Материалы курса.

WEEK 0

ДЭВИД МАЛАН: Это CS50, введение Гарвардского университета в интеллектуальные предприятия информатики и искусства программирования. Меня зовут Дэвид Малан, и если вы находитесь среди тех, кто думает, почему я в классе компьютерных наук, то поймите, что я тоже чувствовал то же самое. На самом деле, на первом курсе я не совсем смел, чтобы учиться этому классу или информатике в целом, и это было в основном потому, что я был напуган этим. Я немного нервничал. Это чувствовало себя хорошо вне моей зоны комфорта. И я действительно не знал в конце дня, что это было на самом деле. Но поймите, если вы тоже чувствуете это немного, или даже если вы относитесь к тем, кто более комфортно разбирается в информатике или программировании, осознайте, что Это так много пробелов, которые мы можем заполнить по пути, чтобы в конечном итоге, в конце семестра, каждый почувствовал себя на одной странице. И до тех пор, будьте уверены, что 68% людей, сидящих слева и справа от вас, сзади и спереди, никогда не проходили курс CS, который вполне может быть тем демографическим, в который вы вписываетесь. Но поймите также, что с такой удивительной структурой поддержки, с таким большим количеством рабочих часов, секций, материалов и не только, вы понимаете, что главное в этом курсе не так много, как в конечном итоге по сравнению с одноклассниками на 10-й неделе, нашей последней неделю, но где вы в конечном итоге по отношению к себе на нулевой неделе. И действительно, это то, где мы сейчас находимся. И, как выясняется, компьютерные ученые начинают считать с нуля. И так в течение следующих 11 недель, мы выведем вас из числа тех, кто менее удобен или, возможно, находится где-то между менее комфортным и более чувствующим, к чувству гораздо более комфортного, уверенного и способного, чем это. Но чтобы попасть туда, нам нужно понять, что такое компьютерная наука. И это было то, чего я не понимал, пока не ступил в такую ​​комнату. И я осмелюсь сказать, что мы можем перенести информатику именно в эту картину. Информатика - это решение проблем. И я знаю, что курсы средней школы обычно рисуют обманчивую картину, о которой речь идет только о программировании, а люди, опустившие голову в компьютерную лабораторию, довольно антисоциально работают над кодом, но реальность такова, что все решает проблемы, и очень часто, решение проблем совместно либо лично, либо используя код, программы, которые другие написали в прошлом. И что значит решить проблему? Ну, вам нужны входные данные. Итак, есть проблема, которую вы пытаетесь решить. Это вход. И вы хотите выходной. Вы хотите решение этой проблемы. И вот что-то вроде секретного соуса компьютерной науки будет всем в этом пресловутом черном ящике посередине в течение следующих нескольких недель, когда вы начнете понимать, что конкретно вы можете с этим сделать. Но для того, чтобы начать решать проблемы, нам нужно просто решить, как группа, как мы будем представлять эти проблемы и что может быть проблемой. Ну, в этой комнате целая группа людей. Если бы мы хотели принять участие или подсчитать количество людей в этой комнате, мне, возможно, нужно было бы отслеживать, сколько людей я вижу. Но как мне представить количество людей, которых я вижу? Ну, я могу сделать это как в старой школе, и я могу просто взять кусок мела или еще много чего и сказать, хорошо. Я вижу 1, 2, 3, 4, 5. Я могу делать такие маленькие стилистические соглашения, чтобы сэкономить место и напомнить себе. 6, 7, 8, 9, 10 и т. Д. Или я могу, конечно, сделать это самостоятельно. Итак, 1, 2, 3, 4, 5 и так далее. Но очевидно, как высоко я могу рассчитывать только на одну руку? Итак, 5 вы могли бы подумать, но это только потому, что мы не очень задумывались над этой проблемой. Получается, что только этими пятью пальцами, не говоря уже об этих пяти пальцах, я могу считать намного выше, потому что, в конце концов, система, которой я пользуюсь хэш-метками на плате или просто сейчас своими пальцами, просто удерживает мои пальцы на самом деле. Но что, если я на самом деле учел порядок своих пальцев и как бы их переставил, так что на самом деле узоры пальцев представляют количество людей в комнате, а не просто присутствие пальца вверх или вниз. Другими словами, это может остаться нулевым. Это все еще может быть один. Но что, если два не только это, очевидное? Но что, если это только это? Подняв только один мой второй палец. Что если тогда три это? Таким образом, у нас есть 0, 1, 2, 3. Это приведет нас к четырем несколько обидно. Но если мы начнем прыгать вперед до пяти, я мог бы теперь переставить этот палец и этот палец вверх. И если я хочу теперь представлять шесть, я мог бы сделать это. А сейчас семь. Другими словами, я уже выразил гораздо больше паттернов, и если мы продолжим это делать, Я думаю, что на самом деле я могу изобразить больно, может быть, как 32 разных образца, и, следовательно, 32 разных человека, только на моих руках. Или 31 человек, если я начну считать с нуля. Так что это ... какие отношения и как мы сюда попали? Что ж, получается, что компьютеры немного упрощены, как наши руки здесь. В конце концов, ваш компьютер подключен к стене или у него есть батарея, поэтому он либо имеет, либо не имеет электричества. В конце концов, это физический ресурс, который управляет этими вещами, нашими телефонами и всеми технологиями сегодня. Так что, если есть электричество или нет, такие карты прекрасно подходят для пальцев или да пальцев. И действительно, компьютеры, как вы, наверное, знаете, говорят только на каком языке? Какой алфавит, так сказать? Да уж. Binary. Би означает два. И действительно, это относится к тому факту, что в двоичном коде в компьютерах у вас есть только две цифры - ноль и одна. Мы, люди, конечно, имеем 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, и тогда мы можем объединить их, чтобы считать еще выше. Но у компьютеров только 0, 1, и все тут. Потому что, в конце концов, на самом деле существует прямое сопоставление между отключением питания и включением нуля или включением питания и тем, что оно равно единице, или несколькими электронами или чем-то, что течет от вашей батареи или от стены. Вот почему компьютеры, как правило, говорят только в двоичном формате, потому что в конце концов они просто очень четко отображают то, что в первую очередь приводит их в действие. Но как это на самом деле полезно? Если у компьютеров есть только нули и единицы, как они могут сделать что-нибудь полезное? Хорошо, подумайте о нашем человеческом мире, где у вас может быть этот образец символов. Это десятичное число, dec означает 10, потому что у вас от 0 до 9. И это, конечно, 123. Но почему? Если вы не задумывались об этом в течение достаточно долгого времени, на самом деле это просто набор из трех символов, одной, двух и трех фигур, или глифов, на экране. Но мы, люди, начиная с начальной школы, начали придавать значение каждому из этих чисел, верно? Если вы вспомните, это столбец единиц, столбец десятков, столбец сотен и так далее, и мы могли бы продолжать. И так, почему этот шаблон - один, два, три - означает 123? Ну, это потому, что все мы вроде как интуитивно в наше время просто быстро в нашей голове делаем 100 раз 1 плюс 10 раз 2 плюс 1 раз 3, что, конечно, дает нам 100 плюс 20 плюс три, а затем число, которое мы знаем математически как 123 Но мы все делаем это так быстро, ты не больше не думаю об этом. Ну, компьютеры работают в основном так же. У них не так много цифр - от 0 до 9 - как у нас. У них только нули и единицы. И поэтому, если они будут хранить значения, вы будете видеть только нули и единицы на экране, но эти нули и единицы означают разные вещи. Вместо того, чтобы иметь одно место, десятки, сотни, у них будет одно место, место двойок, место четверок, а затем восьмерки и 16 и далее. Теперь почему? Ну, а 1, 10 и 100, получается, что это степени от 10. 10 до 0 - это технически 1. 10 к 1 - это просто 10. 10 к 2 - это 100. И поэтому у вас есть единицы, десятки, сотни, тысячи , и так далее. Компьютеры, очевидно, используют степени 2. Не удивительно. Двоичный - два. Так что если у вас есть только единицы, двойки и четверки в качестве заполнителей, если компьютер хранил эти цифры - 0, 0, 0 - этот компьютер, по-видимому, хранит какое число, насколько мы, люди, понимаем его? Ну, вот как компьютер будет хранить ноль. Если компьютер хранит буквально 0, 0, 0, как в нашем человеческом мире, то это тоже 0, но это технически, потому что это 4 раза 0 плюс 2 раза 0 плюс 1 раз ноль, что, очевидно, равно нулю. Между тем, если компьютер на самом деле хранит не просто, скажем, 0, 0, 0, но вместо этого сохраняет это значение в двоичном формате, то что это означает в десятичном формате? Так вот. А теперь, почему, если мы изменим эти 0 и 1 на это значение, это два? Ну, математически, по тем же причинам. И так раньше у меня было пять пальцев, но если учесть только мои первые три, когда я делал это, держа один палец, я представлял два. И если я хочу представлять три, Напомним, что я поднял второй палец. Итак, причина, которая могла бы хорошо представить три, состоит в том, что все, что я делал со своей человеческой рукой, считало в двоичном виде. И я мог продолжать считать все больше и больше и больше. И поэтому, если у меня есть пять пальцев или пять битов, бит, означающий двоичные цифры, я мог бы подсчитать, оказывается, если мы выполним математические вычисления, до 31, начиная с нуля. Это будет трудно сделать физически, но мы могли бы. Так почему это полезно? В конце концов, компьютер, таким образом, может представлять любое количество значений от 0 до 1, от 2 до 3 до некоторого числа, намного, намного, намного выше, чем это. Все, что ему нужно, это достаточно битов, достаточно нулей и единиц. Ну, что это за биты? Ну, у всех нас есть в наши дни в наших телефонах источники света, например. Таким образом, я мог бы сказать, что это физическое устройство сейчас - может быть, немного сложно сказать - у него есть фонарик, и в настоящий момент оно технически выключено. Но если я включу этот фонарик, используя тем самым часть электричества, теперь я сохраню один. И вот телефон включен. Теперь он выключен. Теперь он включен. И если я увижу ... могу я позаимствовать чей-то телефон очень быстро? Могу ли я? ХОРОШО. И фонарик. Как включить фонарик? Ой. Потряси. Это нормально. ХОРОШО. Спасибо. Ой. Спасибо. ХОРОШО. Так что это здорово. Теперь я могу считать выше. Теперь, это число, которое, если у меня есть две лампочки или два включенных в настоящее время? Да уж. Три. Поскольку у меня есть один, у меня есть один, и у меня есть два, которые, конечно, в конечном итоге равняются трем. И если бы я как-то взял третий телефон, я мог бы посчитать еще выше. Технически, если бы у меня было три лампочки - один, один, один - какой была бы эта ценность? Семь. Потому что это четыре плюс два плюс один и так далее. Большое спасибо за спонтанность. Так почему же это не ведет к ограничениям для нас? Я могу считать в десятичном числе столько, сколько я хочу. Теперь я могу считать в двоичной системе столько, сколько хочу, если у меня достаточно битов. Но как я на самом деле представляю другую информацию? Ну, если я хочу представить что-то вроде письма, как мне туда добраться? Если в компьютерах есть только электричество, и они используют двоичные числа для подсчета, и все-таки они гораздо полезнее, чем просто математика - у них могут быть текстовые сообщения, электронные письма, веб-сайты, видео и многое другое - как мы можем получить от нулей и единиц до букв? Ну, мы ... да. Сожалею. Немного громче. Да уж. Нам просто нужно связать числа с буквами. Другими словами, все люди в этой комнате просто должны в какой-то момент решить, знаете, что? Если мы хотим представить что-то вроде заглавной буквы A, нам просто нужно определиться с набором битов, набором пальцев, который будет представлять A. И оказывается, что люди много лет назад в одностороннем порядке решили, что 65 будет десятичным числом который представляет заглавную букву А. И вы можете догадаться, что заглавная буква B представлена ​​каким десятичным числом? 66. А потом С - 67 и так далее. И есть сопоставление примерно 128 или даже 256 возможных значений для всех клавиш, которые вы можете видеть на типичной клавиатуре для представления букв. Теперь, как компьютер отличает цифры от букв? Ну, просто зависит от контекста. Если вы используете как калькулятор на вашем Mac или ПК или iPhone или Android, ну, компьютер, устройство, только собираюсь узнать контекстуально, позвольте мне интерпретировать этот образец нулей и единиц как фактические числа, чтобы сделать математику. Но если вы используете приложение SMS или приложение сообщений на своем телефоне, вы на самом деле будете находиться в текстовом контексте, и ваш телефон будет интерпретировать тот же шаблон нулей и единиц или лампочек, являющихся отключить или, в конце концов, транзисторы, крошечные аппаратные средства и компьютеры, которые либо включены, либо выключены - это будет интерпретировать эти шаблоны как просто представление буквы. Если вы находитесь в контексте приложения для обмена текстовыми сообщениями, Microsoft Word, Документов Google и т. П., Это полностью зависит от контекста. Система, которую мы, люди, придумали, называется просто ASCII, американский стандартный код для обмена информацией. Имя не интересно, но тот факт, что все мы согласились несколько лет назад, что 65 - это A и так далее, это то, что важно. И так, например, если мы посмотрим на это отображение здесь только первых нескольких букв, что это значит? Если бы я теперь получил текстовое сообщение, и у меня была возможность каким-то образом заглянуть под капот, так сказать, в шаблон нулей и единиц, которые кто-то только что написал мне, и этот шаблон, если я преобразую его в десятичную, технически сказано, скажем, 72 и 73 - так что я получаю целую кучу нулей и единиц. Я занимаюсь математикой, и я понимаю, хорошо, я только что получил 72 и 73, но это текстовые сообщения, и это не просто цифры, которые мой друг посылает мне. Это сообщение. Какое сообщение скорее всего отправит мне мой друг, если он или она отправит 72, а затем 73? Да уж. Привет. ПРИВЕТ. Потому что если вы скользите вперед прямо там, в ASCII это просто отображение между 72 и 73, H и I. Если технически, у сообщения был третий байт, если хотите. Байт, если вы когда-нибудь задумывались, это всего восемь бит. Удобно говорить не с точки зрения единичных битов, где вы не можете считать очень высоко, но с байтом или восемью битами вы можете считать выше. И так получается, если бы я получил третий байт, еще одну последовательность из восьми нулей и один вместе - 33. Как бы мы узнали, что это за сообщение сейчас? Да уж. Таким образом, оказывается, что вы не узнаете об этом, кроме как отгадывать, гуглить или просто входить с этим знанием. Это теперь "Привет!" с восклицательным знаком, потому что 33 происходит, если вы посмотрите вверх, чтобы сопоставить с восклицательным знаком, а также. Теперь, если бы мы на самом деле посмотрели на двоичный код этого, вы бы фактически увидели этот паттерн из нулей и единиц. Вот как вы представляете 72 в двоичном виде. Это вы представляете 73. И это то, как вы представляете 33. И обратите внимание, я использовал только один, два, три, четыре, пять, шесть битов, хотя технически я склонен получать вещи в единицах по 8, единиц байтов , Но почему я не потрудился написать еще одну здесь и еще одну здесь? Имеет ли значение, когда ты пишешь эти вещи? Нет, не совсем. Как и в английском, в нашем человеческом мире, если бы вы написали один, два, три, это 123. Если вы должны были написать 0, 1, 2, 3, это все равно 123. Так что, хотя мы склонны собирать их в кластеры из 8, нам не обязательно писать их, когда мы просто говорим о них. Итак, что мы сделали? Хорошо, позвольте мне представить более красивое слово, теперь известное как абстракция. Абстракция - это просто термин, обычно используемый в информатике, и мы Вскоре в программировании мы увидим, что нужно взять какой-то низкий уровень - например, буквально низкий уровень - детали реализации, например даже мелочи, и понять их в какой-то момент, но затем решить, что это не является полезным концептуальным уровнем для размышления о проблемах. Я действительно не хочу решать проблемы в этом мире мышления в 0 и 1. Я бы предпочел думать о вещах минимально в десятичной или, что еще лучше, в контексте букв, если я на самом деле получаю текст, или даже какое-то другое представление. Таким образом, абстракция заключается в том, чтобы брать довольно низкоуровневые детали и просто упрощать их, чтобы мы могли вести более полезный разговор и никогда больше не беспокоиться о том, откуда исходит электричество. Мы можем просто указать, что мой компьютер может представлять нули и единицы. Следовательно, он может представлять числа. Следовательно, он также может представлять ASCII или буквы. И мы можем двигаться дальше и начать решать более интересные проблемы. Но может показаться, что мы не можем решить все проблемы, потому что на моей клавиатуре здесь, на этой американской клавиатуре, есть целая куча символов, например, 100 или 200, если мы действительно нажмем клавиши shift и option и все такое. Но то, что вы не видите, это довольно общие символы. Что я могу даже не печатать на этой клавиатуре, особенно в международной аудитории? Какие виды символов? Да уж. Что-нибудь с акцентом. Если у вас есть акценты над гласными или другими буквами. Что-то еще? Мне жаль. Умлауты или другие символы над буквами. Да уж. Как символ фунта? Ой. Как символ британского фунта. Конечно. И другие страны тоже. Любое количество азиатских языков. Там так много символов, которые не изображены на этой клавиатуре, и все же как-то, Все мы с международными клавиатурами или телефонами, безусловно, можем выразить себя. Но это потому, что телефоны и компьютеры в наши дни не просто используют ASCII. ASCII буквально использовал всего восемь битов. Технически семь, но потом, в конце концов, действительно восемь. И с восемью битами, если вы действительно делаете математику - если у вас есть восемь битов или восемь пальцев, вы можете переставить их только 256 возможных путей, то есть вы можете представить только 256 символов, используя ASCII с числами под капюшон, и этого недостаточно, чтобы представить так много разных символов, как перечисленные здесь. Вы не можете представить акценты, которые вы, тем не менее, можете печатать на своих компьютерах Mac и PC, и вы, конечно, не можете печатать эти вещи, которые очень популярны. Которые, даже если они картинки, на самом деле они просто персонажи. Поскольку, оказывается, несколько лет назад мир решил, что восьми бит недостаточно. Давайте начнем с того, что называется Unicode, где вы на самом деле используете один, два, три или даже четыре байта. Итак, восемь или 16 бит, 24 или даже 32 бита для представления символов. И теперь у нас есть возможность представлять тысячи или даже миллионы персонажей. И, честно говоря, результат этого огромного количества отчасти объясняется тем, что в наши дни так много всего этого. И они просто продолжают делать больше, потому что у нас так много разных номеров. Таким образом, Unicode часто является специфической версией UTF-8, которую мы увидим в ближайшее время. Но позвольте мне задать этот вопрос здесь. Это плачущее лицо от радости, я думаю, это называется. Так получается, согласно Apple или iOS, это самый популярный смайлик, который, по крайней мере, люди iPhone посылают друг другу. Поэтому, когда вы получаете это, хотя, если мы действительно сможем извлечь из этого удовольствие, какую модель бит вы на самом деле получаете от своего друга? Он или она явно пытается выразить какую-то эмоцию, но на самом деле то, что посылает вам ваш друг - десятичное число 128,514. Или действительно, если вы смотрели на нули и те, которые приходят к вам по Интернету или радиоволнам, вы получаете эту схему нулей и единиц, которая вряд ли радостна или едва ли описательна, но все, что делает ваш телефон или компьютер, видит это образец битов, ища это как маленькая шпаргалка, и говоря, о. Всякий раз, когда я вижу этот образец битов в контексте текста, такого как текстовые сообщения, я должен фактически отображать его как эту картинку. Сейчас, на этой картине много желтого и других цветов, но как нам вообще туда добраться? Что ж, получается, что та же самая схема чисел - 72, 73, 33 - которая, чтобы быть уверенной, минуту назад, что означала? Привет. В контексте текстовой программы, такой как Microsoft Word, Google Docs, текстовые сообщения, это означает «привет». Но что, если вы увидели эту же последовательность байтов - и снова мы могли бы нарисовать нули и единицы, но это больше не интересно, поэтому мы собираемся абстрагироваться на десятичном уровне. Если вы получили такой же шаблон нулей и единиц или чисел в контексте, например, Photoshop, браузера или какой-то программы для работы с фотографиями, возможно, было бы более разумно интерпретировать их не как текст, а как образы, какие-то цвета. , Ну, оказывается, в мире есть другая система - вы могли видеть эту аббревиатуру раньше - она ​​называется RGB - красная, зеленая, синий. И это просто способ стандартизации людей много лет назад, что вы знаете, что? Если мы хотим представить точку на чьем-то экране, иначе известную как пиксель крошечный квадрат на экране вашего телефона, вашего ноутбука или даже телевизора, мы будем использовать три байта - один байт, чтобы указать сколько красного должно быть в этом конкретном пикселе, еще один байт, чтобы указать, сколько зеленого должно быть объединено с красным, чтобы сформировать этот пиксель, а затем еще один байт, третий, чтобы показать, сколько синего нужно объединить с этими двумя другими цветами, чтобы сделать новый цвет все вместе. Так что это похоже на комбинирование красок, за исключением того, что в данном случае это скорее волны света, чтобы получить определенный цвет, используя только красный, зеленый и синий в качестве палитры. И поэтому, если бы мы увидели этот красный, зеленый, синий узор и сказали, знаете что? Дай мне 72 красных, 73 зеленых и 33 синих. Если общий возможный диапазон, на который я ссылался ранее, составляет от 0 до 256 или технически от 0 до 255, если вы начинаете считать в компьютерных науках с нуля, это похоже на среднее количество красного, среднее количество зеленого и совсем немного синего цвета, если диапазон колеблется от 0 до 255. Так что, если вы объедините эти три вещи вместе, и вы хотите знать, какого цвета вы получаете - да. Так что это светло-желтый, который выглядит следующим образом. Таким образом, если компьютер хранит одну точку на экране, которая оказывается желтой, то на самом деле компьютер хранит не эту точку физически, а комбинацию из трех байтов - сколько красного, сколько зеленого, сколько синего должно дисплей компьютера в этой конкретной точке. Поэтому, если мы посмотрим на это плачущее лицо радости и увеличим или увеличим его здесь, на самом деле вы можете видеть, как он начинает пикселизировать, так сказать, где вы начинаете видеть точки. Если я добавлю немного больше, теперь вы действительно можете начать видеть точки на экране. И если я пойду еще дальше, вы сможете увидеть крошечные квадраты, составляющие это изображение, большинство из которых на уровне масштабирования желтые, но куча которых черная, кучка которых светло-черная или темно-желтая. И это то, что в конечном итоге составляет этот образ. Так что это означает, что если вы подсчитаете все пиксели на экране, а затем умножите его на один, два, три байта, это будет то, сколько байтов, килобайт или мегабайт, если вы слышали эти термины, будут сохранены. на вашем компьютере только для представления изображения. Итак, мы перешли от электричества к нулю, так сказать, к нулям и единицам, к десятичному знаку, теперь к цветам. Ну, с цветами, Вы можете получить изображения. Что происходит после изображений? Ну, мы все смотрели видео или фильмы, безусловно, в цифровом виде в эти дни. Ну что такое фильм или видеофайл? Как это может быть реализовано? Скажи это немного громче. Да уж. Это коллекция изображений. Если вы когда-либо слышали о кадрах в секунду - например, фильмы, как правило, имеют скорость 24 кадра в секунду или 30 кадров в секунду - это просто означает, что в типичном фильме каждая секунда показывает вам 24 или 30 изображений в секунду, и они ' Вы летите так быстро, что вы даже не замечаете, что просто смотрите последовательность статичных изображений. Это как в детстве, если у вас когда-нибудь была одна из тех бумажных перевернутых книг, в которых тонны рисунков, и, пролистывая страницы, вы видите, как что-то движется, но это… Просто потому, что ваши глаза просто так быстро видят маленькие снимки чего-то движущегося на бумаге. Это все видео файл на самом деле. Так что, если у вас есть iPhone, и вы когда-либо играли с этими анимодзи, так сказать, ну, все это небольшие видеофайлы, состоящие из множества, большого и большого количества изображений, которые вы сохранили на своем телефоне или отправили кому-то другому. , И если мы просто думаем, что сейчас, мы находимся в момент видео, но это нормально. Видео это просто куча картинок. Изображения - это просто пучки цветов. Цвета - это просто образцы битов. И биты, в конце концов, являются просто результатом электричества, поступающего в мою машину, или включения и выключения транзисторов. Как будто у нас вдруг появилась вся эта история, но никому из нас никогда не понадобится по-настоящему думать о двоичном коде в контексте видео, потому что видео - это просто абстракция поверх групп изображений, а изображения - просто абстракция над пикселями и так далее. Таким образом, мы можем продолжать рисовать эту иерархию, которая просто позволяет нам говорить о вещах на более полезном уровне, и причина, по которой мы провели этот разговор, заключается просто в том, что нам нужен был способ представления входных и выходных данных для проблем. Позвольте мне остановиться на секунду, чтобы посмотреть, есть ли вопросы. Вообще ничего? Нет? Отлично. Так что внутри этого черного ящика? Что ж, получается, что именно здесь начинает происходить действительно интересная работа и начинает приходить мысль. Это общеизвестные алгоритмы, пошаговые инструкции для решения какой-то проблемы. И некоторые из вас могли бы решить эту проблему раньше, в цифровом или текстовом формате, но, конечно, если у вас есть контакты в вашем телефоне в настоящее время, и у вас есть куча друзей и семьи, скорее всего, они в алфавитном порядке по имени или фамилии. И у вас есть автоматическое заполнение в эти дни, но на самом деле это просто длинный список имен и номеров. Это не все, что отличается от прошлой реализации той же самой проблемы, которая была этим устройством здесь, телефонной книгой. Теперь в этой телефонной книге может быть наш друг, скажем Майк Смит, чья фамилия начинается с S. И я, конечно, мог бы, если попытался найти Майка Смита, начать с просмотра первой страницы, второй страницы, третья страница, четвертая страница, и, в конце концов, мы надеемся найти Майка Смита. Действительно ли этот алгоритм, этот пошаговый процесс, подходит для поиска кого-то вроде Майка Смита? Да уж. Это правильно. Это' глупо и медленно, возможно, потому что это займет целую вечность в телефонной книге такого размера, но это правильно, потому что, если Майк здесь, я, фактически, найду его. Но я мог бы сделать это лучше. Я мог сделать это как бы два за один раз. Итак, два, четыре, шесть, восемь, 10 - или несовершенно - 10, 12, 14. Это быстрее? Очевидно, что это идет в два раза быстрее. Это правильно? Нет. Почему это не правильно? Да уж. Я могу скучать по нему, верно? Майк просто случайно может в конечном итоге оказаться зажатым между двумя страницами, и мне не повезло просто скучать по нему. Теперь это исправимо? Да уж. Я, наверное, могу, как только я попаду, например, в SN или в раздел T. Могу просто сказать, что я зашел слишком далеко для Майка. Позвольте мне просто сдвинуть назад одну или несколько страниц. Так что это поправимо. И пока я сэкономил время, пролетев через это в два раза быстрее, могу ли я хотя бы позволить себе потратить еще несколько шагов в самом конце, чтобы найти Майка Смита? Но никто из нас не собирается этого делать. И наши устройства Apple и устройства Android, безусловно, не делают этого для эффективности сегодня. Скорее всего, большинство из нас собирается сделать то, что найти кого-то в любой книге, как эта? Да уж. Открыто примерно до середины или, возможно, смещаемся к концу, потому что S находится после середины. Но вы знаете, я нахожусь в середине телефонной книги здесь. И теперь, если я знаю, что Майк в S и, следовательно, здесь, откуда я знаю, что он не? Он не в начале, и я могу буквально разорвать такую ​​проблему пополам, образно и буквально отбросить половину проблемы, и у меня останется та же самая проблема, но она вдвое больше. Я пошел как ... независимо ... 1, 000 страниц до 500 страниц, и теперь я могу повторить этот алгоритм. Я смотрю вниз. Я слишком далеко. Я сейчас в разделе Т ХОРОШО. Я снова могу разорвать проблему пополам, выбросить эту половину, вытащив байт на 500 страниц, байт на 250 страниц, и теперь у меня остается только 250 страниц. И обратите внимание, как быстро я сюда попал. Первые два алгоритма дали мне от 1000 до 999 до 998 или от 1000 до 998 до 996. Но здесь я перешел от 1000 до 500 до 250. Такое ощущение, что мы приходим сюда. И действительно, если я продолжу повторять этот процесс, надеюсь, у меня останется только одна страница книги, на которой Майк или нет, и в этот момент я позвоню ему. И вот этот алгоритм, который честно использует, вероятно, всю нашу интуицию и многое из того, чем будет заниматься программирование, думает о такой проблеме, выяснить, как разделить и победить его, а затем выразить себя таким образом, чтобы компьютер мог решить эту проблему для вас. И просто нарисуйте картину, насколько лучше этот алгоритм, ну, если это просто очень абстрактная диаграмма, где у нас есть по вертикали или по оси Y, сколько времени требуется для решения проблемы, и по горизонтали Ось, насколько велика проблема - поэтому чем дальше вы идете этим путем, тем больше страниц в проблеме, тем больше страниц в телефонной книге. И чем выше вы поднимаетесь сюда, тем больше секунд или переворачиваний страницы это займет. Этот первый алгоритм, так сказать, похож на линейный наклон, потому что для каждой дополнительной страницы в книге мне может потребоваться еще одна секунда. Вплоть до. Вплоть до. Это просто отношения один на один со страницами. Второй алгоритм, если я его построю, где я м. пролетая в два раза быстрее, как будет выглядеть эта линия? Да уж. Это будет выглядеть ниже, чем этот. Это все равно будет прямой линией, потому что теперь есть отношения два к одному, но если у вас есть телефонная книга, которая имеет столько страниц, и в первом алгоритме это заняло много времени, здесь, ну, в общем, Во втором алгоритме потребуется вдвое меньше шагов, плюс или минус или два, если вам нужно немного сдвинуть назад. Но этот третий алгоритм мы будем называть логарифмическим. Если n - это количество страниц в телефонной книге, первый алгоритм, в самом худшем случае, может занять все n страниц, чтобы найти Майка Смита. Второй алгоритм будет проходить вдвое меньше шагов, потому что я прохожу его по два за раз. Но третий алгоритм будет выглядеть и чувствовать себя так. Это' будет изогнутым и очень медленно подниматься, подниматься и подниматься, следствием чего является то, что Verizon или телефонная компания удвоит количество страниц в телефонной книге в следующем году, потому что Кембридж и Сомервилл слились воедино в телефонной книге, и теперь у нас есть 2000 страниц. Ну, а сколько еще шагов предпримет мой третий алгоритм? Только один. Потому что я могу укусить 1000 страниц от проблемы с этим умным алгоритмом, в то время как мои первые два алгоритма будут брать это по одной или только двум страницам за раз. То есть мы должны значительно увеличить размер этой проблемы только на то, чтобы количество секунд или поворотов страницы заметно увеличилось. И так как мы начинаем изучать программирование, это, опять же, будет использовать эту интуицию для того, чтобы на самом деле решать проблемы и код более эффективно, чем мы могли бы без этой интуиции. Итак, давайте формализуем это сейчас. Так что это был довольно интуитивный способ разделения и преодоления проблемы. Просто имело смысл пойти посередине, порвать его, затем перейти к другой половине или другой половине и снова порвать его, и так далее. Но компьютер, даже такой крутой, как Alexa и Google Home, и все это, вы не можете просто поговорить с ними как с другим человеком и заставить их выполнять вещи правильно. Я изо всех сил пытаюсь заставить Сири установить таймер на моем телефоне. Таким образом, мы еще не совсем готовы, поэтому мы все еще находимся в том возрасте, когда мы должны быть очень точными с компьютерами, голосовой активацией или иным образом, и, таким образом, вводим псевдокод на данный момент. Псевдокод не имеет формального определения. Это просто способ сказать, что используйте английский, такой как синтаксис или любой разговорный язык, и просто выражайте себя лаконично и правильно, чтобы компьютер, робот или даже другой человек могли понять, что вы пытаетесь сказать. Итак, здесь, я предлагаю, алгоритм, написанный на псевдокоде, английском, похожем на синтаксис, который просто передает мою точку зрения. И я мог бы написать это любым количеством способов. Я пронумеровал шаги от нуля до, просто для обсуждения, но, похоже, это отражает то, что я там делал. Поднимите телефонную книгу. Откройте до середины телефона. Посмотри на имена. Если Смит среди имен, позвони Майку. Иначе, если Майк Смит находится в начале книги, перейдите налево, особенно в середину левой половины книги, а затем вернитесь к шагу два. Потому что на самом деле, я просто делал то же самое снова и снова, и причина, по которой я не делал это вечно, заключалась в том, что каждый раз, когда я повторял себя, открывая и разрывая, я уменьшал проблему. И я могу только несколько раз уменьшить проблему некоторого фиксированного конечного размера, пока не получу только одну страницу, и поэтому, если я продолжу эту логику, смотря вправо или влево, или просто выйду, если я вообще не найду Майка на последней странице это, кажется, более точно отражает этот код. Что ж, давайте на самом деле извлечем из этого несколько понятий, а затем начнем применять их к реальному коду. Я бы сказал, что здесь желтым цветом выделены все глаголы или действия. Эти функции, как мы собираемся их вызывать, в этом алгоритме. Функция - это просто определенный шаг, конкретное действие, которое вы предпринимаете, чтобы что-то сделать. И вот в желтом здесь - взять, открыть, посмотреть, позвонить, открыть, бросить все действия или глаголы. Отныне мы будем называть их функциями. Между тем, выделено желтым здесь - если, еще, если, еще если, еще. Это своего рода начало задавать вопросы. Как их можно назвать, если у вас есть знакомство? Да. Оказывается, многие языки программирования, если вы видели раньше, будут называть эти условия. Это ветки или вилки на дороге. Если это правда, иди этим путем. Иначе, может быть, пойти другим путем, или, возможно, третье или четвертое направление в целом. Между тем, если мы действительно посмотрим на эти выделенные фразы - если Смит находится среди имен или если Смит находится в книге раньше, или Смит позже в книге - это конкретные вопросы, которые мы задаем, чтобы принять это решение. Они известны как логические выражения, названные в честь джентльмена по фамилии Бул несколько лет назад. Таким образом, логическое выражение - это просто вопрос с ответом «да» или «нет», истинно ложным ответом, если хотите, с одним нулевым ответом. И это хорошее отображение того, в чем компьютеры действительно хороши. Таким образом, в определенных условиях у вас есть логические выражения, чтобы решить, на какой развилке вы хотите пойти. И, наконец, выделенный желтым цветом здесь, вернитесь к шагу 2 в нескольких местах. Это вызывает некоторый цикл или цикл, который говорит компьютеру делать что-то снова, снова и снова. Короче говоря, у нас есть эти строительные блоки уже концептуально. И, оказывается, теперь мы можем начать переводить их на реальный язык программирования. Первый из языков, которые мы представим в CS50, называется Scratch. Оказывается, это не текстовый язык, как в моем английском псевдокоде, но Графика и вещи выглядят как кусочки головоломки, которые вы можете перетаскивать, и они соединяются, если это логично. И на самом деле, некоторые из вас, возможно, играли в эту игру в детстве или даже совсем недавно, потому что на самом деле она нацелена, как правило, на учащихся, как после школьных программ, которые просто хотят выучить более методичное, более алгоритмическое или вычислительное мышление. И мы собираемся использовать его, чтобы исследовать не только эти строительные блоки, но и несколько других. Оказывается, на других языках, которые мы изучим в CS50 и более поздних версиях, это такие языки, как C, к которым мы фактически перейдем так же быстро, как и на следующей неделе, чтобы затем перевести то, что мы делаем на этой неделе в Scratch, на следующую неделю в C. И в таких языках, как Python, JavaScript и SQL, которые мы также рассмотрим, есть ли у нас другие возможности - способность хранить данные в переменных, так сказать, использовать потоки, что означает, что компьютер должен выполнять несколько действий одновременно, события, то есть прослушивать происходящие события, например щелчок по странице или человек печатает или даже что-то говорит. Мы сможем делать все то, что вы считаете само собой разумеющимся в ваших собственных телефонах. И мы сделаем это сначала с помощью этого парня. Так что это Scratch, кот по умолчанию, который поставляется с этим языком программирования из медиа-лаборатории MIT. И с помощью Scratch мы можем начать программировать его, чтобы он двигался вверх, вниз, влево, вправо, что-то говорил, что-то произносил и другие команды вместе. На самом деле, позвольте мне пойти дальше и переключить контексты здесь, чтобы показать вам самое первое, что я когда-либо писал в Scratch. Это было в тот день, когда я учился в аспирантуре, и MIT только что изобрел Scratch. Позвольте мне пойти дальше и открыть это. И я назвал это Оскаром Времени. И если бы мы могли пригласить добровольца на мгновение. Вы должны чувствовать себя комфортно на сцене и в Интернете. Как насчет здесь, в белой рубашке? Сначала я увидел твою руку. Давай вниз. Так что это время Оскара. Это реализовано на языке Scratch. И в конце концов, все, что находится под капотом этой программы, это функции, циклы и условия, а также некоторые другие из этих концепций. Привет. Как вас зовут? АВИВА: Авива. ДЭВИД МАЛАН: Авива. Дэвид. Рад тебя видеть. Давай сюда. И через мгновение я собираюсь пойти дальше и нажать зеленый флаг в верхнем левом углу, который будет играть в эту игру. И мы' увидим на экране инструкцию. [ВОСПРОИЗВЕДЕНИЕ МУЗЫКИ] ОСКАР: (поет) О, я люблю мусор. Все грязное, грязное или пыльное. Все рваное, гнилое или ржавое. Да, я люблю мусор. Если вы действительно хотите увидеть что-то дрянное, посмотрите на это. У