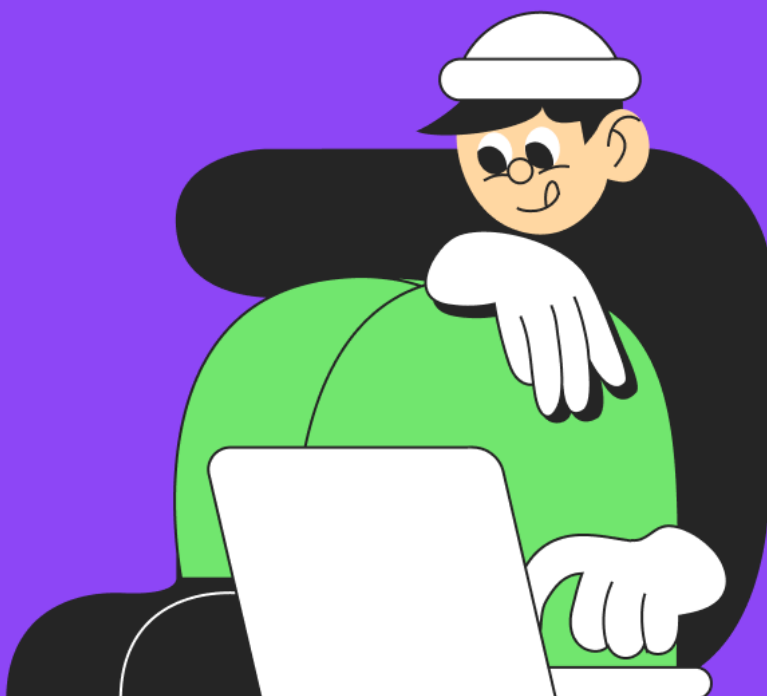
GeekBrains

Сложность и стоимость алгоритмов



Оглавление

[00:01:36]

Приветствие

Всем привет!

Рад вас видеть на третьей лекции курса «Введение в программирование» программы «Разработчик». Это заключительная лекция курса. Если кто-то по каким-то причинам первые две лекции пропустил или до сих пор не запомнил, я напомню, знакомый вам уже слайд. Я Ильнар Шафигуллин методолог программы «Разработчик» и сотрудник Казанского университета, математик и программист.

[00:02:05]

На этом уроке

Теперь перейдём к теме сегодняшней лекции. За первые две лекции мы:

- провели аналогии между обычными языками и языками программирования;
- рассматривали различные алгоритмы;
- коснулись структур данных.

Но вот задачка, казалось бы, такая же простая, как и те, что мы рассматривали. Но иногда её дают даже на собеседованиях для junior (начинающих программистов).

[00:02:31]

Найти второе максимальное число

Пока на эту точку мы не смотрим, как будто мы её не видим. Попробуем найти второе максимальное число из уже знакомых нам гирек и стаканчиков. Мы умеем находить самое большое число как среди пяти отдельных элементов, так и в массивах. Этот алгоритм мы уже делали. Каким-то способом мы смогли найти, что 8 — это самая большая гирька. Имеется дружок пирожок наша морковка и мы можем отметить, что вот этот элемент самый большой. Но по условиям текущей задачи необходимо найти не самое большое число, а второе по величине число 6. То есть нам нужно видоизменить алгоритм или написать новый, кому как удобнее, и сделать так, чтобы на выходе алгоритма находилось не самое большое число, а второе по величине.

Для этого есть несколько разных способов. Можете придумать собственный, и сейчас вам это будет нужно сделать. Но при этом стоит держать в уме, что может быть не 5 чисел, а 10, 100 или даже такое число с множеством нулей. Соответственно, сейчас нужно взять листочек и ручку или воспользоваться той программой, в которой вы составляете алгоритмы, как будет удобнее. И попробовать сначала решить задачку, например, для пяти гирек. Если сможете сразу для множества, попробуйте рассмотреть общий случай. Напомню, нам необходимо на выходе этого алгоритма найти второе по величине максимальное число.

