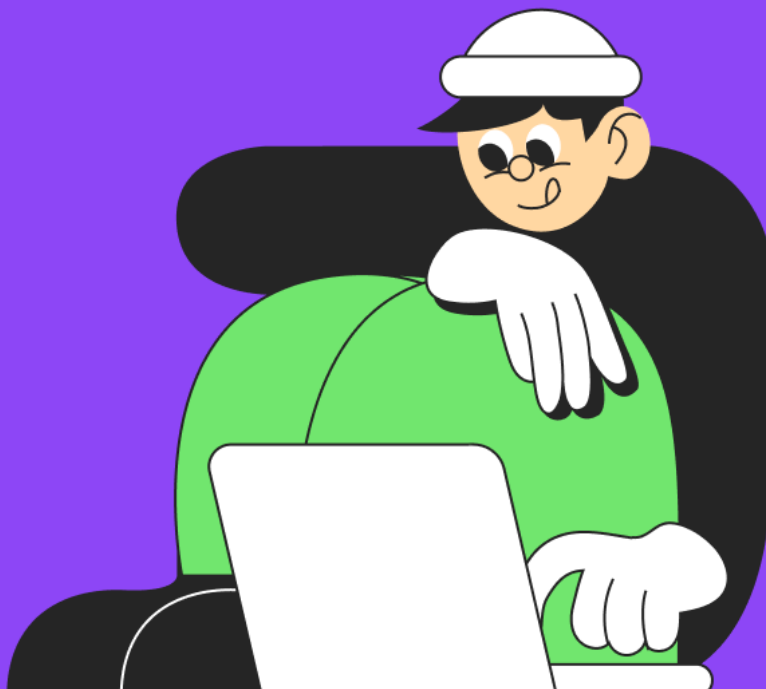
GeekBrains

Знакомство с языком программирования C#



Оглавление

[00:01:35]

Приветствие

Друзья, всем привет. На сегодняшней лекции мы начнём знакомиться именно с программированием и, в частности, с языком C#. Именно его мы выбрали, чтобы ввести вас в эту интересную сферу. приступим.

Меня зовут Сергей Каменецкий. С программированием я познакомился в 2008 году, ещё на первом курсе университета. Конкретно язык C# я начал изучать в 2011 году. И если кому-то интересно, то основными моими языками сейчас является именно язык C#. Плюс ещё «предпроекты» делаю на Swift и в некоторых случаях использую Python.

[00:02:15]

Почему мы выбрали язык программирования C#?

Чтобы начать программирование, нужно выбрать язык C#. Почему мы остановили выбор на языке C#? Здесь и далее я буду делать небольшие отсылки к предыдущим курсам. Напоминаю о том, что в мире более 7 тыс. разговорных языков. С языками программирования ситуация немного проще, но речь всё равно идёт о нескольких сотнях наименований. В то же время основных или наиболее популярных всего около 30. Если посмотреть различные рейтинги популярности, где-то C# занимает 4 позицию, где-то 8, но всегда в топ-10 языков он точно входит.

Так почему мы выбрали язык C#? Во-первых, как я уже сказал, он достаточно популярный. А во-вторых, на нём можно делать всё что угодно. Например:

- Писать проекты для мобильных телефонов, используя фреймворк Xamarin.
- Создавать игры, используя игровой движок Unity, где C# выступает скриптовым языком.
- Писать десктопные приложения для Windows и Mac OS.
- Создавать веб-проекты.
- Технически можно даже создавать проекты для «интернета вещей». В частности, для этого предназначены RaspberryPi, старше версии 2, и Windows-IT.

[00:03:33]

История C#

Далее, «по канону», должна быть информация о том «как, кем и когда» был создан этот язык. Но полезна ли эта информация для программирования? Для личного развития эта информация доступна в поисковике. Я расскажу коротко: в конце 90-х - начале 00-х Microsoft решила конкурировать с Sun microsystems и её набиравшим популярность языком Java. Аналогом платформы Java RE стала платформа Dot.net, а соперником языка Java — язык C#. На старте C# очень многое взял из других языков, например, из C++ и Java, поэтому они очень похожи. Однако позднее, уже Java, начала многое заимствовать у C#.

[00:04:17]

Язык программирования — это инструмент

На этом курсе необходимо запомнить главное: «Язык программирования — это инструмент». Неважно какой язык: C#, Java, Swift, Python или какой-то другой. Задача этого курса научить конкретно программированию. Часто, особенно начинающие, путают процесс программирования и процесс изучения синтаксиса языка. Синтаксис языка – возможно выучить за несколько дней. А программирование, написание кода и создание программ, которые бы работали, и решали поставленные вами задачи, занимает гораздо больше времени. Мы ещё раз отметим главное — синтаксис точно учится, а программированию необходимо будет научиться на этом курсе.

Основное для нас в процессе обучения — это научиться мыслить алгоритмами и перекладывать эти алгоритмы на синтаксис языка C# для создания программы.

[00:05:27]

Кто такой программист?

Важный момент: иногда программистов путают с хакерами или «ты же программистами», когда нужно починить бытовую технику или компьютер, что-то взломать или получить куда-то доступ, в общем, заниматься всем, что связано с электроникой или информацией. В таких случаях можно смело давать отпор. Хакеры — это люди, которые в первую очередь, очень хорошо разбираются в информационной безопасности. «Ты же программисты» – это «мастера на все руки», в ИТ-сфере чаще называемые «эникейщики». А программисты — это люди, которые пишут код, воплощают алгоритмы, полученные из разных источников, превращая их в программу. Это ваше ремесло программирования.

[00:06:09]

Первые шаги написания программы?

С чего начинается написание программы? Неважно, когда это случится сейчас, через месяц, год или несколько лет. В первую очередь, после того как вы получаете задачу, вы должны чётко осознать её. Если какие-то детали неясны, их надо уточнить. Возможно, не один раз и не два. И только после этого можно начать писать код.

Мы на пороге создания нашего первого приложения. А первое приложение будет банальное «hello world» или «приветствие мира», как его ещё называют. Для этого нам необходимо будет настроить рабочее место, окружение и написать тот самый «hello world». Не надо переживать, что всё это умеют и всё это писали. Дело в том, что эта программа показывает: всё правильно настроено, работает, и готово к выполнению более сложных задач. Практически все программисты начинали с этой же программы, даже создатель C# Андерс Хайнсберг, как и признанный эксперт, Джеффри Рихтер, с трудами которого стоит ознакомиться, если основным языком вы выберете C#.

[00:07:30]

Что такое программа?

Для простых людей программа представляет собой набор кнопок, и скрытая внутри «магия кода». С точки зрения программистов — всё гораздо прозрачнее и прозаичнее. Есть 3 слоя:

1. Интерфейс взаимодействия — то, что видит пользователь. Например, веб-приложение (почта, страница социальной сети), десктопное приложение (калькулятор или блокнот) и т. п.
2. Логика — механизм работы программы, которым мы озадачены больше всего, и надо описывать максимально чётко.
3. Данные — «Пища» для второго слоя, то, что будет обрабатывать наша «логика».

