нем. К примеру, рассмотрим две крайности - очевидно это Assembler и Java. В первом случае вы можете добиться фантастической быстроты своей программы, но вам придется очень долго заставлять ее работать правильно не на вашем компьютере. В случае же с Java - вы получаете защищенность, независимость от платформы, но, к сожалению, скорость вашей программы вряд ли совместима со сложившимся представлением о скорости, например, какого-либо отдельного клиентского приложения (конечно существуют оговорки - JIT компиляция и прочее). Рассмотрим C++ с этой точки зрения - на мой взгляд соотношение в скорости и защищенности близко к желаемому

результату, но на основе собственного опыта программирования я могу с уверенностью сказать, что практически всегда лучше понести незначительную потерю в производительности программы и приобрести такую удобную особенность, как "сборка мусора", которая не только освобождает вас от утомительной обязанности управлять памятью вручную, но и помогает избежать вам многих потенциальных ошибок в вашем приложении. В действительности скоро "сборка мусора", да и любые другие шаги к устранению потенциальных ошибок стану отличительными чертами современного языка. В C#, как в несомненно современном языке, также существуют характерные особенности для обхода возможных ошибок. Например, помимо упомянутой выше "сборки мусора", там все переменные автоматически инициализируются средой и обладают типовой защищенностью, что позволяет избежать неопределенных ситуаций в случае, если программист забудет инициализировать переменную в объекте или попытается произвести недопустимое преобразование типов. Также в C# были предприняты меры для исключения ошибок при обновлении программного обеспечения. Изменение кода, в такой ситуации, может непредсказуемо изменить суть самой программы. Чтобы помочь разработчикам бороться с этой проблемой C# включает в себя поддержку совместимости версий (vesioning). В частности, в отличии от C++ и Java, если метод класса был изменен, это должно быть специально оговорено. Это позволяет обойти ошибки в коде и обеспечить гибкую совместимость версий. Также новой особенностью является native поддержка интерфейсов и наследования интерфейсов. Данные возможности позволяют разрабатывать сложные системы и развивать их со временем. В C# была унифицирована система типов, теперь вы можете рассматривать каждый тип как объект. Несмотря на то, используете вы класс, структуру, массив или встроенный тип, вы можете обращаться к нему как к объекту. Объекты собраны в пространства имен (namespaces), которые позволяют программно обращаться к чему-либо. Это значит что вместо списка включаемых файлов заголовков в своей программе

вы должны написать какие пространства имен, для доступа к объектам и классам внутри них, вы хотите использовать. В C# выражение using позволяет вам не писать каждый раз название пространства имен, когда вы используете класс из него. Например, пространство имен System содержит несколько классов, в том числе и Console. И вы можете писать либо название пространства имен перед каждым обращением к классу, либо использовать using как это было показано в примере выше.

Важной и отличительной от C++ особенностью C# является его простота. К примеру, всегда ли вы помните, когда пишите на C++, где нужно использовать "- >", где "::", а где "."? Даже если нет, то компилятор всегда поправляет вас в случае ошибки. Это говорит лишь о том, что в действительности можно обойтись только одним оператором, а компилятор сам будет распознавать его значение.

Так в C#, оператор"->" используется очень ограничено (в unsafe блоках, о которых речь пойдет ниже), оператор "::" вообще не существует. Практически всегда вы используете только оператор "." и вам больше не нужно стоять перед выбором.

Еще один пример. При написании программ на C/C++ вам приходилось

думать не только о типах данных, но и о их размере в конкретной реализации. В C# все упрощено - теперь символ Unicode называется просто char (а не wchar_t, как в C++) и 64-битное целое теперь - long (а не __int64). Также в C# нет знаковых и беззнаковых символьных типов.

В C#, также как и в Visual Basic после каждого выражения case в блоке

switch подразумевается break. И более не будет происходить странных вещей если вы забыли поставить этот break. Однако если вы действительно хотите чтобы после одного выражения case программа перешла к следующему вы

можете переписать свою программу с использованием, например, оператора goto.

Многим программистам (на тот момент, наверное, будущим программистам) было не так легко во время изучения C++ полностью освоиться с механизмом ссылок и указателей. В C# (кто-то сейчас вспомнит о Java) нет указателей. В действительности нетривиальность указателей соответствовала их полезности. Например, порой, трудно себе представить программирование без указателей на функции. В соответствии с этим в C# присутствуют Delegates – как прямой аналог указателя на функцию, но их отличает типовая защищенность, безопасность и полное соответствие концепциям объектно-ориентированного программирования.