Сразу отмечу задачка непростая. Если вдруг у вас будут какие-то сложности, это абсолютно нормально. Повторюсь, эту задачку иногда дают даже на собеседованиях для junior-программистов. Прежде чем я покажу вам решение, скажу по секрету, что их будет несколько. Вам необходимо самостоятельно подумать над этой задачей. Посмотреть, какие сложности возникают, когда вы пытаетесь найти второе по величине максимальное число. Потом мы разберём, как эти сложности можно решить. Но сначала необходимо познакомиться с этими сложностями.

Итак, выделим даже не 5, а хотя бы 7–8 минут. Вам нужно подумать, каким образом можно найти второе по величине максимальное число. И есть ли какие-то сложности. Например, вы не знаете, как пропустить какой-то элемент или ещё раз что-то сделать. Если какая-то сложность возникает, с которой вы столкнулись и вот если бы её решили, тогда бы задача решилась, но не знаете, как её решить. Можете написать об этом в комментариях. Будет интересно посмотреть, кто на чём споткнулся и остановился. Если же вы смогли составить алгоритм, то также будет здорово почитать в комментариях, как вы эту задачку решили. Как можно найти второе по величине максимальное число из уже представленных. Напомню, что самым большим будет здесь 8, а правильным ответом вот для этой задачи будет число 6. Итак, вам даётся 7 минут. Попробуйте решить эту задачу.

[00:12:58]

Рассмотрим алгоритм решения

Надеюсь, что вы уже смогли составить какие-то алгоритмы, найти трудности, которые вам мешают, чтобы найти второе по величине максимальное число или же вы решили эту задачу и есть у вас для этого готовый алгоритм.

Посмотрим, как в большинстве случаев рассуждают, пытаясь решить эту задачу. Можем сначала поискать первое наибольшее число. Мы умеем это делать. Потом исключить найденное число из массива, а затем из того, что осталось попробовать ещё раз найти самое большое. И если первое самое большое мы исключили, то есть нашли на первом шаге и исключили на втором. То когда мы заново, уже знакомый алгоритм, пропустим по оставшимся элементам, мы найдём второе по величине максимальное число. Например, у нас уже есть несколько алгоритмов для поиска самого большого максимального числа. Отметим его нашей морковкой. Этот элемент самый большой. Теперь когда мы отметили его и заново будем пропускать алгоритм, тогда сможем убрать это число из выбора максимального. И из оставшихся самым большим сейчас является 6. И теперь мы сможем найти 2 по величине максимальный элемент. То есть прогнав этот элемент два раза и забыв, исключив этот элемент в 1 проходе, мы сможем всё это сделать.

Теперь вопрос технического характера. А каким образом можно забыть этот элемент? Как во втором проходе мы можем перестать учитывать его в алгоритме? Есть несколько способов. Один из способов — поиск индекса максимального элемента. Если раньше, решая задачу, мы находили величину (массу самой тяжёлой гири), то сейчас, чтобы пропустить этот элемент, нужно найти его индекс. Тогда, прогоняя алгоритм ещё раз, мы сможем каждый раз проверять, совпадает ли с индексом элемент, который сейчас смотрим, с индексом, который надо пропустить, или нет. Если он совпадает с индексом, который необходимо пропустить, например, в этом случае равен одному, то мы этот элемент не должны учитывать в сравнениях.

Нужно пройти дальше. Таким образом, если найти индекс самого большого элемента (его позицию в массиве), то на втором прогоне можно будет исключить его из соревнования.

Как можно это сделать? Посмотрим на блок-схему, демонстрирующую этот алгоритм. Мы можем задать наш массив. Зададим некоторый размер массива и появятся две новые переменные. Current index (с английского — текущий индекс), раньше мы называли его просто индекс и max number index (индекс максимального элемента). Если раньше был просто max (максимальное значение), то сейчас уже в названии переменной указано, что это индекс максимального элемента. Изначально мы считаем, что самый большой элемент это первый, так же как мы всегда и делали. Дальше в качестве max мы указываем самый первый элемент, то есть элемент с индексом 0.

Обратите внимание. Раньше мы писали переменные в другом ключе. До этого был так называемый camel case, когда мы слепляли слова и вместо разделителей просто начинали каждое новое слово с большой буквы. Сейчас же используем подчёркивание как разделитель между словами. Этот стиль именования переменных называется snake case.