На картинке отражено то, что 95% населения, как раз таки не будут понимать, чем вы будете заниматься следующие несколько недель или лет. Но никакой магии здесь нет. Только чистая логика и математика.
Что такое программа? Сейчас с этим разберёмся. Рассматривать программу саму по себе, смысла мало. В узком смысле — это набор алгоритмов. Эти алгоритмы получают на входе какие-либо данные, а на выходе мы получаем какой-то результат. Программа в широком смысле – это набор алгоритмов, которые получают данные на вход и по результатам работы дают данные на выходе. Она может быть абсолютно разной. Например:

- Текстовый редактор. На вход подаётся текст, на выходе также текст, отредактированный, отформатированный или исправленный.
- Калькулятор. На вход подаются одни числа, на выходе результат действий над этими числами.
- Видео редактор. На вход подаётся несколько видеофайлов, на выходе получаем один красочный видеоряд.

И так любые программы, с любым типом данных на входе и выходе.

[00:09:40]

Первая программа
Создаём алгоритм

Основная идея понятна. Настало время написать свою первую программу. Ранее упоминалось, что это будет программа «приветствие мира». Перед тем как начинать писать код, настоятельно рекомендуется ненадолго остановиться и создать правильный алгоритм. На бумаге, или в электронном виде. В виде текста или блок-схемы. Чем сложнее программа, тем сложнее алгоритмы. Но с опытом, часть простых алгоритмов будет просчитываться в уме, без переноса на носители информации. После того как вы опишете основные действия, которые будут происходить в приложении, и проиллюстрируете их блок-схемой, можно будет перейти к написанию кода.

[00:10:25]
Настройка окружения

Первым делом необходимо настроить наше окружение. После этого можно написать первую программу. Повторю, что не стоит её недооценивать. Написание этой программы показывает, что всё работает правильно, настроено хорошо и вы не допустили ошибок.

Для настройки окружения сначала его необходимо скачать и установить. Все ссылки на установочные пакеты C# будут в описании к лекции. Для начала смотрите и наблюдайте, во время перерыва вы сможете повторить все действия самостоятельно. Это важно: не старайтесь на ходу повторять все действия. Лучше наблюдайте и за вопросы в комментариях, чтобы получить все интересующие ответы в режиме реального времени. Смотрим и запоминаем.

Переходим по ссылке на установочный файл. Обратите внимание, если компьютер на базе Mac OS, то нужно скачивать пакет x64. Даже если у вас новый ПК на базе Apple M1, RM версия может не заработать, поэтому лучше поставьте x64. Если у вас Windows, качаете либо x86 и либо x64 в зависимости от разрядности вашей ОС. После установки пакета необходимо перейти в Visual Studio Code. На вкладке расширений («extension») вводим «C#». Первым пунктом отобразится расширение C#. Обратите внимание на разработчика — это официальный пакет Microsoft, его и необходимо установить. В моём случае он уже установлен, но технически здесь будет кнопка «install». Нажимаем её и через какое-то время расширение будет установлено. Если вы будете писать более сложные программы, можно поставить расширение «C# extensional». В некоторых случаях для сложных проектов, оно очень упрощает написание кода. Сейчас оно не является обязательным, но рекомендуется для более детального и полного изучения C# в будущем. После установки необходимых пакетов можно приступать к написанию программы.

[00:13:00]
Создаём первый проект

Перейдём на вкладку «Explorer» и нажмем кнопку «open folder». Далее мы можем создать новую папку. Пусть это будет папка будет называться «Examples». Добавим её в наше рабочее пространство.

Следующим шагом в рамках текущей папки сделаем следующее: запустим терминал и напишем команду “git init”. Далее правой кнопкой в рамках текущей папки мы создадим новую папку. Это будет наш первый проект «Example001», назовем его «HelloConsole». Следующим этапом, можно в терминале набрать «change directory» и перейти в эту папку. Либо нажать правой кнопкой и выбрать "открыть в интегрированном терминале". Обратите внимание, что открывается новая копия терминала. Для того, чтобы подготовить эту папку для написания проекта, нужно ввести следующую команду: «dotnet new console». Это будет подготовкой к созданию нового проекта. После осуществления данных действий, вы увидите, что в папке происходит создание проекта. Добавляется некоторое количество конфигурационных файлов. Большая часть из них не являются важными. Для нас важным является файл «Program.cs». В нем содержится основной код нашего приложения.

[00:14:42]

Код для программы «Hello, World»

Немного уменьшим размеры панели навигации и терминал. Любой код, который написан после двух обратных («//») слэшей называется «комментарием к коду» и не влияет на работу приложения. Комментарий нужен, чтобы программист, например, не забыл, что создавал или для чего нужен тот или иной блок. Настоятельно рекомендую писать такие комментарии, особенно на старте изучения. Для нас верхний комментарий не является чем-то важным, поэтому его можно удалить. Дальше строка «Console.WriteLine» и в скобках «hello world». Что значит «Console»? «Console» — это команда, позволяющая работать с основным окном терминала. Соответственно, мы в него будем какие-то данные выводить, и какие-то данные в нашу программу вносить, путём ввода этих данных напрямую в терминале. Оператор или команда «WriteLine» — нужна для непосредственного вывода, чего-то в окно терминала. Конкретно в этом случае текст, который пишется в круглых скобках и двойных кавычках ("Hello, World"). Этот текст будет выведен в терминал.

[00:15:51]

Запускаем код

Чтобы запустить и проверить работоспособность приложения, необходимо ввести команду «dotnet run». Нажимаем Enter и происходит компиляция проекта. Создаётся ещё некоторое количество файлов и папок в папке проекта. После того как появляется и обновляется файл Bin, в терминале выводится «hello world».

Что дальше? Дальше можно убедиться в том, что работает именно текущий проект и внести изменения. Например, изменим фразу на «2 Hello world». И перезапустим приложение командой «dotnet run». Все изменения отобразились.

[00:16:42]

Рекомендации по настройке Visual Studio Code

Небольшая рекомендация по настройке Visual Studio Code для тех, кто будет писать свои приложения. До текущего момента вы знали о том, что постоянно нужно нажимать кнопки «control+s». Чтобы этого не делать, перейдём в настройки Visual Studio Code, переходим на первую вкладку, на которой написано «Auto Save» и вместо «off», по умолчанию, выберите значение «on focus change». За что это отвечает? если оставить off и внести какие-то изменения, файл не изменится, пока не нажмёте кнопки Ctrl или Command+s. Соответственно, при переходе в терминал ничего не изменится и в некоторых случаях это вызывает проблемы при отладке. Вы вроде бы внесли изменения, но ничего не изменилось. Верный признак того, что вы не сохранили изменения в файл. Придётся нажать CTRL+s и повторить действие. Для избегания таких ситуаций настоятельно рекомендуется в настройках выставить «on focus change». Но если в рамках вашего проекта много постоянно обновляющихся файлов, это может вызвать замедление или зависания при работе. В дальнейшем помните, если начинает тормозить visual studio code, то, скорее всего, надо будет эту настройку перевести на положение по умолчанию, т. е. off.