Хотелось бы подчеркнуть современное удобство C#. Когда вы начнете работу с C#, а, надеюсь, это произойдет как можно скорее, вы увидите, что довольно большое значение в нем имеют пространства имен. Уже сейчас, на основе первого примера, вы можете судить об этом - ведь все файлы заголовков заменены именно пространством имен. Так в C#, помимо просто выражения using, предоставляется еще одна очень удобная возможность – использование дополнительного имени (alias) пространства имен или класса.

Современность C# проявляется и в новых шагах к облегчению процесса отладки программы. Традиционым средством для отладки программ на стадии разработки в C++ является маркировка обширных частей кода директивами #ifdef и т.д. В C#, используя атрибуты, ориентированные на условные слова, вы можете куда быстрее писать отлаживаемый код.

В наше время, когда усиливается связь между миром коммерции и миром разработки программного обеспечения, и корпорации тратят много усилий на планирование бизнеса, ощущается необходимость в соответствии абстрактных бизнес процессов их программным реализациям. К сожалению,

большинство языков реально не имеют прямого пути для связи бизнес логики и кода.

Например, сегодня многие программисты комментируют свои программы для объяснения того, какие классы реализуют какой-либо абстрактный бизнес объект.

C# позволяет использовать типизированные, расширяемые метаданные, которые могут быть прикреплены к объекту. Архитектурой проекта могут определяться локальные атрибуты, которые будут связанны с любыми элементами языка - классами, интерфейсами и т.д. Разработчик может программно проверить атрибуты какого-либо элемента. Это существенно упрощает работу, к примеру, вместо того чтобы писать автоматизированный инструмент, который будет проверять каждый класс или интерфейс, на то, является ли он действительно частью абстрактного бизнес объекта, можно просто воспользоваться сообщениями основанными на определенных в объекте локальных атрибутах.

3. Особенности и ограничения C#

Особенности заключаются в следующем:

- Полная поддержка классов и объектно-ориентированного программирования, включая наследование интерфейсов и реализаций, виртуальных функций и перегрузки операторов.
- Полный и хорошо определенный набор основных типов.
- Встроенная поддержка автоматической генерации XML-документации.
- Автоматическое освобождение динамически распределенной памяти.
- Возможность отметки классов и методов атрибутами, определяемыми пользователем. Это может быть полезно при документировании и способно воздействовать на процесс компиляции (например, можно

пометить методы, которые должны компилироваться только в отладочном режиме).

- Полный доступ к библиотеке базовых классов .NET, а также легкий доступ к Windows API (если это действительно необходимо).
- Указатели и прямой доступ к памяти, если они необходимы. Однако язык разработан таким образом, что практически во всех случаях можно обойтись и без этого.
- Поддержка свойств и событий в стиле VB.
- Простое изменение ключей компиляции. Позволяет получать исполняемые файлы или библиотеки компонентов .NET, которые могут быть вызваны другим кодом так же, как элементы управления ActiveX (компоненты COM).
- Возможность использования C# для написания динамических web-страниц ASP.NET.

Ограничения заключаются в следующем:

Одной из областей, для которых не предназначен этот язык, являются критичные по времени и высокопроизводительные программы, когда имеет значение, занимать исполнение цикла 1000 или 1050 машинных циклов, и освобождать ресурсы требуется немедленно. *C++* остается в этой области наилучшим из языков низкого уровня. В *C#* отсутствуют некоторые ключевые моменты, необходимые для создания высокопроизводительных приложений, в частности подставляемые функции и деструкторы, выполнение которых гарантируется в определенных точках кода.

4. Сравнение C# с другими языками программирования

C#, являясь последним из широко распространенных языков программирования, должен впитать в себя весь имеющийся опыт и вобрать лучшие стороны существующих языков программирования, при этом являясь специально созданным для работы в .NET. Сама архитектура .NET продиктовала ему (как и многим другим языкам, на которых можно писать

под .NET) объектно-ориентированную направленность. Конечно, это не является правилом, возможно создание компиляторов даже функциональных языков по .NET, на эту тему существуют специальные работы.