Итак, мы ввели переменные. Получили массив с размером, текущий индекс и индекс максимального элемента. Начинаем искать максимальный элемент. Алгоритм очень похож на то, что делали раньше. Но какие есть отличия? Мы точно так же сравниваем элемент с максимальным, и если элемент больше максимального, то фиксируем не только его величину (значение). Вот у нас max равен numbers и current index. Это тот же индекс, который рассматривали ранее. Но теперь у нас два разных индекса. Есть текущий индекс и индекс максимального элемента, и чтобы они явно отличались, мы просто назвали их немножко по-другому. Так вот, мы зафиксировали максимальный элемент, если вдруг он стал другим и на какой позиции он был найден. Элемент, который мы взяли, его индекс, записываем в переменную max number index. А дальше всё совершенно то же самое. Как только мы выйдем из нашего массива, то есть пройдём через все элементы, выводим на экран индекс максимального элемента. Таким образом, немного видоизменив блок-схему на выходе из алгоритма, мы находим не значение максимального элемента, а находим индекс (позицию), на которой он находится.

[00:18:51]

Запись алгоритма на псевдокоде

Пока посмотрите немного на блок-схему. Всё ли вам понятно? Отличие очень небольшие, но стоит обратить внимание на другие имена переменных и сам алгоритм делает немного другие вещи. Обратите внимание, мы всегда должны решать именно стоящую перед нами задачу. Если раньше нужно было найти значения максимального элемента, то индекс мы не находили, потому что это дополнительная память, которую мы расходуем. В этой задаче требуется не только найти величину максимального элемента, но и позицию, на которой он находится. Соответственно, именно эту задачу сейчас мы решаем.

Теперь, посмотрев на алгоритм, мы можем записать программу на псевдокоде или другом языке программирования. Так же, как раньше мы задаём входные значения. У нас есть некоторый массив и его размер. Мы вводим две переменные, которые сейчас добавили. Это текущий индекс, он равен 0, и индекс максимального элемента. Также мы первый элемент, который только взяли, рассматриваем как самый большой сейчас. То есть алгоритм никак не

меняется. И запускаем цикл. В этом цикле проверяем, появился ли элемент больше, чем считающийся сейчас самым большим. Если появился, то поменяем значение максимального элемента и запишем индекс текущего элемента, как индекс самого большого. Далее мы просто перекинем наш счётчик и перейдём к следующему элементу.

Напомню, это стандартный приём, как можно от одного элемента переходить к другому. А сама конструкция с циклом, внутри которого перескакивает счётчик, позволяет нам пробегать от одного элемента к другому. Вы очень часто будете видеть эту конструкцию, при написании или чтении программ. Но эта конструкция помогает нам перебрать все интересующие нас элементы. Итак, когда этот цикл закончится, у нас будет как величина максимального элемента, так и индекс, то есть позиция этого элемента в массиве.

В этой программе есть ошибка. Уверен, многие её уже нашли. Но прошу вас не просто указать на саму ошибку, а указать вид ошибки. Напомню, есть 3 вида ошибок, рассмотренных на прошлом занятии, синтаксические ошибки, ошибки выполнения и логическая ошибка. От вас в комментариях требуется написать какая это ошибка и где она расположена. Внимательные студенты, я думаю, уже это сделали. Если вдруг не смогли сделать сразу, то вернитесь к процессу, после окончания лекции.

[00:22:10]

Именование переменных

Чтобы всё хорошо запомнилось. Посмотрим на 2 основных способа (стиля) именования переменных. Snake case и camel case. Camel case или верблюжий стиль. Почему он так называется? Программисты увидели, что если буквы расположены в ряд и часть из них заглавные, тогда это похоже на горбики верблюда, поэтому назвали camel case. А когда вместо разделителей используется знак нижнего подчёркивания, то программисты решили, что это похоже на ползущую змею и назвали стиль snake case. Чтобы было проще запомнить здесь есть две симпатичные картинки, которые не позволят забыть, что есть 2 стиля именования переменных.