Следующая рекомендация относится к настройкам шрифтов. В частности, для меня, важным является размер шрифта в терминале, поэтому в поиске настроек можно написать «terminal font» и изменить «font size 14», например, на 20 для увеличения шрифта в терминале. Можете для себя настроить максимально удобные размеры. Я верну, как было. На этом с настройками можно закончить. Дальше мы удалим всё лишнее и ещё раз убедимся в том, что всё работает.

[00:18:53]

Новые слова

Первое приложение написано, добавляем в копилку новых слов:

- Dotnet new console — эта команда нужна, чтобы создать новый проект.
- Dotnet run — эта команда нужна, чтобы запустить проект. Позднее я покажу, что будет, если у вас возникают какие-то ошибки, как нужно будет действовать и каким образом их чинить.
- Через Console мы обращаемся к окну терминала.
- С помощью «WriteLine» можем в консоли данные выводить.

[00:19:20]

Код для программы запроса данных у пользователя

Теперь попробуем написать приложение, которое будет запрашивать данные у пользователя, чтобы дальше с ними работать. Для этого мы поприветствуем теперь уже не мир, а пользователя. Как я и говорил ранее, чтобы начать создавать своё первое приложение, мы настроили окружение. Теперь создаём второе приложение. Снова возвращаемся к описанию алгоритма. В первую очередь вы пишите алгоритм для себя. Возможно, визуализируете блок-схему. Разбиваете на отдельные шаги. После этого начинаете писать код. Ни в коем случае не наоборот.

Ходят слухи о том, что вы достаточно хорошо уже знакомы с Git, поэтому запушим наши изменения в репозиторий. Для этого добавим, новый comit. Обязательно перейдём в нужный терминал, который завязан на наши папки. И напишем для начала git add и добавим нашу первую папку Example001_HelloConsole. Сделали. Дальше git commit -a. Добавляем комментарий Example 001 Hello Console project. Следующим этапом пишем get push origin main. Затем идём в наш репозиторий и смотрим. Да, у нас действительно появилась папка. Обратите внимание, есть наш файл Program.cs и ещё много разного. Нужно ли оно здесь? Это отдельные вопросы и позже я коснусь того, почему не нужно эти папки добавлять в git и как от этого можно избавиться.

[00:21:48]

Код для программы приветствия пользователя

Создаём новую папку

Теперь в нашей папке Example создадим новую папку, назовём её Example002_HelloUser. Снова откроем в терминале. Кстати, предыдущий можно закрыть. Здесь добавим команду dotnet new console. Создаётся структура проекта.

Далее убедимся, что проект создан корректно. Кстати, это файлик не тот. Закрываем и открываем нужный. Соответственно, будем с вами теперь вносить необходимые изменения.

В первую очередь напоминаем себе основной алгоритм. Для начала нужно ввести имя пользователя, дальше вывести условное «привет» и после этого вывести то, что введёт пользователь.

Теперь воспользуемся командой Console.Read Line. Именно с её помощью происходит считывание данных. Но эти данные нужно где-то сохранить. Что-то похожее мы использовали в математике, когда обращались к буквенным выражениям. Наша задача в том, чтобы данные, которые ввёл пользователь, где-то нужно сохранить. Это «где-то» в контексте программирования называется «переменной», для неё справедливо понятие — объём занимаемой памяти или «тип данных».

[00:23:52]

Пишем код

Сейчас нужно создать некий контейнер (коробку), куда можно положить значения, введённые пользователем. Соответственно, пишем.

```
string username = Console.ReadLine();
Console.WriteLine("Привет,                                                    ");
Console.WriteLine(username):
```

Пусть данные, которые вводит пользователь, очевидно, это будут строки, мы так и пишем string (строка). Дальше даём наименование нашему контейнеру (имя переменной или идентификатора). Назовём username и при помощи оператора равенства, которое не является классическим равенством из математики, где оно означает «то, что справа равно тому, что слева». Здесь это работает по-другому. Такой оператор нужно воспринимать, как «возьми то, что справа и положи в то, что слева». В некоторых языках программирования, например, в Паскале оператор присваивания выглядел как « := », где-то это может выглядеть как « ::= », где-то иногда это пишут « <- ». Мы остановимся на « = ».

Далее мы хотим вывести приветствие в консоль. Значит, пишем Console.WriteLine. Здесь добавим «Привет, » и на следующей строке мы с вами напишем console.writeLine (username). Обратимся к терминалу и повторим запуск.

Сейчас попробуем написать здесь, например, «Сергей» и нажмём Enter, который является признаком окончания ввода строки. Дальше мы видим строчку, «Привет» и ниже наше имя.

[00:25:45]
Добавляем вывод строки «Введите ваше имя»

Чтобы пользователь понимал, что в момент запуска приложения, нужно вводить данные, я предлагаю сделать так называемое приглашение к вводу. Добавим ещё одну команду Console.WriteLine возьмём двойные кавычки и напишем «Введите ваше имя». Сохраним и запустим (dotnet run). Теперь мы видим приглашение к вводу «Введите ваше имя», соответственно, вводим имя, нажимаем Enter и наблюдаем «Привет, Сергей».

```
Console.WriteLine("Введите Ваше имя ");
string username = Console.ReadLine();
Console.WriteLine("Привет, ");
Console.WriteLine(username);
```

Дальше я рекомендую внести небольшие изменения. В частности, у нас всё работает хорошо, но попробуем сделать небольшую ошибку и забыть поставить « ; ». На языке C# точка с запятой является признаком окончания оператора, так что она важна. Но, допустим, мы забыли её прописать. Снова пытаемся запустить и видим, что у нас сообщается об ошибке. Поэтому внимательно читайте, что именно вам будет писать терминал и, скорее всего, всё будет хорошо.

Итак, всё запустилось. Поэтому внесём имя и действительно всё хорошо.

[00:27:08]
Команда Console.Write

Следующая команда, которая вам может пригодиться. Помимо Console.WriteLine, которая просто выводит текст в наш терминал и переводит курсор на новую строку, то есть мы начинаем писать с новой строчки. Есть ещё команда Console.Write воспользуемся ей и перезапустим наше приложение. Сделаем так и после нажатия на Enter у нас всё выводится в одну строку.

```
Console.Write("Введите Ваше имя ");
string username = Console.ReadLine();
Console.Write("Привет, ");
Console.Write(usrname);
```

Теперь мы наблюдаем, что у нас «Введите ваше имя» на одной строке. Здесь же мы можем напечатать это самое имя. Но обратите внимание, что появляются такой артефакт в виде значка процента. Не пугайтесь, это не вы сделали какую-то ошибку, это тонкости работы с dotnet в visual studio code.