меня здесь кроссовки, которые изодраны и изношены. Это все дырявые и кружева порваны. Подарок от мамы на день моего рождения. Я люблю это, потому что это мусор. О, я люблю мусор. Все грязное, грязное или пыльное. Все рваное, гнилое или ржавое. Да, я люблю мусор. Вот еще несколько гнилых вещей. У меня есть газета 13 месяцев. ДЭВИД МАЛАН: Хорошо. Все, аплодируйте Авиве за то, что она пришла. Спасибо. Вот. Aviva. [Аплодисменты] Небольшой стресс CS50. Так что достаточно сказать, если вы устали от этой песни, Подумайте, как я устал через восемь часов при отладке и сборке этой программы. Но подумайте, что это мы только что видели. Это интерактивная игра, и все анимированное и играет музыка. Но если вы сосредоточитесь на том, чтобы разложить, так сказать, эту программу на простые базовые блоки, это всего лишь большая абстракция над некоторыми функциональными элементами более низкого уровня. Как этот мусорный бак здесь. На данный момент это просто картинка, и время от времени, как только Авива бросает что-то в мусорное ведро, поднимается крышка и выходит Оскар, он что-то говорит, а затем возвращается обратно. Но эта анимация супер упрощенная. Это была просто последовательность из 1, 2, 3 или около того изображений, отображаемых и затем возвращающихся вниз, чтобы создать иллюзию анимации. Между тем, каждый раз, когда Оскар что-то говорил, это отслеживало ее счет в том, что… называется переменной. В алгебре у вас есть x, y и z, но в программировании у вас та же идея, но, как правило, более полезно называть их более наглядно, как ваш счет. И поэтому в этой игре, вероятно, есть переменная, называемая счетом, которая просто отслеживала, сколько раз Aviva бросала что-то в корзину. Между тем, сам мусор, обувь и газета - и даже больше вещей, в конце концов, случаются - падали с неба в случайных местах, и это потому, что я запрограммировал игру так, чтобы она начала мусор здесь или здесь, просто чтобы сделать его немного более сложным, как игра началась. И на самом деле, вещи начинают падать все быстрее и быстрее, как в обычной игре, становясь все сложнее и сложнее. Итак, как мы можем получить что-то подобное? Что ж, позвольте мне пойти дальше и открыть Scratch себя и представить окружающую среду. Итак, в Scratch у вас есть три основных области. И это веб-интерфейс, и поэтому вы можете сделать это на любом компьютере. А в левой части у вас есть те кусочки головоломки, о которых я говорил ранее. Все эти кусочки головоломки соответствуют функциям, циклам, условиям или переменным, вещам, которые мы видели раньше, и я собираюсь перетащить их в середину, чтобы соединить их и написать мою программу, которую мы будем сделать в мгновение Тем временем Скретч живет на этой стадии, в этом мире, где он может двигаться вверх, вниз, влево, вправо. Вы можете изменить то, как выглядит Царапина. Вы можете добавить других персонажей, также называемых спрайтами, чтобы одновременно происходило несколько событий. И, конечно, вы можете полноэкранный режим. И вот игра «Оскар Тайм» минуту назад была на самом деле целой кучей спрайтов. Мусорный бак Оскара был один. Каждый кусок мусора был другим спрайтом. Газета была спрайтом и так далее. Таким образом, каждая из них представляла собой отдельные программы, работающие параллельно. Итак, давайте заставим его что-то сделать. Оказывается, что если я прыгну вниз, скажем, к событиям, я увижу один из самых мощных блоков с самого начала, то есть когда щелкнет зеленый флаг. Именно так я начал игру с Aviva, щелкнув прямо над миром Скретча этот зеленый флаг. И если я хотел остановить это, как я сделал, вы можете щелкнуть красный знак остановки, чтобы сказать стоп. Между тем, зеленый флаг я могу постоянно слушать, перетаскивая этот кусочек головоломки. Когда я нажимаю зеленый флажок, что я хочу сделать? Хорошо, позвольте мне пойти посмотреть. И это просто разные категории. И мы можем прокручивать все разноцветные блоки, но они в основном делают то, что говорят. Я собираюсь пройтись по взглядам, где я знаю, что там будет блок, который называется сказать, и я собираюсь пойти дальше и напечатать наиболее каноническую вещь по информатике - привет, мир - в этой коробке. Итак, обратите внимание, что сами функции могут фактически принимать входные данные, а входные данные для этой функции, скажем, будут привет миром. Если я сейчас подойду к зеленому флагу и нажму на него - привет, мир. Отлично. Так что не все так сложно. Не все так интересно. Но на самом деле это сделано, и моя программа действительно именно такая. Ну, как я могу сделать это немного интереснее? Просто сказать, привет мир все время не так уж и убедительно. Ну знаешь что? Дай мне подумать. Позвольте мне отменить это. Позвольте мне прокрутить до ощущения. И заметьте это. Функции также могут принимать данные от человека, а функции могут возвращать вам значение, так называемое возвращаемое значение. Итак, этот блок здесь, спросите что-то - по умолчанию, он говорит, как вас зовут и вес - это еще одна функция, встроенная в Scratch, которая позволяет мне делать это. Так что я собираюсь пойти дальше и перетащить это сюда, и я позволю этому сказать, как тебя зовут? Теперь обратите внимание, что под этим блоком находится специальный блок, независимо от того, какой блок он возвращает. Так что ответом будет то, что наберет человек. И если я хочу сейчас сказать, что набрал человек, позвольте мне снова взглянуть. Иди сказать. И обратите внимание, что эти блоки являются своего рода магнитными. Они хотят соединиться. Так что я собираюсь пойти дальше и отпустить туда. И если я вернусь к восприятию и получу ответ, заметьте, что даже если Она не совсем того же размера, она будет расти и заполняться, и теперь я могу попросить мою программу спросить пользователя, как его зовут, и затем сказать, что это будет за ответ. Итак, позвольте мне пойти дальше и остановиться и снова нажать кнопку воспроизведения. Обратите внимание, что меня спрашивают, как меня зовут, так что позвольте мне пойти дальше и напечатать «Дэвид». Войти. ХОРОШО. Это немного странный способ приветствовать кого-то. Дэвид. Так что было бы неплохо почистить это немного. Так ты знаешь что? Я знаю это только из того, что тыкала раньше. Не все это на первый взгляд очевидно. Но оказывается, что под операторами, категорией, есть эта вещь - присоединяйся к яблоку и банану. Которые являются просто значениями по умолчанию. Вы можете изменить их. Потому что что я хочу сделать? Я хочу сказать привет, Дэвид или кто-то еще, так что я хочу сказать привет, запятая, а потом, Дэвид - все, что напечатал человек. s, что позволяет вам присоединиться. Это позволяет вам присоединиться или объединить две фразы, которые так или иначе предоставлены вами или пользователем. Итак, позвольте мне вытащить это, ответ. Позвольте мне пойти дальше и взять блок соединения. Заметьте, это тоже будет расти, чтобы заполнить. Позвольте мне пойти дальше и сказать: привет, запятая, пробел, а теперь перетащите ответ туда. И обратите внимание на это вложение. Как в математике. Это вложение функций. Я могу сначала присоединиться к привету и ответить, взяв эти две вещи в качестве входных данных, а затем передать их, чтобы сказать, как еще один вход, потому что эти вещи расположены сверху. И теперь, если я остановлю это, сыграю снова и скажу: Дэвид ... привет, Дэвид. Теперь у нас есть создание более интересной интерактивной программы, которая не просто жестко запрограммирована. Конечно, это не так слышно, как «Оскар Тайм» минуту назад. Итак, позвольте мне пойти дальше и сделать это. Позвольте мне начать все сначала и относиться к Скретчу как к коту, которым он является, и просто начать звук, называемый мяу. Оказывается, есть категория блоков, называемых звуком, и в звуке есть звуки по умолчанию. Так что начните звучать мяу. И теперь все станет немного привлекательнее. [MEOW] Оу. А теперь снова. [MEOW] И я могу имитировать кота, [MEOW], стоящего здесь некоторое время и продолжающего нажимать эту кнопку. Но вы знаете, что? Позвольте мне сделать его мяуканье несколько раз, потому что это более реалистично. Итак, позвольте мне взять второй и третий. И вы можете получить этот бесконечный запас блоков. Позвольте мне ударить игру. [MEOW] Похоже, ошибка. Давай еще раз попробуем. Играть. [MEOW] Это моя первая ошибка или ошибка. Это выглядит правильно. Там написано, что когда щелкнул зелёный флажок, начать звучать мяу, начать звучать мяу, начать звучать мяу. Почему я слышу только один мяу? Да уж. Да уж. Они как будто в одно и то же время или так близко к тому же времени, что звуки как бы пересекаются друг с другом и просто перекрываются, верно? Блок буквально говорит: начни звучать мяу. Но компьютеры действительно быстрые. Если вы слышали о выражении гигагерц, это единица измерения. И если ваш компьютер имеет один гигагерцовый процессор, центральный процессор или мозг, это означает, что он может буквально выполнять миллиарды операций в секунду. Конечно, он может начать три звука очень быстро. И если все они фактически происходят один за другим еще до того, как звук закончится, вы услышите только один чистый эффект. Итак, как мы можем это исправить? Ну, я действительно могу пойти и исправить это с этим блоком здесь - играть звук мяу, пока не будет сделано. Играйте звук мяу, пока не закончите. Играйте звук мяу, пока не закончите. И сейчас. [MEOWING] ОК. Это немного несчастно, этот конкретный кот, но, по крайней мере, теперь он более правильный. И, как выясняется, если я пойду на контроль - знаешь что? Здесь есть этот блок - подождите несколько секунд. Я могу пойти дальше и вставить это здесь. Позвольте мне сделать еще один здесь. А теперь нажмите Play. [MEOWING] Вы знаете, это не плохо. Теперь это звучит немного более реалистично. Но, если честно, если я продолжу это делать - или знаешь что? Вы можете фактически щелкнуть правой кнопкой мыши или управлять щелчком на блоках, дублировать их, и просто копировать и вставлять еще больше, если хотите. Так что, если бы я сделал это, сейчас, это будет идти шесть раз. И тогда я мог бы скопировать это снова и пойти 12 раз. Но должен быть лучший способ, верно? Это сейчас плохое программирование. Это плохой дизайн. Потому что я буквально копирую и вставляю, хотя и графически. Но мы' Мы уже видели строительный блок, с помощью которого мы можем лучше разработать эту программу. Это правильно, но не очень хорошо продумано. Что будет строительным блоком, чтобы мне было немного чище? ХОРОШО. Четыре цикла. Не совсем существует в Scratch. Но цикл в принципе делает что-то циклически. И действительно, если я пойду под контроль и начну ковыряться, вы заметите, что здесь может быть несколько блоков. Есть блок повтора несколько раз или навсегда, оба из которых звучат как циклы или циклы. Поэтому, конечно же, позвольте мне идти вперед, и я могу выбрасывать блоки, просто перетаскивая их влево. Позвольте мне вытащить это на секунду. А потом просто скажи, что всегда проигрывай этот звук, а потом подожди одну секунду. Итак, теперь моя программа выглядит следующим образом. [MEOWING] Вы знаете, мы никогда не узнаем, если это Это технически правильно, потому что это будет продолжаться, мы думаем, навсегда, но похоже, что это правильно. И это было намного меньше кода, и его гораздо проще поддерживать, потому что, если я хочу, чтобы он немного заснул, я мог бы вместо этого сказать две секунды. [MEOW] Вы знаете, и мы можем настроить это на ходу. Но давайте начнем объединять некоторые из этих идей сейчас и изменим их конечный эффект. Позвольте мне пойти дальше и открыть пример, который я сделал заранее. Этот называется Граф Ноль. И мы разместим это на сайте позже, чтобы вы могли поиграть, если хотите. И это как бы противоположность подсчета овец. Вместо меня или человека, который спит, считая овец, эта овца будет считать себя. Итак, позвольте мне пойти дальше и просто поиграть. И восхитительно, он, кажется, просто считает 1, 2, 3. Но почему это так? Он' Просто буду считать навсегда. Но давайте посмотрим на блоки, с которыми он рассчитывает вечно. Когда зеленый флаг нажал, установите счетчик. Оказывается, этот оранжевый прямоугольник - это то, что мы назвали переменной. Так что в алгебре это будет похоже на x или y или z. Это не описательные. Вместо этого я назвал этот счетчик, но я мог бы назвать его x или y или z. И затем я всегда говорю счетчик в течение одной секунды, затем жду одну секунду, а затем меняю счетчик на единицу, что технически означает просто увеличивать его. Добавьте 1 к этому. И поэтому овца просто собирается считать вверх и вверх и вверх. Теперь это немного утомительно, но в этом и заключается смысл подсчитывать овец, конечно, чтобы заснуть. Но что, если овце на самом деле нравится считать немного быстрее? Хорошо, позвольте мне пойти под операторами здесь. Умножение звучит так, как если бы оно могло нас быстрее найти. И позвольте мне пойти дальше и перейти к переменным. И вместо того, чтобы менять счетчик на один, позвольте мне продолжить и просто установить его на что-то другое. Итак, позвольте мне перетащить это. Установите счетчик равным чему-то, умноженному на что-то, в частности счетчик, умноженный на два, таким образом удваивая, удваивая, удваивая, удваивая. Похоже, что это будет расти намного быстрее. Посмотрим. 1, 2, 4. Значит, он считает быстрее, но все равно утомительно. Что если мы вместо этого сделаем это? Давайте перестанем ждать, и давайте продолжим и, с внешностью, не скажем счетчик на одну секунду, но давайте просто быстро скажем счетчик. Так что я собираюсь сказать счетчик. Упс. Я собираюсь сказать счетчик, а затем я собираюсь установить его на себя раз 2. Так вот, где мы находимся. Инициализируйте или сначала установите счетчик в 1, произнесите это, затем удвойте его, затем удвойте это, затем удвойте это, говоря это по пути. Итак, поехали. Это поразительно. Так что теперь овцы насчитывают до 10 до 60 до сих пор. С 10 до 100 ХОРОШО. Теперь, это даже не вписывается в речевой пузырь, но он все еще идет. Как высоко он может пойти? Какое наибольшее число вы можете рассчитывать на компьютере? Кто-нибудь хочет угадать? Может быть здесь некоторое время. С 10 по 270 сейчас. Как высоко вы можете сосчитать, а точнее - хорошо. Поэтому мы сдались и просто назвали это бесконечностью. Получается, что бесконечность имеет точное значение - от 10 до 250 или около того. Но что здесь происходит? Ну, потому что компьютеры, в конце концов, просто хранят информацию в цифровом виде - но эта информация должна храниться в цифровом виде с использованием электричества, используя эти переключатели более низкого уровня, называемые транзисторами. В конце дня мой телефон, мой ноутбук, у любого рассматриваемого устройства есть только конечное количество этих вещей. У меня только конечное количество пальцев. Используя унарный, мой старый хэшмарк подход, я могу сосчитать до пяти на этой руке. Используя бинарный файл, я сказал, что могу рассчитывать до 31 на этой руке. Но это все еще конечно. Я не могу рассчитывать на бесконечность на этой руке, потому что у меня только пять пальцев. Точно так же компьютер имеет только так много транзисторов или так много байтов или битов памяти, и в какой-то момент программист должен думать о том, что он или она собирается делать, когда пользователь хочет считать так высоко, что вы не можете физически это больше подходит. Вы должны так сдаться и сказать что-то наполовину точно, или вы должны решить эту проблему другим способом. И мы увидим, когда мы доберемся до C, то, как вы справитесь с этой проблемой, не обязательно будет простым, и действительно, многие программы не справляются с этой проблемой. И, скорее всего, у всех нас есть программы, которые, если вы наберете в них достаточно большие слова или достаточно большие цифры, вполне могут сломаться, разбиться или заморозиться, потому что люди, в отличие от MIT, не ожидали, что это действительно может произойти, и справятся с этим. , Хорошо, позвольте мне пойти дальше и сделать это. Позвольте мне открыть эту программу и посмотреть, не можем ли мы сейчас прочитать код. Это называется Pet Zero, и это программа, которая имитирует петтинг. Так что, если я нажму кнопку воспроизведения и не коснусь клавиатуры, то, похоже, ничего не происходит, но если я сейчас наведу курсор на кошку ... [MEOW] Aw. Это довольно мило. [MEOW] Прямо сейчас, это больше мяуканье по требованию, когда вы гладите кошку. Зачем? Ну, обратите внимание, я добавил несколько других строительных блоков. Мы не использовали это раньше, но это интуитивно, вероятно, имеет довольно ясный смысл. Когда щелкнет зеленый флаг, навсегда сделайте следующее. Если кошка касается указателя мыши - эта синяя вещь - то, что мы называли ранее логическим выражением. У него есть ответ да / нет, ответ истина / ложь, ответ один / ноль. А прикосновение к указателю мыши - одна из опций в небольшом выпадающем меню, если вы повозитесь с ним. Так что, если кошка касается указателя мыши, тогда и только тогда, играйте звук мяу, пока не закончите. Итак, теперь мы объединили функции с циклами с условием. Но почему петля? Кошка только мяукает один раз, когда я ласкаю его. Почему я здесь что-то делаю вечно? Кто-то ... да. Да уж. Я мог бы хотеть погладить это снова, поэтому я хочу, чтобы программа предвидела это. И, честно говоря, если я пропустил этот вечный блок, а моя программа выглядела именно так - позвольте мне избавиться от этого и от этого - и затем я нажал кнопку воспроизведения, и теперь я наводю курсор на него, почему он не работает ни разу? Скажи это снова. Да уж. Я имею в виду, что на данный момент, если я могу подвести итог, компьютер настолько чертовски быстр, что это уже произошло к тому времени, когда я переместил курсор на кошку, и в тот момент, когда я нажал кнопку воспроизведения, я не касался кошки. Эти блоки выполнены, так сказать, сверху вниз. Вот и все для программы. Так что, когда я подведу курсор к коту, программа закончится. Это не слушание. И так всегда, таким образом я могу слушать бесконечно, чтобы что-то действительно произошло. Что делать, если я хочу что-то сделать не только если что-то верно, но и обработать два случая? Если или иначе. Хорошо, позвольте мне пойти дальше и открыть Pet One. И это еще один пример. И может ли кто-нибудь, после прочтения этого кода, описать, что вместо этого будет делать эта программа? Да уж. Именно так. И позвольте мне подвести итог более устно. Так что, если на этот раз вы дотрагиваетесь до кошки, она заревет. Иначе, это будет сладко мяукать. Так что на этот раз она мяукает постоянно раз в секунду, но если вы дотрагиваетесь до этой конкретной кошки, ей это не понравится. Так что играй. [Мяу] Мяу. [Мяу] Мяу. [MEOW] И сейчас. [Рев] Не трогай кота. Так что теперь мы могли бы взаимодействовать двумя разными способами, имея две разные дороги, по которым вы можете идти вниз. Что ж, давайте сделаем что-то более интерактивным. Позвольте мне пойти дальше и открыть еще один пример. Это называется Bounce Zero, потому что теперь мы можем начать видеть некоторые элементы дизайна из того, что было Оскаром Времени. Вот так сейчас становится немного интереснее. Что на самом деле здесь происходит? Итак, позвольте мне увеличить блоки здесь. Этот блок просто говорит, навсегда переместить 10 шагов, что является еще однимБлок мы не видели. Но 10 шагов это как 10 пикселей. Так что переместите 10 пикселей на экране. Но если вы касаетесь края, то развернитесь на 180 градусов. И вы можете видеть, что именно происходит. Царапина поворачивается на 180 градусов, и этот стиль вращения означает просто двойное возвращение. Не люблю петлю около 180 градусов. Это круто. Но это не так, как люди или кошки ходят. Как то, что явно неестественно в этом? Да уж. Я имею в виду, я не могу даже симулировать это, верно? Как будто его ноги находятся в статичном положении, но скользят вперед и назад по экрану. И все же, это не то, что ходьба. Например, ходьба имеет какое-то движение и что? Ну, мы могли бы просто имитировать это как ... Хорошо, я мог бы просто ходить - гулять, и вы можете себе представить, как бы очень быстро сфотографировать мои ноги или кота. s нога движется, а затем просто решив, что это фото будет репрезентативным для одного шага. Это фото будет представителем другого. И знаете, с помощью всего лишь двух из этих шагов, я бы держал пари, что мы действительно могли бы довольно хорошо сымитировать, как выглядит ходьба. На самом деле, если я вернусь к тому, с чего мы начали, к этой картине Царапины, что, если я просто слегка пошевелю его ногами, потом вернусь, а затем пойдет вперед? И даже только в своем PDF-файле я могу имитировать анимацию, нажимая стрелку вверх, стрелку вниз, стрелку вверх, стрелку вниз, потому что это похоже на то, как он идет сейчас, когда действительно ваши человеческие глаза снова и снова видят две разные картинки. Так как я могу это сделать? Что ж, если я вернусь к Scratch, он все еще идет. Позвольте мне пойти дальше и открыть Bounce One, второй вариант этого, а теперь сделайте это. ХОРОШО. Итак, как я добавил это? Там' маленький фиолетовый блок, который мы еще не видели, но если вы будете разбираться в категориях, вы увидите другие блоки, как этот следующий костюм, который просто меняет костюм, который он носит. Оказывается, Царапина существует в виде картинки, и его изображение по умолчанию - он не двигается, но если я подойду к верхнему левому углу и нажму на костюмы, вы можете увидеть, что это его единственный костюм. Вот его второй костюм. И вот этот фиолетовый блок, который говорит следующий костюм, потому что он находится в бесконечном цикле, он просто продолжает делать следующий, следующий, следующий, следующий, следующий, просто показывая один костюм или другой. Они явно имитируют ходьбу. Теперь это не очень неестественно. Почему бы нам не замедлить его, скажем, до пяти шагов за раз и заставить его идти снова? Теперь, это все еще идет довольно быстро. Позвольте мне пойти дальше и сказать - мы могли бы иметь контроль. Мы могли бы попросить его подождать секунду после очень резкого движения. Возможно, мы могли бы ускорить это. Итак, давайте подождем 1/10 секунды, 0,1. Или, может быть, давайте сделаем 0,01, 1/100 секунды. Теперь это становится немного более реалистичным. Но это то, что анимация. Если вы когда-либо смотрели мультфильм или фильм, основанный на таких картинках, вы просто настраиваете некоторые из этих параметров, эти входные данные, чтобы получить этот выходной сигнал, понимая, каково фундаментальное представление этих вещей, которое в в данном случае это просто картинки, снова и снова и снова для создания этой анимации. Но как насчет интерактивности? Позволь мне сделать это самому. Позвольте мне пойти дальше и избавиться от этого, вернуться к событиям и сказать, когда щелкнул зеленый флаг. Затем, позвольте мне пойти дальше и взять блок навсегда, чтобы это продолжалось снова и снова. И затем, позвольте мне идти вперед, чтобы двигаться. Оказывается, что в движении, есть этот блок, который мы не видели - указывают на указатель мыши. И позвольте мне пойти дальше и вытащить это сюда. И затем, позвольте мне сделать это как один шаг за раз вместо 10. Что это будет делать? Что делает эта программа? Да уж. Скажи это снова. Следуй за мышью. Да уж. Это своего рода способ взять свою кошку на прогулку. Возможно, не совсем то животное, которое мы намеревались, но он последует за курсором. И я могу немного ускорить это. Так что давайте сделаем 10 шагов. ХОРОШО. Теперь поехали. Итак, теперь он двигается вверх и вниз, и теперь он интерактивен. Так что вы можете вспомнить это, когда мы раньше играли в «Оскар Тайм» и брали трубку - ОК. ХОРОШО. Не делай этого. Видишь, это ошибка. Он просто в замешательстве. Он' постоянно движется к этому, но ты уже ... ОК. Итак, мы собираемся остановиться. ХОРОШО. Итак, теперь он следит, но именно так мы могли бы сейчас создать, например, возможность перемещать эти куски мусора вокруг и заставлять их следовать за курсором мыши. Если вы вспомните время Оскара, каждый раз, когда вы собирали кусок мусора, он следовал за курсором, потому что был бесконечный цикл и такой блок, как этот, указывающий на указатель мыши. Что ж, теперь давайте интегрируем несколько идей и фактически создадим несколько сценариев. Ранее я предположил, что программы могут иметь несколько потоков. Поток - это просто причудливый способ сказать в нашем контексте несколько сценариев. Несколько скриптов в одной программе, которые происходят по существу параллельно. Компьютер может эффективно выполнять несколько задач одновременно благодаря многопоточности, и еще об этом в будущем. Так что они более сложны, но давайте посмотрим, сможем ли мы - давайте сначала разберемся, что делает эта программа. Позвольте мне пойти дальше и нажать «Играть». И этот имеет тенденцию быть немного громким. [МОРЕ ЛЬВА] Итак, морской лев просто бесконечно лает. Итак, прочитав код, как я могу остановить его от лая? Хит пробел. Отлично. Итак, нажмите пробел. ХОРОШО. Я мог бы просто остановить программу, но технически она все еще работает. Но почему это работает? Хорошо, обратите внимание на то, что слева находится первый скрипт. Когда щелкнет зеленый флаг, установите эту переменную, которую я назвал muted, в false. Можно было бы назвать это x или y или z или counter, но ни один из них не имеет смысла, поэтому я назвал это приглушенным. И я установил его равным false, что опять же является логическим значением. Правда или ложь просто означает да или нет. Навсегда, если нажата клавиша, сделайте это. Если muted в настоящее время имеет значение false, тогда измените muted на true. Иначе, изменить приглушенный на ложный. Так что, если muted - это false, измените его на true. Если параметр muted имеет значение true, измените его на false. Каждый раз, когда человек нажимает пробел, обновляйте эту переменную. Теперь, если мы посмотрим на другой сценарий, который также управляет морским львом, что он делает? Навсегда, если отключен, ложь. Так что, если он не отключен. Если muted - это false, значит не отключен. Запустите звук морского льва, затем подумайте: привет, привет, привет в течение двух секунд, а затем подождите еще одну секунду. А потом просто повторяй, повторяй, повторяй. Но если я изменю с отключенным пробелом на true, он скажет, если muted равно false, это не так. Я не собираюсь играть звук на этот раз. И теперь, у нас есть возможность объединить несколько сценариев вместе для достижения более интерактивного результата. И что по этому поводу? Вернувшись, когда я был ребенком, возможно, играл летом Марко Поло. Супер простая игра. По какой-то причине мы играли в пул, где один человек в пуле очень безопасно с завязанными глазами, а потом он или она кричит Марко. И затем, все вокруг него или ее должны кричать поло. И затем, человек с завязанными глазами должен пойти преследовать других детей в бассейне и пометить их, и тогда они станут этим. Но, другими словами, это как сигнальный механизм. Кто-то кричит, Марко и все остальные отвечают на эту передачу слова Марко. Что ж, получается, что мы можем смоделировать это с этими двумя марионетками. Этот парень здесь - обратите внимание, что я выделил оранжевую куклу, потому что там вторая голубая марионетка там. Отдельные спрайты. И это только фотографии, которые мы загрузили в игру. Навсегда, если нажата клавиша пробела, а если нажата клавиша пробела, произнесите Марко в течение двух секунд, а затем передайте событие. Между тем, у синей марионетки здесь очень простой блок, но он принципиально отличается от тех, что мы видели. Он не запускается, когда зеленый флаг нажимается. Он начинает только тогда, когда он получает событие. Таким образом, оказывается, что спрайты и Царапина не могут слышать или видеть то, что говорит другой в этих речевых пузырях. Вы должны использовать более изощренную технику, которая является специальным блоком, называемым широковещательной передачей, который похож на цифровую передачу заметки от одного спрайта к другому, которую, так сказать, можно прочитать или получить другому. Таким образом, только когда он получает это событие, так сказать, он говорит поло в течение двух секунд. И снова оранжевая марионетка отправляет это секретное сообщение, просто используя этот другой кусок головоломки. Трансляция события, как передача заметки, которую человек на самом деле не видит. Поэтому, если я сейчас нажму зеленый флаг и нажму пробел, оранжевый кричит Марко. Синий парень кричит в поло в ответ. Но они не рассчитаны вместе. Скорее, синий парень слышит с оранжевым, который сказал, тем самым позволяя множеству спрайтов фактически общаться. Так как мы сюда попали? Хорошо, напомним, что у нас были все эти строительные блоки минуту назад. Сначала мы начали с простых функций и условий, а также с логическими выражениями и циклами. Теперь мы добавили к этому возможность хранить информацию в переменных и потоках для одновременного выполнения нескольких задач, а затем, если у вас действительно происходит несколько событий, события, в которые они могут войти, каким-либо образом взаимодействуют. Еще один строительный блок. Так что, если мы сейчас сделаем шаг назад и рассмотрим, как мы можем создавать собственные функции, у нас будет последний кусок головоломки, так сказать. Позвольте мне пойти дальше и сделать это. Позвольте мне пойти дальше и создать простую программу, когда зеленый флажок обрезается, который просто симулирует кашель для кошки. Так что этот кот скажет не привет, а кашляет одну секунду. А потом он собирается идти вперед и ждать одну секунду. А потом я собираюсь скопировать вставить, как я делал раньше - это один из тех, кто делает, как я говорю, а не как я, - чтобы реализовать эту программу здесь, где он три раза кашляет. Мы уже знаем, однако, из более раннего, что это не очень хороший дизайн. Зачем? Ты повторяешься. Не повторяйся. СУХОЙ это аббревиатура, на самом деле. Не повторяйся, потому что ты Вы делаете в три раза больше того, что вам действительно нужно сделать только один раз. Конечно, раньше было просто использовать какой-то цикл. Итак, позвольте мне на самом деле убрать это. Позвольте мне использовать блок повтора, замените 10 на три, а затем просто используйте два из этих блоков. И обратите внимание, программа намного компактнее. И теперь, если я хочу изменить три на 30 или на 10 или любое число, я просто изменяю одно простое значение. Мне не нужно переписывать или копировать вставлять или удалять вещи. Я могу гораздо легче обновлять программу, и теперь то же самое произойдет с простым кашлем, кашлем, кашлем. Но оказывается, что отныне было бы неплохо абстрагироваться от этого, верно? Я просто хочу, чтобы любая программа, которую я пишу, умела кашлять. И кашель на самом деле просто говорит что-то, возможно, несколько раз. Но оказывается, что мы можем абстрагировать это в коде. Позвольте мне перейти к моим блокам здесь, и это позволит мне нажать на эту кнопку - сделать блок. Это позволяет мне сделать свою собственную функцию. Я получаю это диалоговое окно здесь. И я просто собираюсь назвать этот блок кашлем. Я собираюсь пойти дальше и нажать ОК. И теперь у меня есть этот новый розовый блок, который сам может иметь блоки под ним. И ты знаешь, что я собираюсь сделать? Я собираюсь пойти дальше и сделать это. Я собираюсь пойти дальше и сказать кашель под там. А теперь, обратите внимание, у меня теперь есть доступ к этой новой розовой части. Теперь я могу поставить это здесь. Итак, обратите внимание, хотя, да, именно так кашель реализован слева здесь, в следующий раз, когда я пишу программу, я просто хочу вызвать кашель. И меня не волнуют детали реализации более низкого уровня. Я не не волнует вечеринка или что-то в этом роде. Я просто хочу, чтобы это была абстракция. Но я мог бы добиться большего, чем это, было бы неплохо, если бы вместо того, чтобы просто повторить кашель три раза, что, если бы я сделал это признаком кашля? Итак, позвольте мне сделать это. Я могу пойти дальше и щелкнуть правой кнопкой мыши на этом розовом куске, и я могу отредактировать его. Это поднимает то же самое окно раньше. И заметьте это. Добавьте вход. Поэтому, когда я делаю пользовательский блок, я на самом деле могу создавать довольно причудливые блоки, точно такие же, как те, которые MIT дает нам с программным обеспечением, и теперь я могу напечатать что-то вроде n. И если я добавлю ярлык только для того, чтобы сделать его более наглядным, я могу просто сказать время. Итак, теперь я создал специальный кусочек головоломки, который несколько раз говорит «кашляю», где n для числа - это просто переход к переменной, которую обычно используют программисты. Так что теперь я могу перевести этот повторяющийся блок в сам кашель, но вместо жесткого кода 3, обратите внимание на это. Я могу украсть эту переменную и теперь сказать кашель много раз, повторяя это снова и снова и снова. И теперь, когда я кашляю в своей настоящей программе, я просто набираю три здесь. Так что теперь у меня есть эта прекрасная абстракция, так сказать. Кашель это много раз, и мне, и никому другому в мире больше никогда не нужно заботиться о том, что значит кашлять, потому что мы уже реализовали это раньше. И так же, как MIT дал нам так много функциональности, о которой нам самим не нужно думать, так и теперь я могу создать функциональность, о которой мне не нужно думать. И по мере продвижения к языкам более высокого уровня, таким как C, JavaScript и Python, мы собираемся продолжить этот процесс, иногда решая проблемы самостоятельно, создавая собственные кусочки головоломки, но очень часто используя такие вещи, как библиотеки, код, который другие люди написали перед нами, просто полезен для выполнения работы, как это сделал Scratch здесь для нас. Позвольте мне пойти дальше и собрать все это вместе, открыв здесь другой пример. Позвольте мне пойти дальше и открыть этот, который мы не видели, но это своего рода интерактивная игра, подобная этой, созданная бывшим студентом. [ИГРАЕТ МУЗЫКА] Должны ли мы иметь яблоко? Да. Немного анимации. ХОРОШО. Это не закончилось хорошо. Давай еще раз попробуем. Играть снова. И обратите внимание, что говорят блок происходит. Там какой-то блок вопросов. Студент проверял, печатал ли человек «да» или «нет». Давайте напишем нет на этот раз. Нет яблока. Ooh. Кекс. ХОРОШО. Да. Войти. ХОРОШО. Не делай этого. Еще одна жизнь. Вот так. [ВОСПРОИЗВЕДЕНИЕ МУЗЫКИ] ОК. Нет яблока. Нет кекс. Маленькая переменная. [Кричит] [СМЕХ] ОК. Итак, я выиграл игру. В наши последние моменты здесь позвольте мне пойти дальше и открыть один последний пример. Как вы знаете, CS50 предлагается не только в Гарварде, но и в Йельском университете, так что, кажется, уместно закончить на ноте, которая противопоставляет один кампус, возможно, другому, с помощью другой игры, которую бывший студент написал под названием Ivy's Hardest Игра. Но для этого, я думаю, нам нужен один последний волонтер, который готовит компанию. ХОРОШО. Из первых рук Прямо там. Давай вниз. Так что в самой сложной игре Ivy это игра с клавиатурой. И хотя на первый взгляд это может показаться немного ошеломляющим, точно так же, как это делал Оскар Тайм и точно так же, как и анимация пряников, вы поймете, что если вы разложите его, просто взглянув в уме, думая о том, что представляют собой эти отдельные строительные блоки, вы, вероятно, можете догадаться что такое кусочки головоломки Привет. Какие' твое имя? АНДРЕА: Привет. Я Андреа. ДЭВИД МАЛАН: Андреа. Дэвид. Рад тебя видеть. Вот самая сложная игра Айви. Противостоит всем Айви здесь. И тогда сразу после этого мы будем откладывать кексы в трансепте. Готовы? [ВОСПРОИЗВЕДЕНИЕ МУЗЫКИ] [МУЗЫКА - MC HAMMER, «ВЫ НЕ МОЖЕТЕ СДЕЛАТЬ ЭТО»] MC HAMMER: (ПЕТЯ) Вы не можете коснуться этого. Вы не можете прикоснуться к этому. Вы не можете прикоснуться к этому. ДЭВИД МАЛАН: Отлично. [МУЗЫКА - MC HAMMER, «ВЫ НЕ МОЖЕТЕ ПРИКАСАТЬСЯ К ЭТОМУ»] MC HAMMER: (Поет) Вы не можете прикоснуться к этому. Мой, мой, мой, мой. Музыка поражает тебя так сильно. Заставляет меня говорить, о мой господин. Спасибо, что благословил меня умом на рифму и двумя шумихой. Приятно, когда ты знаешь, что ты не в себе. Супер-дурак из Оактауна. И я известен как таковой. И это удар, который вы не можете коснуться. Я сказал тебе, домашний парень, ты можешь не трогай это. Да уж. Вот как мы живем, и ты знаешь, что не можешь этого коснуться. Смотри мне в глаза. Чувак, ты не можешь коснуться этого. [Аплодисменты] [МУЗЫКА - MC HAMMER, «ВЫ НЕ МОЖЕТЕ СДЕЛАТЬ ЭТО»] (ПЕТЬ) Свежие новые удары ногами и штаны. Вы получили это так, и теперь вы знаете, что хотите танцевать. Так что отойди от своего места и найди девушку-муху и поймай этот удар, пока он катится. Оставайтесь на линии. Прокачай немного и дай мне знать, что так и происходит. Как это. Холод на миссии, так что тяни на спину. Дайте им знать, что вы слишком много, и это удар, который вы не можете коснуться. Эй, я же говорил, ты не можешь это трогать. Почему ты стоишь там, чувак? Вы не можете прикоснуться к этому. Эй, звони в колокола. Школа в, лох. Вы не можете прикоснуться к этому. Дайте мне песню или ритм. Заставить их потеть. Вот что я им даю. Теперь они знают, что когда вы говорите о «Молоте», вы говорите о шоу, которое раскручено и жестко. Певцы потеют, поэтому передайте им вайп или кассету, чтобы узнать, что нужно сделать в 90-х, чтобы записать чарты. ДЭВИД МАЛАН: Со второго по последний уровень. [МУЗЫКА - MC HAMMER, «ВЫ НЕ МОЖЕТЕ ПРИКАСАТЬСЯ К ЭТОМУ»] MC HAMMER: (ПЕТЬ) Либо усердно трудитесь, либо вы могли бы также уйти. Это слово, потому что ты знаешь, что не можешь этого коснуться. Вы не можете прикоснуться к этому. Сломай. Стоп. Молоток времени. Говорите, плывите по течению. Если вы не можете с этим справиться, то вы, вероятно, мертвы. Так размахивайте руками в воздухе. Перебрать несколько ходов. Проведите пальцами по волосам. Это для победителя. Танцуйте под это, и вы станете худее. Переехать. Слайд попой. Только на минуту, давайте все сделаем удар, удар, удар. Да уж. Вы не можете прикоснуться к этому. Смотри, чувак. Вы не можете прикоснуться к этому. Тебе лучше получить ажиотаж, мальчик, потому что ты знаешь ... [Толпа кричит] [МУЗЫКА - МАК ХАММЕР, "ЭТО НЕ ТРОГАТЬ"] (ПЕТЬ). ДЭВИД МАЛАН: [неразборчиво] АНДРЕА: [неразборчиво] ДЭВИД МАЛАН: Нет. Все в порядке. [МУЗЫКА - МАК ХАММЕР, «ЭТО НЕ ТРОГАТЬ»] Еще одна жизнь. [МУЗЫКА - MC HAMMER, «ВЫ НЕ МОЖЕТЕ ПРИКАСАТЬСЯ К ЭТОМУ»] MC HAMMER: (Поет) Стоп. Молоток времени. ДЭВИД МАЛАН: Хорошо. Аплодисменты Андреа, если бы мы могли. [Аплодисменты] ОК. Вот и все для CS50. Смотрите веб-сайт для деталей. Увидимся на торте в трансепте. Добро пожаловать на борт. Нет, все нормально. [МУЗЫКА - МАК ХАММЕР, «ЭТО НЕ ТРОГАТЬ»] Еще одна жизнь. [МУЗЫКА - MC HAMMER, «ВЫ НЕ МОЖЕТЕ ПРИКАСАТЬСЯ К ЭТОМУ»] MC HAMMER: (Поет) Стоп. Молоток времени. ДЭВИД МАЛАН: Хорошо. Аплодисменты Андреа, если бы мы могли. [Аплодисменты] ОК. Вот и все для CS50. Смотрите веб-сайт для деталей. Увидимся на торте в трансепте. Добро пожаловать на борт. Нет, все нормально. [МУЗЫКА - МАК ХАММЕР, «ЭТО НЕ ТРОГАТЬ»] Еще одна жизнь. [МУЗЫКА - MC HAMMER, «ВЫ НЕ МОЖЕТЕ ПРИКАСАТЬСЯ К ЭТОМУ»] MC HAMMER: (Поет) Стоп. Молоток времени. ДЭВИД МАЛАН: Хорошо. Аплодисменты Андреа, если бы мы могли. [Аплодисменты] ОК. Вот и все для CS50. Смотрите веб-сайт для деталей. Увидимся на торте в трансепте. Добро пожаловать на борт.