При этом заголовок слайда «Самое сложное в программировании — именование переменных» на самом деле относится не к стилям, а к значениям, хранящимся в переменных, которые должны быть отражены в их названии. Далеко не всегда это легко сделать, но очень важно. Если вы назовёте переменные, так чтобы читатель смог понять, что вы хотите от них, то работать с вашими программами будет гораздо проще.

Обратите внимание, возможно, сейчас вам кажется, что пишете и работаете с программой только вы. Но на самом деле есть статистика, показывающая, что программы читаются во гораздо чаще, чем пишутся, особенно если вы работаете в большом коллективе. Например, вы написали какой-то свой блок кода, а потом ушли в отпуск и кто-то другой дорабатывает вашу программу. Он будет читать и пытаться разбираться, а что такого вы хотели в этой программе сделать. И если программа легко читается, то вам гораздо проще будет и работать в коллективе и читать чужие программы. Потому что если другой человек называет свои переменные верно, смысл того, что должно храниться в этих переменах, отражается в их названии, то программа превращается в удобно читаемый текст и вам гораздо проще с ней разобраться. Если же переменные именуются странными символами, сочетанием символов, то разобраться в такой

программе становится очень сложно. Например, A1, а потом оказалось, что A1 уже занят, кто-то решил, что надо A2 добавить, потом A3 появилось и смысл никакой в этом не передаётся. Поэтому считается, что самое сложное в программирование — это именование переменных. Конечно, это в некотором роде шутка, но всё-таки это очень важная часть при написании программ.

[00:25:08]

Разберём всю программу

Теперь мы посмотрим на всю программу целиком. Разобрали алгоритм, пока на словах. Сказали, что сначала найдём самый большой элемент. После этого запомним его индекс. И когда мы ещё раз будем пробегать этот же массив, то просто пропустим этот элемент. И чтобы это продемонстрировать, напишем такого вида программу. Она выглядит очень странно и очень страшно. Наверное, здесь очень много строк кода и непонятных английских слов, но сейчас постепенно со всем разберёмся. Чтобы было удобнее, ведь не просто в уме программировать, мы справа на слайде добавили значения всех переменных. Смотрите, у нас жёлтая стрелочка находится на пятой строке. Это значит, что первые пять строчек в нашей программе уже будто бы выполнены. И по результату этих пяти строчек у нас есть несколько переменных и значения этих переменных. Имеется массив numbers, хранящий числа 1, 8, 3, 2 и 6, мы задали их в самой первой строке. Есть переменная size = 5. Переменная current_index = 0, она появилась в 3 строке. Есть переменная max_namber_index = 0, она появилась в следующей строке. И переменная max = 1, потому что в пятой строке мы в переменную max записали нулевой элемент. То есть элемент с индексом 0 из массива numbers, а это 1. Когда эти пять строчек программы будут выполнены, получим вот такие переменные и их значения. Собственно, вы видите на экране.

Теперь попробуем представить, что, написанное здесь, будет выполнять не компьютер, а мы. Тем более код написан на псевдокоде и компьютер с ним не справиться. Если немного перешагнём и представим, что мы перешли с пятой строки на седьмую, то есть выполнили шестую. Что нам даст строка 6? Мы сравнивали current_index и size. Current_index = 0, size = 5. Очевидно, что это условие выполнится, и мы провалимся внутрь цикла. Тогда мы сможем посмотреть, чему равны разные переменные. Нам необходимо ответить на вопрос numbers current_index больше максимального или нет. Здесь появилась строчка numbers current_index, мы знаем current_index = 0, поэтому numbers с индексом 0 равно 1. Собственно, что здесь и записано.

Теперь нужно ответить на вопрос: «А единичка, которая здесь хранится больше максимального или нет?». В максимальном вы можете видеть справа, тоже хранится единичка. Ответ будет отрицательным, потому что 1>1 неправильно. 1=1. Поэтому перескакиваем. Всё, находящиеся внутри, конкретно в этот раз, выполняться не будет и мы переходим на десятую строку.