[00:28:00]
Отправляем изменения в репозиторий

Следующий этапом нам нужно все эти изменения закоммитить и отправить в репозиторий. Но, как я показал на предыдущем примере вместо 4 строчек, у нас появляется ещё много дополнительных конфигурационных исполняемых файлов и т. д. Они в git уже не нужны. Папки bin и auction в репозитории считаются признаком плохого тона.

Чтобы всё хорошо работало, мы можем сделать следующие. В рамках папки Examples добавим файл gitignor. Он будет содержать некоторые инструкции, показывающие git, что именно нужно пушить в репозиторий. Для dotnet я подготовил другой файлик. Мы его просто скопируем. Его не нужно изучать наизусть, скорее всего, для 89% того, что вы будете делать файла gitignor более чем достаточно. Соответственно, мы его сохраняем. Далее идём к терминалу, и теперь наша задача сделать git add Example002_HelloUser и сделаем git commit. И добавим комментарий, вы делали это на курсе через атрибут -m, а я сделаю при помощи a.

add Example 002 Hello User.

Дальше выходим и пушим в нашу веточку. Идём к репозиторию, обновляем и появляется 2 пример. Теперь у нас нет ничего лишнего, только файл проекта. Это то, при помощи чего можно будет в дальнейшем, например, открыть этот проект на платформе Windows, то есть язык C#, он кроссплатформенный, поэтому, написав на одном компьютере, вы можете спокойно продолжить написание этого кода на другом компьютере под управлением другой операционной системы.

[00:30:40]

Новые слова

Хорошо, мы сами создали второй проект. Добавляем ещё новых слов.

- WriteLine — вывод в консоль с переходом на новую строку.
- Write — это вывод в консоль без перехода на новую строку.
- ReadLine — позволяет считывать данные из нашего терминала.

[00:31:00]

Код для программы сложения двух чисел

Определяем и уточняем условия

Теперь попробуем написать более сложную программу, а именно сложить два числа. С чего начинаются все действия? Они начинаются с создания алгоритма. Казалось бы, есть одно число, есть второе число, нужно их сложить, показать результат. Итак, каким образом звучит задача? Написать программу сложения двух чисел. Но, прежде чем начать что-то писать, нужно кое-что уточнить. В первую очередь, на каком языке программирования нужно написать приложение. Допустим, вы знаете язык C#, а заказчику требуется программа на языке JavaScript. После уточнения языка программа, техническое задание или функциональные требования, предъявляемые к этому приложению, звучат, как «написать программу на языке C#, которая складывает два числа». Всё ли на этом? Оказывается, нет. Дальше вспоминаем о том, что язык C# позволяет писать приложения для различных платформ. Это может быть Windows или Mac OS, устройство интернета вещей или очки дополненной реальности Hello Wins и т. д.

Теперь, допустим, заказчик сказал, что это приложение должно работать на компьютерах под управлением Windows и Mac OS. Соответственно, наши функциональные требования звучат, как «написать программу на языке C# работающую на Windows и на Mac OS, которая складывает два числа».

Дальше снова могут появиться уточнения, а именно, откуда берутся эти числа? Технически числа могут вводиться из терминала, считываться из какого-то файла или забираться из какого-то сервиса или базы данных и т. д. Вы должны чётко представлять, откуда у вас берутся данные. Производим уточнения, и оно звучит, как «написать программу на языке C# для Windows и Mac OS, в которой пользователь вводит два числа, или числа генерируются автоматически, или считываются из файла. Дальше получаем их сумму». Всё ли на этом? И снова нет. Что значит «получаем сумму»? После того как складываем числа, мы должны показать их сразу в терминал или, быть может, нужно сохранить куда-то в базу данных или отправить какому-то телеграмм бот или ещё куда-то? Нужно понимать, что с этой суммой дальше делать.

Соответственно, снова производим уточнение, добавляем что-то и после этого можем приниматься к написанию программы. Но и это не всё. Проблема в том, что, с какими числами мы работаем? Из курса математики вы помните, что есть натуральные, вещественные или рациональные числа и т. д. Так с какими именно числами наше приложение должно работать. Ограничимся уточнением целых чисел.

Следующим этапом по аналогии с предыдущими проектами мы с вами должны будем описать алгоритм, последовательность действий или блок-схему.

[00:34:08]

Пишем код

Продолжим написание кода и создадим новую папку. Это будет Example003_Sum. Правой кнопкой в терминале, напоминаю, что это можно сделать через команду CD, но так будет немного быстрее и, может быть, проще. Но делайте так, как вам удобно. Дальше пишем dotnet new console, нажимаем Enter и ждём. Производится создание проекта. Затем убеждаемся в том, что оно работает правильно. Теперь проект собран и надо внести в него правки, чтобы получить программу сложения двух чисел.

```
int numberA = 3;
int numberB = 5;
Console.WriteLine(numberA + numberB );
```

Итак, начнём с определения переменной. Пусть первая переменная называется numberA в неё положим конкретное значение, допустим, 3. На следующем этапе необходимо сказать компьютеру, сколько памяти нужно выделить и определить, каким это число является. По аналогии с математикой, если это целое число, значит, множество Z, а если вещественное число, то множество R и т. д. Для компьютерных программ примерно то же самое. При хранении различных чисел используются разные типы данных. Тип данных для целых чисел называется int (целое число). Мы ввели первое число. Дальше сделаем numberB. Отмечу, что в наименование переменных допускается использование больших и маленьких латинских букв. Использовать можно и цифры, но они не должны стоять на первом месте, например, можно указать number2. Но 2number мы точно делать не можем, поэтому тоже будьте внимательнее. Помимо этого, можно использовать символы подчёркивания « _ », но в языке C# для именования переменных это не принято. Дальше возьмём конкретное число 5 и покажем на экране Console.WriteLine. Чтобы таким образом вывести сумму. Запустим и посмотрим, что будет действительно сумма. Видим 8.

```
int numberA = 31;
int numberB = 5;
int result = numberA + numberB;
Console.WriteLine(result);
```

Можно было использовать промежуточную переменную. Создать некоторую дополнительную переменную (контейнер) result и положить в неё сумму. А потом показывать её в консоле. Перезапустим. Опять же изменим значение. Например, 31 здесь поставим. И спустя какое-то время мы с вами наблюдаем сумму чисел. Вроде бы всё. Сравним с нашей блок схемой. Убедимся, что определили numberA и numberB , показали сумму. Также мы добавили дополнительный шаг — сохранение результатов в промежуточную переменную. Можно было этого и не делать.

Хорошо, теперь по аналогии с предыдущими примерами, вернёмся к вашему терминалу и сделаем git add Example003_Sum и добавим комментарий Example 003 Sum. Дальше git push origin main. Готово.