WEEK 1

Lecture 1

ДОКЛАДЧИК 1: Это CS50, а это неделя 1. И к концу дня вы узнаете, как создавать программы, которые выглядят так. Так что это, конечно, бинарный. Это единственный язык, который машины в конечном итоге понимают. Но, к счастью, за последнюю неделю было так много абстракций и так много людей, которые предстали перед нами, что нам на самом деле не нужно ничего писать на этом уровне. Мы можем абстрагироваться намного выше, как мы это уже делали с Scratch, и как мы начнем сегодня с C. Но кто-нибудь, тем не менее, хочет высказать предположение о том, что на самом деле делает эта программа при подаче на ваш Mac или ПК? Кто-нибудь узнает? Кто-нибудь хочет рискнуть предположение? Это, пожалуй, самая простая программа, которую вы можете написать. Действительно, при подаче на мозг вашего компьютера, так называемый процессор просто печатает это. Так как же нам на самом деле добраться до этого? Ну, вспомните, где мы начали этот разговор в прошлый раз, говоря о компьютерных науках в более общем плане и о решении проблем - мы предложили, что это действительно может быть искажено. У вас есть некоторые материалы. Вы хотите некоторые результаты. И где-то посередине вам нужно что-то сделать с этими входами. И чтобы добраться до этого момента, мы должны были представить эти входы и выходы. Мы просто должны были решить, как люди, как мы представим все входные данные для нашей проблемы, когда наступит время, когда компьютер действительно их обработает. И в конце концов, все телефоны и компьютеры, которые мы все используем только в конце дня, подключаются к стене, чтобы получить свои физические ресурсы, электричество, и они могут временно хранить их в батарее. Но это действительно наш единственный вклад. Это' S либо подключен, либо нет. Это либо 1, либо 0, истина или ложь. Таким образом, мир действительно сводится к этим двум состояниям, так сказать. И тогда вы можете думать об этих состояниях как о лампочках, включенных или выключенных. В последний раз я включил свой мобильный телефон, чтобы включить или выключить фонарик, 1 или 0, true или false. Теперь, конечно, если у вас есть только одна лампочка, вы можете считать только от 0 до 1. Но если вы начнете иметь кучу их спина к спине, вы можете переставлять их, как я сделал мой палец-- 0, 1, 2, 3 и т. Д. И поэтому мы начали говорить о двоичном коде в более общем смысле. И вот, например, здесь были три последовательности 0 и 1, и каждая из них представляла что-то, но нам не нужно думать о мире на этом уровне. Мы можем абстрагироваться от этого. Все мы, конечно, намного лучше знакомы с десятичной дробью, и действительно помните, что это были только 72, 73 и 33, что, если кто-нибудь вспомнит, когда вы используете ASCII - какой это глобальный стандарт для отображения чисел в буквы - какое сообщение мы получили? Да, это было просто высоко, столица H, столица I, восклицательный знак. Итак, это абстракция поверх тех двоичных чисел. Но мы не должны моделировать только текст, используя числа. В конце концов, нашим единственным ресурсом по-прежнему остается это электричество, и единственный способ, которым мы думаем об этом в цифровом виде, это все еще нули и единицы. Но если мы возьмем одно и то же значение - 72, 73, 33 - и будем рассматривать их в контексте Photoshop, фотопрограммы или графической программы, мы можем вместо этого интерпретировать их как некоторое количество красного, некоторое количество зеленого, Некоторое количество синего, которое дало нам в прошлый раз, напомним, этот желтоватый цвет. Так что теперь у нас была еще одна абстракция поверх двоичных цветов, и это всего лишь один пиксель. Что вы можете сделать, если у вас есть более одного пикселя? Что вы можете представить дальше? Да, верно, изображения. Таким образом, мы продолжаем разговор вверх и вверх и вверх, и мы можем представить что-то вроде графического смайлика на экране, который имеет более чем одну желтую точку. Там есть целая куча желтых точек и других цветов. И помните, что, если мы хотим оживить вещи, будь то с помощью глупых вещей, таких как анимоджи на телефоне, или просто более подходящих видео и фильмов, ну, это просто последовательности изображений, пролетая мимо ваших человеческих глаз очень быстро. Так вот где мы как бы остановились в прошлый раз, начиная с базового уровня и абстрагируясь, чтобы потом можно было оговаривать, что мы можем представлять входные данные и представлять выходные данные, какими бы они ни были. И здесь, мы не Не нужно думать на этом уровне. Мы можем просто предположить, что мы все знаем, как это сделать. И даже если это в конечном итоге станет чем-то вроде далекой памяти, мы знаем, что кто-то действительно может это сделать. И это ценность абстракции. Но внутри этого черного ящика находятся так называемые алгоритмы, секретный соус - вот где проблемы на самом деле решаются. И мы говорили не только о том, что это за алгоритмы, но и, например, насколько они эффективны. Итак, напомним, что эта красная линия представляет собой очень простой алгоритм, просто переворачивая телефонную книгу постранично по очереди. И причина того, что это прямая линия, заключается в том, что существует взаимно однозначное соответствие между тем, сколько страниц в книге и сколько разворотов страниц - одна страница, еще одна страница, еще один поворот и так далее. Если я пройду через него с удвоенной скоростью - 2, 4, 6, 8-- Я могу сделать лучше. И так, чтобы желтая линия теперь, напомним, была ниже на графике. Если вы просто посмотрите на любые две точки, желтую и красную, желтый ниже красного, говоря, что это занимает меньше времени. Но это было не совсем правильно. Была одна ошибка, когда я искал Майка по две страницы за раз. Что это была за проблема? Да, я могу по нему скучать. Он может случайно оказаться зажатым между двумя страницами - не так уж много, потому что я могу это исправить, но я должен это исправить. Я должен применить эту дополнительную логику и сдвинуть назад хотя бы страницу, если я иду слишком быстро. Но, конечно, последний алгоритм - и, честно говоря, вся наша первоначальная интуиция, вероятно, - заключался в том, чтобы разделить и победить, открыть его примерно посередине, посмотреть вниз, а затем повернуть налево и направо, и просто повторить этот процесс как проблема в том, что от этого большого до такого большого до такого большого осталось всего одна страница. Так что это все об эффективности. Но чтобы добраться до этого момента, нам нужно было выразить себя более точно. И поэтому мы ввели псевдокод. Там нет формального определения. Это может быть английский, английский как. Это просто значит быть кратким и донести смысл. И помните, что по пути мы ввели целый ряд концепций, многие из которых вы, вероятно, экспериментировали с Scratch, например, циклы и условия, логические выражения, переменные и так далее. И это были строительные блоки, которые вышли из такой демонстрации здесь. Но, честно говоря, даже в этой демонстрации, в этом псевдокоде было множество предположений. Если вы читаете эти инструкции по одному и держите телефонную книгу самостоятельно, скорее всего, вы можете выполнить этот псевдокод, этот алгоритм. Но что это действительно значит, скажем, открыть до середины телефонной книги? У всех нас есть интуитивное понимание того, что это значит. Но, честно говоря, если вы объясняли, что ребенку или кому-то, кто впервые изучает английский или любой другой язык, откройте середину телефонной книги, вам, вероятно, следует изложить некоторые предположения. Хорошо, у этой вещи перед вами 1000 страниц, кусочки бумаги. Перейдите на 500-ю страницу, и давайте назовем это серединой. Это очень быстро станет утомительным, если все мы, люди, будем говорить с таким уровнем детализации. И поэтому мы абстрагируемся от более широких утверждений, таких как открытая середина телефонной книги, но это абстракция. И это не так точно, как, возможно, идеально, особенно, передавая этот алгоритм новичку, роботу или компьютеру. Но это' Это полезно, потому что мы можем создать 12-шаговую программу вместо 20-шаговой, разработав слишком много. И, например, здесь у нас были свои циклы, условия и так далее, но даже позвонили Майку. Что это значит? Ну, а если представить, что человек знает, как пользоваться телефоном, то это само собой разумеется. Но если он или она также должны быть запрограммированы для использования телефона, вы должны объяснить - поднимите его, нажмите эту кнопку, наберите эту последовательность кнопок и так далее. Так что называть Майка это тоже абстракция. Таким образом, эти абстракции полезны, но иногда они могут мешать, особенно если вы недостаточно точны, чтобы правильно программировать компьютер. И чтобы нарисовать эту картину, я подумал, что мы могли бы начать немного от души здесь. Я принесла завтрак, если вы не успели сделать это по соседству или дальше. Просто нужна пара добровольцев, если вам удобно появляться на сцене и в Интернете здесь. Дайте мне вроде ... здесь много света. Как насчет там слева и здесь, спереди? Да, прямо там Я думаю твоя рука была поднята. Давай вниз. И Брайан, ты тоже не против подать нам руку? Давай вниз. Если вы хотите взять контроль здесь, позвольте вам идти дальше и переключиться на другую программу для вас. Как вас зовут? Жан: Жан. ДЭВИД: Жан, Дэвид. Рад тебя видеть. Садитесь в крайнее левое положение. И ваше имя? ЭББИ: Привет, я Эбби. Приятно познакомиться. На крайнем правом, если вы могли. Итак, Джин и Эбби, ты хочешь что-нибудь быстро рассказать о себе? ДЖИН: Я Жан. Я уроженец Массачусетса, и я беру CS впервые. Это мое первое кодирование или что-то еще. Или я Я делаю, и я наслаждаюсь этим. ДЭВИД: Хорошо, рад, что ты с нами. А Эбби? ЭББИ: Привет, я Эбби. Я воспринимаю это как второкурсник и ничего не знаю о компьютерах или информатике. Так что я, наверное, беру это на стат. ДЭВИД: Хорошо, хорошо, что ты тоже. Итак, перед нами целая куча ингредиентов, и, надеюсь, мы сможем начать этот семестр мягко. И если мы добьемся успеха, мы действительно быстро перекусим. Но мы подумали, что подождем здесь аудиторию, и Брайан собирается писать, как мы идем. И все, что мы хотим сделать этим утром, это просто сделать сэндвич с арахисовым маслом и желе, по одной инструкции за раз. И каждый из нас просто исполнит то, что услышит. Как это звучит? Хорошо, если кто-то может добровольно дать первую инструкцию, и Брайан напечатает ее. АУДИТОРИЯ: Открытый хлеб. DAVID: Открытый хлеб мы слышали. Открытый хлеб - первая инструкция. Поэтому каждый из вас хотел бы казнить открытый хлеб. Нет, не смотри на меня. Хорошо, так что мы вроде как в пути. Я думаю, что Эбби сделала это лучше, конечно, но мы сделали это правильно, возможно. Итак, давайте перейдем к шагу 2 и посмотрим, не сможем ли мы улучшить. Вынь хлеб. Добро пожаловать в команду сейчас. Хорошо, хорошо, шаг третий. Да уж. АУДИТОРИЯ: Положите два куска хлеба на стол. ДЭВИД: Положи два куска хлеба на стол. Не обращайте внимания на тарелки. ОК, шаг четвертый. АУДИТОРИЯ: Поверните крышку желе, пока она не откроется. ДЭВИД: Поворачивай крышку желе, пока она не откроется. Спасибо. Шаг пятый? Шаг пятый? Да уж? АУДИТОРИЯ: Положите крышку в сторону. ДЭВИД: Спасибо. Поместите крышку в сторону. Я взял некоторые свободы сам. АУДИТОРИЯ: Возьми нож. ДЭВИД: Возьми нож. Снять крышку желе. На наших обложках нет. Вставить нож в бутылку? От верхней. Придерживайся - шаг девятый. АУДИТОРИЯ: Вращайте руки, чтобы желе закончилось. ДЭВИД: Вращайте руки так, чтобы желе закончилось - хорошо, шаг - быстро - 10. Да, шаг 10? АУДИТОРИЯ: Вытащите нож. ДЭВИД: Вытащи нож. ОК, шаг 11. Желе стороной вниз на хлеб. Хорошо, шаг 12. Шаг 12, кто-нибудь? Да. Спасибо. Шаг 13. Вылейте желе на хлеб. ДЖИН: Лей желе ... ДЭВИД: Лей желе, желе. Все это? Хорошо, теперь ты просто балуешься с нами. Шаг 14. АУДИТОРИЯ: Положите желе вниз. ДЭВИД: Положи желе. Спасибо. 15? Возьмите арахисовое масло. АУДИТОРИЯ: Снять крышку. ДЭВИД: Сними крышку. Спасибо. Снять крышку. Спасибо. Шаг 18? Подберите нож тупым концом, совок. Совок. Шаг 20? Положить арахисовое масло на хлеб? 21? АУДИТОРИЯ: Переместите нож слева направо. ДЭВИД: Двигайся, двигай нож слева направо, пожалуйста. Слева направо. Шаг 22? АУДИТОРИЯ: Положи нож. ДЭВИД: Спасибо. 23? АУДИТОРИЯ: Положи банку. ДЭВИД: 24. Что это было? Ешьте бутерброд. Хорошо, я думаю, мы ... ну, почему бы каждому не откусить? И аплодисменты, если можно, нашим добровольцам. Спасибо. ЭББИ: Не возражаешь, если я возьму немного для ссоры? ДЭВИД: Если хочешь, можешь взять немного для ряда. Спасибо. Так что - спасибо, Брайан. Хорошо, теперь мне нужна минутка. Спасибо. Достаточно сказать, что это, очевидно, продемонстрировало даже больше, чем пример телефонной книги, где есть наши определенные предположения и наши абстракции. И, честно говоря, почти все время они полезны. И, конечно, мы вроде как все испортили. И я думаю, что инструкции как бы помогли с этим здесь. Но когда придет время программировать с Scratch и, конечно, с C, начиная с этой недели, вы можете на самом деле больше таких предположений нет, потому что, если вы не справитесь с этими угловыми случаями и не задумаетесь над тем, что означает эта инструкция, вы получите вращающийся пляжный мяч или песочные часы, которыми вы пользуетесь. знакомы на вашем Mac или ПК. Программа будет аварийно завершена, что-то пойдет не так, просто потому, что вам не хватает какой-то специфичности или точности. Теперь мы полны арахисового масла. Итак, со временем мы обнаружим, что, как и в Scratch, мы смогли создать свои собственные строительные блоки. Вы могли бы вспомнить короткие примеры, которые мы сделали с примером кашля, где у меня был кашель 0, а затем кашель 1 и кашель 2, где я делал свою собственную часть головоломки в Scratch. Это было полезно, потому что после этого примера теоретически мне больше не нужно думать или беспокоиться о том, как вызвать кашель. Я могу просто использовать эту абстракцию. Но кто-то должен их реализовать, и иногда это будут другие люди, которые были до нас. И иногда это будут мы. Таким образом, это не означает, что программирование оказывается настолько утомительным, что вам приходится указывать на каждую мелочь, но вы или кто-то должен делать этот уровень точности хотя бы один раз. И, как ни странно, в Scratch MIT сделал большую часть этой работы за вас. У всех нас были строительные блоки, с помощью которых мы могли создавать нашу собственную анимацию, игру, произведения искусства или тому подобное. Но даже тогда вам, вероятно, пришлось соединить несколько десятков кусочков головоломки или даже больше, чтобы заставить эти основы делать то, что вы хотели. Итак, сегодня мы собираемся начать переход от Scratch, этого графического языка программирования, который предназначен для младших школьников, обычно является представителем множества тех же понятий, которые теперь будут использоваться в течение всего семестра. Но сегодня мы представим более старый, более традиционный язык, основанный только на тексте. И как таковая, она намного мощнее. Но на первый взгляд все будет выглядеть более загадочно. Фактически, вместо того, чтобы писать нули и единицы, начинающиеся сегодня, мы вместо этого собираемся написать что-то вроде этого. Теперь, если вы никогда раньше не программировали, шансы, на первый взгляд, выглядят довольно загадочно, и в них много символов, пунктуация с клавиатуры. Там наверняка есть знакомые английские слова. И, честно говоря, даже после того, как кто-нибудь поцарапал, даже без предшествующего опыта, Возможно, вы рискуете догадаться, что эта программа, написанная на другом языке, называемом C, делает при запуске. Это просто печатает привет мир. Теперь при условии, что синтаксически приличная сумма накладных расходов. Есть множество вещей, которые вы должны напечатать, чтобы заставить эту программу делать то, что вы хотите, чтобы она делала. Но в конце концов, это все, что он собирается сделать. И это одна из самых простых программ, которые мы собираемся добавить в наши кусочки головоломки, так сказать, сегодня и увидеть некоторые из тех же концепций, которые мы видели и в прошлый раз. Итак, давайте сделаем это сначала, хотя. Позвольте мне на минутку сравнить Scratch с C, потому что самый важный вывод на сегодня будет таким, даже если синтаксис не выглядит таким очевидным - и, честно говоря, даже если ваши первые минуты или часы с написанием собственного кода в Си расстраивает, потому что, черт возьми, Вы оставили точку с запятой или, о, у меня были круглые скобки в неправильном месте. Есть много этих глупых синтаксических зависаний, которые заставляют вас довольно часто чувствовать, что вы действительно не понимаете этого. Но это не главное. Большая часть синтаксиса наименее важна. Это совсем не интеллектуально интересно. Поэтому постарайтесь увидеть прошлое и постараться успокоиться тем, что принципы будут важны. И, честно говоря, только мышечная память и практика, все остальные вещи, которые поначалу будут вызывать периодическое разочарование, просто начинают уходить, когда вы начинаете видеть это таким, какой он есть, а не только для синтаксиса, который вы видите. на первый взгляд. То есть эта программа справа на C эквивалентна той, что мы делали неделю назад с двумя кусочками головоломки в Scratch. Теперь на моем Mac или моем компьютере не будет зеленого флажка, так как мы продвигаемся вперед, что вы можете просто нажать. Мы собираемся запустить эти программы немного по-другому, но это все, что делает код справа. Это эквивалентно коду слева. Итак, давайте сделаем это снова и снова для некоторых из этих концепций прошлого раза, а затем мы начнем писать некоторые из наших собственных программ. Итак, это был пример, этот фиолетовый блок, какой концепции в программировании? Да, функция. Так что это был глагол. Это было действие, и мы будем называть эти общие функции. Они просто имеют встроенную функциональность. Так, как мы делаем это в C? Ну, вы могли бы вспомнить только минуту назад, потому что одна из строк кода представляла это, у него был некоторый синтаксис. Так на самом деле, если бы я перевел блок слева в Scratch на эквивалентный код на этом другом текстовом языке, называемом C, я начну с написания print, затем открою скобки, а затем закрою скобки. И эти круглые скобки представляют овал, белый овал слева, в который мы ввели привет мир раньше. Теперь в C это не так буквально, как это. Функция, или глагол, на самом деле не называется print. Это называется printf, а F обозначает отформатированный. И это просто означает, что в C вы можете отформатировать текст по-разному. Итак, мы увидим это в ближайшее время, и получается, что вы не просто пишете привет мир между этими скобками, как мы делали в Scratch. Вы также должны окружить их двойными кавычками в C. Не такая уж большая проблема, но то, что вам раньше не приходилось делать. Но в С, Вы также будете вообще хотеть быть супер определенным для компьютера. И когда вы хотите, чтобы курсор текста на экране переместился вниз на следующую строку, вам нужно сообщить компьютеру об этом, буквально набрав обратную косую черту n. Человек не увидит обратной косой черты и n. Он или она действительно увидит, как курсор переместится на следующую строку экрана, как в Google Docs, Microsoft Word и т. П. Но это просто говорит о точности, которая вам нужна при общении с компьютером на этом уровне, а не только с кусочками головоломки. А потом еще одна вещь - и я упоминал об этом раньше, потому что это рано для многих программистов. Большинство строк кода на C должны заканчиваться точкой с запятой. Это своего рода код, эквивалентный точке в предложении на английском или некоторых других языках. Итак, это все. Нам понадобилось немного времени, чтобы создать это. Но это все, что есть. Идея слева сказать что-то такая же в C, как и печать чего-либо с помощью этой функции с именем printf. И прежде чем я начну другие сравнения, есть вопросы по этому переводу? АУДИТОРИЯ: Как ты пишешь обратную косую черту? ДЭВИД: Как ты пишешь обратную косую черту? Хорошо, так что если подумать, это может затруднить буквальное отображение обратной косой черты пользователя n. Что ж, получается, что это обратная косая черта, потому что это не очень распространенный символ, мир программирования использует его как то, что называется escape-символом. Это тот, который вы используете, когда хотите скрыть информацию и показать ее немного по-другому. Таким образом, способ, которым вы бы буквально показывали человеку фактический обратный слеш n, заключается в том, чтобы фактически в вашем коде сделать обратный слеш обратный слеш n, потому что второй обратный слеш - это как сказать, что следующий символ обрабатывается специальным образом и фактически показывает его человеку. И есть другие подобные примеры этого. Так как насчет этого? Этот оранжевый блок был примером одной концепции в Scratch? Да, так что это была переменная, как х и у в алгебре. Это был просто заполнитель для данных, и вы могли хранить цифры. Оказывается, вы можете хранить слова. Вы также можете хранить другие вещи на других языках. Итак, в C мы собираемся сделать это. Мы собираемся буквально сказать имя переменной, которую мы хотим, например, counter. Но мы можем назвать все, что захотим - равными нулю, если мы устанавливаем это изначально равным нулю. Но C немного более педантичен. Вы' Мы также должны сказать компьютеру, тип переменной, которую я хочу, определенно для целого числа, иначе сокращенно int. Поэтому вы должны заранее сообщить компьютеру, какой тип данных вы собираетесь на нем хранить. И догадайся. Вы должны закончить мысль С. Что еще нам нужно добавить к ... да, просто точка с запятой. И это все. Это выглядит немного более загадочно, но идея в принципе та же. Так что, если мы хотим сделать это на пустом месте? Измените счетчик на 1 - это было эквивалентно увеличению или добавлению 1 к счетчику. Что ж, позвольте мне пойти дальше и предложить, чтобы вы могли буквально просто сделать это в C. Установите счетчик равным тому, который в настоящее время равен плюс 1. Это, кажется, правильная интуиция. И теперь обратите внимание, что здесь важно отметить, что этот знак равенства не Сказать, что счетчик равен счетчику плюс 1, потому что это просто не представляется возможным. Если вы выберете любое значение для счетчика, например, число 1, то одно точно не равно 1 плюс 1, то есть 2. И 1 не равно 2, и вы можете придумать бесконечное количество беспокоящих некорректных сравнений. Так что знак равенства в C, как и многие языки, которые мы увидим в классе, на самом деле означает присваивание. Скопируйте значение справа в значение слева. Так что установите счетчик равным плюс 1. Что? Мы должны закончить мысль. Итак, нам нужна точка с запятой. Мне не нужно упоминать int, и почему это может быть? Да, я уже сказал компьютеру, что это целое число. Вам не нужно повторяться, снова упоминая int, предполагая в этом контексте, хотя мы смотрим на это только на слайде, на самом деле был создан раньше, так же, как вы сделали с Scratch, говоря make переменную. Так что получается, что вы можете быть немного более лаконичны в Си и многих языках. Если вы найдете это немного утомительным, чтобы напечатать ... и это немного многословно. Это куча нажатий клавиш. Вы можете фактически сократить это только этим. Так что «плюс равно» - это просто синтаксический сахар, как сказал бы программист. Это просто замечательная функция, которая позволяет писать меньше слов или символов, но делать то же самое. И, честно говоря, мы можем сделать немного лучше. И если вы взяли PCS, вы могли бы увидеть это и на Java. Вы также можете упростить это еще больше, чтобы просто счетчик плюс плюс точка с запятой. Вот и все - все равно. Это немного более эффективно. А поскольку вы получаете более удобное программирование, сохранение нажатий клавиш просто экономит ваше время. Теперь это, конечно, был пример того, что в Scratch в отличие? Да, мы назвали это условием. И это было логическое выражение, о котором мы задавали вопрос. В этом случае мы, по-видимому, спрашиваем в Scratch, что x меньше, чем y, и, если так, скажите это на экране. Итак, как мы можем перевести это, чтобы увидеть? Ну, оказывается, мы можем довольно просто перевести это буквально. До сих пор мы видели почти все строительные блоки, но мы должны представить здесь кое-что новое. Обратите внимание, что строка printf практически идентична той, что я использовал ранее для всего привет. Я, очевидно, только что изменил слова в нем, но у меня все еще есть обратный слеш n. У меня все еще есть цитаты, все еще есть точка с запятой. В остальном все то же самое. Теперь, если это новое, но это перевод один на один. Скретч называет это если. С называет это если. И единственное, что вам нужно в C - это круглые скобки вокруг логического выражения. Так вот что занимает место маленького зеленого блока там. И затем, предполагая, что x и y действительно являются переменными, которые мы создали ранее, вы можете просто сравнить их вот так и использовать для сравнения символы больше, чем и другие символы. Но есть кое-что немного интересное, и большинство из нас не часто имеют возможность даже использовать эти клавиши на нашей клавиатуре. Фигурные скобки, на клавиатуре США они, как правило, находятся в правом верхнем углу над клавишей Enter. Это всего лишь С эквивалент этой формы. Обратите внимание, что у большинства желтых блоков в Царапинах была такая обнимающая или обнимающая их форма. Вы можете смоделировать это в C, имея то, что называется открытой фигурной скобкой, а затем закрытой фигурной скобкой. Так что это точно такая же идея. Кроме того, технически вам не всегда нужны эти фигурные скобки. Если вы только что получили один лайнер, как этот, вы можете опустить их, как вы можете увидеть в Интернете или в учебниках. Но мы всегда будем рисовать их для согласованности, чтобы код C всегда выглядел так. Что если вы захотите это выразить? Если x меньше, чем y, то скажем, что x меньше, чем y, иначе скажем, что x не меньше, чем y. Ну, оказывается, это почти идентично. Первые четыре строки, совершенно такие же, как и раньше. Но, оказывается, в C вы можете буквально сказать другое после этой закрывающей фигурной скобки. А затем просто распечатайте все, что вы хотите сказать. Так что это как развилка на дороге. Если ты пойдешь в одну сторону, скажи это. Если ты пойдешь другим путем, скажи это другим. Есть ли какие-либо вопросы по поводу этих сравнений до сих пор? Да уж. АУДИТОРИЯ: Должны ли мы поставить первую скобку на той же строке, что и if? ДЭВИД: Действительно хороший вопрос. Можете ли вы или вы положили фигурную скобку на той же линии, если? Вы можете, и мы будем говорить об этом в ближайшие пару недель, это вопрос стиля. Есть разные способы выразить этот точно такой же код. Честно говоря, я мог бы написать весь этот код без пробелов вообще. На самом деле, просто чтобы подчеркнуть это, если я пойду дальше и просто открою здесь простой текстовый редактор - не для того, чтобы на самом деле программировать, а просто для того, чтобы что-то напечатать - я мог бы сделать что-то подобное, если x меньше y. Затем продолжайте и распечатайте x меньше, чем y обратная косая черта n фигурная скобка с запятой, иначе выведите и так далее - полностью нечитаемым в конце дня или не поддерживаемым, особенно когда код усложняется. Но пробелы не имеют значения для компьютера, но это имеет значение для человека. И, как вы намекаете на некоторых языках, это обычно делается, когда вы фактически держите фигурную скобку на одной строке. И действительно, вы могли бы видеть, что учебники делают это также. Некоторые люди даже сделают это. Все это короткие рассказы о стиле. В CS50, в первые недели занятий, мы будем настаивать на том, чтобы все следовали одному и тому же стилю, чтобы у нас была основа для сравнения. Но в конце концов, это то, что, как и в вашем собственном английском языке или на любом другом языке, на котором вы склонны писать, у вас есть свой собственный стилистический или лингвистический стиль. Код также имеет это. Другие вопросы? Да уж. АУДИТОРИЯ: Когда вы устанавливаете контр-переменную, вы всегда должны говорить, чему она равна, или вы можете просто сказать int counter section? ДЭВИД: Действительно хороший вопрос. Когда вы объявляете переменную, создаете переменную, нужно ли вам сразу устанавливать ее равной чему-то? Короткий ответ, нет. И мы скоро увидим примеры этого, где вы можете сказать, дайте мне переменную counter, но на самом деле не устанавливайте ее равной некоторому значению. Вернемся к этому немного позже. Так что, если мы хотим добавить эту логику? Честно говоря, в Scratch это выглядит немного ошеломляюще. Но это всего лишь трехсторонняя развилка на дороге. Если x меньше, чем y, скажем так, иначе, если x больше, чем y, скажем так, иначе, если x равен y, тогда продолжайте и скажите, что они равны. И в C, мы можем сделать этот перевод довольно прямо, а также. Фактически, теперь первые восемь строк кода идентичны предыдущим, за исключением этой средней здесь, где я добавляю второе логическое выражение. Х больше чем у? И тогда у меня есть это третье условие, иначе, если х равен у. Но, кажется, здесь может быть опечатка или что-то аномальное. Так что-нибудь выпрыгнуть? Да уж. У меня есть двойной знак равенства, который, возможно, является просто опечаткой с моей стороны, но оказывается, что это не так. Это намеренно. Но почему? Это похоже на наш первый пример того, где Scratch не совсем точно отображается на C? АУДИТОРИЯ: Ну, потому что знак равенства похож на задание. И поэтому подсчет знака равенства активно устанавливает его равным ... ДЭВИД: Точно. Мы уже минуту назад решили, что люди - или, на самом деле, годы назад - на самом деле равные - это в контексте C будет присваивание - скопируйте значение справа в значение слева. И поэтому мы как бы загнали себя в угол. Мы, как люди, все еще как программисты, хотим иметь возможность выражать понятие равенства и сравнения. Но если мы мы уже использовали знак равенства для присваивания, нам нужен другой шаблон символов для представления равенства. И, как оказалось, люди просто выбрали два знака равенства - они немного отличаются от нуля. Причина, по которой Scratch делает это таким образом, заключается в том, что вы действительно не хотите вдаваться в эти сорняки, конечно, когда целевая аудитория - 8-летние, которые просто учатся программировать в первую очередь. Это не важно и не очень важно для нас. Но для нас будет логическое различие, потому что, если мы используем неправильное, поведение будет неправильным. Если бы у нас был только один знак равенства, мы бы буквально меняли x на y, а не просто сравнивали его. Была ли здесь рука? Да уж. АУДИТОРИЯ: Просто быстрый вопрос. Итак, если бы вы хотели выразить больше или равно, вы бы написали равным и больше чем? DAVID: Хороший вопрос. Если вы хотите выразить больше или равно, как вы могли бы это сделать? Оказывается, есть способы сделать это. И если я пойду вперед и просто предоставлю себе какое-то время для рисования здесь, вы действительно сможете сделать меньше или равно или больше или равно. На обычной клавиатуре нет способа поместить их друг на друга, как вы могли бы вспомнить по математике. Вы просто кладете их рядом друг с другом. Смотря как. Мне нужен здесь двойной знак равенства, потому что я хочу явно проверить этот третий случай и сказать, что x равен y. Так что это была моя цель. Но, по логике, в этом нет необходимости. Давайте сделаем программу немного лучше. Сколько возможных случаев при сравнении двух целых чисел x и y для большего, меньшего или равенства? Ну, я вроде только что ответил на вопрос, не так ли? Три. Отлично! Там' Есть три сценария - x меньше или больше или равен или равен. И мне трудно думать о четвертом. Так нужно ли мне это количество специфичности? Что я могу сделать, чтобы немного оптимизировать себя, немного улучшить код, чтобы сэкономить немного времени на его написание и, может быть, даже на компьютере, на котором он работает? Да уж. АУДИТОРИЯ: Вам не нужно последнее условие. ДЭВИД: Да, мне не нужно последнее условие, потому что, если мы все согласны логически, что либо x меньше y, либо больше y, либо может быть равно y, ну, если есть только третий и последний случай, который может быть просто мой так называемый другой. Просто сделайте так, чтобы это был так называемый случай по умолчанию. И на самом деле, хотя большинство людей называют это чрезмерной оптимизацией, вы экономите время на компьютере. Потому что предположим, что x на самом деле равен y, и они оба имеют значение 1. Значит, 1 меньше 1, когда эта строка кода выполняется? Да или нет? Нет, очевидно нет. 1 не меньше 1. Таким образом, этот код не выполняется. Но логическое выражение оценивается, так сказать. Вопрос задан. 1 больше 1? Нет, и поэтому этот код не выполняется, но это логическое выражение. Таким образом, мы просто потратили еще один шаг или секунду, или как быстро компьютер. 1 равно 1? Да, так и есть. Так что это на самом деле печатает. Но, на ваш взгляд, вам не нужно задавать этот вопрос. И на самом деле, вы просто увеличили на 50%, сколько вопросов вы задаете. Таким образом, вы просто потратили немного времени. Кроме того, наши Mac, ПК и телефоны в наши дни, я имею в виду, они снова работают на гигагерцовой скорости, один миллиард вещей в секунду. Итак, на практике, кого это волнует, если вы задаете этот третий вопрос? И, честно говоря, если это делает ваш код более читабельным или для вашего преподавателя, или для коллеги или друга, который работает над программой для вас, тогда это здорово. Если из кода более понятно, что происходит, оставьте это так. Но это те дизайнерские решения, которые мы сейчас примем. И, возможно, эта версия программы и эта версия кода на C немного лучше спроектированы, потому что зачем писать больше кода, чем нужно, чтобы выразить ту же идею. Так что по этому поводу? Это была петля в Scratch. Это был бесконечный цикл, потому что он просто навсегда говорил привет миру. Теперь в C это становится немного менее прямым переводом. Оказывается, с использует ключевое слово while. Так что в С нет ключевого слова навсегда, но есть слово пока. И, конечно, я собираюсь использовать свои фигурные скобки - фигурные скобки, фигурные скобки, чтобы охватить следующие строки кода. Строка кода, которую я хочу найти, это просто еще один printf. Так что это точно так же, как и раньше, но недостаточно просто сказать время. Получается, что while хочет, чтобы вы задавали ему вопрос каждый раз, когда выполняется цикл. И он собирается проверить этот вопрос. И если ответ «да», он будет запускать цикл. Но если ответ на вопрос, заданный в C, будет «нет» или «ложь», он не выполнит код и просто перейдет к дальнейшим строкам кода, расположенным ниже в файле. Так что в C вам на самом деле нужна пара скобок после ключевого слова while. И тогда вам нужно задать вопрос. Вам нужно задать вопрос, как, на х меньше у или вопрос типа х больше у или х равен у. Но ни один из этих сценариев не применим, потому что вся цель этого блока Scratch в буквальном смысле сделать что-то навсегда. Так какой вопрос мы могли бы задать, ответ на который, безусловно, правдив? 