Теперь current_index = current_index + 1. Он увеличился был 0, стал 1. Переменные, изменившие своё значение, подкрашены жёлтеньким цветом. Current_index стал равен 1, в таком случае numbers current_index это элемент с индексом 1. То есть число 8.

Переходим дальше. Ответим на вопрос: «Проваливаемся мы в цикл или нет?». Выполняем ли условие, позволяющие продолжать цикл или его пора заканчивать. Убеждаемся, что

current_index = 1, меньше, чем size = 5. Поэтому будем выполнять то, что происходит в цикле. Когда пройдём через 7, 8 и 9 строки у нас будет значение переменных, которое видите на экране.

Но почему мы провалимся внутрь условия if? Посмотрим. Numbers current_index = 8, а max = 1, когда мы только сюда проваливались. Именно поэтому 8 > 1, и мы попадаем на восьмую строчку. То есть условия выполнены и нужно перейти к 8 и 9 строке. В восьмой строке нужно в max записать numbers current_index. Numbers current_index = 8, поэтому в переменный max, можно увидеть жёлтым цветом, находится новое число 8. Max current_index также поменялся. Он теперь равен current_index, следовательно, max numbers_index = 1. Тоже подсвечено жёлтым цветом.

Надеюсь, что такой подход к анализу кода вам более или менее понятен. Можете прокрутить цикл до самого конца и ответить на вопрос: «Что будет в 11 строке, когда мы в current_index записываем 0?».

Теперь переместимся на эту строчку и посмотрим, что произойдёт с нашими переменными. В current_index мы записываем 0. Когда мы выходили из цикла, значение current_index было равно 5, если было бы меньше 5, то из цикла выйти нельзя. То есть выйти можно только тогда, когда в current_index будет записано 5. Но в 11 строке мы его затёрли и в current_index записали 0.

В max number index будет индекс самого большого элемента. Самый большой элемент 8, а его индекс равен 1. Поэтому значения будут такими, как указаны у нас на экране.

Если перейдём дальше, то появляется следующая переменная. В max будем записывать самый большой элемент, а в second max запишем второй самый большой элемент. Здесь я записываю в него снова первый элемент, то есть элемент с индексом 0 из нашего массива. Также, как это делали в 5 строке. То есть на текущий момент я считаю, что вот этот элемент является самым большим вторым по величине в этом случае.

Далее к строкам 13 и 14 мы ещё вернёмся. И в самом конце, когда этот код выполнится, у вас должно получиться следующее. Вам на это, мы выделим 8 минут. В течение 8 минут вам нужно прогнать эту программу до самого конца и ответить на 2 вопроса. Первый вопрос: «Для чего нам нужны 13 и 14 строчки?» Что случится, если я уберу эти строчки? Что будет в результате? Нужно ли их убирать и являются ли они лишними? Что действительно в таком виде результат будет идентичен, показанному на экране? То есть второй по величине максимальный элемент будет равен 6. Как сделаете, напишите, пожалуйста, об этом в комментариях. Выделим на это 8 минут.

[00:40:39]

Недостаток алгоритма

Ну что же время у нас истекло. Надеюсь, у вас получилось пройтись по этому коду и разобраться, что в нём происходит, какие значения принимают переменные. Это очень полезное упражнение. Рекомендую делать его не только с программами, разбираемыми на лекции, но и с любыми встречающимися. Если вы находите их в интернете или в какой-то книжке о них читаете, попробуйте не просто в уме рассчитывать происходящие, а выписывать

значения переменных, меняющихся в программе. Это позволит лучше понимать, как работает алгоритм.

У нас алгоритм работающий. Он находит второе по величине максимальное число, но есть большой недостаток. Предположим, что у нас не 5, а 5 тыс. элементов. Чтобы найти первый максимальный элемент, мы просто один раз посмотрели все элементы и за 5 тыс. сравнений смогли найти самый большой элемент. Если же надо найти второй элемент, и мы сначала пробежались по всем элементам один раз, а потом ещё раз пробежались, чтоб найти второй по величине максимальный элемент, то мы делаем много дополнительной работы. То есть необходимо два раза пробежать эту дистанцию. Недостаточно сделать это за один проход. А если у нас не 5 тыс., а 5 миллионов элементов? Когда мы решаем реальные задачи с множеством элементов, например, связанные с базами данных, какими-то сайтами или реестрами, каждый дополнительный проход может занимать дополнительное время. Время очень дорого как по электричеству, так и по другим ресурсам. Помимо этого, если сайт долго обрабатывает информацию или программа медленно считает, то это плохая программа. Никому не понравится ею пользоваться. Поэтому недопустимая роскошь решать задачу не рациональным путём.