[00:38:10]

Программа для вычисления частного двух чисел

На следующем этапе предлагаю вам создать новое приложение: показать на экране частное двух чисел. Казалось бы, что может быть проще: заменить плюс на деления, и вроде бы всё. Но попробуем сделать это и убедиться, что всё не так просто, как кажется.

Пишем код

Итак, создадим новый проект. Устойчивое запоминание, как говорят некоторые математики, начинается после 19-кратного повторения. Поэтому кликаем правой кнопкой на «Example», выбираем «New folder». Называем папку Example004_Div. Дальше правой кнопкой открываем папку в терминале, пишем «dotnet new console». Закрываем всё лишнее. Сразу же добавляем всё в git: пишем «git add Example004_Div». Готово.

Далее открываем файлик 3Programs.cs. Здесь мы всё перепишем. Сделаем numberA = 10, numberB = 5. Далее можно использовать вспомогательные инструменты, но для первых программ вводить всё

вручную. Вводим Console.WriteLine и в круглых скобках пишем numberA, значок деления (на языке C# он определяется прямым слэшем) и numberB.

```
int numberA = 10;
int numberB = 5;
Console.WriteLine(numberA / numberB);
```

Запускаем проект — вводим dotnet run — и на экране мы ожидаем получить двойку (10 : 5 = 2). Именно такой результат мы получаем, всё хорошо.

Целые и вещественные числа

Попробуем внести небольшие правки. Я изменю значение numberA с 10 на 12. Что будет, если 12 разделить на 5? Перезапускаем и снова получаем 2, хотя по всем правилам должно получиться 2,4.

```
int numberA = 12;
int numberB = 5;
Console.WriteLine(numberA / numberB);
```

Почему мы получили именно такой результат? Дело в том, что, мы работаем в целых числах. Если вы помните, в модуле математики и информатики мы обсуждали деление в целых числах. Здесь мы увидели реальный пример.

Поправим наше приложение, чтобы было деление. Для этого нужно контейнер «integer» (целое число) заменить на контейнер «double», позволяющий работать с вещественными числами.

```
double numberA = 12;
double numberB = 5;
Console.WriteLine(numberA / numberB);
```

Перезапускаем проект и видим на экране 2,4 — тот результат, который и должны были получить. Всё хорошо. Теперь приложение нужно запушить. Вводим в терминале git commit -a и добавляем комментарий dd Example 004 Div. Далее вводим git push origin main. Готово! У нас есть 4 проекта. Продолжим изучение языка C#.

[00:42:14]

Ещё немного о типах данных

Типы данных integer и double

На этом этапе вы должны были запомнить целый тип данных (integer) и вещественный тип данных (double). Обратите внимание, что типы данных различаются не только по возможности хранить вещественные числа, но и по объёму памяти, который занимают. Если целые числа позволяют хранить диапазон от -2 147 483 648 до 2 147 483 647, то тип «double» может хранить числа от ±5,0 × 10-324 до ±1,7 × 10308. Это очень большие значения и, скорее всего, на начальных этапах они вам не понадобятся.

Тип данных string

Также важно помнить, что есть строки (тип данных «string»). Он может содержать от 0 (это называется пустой строкой) до большого количества символов, в общем случае до 2 Гб. Это много. Для сравнения, например, один том «Война и мир» весит около 13 Мб, хотя, возможно, это вес всех книг серии «Война и мир». То есть строки могут быть очень и очень большими.

Тип данных bool

Четвёртый тип данных, с которым нужно познакомиться на старте, это логический тип данных. Он записывается, как bool и позволяет хранить только 2 значения: либо истина, либо ложь. Попробуем познакомиться с ним в следующем проекте.

[00:43:40]

Операторы для арифметических действий в C#

Классические арифметические операции в языке C# определяются следующими операторами:

+ — плюс

- — минус

/ — прямой слэш

* — умножение

% — значок процента (означает нахождение остатка от деления)

( ) — классические скобки, которые могут менять приоритет операций

Здесь всё работает так же, как в математике. Приоритетом являются умножение и деление. Далее, если скобок нет, выполняется сложение и т. д.

[00:44:05]

Команды-помощники программиста на C#. Генератор псевдослучайных чисел

Учитывая тот факт, что мы будем писать много приложений, не всегда удобно всё вводить руками. В некоторых случаях можно обратиться к вспомогательным инструментам. В частности, к генератору псевдослучайных чисел. Он выглядит страшновато, но пока что вы можете читать этот оператор просто как английский текст.

Итак, если я напишу new Random().Next(min, max), что это значит? Это значит, что я хочу новое случайное следующее число в диапазоне от min до max-1. В модуле математики и информатики, вы, наверное, вспоминали о том, что есть отрезки, интервалы, полуинтервалы. Так вот, «min, max» здесь выступает полуинтервалом, в котором min будет включён в полуинтервал, а максимальное число будет на единицу меньше. Обратите внимание, что Next выдаёт случайное целое число. То есть оператор new Random ().Next(min, max) даст случайное целое число в полуинтервале [min, max-1] или [min, max].

Пример работы оператора

Попробуем воспользоваться этим конструктом на нашем примере с суммой чисел. Перейдём в него (в папке Example003_Sum). Для numberA вместо 31 напишем «случайное целое число из диапазона от 1 до 10». Добавим комментарий: «// 1 2 3 4 … 9». И у нас будет случайное число 1,2, 3,4 и т. д., а максимальным числом будет 9.

Зададим такое же значение — new Random().Next(1,10) — для numberB. Перезапустим (dotnet run) и посмотрим, что получится. На первом этапе мы получили 10. То есть это могла быть сумма чисел 2 и 8, а может быть, 5 и 5. Снова перезапускаем и получаем 13.

```
Int numberA = new Random().Next(1,10);
Int numberB = new Random().Next(1,10);
int result = numberA + numberB;
Console.WriteLine(result);
```

Модифицируем код

Чтобы не гадать, мы снова внесём небольшие правки. Мы можем добавить Console.Write() и показать первое число — Console.Write(numberA). Хотя лучше сделать это через консоль Console.WriteLine, чтобы видеть каждое число с новой строки. Добавляем Console.WriteLine(numberB) и перезапускаем. Ожидаем увидеть тройку чисел. Первым было сгенерировано число 4, вторым — 3, а их сумма равна 7. Таким образом, мы можем убедиться, что всё работает.

```
Int numberA = new Random().Next(1,10);
```

```
Console.WriteLine(numberA)
Int numberB = new Random().Next(1,10);
Console.WriteLine(numberB)
Int result = numberA + numberB;
Console.WriteLine(result);
```

Сразу добавим новый commit -a. В нашем случае это «add random», больше здесь особо писать нечего. И пушим его (git push origin main).