1 равно 1? Мы могли бы придумать произвольный, но очень подходящий математически правильный сценарий. Мы можем просто сказать, что 1 равно равно 1. Но оказывается, что вы можете быть еще более кратким, потому что в C есть пара ключевых слов, одно из которых является истинным, а одно - ложным. И слово true по определению всегда истинно, а слово false по определению всегда ложно. Таким образом, вам не нужно придумывать какое-то произвольное, но правильное представление о том, действительно ли 1 равно 1 или 50 равно 50. Вам не нужно просто придумывать какое-то произвольное решение. Вы можете буквально просто сказать «истина», потому что это ключевое слово «истина» никогда не меняет значение. Так что, хотя это выглядит немного странно, вы заставляете что-то происходить вечно. Вы снова задали один и тот же вопрос и предположили, что на этот вопрос всегда один и тот же ответ. Есть вопросы по этому вопросу? Да, в спину. АУДИТОРИЯ: Пространства имеют значение? Можете ли вы вывести пробел между y и 0? ДЭВИД: Хороший вопрос. Имеют ли место пробелы? Короткий ответ, нет, не в этом случае. Фактически вы можете удалить все пространство здесь, кроме одного в английской фразе, и оно все равно будет функционально правильным. Вы можете даже добавить пробелы в любом месте, где хотите Вы можете сделать это выше, нажав Enter несколько раз, вкладки, пробелы вокруг слова true. Все примеры, которые я покажу здесь сегодня, и вы Посмотрим в ближайшие недели, это лучший способ сделать что-то, потому что они более читабельны. Но опять же, когда вы освоитесь с кодом или если у вас уже есть какой-то опыт, у вас уже может быть собственное мнение. И, честно говоря, это просто религиозная дискуссия среди программистов, которая является правильным способом написания вашего кода. И это нормально. Как только вы почувствуете себя комфортно, главное, чтобы вы были последовательны. Вам не нужно придерживаться одного человека или другого. Так как же этот код работает логически? Ну, первое, что компьютер, ваш Mac или ПК или ваш телефон или что-то еще будет делать, он задаст вопрос. Ну, правда. Ну, правда всегда правда. Так что он собирается приступить к выполнению строки кода. Но после этого, потому что это весь код, который Между фигурными скобками у нас может быть больше строк. Это просто короткие программы. Компьютер собирается проверить, хорошо, правда все еще верно. Да. Так что он собирается выполнить это снова. Затем он снова задаст вопрос. Правда все еще правда? Да, поэтому он будет выполнять код снова, и это будет повторяться буквально навсегда. Но что, если вы не хотите повторять что-то вечно? Что если вы хотите повторить это только 50 раз? Царапина не заставляет вас задуматься об этом. Люди просто выясняют, как отследить 1, 2, 3, 4, 5 и до 50, а затем остановиться. Это мило. Это позволяет легко использовать блок. C и многие языки не так удобны для пользователя. Позже в семестре вы увидите, что новые языки немного ближе к тому, что предлагает Scratch. Но в С, мы должны быть более явными, но это шанс использовать некоторые из этих более примитивных строительных блоков. В C эквивалент повторения будет предложением только потому, что пока. И затем, как и прежде, если мы хотим что-то делать снова и снова в этом цикле, мы будем использовать фигурные скобки, похожие на маленький оранжевый блок там. И что тогда я собираюсь делать? Я собираюсь делать это каждый раз, надеюсь, 50 раз, распечатать привет. Так что теперь мне просто нужно выяснить и посмотреть, как выразить количество раз, в частности, 50. Итак, получается, что в C - используйте круглые скобки - это будет довольно распространенной характеристикой для большого количества кода, который мы пишем , И тогда вам нужно сделать три вещи. Теперь на нас, программист, ложится бремя, чтобы отслеживать, сколько раз мы хотим выполнить этот код, сколько раз мы уже выполняли этот код, а затем постоянно следить за тем, чтобы один не превосходил другой. Таким образом, мы останавливаемся, когда достигаем 50. Итак, что же является фундаментальной конструкцией, которую мы используем для отслеживания чего-либо в программе? Счетчик, который был примером переменной. Поэтому нам просто нужно использовать переменную. Теперь это на самом деле будет в скобках на этот раз. Так что это не само по себе, как это было совсем недавно, но синтаксис тот же. Я мог бы назвать это счетчиком, но реальность такова, что в программировании принято просто использовать более короткие переменные, когда вы просто делаете что-то обыденное. И если все, что вы делаете, это зацикливание - я обозначаю целое число, это как бы многие программисты переход к имени переменной, а не более подробный, но правильный счетчик или что-то еще. Итак, это говорит, эй, компьютер, дай мне переменную под названием я. Позвольте мне сохранить целые или целые числа в нем и установить начальное значение в 0. Почему? Ну, почти все в этой комнате, вероятно, начинают считать с 1. Компьютеры, как правило, начинают считать с 0. Но почему? Какой смысл начинать отсчет с 0, возможно, исходя из прошлой недели? Почему это имеет смысл? Да, что ты думаешь? АУДИТОРИЯ: Ну, потому что это единицы и нули, и это двоичный код. ДЭВИД: Да, это только единицы и нули, и какое наименьшее число, кроме отрицательных значений, вы можете представить в двоичном виде? Ну, это просто 0, 0, 0, куча нулей. Так зачем вам тратить это представление, эту перестановку битов? Давайте просто начнем считать с 0, а затем добавим к этому. Таким образом, вы можете начать считать с 1 в C, но в большинстве языков принято считать от 0. Так что мы тоже встанем на эту ногу. И вы можете вспомнить даже то, что в нашем PBJ, для арахисового масла и желе - не для кода PDJ - для псевдокода телефона, я фактически намеренно начал нумерацию строк от 0 до 1 для 2 для той же интуиции. Итак, вот как вы тогда говорите компьютеру проверить, если вы хотите, чтобы я был меньше 50. Теперь, изначально он будет явно меньше 50, потому что ноль меньше 50, но это же условие будет проверено снова и снова и снова, когда этот цикл выполняется. И затем, вспоминая ранее, мы можем просто добавить плюс плюс переменную, чтобы добавить к ней 1. Мы можем сделать это более многословно. Можно сказать, что я равняюсь плюс 1, но это s просто более привычно писать я плюс плюс просто сказать то же самое более лаконично. Так что же будет дальше по логике? Это код, который я написал. Что компьютер делает с этим? Ну, он инициализирует i в 0 и готовится хранить в нем целые числа. Он проверяет условие на тот случай, если вы инициализировали слишком большое значение. Возможно, вы не захотите, чтобы цикл выполнялся вообще. Но очевидно, что 0 меньше 50. Таким образом, эта строка кода выполняется. Догадайтесь, что будет дальше. Да, вы, вероятно, хотите сделать «плюс-плюс», потому что вы выполнили все строки кода между фигурными скобками, даже если есть только одна. Итак, давайте идти вперед и увеличивать я. Итак, я теперь 1. Давайте теперь удостоверимся - 1 меньше 50? Очевидно. Выполните код. Я плюс плюс - это 2 меньше, чем 50. Очевидно, выполнить код. я плюс плюс-- 3 меньше 50, очевидно. Теперь идите вперед и выполните код, и снова, и снова, и снова. И в какой-то момент, мы собираемся подняться до I равно 49, а 49 меньше 50? Очевидно. Таким образом, мы распечатаем привет мир. А потом я плюс плюс плюс, а потом, это 50 меньше, чем 50. Нет. Так подождите, что похоже на логическую ошибку, нет? Должен ли я проверять, меньше или меньше 50? Да, потому что если я начал с 0, я уже провел этот дополнительный цикл. Таким образом, я могу считать от 0 до 49, что, кажется, работает, или от 1 до 50, но честно говоря, в программировании принято начинать считать с некоторого значения, а затем считать до некоторого значения, а не только потому, что. Но, по логике вещей, вы можете реализовать это в полдюжине разных способов, скорее всего. Позволять' Рассмотрим один последний пример, который позволил нам получить пользовательский ввод в Scratch. Вспомните, что мы использовали этот блок, чтобы фактически получить имя кого-то на лекции, и мы также в анимации с пряничным домом использовали его, чтобы получить да или нет - хотите ли вы кекс, яблоко или тому подобное. Так что это пример функции в Scratch, которая на самом деле принимает входные данные, такие как предложение «как вас зовут», но она также возвращает значение, которое в этом случае было просто жестко закодировано в Scratch MIT, чтобы называться answer. Таким образом, это как специальная переменная с именем answer, но фактически она возвращается пользователю. Так как мы можем думать об этом? В C получается, что вы можете выразить эту строку кода более многословно, чем раньше, но с помощью новой функции, называемой get string - так что строка get подчеркивания - это имя функции. Подчеркивание - это соглашение на языке C. Если вы когда-нибудь захотите иметь пробел, у вас не должно быть пробелов в именах функций. Так что люди просто начали использовать подчеркивания, как вы, возможно, в своих именах пользователей в социальных сетях и т. П. - это тоже соглашение. Вот предложение, которое я хочу отобразить, и я собираюсь начать называть это более формально строкой. Строка в языке программирования - это просто последовательность символов. Это слово, это фраза, это персонаж, это абзац. Это строка Все, что находится в двойных кавычках, является строкой в ​​C, а обратная косая черта n - это просто конец строки, как и раньше. У нас все еще есть точка с запятой, но это не совсем буквальный перевод того, что происходит только потому, что мне теперь нужно что-то сделать с ответом. Так что, если get string - это функция, которая фактически получает ввод от пользователя, как через его или ее клавиатуру, так же, как синий блок в Scratch, в C нам нужно быть немного более явным относительно того, куда мы помещаем возвращаемое значение от той функции, что это она возвращает. И поэтому я могу хранить в переменной с именем answer. Я мог бы назвать это как угодно. Но для согласованности с Scratch, давайте назовем это ответом. Но вспомните, что мы должны делать в Си каждый раз, когда мы создаем переменную. Мы должны быть более точными. Да уж. АУДИТОРИЯ: Определите его класс как строку. ДЭВИД: Мы должны определить его - позвольте мне назвать его типом или классом, если вы взяли предыдущий класс. Это тип, и он не будет int, потому что, вероятно, вводимые слова не являются числами. В этот раз я буду называть это строкой. И так, действительно, мы объявили бы переменную слева, сказав, что дайте мне строку, назовите ее answer и присвойте ей все, что находится справа. Ну, что справа? Справа находится то, с чем эта функция возвращает строку и хранится справа налево. Так как же мне теперь назвать имя этого человека? Ну, в Scratch я просто говорю, а затем перетаскиваю переменную answer, и все готово. Что за функция в C, с помощью которой мы можем что-то сказать на экране? Так что printf, напечатайте отформатированную строку, хотя мы до сих пор не видели никакого форматирования. Оказывается, в C вы должны фактически сказать d, если вы не передаете жестко закодированную строку или предложение, вы должны передать в printf то, что называется кодом формата или строкой формата - это первый ввод в printf. Теперь, кажется, printf принимает две вещи. Первый - это перед запятой. Вторая вещь после запятой. И мы еще не видели этого раньше в Си. Так что printf велят, иди вперед и распечатай строку, которая выглядит следующим образом. Процент S является заполнителем, а S обозначает строку. И это буквально заполнитель говорит: printf, я собираюсь дать вам строку для подключения к этому первому входу. Что это за строка? Буквально, переменная ответа. Теперь кажется, что мы прыгаем через обручи здесь. Было бы неплохо просто сказать printf, открыть круглые скобки, ответить, закрыть круглые скобки, точку с запятой и покончить с этим. Это просто не то, как работает printf. В старых версиях вы могли бы сделать что-то более простое, как это. Но, честно говоря, мы обычно он не собирается распечатывать только то, что напечатал человек. В конце концов, в данный момент это глупый пример. Я набираю слово. Вы просто говорите это на экране. В Scratch мы уже решили, что это отстойно. Было бы неплохо, по крайней мере, иметь программу, не просто сказать, Дэвид или как там ее зовут ... но что мы делали в прошлый раз? Как привет запятая Дэвид. Но это, кажется, дает нам такую ​​возможность. Прямо сейчас я буквально просто печатаю имя человека на С, но позвольте мне немного изменить это так же, как мы делали это на Scratch. Напомним, что в C мы сделали этот зеленый блок соединения, где я буквально пропустил соединение с двумя аргументами. Первый был привет запятой. Вторым был ответ, и это связано. Это объединило эти две струны. Ну, в C, благодаря printf, мы можем сделать то же самое. Это просто другой синтаксис. Printf сначала получает один аргумент, то есть строку, которую вы хотите отформатировать, т.е. F в printf. Но на этот раз я собираюсь буквально сказать HELLO запятую в процентах от S для строки, а затем дать printf второй аргумент, который является его инструкцией, чтобы идти дальше и подключать эту переменную к тому, чем является этот заполнитель. И вот здесь мы теперь эффективно соединили две строки и, таким образом, родилась наша первая отформатированная строка. Ну, есть вопросы по этому поводу? Да уж. АУДИТОРИЯ: Что делать, если вы хотите сказать что-то дополнительное после этого? ДЭВИД: Что, если ты захочешь сказать что-нибудь еще после? Вы, конечно, могли бы продолжить логику. Вы не должны заканчивать это цитируемое выражение процентом S. Вы могли бы сказать, привет, запятая, процент S, запятая, приятно познакомиться. И то, что будет делать printf, это только заменит ту переменную, которая называется answer, где процент S. И если вы хотите дать 2% S, вы можете просто добавить еще одну запятую здесь и передать другую переменную и третью переменную и даже больше, таким образом форматируя строку еще более подробно. Вопрос здесь. Да уж. Другие вопросы? Да, в спину. АУДИТОРИЯ: Как вы проводите различие между заполнителями, если у вас разные переменные? ДЭВИД: Как вы проводите различие между заполнителями, если у вас разные переменные? Это порядок слева направо. Так что в данном случае это тривиальный пример, потому что есть только одна переменная и один заполнитель. Но если, как вы намекаете, у меня было несколько процентов S что-то что-то, процент S что-то что-то, Я бы просто передал printf первую переменную через запятую, вторую переменную через запятую третью переменную и так далее слева направо. Другие вопросы? Да уж. АУДИТОРИЯ: Почему нет обратной косой черты? ДЭВИД: Черт возьми. Потому что я облажался и не включил это, и я собирался исправить это быстро после класса. Ошибка, это ошибка. Да уж. АУДИТОРИЯ: Что если вы хотите использовать int дважды в строке? Итак, вы хотели сказать, привет, Дэвид, привет, Дэвид. ДЭВИД: Конечно, точно такая же вещь - ответ через запятую, ответ через запятую с 2% S. Если вы хотите сказать одну и ту же переменную дважды в двух местах по любой причине, два заполнителя, а затем ответьте через запятую, чтобы вставить это дважды. Другая очередь ... да. АУДИТОРИЯ: Является ли процент универсальным заполнителем с точки зрения целых чисел? ДЭВИД: Нет, и мы скоро увидимся с некоторыми другими. Оказывается, есть и другие. Это процент от целого числа, и будет еще больше - процент с для одного символа и более. Другие вопросы? Да уж. АУДИТОРИЯ: Поскольку обратная косая черта n является переменной, вы бы поставили ее после n? Так вы положите это в цитату? ДЭВИД: Хороший вопрос. Если бы у меня было правильно - и если бы это был не PDF, я бы просто отредактировал его на лету - если бы у меня был процент n, он всегда должен идти в отформатированной строке в первом аргументе. Таким образом, единственное, что следует после первого аргумента printf - это переменная запятая переменная запятая переменная. Другие вопросы? Итак, давайте продолжим и сделаем что-нибудь с кодом. Я собираюсь пойти дальше и открыть другое окно, и это инструмент, называемый CS50 Sandbox, и это инструмент через Интернет, с помощью которого вы действительно можете играть с кодом. И я' Через минуту я покажу, как я доберусь до этого конкретного места, но позвольте мне сначала объяснить пользовательский интерфейс так же, как мы начали разговор с Scratch. Поэтому мне нужно место для написания кода. Реальность такова, что я мог бы просто использовать свой собственный Mac. Я мог бы просто использовать свой собственный компьютер. Честно говоря, я мог бы даже использовать определенные мобильные устройства в эти дни. Но тогда у нас были бы сотни других людей в классе с немного отличающимися конфигурациями на их Максе, их ПК, их телефонах и тому подобное. И поэтому у всех будет разное программное обеспечение и разные настройки, и они просто никогда не будут работать очень хорошо. Таким образом, в начале курса мы просто стандартизируем все, фактически используя веб-среду, аналогичную Scratch, благодаря которой у всех нас будет доступ к одному и тому же компьютеру, но виртуализированному в так называемом облаке. Если ты' Вы когда-нибудь задавались вопросом, что такое облако, это просто сервер других людей где-то в Интернете, который люди могут использовать бесплатно или арендовать, и им не нужно размещать эти физические серверы самостоятельно. Итак, CS50 Sandbox, как и Scratch, представляет собой облачное приложение, написанное кем-то другим, которое размещено в Интернете, и пользовательский интерфейс, на первый взгляд, выглядит именно так. Есть только две составляющие. В верхней части пользовательского интерфейса CS50 Sandbox находится всего лишь редактор кода, очень простой текстовый редактор, похожий по духу на Google Docs и Microsoft Word и т. Д., Но гораздо более простой. Там нет форматирования. Там нет смелой облицовки и центрирования. Вы можете просто ввести слова текста. Здесь находится так называемое окно терминала, но мы вернемся к этому через минуту. Позвольте мне пойти дальше и написать свою первую программу. Позвольте мне пойти дальше и написать: включить стандартную точку ввода-вывода в основной пустоте, открыть фигурную скобку printf hello world backslash и точка с запятой, готово. Теперь мало кто в этой комнате мог бы быстро создать очевидную программу, если у вас нет предыдущего опыта. И если вы взяли APCS или что-то еще, это выглядит как Java, но не совсем так. Но это моя первая программа. Теперь вспомним, что раньше это была черно-белая программа, которую мы видели на слайде совсем недавно. И даже если вы не совсем понимаете, что делает весь этот прикольный синтаксис, у всех нас, вероятно, была интуиция того, что делает эта программа, то есть просто распечатывать слова в конце дня, привет. И мы будем немного дразнить, что делают все эти различные линии. Но самое интересное то, что выделено здесь зеленым цветом, и это только одна из особенностей песочницы CS50. Он будет раскрашивать разные концепции в вашем коде, чтобы они просто выскочили на вас. Цвета на самом деле не там. Вы не должны раскрашивать вещи самостоятельно. Он просто делает это автоматически, так что вы можете видеть различные компоненты, точно так же, как цвет MIT - это разные кусочки царапины. Так что это программа, которую я хочу назвать привет. Это в файле. Это просто вкладка вверху, которая называется hello dot C, потому что оказывается, что когда вы пишете программу на C, вы сохраняете ее в файлах в соответствии с человеческим соглашением, независимо от точки C в качестве расширения файла, так сказать. Как мне запустить эту программу? Там нет зеленого флага, чтобы нажать, который Скретч дал нам. Так как же я на самом деле запустить программу. И, честно говоря, зеленый флаг, кажется, меньше всего меня беспокоит. На каком языке понимает любой компьютер, будь то мой Mac или облачный сервер, где эта штука? Нули и единицы, верно? И мы начали с этого огромного слайда, состоящего из множества нулей и единиц, и именно к этому мы должны прийти. Но, надеюсь, мы сами не должны писать на этом уровне скуки. Таким образом, нам нужен какой-то способ преобразования этого кода из C, который мы начнем называть исходным кодом, который представляет собой код, подобный английскому, который мы видим на экране, и его приятно писать, а не только нули и единицы. Но, тем не менее, нам нужно как-то преобразовать его в нули и единицы. Таким образом, способ, которым мы можем это сделать, заключается в следующем. Если у нас есть то, что мы начнем называть нашим исходным кодом, который может быть написан, в нашем случае, на C, но вы можете написать исходный код на Java, на C ++, на Python, на десятках других языков. Исходный код - это общий термин. Это просто означает код, который мы, люди, написали. Нам нужен какой-то способ преобразования его в нули и единицы, которые отныне мы будем называть машинным кодом, который выглядит как разумное имя. Это нули и единицы, которые понимает машина. Как машина узнает, что нули и те, чтобы понять? Ну, вот и вся причина наличия процессора, центрального процессора, мозгов компьютера. Они просто встроены на заводе, так сказать, на заводе Intel, чтобы понять определенные шаблоны нулей и единиц. Но для нас сейчас важно то, что нам нужно взять исходный код, например, программу на C, которую я написал недавно, которая должна напечатать hello world, и каким-то образом преобразовать его в машинный код. Таким образом, оказывается, что это тот шаг, который люди, которые Мы пришли раньше, чем мы решили для нас. Другие люди уже написали программы, которые мы собираемся начать вызывать компилятором, который позволяет нам преобразовывать исходный код в машинный код. Это просто еще один шаг. Этого шага не было в Scratch, но мы собираемся запустить программу, которая обычно называется компилятором, которому мы передаем нашу программу в качестве ввода, и мы получаем в качестве выходного машинного кода, тем самым прекрасно подводя нас к тому, что такое информатика. в настоящее время в контексте программирования - ввод исходного кода, вывод машинного кода. Алгоритм или специальное программное обеспечение, которое мы собираемся использовать в одно мгновение, называется компилятором, который просто преобразует один в другой, так что никто из нас не должен думать или писать в 0 и 1. Так что это старая школа, как ты это делаешь. С нуля,люди позаботились об этом для вас. Мы должны быть немного более внимательны к этому, и вот тут вступает в действие вторая часть пользовательского интерфейса CS50 Sandbox. Обратите внимание, у меня есть мигающий запрос здесь. Слева есть знак доллара, который является обычным соглашением. Знак доллара имеет тенденцию к тому, чтобы на компьютерах такого типа отображалась подсказка. Он ждет, когда я что-то наберу, и действительно буквально мигает, ожидая, что я что-то наберу. Это пример окна терминала, и ваш собственный iMac и ваш компьютер на самом деле имеют или не могут иметь такую ​​же функцию. Просто сегодня мы все работаем с графическим интерфейсом пользователя. Таким образом, у нас есть кнопки, меню и другие вещи, которые можно перетаскивать и щелкать, но в те времена - и, как правило, в программировании - вы не беспокоитесь об этой эстетике. На самом деле ваши руки становятся еще грязнее, если только клавиатура набирает все, что вы хотите сделать. И поначалу может показаться, что это регресс. Мол, почему мы отказываемся от всех этих прекрасных удобств современных компьютеров? Но это более мощный и более явный. Это позволяет вам делать именно то, что вы хотите, отправляя команды на компьютер. Так что это мой терминал 1. Я могу создавать другие, чтобы иметь несколько окон, но это дает мне доступ к базовому серверу, к которому у меня теперь есть доступ. Поэтому, если кто-то из вас, когда придет время для первого набора проблем, войдет в один и тот же инструмент, у вас не все будут иметь одинаковую среду. У всех вас есть свои собственные изолированные копии одного и того же программного обеспечения, но, так сказать, вашего собственного хранилища. Поэтому мне нужно как-то преобразовать привет точку C в нули и единицы. И как я Я собираюсь сделать это так: Clang, что означает язык C, привет точка C enter. И тот факт, что я вижу, что ничего не происходит, на самом деле является удивительной вещью, потому что, честно говоря, существует бесконечное множество вещей, которые могут пойти не так, и компьютер будет радостно кричать на вас с загадочно выглядящими сообщениями об ошибках, если что-то из этого не получится. Так что видеть ничего, кроме очередной мигающей подсказки со знаком доллара, на самом деле хорошо. Мой код как-то преобразован в нули и единицы. Где эти нули и единицы? Ну, по соглашению, они хранятся в файле, который странно и исторически называется просто точкой, и мы можем это видеть. Если я щёлкну здесь по значку этой папки, вы увидите мой файл hello dot C и другой файл, который теперь называется dot out. Это означает сборку, но по историческим причинам. Теперь позвольте мне закрыть значок папки, потому что мы обычно не собираемся использовать графический интерфейс пользователя. Как мне запустить эту программу? Я не мог просто дважды щелкнуть по значку. Это не Mac. Это не ПК. Это облачная среда Linux. Linux - супер популярная операционная система. Его используют многие ученые-компьютерщики, множество веб-сайтов, множество серверов. Фактически, почти каждый веб-сайт, который вы посещаете в эти дни, работает, если не Windows, от Linux, и его разновидности, называемые Unix и другими разновидностями, все еще. Это просто очень популярная и часто бесплатная операционная система, которую использует сама CS50 Sandbox. Чтобы запустить файл, называемый точкой, который находится в этой папке, так сказать, даже если вы не видите его графическую версию. Вы буквально просто набираете точку. Совершенно неочевидное и довольно глупое название для программы, но это эквивалентно двойному щелчку на значке на вашем Mac или ПК. Позвольте мне идти вперед и нажмите Enter. И когда я это сделаю, я должен увидеть что? Привет, мир. И здесь мы идем. Вау, это наша первая программа. Он делает не так уж много, но, по крайней мере, делает то, что обещали. И это равносильно тому, что в Scratch просто сказать на экране привет миру. Теперь, чтобы быть справедливым, было задействовано больше шагов, и Бог знает, что нужно было написать более загадочный код. Но, в конце концов, все, что мы сделали сейчас, это переосмыслили логику прошлой недели на этом новом языке, но теперь мы собираемся очень быстро представить новые кусочки головоломки, но на языке C. Но сначала давайте разрешим эту небольшую головную боль. Я не хочу говорить друзьям, как, эй, все, Приходите запустить мою программу Dot Out. Давайте дадим ему настоящее имя. Предположим, я просто хочу позвонить моей программе, как вы можете скачать из App Store или Google Play Store. Программы имеют названия. Так как мне это сделать? Что ж, получается в терминальном окне, так называемой среде командной строки, которая является просто причудливым способом сказать, что вы пишете строки команд с помощью клавиатуры, вы можете фактически передать то, что называется аргументами командной строки, дополнительные входные данные для программы, которые являются просто словами, которые вы вводите на клавиатуре, которые говорят ему, как себя вести. Так что вместо того, чтобы просто запускать clang в hello dot C, я на самом деле собираюсь быть более явным и скажу clang, пожалуйста, выводите - как это подразумевается буквально набирая dash 0 для вывода - файл с именем hello. Так что это немного более многословно - привет, черт побери, или Извините, Clang Dash O привет привет точка C. Но сейчас это все равно будет преобразовать исходный код в машинный код, но он сохранит его в файле с именем hello. И действительно, теперь у меня есть привет точка C и точка привет, как изображено в маленькой графической папке там. Так что теперь я могу вместо этого запустить точку слэш привет. Что это должно сказать? Надеюсь, то же самое, введите. Итак, это все. Это так называемые аргументы командной строки, и это просто старый способ, позволяющий текстовым командам вести себя немного иначе, чем по умолчанию. Но, честно говоря, это быстро станет утомительным. Мы не собираемся писать наш код, а затем каждый раз, когда мы хотим преобразовать его в нули, и те, кто его запускает, на самом деле запоминают эти магические заклинания команд. И поэтому люди тоже их абстрагировали. Оказывается, что, если вы хотите превратить программу из исходного кода в машинный код, есть другая команда, которую вы можете использовать. И вы можете буквально набрать make hello, где hello - это имя программы, которую вы хотите создать, эта программа, имя которой make, по умолчанию будет искать файл с именем hello dot C, поэтому вы сэкономите время на его указание. Нажмите Enter сейчас и, о боже, посмотрите, что он только что сделал. У него еще больше вариантов конфигурации, и мы, как сотрудники CS50, настроили песочницу CS50, чтобы иметь эти различные функции. И хотя мы не будем сейчас вдаваться в детали, я собираюсь махнуть рукой на то, что они на самом деле делают. Они просто делают возможными дополнительные функции, к которым мы в конечном итоге доберемся. Но в противном случае это будет команда, которую вы все должны вводить всего за две, три или четыре недели, и никто никогда не сможет вспомнить это. Я конечно не мог. Так что Make просто автоматизирует это для вас. Но когда вы запускаете Make, Make не является компилятором. Make не является чем-то средним при преобразовании исходного кода в машинный код. Это всего лишь вторая программа, которую несколько лет назад написали некоторые люди, которая использует clang автоматически для достижения того же результата. Потому что людям надоело печатать подобные вещи. Итак, кто-то создал программу под названием Make, которая делает это за нас. Любые вопросы? Давайте добавим немного потом к этой программе. Вместо этой версии привет, позвольте мне получить некоторый пользовательский ввод и фактически сделать что-то с этим. Предположим, я действительно хочу получить имя пользователя, а затем распечатать его. Ну, мы видели спойлер для этого всего минуту назад, но позвольте мне продолжить и добавить сюда эту программу. Теперь у меня есть вторая строка кода, и я хочу получить строку от пользователя. И с какой функцией я получаю строку от пользователя? Получить строку было одно, и я помню, я могу получить подчеркивание строки открывающей скобкой. А потом я должен сказать, так сказать, аргумент, например, дать мне свое имя - или вообще, что мы говорили раньше? Как тебя зовут, я думаю была бы обратная косая черта n точка с запятой. Теперь недостаточно просто получить строку. Что я хочу с этим сделать? Да, хранить в переменной. Какой тип переменной? Строка. Поэтому мне просто нужно перейти в левую часть этой строки кода и сказать: «Хорошо, хорошо, дайте мне строку. Я назову это имя, но я мог бы назвать это х или у или что-нибудь. Но имя кажется хорошим дескриптором для него, используя один знак равенства для копирования справа налево. И теперь у меня есть это. Теперь недостаточно просто сохранить значение в переменной. Мне нужно распечатать это. Итак, позвольте мне начать с этого. Автосохранение, Песочница. Так что мне даже не нужно заходить в Файл, Сохранить или что-то еще. Позвольте мне пойти дальше и поздороваться ... о, боже мой, посмотрите на все эти ошибки уже. Ясно, что что-то не так, потому что компьютер любит говорить мне красным. И, честно говоря, это то место, где вы очень быстро сходите с ума или начинаете волноваться, потому что, о боже, я написал только две строки кода. Как у меня 20 строк ошибок как-то? Таким образом, компьютер так же запутан, как и вы. И самое главное, когда вы сталкиваетесь с такой ситуацией, когда это просто загадочный, ошибочный вывод, начинайте сверху. Даже если у вас маленькое окно и поэтому на экране быстро прокручивается куча вещей, прокрутите вверх до верха, потому что шансы есть » Одна ошибка в самом верху, и эта ошибка просто имела каскадный эффект на компиляторе. Тогда это стало действительно запутанным, и он просто продолжал выплевывать сообщения, потому что это было споткнуто рано. Итак, давайте вернемся к вершине здесь. И вот очень длинная команда, которую я сказал сделать автоматы для вас. Так что это не ошибочно. Здесь, кажется, первая ошибка, и это все еще немного загадочно. Но давайте подберем некоторую информацию. Вот знакомая фраза - привет точка C. Позвольте мне пойти дальше и увеличить масштаб здесь. Итак, привет точка C напоминает имя моего файла. Хотя не очевидно, что Clang предлагает мне взглянуть на строку 5 и затем на пятого персонажа. Так что это что-то двоеточие что-то означает символ номера строки или номер столбца, если вы смотрите слева направо. Ошибка означает ошибку. И вот тогда все становится немного сложнее. Использование необъявленной строки идентификатора - вы имели в виду стандартный n? Нет, не знаю, но я распознаю стандарт n, точнее, он похож на стандартный ввод / вывод. Но нет, я не это имел в виду. Я уверен, что этот код правильный. Ну, почему я получаю эту ошибку? Звучит как строка, в строке пять, пятый символ, прямо здесь, это неправильно. Ну, оказывается, нет такой вещи, как строка. C, язык, имеет целые числа, и это имеет логическое значение, оказывается. И у него есть персонажи и несколько других вещей. Это на самом деле не имеет строк. Строки - это слово, которое полезно для описания последовательностей символов, абзацев, слов. Но строка не является типом. Это не тип переменной, если вы не сделаете это так. И на самом деле, это одно из упрощений, которое мы делаем. Оказывается, что всего лишь за первые пару недель курса мы должны добавить одну строчку кода. Нам нужно не только включить стандартный ввод-вывод, который мы вскоре расскажем, но и так называемую библиотеку CS50. Так что в CS50 задействовано много людей, и со временем мы решили, вы знаете, что мы могли бы сделать первый час CS50 немного проще, а рампу - немного чище для людей без опыта, просто придумав некоторые функции, такие как возможность получать строки от пользователя, сами по себе. Получается, что get string также не является функцией, поставляемой с C. Это, так сказать, нестандартная часть головоломки, которую сделал CS50. И где мы создали эту функцию, по сути, находится в файле CS50 dot age. И поэтому, включив точку возраста, вы теперь получите доступ к большему количеству кусочков головоломки, если хотите, что мы создали для вас. И оказывается, что эта строка кода, которая была здесь раньше, также дает вам возможности. Мы просто делаем то, что делают все в программировании, то есть решают проблему один раз, а затем позволяют другим людям использовать это решение. Предположить. Какие функциональные возможности на самом деле реализованы в файле, называемом стандартным вводом-выводом точка-точка? Это просто файл где-то на сервере, который на самом деле поставляется с C, и предоставляет вам такие удобные функции, как что? Скажи еще раз. Еще раз. Printf. Оказывается, что средство, с помощью которого вам разрешено использовать здесь функцию printf, заключается в том, что вы должны включить файл, в котором она объявлена. Итак, несколько лет назад некоторые люди буквально написали функцию, кусок головоломки, называемую printf, и они выяснили, как на самом деле рисовать символы на экране. Затем они сохранили информацию об этой функции в файле, называемом стандартным IO dot age. Если бы я не включил эту, казалось бы, загадочную строку кода в самый верх моей предыдущей программы, даже эта программа hello world не сработала бы, потому что clang, компилятор, не знал бы, о чем я говорю. Что такое printf? Я не знаю, что это, если вы не скажете, чтобы он также включал этот файл, который люди написали много лет назад, в котором был создан printf. И теперь, если я хочу использовать get string, а также новое ключевое слово string, мне нужно сказать clang компилятору, а также искать в CS50.h дополнительные функциональные возможности, такие как string и get string. Итак, позвольте мне пойти дальше и попробовать это снова. Я собираюсь очистить свой терминал здесь и просто повторить ту же самую команду - поздороваться, войти, черт побери. Теперь у меня есть другая ошибка. Но но, но прогресс. Ну, нет, просто так много ошибок, как и раньше, но разные. Заметьте сейчас - подождите, это было раньше. О нет, прости. В нем меньше ошибок. Вот где я запускал команду минуту назад, и теперь вместо этого я получаю эту ошибку. Так что прогресс. Теперь моя ошибка не в строке 5. Это в строке 6, хотя забавный факт, строка 6 раньше была строкой 5. Так что, очевидно, она все еще связана с проблемой. Итак, давайте прочитаем сообщение об ошибке. Проблема в строке 6, что неудивительно в том, что там есть. Но на этот раз все по-другому - ошибка, неиспользуемое имя переменной. Это я понимаю, даже не будучи программистом. Что это