[00:42:41]

Математический юмор

И вот то, как мы нашли решение этой программы, похоже на один смешной для математиков, но не очень смешной для всех остальных анекдот. Для математиков и физиков есть такое сравнение. Предположим, что дана задача вскипятить чайник. И у вас есть ёмкость с водой, е чайник и газовая плита. Как эту задачу будет решать физик? Он наберёт в чайник воду, поставит на плиту, включит плиту, дождётся, пока чайник закипит и выключит плиту. Всё задача решена. Как задачу решит математик? Те же самые исходные данные. Математик набирает воду в чайник, ставят на плиту, включает плиту, дожидается, когда чайник закипит, выключает. Задача решена. То есть абсолютно одинаковый путь к решению.

Теперь меняем водные. Предположим, что в чайник уже налита вода. Что делает физик? Смотрит, на чайник. Видит, что в него уже налита вода. Ставит его на плиту, включает её. Ждёт, пока чайник закипит. И выключает плиту. Решил на один шаг короче. Что делает математик? Увидел чайник с водой. Выливает воду из чайника и сводит решение к уже решённой задаче. То есть говорит вот такую задачу без чайника мы решить можем. Смотри предыдущее решение.

Для математиков это абсолютно нормально. Наша задача сделать так, чтобы задача была решена и нет большого смысла пытаться решить её оптимально. Если теорема доказана, то она доказана, а уж сколько строчек заняло доказательства, никого не интересует. Если мы говорим про реальный мир про физику или программирование, в которых каждое лишнее действие совершаемое нами это дополнительные затраты времени. Или говорим про дополнительную работу исполнителей или сотрудников, это лишняя заработная плата, которую необходимо заплатить. Себестоимость продукта может резко вырасти просто потому, что вы не рационально, что-то делаете. Если говорить про программирование, то каждое дополнительное действие — это некоторое время и усилия, которые должен совершить компьютер для решения задачи. Если усилий становится слишком много, то задача решается

медленнее и это никому не нравится. С точки зрения математиков этот подход правильный, но с точки зрения, реального мира и реального применения, этот подход недопустим.

Мы решили задачу именно как математики. Ну я вас к этому склонил. Сначала мы воспользовались уже решённой задачей, потом внесли небольшие изменения и ещё раз это же самое решение применили. То есть, по сути, решив задачу один раз, мы вылили воду из чайника и решили задачу ещё раз. Это дополнительные усилия, которые нам не нужны.

[00:45:27]

Костыли в программировании

Как можно решить эту задачу по-другому? Ну вот здесь вот опять от Ксюши у нас небольшая картинка. Когда мы сводим, уже решённую ранее, задачу, то можем наткнуться на так называемые костыли. Придумывая своё решение, кажущиеся изначально простыми, но по мере углубления, пытаемся каким-то образом подстроить нашу программу под решение, придуманное нами, приходится использовать ухищрения. Часто в программировании их называют костылями. Использование костылей, как говорят, попытки натянуть сову на глобус. Когда к понравившемуся вам решению нужно подстроить всю задачу, хотя ваше решение из совершенно другой, это приводит к добавлению себе дополнительных сложностей.

[00:46:17]

Три вида сложностей

Сложности существуют несколько видов:

- необходимая сложность;
- необязательная сложность;
- случайная сложность.

Что такое необходимая сложность? Это минимальная сложность, без которой задачу решить нельзя. Например, если необходимо завязать шнурки, то необходимая сложность — это умение завязывать узелок и бантик сверху. Не умея этого, задачу «завязать шнурки» не решить. То есть это необходимая сложность, без которой задача никак не может быть решена.