[00:47:09]

Код для персонального приветствия

Мы уже создали некоторое количество проектов. Теперь продолжим изучать синтаксис языка C# и заодно учиться программировать. Сейчас я предлагаю написать приложение, которое будет разных пользователей приветствовать по-разному, а именно: выделим для себя условного любимчика, и наше приложение будет реагировать на него по-особенному. Для чего это нужно? В классическом академическом примере это необходимо для знакомства с синтаксисом. А в остальном — смело можно сказать, что 99,9% всех приложений, которые вы будете создавать, будут использовать этот оператор и похожие конструкции.

Здесь нас интересует конструкция ветвления. Приложение будет выглядеть следующим образом:

- Начало
- Имя пользователя (вводим)
- ЕСЛИ имя пользователя совпадает с «Маша», то будем приветствовать его по-особенному.
- ИНАЧЕ — поприветствовать по-обычному (если, например, имя пользователя — Серёжа или Миша, программа просто напишет «Привет» и соответствующее имя)
- Конец приложения

На экране вы можете видеть блок-схему этого приложения.

Перейдём к синтаксису этого оператора. У нас есть ключевое if, после него в круглых скобках пишется условие. Если это условие истинно, будет выполнен один блок операторов (набор действий). Если условие окажется ложным, будет выполняться блок else и второй набор действий.

[00:48:35]

Пишем код для приложения

Напишем наше приложение. Создадим новый проект и новую папку для него, назовём её Example005_ConditionIfElse. Далее правой кнопкой мыши открываем её в терминале и пишем «dotnet new console».

Закрываем всё лишнее и начинаем писать. Для начала просим ввести имя пользователя. Сразу же делаем приглашение к вводу, чтобы, с одной стороны, наш пользователь понимал, что конкретно от него хотят, и, с другой стороны, чтобы было понятно, что наше приложение запустилось. В скобках мы пишем «Введите имя пользователя». Далее указываем тип данных «строки» — string. Здесь будет наименование переменной username, считываем данные мы при помощи Console.ReadLine. Дальше мы хотим сделать проверку. Если наш username равен — так и пишем — «Маша». Дальше ставим фигурные скобки. Внутри скобок мы пишем Console.WriteLine и какой-то текст специально для Маши. Ранее мы договорились показывать «Ура, это же Маша!». Так и напишем.

Далее мы опишем действие для рядового пользователя: Console.Write("Привет, "); и Console.WriteLine(username). Если вдруг кто-то хочет заменить несколько операторов одним, технически это сделать можно. Я покажу, как это можно сделать.

```
Console.Write (“Введите имя пользователя: ”);
string username = Console.ReadLine();
if(username == “Маша”)
{
Console.WriteLine(Ура, это же МАША);
```

```
}
else
{
Console.Write("Привет, ");
Console.WriteLine(username);
}
```

Для начала запустим и убедимся в том, что приложение работает правильно. Вводим dotnet run, нажимаем Enter, видим «Введите имя пользователя», пишем здесь «Маша» и «Ура, это же МАША». Наверное, восклицательного знака не хватает. Добавим его, перезапустим и увидим «Ура, это же МАША!». Отлично. Ещё раз запускаем и вводим другого пользователя, например, «Джон». И мы видим «Привет, Джон».

Проблемы и ограничения в работе приложения

В чём могут заключаться проблемы и есть ли они у нас? Оказывается, есть. Продемонстрирую эти проблемы. Попробуем вместо «Маша» с большой буквы написать «маша» с маленькой буквы. Теперь username не совпадает с именем «Маша», написанным с большой буквы, а значит, таких «Маш» можно придумать очень и очень много (варианты: «МАша», «МАШа» и т. д.). Можно ли описать все эти ситуации? Технически да, можно. Для этого нужно использовать сложные условия, в этой части курса мы не будем их затрагивать.

Каким же образом тогда решать эту проблему? Сделать это можно очень просто. Мы поставим точку после «username» и вызовем операцию ToLower(). Она позволяет перевести все символы в вашей строке в нижний регистр.

```
Console.Write (“Введите имя пользователя: ”);
string username = Console.ReadLine ();
if(username.ToLower() == “Маша”)
{
Console.WriteLine(Ура, это же МАША);
}
else
{
Console.Write("Привет, ");
Console.WriteLine(username);
}
```

Поэтому теперь для нас референтным значением будет являться «маша» с маленькой буквы. Таким образом, мы решили большую часть проблем. Для начала запустим «Маша» с большой буквы. Всё отработало корректно. Далее введём «МАшу» для демонстрации того, что ничего не поломалось. Введём «Том», например, и получим «Привет, Том».

Вновь обратимся терминалу. Пишем git add Example005_ConditionIfElse, далее сделаем commit -a и добавим условия (вводим add Example 005 Condition If-Else). После этого можем запушить (git push origin main).

[00:54:10]

Программа для решения задачи с гирями

Мы постепенно знакомимся с синтаксисом языка C# и начинаем создавать первые программы, описывая первые алгоритмы. Здесь у нас небольшой flashback из модуля «Введение в программирование». Там вы решали задачу с перетаскиванием гирек. Теперь попробуем реализовать её ввиде программы. У вас был набор действий, которые нужно проделать. Помимо этого, была создана блок-схема с путём и указанием конкретных значений в контексте этой блок-схемы. Дальше вы определили, что для вас важен именно алгоритм, а не просто какой-то код, и посмотрели примеры кода на языках Java и Python. Попробуем переписать это приложение на C#.

Пишем код

Создадим папку Example006_Base. Добавим папку, откроем её в терминале, пропишем dotnet new console и проделаем все нужные действия. В нашем примере фигурировали конкретные числа. Соответственно, конкретные числа здесь и пропишем. Int a = 1, int b = 2, int c = 6. Int d поставим, допустим, равный 8. И Int e = 4. Напомню, друзья мои, я проговаривал «равно», но на самом деле = — это оператор присваивания.

Далее мы определяем переменную max, в которую кладём значение первой гири, и теперь нужно прописать набор условий, не нарушая общности. Если значение первой гири получается больше, чем максимум, то в максимум нужно положить новое значение. Немного форматируем код и проделаем остальные действия. Если b больше, чем max, то в этом случае в max положим значение второй гирьки. Дальше, если c больше, чем max, то в этом случае в max кладём c. Если d больше max, то в max кладём d и, соответственно, если e, то в max кладём e.

После этого можно отформатировать текст, чтобы было красиво, и показать значения max на экране, чтобы пользователь тоже понимал, что происходит. Вводим Console.Write и напишем здесь, max равен вот такому действию: Console.Write(“max = ”). Запустим. И на экране мы ожидаем увидеть восьмёрку, но видим ошибку. Ошибка заключается в «; expected» — я забыл добавить точку с запятой в строке 15. Исправляю, делаем dotnet run и — мы получили 8, ровно то, что хотели.

```
int a = 1;
int b = 2;
int c = 6;
int d = 8;
int e = 4;

 int max = a;

if (a > max ) max = a;
if (b > max ) max = b;
if (c > max ) max = c;
if (d > max ) max = d;
if (e > max ) max = e;

Console.Write(“max = ”);
Console.WriteLine(max);
```

[00:58:08]

C# vs Java: сходства и различия

Теперь внесём небольшие изменения, но перед этим сравним, как это выглядело на языке Java, и то, как это выглядит на языке C#. То есть технически, можно сказать, код совпадает один в один, кроме того, что там была system out print, а здесь мы с вами наблюдаем Console.WriteLine, вот и всё.