значит? Да уж. Может быть, объявить перед использованием, но оказывается, что вы так и объявляете. Но я на самом деле ... да. АУДИТОРИЯ: На самом деле вы не используете объявленную вами переменную. DAVID: Да, я просто трачу время на компьютер. Я создаю это. Так что строка 6 слева верна. Эй, компьютер, дай мне строковую переменную, назови ее name и вставь в нее значение. Но какой смысл в этом упражнении, если вы никогда, как вы говорите, ничего с ним не делаете. И на самом деле, вспомните из слайда минуту назад, как мне с этим что-то сделать? Ну, это не то, как вы что-то делаете с этим. Если я продолжу успешно запускать эту программу, что я на самом деле увижу на экране? Буквально привет имя, ПРИВЕТ запятая ИМЯ, очевидно, не правильно. Итак, как мне подключить переменную? В чем был подвох? Да, процент S для строки, код формата, так сказать - отсюда и название printf. И тогда мне нужно передать второй аргумент в printf, и я делаю это с запятой, а затем с именем переменной, которую я хочу подключить. Теперь обратите внимание, что в этой строке есть две запятые. Если я увеличу масштаб, заметим, что есть две запятые, но есть только два аргумента или ввода для printf. Входные данные для функции обычно называют аргументом или также называют параметром. Итак, есть две запятые, но эта английская запятая просто отделяет привет от имени человека. Эта белая запятая здесь, с цветовой кодировкой, потому что Песочница делает для меня фактически отделяет - извините меня - первый аргумент от второго аргумента. Так что теперь, в третий раз, заставьте привет войти. Боже мой, спасибо. Теперь это сработало. Она по-прежнему выдает эту довольно длинную загадочную команду белым цветом, но это нормально. Это, опять же, автоматизированная команда, которую Make делает возможным для нас. Но тот факт, что я не вижу красного, ошибок нет, просто мигающее сообщение означает, что моя программа была создана. Итак, позвольте мне пойти дальше и сделать ... как мне запустить программу, если она называется привет? Да, косой чертой привет, и мы увидим, почему у вас есть глупая точка в начале. По сути, это означает, что нужно запустить программу с именем hello, которая находится прямо в вашей текущей папке на сервере - точка-косая черта hello. Как тебя зовут? Очень хорошо. Дэвид, входи. Привет Дэвид. Интересно. Давайте сделаем один твик, потому что я сделал это случайно, как вы заметили ранее. Что, если я остановил, например, одну из этих обратных косых черт, которая буквально сейчас не говорит компьютеру переместить курсор на другую строку? Итак, позвольте мне продолжить и перезапустить программу. Подождите минуту. Это выглядит так же. Я только что изменил код, но он продолжает вести себя точно так же. Где мое замешательство? Я не' перекомпилировать это. В отличие от Scratch, который удивителен, потому что вы просто нажали зеленый флаг и он снова запускает код, у нас есть второй промежуточный шаг. Я должен повторно запустить код. Теперь, как вы выходите из программы? Я мог бы просто нажать Enter. Вы также можете нажать Control C для отмены, и это просто избавит вас от любой путаницы, в которой вы находитесь. Позвольте мне продолжить и повторить, сделать привет - вроде бы все в порядке - точка косая черта привет, введите. Хорошо, вот почему у меня были все эти обратные слэши. Позвольте мне увеличить масштаб происходящего. Я имею в виду, что это не выглядит ужасно, но, честно говоря, это как-то раздражает меня, если это то, на что похожа моя программа, когда я печатаю ввод пользователя. Я имею в виду, это выглядит глупо. Как минимум, я должен добавить пробел. Может быть, я могу поставить обратную косую черту n, чтобы переместить персонажа. Это пользовательский интерфейс сейчас. Это не совсем логика. Это просто эстетика, но я думаю, что это выглядит глупо. Вот почему у меня все время есть обратная косая черта n, но именно поэтому они должны быть там, чтобы указывать компьютеру на фактическое размещение вещей там, где вы хотите. Кроме того, вы знаете, что, хорошо, мне это не нравится. Control-C для отмены. Позвольте мне положить это обратно. Что произойдет, если я избавлюсь от этого? И позвольте мне пойти дальше и сначала перекомпилировать код, когда вы заметите, что точка косая черта привет ввод. Хорошо, я убрал эту эстетическую головную боль, войди. Я имею в виду, что это не совсем та же проблема, но это выглядит глупо, потому что знак доллара просто представляет мое приглашение, где я должен вводить команды, и все же привет запятая, приглашение Дэвида. И это просто грязно. Вот почему у нас были все эти новые линии. Теперь вы спросили ранее, Что делать, если вы поместите новую строку в другом месте строки. Ну, предположим, я это делаю. Предположим, я поставил пару из них. Позвольте мне сделать это и без пробелов вообще. Теперь это выглядит немного странно, но компьютер просто будет интерпретировать это буквально - выведите HELLO запятую новую строку, замените в строке процент S, а затем еще одну новую строку. Так сколько строк вывода это будет отображать? Я слышал четыре. Другие ценности? Посмотрим. Давайте продолжим и сделаем привет, а затем запустим dot slash hello, как вас зовут, как и раньше, введите hello запятую, новую строку David-- итак, четыре строки, конечно, или две строки от самого компьютера. Итак, просто напомнить, с таким кодом, сколько функций я использовал в этой конкретной программе? Сколько функций? Итак, это два - printf, который мы использовали, и получить строку, которая является новой. Где объявлена ​​строка? CS50 точка h, printf между тем объявлена ​​в стандартном вводе-выводе стандартного ввода-вывода. Между тем, строка, этот тип данных также исходит из самого CS50, а затем мы использовали коды формата и использовали переменные для достижения того же результата. И давайте просто забить этот дом. Брайан и Карим, ты не против на секунду? Или кто там? Эрин, давай. Просто чтобы прояснить это, потому что теперь есть некоторая терминология, которую мы хотим использовать. Посмотрим. Вы хотите быть ... вы получите строку. Итак, у нас есть несколько тегов имен, которые вы можете использовать на мероприятиях. Так что Эрин должна быть подчеркнута. Если вы хотите пойти дальше и надеть это. И Брайан, ты хочешь быть принтом. Поэтому мы не будем разыгрывать все наши действительные программы, потому что это быстро станет очевидным. Так что Брайан это принт. Так что этот момент, хотя, остается в том, что это хорошо, что у меня есть несколько коллег, с которыми я работаю здесь, потому что я действительно не хочу делать всю тяжелую работу, чтобы все произошло. И поэтому, если я вместо этого являюсь программистом или организатором целого ряда происходящих вещей, я действительно могу реализовать этот код сейчас более физически. И давайте на минутку сосредоточимся на том, каковы основные выводы. Во-первых, функции могут принимать данные. Эти входные данные называются аргументами или параметрами, и функции могут возвращать вещи. Они могут иметь возвращаемые значения. Printf, например, принимает ли он ввод? Например, что вводит printf? Да, как привет мир. Все, что вы хотите напечатать, по определению является входом для printf. Возвращает ли print значение? Это действительно делает что-то на экране, конечно. Он печатает вещи на экране, но у нас нет Мы не видели, чтобы это возвращало что-то, потому что мы не видели это со знаком равенства слева. Теперь оказывается, что printf возвращает вещи. Просто это не часто так полезно для использования, но мы пока только видели, что printf принимает входные данные - подробнее об этом в другой раз. Тем временем получить строку - получает ли строка входные данные? Сколько входов? Только один, подсказка, которую вы хотите, чтобы человек увидел. Первое приглашение, которое я использовал, было как тебя зовут. Я мог бы сделать подсказку что угодно, но это вопрос, который задает get string. Возвращает ли строка значение? Оно делает. Это, конечно, в Scratch называется ответом, жестко закодированным как ответ. Мы можем хранить в любой переменной, которую мы хотим. Итак, позвольте мне пойти дальше и реализовать эту программу. Эрин, иди принеси мне строку и спроси человека, как его зовут. Итак ... Эрин: Как тебя зовут? АУДИТОРИЯ: Элизабет. ДЭВИД: Так она записывает Элизабет сейчас. Но Эрин получила от меня информацию. Эрин, иди, возьми чье-нибудь имя и спроси, как его зовут. И теперь вы производите продукцию для меня. Спасибо. Это возвращаемое значение, сохраняющее значение Элизабет, и я собираюсь пойти дальше и забрать его в переменной с именем name, как здесь, на листе бумаги. Хорошо, Брайан, не могли бы вы пойти и поздороваться с Элизабет. Так что здесь происходит? Я просто делаю меньше работы. Я пишу эту программу ... Эрин, иди, найди чье-нибудь имя, Брайан, не мог бы ты распечатать это. Это то, что я делал, это программирование, просто делегирование функциональности другим функциям или, в данном случае, другим людям, которые знают, как это сделать. И, честно говоря, мне не нужно знать, как Эрин получила это имя. Она только что получила это. Я не Я должен знать, как Брайан писал на экране именно в таком стиле английского языка. Я просто знаю, что он может это сделать, и теперь моя программа завершена. Большое спасибо обоим из этих двух здесь. Мы продолжим в следующий момент следующим образом. Поэтому все это время мы считали само собой разумеющимся, что у нас есть настоящий компьютер, на котором мы можем выполнять код, и я продолжаю говорить, что внутри Intel это глупый лозунг, который вы видите внутри большинства компьютеров Mac и ПК с аппаратным обеспечением, которое у них есть. Но процессор - это всего лишь мозг компьютера. И в конце дня, напомните, что цель состоит в том, чтобы на самом деле компьютер превратил что-то вроде этого - исходный код - в реальный машинный код, нули и единицы. И это все, что Clang на самом деле делал для нас. Конечно, мы только сейчас поцарапали поверхность того, что мы можем сделать. Оказывается там Это будут не только те команды, которые мы можем запустить, но и другие функции C и, в частности, библиотека CS50. До сих пор мы только видели, как получить строку, но вы можете получить целые числа и символы, а также такие забавные вещи, как числа с плавающей запятой и двойные числа, которые фактически открывают червячные черви в отношении проблем, которые могут возникнуть в компьютере. И оказывается, что вы можете хранить различные типы переменных, не только целые числа и строки, но и bool для логических и символы для символов. И вы можете отформатировать эти вещи в printf. Мы видели только процент S. Я упомянул процент i ранее, но есть еще несколько других. Но у нас здесь много возможностей. Но давайте продолжим и сделаем, скажем, пятиминутный перерыв, чтобы дать всем немного передышки, включить музыку, вернуться в пять и погрузиться глубже в более сложные программы на C. Итак, напомним, что мы начали со сравнения некоторых блоков Scratch с соответствующим кодом C, но на самом деле мы еще не использовали большинство этих блоков в C. Итак, давайте попробуем несколько примеров, некоторые из которых я напишу на лету при наборе кода на собственной клавиатуре, некоторые из которых мы уже имеем на веб-сайтах курса. Таким образом, мы можем просто открыть их, чтобы сэкономить время. Но позвольте мне обратить ваше внимание на то, что такое CS50 Sandbox, потому что это был шаг, который я пропустил ранее. CS50 Sandbox можно использовать для программирования в группах языков. Сначала мы будем использовать его в семестре для C. Но если вы уже писали на Java или Python или на любом количестве языков, когда вы переходите на Sandbox.CS50.io, вы можете просто выбрать язык, на котором хотите писать. А потом внизу вы Посмотрим, вы можете указать имя файла, который вы хотите создать. Например, раньше я выбрал C сверху, а затем снизу набрал Hello dot C, потому что это было имя файла, который я хотел. И тогда я нажал кнопку «Пуск». И это привело меня именно к интерфейсу, в котором мы только что написали hello dot C, где мой редактор кода появляется вверху, мое окно терминала появляется внизу, и тогда мне разрешено просто начать писать код. Так вот как мы попали туда, где мы были. И если вы хотите следовать некоторым из этих примеров, обратите внимание, что на веб-сайте курса мы подготовили все из них. И вы можете на самом деле щелкнуть ссылки на веб-сайте курса и открыть собственную копию песочницы с этим кодом. Так что, если Wi-Fi сотрудничает, вы Добро пожаловать, чтобы повозиться, поиграть и запустить те же команды. Но все также на сайте курса после. Таким образом, вам не нужно печатать все. Итак, давайте продолжим и приведем лишь краткий пример. Я собираюсь назвать это int dot C просто для того, чтобы мы могли подкрепить кое-что из того, что мы сделали минуту назад. Вместо того, чтобы получить строку, как мы сделали с нашим примером hello, позвольте мне пойти дальше и просто получить, скажем, целое число, а затем распечатать его так же, как мы напечатали имя. Так что мне нужно идти вперед, и, только благодаря мышечной памяти, я помню, что мне нужна стандартная точка ввода / вывода h вершины, а затем в основной пустоте и затем в фигурных скобках. И тогда я могу фактически получить информацию. Так что до вызова get string была функция. Оказывается, есть еще одна функция get int, если вы хотите получить целое число. Так что я могу на самом деле вызвать get int, и я могу сказать что-то вроде integer это приглашение. Как, эй, человек, пожалуйста, дайте мне целое число. Мне нужна точка с запятой в конце этой строки. И как я могу сохранить возвращаемое значение get int в переменной? Когда Эрин передала мне лист бумаги со строкой, я вручил лист бумаги с номером, как мне его где-то хранить? Что я должен буквально печатать здесь пятую строку? Int пространство, а затем число или что-то. Поэтому я собираюсь называть это просто потому, что это целое число, но я мог бы назвать его числом или чем-то еще. А потом я собираюсь пойти дальше и распечатать это. Так что printf-- скажем, что-то вроде привет я косая черта и точка с запятой. Не совсем правильно, правда? Это будет буквально печатать привет запятую. Как мне на самом деле заменить что-то в? Ну, мы Я видел только, как заменить в строках, но я думаю, что я испортил ранее ответ. Если вы используете процент i, это говорит: «Эй, компьютер, поставь здесь целое число». Затем мне нужен второй аргумент для printf, так же как мы передали Брайану аргумент. И я сказал, что я. Я хочу сказать, что я здесь. Но эта программа еще не совсем правильная. Он находится в файле с именем int dot C. Я включил стандартную точку ввода-вывода h. Это в основном, и так, что является главным? Что ж, сегодня мы собирались махнуть руками на это. Но int main void, пожалуй, самый загадочный способ, который вы можете сказать, эквивалентно тому, когда щелкнул зеленый флаг. Это все, что происходит, и мы вернемся через несколько недель, чтобы узнать, почему он int, почему он главный, почему он пуст. Но сейчас люди много лет назад просто решили, что когда вы пишете программу на C и хотите запустить ее, вам буквально нужно ввести int main void с этими круглыми скобками с этими фигурными скобками, и это эквивалент Scratch, когда щелкает зеленый флаг. Но эта программа не будет компилироваться, и я даже не хочу вызывать стресс от просмотра этих ошибок. Как я могу аннулировать это? Да, мне нужно научить компьютер, который получает int, существовать, и я знаю, как это сделать раньше, включая так называемую библиотеку CS50. Технически, CS50 точка h является заголовочным файлом. Точка h означает заголовок, и это просто файл, содержащий код C, в котором создаются функции. Подробнее об этом в другой раз, но это только дает нам доступ к printf. Так что, если я не сделал опечаток, я должен быть в состоянии скомпилировать эту программу, выполнив какую команду? Сделайте инт. Я мог бы сделать Clang. Я мог бы сделать Clang Dash O. Но сейчас я Я собираюсь сделать это проще, просто сделав int и позвольте make автоматизировать процесс компиляции этой программы. Нет сообщений об ошибках это хорошо. Позвольте мне пойти дальше и увеличить масштаб - точка косая черта int Я думаю, что запустит программу. Целое число, а как насчет 42? Привет 42. И обратите внимание, я могу сэкономить время сейчас. И если я хочу запустить его снова, мне не нужно делать slash int все время. Оказывается, что в среде Linux такого типа, в этой операционной системе под названием Linux, вы можете переходить вверх и вниз и видеть предыдущие набранные вами команды, а также некоторые другие, которые относятся к конкретной системе. А если вы просто настроите и войдете, вы можете запустить его снова, набрать 50 на этот раз и посмотреть другой вывод. Так что, есть вопросы по такому примеру? Нет? Хорошо, позвольте мне идти дальше, и я собираюсь сэкономить время в этой среде. Я могу на самом деле создавать файлы здесь, если я хочу, нажав на значок папки, нажав плюс. И тогда я действительно могу сказать, дайте мне файл с именем float dot C. Так что это равносильно возвращению в главное меню и вводу имени файла. Сейчас я собираюсь сделать это немного быстрее в этой графической среде. И я хочу назвать его плавающей точкой C. Это немного странное имя, потому что, по крайней мере, когда вы росли, вы, вероятно, узнали о целых числах. Вы, вероятно, узнали о действительных числах, числах, которые могут иметь десятичные точки, а затем о вещах после десятичной точки. В компьютере те вещи с десятичными точками называются значениями с плавающей запятой или числами с плавающей запятой. И вы можете думать об этом как десятичная точка может плавать влево или вправо, в зависимости от того, насколько велико или мало число или насколько точны числа после десятичной точки. Это поплавок. Итак, позвольте мне продолжить и реализовать очень похожую программу - включить CS50, точка h, включить стандартную IO h int main void - и после 20 лет этого вы сможете сделать это так быстро. Теперь позвольте мне вместо этого получить число с плавающей запятой от пользователя - поэтому действительное число может содержать десятичную точку. Я собираюсь сделать это немного по-другому. Я увеличу масштаб и скажу: «Эй, компьютер, дай мне число с плавающей запятой, как называется тип данных - не int, не string, а float». Я назову это просто потому, что это звучит как поплавок, и это мило и лаконично - равно как поплавок. А потом я просто собираюсь сказать плавать. Это подсказка. Я мог сделать подсказку зеленым, что захочу. И теперь я Я собираюсь напечатать это - printf привет f, но я не хочу f. Я хочу распечатать заполнитель, и вы уже можете догадаться, что это за шаблон - процент f для новой строки с запятой f точка с запятой. Так что это похоже на одну и ту же программу три раза теперь со строкой, с int и с плавающей точкой. Но опять же, это просто для мышечной памяти и прохождения паттерна, но давайте посмотрим, что здесь происходит по-другому. Позвольте мне пойти дальше и наберите make float enter. ОК, хорошо, без ошибок. Позвольте мне увеличить масштаб и запустить его сейчас как точка с косой чертой. И позвольте мне пойти дальше и набрать номер. Я просто скажу 42. Но теперь у компьютера есть возможность сохранять большую точность. До этого это было просто целое число по определению int. Теперь это поплавок. Так что, хотя это довольно точно, эти 42,0000, это действительно настоящее число, сохраняя некоторое количество точности там. Так что получается, что мы можем сделать более интересную математику. Позвольте мне пойти дальше и просто открыть этот пример заранее. Этот будет называться int dot C. Итак, мне не нужно ничего печатать. А в целых точках C мы увидим некоторую математику, написанную в коде, который я предварительно создал, просто чтобы подчеркнуть, что вы действительно можете выполнить некоторую основную арифметику в программе. Я могу увидеть больше кода здесь, просто прокрутив вниз, и позвольте мне прокрутить это вверх, чтобы мы могли сосредоточиться на основном. И позвольте мне увеличить первые несколько строк. На этой первой строке я просто получаю int и называю это x. Недавно мы не использовали переменную с именем x. Но сейчас мы. Это не отличается логически, чем раньше. Вот достань мне другую переменную. Итак, теперь мы видим, что вы можете получить несколько переменных от пользователя, как в Scratch. И теперь в этих строках, выделенных зеленым, есть только строки форматирования - что я хочу, чтобы printf отображал? Я буквально в этой выделенной строке хочу, чтобы printf отображал x плюс y, что-то равно. Что это такое? Обратите внимание на то, что в printf замечательно то, что до того, как он будет передан во ввод, вы можете выполнять простые арифметические операции. Так что, если вы хотите добавить x и y вместе, буквально сделайте x плюс y. Тогда сумма этих чисел и будет передана printf в качестве входных данных. Точно так же, как я передал листок бумаги Эрин Брайану в качестве входных данных, я передаю не х и у Брайану в этом случае, а х плюс у или некоторое значение, фактическую сумму. Точно так же вычитание - это дефис на вашей клавиатуре. Для умножения это не х. Это было бы странно, xxy. Это вместо звездочки или звездочки на клавиатуре. Разделение - это единственная косая черта, а затем это немного забавно, но мы придумаем некоторые варианты использования для этого. Вы можете фактически сделать модульную арифметику или просто более просто остатки. Если вы сделаете x процентов y, вы вернете остаток от деления x на y. А что остаток? Так что, если x равен 20, а y равен 10, то 20, деленное на 10, в два раза лучше. Так что остаток равен 0, например. Но это было какое-то время. Итак, обратите внимание, что здесь любопытного. В этом контексте проценты не являются заполнителями. Это не процент С. Это не процент я. Это не проценты Обратите внимание, что это не внутри строки формата printf. Это в буквальном смысле математика, математический оператор, который подразумевается под другим синим цветом. Так что, если я на самом деле запустить это - пусть Иди и запусти эту программу. Я собираюсь пойти дальше и сделать наши целые числа - множественное число, потому что это имя файла - введите точечные косые черты. И позвольте мне увеличить и очистить экран. Войти. Дайте мне номер. 2 Я слышал. И другой. 10 я слышал. Таким образом, к вашему сведению, 2 плюс 10 - это 12, 2 минус 10 - это отрицательное значение 8, 2 раза 10 - это 20, 2, деленное на 10 - это 2 mod y или 2, а затем взять остаток, если деление на y означает, что mod означает 2 Так что я получаю четыре из пяти за правильность? Что тут прикольного? Ага, 2 делится на 10? Мол, я уверен, что это как 2/10 или, может быть, одна пятая или 0,2. Я имею в виду, я буду принимать любое количество ответов, но не 0. Так что происходит? Ну, это вопрос представительства. Оказывается, в компьютерной программе мы заранее решили, что я собираюсь хранить ценные бумаги. Int - это то, что не имеет десятичной точки. И все же здесь я довольно самонадеянно пытаюсь сделать 2, целое число, деленное на 10, целое число, и ожидаю чего-то другого, кроме целого числа. Нет, я буквально делаю целочисленную арифметику. Так что, очевидно, делает компьютер просто интуитивно? Почему, когда я делю x на y, как я это сделал здесь - или, в частности, в этом примере, который вы предложили, 2 делите на 10 - куда идут мои 2/10? Да, это технически что? Предполагается, что 0,2 или 0,2. Это на самом деле решение, потому что, если это 0,2, но целые числа не могут хранить десятичные точки или что-то после них, что у вас осталось? Просто ноль в начале. Так что целочисленная арифметика - это хорошо, если вы работаете с целыми числами, но если вы хотите арифметику с плавающей запятой, вам нужно будет внести некоторые изменения. И поэтому я могу это исправить. На самом деле, позвольте мне пойти дальше и написать другую программу здесь. Позвольте мне пойти дальше и открыть с сайта курса плавающую точку C. Это даст мне этот пример, который реализован с использованием значений с плавающей запятой. Так что, как только это загрузится, я увижу программу, которую я написал заранее. Это немного короче, потому что теперь я забочусь только о рассмотрении одной проблемы. И обратите внимание, теперь x и y теперь являются числами с плавающей точкой, а не целыми числами, еще одним типом данных, который существует. И я использую get float, который также взят из библиотеки CS50. И тогда эта линия почти такая же, но знаете что? Позвольте мне настроить это. Позвольте мне сделать это точно так же. Теперь эта строка, которую я выделил, точно такая же, как и раньше. Так что, если я введу тот же номер, пусть s идти вперед и увеличивать масштаб и делать поплавки во множественном числе и разбивать поплавки. Я дам ему 2 и 10, и я надеюсь, что теперь посмотрим? 0.2. Это очень хорошо, довольно точно. Но вы знаете, что? Ненавижу вам говорить, но давайте посмотрим немного дальше. По умолчанию получается, что когда вы задаете процент f, вы видите только несколько десятичных разрядов, например, пять или около того, по умолчанию. Позвольте мне увидеть еще несколько. Это был один один, два, три, четыре, пять, шесть очков после десятичной точки. Так ты знаешь что? Я собираюсь сказать, эй, компьютер, дай мне десятичные точки. Это выглядит совершенно загадочно, и вам просто нужно запомнить это или посмотреть, если вы забудете. Если вы поставите точку и число между процентами и f, это загадочный способ заставить компьютер показать мне число с плавающей запятой, но с таким количеством десятичных знаков, пожалуйста. Так что это просто дает мне десятичные разряды, странно, как выглядит выражение. Теперь, надеюсь, я просто увижу еще несколько нулей. Итак, позвольте мне идти вперед и делать поплавки, и позвольте мне идти вперед и увеличивать масштаб и делать точечные косые черты 2 10 enter. ОК, все еще правильно. Позвольте мне стать немного любопытным. Давайте посмотрим много нулей, например, 50 из них. Позвольте мне пойти сюда и делать плавания, потому что я изменил код, точка-косая черта 2 10. Ха, твои учителя начальной школы солгали тебе. 2 делится на 10, по-видимому, не 0.2000000 бесконечно. Это, очевидно, 0.20000000298023223876953125, а затем все эти нули. Что, черт возьми, происходит? Где ошибка? Где моя ошибка? Где мое недоразумение? Чем это объясняется? Что ж, если я скажу вам, что внутри вашего компьютера есть такие вещи? Это ОЗУ или память, и вы Мы, вероятно, общеизвестны этой идеи Они просто хранят файлы. Они хранят музыку и видео. Вам нужна память, какое-то пространство. Место на жестком диске - постоянное хранилище. ОЗУ или оперативное запоминающее устройство является временным хранилищем. Поэтому, когда ваш ноутбук открыт, а настольные компьютеры включены или ваш телефон включен, вы используете ОЗУ для всех программ, которые вы одновременно запускаете. Если вы открываете файл, он сохраняется в оперативной памяти, но постоянно хранится на вашем жестком диске. Так что есть разные типы памяти. Но обратите внимание, это увеличено. На самом деле, это как пара дюймов в ширину и, может быть, в высоту. Так что он довольно маленький, но не имеет значения, насколько он велик. Просто важно, что он конечен по размеру. У вас есть физическое оборудование на коленях, в карманах или дома, которые настолько велики и поэтому имеют только очень много деталей и, следовательно, имеют столько транзисторов и другого оборудования, которое фактически выполняет работу по хранению информации. И поэтому, если у вас есть только ограниченный объем памяти, как в мире мы будем представлять бесконечное число чисел? Поскольку я помню из начальной школы, меня учили, что существует бесконечное число чисел, и, конечно, действительных чисел, где десятичная точка может существовать вечно. Это проблема, если вы хотите представить все возможные числа во вселенной, а это бесконечно много, с конечным количеством оборудования. Так что в какой-то момент компьютер должен начать сокращать некоторые углы. И так, что вы На самом деле мы видим, что компьютер настолько близок к тому, чтобы точно хранить эту дробь для вас, и я стал немного жадным. Я посмотрел слишком далеко вправо. И, разумеется, это бесконечно малые значения. Это не очень, очень отключено, но отключено, потому что они не могут ожидать, что компьютер будет представлять бесконечное число значений, используя ограниченный объем памяти. Он должен округляться здесь или там и быть неточным, так сказать. Так это проблема? Я имею в виду, мы бы никогда этого не узнали, если бы я не стал жадным и посмотрел бы на 50 десятичных знаков вместо 7, что было уже довольно точно. Это проблема? Да, как почему? Зачем? АУДИТОРИЯ: Если вы используете функцию равенства, [неразборчиво] .. ДЭВИД: Да, это хорошо. Логично, что если я начну использовать equals equals для сравнения вещей на равенство, Мне будет действительно трудно когда-либо сравнивать что-то для 2/10, поскольку это значение, потому что я собираюсь буквально запомнить или записать или выяснить это значение и сравнить с этим, а не просто более свободно сравнивать с 0,2. И это правда, вы никогда не должны сравнивать значения с плавающей точкой в ​​коде на равенство. Я мог бы, вероятно, сойти с меньшим или большим чем, но даже тогда это будет немного отличаться от того, что я ожидаю. Почему еще эта неточность может вызывать беспокойство? Когда вы не хотите, чтобы ваш компьютер был неточным? Какие домены? Какие миры вне классной комнаты? Да уж? Что это такое? АУДИТОРИЯ: Ракета. ДЭВИД: Да, так что ракеты, или что-нибудь, связанное с математикой, физикой и опасностью. Вы не хотите, чтобы числа были слишком незначительными. И если вы думаете об этом, ракеты - хороший пример, потому что я мало что знаю о ракетах, но я знаю, что они идут довольно быстро, и, вероятно, в этом есть углы, потому что вы пытаетесь удержать их на траектории. И это нормально. Но если ваша траектория немного отклоняется и что-то идет очень быстро и очень далеко, я уверен, что в конечном итоге эти небольшие неточности начнут складываться. И действительно, были исторические случаи, когда подобная неточность действительно накапливалась в сфере милитаристских операций или финансовых операций. На самом деле, если вы когда-нибудь видели Office Space или еще когда-то, когда играли в Superman 3, то некоторые люди зарабатывали деньги, потому что просто хранили все доли копейки, которые компьютерные системы просто игнорировали. И со временем они начинают складываться. Короче говоря, в любое время, когда у вас есть научные или финансовые или какие-либо большие наборы данных, которые включают в себя большие числа, их много и много времени, это проблема. И это почти говорит о том, что вы не должны относиться к компьютерам, не говоря уже о компьютерах, если мы не решим эту проблему. Теперь, как спойлер, люди решили эту проблему, и вы можете использовать все больше и больше битов, но не бесконечно много битов. В какой-то момент вы должны нарисовать линию, но именно поэтому, например, фондовая биржа может представлять только две десятичные точки точности для долларов или, может быть, четыре десятичных знака для тысячных долей для долларов и центов. И они просто должны решить, что это та точность, которую мы можем точно хранить. Но вы должны решить, как справиться с этим, а не просто игнорировать проблему. Но мы можем сделать немного лучше. Знаешь что? Оказывается, что в большинстве компьютеров число с плавающей запятой занимает, да, ограниченное количество места, но очень точно 32 бита или 4 байта. Напомним, что байт составляет 8 бит. Таким образом, 4 байта - это 32 бита, и это просто очень распространенная единица измерения. Но есть еще один. Оказывается, если вам нужно вдвое больше битов, вы можете буквально использовать тип данных, называемый double. А в библиотеке CS50 есть функция get double. И если я пойду дальше и сделаю это здесь, я теперь могу перекомпилировать этот код, сделать float, даже если они больше не являются технически плавающими типами. И позвольте мне идти вперед и делать ввод точки с косой чертой, и позвольте мне ввести 2 и 10. И теперь это все еще неточно. Но обратите внимание, вместо семи нулей, которые, я думаю, у меня было раньше, теперь у меня есть 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15 нулей , Так что я' Мы как бы выдвинули проблему, так сказать, чтобы она была более точной. Но это не идеально. Вы не можете получить идеальные значения, если хотите иметь возможность представлять бесконечный диапазон. Есть вопросы по этому поводу? И мы вернемся к некоторым последствиям. Да, здесь АУДИТОРИЯ: Так будет ли лучше всегда использовать удвоения, потому что это более точно? ДЭВИД: Хороший вопрос. Всегда лучше использовать двойные, потому что они более точные? Короткий ответ, да, но мы собираемся увидеть тематически в этом курсе, а в информатике в целом всегда есть компромисс. И да, если вы используете двойной, вы избежите этой проблемы немного больше, но какую цену вы заплатите, так сказать? Может быть, вычислительная мощность, потому что он должен иметь дело с большим количеством битов одновременно, и даже более буквально, с большим количеством места. Я имею в виду, Конечно, мы можем принять ваше решение, но я собираюсь взять с вас вдвое больше битов. И назад, в то время, десятилетия назад, когда C впервые был изобретен, и компьютеры действительно вступали в игру, потратив вдвое больше битов, даже если это всего лишь 32 крошечных нуля и единицы, это было на самом деле дорого в финансовом отношении, и это добавляет вверх. И даже сегодня, если вы хотите потратить больше места, это нормально, но вы собираетесь потратить. Больше места, а значит больше денег и, следовательно, меньше места для других вещей. Так что это просто компромисс. И вы, как разумный человек, должны решить, где правильная точка перегиба находится между тем, что важнее. Позвольте мне открыть совершенно другой пример, который теперь называется паритетом, просто в качестве примера. И позвольте мне спросить это, откуда вы знаете, является ли число четным или нечетным? В чем подвох? Да уж, если он делится на 2--, поэтому 0 и 2 и 4 и 6 даже потому, что если вы разделите на 2, вы не получите остаток. Так что, на самом деле, если вы просто хотите увидеть быстрый пример того, почему вы можете использовать остаток, даже если он находится вне контекста. Вот пример для паритета. Четность - это просто термин, относящийся к четному или нечетному в этом контексте. Как мы можем использовать это? Ну, заметьте, я могу получить int от пользователя вверху. Затем я могу проверить четность целого числа - четного или нечетного - с помощью синтаксиса, подобного этому. Сейчас я объединяю некоторые из этих операторов, поскольку вы можете быть склонны интуитивно. Если n, число, набранное пользователем, mod 2 или разделенное на, а затем проверить остаток от, но это полный набор равен 0, вы просто сказали, что это четное число - так что печатайте четное, иначе печатайте нечетное. Потому что что' Возможен ли другой остаток? Если вы введете n или любое значение, разделенное на 2, вы можете получить остаток от 0 или 1. Я должен проверить только один из них, 0, потому что остальное подразумевает другое. Итак, еще раз, очень простой пример. Но, честно говоря, у всех нас, вероятно, есть интуитивное понимание того, что является четным и нечетным. Этому нужно научить компьютер, и поэтому есть программа, которая делает именно это. Позвольте мне открыть более крупную программу, называемую условиями, и вместо того, чтобы печатать эту, потому что это несколько строк кода, позвольте мне просто открыть ее, потому что она завершает именно тот код, который мы видели несколько раз назад на слайде, когда мы сравнивали это к подобному блоку C. В этой программе, в моей основной функции - позвольте мне сосредоточиться на первых нескольких строках - у меня есть int с именем x, через который я получаю get int, другой int с именем y, который я ' я получаю через get int, а потом я просто делаю несколько простых сравнений. Мы увидели это снова, когда сравнили его с Scratch, но