Есть необязательная сложность. Это если помимо решения основной задачи, вы ещё что-то дополнительное хотите сделать. Например, если мы захотим завязывать шнурки и успеть ещё через скакалку попрыгать.

Если говорить про задачи, которые мы решали, то вспомните, пожалуйста, первую лекцию. И самое первое решение задачи на поиск максимального элемента. Там мы переставляли гирьки. Нас просили найти только самый максимальный элемент. Мы его нашли, но при этом делали какие-то дополнительные вещи, о которых у нас никто не просил. Перемещали элементы так, чтобы у нас самый большой элемент оказался с краю. Это необязательная сложность. Мы для себя придумали дополнительную сложность, не требующуюся в задаче. Также и здесь это может быть со шнурками или с любыми другими задачами, которые есть.

Помимо этого, есть ещё случайная сложность, связанная с выбором вами решения. Вы почему-то решили делать именно так. Да вы задачу, наверное, решите, но сложности вы явно в неё добавили.

[00:48:00]

Примеры сложностей

Например, если вы захотите завязать шнурки в варежках. Вы добавили сложности, ведь сделать это без варежек гораздо легче. Но вы сами это выбрали. Обратите внимание, что любую сложность всегда можно разложить на три составляющие (необходимая, необязательная и случайная сложность).

Пока мы не перешли к другому решению нашей задачи с поиском второго максимального элемента, вам ещё одно маленькое задание. Буквально за 3 минутки вам нужно придумать пример, как сейчас со шнурками, скакалкой и варежками, желательно из более или менее реальной области, где задача может быть решена с необходимой сложностью, то есть сделано именно то, что нужно и с минимальными усилиями. С необязательной сложностью, когда вы всё, что нужно сделали, но при этом какую-то дополнительную работу совершили, которой от вас не требовалось. И случайная сложность, когда вы просто выбрали, какое-то не оптимальное решение, и оно к чему-то привело. Это не обязательно ваш личный опыт. Просто какой-то пример, где сложность раскладывается на три составные части. Это весьма интересное упражнение, когда вы будете смотреть на работу и оценивать эту работу с точки зрения необходимой, не обязательной и случайной сложности. Вот буквально 3 минутки жду ваши версии в комментариях.

[00:52:33]

Более эффективное решение задачи

Теперь перейдём к решению нашей программы и постараемся ограничиться необходимой сложностью, не забегая на необязательную и на случайные сложности.

Попробуем упростить нашу задачу. Этот подход к решению задачи мы подсмотрим у математиков. Математики очень часто так делают. Если у вас есть какая-то сложная задача, попробуйте сначала посмотреть простой случай. Так вот, когда у нас есть большой массив, из которого необходимо найти второе по величине максимальное число, мы можем взять самый простой случай. А самый простой случай, когда всего есть два элемента. Возьмём массив из 2 чисел и попробуем найти второе максимальное по величине число. Решается очень просто. Псевдокод у вас на экране. Что мы делаем? Мы вводим две перемены first и second. First — первая максимальная и second — вторая по величине. И если первый элемент, то есть элемент с индексом 0, больше элемента с индексом 1. Тогда, очевидно, самым большим из двух будет элемент с индексом 0, потому что условие так выполнилось. А вторым по величине будет оставшийся. Если это условие не выполнится, то появляется новое ключевое слово else, с которым мы пока не встречались. Сделаем, наоборот, самым большим объявим элемент с индексом 1, то есть второй элемент, потому что условие не выполнилось, а вторым по величине обозначим с индексом 0.

Теперь разберёмся с 4 строчкой. Как можно её прочитать с точки зрения программирования или английского языка. Мы говорим, если это условие выполнится, тогда делай 2 и 3 строчку. С этим мы знакомы — добавляется это.

Что получится: если это условие выполнится, тогда делай это иначе (else переводится иначе) делай вот это. То есть, если условие выполняется, я буду делать 2 и 3 строки. Если условия не выполняются, тогда буду делать 5 и 6 строки.

Таким образом, буквально в несколько строк мы можем решить задачу с поиском второго по величине максимального элемента, если у нас всего лишь два этих элемента есть. Всё просто.