Сохраняем проект

Запушим наши изменения в репозиторий. Для этого перейдём в соответствующий терминал. Здесь выберем git add нужную папку — Example006_Base. Дальше git commit -a, добавление нашего очередного проекта. И пушим.

[00:59:07]

Оператор цикла

Итак, вы потихоньку изучаете новые операторы. Добавим в нашу копилку ещё один, а именно оператор цикла. Для чего он нужен? В жизни вы проделываете достаточно много повторяющихся действий. К таким действиям относится, например, сервировка стола, когда нужно взять тарелку, потом поставить тарелку, затем взять ложку и куда-то её положить, далее взять бокал — поставить бокал. Представьте, что это нужно проделать 100 раз. Очевидно, что делать это руками — такое себе занятие, а вот нанять какого-нибудь робота было бы прекрасно.

И возникает вопрос: каким образом от этой рутины можно избавиться? Рассмотрим это на примере. Возьмём 3 точки. Для нас это будет треугольник. Выберем две случайных точки и найдём середину отрезка, соединяющего их. Отрезок не нужен, нам потребуется исключительно точка. Поставим её. На следующем этапе снова выберем какую-то из точек – вершин треугольника, снова найдём середину получившегося отрезка и поставим точку. От этой точки определим, какую-то случайную вершину, снова найдём середину, снова поставим точку и т. д.

В идеале таких действий у нас может быть много. Я проделаю около 10, но появляется ли какая-то картина? Пока что не очень понятно. Здесь я предлагаю взять перерыв на 5 минут, самостоятельно проделать эти действия на листочке и посмотреть, что получится. А после перерыва мы попробуем написать приложение, которое будет делать что-то подобное.

Что же, надеюсь, что-то у вас получилось. Теперь определим чёткий алгоритм того, каким образом мы будем писать программу.

Алгоритм:

1. Определить три точки
2. Выбрать две любых
3. Найти середину
4. Поставить точку
5. Выбрать случайную вершину треугольника
6. Соединить её с полученной на 4 шаге точкой
7. Перейти к шагу 3
8. Шаги 3-7 построить 9, 28, 31 раз

Основным пунктом для нас является шаг 8, а именно повторение этих действий 9, 10, а может быть, 100 раз. Как реализовать это в приложении?

Чтобы это сделать, нам потребуется использовать новую конструкцию, называемую циклом. Здесь я привожу пример цикла «while». Выглядит он как while, дальше в скобках мы пишем условие продолжение и действия, которые будут производиться некоторое количество раз. Это количество раз мы можем обозначить как бесконечное, и тогда программа будет зацикливаться. Или мы можем определить, например, не более чем 100 раз.

```
while( УСЛОВИЕ ПРОДОЛЖЕНИЯ)
{
Набор действий
}
Int count = 0;



while( count < 100)
{
Набор действий
count = count + 1
}
```

Мы можем определить некоторый счётчик. Здесь это «int count = ноль». Дальше мы будем проверять, пока count < 100, и выполнять какие-то действия. При этом нужно будет на каждом новом заходе, вновь проделывая эти операци, и увеличивать значение счётчика, чтобы в какой-то момент он достиг 100 и мы вышли из цикла.

[01:07:02]

Приложение с оператором цикла

Итак, попробуем написать такое приложение. Создадим новую папку — Example007_Magic. Правой кнопкой открываем её в терминале. Далее вводим «dotnet new console» и удаляем всё лишнее.

Чтобы каким-то образом рисовать треугольники, нам потребуется добавить (или изучить) новую операцию. Первое — это возможная очистка консоли — Console.Clear. В нашем случае это будет не совсем очисткой, но запустим и посмотрим, как это будет выглядеть. Вводим dotnet run, и перед тем, как наше приложение будет запускаться, всё, что было в консоли, окажется удалённым.

Второе. Команда, которая будет «рисовать». Мы можем использовать: существует команда Console.Set.CursorPosition, и в качестве аргументов мы можем указать отступ от левого края и отступ от верхнего края. Пусть отступ от левого края будет 10 символов, а отступ от верхнего края — 4 символа. Теперь сделаем Console.WriteLine и укажем символы, которыми будем рисовать (пусть это будут плюсики).

```
Console.Clear.();
Console.Set.CursorPosition(10, 4)
Console.WriteLine(“+”);
```

На этом этапе мы ожидаем, что после запуска в терминале плюс должен появиться на расстоянии в 10 символов и 4 строки сверху. От себя отмечу, что строки и символы здесь нумеруются с 0. Соответственно, проходим нулевую, первую, вторую и третью строки, и на четвёртой мы видим то, что и хотели — плюс. Готово. Что дальше?

[01:09:24]

Пишем код

Изучив эти команды, я их закомментирую, просто чтобы они были перед глазами, и начнём писать. Для начала необходимо определить вершины треугольника. Определим мы их следующим образом. Для первой точки, например, «xa», отступ по X будет в 1 символ, и для «ya» отступ также будет равняться 1 символу.

Следующая вершина будет для второй точки — «xb». Она будет находиться по аналогии с примером. Если до этого отступ равнялся 1, то здесь мы отступим хотя бы 10 точек, хотя можно и побольше. Зададим отступ от левого края снова 1 символ, а для отступа от верхнего края — «yb» — отступ будет 30 точек.

```
Console.Clear.();
Console.Set.CursorPosition(10, 4)
Console.WriteLine(“+”);

int xa = 1;
int ya = 1;
int xb = 1;
int yb = 30;
```

От себя отмечу, что конструкцию можно сделать более компактной и перечислить вершины просто через запятую. Это никак не влияет на скорость работы приложения или его корректность. Просто один из вариантов записи.

```
int xa = 1, ya = 1,
xb = 1, yb = 30
```

Определим 3 точку (xc; yc). Если на рисунке у нас точка примерно 1;30, отступ по игреку будет такой же — 30, а по иксу сделаем, например, 40. Если понадобится, позже мы подправим эти числа.

```
int xa = 1, ya = 1,
xb = 1, yb = 30,
xc = 40, yc = 30;
```

Покажем вершины. Для начала нужно будет сделать Console.Set.CursorPosition. Здесь мы устанавливаем точку (xa, ya), и сразу же показываем на экране Console.WriteLine (); и в скобках ставим плюсик. Далее делаем то же самое для (xb, yb) и (xc, yc).

```
 Console.Clear.();
Console.Set.CursorPosition(10, 4)
Console.WriteLine(“+”);

int xa = 1, ya = 1,
xb = 1, yb = 30,
xc = 40; yc = 30;

Console.Set.CursorPosition(xa, ya)
Console.WriteLine(“+”);

Console.Set.CursorPosition(xb, yb)
Console.WriteLine(“+”);

Console.Set.CursorPosition(xc, yc)
Console.WriteLine(“+”);
```

Запускаем программу

Сделаем терминал побольше и посмотрим, что у нас получится. Мы ожидаем увидеть три точки. Всё хорошо. Единственное, последнюю вершину можно немного сдвинуть в сторону. То есть xc = не 40 точек, а 60. Сделаем для неё отступ в 60 символов. Перезапустим: dotnet run. Как мы видим, отступ можно ещё немного увеличить. Зададим для этой точки xc = 80 и снова перезапустим. Теперь мы получили фигуру, похожую на треугольник. Здесь отмечу, что когда мы изменяем размер терминала, треугольник в нём также может выглядеть по-разному.