это просто тот же самый код в контексте, а не просто статичный просмотр на слайде. Итак, позвольте мне пойти дальше и скомпилировать это - сделать условия. Вроде бы скомпилировать. Позвольте мне увеличить масштаб и войти в условия косой черты. х будет, скажем, 2 снова, у будет 10. х действительно меньше, чем у. Если я запустлю его снова - и я смогу сэкономить время, просматривая мою историю и просто нажимая Enter. Давайте сделаем 2 и 2, и действительно, x равен y и так далее. Опять же, точно такой же код, что и раньше, но теперь в контексте работающей программы. Что, если бы я действительно хотел, чтобы пользовательский ввод был похож на то, что делал наш прошлый студент, получая ответы «да» или «нет» на вопрос об яблоке и кексах? Хорошо в ответе точка C, получается, что вы действительно можете получить текстовый ввод от пользователя, возможно, просто да или нет или просто y или n для краткости. И в этом случае, если я просто верну один символ, оказывается, что для символа есть отдельный тип данных. Если вам не нужна целая строка, например, целое слово, абзац, предложение или что-то еще, вам нужен только один символ. Вы можете использовать то, что называется символом или персонажем. И вот здесь я использую еще одну функцию с именем get char. Я храню его в переменной с именем C, потому что она хороша и лаконична типа char, и затем замечаю это. Возможно, вы не видели этот синтаксис раньше, особенно если вы никогда не программировали раньше, даже на пустом месте. Но вы, возможно, видели этот блок, похожий на этот в Scratch. Что, вероятно, подразумевает вертикальная двойная полоса? Или же, вот и все. Так что в Scratch это приятно и удобно для пользователя. Они буквально просто говорят ИЛИ. В программировании часто используется просто больше загадочных последовательностей символов и двух вертикальных полос, которые обычно находятся над клавишей Enter на клавиатуре США. Если C равно равно капиталу y или C равно строчно y - давайте предположим, что пользователь хотел сказать да, и идти вперед и сказать да. Иначе, если пользователь набрал n прописными или строчными буквами, давайте предположим, что он имел в виду «нет» и соответственно сказал «нет». И что мы будем делать иначе? Видимо, ничего, и это нормально. Вам не нужно иметь регистр по умолчанию, если вы хотите, чтобы программа ничего не делала. Это может немного сбивать с толку, потому что пользователь может ввести какое-то случайное слово и не получить никакого вывода, но это дизайнерское решение. Логически это как раз то, как мы можем выразить это. А как на самом деле строить наши собственные блоки? Есть вопросы, прежде чем мы начнем создавать? Так что вспомните, что в Scratch у нас был пример с кашлем. Позвольте мне пойти дальше и быстро создать файл, который называется "кашель, ноль, точка C" и просто воссоздать то, что мы делали на прошлой неделе. Включите стандартную точку ввода-вывода в основной пустоте - опять же, только мышечную память - и затем в кавычках printf снимите кавычки с обратным слешем "кашель" и точку с запятой. А ты просто мышечная память сейчас - и затем printf цитирует кавычки "кашель" обратная косая черта n точка с запятой. А ты просто мышечная память сейчас - и затем printf цитирует кавычки "кашель" обратная косая черта n точка с запятой. И йоты знаешь что? Позвольте мне идти вперед и кашлять не один, а два раза, три раза. В тот момент, когда вы начинаете копировать и вставлять, вы, вероятно, не пишете хороший код. Это не очень ремонтопригодно. Теперь, если я хочу изменить слово или перевести его на другой язык, я должен изменить его в трех местах. На прошлой неделе мы уже решили, что это плохо. Так что было бы лучше в C или Scratch или вообще чем этот подход? Да, так что для цикла. Итак, позвольте мне сделать это, позвольте мне создать еще один файл. Я собираюсь назвать этот кашель одна точка C, это моя вторая версия. Позвольте мне сейчас пойти дальше и просто скопировать и вставить оригинальный код, и давайте просто улучшим его. Давайте избавимся от двух из них, и давайте посмотрим, не можем ли мы выразить 4. Итак, это было 4 - позвольте мне увеличить - int, я получаю 0. Я меньше некоторого числа. Раньше было 50. Теперь у меня будет 3. я плюс плюс фигурные скобки, а теперь позвольте мне переместить блок кашля внутрь и сделать отступ, чтобы он был красивым. И обратите внимание стилистически, я делал это инстинктивно в течение некоторого времени. Все хорошо выделено только для того, чтобы сделать его более читабельным, совсем как блоки Scratch, хотя опять-таки много пустого пространства не имеет значения для компьютера. Так что, если я пойду вперед и запущу это ... позвольте мне открыть окно терминала, чтобы я мог его увидеть. Сделайте кашель один, введите - выглядит хорошо - точечный удар кашель один кашель кашель. Это хорошо, но помните, что мы фактически улучшили этот дизайн, абстрагировав его. Позвольте мне пойти дальше и сделать свою собственную функцию сейчас. Позвольте мне пойти дальше и открыть новый файл, кашель с 2 точками C, точно так же, как у меня была другая скретч-программа. Снова, включите стандартную точку ввода-вывода в основной пустоте, а потом здесь позвольте мне пойти дальше и что делать? Ну, для int я получаю 0, я меньше 3, я плюс плюс плюс фигурные скобки. Тогда позвольте мне пойти дальше и просто вызвать кашель. Было бы хорошо, если бы кашель существовал, но, к сожалению, кашля не существует. Его нет даже в библиотеке CS50. Так что это не поможет нам. Я должен сделать свою собственную функцию. Таким образом, в Scratch вы пошли к блоку, и вы делаете свой собственный блок, и появляется большая подсказка, и вы делаете свою новую часть головоломки. Здесь мы должны быть немного более взвешенными, и получается, что вы можете сделать это следующим образом. Поначалу некоторые из этих деталей будут неочевидны, но я собираюсь продолжить и назову функцию кашель. И кашля на данный момент не нужно принимать никакие входные данные. Таким образом, ключевое слово там пустота. И мы на самом деле видели это раньше. Главный также не принимал никаких вкладов. Вот почему у нас было слово void, но об этом в другой раз. И кашель тоже не вернет ничего. Он будет печатать на экране, как Брайан. Но Брайан, напомню, не вернул мне ничего физически. Так что нет возвращаемого значения. Так что я собираюсь сказать void слева от кашля. Таким образом, для сегодняшних целей это просто означает, что кашель не принимает ввод и не возвращает значение в качестве вывода. Вот и все, пустота, пустота. Теперь, так как тело этой функции, так сказать, я просто собираюсь пойти дальше и сказать, просто кашлять обратная косая черта n точка с запятой. Вот и все. Так что теперь у меня есть кусок головоломки, если хотите, чья цель в жизни - кашлять, а это значит, что теперь я могу волшебным образом назвать его по его имени здесь столько раз, сколько захочу. Итак, давайте продолжим и скомпилируем это. Я' Я действительно в хорошем рулоне. Пока все работает отлично - кашляю 2, вводи, красные ошибки. Так что это интересно, и этот вид напоминает мне о предыдущей ошибке. Итак, во-первых, на какой строке моя ошибка? Семь и девятый символ, если вам все равно, но это семь, в седьмой строке - неявное объявление функции кашля недопустимо в C99. C99 имеет в виду буквально 1999 год, когда была изобретена эта версия C, и поэтому неявное объявление функции кашляло. Но это прямо здесь. Хорошо, подожди минутку. Инстинкт, позволь мне переместить это. Позвольте мне поставить это на первое место. Давай посмотрим что происходит. Сделай кашель 2 - Боже мой. Это исправило это. Зачем? Даже если вы никогда не программировали раньше, интуитивно объясните, почему это что-то решило. АУДИТОРИЯ: Вы держите функцию, которую вы объявили ранее. Так что, хотя вы делаете это недействительным, у вас по крайней мере есть [неразборчиво] .. ДЭВИД: Точно. Потому что раньше я пытался использовать кашель рано в строке 7, но я только учил компьютер, что кашель был дальше в файле. Честно говоря, C вроде тупой. Он буквально читает ваш код сверху вниз слева или справа. И если вы попытаетесь что-то сделать до того, как научите компьютер, как это делать, вы получите такой необъявленный идентификатор, потому что он просто еще не знает, что это за слово. Теперь в Scratch это не имеет большого значения. Вы просто перемещаете кусочки головоломки куда хотите. Порядок блоков физически на экране не имеет значения. Но в С это так. Но, честно говоря, это немного раздражает, так как теперь основная программа здесь все больше и больше отталкивается от сложности, которую я хочу добавить в свою программу. Так что есть другое решение. Позвольте мне на самом деле пойти дальше и положить это обратно, где это, потому что я немного конкретен. Мне просто нравится, что главное - быть на вершине, и, честно говоря, это хороший стиль. Если main - это самая важная функция в вашей программе по умолчанию - потому что это главная функция и это то, что по умолчанию вызывается компьютером по умолчанию - почему я собираюсь нажать ее до конца, чтобы обойти это? глупая деталь? Ну, мне просто нужно научить компьютер, какова функция, и я могу сделать это немного излишне, просто сказав это. Это то, что мы будем называть прототипом для функции. Если вы буквально просто скопируете самую первую строку, которая имеет свое имя, входные данные, если они есть, и выходные данные, если они есть, это точка с запятой. Это буквально копировать вставить из самой функции, но теперь этого достаточно, чтобы сказать: эй, компьютер, эта оболочка существует. Этого достаточно для того, чтобы вы могли позвонить ему, потому что компьютер, пока я видел имя функции раньше, будет в порядке, если нули и, так сказать, реализации, появятся немного позже. И это более традиционный способ решения этой проблемы. Так что, просто интуитивно, догадайтесь, если это не слишком большой косвенный скачок, что в стандартном вводе-выводе h, что в CS50 точка h. Такого рода мы будем называть их декларациями. Таким образом, буквально в стандартном IO точка h представляет собой строку кода, которая обучает компьютер тому, что представляет собой ввод printf и каков вывод printf, если он есть. В CS50 точка h буквально есть строка кода, которая сообщает компьютеру, что такое ввод строк и каков их вывод. И то же самое для get int, get float, char и прочего - это все, что есть в этих заголовочных файлах. Нули и единицы, так сказать, на самом деле находятся в файлах, которые буквально называются стандартными точками ввода-вывода C и CS50, хотя это технически исходный код. Нули и единицы находятся в скомпилированном файле в другом месте системы. Но все эти вещи мы воспринимали как должное, и теперь, надеюсь, это имеет немного больше смысла, потому что тот факт, что я делаю резкое включение сверху, это всего лишь решение проблемы. В этом файле достаточно информации, чтобы научить компьютер, что такое printf или что такое строка, и тому подобное, так что мне не нужно беспокоиться о перемещении вещей вокруг себя или копировании и вставке того, кто написал printf, его или ее код, в мою программу. Теперь давайте Сделайте один последний пример с кашлем, продолжайте, и назовите его так называемый кашель, 3 точки C, и вставьте мой тот же код, что и минуту назад, просто чтобы начать. И помните, что последний шаг нашего примера с кашлем на прошлой неделе был на самом деле дать кашель вклад. Я бы хотел свести этот код к буквально кашляющему 3, потому что это действительно хорошая абстракция. Я не хочу, чтобы главный думал о том, сколько раз - просто кашляй три раза. Это хорошая, полезная человеческая абстракция. Теперь давайте разместим функциональность здесь. Так что, если я хочу, чтобы кашель мог выдержать ввод, такой как 3, какие упоминания о пустоте предположительно нужно изменить? Тот, что слева или справа? Право, тот, что в скобках. И получается, так же, как вы можете объявить переменные внутри функции, как мы сделали, так что вы можете объявить аргументы для функции, как это. Таким образом, вы можете называть это как угодно, хотя тип данных имеет значение, но сейчас говорится, эй, компьютер, кашель не возвращает значение, как Эрин действительно вернула листок бумаги. Эй, компьютер, кашель требует одного входа. Это целое число и просто назовите его n. И теперь, когда вы это сделали, теперь у вас может быть строка кода, подобная этой. Для int я получаю 0, я меньше, чем ni плюс плюс - и затем, ОК, здесь нажатием клавиш. Тогда я могу переместить это внутрь, сделать отступ красиво. И теперь обратите внимание, вся сложность кашля была преобразована в функцию, мою собственную часть головоломки, если хотите, которая даже принимает аргумент, так что теперь вы можете буквально, если я отодвину это далеко и с ума, теперь ваша программа становится довольно интересной, потому что она действительно делает то, что говорит. И это хорошая функциональная абстракция, если хотите, так что теперь у меня есть новый глагол, новое действие, новая функция под названием кашель. Есть вопросы по этому вопросу? Да уж. Извини, скажи еще раз. АУДИТОРИЯ: Какое целое число вернётся? ДЭВИД: Какое целое число вернётся? Оказывается, мы вернемся к этому. Он будет возвращать 0 почти всегда по умолчанию, но это оставляет вас с почти бесконечным числом ненулевых значений, которые представляют все многие вещи, которые могут пойти не так. Так что об этом, когда мы начнем создавать больше ошибок. Давайте посмотрим друг на друга. Позвольте мне пойти дальше и заранее открыть этот файл, который называется «положительная точка С.». Предположим, я не доволен, чтобы просто получить доступ к int. Я хочу, чтобы функция называлась get positive int, потому что по какой-то причине моя программа, моя игра, все, что нужно, должны знать положительное значение. Может я' Я спрашиваю пользователя, сколько игроков там, и это не должно быть отрицательным. Это должно быть положительное целое число, например, один, два или более. Так что получается, что я могу написать программу, если захочу, чтобы она выглядела следующим образом. Здесь в этой строке вызывается функция с именем get positive int, передает ее в приглашении, а затем сохраняет значение, по-прежнему в целочисленном виде, с левой стороны, а затем просто идет вперед и распечатывает его. Получите положительный int имеет этот прототип в верхней части файла. Обратите внимание, что это не функция, которая поставляется с библиотекой CS50s, CS50 точка h. Функция называется получить положительный int. Как вы надеетесь, он возвращает int и принимает строку в качестве подсказки, какие бы слова вы не хотели видеть человеком. Давайте прокрутим вниз сейчас, и этот выглядит немного более сложным, и это не та особенность, которой обладает Scratch. Но давайте посмотрим. Первая строка идентична прототипу, потому что я буквально скопировал и вставил его. Все, что находится между этими скобками, является самой функцией. И здесь, чтобы ответить на чей-то вопрос с самого начала, нужно ли объявлять переменную, а затем использовать ее сразу? Нет, и это на самом деле полезное решение проблемы, которую мы увидим через минуту. Обратите внимание, что это новое ключевое слово - раньше его не видели - выполните следующее, пока n меньше 1. Ранее мы видели цикл while и цикл for. Мы не видели цикл do while. И цикл do while, хотя он звучит явно похоже на цикл while, что может показаться немного другим? Когда этот блок навсегда был переведен ранее, в то время как true, каков был порядок операций? Мы проверили состояние, правда, а потом распечатать привет мир? Или мы просто распечатать привет мир, а затем проверить состояние? Да, вы можете не вспомнить точно, но я на самом деле ... Я проверил, правда ли это, и мы все сказали да, очевидно. Printf-- это правда? Printf-- это правда? Printf-- так что сначала проверил условие. Тогда вы можете сделать вывод, что этот цикл немного отличается. У него есть другое слово, делай. Это буквально будет делать это сначала, а затем проверять условие и делать это снова, только если условие истинно. Так что это хороший способ просто перевернуть вещи с точки зрения порядка, чтобы сделать что-то хотя бы один раз, а не вообще никогда, как это было раньше. Так что же мы делаем? Получите int, передавая это приглашение, сохраните его в n. И если пользователь вводит значение, которое меньше 1, будет ли это истина или ложь, если n меньше 1? Так что, если человек вводит 0, 0 меньше 1? Верно, да, так что, что происходит, вы возвращаетесь к делу и делаете это снова. Если пользователь вводит отрицательный 1, отрицательный 1 меньше 1? Да или правда. Таким образом, вы делаете это снова. Если он или она вводит отрицательный 2, снова, снова. Что если он наберет 50? Ну, 50 не меньше, чем один. Так что это неверно. И тогда вы переходите к следующей строке кода в целом. Но что интересно в следующей строке кода, так это то, что в отличие от примера cough, у которого в качестве возвращаемого типа использовался void, по умолчанию он принимает положительное значение int, он должен возвращать int, как и Эрин передал мне листок бумаги с строка на нем. И вот здесь, если я хочу, чтобы моя собственная пользовательская функция с именем get Positive int возвращала значение, в C. есть еще одно слово Вы буквально пишете return, а затем имя переменной или значение, которое вы хотите передать на метафорическом листе бумаги любому коду, использующему это. Так что это за странность? Почему я не могу этого сделать? Если бы я подражал коду, который мы написали ранее, таким образом, почему эта строка кода не работает просто логически, используя некоторые ментальные модели, которые мы имели до сих пор? АУДИТОРИЯ: Объявление кода снова. ДЭВИД: Скажи еще раз. АУДИТОРИЯ: Объявлять снова и снова. ДЭВИД: Да, так что объявлять просто означает создавать, это модный способ программирования создания слов. Итак, это говорит: «Эй, компьютер, дай мне целое число, назови его n и установи его равным возвращаемому значению get int. Поэтому, что бы ни передала мне функция или Эрин, положи сюда. Но проблема в том, что в C переменные имеют область видимости. Сфера - это причудливый способ сказать, что они существуют только между фигурными скобками, между которыми они были объявлены. Таким образом, это означает, что эта переменная n буквально существует только между здесь и здесь, а затем она просто исчезает. Компьютер больше об этом не знает. Но это проблема, потому что, по какому номеру строки мы должны знать n? Похоже, 21, и это за пределами фигурных скобок. Так что, исходя из этого базового определения, область действия - это две фигурные скобки, между которыми объявлена ​​переменная. Это не существует вне их. Этот код просто не будет работать. И я исправлю это позже, чтобы вы увидели правильное сообщение об ошибке. Почему это не работает? Ну, вы объявляете внутри этих фигурных скобок. Так как же этого избежать? Что ж, получается, как кто-то утверждал ранее, просто объявите это сами, даже не придав этому значения. И действительно, синтаксис для этого просто сделать половину мысли - в точке с запятой. Это не имеет значения, которое мы знаем еще. Это имеет мусорное значение, но об этом в другой раз. Но это сейчас существует. А теперь обратите внимание, какие фигурные скобки существуют внутри - эта и эта, что означает, что теперь она доступна везде. И если вы в своих Scratch-программах действительно использовали переменные, вы могли заметить, что вам пришлось выбирать. Вы должны были принять решение для этого спрайта или для всех спрайтов. Это был намек на то, что называется в программировании локальной или глобальной переменной. Они все еще местные, и мы еще вернемся к этому термину. Но это связано с областью действия, потому что, если бы вы указали только для этого спрайта, MIT позволил бы вам использовать эту переменную только для этого конкретного спрайта, тот кот или овца или что-то еще, что ты программировал. Как и в C, теперь это означает, что n можно использовать здесь и здесь, но не где-либо еще, как выше в моей программе. Это вопрос сферы. Итак, давайте теперь посмотрим, что может пойти не так. Позвольте мне пойти дальше и открыть это, потому что оказывается, что при программировании могут возникать другие проблемы, а не только точность с плавающей запятой, как я описал это ранее. Оказывается, есть и другие проблемы, которые могут пойти не так, даже с целыми числами, которых мы вроде как вообще избежали. Итак, напомним, что мы начали говорить о 1, 2 и 3 и почему это 123 на прошлой неделе. Ну, а что будет в десятичной системе, если вы добавите 1 к 123? Какой номер вы получаете? Очевидно, 124. Если мы сделаем это снова, 125, 26, 27, 28, 29. Что произойдет в десятичной системе, если я добавлю 1 к 9? Ну, я должен получить 10, но это Не так, как мы это напишем. Вместо этого вы ставите 0 и несете 1. Помните эти умственные эвристики? Так что это все, что мы сделали там. И затем это 2 плюс 1. Так вот почему 129 плюс 1 - это 130, потому что вы ставите 0, вы несете 1 и так далее. Так что теперь мы все делаем это интуитивно. Но это влияет и на то, что делают компьютеры, потому что предположим, что мы рассматриваем большее число, например 999. И что вы получите, добавив 1 к 999? Ну, вы несете 1, вы несете 1, и вы получите, надеюсь, 1000. Но что, если на вашем компьютере есть место только для трех цифр? Или что если ваш телефон или что, если ваш будильник или другое устройство буквально имеет место только для 3 цифр, что такое 999 плюс 1, если ваше оборудование имеет только три цифры? Ну, это, по-видимому, 0. Итак, вы получаете это переполнение 998, 999, 0. Это, так сказать, переполнение. Концептуально один вид падает, и вы переходите к следующему значению, которое равно 000. Так что же в двоичном? Что это за двоичное число, если вы перевели десятичную дробь в своей голове? И запомните, это столбец одни, столбцы двойки и столбец четверок. Так что это было 7 в двоичном коде - 1, так что это 4 плюс 2 плюс 1, так что 7 в двоичном. Так как вы делаете арифметику с двоичным? Это на самом деле то же самое. Просто у вас нет двойки, тройки, девятки или чего-то промежуточного. У вас просто есть единицы и нули. Итак, что вы получаете, когда вы добавляете 1 к 111? Ну, это та же идея. Вы кладете 0 и несете 1, потому что 1 плюс 1 вы хотите сказать 2 в десятичной дроби, но нет 2. Таким образом, он переносится на 0. Но вы несете 1. 1 плюс 1 равно 2, но, ОК , это 0, нести 1. 1 плюс 1, это 2. Но у меня нет 2. Итак, я возвращаюсь к 0, несу 1. Итак, в двоичном формате, если у вас есть только три байта или бита, если у вас есть только три бита, что вы получите, когда вы добавите от 1 до 7 в двоичном виде? Вы очевидно получаете 0. И теперь это становится более реальным. В моем компьютере, в моем телефоне, во всех ваших аппаратных средствах это просто конечный объем памяти, ОЗУ, тот маленький чип, который я показал на экране со всеми маленькими микросхемами. И это, конечно, имеет более 3 бит памяти, но это конечно. И если мы обычно используем 32 бита для представления вещей или 64 бита, возможно, если мы используем double или что-то еще, называемое long - long - это 64-битное целое, тогда как integer обычно 32 бита Кажется, что в какой-то момент числа могут переполниться, и у нас действительно будет некоторая, так сказать, неточность, т. Е. Целочисленное переполнение. Таким образом, вы можете увидеть это или защититься от этого в реальном мире. Так что это скриншот из игры, распространенной на нескольких разных платформах, и это игра, которая позволяет реально накапливать монеты или очки, или маленькие кусочки Lego. И если вы наберете эти очки, вы заметите, что в конечном итоге, если у вас будет слишком много свободного времени, вы сможете набрать в этой игре только такой высокий балл. Каков, по-видимому, самый высокий балл, согласно скриншоту с того, кто взял это после игры в течение слишком многих часов? Это был не я. 4 миллиона - нет, 4 миллиарда, 4 миллиарда. Это почему? Что ж, получается, что если числа и компьютеры, как я предлагал, обычно хранятся с использованием 32 бит. Такого рода возникает вопрос, а как высоко вы можете считать с 32 битами? Ну, 32 бита означает, что у вас есть 32 нуля и единицы. Самые большие они могут быть как 11111, 32 из них. И если вы на самом деле делаете математику, используя наши маленькие колонки и так далее, это примерно 4 миллиарда, чуть больше 4 миллиардов. Так что авторы этой игры Lego просто решили, вы знаете, что, скажем так, максимальное количество очков или монет, которые вы можете накопить в этой игре Lego, составляет даже 4 миллиарда. Зачем? Это выглядит даже чище, чем на самом деле. Но почему? Сколько битов они используют для сохранения вашего счета в этой игре? 32 бита или 4 байта, и это просто соглашение. На каком бы языке они не программировали эту игру, вероятно, имеет тип данных, называемый int, и этот int по соглашению использует 32 бита. Поэтому в какой-то момент они должны были решить, что мы можем использовать больше памяти, как вы предлагали ранее для удвоений. Давайте использовать 64 бит. Тогда у вас может быть сумасшедшее количество часов, играя в игру и получая все больше и больше очков. Или мы можем просто сказать, что достаточно очков, чтобы накапливаться в игре. Вот когда вы действительно ожидаете этого. Это не всегда происходит на самом деле. Если мы пойдем дальше и посмотрим на пример кода - позвольте мне продолжить и открыть точку переполнения C. В этой программе вы увидите строку 8 с синтаксисом косой черты, и у меня было несколько таких до сих пор. Но я на самом деле не упомянул их. Оказывается, в C, просто в Scratch - скорее всего, вы не заметили эту маленькую особенность Scratch - вы можете получить так называемые комментарии. Комментарий - это просто записка для вас, вашего TF, вашего друга, вашего коллеги, с кем бы вы ни писали код. И это просто примечание для себя, чтобы напомнить себе о том, что делает код. Без этой строки, я мог бы однажды, я достаточно комфортно программировать, понять, что делают эти строки кода. Но, честно говоря, я напрасно написал код. И если я смотрю на это недели или месяцы спустя или кто-то другой написал это, просто скажите мне, что он делает. Таким образом, комментарий в программе - это просто краткое изложение нескольких строк кода или резюме на английском языке или на любом другом разговорном языке, который описывает, что в действительности могло бы делать загадочный код. Так что вам не нужно слишком много думать об этом, чтобы понять программу. Так итеративно удваивается, итеративно означает циклично, снова, снова и снова. Это в стиле фанк. Мы не видели этого раньше, но вы можете догадаться, что это делает. Что делает звезда, равная? Это удваивает. Это как плюс равно добавляет 1, звезда равно двойному, если значение справа равно 2. Итак, начнется печать 1, затем 2, 4 и 8 и так далее. И обратите внимание на эту функцию. Это называется сном. Он буквально засыпает на секунду, и у этой функции сна есть прототип, который кто-то другой записал в файл с именем uni standard dot h. Я только знаю это, ища это в документации. Но это новый файл только для сна. Сделайте переполнение, которое здесь подходит. И позвольте мне пойти дальше и сделать окно терминала еще больше для этого - переполнение косой черты. Хорошо, идет Это будет. Это будет происходить все быстрее и быстрее, так сказать, потому что мы добавляем все больше и больше с удвоением. 2000, 4000, 8000, 16000 - это все еще идет - 64 000, 65 000. Теперь мы в миллионах - 2 миллиона, 4, 8, 16 миллионов. Это' Все больше и больше становится все больше. Ох, интересно. Что сейчас произошло? Таким образом, получается, что если вы удвоите достаточно большие числа, вы в конечном итоге получите 0, что также, вероятно, вас не учили. Так что же на самом деле произошло? Control-C, мы отменим это. Что случилось? Я имею в виду, что программа пытается сказать мне, хотя это немного загадочно, целочисленное переполнение со знаком. Подписано просто означает, что оно по существу перешло от положительного к отрицательному. Так что же случилось? Что это такое? Да, все кончилось. Я удваиваю число снова и снова и снова. И в какой-то момент мы перенесли, так сказать, одну, и это была 33-я единица, следовательно, за границы 32-битного значения, и она просто перешла в явно отрицательное значение, потому что в какой-то момент - и у нас нет Я говорил об этом - вы можете использовать как крайний левый бит, чтобы сказать положительный или отрицательный. Мы только что говорили о положительном. И тогда, честно говоря, компьютер просто сдался, не зная, что вы намерены делать дальше. Так что, если вы не напишите код, чтобы справиться с этой ситуацией, и убедитесь, что ваши цифры меньше 4 миллиардов, прежде чем вы перевернетесь, будут только ошибки. И это может показаться надуманным, но это произошло не так давно. Итак, 1999 год был незадолго до того, как многие думали, что конец света наступит из-за так называемой ошибки Y2K, и на самом деле это была не столько ошибка, сколько недостаток осознанности или недостаток возможностей. В чем проблема Y2K в двух словах? Кто-то хочет предложить? Даже в нетехническом смысле, да. АУДИТОРИЯ: Компьютеры не могли отобразить число 2000. ДЭВИД: Да, Итак, позвольте мне подвести итог здесь. Поэтому, если они используют только две цифры для отображения значений, вы могли бы перепутать 2000 год с фактически 1900 годом, потому что, короче говоря, то, что люди делали довольно разумно десятилетия назад, было - пространство было дорогим. Компьютеры были дорогими. Памяти было не так много, как сейчас со всем облачным хранилищем и тому подобным. Так ты знаешь что? Если это было похоже на 1970 год, действительно ли мы заботимся о 1969, 1968, не говоря уже о 1900? Не совсем, давайте просто предположим, что мы все в 1900-х годах и никогда не показываем и не храним один 9. Давайте просто сохраним две цифры на каждый год. Итак, 70 - это 70, 99 - это 1999. Но проблема в том, что люди закончили тем, что запускали код, который они написали много лет назад, десятилетия назад, намного дольше, чем люди думали, почему. Ну, это дорого, это занимает много времени, чтобы изменить код, код работает. Зачем пытаться это сломать? Проблема в том, что, поскольку люди в возрасте и скончались, все меньше людей, которые даже знали языки, на которых были написаны эти программы. И теперь кто же собирается обновить программное обеспечение? Страшно было столкнуться с множеством проблем, и это действительно сводится к тому, что 1999 год мог бы переполниться не до нуля как такового, а как неявного 1900 года. И действительно, это определенно произошло, хотя и не в том масштабе, как думали люди. Но это происходит даже в более реальных условиях всего несколько лет назад. Это Боинг 787, настоящий самолет, который нужно было заземлить в течение некоторого времени, потому что в нем была ошибка программирования. И его кратко изложено здесь в онлайн-статье. Оказалось, что самолет 787, который работал непрерывно в течение 248 дней, был предупреждением, может потерять всю свою электроэнергию из-за перехода генератора в безопасный режим. Это почему? Это условие вызвано внутренним по отношению к генератору программным счетчиком, который переполняется через 248 дней непрерывной работы. Так что переведите это. Это просто означает, что программное обеспечение работает на Boeing 787. Они использовали 32-битные целые числа. Они использовали эти целые числа для хранения сотых секунд. И в какой-то момент, если вы оставите свой самолет включенным на 248 дней, каждый из которых имеет 24 часа, 60 минут в час, 60 секунд в минуту и ​​100 десятых секунды или 100 сотых секунды в секунду, этот продукт умножения вещей становится большим довольно быстро. И в день 249, Теоретически самолеты отключались бы даже в середине полета по вполне реальным причинам, потому что действительно большое число опрокидывалось до нуля, и это приводило в замешательство генератор. И это настоящие инженеры «умных» самолетов, совершающие подобные ошибки из-за программного обеспечения, не ожидая ни одной строки кода, ни некоторого количества строк кода. Или, как вы предложили, почему они просто не использовали больше битов? И опять же, это очень реальные проблемы. Так что это, к счастью, было решено и решено, но не раньше, конечно, там был настоящий страх. Оказывается, в старой игре это была игра Civilization. Оказывается, что один из персонажей, за которого вы можете играть, Ганди, на самом деле не такой мирный персонаж в игре, как вы могли бы подумать. И давайте для контекста просто посмотрим здесь на секунду. Если мы на самом деле посмотрим на еще несколько двоичных файлов, это в двоичном виде, какое число в десятичном? ОК, 1. И это 8-битный. Так что это полный байт. 8 бит - это байт - 1, 2, 3, 4, 5, 6, 7, 8. Итак, что вы получите, если вы сделаете 1 минус - ну, если вы вычтете 1 из этого, вы, очевидно, получите что? Все нули. Итак, 1 минус 1 - это просто 0. Что если вы вычтете 2 из этого значения? Что просходит? Это на самом деле называется целочисленным переполнением, что как раз наоборот, но на самом деле вариантов не так много, чтобы думать об этом. Если у вас есть только нули и единицы, вы, вероятно, можете представить себе, что это за плохой сценарий. Если 0000001, если вы вычесть 1, уходит в нули, а затем вы делаете это снова, то теперь у вас недостаточно, что просто приводит вас к противоположному 11111111. Итак, если у вас есть 8 единиц, какое значение это имеет, если вы делаете математику? Одиночки, двойки, четверки, восемь. Оказывается это с 255, если вы действительно делаете всю математику. Вот и получается, что в этой игре Civilization использовался один байт для представления уровня агрессивности каждого персонажа в игре. И Ганди, как и следовало ожидать, по умолчанию инициализирован на 1, очень неагрессивный. К сожалению, в этой игре Цивилизация, когда игрок принимает демократию в своей цивилизации, его агрессия будет автоматически уменьшена на два. И поэтому, если Ганди станет демократическим, его агрессия не станет отрицательной 1. Она вернется к нелепо высокой цифре 255, что сделает его таким агрессивным, каким только может быть цивилизация. Конечно, менее эффектный, чем пример самолета, но эти проблемы вездесущи. И если вы начнете следить за популярными средствами массовой информации, или когда появятся ошибки, хаки или эксплойты, это ' Это так часто, потому что программист допустил ошибку в своем коде. Они не ожидали сценария или принимали, возможно, разумные решения несколько лет назад, но в конечном итоге это оказалось наивным, поскольку мы все еще работаем с одним и тем же кодом, числа становятся слишком большими, их математика неверна. И так происходят очень реальные вещи. Но самое главное для нас - просто понять, как и почему это происходит. И что мы будем делать в ближайшие дни? Поэтому следующим домашним заданием, как и в Scratch, будет программирование чего-либо, но на этот раз на C. Вы будете использовать среду под названием CS50 lab, которая, по сути, является песочницей CS50, с которой мы сегодня работали. Но это добавляет к нему инструкции и спецификации проблем, которые вы хотите решить. И он будет держать ваши руки первоначально через некоторые из этих шагов. Вам не нужно записывать все и запоминать все, что я напечатал сегодня, но не стесняйтесь зайти на сайт курса и поиграть с любым из этих примеров. Среди предстоящих задач будет воссоздание некоторых фрагментов игр прошлых лет, размышляя о том, как вещи, которые вы, возможно, видели, выросли, могут быть преобразованы в реальный код. И, несомненно, среди первых вещей, которые вы испытаете, это разочарование. Вы забудете глупую точку с запятой или куда уходят скобки? И вам придется оглянуться на код. Но имейте в виду, ничего из этого не имеет значения. Поначалу это абсолютно разочаровывает, но самое главное - это идеи и, если честно, чувство удовлетворения, которое вы, как и все сотрудники CS50 до вас, в конечном итоге почувствовать, когда на самом деле строит и создает что-то свое. Давайте назовем это там днем, и увидимся в следующий раз.