Сравнение C, C++ и C#

Критерии сравнения		**C**	**C++**	**C#**
Парадигмы	Императивный	Присутствует	Присутствует	Присутствует
	Объектно-ориентированный	Отсутствует	Присутствует	Присутствует
	Функциональный	Отсутствует	Поддерживается очень ограниченно	Поддерживается не полностью
	Обобщённое программирование	Отсутствует	Присутствует	Присутствует
	Логический	Отсутствует	Отсутствует	Отсутствует
	Декларативный	Отсутствует	Отсутствует	Поддерживается очень ограниченно
Типизация	Статическая типизация	Присутствует	Присутствует	Присутствует
	Динамическая типизация	Отсутствует	Поддерживается не полностью	Отсутствует
	Явная типизация	Присутствует	Присутствует	Присутствует
	Неявная типизация	Отсутствует	Поддерживается не полностью	Поддерживается не полностью
	Неявное приведение типов без потери данных	Присутствует	Присутствует	Присутствует
	Неявное приведение типов с потерей данных	Присутствует	Отсутствует	Присутствует
Управление памятью	Создание объектов на стеке	Присутствует	Присутствует	Присутствует
	Неуправляемые указатели	Присутствует	Присутствует	Присутствует
	Ручное управление памятью	Присутствует	Присутствует	Присутствует
	Сборка мусора	Отсутствует	Отсутствует	Присутствует
Управление потоком вычислений	Инструкция goto	Присутствует	Присутствует	Присутствует

	Инструкция break без метки	Присутствует	Присутствует	Присутствует
	Инструкция break с меткой	Отсутствует	Отсутствует	Отсутствует
	Поддержка try/catch	Отсутствует	Присутствует	Присутствует
	Блок else (исключения)	Отсутствует	Отсутствует	Присутствует
данных	Цикл foreach	Отсутствует	Поддерживается не полностью	Присутствует
	Алгебраические типы данных	Отсутствует	Отсутствует	Отсутствует
Типы и структуры	Многомерные массивы	Присутствует	Присутствует	Присутствует
	Динамические массивы	Отсутствует	Поддерживается не полностью	Поддерживается не полностью
	Контроль границ массивов	Отсутствует	Поддерживается очень ограниченно	Присутствует
	Целые числа произвольной длины	Отсутствует	Отсутствует	Присутствует
	Целые числа с контролем границ	Отсутствует	Отсутствует	Отсутствует
Возможности ООП	Интерфейсы	Отсутствует	Поддерживается не полностью	Присутствует
	Мультиметоды	Отсутствует	Отсутствует	Поддерживается очень ограниченно
	Множественное наследование	Невозможно	Присутствует	Отсутствует
Разное	Шаблоны	Отсутствует	Присутствует	Присутствует
	Перегрузка функций	Отсутствует	Присутствует	Присутствует
	Динамические переменные	Отсутствует	Отсутствует	Нет данных
	Именованные параметры	Отсутствует	Отсутствует	Присутствует
	Значения параметров по умолчанию	Отсутствует	Присутствует	Присутствует

Таким образом, можно сделать вывод о том, что C# обладает большим количеством возможностей, по сравнению с C и C++, так как поддерживает полностью или частично некоторые отсутствующие или слабо реализованные в предыдущих версиях возможности.

5. Интерфейсы

Интерфейс содержит определения для группы связанных функциональных возможностей, которые не абстрактные class или struct должны реализовываться. Интерфейс может определять методы static, которые должны иметь реализацию. Интерфейс может определить реализацию по умолчанию для членов. Интерфейс может не объявлять данные экземпляра, такие как поля, автоматически реализованные свойства или события типа свойств.

С помощью интерфейсов можно, например, включить в класс поведение из нескольких источников. Эта возможность очень важна в C#, поскольку этот язык не поддерживает множественное наследование классов. Кроме того, необходимо использовать интерфейс, если требуется имитировать наследование для структур, поскольку они не могут фактически наследовать от другой структуры или класса.

Вы определяете интерфейс с помощью ключевого interface слова, как показано в следующем примере.

C#Копировать

```
interface IEquatable<T>
{
   bool Equals(T obj);
}
```

Имя интерфейса должно быть допустимым именем идентификатора C#. По соглашению имена интерфейсов начинаются с заглавной буквы I.

Любой объект (класс или структура), реализующий интерфейс IEquatable<T>, должен содержать определение для метода Equals, соответствующее сигнатуре, которую задает интерфейс. В результате можно рассчитывать на класс типа T , реализующего IEquatable<T> Equals метод, с

помощью которого экземпляр этого класса может определить, равен ли он другому экземпляру того же класса.

Определение IEquatable<T> не предоставляет реализацию для метода Equals. Класс или структура может реализовывать несколько интерфейсов, но класс может наследовать только от одного класса.