[01:13:10]

Решаем задачу

Далее мы можем сделать так, чтобы вершина треугольника, наиболее близкая к верхней границе терминала, была не сбоку, а по центру экрана. Сделать это очень просто. Если нижнее основание — это 80 точек, то чтобы поставить вершину примерно посередине, мы можем указать xa = 40, и получится что-то похожее на равнобедренный треугольник.

Переводим алгоритм в код

А теперь мы должны воплотить алгоритм, описанный выше, в виде кода. Для начала я определю случайную точку x и в неё поставлю нашу первую точку (xa, xb). Дальше я указываю некоторый счётчик count, который будет определять, какое именно количество раз нужно будет проделывать операцию нахождения отрезка и деления его пополам. Вводим count = 10. Дальше вводим оператор while. В скобках у оператора мы пока пишем count < 10. Кстати, изначально count равен 0 (в строке 21), а здесь ноль, а здесь (строка 23) мы проверяем. Пока count, например, меньше 10, позже мы можем поправить на меньше 100 и так далее.

Наши действия здесь заключаются в том, чтобы сгенерировать какое-то случайное число. Пусть оно будет называться «what». Делать мы это будем при помощи генератора псевдослучайных чисел. Мы будем выбирать одну из трёх точек, поэтому диапазон чисел нам нужно указать, как (0, 3). Напоминаю, что в этом случае у нас будет такой вот полуинтервал [0; 3), и могут выдаваться числа 0, 1 или 2.

(продолжение кода)

```
int x = xa, y = xb
 int count = 0;
while(count < 10)
{
int what = new Random().Next(0, 3);
}
```

После того, как мы определили случайное число, будем выполнять проверку. Если число what = 0, то в этом случае мы должны будем в икс положить середину отрезка (считается это как x + xa на первой вершине, делённое пополам), и аналогичным образом будет для y: (y + ya)/2. Кстати, код можно отформатировать, чтобы он смотрелся немного красивее. Итак, первую точку определили.

(продолжение кода)

```
int x = xa, y = xb
int count = 0;
while(count < 10)
{
int what = new Random().Next(0, 3);
if (what == 0)
{
x = (x + xa) / 2;
y = (y + ya) / 2;
}
```

И таких действий нужно будет проделать несколько. Если what у нас равен 1, то для x будет браться середина отрезка, построенного с учётом вершины «xb»; y = (y + yb) / 2. Соответственно, меняем третью вершину: проделываем те же действия для числа 2 (и точки (xc, yc).

```
{
int what = new Random().Next(0, 3);
if (what == 0)
{
x = (x + xa) / 2;
y = (y + ya) / 2;
```

```
}
if (what == 1)
{
x = (x + xb) / 2;
y = (y + yb) / 2;
}
if (what == 2)
{
x = (x + xc) / 2;
y = (y + yc) / 2;
}
```

После того как мы определили середину, нужно её нарисовать сначала при помощи установки курсора в нужную позицию через команду Console.Set.CursorPosition, позиция у нас будет (x, y). И после этого Console.WriteLine(“+”). Сделано.

Определяем значение оператора цикла

Теперь нам нужно определить, сколько именно это действие будет выполняться. Мы указали 10, но чтобы наш цикл не ушёл в бесконечное действие, мы должны будем сказать count: «пожалуйста, на каждой итерации становись на единичку больше» — count = count + 1;.

От себя отмечу, что программисты придумали специальные конструкции, и чтобы не писать каждый раз «count = count + 1», её можно переписать, например, как count+=1, или использовать короткую запись count++.

```
Console.Set.CursorPosition(x, y);
Console.WriteLine(“+”)
count++
}
```

Попробуем запустить и посмотреть, что получится. Вводим dotnet run и получаем результат. Как вы можете видеть, 10 точек — очень мало для нашей фигуры, и красивой картины здесь не наблюдается.

Запускаем приложение

Сделаем количество точек немного больше, и будем повторять действие не 10, а, например, 10 тыс. раз (строка 23: while(count < 10000)). Перезапускаем через dotnet run и смотрим на результат.

У нас получилась фигура, очень напоминающая фрактальное изображение треугольник Серпинского. Здесь математика и информатика, можно сказать, слились воедино, ещё и программирование добавили. Очень хорошо и быстро визуализировали математически красивый объект.

[1:18:13]

Заключение

Итак, вы узнали ещё один оператор цикла — конструкцию while, для которой обязательно нужно использовать счётчик. Пожалуйста, помните об этом. На семинарских занятиях мы ещё потренируемся и решим задачу про двух друзей и собаку, потому что алгоритм и все формулы у нас уже есть, осталось только превратить это в код. Если хотите, можете до семинара попробовать самостоятельно (алгоритм решения задачи вы можете видеть на экране). А на семинаре можно будет уже предметно побеседовать и, быть может, обсудить ваши первые ошибки. От себя отмечу: на первых этапах, пока вы пишете код, вы будете для себя как разработчиком, человеком, который что-то создаёт, так и тестировщиком, который эти ошибки должен будет выявлять и проверять.

И есть одна интересная шутка, связанная с тем, что, если тестировщики ищут ошибки, получается, что программисты их создают. Так что вы, на первых этапах вы будете создавать очень много ошибок. Пусть вас это не расстраивает. Ничего страшного в этом нет. Все допускают ошибки, в том числе и я, когда пишу код. И это необязательно должны быть сложные примеры: даже сегодня я допустил несколько ошибок, и здесь нет ничего страшного.

Подведём небольшой итог. Сегодня мы узнали:

- Операторы ввода и вывода: Console.Write для печати чего-либо в консоль или терминал и Console.ReadLine, необходимая для того, чтобы что-то считать.
- Типы данных: целочисленных, вещественных, строковых, логических, где мы проверяли, например, условия «пока число count меньше 100», или «если username = Маша». Здесь было и то самое логическое условие bool.
- Конструкции ветвления If-Else и конструкции цикла while.

На этом я с вами прощаюсь и до встречи на семинарах.