ДУГ ЛЛОЙД: ОК. Итак, давайте поговорим о том, как использовать командную строку Linux. Теперь IDE CS50, или даже устройство CS50, если вам это знакомо, или вы используете более старую версию CS50, представляет собой облачную машину с Ubuntu, которая является одной из многих. разновидности операционной системы Linux. Операторы Linux предпочитают операционную систему, потому что она круче, правда?

Многие современные дистрибутивы Linux имеют графический пользовательский интерфейс, который мы также называем GUI, GUI, чтобы обеспечить легкую навигацию на основе мыши, с которой вы, вероятно, знакомы, если вы пользователь Windows или Mac, перемещаете мышь, нажатие на иконки и так далее. Тем не менее, будучи программистом, и даже несмотря на то, что в среде IDE есть возможность выполнять какие-то графические пользовательские операции, щелкать, перетаскивать и все такое, вы все равно будете использовать окно терминала довольно часто. И вы можете выполнять многие из тех же задач, которые вы можете выполнять с помощью мыши с помощью клавиатурных команд. И мы немного поговорим о том, что представляют собой некоторые из этих команд прямо сейчас.

Теперь эти команды можно использовать в любой операционной системе на основе Unix, которая включает в себя Linux, но также и Mac OS. Если вы откроете терминал на своем Mac, вы можете использовать эти точные команды. В Windows также есть командная строка, но некоторые команды немного отличаются, поэтому на самом деле она не работает, поскольку Windows не является системой на основе Unix.

Итак, давайте посмотрим на некоторые из этих команд Linux. Первое, что вы, вероятно, будете использовать довольно часто, это ls. Это строчная l, за которой следует строчная s, что сокращенно для списка. И что команда list делает, это дает вам считывание всех файлов и папок в вашем текущем каталоге. Таким образом, вы можете увидеть все, что вы можете получить от того, где вы находитесь в данный момент.

Итак, я открыл здесь CSE IDE. И я собираюсь увеличить масштаб в секунду, чтобы вы ближе познакомились, но вот общая картина того, как выглядит IDE. Слева вы видите, что у нас есть дерево файлов, с которым вы, вероятно, знакомы, двойной щелчок, файлы и папки, и все такое. Так что это все еще есть в устройстве CS50. В центре сверху находится место, где вы собираетесь писать свой код, как только вы нажмете на файл. И внизу у нас есть окно терминала, где мы можем выполнить эти команды терминала.

Я собираюсь увеличить масштаб и зайти сюда, просто чтобы показать вам, что, на самом деле, я могу нажимать на эти файлы и папки. Ясно, что там, где я сейчас нахожусь, у меня есть две папки, называемые pset0 и pset1, и три файла, один с именем hello, один с именем hello.c и один с именем hello.txt.

Итак, давайте спустимся к окну терминала и познакомимся поближе. Итак, мы снова говорили о том, что у нас есть три файла и две папки в текущем каталоге. Если я наберу ls, что опять-таки является командой для отображения содержимого текущего каталога, а затем я нажму Enter, посмотрите, что я вижу, hello, hello.c, hello.txt, pset0 и pset1.

pset0 и pset1 окрашены в синий цвет, чтобы показать вам, что это каталоги, в которые мы могли бы перейти. И мы узнаем немного о том, как перемещаться по каталогам через минуту. И все остальное окрашивается в черный цвет, если это текстовый файл или файл исходного кода, и зеленый, если это исполняемый файл. Ясно, что это означает, что я могу запустить программу под названием «Привет». Вот что значит зеленый. Но в основном, ввод команды ls позволил мне посмотреть на все, что существует в моем текущем каталоге, что соответствует тому, что мы видим здесь, на графическом дисплее того же самого. Следующая команда, которую вы, вероятно, будете использовать совсем немного: cd, строчная c, строчная d, что означает сокращение каталога изменений. Это позволяет нам делать то, о чем я говорил секунду назад, то есть перемещаться между каталогами в командной строке, в отличие от двойного щелчка по папкам. Поэтому, если мы введем cd, а затем имя каталога, мы можем попасть в этот каталог.

Кроме того, знайте, что имя текущего каталога всегда точка, а имя каталога на один уровень выше того места, где мы сейчас находимся, то есть имя папки, в которой находится наша папка, точка, точка - или в что наша папка, это точка, точка. И если вам когда-нибудь интересно узнать имя вашего каталога, вы можете набрать pwd, что означает текущий рабочий каталог. Сейчас мы рассмотрим все это, возвращаясь к CS50 IDE. Так что я вернулся в свое рабочее место сейчас. И я снова увеличу масштаб терминала, чтобы мы могли взглянуть на перемещение внутри IDE. Так что я собираюсь перечислить содержимое моего каталога снова, просто чтобы пересмотреть нас в том, где мы находимся. Так что, если я наберу ls, который снова для списка, я вижу, что я могу добраться до pset0 и pset1. Это каталоги, в которые я могу попасть отсюда. Я знаю это,

Допустим, я хочу попасть в свой каталог pset1, потому что я работаю над набором задач 1. Я могу снова ввести cd--, сокращение от change directory-- space, pset1. И если я нажму Enter, обратите внимание, что происходит. Похоже, многое не произошло. Но если вы посмотрите на приглашение, теперь оно говорит мне, что я нахожусь в ~ / workspace / pset1. Я перешел в папку pset1, которая была в моей рабочей области.

И если я наберу ls, я вижу здесь что-то другое, верно? Это не тот список, который я видел раньше. Я вошел в pset1. И вот теперь, когда я набираю ls, я получаю контекст того, что я могу видеть из папки pset1.

Теперь я собираюсь набрать control l, который просто очищает экран. И я собираюсь перечислить содержимое каталога снова, чтобы вы могли видеть. Я просто хотел сделать это, чтобы очистить кое-что из того, что вы видели внизу, и не допустить, чтобы это зашло слишком далеко за пределы диапазона.

Я уже говорил ранее, что если я хочу перейти к текущему каталогу, я могу набрать cd space точка. Нажмите Enter. Это ничего не делает, верно? Я меняю каталоги на текущий каталог. Вы не всегда найдете необходимость в одной точке, но вы будете иногда.

Допустим, я хочу подняться на один уровень выше. Я хочу вернуться в мой каталог рабочей области. Я не могу набрать рабочую область CD, там нет такого файла или каталога. И причина этого - если я еще раз наберу ls - в том, что в моем каталоге pset1 нет каталога с именем workspace. Я собираюсь очистить свой экран снова с контролем л. Помните, что я сказал ранее, что мы можем вернуться назад с точкой, точкой. Это имя родительского каталога. Поэтому, если я наберу cd, space, dot, dot, а затем нажму Enter, теперь посмотрим, что произошло. Моя командная строка сообщает мне, что я вернулся в каталог ~ / workspace. Я поднялся на один уровень, благодаря точке, точке.

Теперь предположим, что я использую операционную систему на основе Linux, но она не обязательно говорит мне, где я. Этот случай говорит нам, что я сейчас нахожусь в ~ / workspace, прямо в приглашении. Но я мог быть полностью потерян в беспорядке папок, и я понятия не имею, где я нахожусь, и понятия не имею, куда я хочу вернуться. Есть две вещи, которые я могу сделать.

Прежде всего, я могу выяснить, где я, набрав pwd. Это мой настоящий рабочий каталог. И если я нажму Enter, он точно скажет, где я. Теперь / home / ubuntu - это длинный способ сказать, тильда, которая является вашим домашним каталогом. Но это говорит мне, что я нахожусь в домашнем / ubuntu / workspace или ~ / workspace.

Я снова собираюсь перейти в каталог pset1 и перечислю содержимое. И я вижу, что у меня есть еще один каталог, который называется extras. Итак, я собираюсь перейти на дополнительные материалы, а затем я собираюсь очистить свой экран.

Так что теперь я довольно далеко, верно? Что если я хочу немедленно вернуться в рабочее пространство? Есть пара вещей, которые я мог бы сделать. Я мог набрать cd, dot, dot, slash, dot, dot, чтобы подняться на один уровень, а затем на другой. Но это немного раздражает.

Так что, если я когда-нибудь захочу вернуться только к своему домашнему каталогу, тильда, я могу набрать cd без него. CD, Enter. А теперь я в тильде. И если я хочу попасть в рабочее пространство, я могу просто набрать cd workspace. И именно так вы работаете, меняя каталоги в IDE CS50 или любой операционной системе Linux в командной строке. Отлично. Следующий, который может пригодиться, это mkdir, что означает сокращение каталога, если мне нужно создать новую папку. Если вы знакомы с операционной системой GUI, вы можете щелкнуть правой кнопкой мыши. И затем, когда появится контекстное меню, выберите «Новая папка». Это, вероятно, как вы сделали это раньше. Но мы также можем создавать каталоги в командной строке. Итак, мы вернулись в IDE. Я увеличу масштаб терминала и снова перечислю содержимое моего каталога, просто чтобы дать нам ориентир. Позволять'

Ну, опять же, я мог бы щелкнуть правой кнопкой мыши в левой части, выбрать «Новая папка» и создать каталог pset2. Это тоже сработало бы. Но затем я также делаю это в командной строке довольно быстро, набирая mkdir, пробел - я набрал n, но - пробел pset2. Если я нажимаю Enter, а затем снова перечисляю содержимое своего каталога, я вижу, что теперь у меня есть папка pset2. И я могу перейти к этому с помощью cd и выполнить всю работу, которую мне нужно сделать для pset2. Кстати, я просто очень быстро открою здесь дерево файлов. И вы можете видеть, что и в графическом проводнике файлов мы видим, что там также был создан каталог pset2. И я могу перейти к нему, используя графический интерфейс. В следующий раз, и это, вероятно, пригодится, это cp, что означает сокращение для копирования. Копировать, в отличие от всех других команд, которые мы видели раньше, принимает два аргумента: источник, имя файла, который вы хотите скопировать, и место назначения, куда вы хотите скопировать файл. Копировать файл довольно легко, так что давайте сделаем это.

Итак, мы вернулись в IDE. Я собираюсь перечислить содержимое моего текущего каталога с помощью ls. Теперь предположим, что я хочу сделать копию hello.txt. Опять же, из дерева файлов слева, графического интерфейса, я могу щелкнуть правой кнопкой мыши на hello.txt, сделать копию, вставить копию. Но я могу сделать это довольно быстро и в командной строке.

Допустим, я хочу скопировать hello.txt в hi.txt. Я могу cp, space, hello.txt. Это мой исходный файл, поэтому я выберу его первым. И тогда мне нужно назвать файл назначения, hi.txt. Я нажал Enter. И если я снова перечислю содержимое моего каталога, там будет hi.txt. Я сделал копию этого. И на самом деле, если я зайду в hi.txt, я смогу увидеть, что это будет точная копия всего, что существовало в моем файле hello.txt.

Так вот, как вы копируете файл. Но что, если вы хотите скопировать весь каталог? Итак, на секунду давайте посмотрим, что находится в моем каталоге pset0. Если я cd pset0 и перечислю содержимое, у меня будет каталог с названием sample и scratch file, scratch.sb2. Так что это приятно знать. Итак, давайте очистим экран, и я на секунду вернусь к своему каталогу рабочей области.

Допустим, теперь я хочу сделать копию моего каталога pset0. Я не могу просто сказать, например, cp pset0 pset3. Вы получаете это странное сообщение, опуская каталог pset0. Почему вы получаете это сообщение? Что ж, получается, что когда у вас есть каталог, в котором есть другие вещи, команда cp не обязательно знает, что с ним делать.

Нам нужно явно указать Linux, терминалу, что я хочу, чтобы вы скопировали каталог pset0 и скопировали каждую папку, которая существует внутри него, и каждый файл, который существует внутри него. Другими словами, мне нужно, чтобы вы рекурсивно погрузились в pset0 и сделали копию всего, что там есть.

Если я хочу это сделать, то я могу сделать cp-r для рекурсии pset0 pset3. Нажмите Enter. Теперь, если я перечислю содержимое моего каталога, я вижу, что есть каталог pset3, с которым я могу работать. И если я перейду в pset3 время от времени перечислю содержимое, посмотри, есть sample и scratch.sb2 снова. Это довольно круто. Так вот, как вы можете скопировать весь каталог, а не только один файл. Поэтому, если вы хотите скопировать каталог, просто не забудьте использовать флаг -r при работе с командой cp. Отлично. Итак, я скопировал файл, но сделал это по ошибке. И теперь я хочу избавиться от этого. Как я могу это сделать? Опять же, если вы знакомы с интерфейсом GUI, вы можете щелкнуть правой кнопкой мыши и просто выбрать «Удалить». И он отправит его в корзину или корзину. Но в командной строке мы можем просто набрать rm для удаления,

Теперь Р.М. очень осторожен. Он выполняет многократную проверку, чтобы убедиться, что вы действительно хотите удалить файл. Он не хочет делать никаких ошибок, потому что здесь нет корзины. Как только мы удалим файл, он исчезнет. И нет никакого способа восстановить это. Поэтому мы должны быть очень осторожны, и поэтому мы будем очень осторожны с нами.

Итак, давайте вернемся к IDE и избавимся от того файла hi.txt, который я создал минуту назад с копией. Итак, мы в IDE. И я снова перечисляю содержимое моего каталога, просто чтобы дать нам некоторый контекст. Я имею в виду, вы можете видеть, что есть hi.txt, который я создал недавно. Теперь я хочу избавиться от этого. Как мне это сделать?

Опять просто гм. Поэтому я могу набрать rm hi.txt и нажать Enter. И мы очень осторожны и уверены, что мы действительно хотим удалить этот файл. Вы действительно хотите удалить обычный пустой файл hi.txt? Это похоже на операционную систему, такую ​​как Windows или Mac, с которой вы, возможно, уже знакомы, если всплывающее окно говорит, что вы действительно уверены, что хотите это сделать.

Я почти уверен, что хочу этого, поэтому я собираюсь напечатать, да ... или я мог бы просто напечатать y-- и нажать Enter. И если я снова перечислю содержимое моего каталога, hi.txt исчезнет. Не плохо, правда? Давайте нажмем Control L, просто чтобы избавиться от всего этого и вернуть нас вверху экрана.

Теперь есть способ короткого замыкания rm, чтобы он не задавал нам этот вопрос. Мы действительно знаем, что хотим избавиться от этого файла, и даже не хотим, чтобы нам задавали вопрос. Как мы это делаем? Ну, мы можем указать дополнительный флаг, как мы сделали с cp, где мы могли бы добавить флаг -r, для рекурсивного копирования в каталог. Есть еще один флаг для rm, который -f, который заставляет rm делать именно то, что мы говорим.

Итак, давайте скажем, что теперь я хочу избавиться от моего файла hello.txt. Я тоже этого не хочу. Что я могу сделать? Ну, я могу rm -f hello.txt. Как вы думаете, что здесь произойдет? Это прошло. Даже не задавал мне вопрос. И на самом деле, если я снова перечислю содержимое своего каталога, он исчезнет. Там нет отмены здесь. hello.txt пропал, и меня даже не спросили, хочу ли я удалить его. Что если я хочу избавиться от целого каталога? То, как вы это делаете, очень похоже на копирование. Если вы хотите удалить каталог, вы не просто хотите удалить каталог, вы хотите удалить все внутри него. И поэтому вы используете -r, чтобы рекурсивно удалить каталог.

Итак, давайте rm -r pset2. На самом деле я не работаю над pset2, поэтому могу избавиться от этого каталога. Я нажал Enter. Хочу ли я удалить каталог, pset2? Да. И если я снова перечислю содержимое моего каталога, он исчезнет. Отлично?

Еще одна вариация на эту тему. Так что я снова очистлю экран, чтобы все было сверху. Хит ЛС. Теперь я хочу избавиться от своего каталога pset3, и я действительно знаю, что хочу избавиться от своего каталога pset3. Я могу рекурсивно и принудительно удалить мой каталог pset3.

Теперь вы будете очень осторожны, когда используете rm -rf. По мере того, как вы будете программировать больше, вы будете делать это очень быстро. И на самом деле, я делаю это все время. Но это может привести к некоторым катастрофическим последствиям, если вы случайно удалите не ту вещь. Поэтому я хочу, чтобы вы знали, что этот вариант доступен для вас, но используйте его с осторожностью и используйте его с осторожностью.

Я точно знаю, что больше не работаю над своей проблемой 3, поэтому я собираюсь избавиться от всех этих файлов. И я знаю, что не хочу, чтобы меня каждый раз спрашивали, хочу ли я удалить файл, поэтому я собираюсь выполнить команду rm -rf pset3. Даже не спросил меня, хочу ли я от этого избавиться. Я ударил ЛС. pset3 ушел. Так что это все варианты, которые вы можете использовать, чтобы избавиться от файлов с помощью rm.

Последняя команда командной строки, которая будет очень полезна для вас, это mv, что означает сокращение для перемещения. Перемещение в основном эквивалентно переименованию. Он перемещает файл из одного места в другое, от источника к месту назначения. Давайте посмотрим, где использование перемещения может быть удобно в окне терминала.

Так что я нахожусь в моем каталоге pset1, и я заметил, что что-то пошло не так. Я работал над своей жадной проблемой, но случайно назвал ее greddy. Поэтому, когда я пытаюсь запустить его через check50, он на самом деле не работает. У меня есть пара вариантов. Мы могли бы сделать то, что мы делали раньше, то есть сделать копию файла. Я мог бы скопировать greddy.c в greedy.c, верно? Нажмите Enter. Я вижу, что оба файла есть. И тогда я смогу удалить, greddy.c. Это избавляет от этого. Так что это сработает после того, как я подтверждаю, что хочу избавиться от этого. И я фактически переименовал greddy.c в greedy.c. Я скопировал его, а затем удалил оригинал.

Но это многошаговый процесс. И, конечно, есть лучший способ. На самом деле, есть. Итак, давайте нажмем Control L, просто чтобы восстановить это, и снова перечислить содержимое моего каталога.

Так скажем, о, я допустил ошибку. Я действительно очень хотел назвать этот файл greddy.c. Одним махом я могу напечатать move greedy.c к greddy.c. Нажмите Enter. И теперь мне не нужно было копировать и удалять его, я просто смог переименовать его. Шоппинг с одной остановкой.

Сейчас есть много других базовых утилит командной строки, которые вы можете использовать с командной строкой Linux. И мы обсудим многие из них в будущем в CS50. Но пока, просто погрузившись в работу с этой терминальной средой, эти пять команд позволят вам достаточно далеко ориентироваться в работе с файлами в вашей IDE или в любой операционной системе на основе Linux, которую вы используете.

Если вам любопытно, и вы хотите немного заглянуть в будущее по некоторым командам командной строки, которые мы будем использовать в будущем, вот список из нескольких наиболее распространенных. Я Даг Ллойд. Это CS50.

Отлично. Итак, теперь давайте поговорим о чем-то действительно полезном в программировании - условных выражениях. Таким образом, условные выражения позволяют вашим программам принимать решения и принимать различные решения, что я упоминал немного раньше, в зависимости от значений переменных или от того, что пользователь вводит в программист, в командную строку или, если вы есть подсказка или что-то в этом роде. В C есть несколько различных способов выражения условных выражений, которые мы также иногда будем называть условными ветвями в ваших программах. И некоторые из них будут выглядеть довольно знакомыми для вас с нуля, так что мы даже потянем их рядом, просто вы можете провести эту аналогию в своей голове. Итак, если-- если это довольно простое условие. Если вы помните с нуля справа здесь, вы можете заполнить это шестиугольником с синим выражением, если мышь нажата, или если x меньше 10, или что-то в этом роде. И затем, если x меньше 10 или мышь на самом деле нажата, весь код внутри фрагмента головоломки будет выполнен. Все вещи, которые вписываются в эту форму C.

Точно так же у нас, если слева там. Если булево выражение, которое я просто использую вместо одного из булевых выражений, которые мы обсуждали ранее, откройте фигурную скобку, закройте фигурную скобку. Поэтому представьте открытую фигурную скобку и закрытую фигурную скобку как своего рода аналог сэндвич-эффекта блока if справа с нуля.

Если логическое выражение в операторе if истинно, то все строки кода между фигурными скобками будут выполняться в порядке сверху вниз. Если логическое выражение ложно, мы пропустим все, что находится между фигурными скобками, потому что мы хотим идти по этой развилке только в том случае, если логическое выражение истинно.

Мы можем сделать этот шаг вперед, если еще. Таким образом, этот блок Scratch очень похож на тот, который мы видели всего секунду назад, за исключением того, что он использует два разных пути в зависимости от того, что происходит. Так что, если мышь была выключена, или если х был меньше 10, мы сделаем все, что находится между первой вилкой, первым C.

В противном случае, если мышь поднята или х не меньше 10, мы сделаем все во втором наборе. И это аналогично тому, что вы видите здесь для C. Если булево выражение, делайте вещи между первым набором фигурных скобок. Иначе, делайте вещи между вторым набором фигурных скобок. Так что, если логическое выражение истинно, мы сделаем все, что между первым набором. Если булево выражение ложно, это вызовет else, и мы сделаем все, что есть во втором наборе фигурных скобок. Опять сверху вниз, все линии между скобками.

В C можно создать цепочку if-else if-else. На самом деле вы можете иметь if-else if-else if-else, if и так далее, и так далее, и так далее. В Scratch это требовало вложения блоков. Вы добавляете if-else, и вам нужно было поместить еще один внутри else и т. Д., И это стало немного сложным и сложным. Но С, мы не должны это делать. На самом деле мы можем просто иметь такую ​​цепочку. Опять же, как вы могли ожидать, все эти ветви являются взаимоисключающими. Вы можете когда-нибудь спуститься только на одну ветку. Если это правда. В противном случае, если это правда. В противном случае, если это правда. В противном случае, сделайте это. Таким образом, все четыре ветви в этом примере являются взаимоисключающими. Это цепочка if-else if-else.

Возможно, и иногда очень полезно, создать цепочку не взаимоисключающих ветвей. В этом примере только третья и четвертая ветви являются взаимоисключающими. Может случиться так, что вы могли бы удовлетворить первое условие, и вы могли бы удовлетворить второе условие, и вы могли бы удовлетворить третье условие - в этом случае вы пошли бы вниз по первой ветви, затем вы пошли бы по второй ветви, а затем вы бы спуститься в третью ветку. Или, возможно, вы удовлетворяете первое условие и второе условие, но вы не удовлетворяете третье условие. В этом случае вы идете вниз по первой ветви и второй ветви, а затем четвертой ветви,

Причина этого заключается в том, что остальное будет связываться только с ближайшим, если. Так что, хотя здесь есть еще что-то, это не обязательно создает взаимоисключающую цепочку всего. Это только выражение с логическим выражением 3 - это взаимоисключающее с остальным. Так что, как я уже сказал, возможно, а иногда и весьма полезно, создать цепочку не взаимоисключающих ветвей. Давайте посмотрим на другой вид условного выражения, которого вы еще не видели в Scratch. Там что-то называется оператором switch. Оператор switch довольно аккуратный, потому что это условный оператор, который позволяет вам указывать разные случаи, а не полагаться на логические выражения для принятия решений за вас. Так, например, скажем, что у меня есть эта программа, и я Я прошу пользователя предоставить мне информацию. Поэтому я говорю, что int x = Get Int (), и если вы еще не знакомы, get int - это функция, которая также включена в библиотеку CS50, поэтому, если вы #include CS50.H, у вас будет доступ к Get Int () и все его родственники - GetFloat, GetString и так далее. В основном одна функция Get для каждого типа данных, который мы уже обсуждали.

Так что Int x равен GetInt. В основном то, что происходит, я на терминале. Я прошу пользователя ввести номер.

И здесь я переключаюсь, что я делаю, в зависимости от того, что пользователь набрал в командной строке. Так что, если они напечатали один, я распечатал один. И тогда я ломаюсь. Если они напечатают два, я распечатаю два. И тогда я ломаюсь. Важно разрываться между каждым случаем, потому что иначе вы провалитесь. Так что, если бы у меня там не было никаких перерывов, а пользователь набрал один, то получилось бы, если бы он напечатал один, два, три, извините. Это странное поведение, верно? Вы можете так думать. Но на самом деле есть некоторые случаи, когда это может быть довольно полезным. Итак, вот еще один пример оператора switch, где я опускаю разрывы. Но я делаю это специально.

Так что здесь происходит? Подумай на секунду. Вы можете даже хотеть приостановить видео.

Что происходит здесь, если пользователь вводит четыре? Итак, я попросил пользователя для ввода. И они дают значение 4. Что печатается, когда я это делаю? На предыдущем слайде были перерывы между всеми делами. И так было бы просто напечатать четыре, а затем остановиться. Но в этом случае это не так. Что произойдет, вы провалите каждый случай.

Так что в этом случае я организовал свои кейсы таким образом, что если пользователь введет 4, я напечатаю четыре, три, два, один, взорваться. И если бы они набрали 5, я бы начал с пяти и делал то же самое. Если бы они набрали 1, я бы просто сделал один, взлететь.

Так что в этом случае я использую умный переключатель, так что я собираюсь пройти через все случаи. Но, как правило, вы, вероятно, захотите порвать между всеми ними, если только у вас нет ситуации, подобной этой, где вы используете тот факт, что вы провалите дела без перерыва. Так что это второй из основных типов условных операторов. Последний из которых ?: Итак, у меня есть два фрагмента кода C здесь. Один слева и один справа. Тот, что слева, вероятно, должен быть вам знаком.

У меня есть Int x. И мне, вероятно, следовало бы спросить пользователя - это должно быть Int x равно GetInt или что-то в этом роде. И тогда я принимаю решение. Если какое-либо логическое выражение имеет значение true, присвойте x значение 5. В противном случае присвойте x значение 6.

Это слева должно быть довольно знакомо из нашего обсуждения If Else только минуту назад. Вы были бы удивлены, узнав, что линия справа делает то же самое?

Так это называется ?: или иногда называется троичным оператором. И это довольно круто. Это обычно используется как милый трюк.

Но то, что он позволяет вам делать, - это моделировать If Else с очень маленькими, очень тривиально короткими условными ветвями. Вы вообще не будете использовать ?: если бы у вас было шесть строк кода между каждым набором фигурных скобок. Но если вы просто принимаете быстрое решение, если вы собираетесь делать одно или другое, и это очень просто, это может быть примером того, как это сделать?: Троичный оператор. Значит Int x равно выражению? Вещество после знака вопроса - это значение x, если выражение истинно.

Значение после двоеточия - это значение x, если выражение было бы ложным. Вот я и спрашиваю себя: верно ли это выражение? Если это так, присвойте x значение 5. Если это не так, присвойте x значение 6. Снова, как я уже сказал. Обычно это просто милый трюк. И иногда, если вам это действительно удобно, вы сделаете это, потому что это выглядит круто в ваших программах. Обычно я представляю это вам сейчас, так что вы знакомы с ним, если вы видите это. Но, конечно, знайте, что вам не нужно писать это ни в каком коде. Но с этим нужно ознакомиться, потому что вы определенно встретите фрагменты кода здесь и там, где используется этот синтаксис: AKA, троичный оператор.

Итак, краткое описание условий и доступных вам вариантов в C. У вас есть If, if-else и if if if, и так далее. Вы можете использовать логические выражения для тех, кто принимает решения.

С заявлениями переключателя вы используете отдельные случаи для принятия решений. Вы бы конкретно сказали, если это один, или если это два, или если это три, я сделаю это, или это, или это. И?: Можно использовать для замены очень простых ветвей if-else или цепочек if-else, чтобы ваш код выглядел немного причудливым.

Я Даг Ллойд. И это CS50.

[ВОСПРОИЗВЕДЕНИЕ МУЗЫКИ] СПИКЕР: Хорошо. Итак, давайте поговорим о другой вещи, которая в некотором роде уникальна для C, это типы данных и переменные. Когда я говорю «уникально для C», я имею в виду только то, что если вы были программистом в течение очень долгого времени, вы, вероятно, не работали с типами данных, если использовали современные языки программирования. Современные языки, такие как PHP и JavaScript, которые мы также увидим чуть позже в курсе, на самом деле вам не нужно указывать тип данных переменной при ее использовании.

Вы просто заявляете об этом и начинаете его использовать. Если это целое число, оно знает, что это целое число. Если это персонаж, он знает, что это персонаж. Если это слово, оно знает, что это строка, так называемая.

Но в C, который является более старым языком, нам нужно указать тип данных для каждой переменной, которую мы создаем при первом использовании этой переменной. Таким образом, C поставляется с некоторыми встроенными типами данных. И давайте познакомимся с некоторыми из них. А потом мы также немного поговорим о некоторых типах данных, которые мы написали для вас, чтобы вы могли использовать их в CS50.

Первый - инт. Тип данных int используется для переменных, которые будут хранить целочисленные значения. Итак, 1, 2, 3, отрицательные 1, 2, 3 и так далее. Целые числа, которые следует учитывать при тестировании, всегда занимают четыре байта памяти, что составляет 32 бита. В байте восемь битов.

Таким образом, это означает, что диапазон значений, которые может хранить целое число, ограничен тем, что может уместиться в пределах 32-битной информации. Теперь, как выяснилось, давно было решено, что мы разделим этот диапазон в 32 бита на отрицательные целые и положительные целые, каждый из которых получит половину диапазона. Таким образом, диапазон значений, которые мы представляем целочисленным диапазоном от минус 2 до 31-й степени и от 2 до 31-й степени минус 1, потому что вам также нужно место для 0.

Таким образом, в основном половина возможных значений, которые вы можете поместить в int, отрицательна, а половина положительна. И примерно здесь, это примерно от 2 миллиардов до примерно 2 миллиардов. Дай или возьми пару сотен миллионов. Так вот что вы можете поместить в целочисленную переменную. Теперь у нас также есть нечто, называемое целым числом без знака. Теперь целые числа без знака не являются отдельным типом переменных. Скорее, unsigned - это то, что называется классификатором. Это немного изменяет тип данных integer.

И в этом случае, что означает unsigned - и вы также можете использовать другие типы данных без знака, целое число не единственное. Фактически он удваивает положительный диапазон значений, который может принимать целое число, за счет того, что он больше не позволяет принимать отрицательные значения. Так что, если у вас есть числа, которые, как вы знаете, получат более 2 миллиардов, но менее 4 миллиардов, например, что составляет 2 от 32-й степени, вы можете использовать целое число без знака, если знаете, что ваше значение никогда не будет отрицательным ,

Время от времени вы будете использовать для беззнаковых переменных в CS50, именно поэтому я упоминаю об этом здесь. Но, опять же, диапазон значений, которые вы можете представить с помощью целого числа без знака относительно обычного целого числа, составляет от 0 до 2 до 32-й степени минус 1 или приблизительно от 0 до 4 миллиардов. Таким образом, вы фактически удвоили положительный диапазон, который можете уместить, но вы отказались от всех отрицательных значений.

Кроме того, unsigned - не единственный квалификатор, который мы можем увидеть для переменных типов данных. Есть также вещи, называемые короткие и длинные и постоянные. Const мы увидим чуть позже в курсе. Коротко и долго, мы, вероятно, не будем.

Но просто знайте, что есть и другие классификаторы. Неподписанный не единственный. Но это единственный, о котором мы сейчас поговорим. Так что все в порядке. Итак, мы рассмотрели целые числа. Что дальше?