Дополнительные сведения об абстрактных классах см. в статье Abstract and Sealed Classes and Class Members (Абстрактные и запечатанные классы и члены классов).

Интерфейсы могут содержать методы экземпляра, свойства, события, индексаторы, а также любое сочетание этих четырех типов членов. Интерфейсы могут содержать статические конструкторы, поля, константы или операторы. Начиная с C# 11, элементы интерфейса, которые не являются полями, могут быть static abstract. Интерфейс не может содержать поля экземпляров, конструкторы экземпляров или методы завершения. Члены интерфейса по умолчанию являются общедоступными, и вы можете явно указать модификаторы доступа, такие как public, protected, internal, private, protected internal или private protected. Элемент private должен иметь реализацию по умолчанию.

Для реализации члена интерфейса соответствующий член реализующего класса должен быть открытым и не статическим, а также иметь такое же имя и сигнатуру, что и член интерфейса.

Класс или структура, реализующая интерфейс, должна предоставлять реализацию для всех объявленных членов без реализации по умолчанию, предоставленной интерфейсом. Однако если базовый класс реализует интерфейс, то любой класс, производный от базового класса, наследует эту реализацию.

В следующем примере показана реализация интерфейса IEquatable<T>. Реализующий класс Car должен предоставлять реализацию метода Equals.

C#Копировать

```
public class Car : IEquatable<Car>
{
   public string? Make { get; set; }
   public string? Model { get; set; }
   public string? Year { get; set; }

   // Implementation of IEquatable<T> interface
   public bool Equals(Car? car)
   {
      return (this.Make, this.Model, this.Year) ==
         (car?.Make, car?.Model, car?.Year);
   }
}
```

Свойства и индексаторы класса могут определять дополнительные методы доступа для свойства или индексатора, определенного в интерфейсе. Например, интерфейс может объявлять свойство, имеющее акцессор get. Класс, реализующий этот интерфейс, может объявлять это же свойство с обоими акцессорами (get и set). Однако если свойство или индексатор использует явную реализацию, методы доступа должны совпадать.

Интерфейс может наследовать от одного или нескольких интерфейсов. Производный интерфейс наследует члены от своих базовых интерфейсов. Класс, реализующий производный интерфейс, должен реализовывать все члены в нем, включая все члены базовых интерфейсов производного интерфейса. Этот класс может быть неявно преобразован в производный интерфейс или любой из его базовых интерфейсов. Класс может включать

интерфейс несколько раз через наследуемые базовые классы или через интерфейсы, которые наследуются другими интерфейсами. Однако класс может предоставить реализацию интерфейса только однократно и только если класс объявляет интерфейс как часть определения класса (class ClassName : InterfaceName). Если интерфейс наследуется, поскольку наследуется базовый класс, реализующий этот интерфейс, то базовый класс предоставляет реализацию членов этого интерфейса. Но производный класс может повторно реализовать любые члены виртуального интерфейса и не использовать наследованную реализацию. Когда интерфейсы объявляют реализацию метода по умолчанию, любой класс, реализующий этот интерфейс, наследует эту реализацию (необходимо привести экземпляр класса к типу интерфейса для доступа к реализации по умолчанию на члене интерфейса).

Базовый класс также может реализовывать члены интерфейса с помощью виртуальных членов. В таком случае производный класс может изменять поведение интерфейса путем переопределения виртуальных членов.

6. Сфера применения C#

Язык C# практически универсален. Можно использовать его для создания любого ПО: продвинутых бизнес-приложений, видеоигр, функциональных веб-приложений, приложений для Windows, macOS, мобильных программ для iOS и Android.

Видеоигры

C# без преувеличения крайне популярен среди создателей видеоигр. Язык используется для разработки игр под Windows, macOS, Android и iOS. Все дело в Unity – платформе для работы с 3D-графикой. C# лучше остальных языков адаптирован под работу с этим движком. Поэтому программисты обычно не выбирают, а сразу используют связку Unity + C#.

Из популярных проектов стоит выделить такие хиты игровой индустрии, как Bastion (кросплатформенная РПГ-адвенчура с изометрическим видом), Wasteland (популярный шутер в пост-апокалиптической вселенной), знаменитый Doom 3 и Hearthstone (карточная игра во вселенной World of Warcraft, созданная силами Blizzard).

ПО для защиты систем

Безопасность ваших программ и операционных систем обеспечивается благодаря мощным утилитам на базе C#. Колоссальное количество вирусов, на ежедневной основе атакующих компьютеры пользователей, блокируется инструментами, созданными с помощью языка Microsoft. Аналогичная ситуация наблюдается в крупном бизнесе – мировые корпорации защищаются от хакерских атак с помощью ПО, написанного на C#.

Приложения для Windows

Практически вся операционная система Microsoft существует благодаря C#. Привычные вам утилиты и приложения созданы с использованием этого языка и фреймворков, разработанных для него.

В эту категорию попадает мессенджер Skype, браузер Internet Explorer, среда для разработки Visual Studio 2012, Microsoft Office (все его составляющие, включая Word, PowerPoint, Excel, Outlook и так далее).

Сюда же можно отнести продукты компании Adobe (Photoshop, Lightroom), браузер Mozilla Firefox и Winamp.

Мобильные приложения

В некоторых кругах программистов C# считается чуть ли не лучшим языком для проектирования мобильных приложений. Все благодаря возможности создавать с помощью этого языка нативные программы для любых платформ

(iOS, Android). Для создания приложений, которые идеально работают на Айфоне и на Андроид-смартфонах, используется IDE Xamarin.

Из известных программ, написанных на C#, стоит отметить Slack, Pinterest, Tableau, The World Bank и другие. «Плиточные» программы, появившиеся в Windows 8, практически все построены на базе C# и XAML.

Подитожим комплексно основные области применения:

- Web – разработка web-приложений и сервисов для платформ macOS, Windows, Linux и Docker.
- Mobile – разработка единой кодовой базы для построения нативных приложения для iOS и Android.
- Desktop – разработка нативных приложения под Windows и macOS.
- Microservices – разработка независимых компонентов запускаемы в Docker контейнерах.
- Cloud – использование существующих облачных решений или создание собственных. C# поддерживается большинством облачных платформ, такими как Azure и AWS.
- Machine learning – разработка приложений искусственного интеллекта и машинного обучения, решающие проблемы машинного зрения, обработки речи, моделей предсказания, и тд.
- Game development – разработка 2D и 3D игра для самых популярных десктопных и мобильных платформ.
- Internet of Things (IoT) – разработка приложений для интернета вещей, имеющие поддержку Rasbery Pi и других одноплатных компьютеров.

Исходя из вышеперечисленных областей применения видно, что платформа .NET и язык программирования C# покрывают большой спектр проектов на рынке. Это говорит нам о том, что изучив язык программирования C# с легкостью можно найти проект на любой вкус.

Заключение

В заключении данной работы хотелось бы отметить, язык программирования C# зрелый и достаточно современный. Большинство последних подходов добавлены в язык или планируются в ближайших версиях. Основная территория платформы .NET и языка C# это энтерпрайз, а это возможность работать над большими и сложными проектами в различных доменных областях, а относить это к преимуществам или недостаткам, это решать вам. Возможна мобильная разработка и разработка игр. Огромное сообщество разработчиков, множество литературы и ресурсов для изучения, большое количество открытых вакансий на рынке, говорит о стабильности и крепких позициях C# на рынке.

Литература и источники:

1. Лахатин, А.С. Языки программирования. Учеб. пособие / А. С. Лахатин, Л.Ю. Исакова. – Екатеринбург, 1998 – 548с.: ил.
2. Уэйт, М. Язык С. Руководство для начинающих. / М. Уэйт, С. Прага, Д. Мартин. - М.: Мир, 1995. - 521с.: ил.
3. Петцольд Ч. Программирование для Microsoft Windows на C#. В 2-х томах: Пер. с англ. - М.: Издательско-торговый дом "Русская Редакция", 2002.
4. Марченко А. Л. Основы программирования на C# 2.0. - М.: БИНОМ (Лаборатория знаний, Интернет-университет информационных технологий -ИНТУИТ.ру), 2007
5. Воройский Ф. С. Информатика. Новый систематизированный толковый словарь-справочник. - 3-е изд. - М.: ФИЗМАТЛИТ, 2003
6. Brown E. Windows Forms Programming with C#. - Manning Publications Co., 2002
7. Компьютерное Обозрение, #39, 11 - 17 октября 2000
8. Симон Робинсон, Олли Корнес, Джей Глинн и др. "C# для профессионалов"
9. Эндрю Троелсен. "C# и платформа .NET"
10. https://learn.microsoft.com/ru-ru/dotnet/csharp/fundamentals/types/interfaces
11. https://timeweb.com/ru/community/articles/chto-takoe-csharp