Символы. Поэтому символы используются для переменных, которые будут хранить отдельные символы. Char - сокращение от персонажа. И иногда вы можете услышать, как люди произносят это как автомобиль.

Таким образом, символы всегда занимают один байт памяти, который составляет всего 8 бит. Таким образом, это означает, что они могут соответствовать значениям в диапазоне от отрицательных 2 до седьмой степени или от отрицательных 128 до 2 до 7-й степени минус 1 или 127.

Благодаря ASCII давно был решен способ сопоставить эти положительные числа от 0 до 127 различным символам, которые существуют на нашей клавиатуре. Итак, как мы увидим позже в курсе, и вы, вероятно, придете в запоминание в какой-то момент, например, заглавная буква A - заглавная буква A - соответствует номеру 65. И причина этого в том, что это то, что назначено стандартом ASCII.

Строчная буква A - 97. Символ 0, когда вы на самом деле печатаете символ, не представляющий число ноль, - 48. Вы изучите пару из них по ходу дела. И они вам наверняка понадобятся чуть позже в CS50.

Следующий основной тип данных - числа с плавающей запятой. Так что числа с плавающей запятой также известны как действительные числа. В основном это числа с десятичной точкой. Значения с плавающей точкой, такие как целые числа, также содержатся в 4 байтах памяти. Теперь здесь нет графика. Там нет числовой линии, потому что описание диапазона с плавающей точкой не совсем понятно или интуитивно понятно.

Достаточно сказать, что у вас есть 32 бита для работы. И если у вас есть число типа pi, которое имеет целочисленную часть 3, и часть с плавающей запятой, или десятичную часть 0,14159 и т. Д., Вы должны быть в состоянии представить все это - целую часть и десятичную часть ,

Так что вы думаете, что это может означать? Одна вещь состоит в том, что если десятичная часть становится все длиннее и длиннее, если у меня очень большая целочисленная часть, я не смогу быть настолько точным с десятичной частью. И это действительно ограничение поплавка.

У поплавков есть проблема точности. У нас есть только 32 бита для работы, поэтому мы можем быть настолько точными только с нашей десятичной частью. Мы не можем иметь десятичную часть с точностью до 100 или 200 цифр, потому что у нас есть только 32 бита для работы. Так что это ограничение поплавка.

К счастью, есть еще один тип данных, называемый double, который несколько решает эту проблему. Двойные числа, такие как числа с плавающей точкой, также используются для хранения действительных чисел или значений с плавающей запятой. Разница в том, что двойные имеют двойную точность. Они могут вместить 64 бита данных или восемь байтов. Что это значит? Ну, это означает, что мы можем быть более точными с десятичной точкой. Вместо числа пи в семи местах, возможно, с поплавком, мы можем иметь его в 30 местах. Если это важно, вы можете использовать double вместо float. В основном, если вы работаете над чем-то, где важно иметь очень длинный десятичный знак и большую точность, вы, вероятно, захотите использовать двойное плавающее число. Теперь для большей части вашей работы в CS50 достаточно поплавка.

Теперь это не тип данных. Это тип. И это называется пустота. И я говорю об этом здесь, потому что мы, наверное, видели это несколько раз уже в CS50. И вам может быть интересно, о чем это все.

Так что void - это тип. Это существует. Но это не тип данных.

Мы не можем создать переменную типа void и присвоить ей значение. Но функции, например, могут иметь тип возврата void. По сути, если вы видите функцию с возвращаемым типом void, это означает, что она не возвращает значение. Можете ли вы вспомнить общую функцию, которую мы использовали до сих пор в CS50, которая не возвращает значение?

Printf один. Printf на самом деле ничего вам не возвращает. Он выводит что-то на экран, и это в основном побочный эффект от того, что делает printf. Но это не вернет вам ценность. Вы не фиксируете результат и не сохраняете его в некоторой переменной, чтобы использовать его позже. Он просто печатает что-то на экране, и все готово.

Итак, мы говорим, что printf является функцией void. Ничего не возвращает

Список периметра функции также может быть недействительным. И вы тоже это немного видели в CS50. Int главная пустота. Это звонит в колокол? В основном это означает, что main не принимает никаких параметров. Там нет аргумента, который передается в основной. Позже мы увидим, что есть способ передать аргументы в main, но пока что мы видели int main void. Главное просто не принимает никаких аргументов. И поэтому мы уточняем это, говоря void. Мы просто очень четко говорим о том, что это не требует никаких аргументов.

Итак, на данный момент достаточно сказать, что пустота в основном должна служить для вас заполнителем, а не думать о чем-либо. Это на самом деле ничего не делает. Здесь нет возвращаемого значения. Здесь нет параметров. Это пустота. Это немного сложнее, чем это. Но этого должно быть достаточно для большей части курса. И, надеюсь, теперь у вас есть немного больше понятия о том, что такое пустота.

Вот те пять типов, с которыми вы столкнетесь, встроены в C. Но в CS50 у нас также есть библиотека. CS50.h, который вы можете включить. И который предоставит вам два дополнительных типа, которые вы, вероятно, сможете использовать в своих заданиях или просто работать в общем программировании.

Первый из них это bool. Таким образом, логический тип данных, bool, используется для переменных, которые будут хранить логическое значение. Если вы когда-либо слышали этот термин раньше, вы можете знать, что логическое значение может содержать только два разных значения. Правда и ложь. Теперь это кажется довольно фундаментальным, верно? Это своего рода сюрприз, что его нет в C, поскольку он встроен. И во многих современных языках, конечно, логические значения являются стандартным типом данных по умолчанию. Но в Си они на самом деле нет. Но мы создали это для вас. Поэтому, если вам когда-нибудь понадобится создать переменную с типом bool, просто обязательно включите CS50.h в начале вашей программы, и вы сможете создавать переменные типа bool.

Если вы забудете #include CS50.h и начнете использовать переменные логического типа, вы можете столкнуться с некоторыми проблемами при компиляции программы. Так что просто следите за этим. И, возможно, вы можете просто решить проблемы по фунту, включая CS50.h.

Другим основным типом данных, который мы предоставляем для вас в библиотеке CS50, является строка. Так что же такое строка? Строки на самом деле просто слова. Это коллекции персонажей. Это слова. Это предложения. Это параграфы. Может быть, даже целые книги.

Очень короткие или очень длинные серии символов. Если вам нужно использовать строки, например, для хранения слова, просто включите CS50.h в начале вашей программы, чтобы вы могли использовать строковый тип. И тогда вы можете создавать переменные, тип данных которых является строкой. Теперь позже в курсе, мы также увидим, что это еще не вся история. Мы столкнемся с вещами, называемыми структурами, которые позволяют вам группировать то, что может быть целым числом и строкой, в один блок. И мы можем использовать это для какой-то цели, которая может пригодиться позже в курсе.

И мы также узнаем об определенных типах, которые позволяют вам создавать свои собственные типы данных. Нам не нужно беспокоиться об этом сейчас. Но просто знайте, что это что-то на горизонте, что в этом типе есть намного больше, чем я вам сейчас говорю. Итак, теперь, когда мы немного узнали об основных типах данных и типах данных CS50, давайте поговорим о том, как работать с переменными и создавать их с использованием этих типов данных в наших программах. Если вы хотите создать переменную, все, что вам нужно сделать, это две вещи.

Во-первых, вам нужно дать ему тип. Второе, что вам нужно сделать, это дать ему имя. После того как вы сделали это и добавили точку с запятой в конце этой строки, вы создали переменную.

Итак, вот два примера. Int номер; буква чар; Что я здесь сделал? Я создал две переменные.

Во-первых, имя переменной - это число. И число может содержать значения целочисленного типа, потому что его тип - int. Letter - это еще одна переменная, которая может содержать символы, потому что ее тип данных - char.

Довольно просто, правда? Если вы оказались в ситуации, когда вам нужно создать несколько переменных одного типа, вам нужно указать имя типа только один раз. Затем просто перечислите столько переменных этого типа, сколько вам нужно.

Таким образом, я мог бы, например, здесь, в этой третьей строке кода, сказать int height ;, новая строка. Int ширина ;. И это тоже сработало бы. Я бы по-прежнему получал две переменные с именем height и width, каждая из которых является целым числом. Но мне позволено, вещи с синтаксисом C, объединить его в одну строку. Высота, ширина; Это то же самое. Я создал две переменные, одну с именем height, одну с именем width, обе из которых способны содержать значения целочисленного типа.

Точно так же здесь я могу создать три значения с плавающей точкой одновременно. Я могу, возможно, создать переменную с именем квадратный корень из 2 - которая, вероятно, в конечном итоге будет содержать плавающую точку - это представление квадратного корня из 2 - квадратный корень из 3 и пи. Я мог бы сделать это в трех отдельных строках. Поплавок, квадратный корень 2; Квадратный корень с плавающей точкой 3; плавать пи; и это тоже сработало бы.

Но опять же, я могу просто объединить это в одну строку кода. Делает вещи немного короче, не так неуклюже.

В общем, хороший дизайн - объявлять переменную только тогда, когда она вам нужна. И об этом мы поговорим чуть позже, когда будем обсуждать сферу применения. Поэтому не обязательно создавать все свои переменные в начале программы, что некоторые люди, возможно, делали в прошлом или, безусловно, было очень распространенной практикой программирования много лет назад при работе с C. Возможно, вы просто захотите создать переменная прямо тогда, когда вам это нужно. Отлично. Итак, мы создали переменные. Как мы их используем? После того, как мы объявили переменную, нам больше не нужно указывать тип данных этой переменной. На самом деле, если вы сделаете это, вы можете столкнуться с некоторыми странными последствиями, которые мы пока замаскируем. Но достаточно сказать,

Итак, у меня есть четыре строки кода. И у меня есть пара комментариев, просто показывающих, что происходит в каждой строке, чтобы помочь вам разобраться в происходящем. Так что int номер ;. Вы видели это ранее. Это объявление переменной.

Теперь я создал переменную с именем number, которая может содержать значения целочисленного типа. Я объявил это.

В следующей строке я присваиваю значение номеру. Число равно 17. Что там происходит? Я помещаю число 17 внутри этой переменной.

Поэтому, если я когда-нибудь потом распечатаю содержимое числа, они скажут мне, что это 17. Поэтому я объявил переменную, а затем назначил ее.

Мы можем повторить процесс снова с буквой char ;. Это декларация. Буква равна заглавной H. Это задание. Довольно просто тоже.

Теперь этот процесс может показаться глупым. Почему мы делаем это в две строки кода? Есть ли лучший способ сделать это? На самом деле, есть. Иногда вы можете увидеть это называется инициализация. Это когда вы объявляете переменную и присваиваете значение одновременно. Это на самом деле довольно распространенная вещь. Когда вы создаете переменную, вы обычно хотите, чтобы она имела какое-то базовое значение. Даже если это 0 или что-то. Вы просто даете это значение.

Вы можете инициализировать переменную. Int число равно 17 - это то же самое, что первые две строки кода выше. Буква Char, равная h, совпадает с третьей и четвертой строками кода выше. Самое важное, что мы можем сделать здесь, когда мы объявляем и присваиваем переменные, это после того, как мы объявили это, обратите внимание, что я больше не использую тип данных. Я не говорю, что целое число равно 17 во второй строке кода, например. Я просто говорю, что число равно 17.

Опять же, повторное объявление переменной после того, как вы уже объявили ее, может привести к некоторым странным последствиям. Так что будьте осторожны с этим.

Я Даг Ллойд. И это CS50.

DOUG LLOYD: Хорошо, давайте поговорим о петлях. Так что циклы - это круто, потому что они позволяют вашей программе многократно выполнять строки кода. Неоднократно, без необходимости копировать и вставлять или иным образом повторять их. Вы увидите три основных типа петель. Вероятно, у вас будет возможность использовать каждый из них по мере прохождения CS50. Некоторые из этих циклов довольно знакомы вам с нуля, поэтому, опять же, как мы делали с условными обозначениями, мы разместим их рядом, если есть аналогия, которую мы можем провести.

Первый - это навсегда от Scratch, который сравнивается с while (true). Это то, что мы называем бесконечным циклом. Строки кода между фигурными скобками, так же, как кусочки головоломки, которые помещаются внутри С блока навсегда, будут выполняться многократно сверху вниз, снова и снова, снова и снова, навсегда. В конце концов, если вы вспомните наше обсуждение булевых выражений, true всегда будет true. Так что, если мы делаем что-то, пока true - правда, это кажется немного глупым - true всегда верно, поэтому всегда будет работать. Он будет работать вечно, во веки веков, пока мы не найдем способ вырваться из него с помощью оператора break, который, опять же, мы видели некоторое время назад, когда говорили о switch.

Или иначе просто убейте нашу программу, что, кстати, если вы когда-нибудь окажетесь в ситуации с бесконечным циклом, и вы не знаете, как остановить выполнение вашей программы. Просто нажмите Control и C, и это убьет вашу программу для вас. Но это опять-таки называется бесконечным циклом. Это цикл, который будет работать вечно. Теперь, хотя циклы while не всегда бесконечны, потому что мы также можем заменить логическое выражение чем-то более полезным, чем просто true. Итак, вот еще один пример. Хотя и некоторые логические выражения, а затем фигурные скобки. Это в значительной степени аналогично повторять до нуля. Хотя через секунду я объясню разницу.

Таким образом, в C, если логическое выражение внутри цикла while имеет значение true, строки кода между фигурными скобками будут повторяться многократно, пока логическое выражение не станет false. Так, например, у вас может быть какой-то счетчик. Скажем, в начале этого цикла скажем, что int x равен нулю. Хотя x меньше 100, сделайте пару вещей внутри этих фигурных скобок, а затем в конце цикла вы говорите, что x плюс плюс, вы увеличиваете x. В конце концов x достигнет 100, и вы остановитесь, потому что условие x меньше 100 больше не будет истинным, как только x станет равным 100. 100 не меньше 100.

Теперь несколько запутанно, поведение скретч-блока полностью изменено. Так что будьте очень осторожны, если это ваш первый опыт использования петель. По сути, повторение до тех пор, пока блок в Scratch будет повторять что-то снова и снова, пока выражение, логическое выражение в шестиугольнике не станет истинным. Так что он будет продолжать делать это, пока не станет правдой. Между тем, цикл while продолжит делать это, пока не станет ложным. Так что они очень похожи, но есть небольшое отличие, которое нужно соблюдать, особенно когда вы совершаете первый набег от перехода к Scratch в c. Следующий вид цикла на самом деле очень похож. Это называется циклом do while. Этот цикл выполнит все строки кода между фигурными скобками один раз, а затем проверит логическое выражение. Если булево выражение имеет значение true, оно ' Я буду возвращаться и повторять этот процесс снова и снова, пока логическое выражение не станет ложным. Таким образом, этот цикл, в отличие от цикла while, гарантированно будет запущен как минимум один раз. Это может быть довольно полезно.

Наконец, у нас есть цикл for. Поскольку петли являются синтаксически непривлекательными, и здесь многое происходит, так что терпите меня, пока мы пытаемся разбить эти части на части. Как правило, в случае использования цикла for вы хотите повторить что-то определенное количество раз. В этом примере у меня есть блок повтора с нуля, который аналогичен циклу for в C, повторяя что-то 10 раз. И цикл for слева, который представляет собой четыре цикла, которые будут делать то же самое. Это будет идти 10 раз. Считая с нуля, мы увеличиваем каждый проход цикла. И мы продолжаем делать это, пока мне не исполнилось 10 лет.

Так что же происходит? Там много кода. Давайте разберем это с тем, что происходит, шаг за шагом, пока мы копаем цикл for. Первое, что произошло, - это переменная counter - center. Все слева от этой первой точки с запятой в скобках выполняется. В этом случае мы говорим, что int i равно 0. Объявляя новую переменную с именем i, эти переменные являются целыми числами типа, и мы говорим, что значение внутри этой переменной будет равно нулю.

Второе, что мы делаем, - это сразу оцениваем логическое выражение. Логическое выражение находится в середине двух точек с запятой. Мне меньше 10. Это правда, верно? Мы только что сказали, что i равен нулю, и поэтому 0 меньше 10, и поэтому условие истинно, и мы теперь приступим к выполнению тела цикла. Если бы это было ложно. Если, например, я сказал, что вместо i равно 0, то я сказал, что i равен 15, int i равен 15. Ну, 15 не меньше 10, поэтому мы никогда не войдем в тело цикла, потому что там логическое выражение будет иметь значение быть ложным

После прохождения сверху вниз мы сталкиваемся с этой закрывающей фигурной скобкой, происходит третье. Переменная счетчика увеличивается, или, скорее, строки кода оператора после второй точки с запятой внутри скобок цикла for. Итак, начнем с того, что int i равно 0. Затем мы проверяем, является ли логическое выражение истинным. 0 меньше 10, так что это правда. Итак, мы собираемся перейти в тело цикла. В цикле будут происходить события, и когда мы сталкиваемся с этой закрывающей фигурной скобкой, следующее, что мы делаем, мы говорим «плюс» плюс. Я был нулем, теперь я один. Затем мы снова проверяем значение логического выражения в середине. Один меньше 10. Поэтому мы пройдем этот процесс снова и снова. Мы вернемся к закрывающей фигурной скобке снова. Мы будем увеличивать i с 1 до 2 и с 2 до 3. И так далее, и так далее, до тех пор, пока в итоге значение i не станет равным 10. Является ли 10 меньше 10? Нет. Считая от 0, мы прошли 10 итераций цикла. Мы повторили 10 раз, так же, как мы делали это в блоке «Скретч». И так это в основном процесс для четырех циклов.

Теперь убираем реальный код и помещаем его в некоторые основные общие утверждения. Вот что происходит. Все операторы в начале выполняются первыми. Вы можете иметь более одного. Затем проверяется логическое выражение. Если выражение имеет значение true, выполнить тело цикла один раз. Если выражение оценивается как ложное, мы закончили. Мы не выполняем тело цикла вообще. После того, как мы выполнили тело цикла один раз, мы делаем то, что находится в инкрементной части. Что обычно будет что-то вроде i plus, plus или что-то в этом роде, которое модифицирует переменную counter. Затем, после увеличения, мы снова проверяем выражение и повторяем эти шаги снова и снова, пока выражение не перестанет быть истинным.

Итак, каковы варианты использования для цикла? Используйте цикл while, когда хотите, чтобы цикл повторялся неизвестное число раз, но, возможно, он может выполняться не все. Действительно распространенным примером использования цикла while является запуск потока управления для игры. Вы можете не знать, как долго будет играть пользователь, но вы хотите продолжать делать то же самое. Обновление позиций различных спрайтов на доске. Вы хотите, чтобы все происходило постоянно, но вы не знаете, когда пользователь прекратит играть или у него закончится жизнь. Так что цикл while действительно хорош для чего-то подобного. Это хороший вариант использования.

Для цикла do while это очень похоже. Вы хотите, чтобы цикл повторялся неизвестное количество раз, но хотя бы один раз. Вы можете использовать это и для игры, но очень распространенным вариантом использования является запрос пользователя для ввода. Как правило, вы можете сказать что-то вроде: дайте мне положительное целое число, и вы можете поместить это в цикл do while. Так что он всегда будет спрашивать их хотя бы один раз для положительного целого числа. Если они дают вам положительное целое число, вы можете выйти из цикла. Вы могли бы быть сделаны. Булево выражение в do while может быть ложным.

И ты не будешь спрашивать их снова. Если вам дают отрицательное число, когда они набирают какое-то слово или что-то, что вам не очень полезно, вы можете использовать цикл do while, чтобы вернуться назад и делать это снова, снова и снова. Вы определенно хотите попросить их дать вам номер один раз, но вы не знаете, сколько раз после этого они могут быть вредителями. И так, сделайте, пока это отличный вариант использования для запроса пользователя для ввода. А в случае использования циклов for обычно вы хотите повторить цикл дискретное число раз, хотя вы можете не знать, сколько раз в момент компиляции программы.

Например, может быть, у вас есть программа, в которой вы запрашиваете у пользователя номер. И они вводят 100. И тогда ваш цикл for будет в этом случае работать 100 раз. Или, может быть, они вводят 1000 в вашей программе, а затем запускают цикл 1000 раз. Это определенное количество раз. Они не обязательно знают, что это за число в момент компиляции программы. Это не похоже на цикл while, где он может быть бесконечным. Это скорее число, которое вы просто не знаете. Теперь, хотя я обрисовал в общих чертах все эти варианты использования, вы должны знать, что практически в любых обстоятельствах вы можете поменять местами все три из этих циклов. Вы можете использовать цикл for, где бы вы использовали цикл while. Вы можете использовать цикл for везде, где будете использовать цикл do while и так далее. Иногда это может быть немного сложно, так что обычно Хорошая практика придерживаться типичного варианта использования. Используйте цикл do while, если хотите что-то один раз, по крайней мере. Используйте цикл for, если вы хотите сделать что-то определенное количество раз. В конце концов, именно поэтому есть три различных типа петель.

Таким образом, они могут быть использованы в правильном контексте. Но вы можете поменять их местами, если хотите. Если вам удобнее использовать циклы while и вы всегда хотите использовать циклы while. Это то, что вы можете сделать, хотя это может быть немного сложнее синтаксически, где цикл for будет намного проще.

Меня зовут Дуг Ллойд, и это CS50.

DOUG LLOYD: Привет, давайте поговорим об операторах в C. Итак, мы уже видели один, фактически равный оператору присваивания. Это позволяет нам просто поместить значение в переменную. Это оператор присваивания, один знак равенства. Чтобы манипулировать значениями и переменными в C и работать с ними, в нашем распоряжении имеется ряд операторов, которые мы можем использовать. Давайте посмотрим на некоторые из них, начиная с арифметических операторов. Как и следовало ожидать, мы можем выполнять довольно простые математические операции на языке C. Мы можем складывать, вычитать, умножать и делить числа, используя плюсы, минусы, звездочки и косые черты соответственно. Вот пара строк кода, в которых мы это делаем. Итак, у нас есть int x равно y плюс 1. Предположим, что где-то выше этой строки кода, как мы сказали, int y равно 10. Что? Значение х после того, как я выполню эту первую строку кода? Вы сказали 11? Ты был бы прав. Это почему? Ну, у было 10. Некоторые, я говорю, что int x равно 10 плюс 1. 10 плюс 1 равно 11. Таким образом, значение 11 сохраняется в переменной x. Не так уж плохо, правда?

Как насчет этой следующей строки кода? х равно х раз 5. Ну, прежде чем мы выполнили эту строку кода, х было 11. Итак, каково значение х после этой строки кода? Возьми секунду. Итак, х равно х раз 5. х было 11. Значит, х равно 11 раз 5. Или 55. Так что, если бы вы сказали 55, вы были бы правы.

Теперь это может немного сбить с толку, но с тем, как назначение работает в C, значение справа присваивается значению слева. Итак, сначала мы оцениваем x раз 5. Итак, 11 раз 5 - это 55. И затем мы сохраняем это значение в x. 11, которые были там раньше, теперь перезаписаны. Таким образом, значение x теперь 55. Надеюсь, это довольно просто.

Есть еще один оператор, которого вы, вероятно, не обязательно слышали, называли это, но вы наверняка работали с ним в прошлом, если вы помните свои дни долгого разделения в начальной школе. Это называется оператором модуля. Что модуль делает, это дает вам остаток, когда вы делите два числа вместе. Итак, если я скажу 13, деленное на 4, что остаток? И это значение будет рассчитываться оператором модуля.

Итак, у меня есть строка кода, int m равно 13 mod 4. И я говорю здесь в комментарии, что значение m теперь равно 1. Почему я так говорю? Хорошо, сделай длинное разделение в своей голове, если ты потерпишь меня на секунду. Итак, у меня есть 4, деленное на 13. 4 переходит в 13 три раза с остатком 1. Таким образом, в основном все, что делает оператор модуля, это говорит вам, что когда вы делите, вы получаете остаток. Вы можете подумать, что это на самом деле не очень полезная вещь, но на самом деле вы удивитесь тому, как часто этот оператор модуля может пригодиться.

Есть пара проблем, которые мы сделаем с CS50, которые занимаются этим. Это также хорошо для таких вещей, как случайное число. Так, например, если вы когда-либо слышали о генераторе случайных чисел, это даст вам число от 0 до огромного числа. Но, возможно, вам действительно нужно только число от 0 до 20. Если вы используете оператор модуля для того гигантского числа, которое генерируется генератором случайных чисел, вы берете любое огромное значение, делите его на 20 и получить остаток. Остаток может быть значением от 0 до 19. Таким образом, вы используете оператор модуля, чтобы взять это огромное число и свести его к чему-то более значимому. Я уверен, что вы сможете использовать оба из них в будущем в CS50.

Таким образом, C также дает нам способ применить арифметический оператор к одной переменной более кратким способом. Итак, на предыдущем слайде мы увидели, что х равен х раз 5. Это сработало. х раз 5 затем сохраняется обратно в х. Есть более короткий способ сделать это, подумал, и это синтаксис x, равный 5. Это то же самое, что сказать, что x равен x, умноженному на 5. Это просто немного более короткий способ сделать это. И когда вы видите какой-то код дистрибутива или какой-то пример кода, который делает подобные вещи, просто ознакомьтесь с тем, что означает синтаксис. Вы, конечно, не должны использовать его, но если вы это сделаете, это может сделать ваш код немного более гладким. И знайте, что вы также можете использовать любой из различных операторов, которые мы уже видели раньше, а не время. Можно сказать, что х плюс равно 5, минус равно 5, раз, делить и мод. Все эти работы.

Есть также кое-что, что настолько распространено в C, что мы решили уточнить это еще дальше. Увеличение переменной на 1 или уменьшение переменной на 1 является настолько распространенной вещью - особенно когда мы немного позже поговорим о циклах - что мы решили вместо того, чтобы говорить что-то вроде x плюс равно 1, или x равно x плюс 1, мы даже коротко вручили это х плюс плюс. Итак, х равно х плюс 1, х плюс равно 1, а х плюс плюс все делают одно и то же. Все они увеличивают x на 1. Но это увеличение и уменьшение на 1 настолько распространено, что у нас есть плюс плюс и минус, что позволяет нам сокращать это еще дальше.

Итак, давайте переключимся на секунду и поговорим о булевых выражениях. Все это также относится к общей категории операторов. Но булевы выражения, в отличие от арифметических операторов, используются для сравнения значений. Итак, опять же, все булевы выражения в C имеют одно из двух возможных значений, напомним. Правда или ложь. Это единственные два значения, которые может принимать булева переменная. Мы можем использовать результаты логического выражения разными способами в программировании. На самом деле, вы будете делать это довольно много.

Например, мы могли бы решить, хорошо, если какое-то условие выполнено, возможно, я перейду эту ветку к своему коду. Условно, так сказать. Мы тоже скоро узнаем об этом. Или, может быть, пока это правда, я хочу продолжать делать это снова и снова и снова. Цикл. В обоих случаях знайте, что мы используем логическое выражение, истинное или ложное, чтобы решить, следует ли выбрать конкретный путь. Иногда, когда мы работаем с логическими выражениями, мы будем использовать переменные типа Bool. Возможно, вы объявили переменную типа Bool, и вы будете использовать в своем логическом выражении. Но вы не всегда должны делать. Как выясняется, в C каждое значение, отличное от 0, равнозначно произнесению true. Если вы объявили переменную типа Boolean и присвоили ей значение true, это равносильно объявлению целого числа и присвоению ему значения 1, 2, 3, или действительно любое значение, отличное от 0. Потому что в C каждое значение, отличное от 0, является истинным. 0, с другой стороны, ложно. Это может пригодиться позже, чтобы знать, но просто кое-что, чтобы иметь в виду. Нам не всегда нужно использовать переменные логического типа, когда мы работаем с логическими выражениями.

Есть два основных типа логических выражений, с которыми мы будем работать. Логические операторы и реляционные операторы. Язык там не очень важен. Это действительно просто, как я их группирую. И я думаю, что вы наверняка быстро поймете, что такое реляционный оператор, основываясь на том, чем он является, когда мы поговорим о них через секунду. Но не беспокойтесь о необходимости запоминания термина логический оператор или оператор отношений. Я просто использую это, чтобы сгруппировать их логически.

Итак, давайте посмотрим на три логических оператора, которые мы довольно много увидим в программировании на CS50 и в программировании в целом. Логическое И истинно, если и только если оба операнда истинны. В противном случае ложь. Где это значит? Итак, допустим, что я нахожусь в точке своего кода, где у меня есть две переменные, x и y. И я хочу решить, делать ли что-то в моем коде, основываясь на том, что x истинно, а y истинно. Я хочу сделать это, только если они оба верны, иначе я не хочу идти по этому пути, потому что это не поможет мне. Что я могу сказать, так это если x & & y. Это будет логическое логическое выражение, сравнивающее x и y и выбирающее определенный путь в зависимости от их значений. Итак, если x истинно, а y истинно на основании этой таблицы истинности, только тогда мы пойдем по этому пути. Если x, & & y. Это' s only true - и true только если x - true, а y - true. Если один из них является ложным, как мы видим в таблице истинности, то и x, и y не соответствуют действительности. Итак, x & & y ложно.

Логическое ИЛИ истинно, если и только если хотя бы один операнд истинен. В противном случае ложь. Так что логическое И требовало, чтобы и x, и y были правдой. Логическое ИЛИ требует, чтобы x было истинным, или y было истинным, или оба x и y были истинными. Итак, опять же, мы как будто попадаем в ситуацию, когда мы идем к нашему коду, и мы достигли развилки на дороге. И мы хотим пойти по определенному пути, если x истинно или y истинно, но не обязательно, если оба истинны. Но возможно, если оба верны. Так что, если x истинно, а y верно, мы пойдем по этому пути. х это правда. Один из них правда, верно? Если х верно, а у верно. Если x истинно, а y ложно, одно из них все еще верно. Таким образом, х или у все еще верно. Если x ложно, а y истинно, то одно из них все еще верно, верно? у верно, в этом случае. Таким образом, верно, что x или y верно. Только если x ложно, а y ложно, мы не пойдем по этому пути, потому что ни x, ни y не верны. Теперь, если вы смотрите на экран прямо сейчас и задаетесь вопросом, что это за символ для логического ИЛИ, он называется вертикальной чертой. И если вы смотрите на клавиатуру в течение минуты, как я делаю сейчас, она обычно находится над клавишей Enter, на большинстве клавиатур, на той же клавише, что и обратная косая черта. Это также обычно прямо рядом с квадратными скобками. Таким образом, это может быть ключ, который вы не набирали слишком много в прошлом. Но, если вы когда-нибудь будете делать логические сравнения, как мы будем много делать на курсе, будет полезно найти этот ключ и использовать его. Таким образом, он обычно находится на том же ключе, что и обратная косая черта чуть выше Enter. Вы смотрите на экран прямо сейчас и задаетесь вопросом, что означает этот символ для логического ИЛИ, он называется вертикальной чертой. И если вы смотрите на клавиатуру в течение минуты, как я делаю сейчас, она обычно находится над клавишей Enter, на большинстве клавиатур, на той же клавише, что и обратная косая черта. Это также обычно прямо рядом с квадратными скобками. Таким образом, это может быть ключ, который вы не набирали слишком много в прошлом. Но, если вы когда-нибудь будете делать логические сравнения, как мы будем много делать на курсе, будет полезно найти этот ключ и использовать его. Таким образом, он обычно находится на том же ключе, что и обратная косая черта чуть выше Enter. Вы смотрите на экран прямо сейчас и задаетесь вопросом, что означает этот символ для логического ИЛИ, он называется вертикальной чертой. И если вы смотрите на клавиатуру в течение минуты, как я делаю сейчас, она обычно находится над клавишей Enter, на большинстве клавиатур, на той же клавише, что и обратная косая черта. Это также обычно прямо рядом с квадратными скобками. Таким образом, это может быть ключ, который вы не набирали слишком много в прошлом. Но, если вы когда-нибудь будете делать логические сравнения, как мы будем много делать на курсе, будет полезно найти этот ключ и использовать его. Таким образом, он обычно находится на том же ключе, что и обратная косая черта чуть выше Enter. на том же ключе, что и обратный слеш. Это также обычно прямо рядом с квадратными скобками. Таким образом, это может быть ключ, который вы не набирали слишком много в прошлом. Но, если вы когда-нибудь будете делать логические сравнения, как мы будем много делать на курсе, будет полезно найти этот ключ и использовать его. Таким образом, он обычно находится на том же ключе, что и обратная косая черта чуть выше Enter. на том же ключе, что и обратный слеш. Это также обычно прямо рядом с квадратными скобками. Таким образом, это может быть ключ, который вы не набирали слишком много в прошлом. Но, если вы когда-нибудь будете делать логические сравнения, как мы будем много делать на курсе, будет полезно найти этот ключ и использовать его. Таким образом, он обычно находится на том же ключе, что и обратная косая черта чуть выше Enter.

Последний логический оператор - это НЕ. И НЕ довольно просто. Он инвертирует значение своего операнда. Если х истинно, то не х ложно. Если х ложно, то не х истина. Иногда вы будете слышать этот символ, произносимый как взрыв или восклицание или нет. Это в значительной степени все то же самое. Если вы слышите эти слова и не знаете, что это значит, это просто восклицательный знак, но иногда это называется парой разных вещей. Хорошо, так что позаботимся о логических операторах. Итак, давайте поговорим о реляционных операторах. Опять же, если вы знакомы с этой арифметикой еще в начальной школе, вы, вероятно, знакомы с тем, как они уже работают. Они ведут себя точно так, как вы ожидаете. Так что меньше, чем это правда, в этом примере, если х меньше, чем у. Таким образом, если х равен 4, а у 6, х меньше, чем у. Тот' это правда. Меньше или равно работает почти одинаково. Если x равен 4, а y равен 4, то x меньше или равен y. Лучше чем. х больше чем у. И больше или равно, х больше или равно у. Если это правда, тогда вы пройдете это выражение и пойдете по этому пути по дороге. Если у вас есть, если x больше, чем y, а x на самом деле больше, чем y, вы будете делать все, что подпадает под это условие.

Обратите внимание, что у нас нет ни одного символа меньше или равного, как вы, возможно, знаете из учебников по математике. Итак, у нас есть символ меньше, за которым следует знак равенства. Вот как мы представляем меньше или равно. И аналогичным образом, мы делаем это для больше или равно.

Последние два важных реляционных оператора - это проверка на равенство и неравенство. Таким образом, если x равно равно y, значение true, если значение x и y одинаково. Если x равен 10, а y равен 10, то x равно равно y, верно. Если x равен 10, а y равен 11, то x равно равно y не соответствует действительности. Мы также можем проверить неравенство, используя восклицательный знак или удар или НЕ, опять же. Если x не равен y, если это тест, который мы здесь используем, мы были бы хороши. Итак, если x не равно y, мы пойдем по этому пути.

Будьте очень осторожны здесь. Это действительно распространенная ошибка - и я, безусловно, совершил довольно много, когда только начинал - случайно ошибиться в операторе присваивания, равных единичным, для оператора сравнения на равенство, двойных равных. Это приведет к некоторому странному поведению в вашем коде, и обычно компилятор предупреждает вас об этом, когда вы пытаетесь скомпилировать свой код, но иногда вы можете подкрасться к нему. Это не обязательно хорошо, что вы крадете это, хотя. Точно так же, если вы выполняете тест на неравенство, если вы проверяете, имеют ли две разные переменные одинаковое значение внутри них, убедитесь, что используются равные равные, а не одиночные равные. И таким образом ваша программа будет иметь поведение, которое вы намерены. Я Даг Ллойд, а это CS50.