Теперь идём по нарастающей. Представим, что у нас не 2 элемента, а 3 элемента. Что будет в этом случае. В этом случае я снова завожу first и second. Изначально возьму, что они оба равны первому элементу. Теперь, получившиеся, я сравню со вторым элементом (с индексом 1). Что у меня получится. Если 1 элемент больше, чем first тогда я скажу, что самый большой элемент — это элемент с индексом 1. Иначе самым большим элементом станет нулевой, как я его и обозначил, а second станет вторым по величине.

Чтобы нам было удобнее, я ещё раз верну наши стаканчики. Мне достаточно трёх стаканчиков, потому что задача у нас сейчас на 3 элемента. Мы их расставим в произвольном порядке и посмотрим, на происходящие в нашей блок-схеме. Мы говорим, что у нас есть 2 переменные first и second. Они обе равны первому элементу нашего массива. То есть элементу с индексом 0. First и second у меня равны, теперь 1. Дальше если следующий элемент, то есть элемент с индексом 1 больше, чем first, а first = 1. First буду обозначать нашей морковкой. Если этот элемент больше, чем first это действительно так, то мы идём на 4 строчку, потому что условие выполнилось. Теперь first = 8, а second = 1. И действительно, если было всего лишь два элемента, то действительно этот элемент был самым большим, а этот был вторым по величине. Пока всё правильно. Else пока нас не интересует, потому что условие выполнилось, и мы выполнили 4 строчку. Переходим на 7 строчку. Теперь добавляем в проверку следующий элемент. Сравниваем третий элемент с самым большим. Если третий элемент больше, чем first, а это не так, то условие не выполнилось и эти два действия я совершать не буду.

Перехожу на веточку else (иначе). Если третий элемент больше второго. Теперь мне необходимо третий сравнить с second. Second = 1. Третий элемент больше. Это условие выполнилось тогда у меня теперь 2 это numbers 3. Второй стал этим элементом.

Про это временно забудем. First и second здесь. Действительно, этот алгоритм для конкретного случая нашёл, что второй по величине элемент будет равен, трём.

Для чистоты эксперимента мы поменяем наши элементы местами и посмотрим, что же у нас произойдёт в другом случае. Опять начнём с самого начала наш алгоритм. First и second это у нас первый элемент. Теперь это элемент с массой 3, с индексом 0.

Идём на третью строчку. Numbers 1 больше, first? Проверим. Нет, условия не выполнены, поэтому пропускаю 4 строку и перехожу на 5 else — что случится иначе. Второй элемент numbers 1. Теперь у меня first это 3, а second это 1. Опять для двух элементов всё найдено верно. То есть второй по величине из этих двух это 1. Добавляем в рассмотрение третий элемент. Теперь сравниваю, а третий элемент больше, чем first или нет? Да, действительно третий элемент больше. Что нужно сделать? Нам необходимо в переменную second записать тот, что

сейчас является самым большим. Сейчас самым большим является first. В переменную second я записываю его. Теперь он переменная second. А first это самый большой, который пришёл новенький. И опять получаем, что у нас самый большой элемент — 8, а второй по величине элемент — 3. Можете по-разному расставить ваши гирьки, но этот алгоритм всегда должен давать правильный ответ. В переменной first у вас будет записан самый большой элемент, а в переменный second второй по величине.

Но опять же в этом коде, мы будем прокачивать в вас навыки тестировщиков, есть ошибка. Вам необходимо указать, какая это ошибка. Напишите об этом в комментариях. Хотя я уверен, что уже несколько комментариев на тему ошибок в этой программе под уроком появилось. Думаю, логика ясна. Мы сначала решили задачу для самого простого случая, взяли два элемента и решили. Добавили новый элемент и посмотрели, что же происходит.

Теперь мы можем посмотреть общий случай. Когда мы добавим следующий элемент. У нас происходит то же самое. Ещё следующий элемент опять происходит то же самое. То есть один и тот же алгоритм позволяет нам каждый раз сдвигать переменные first и second, чтобы они всегда были на месте самого большого и второго по величине элемента.

Учитывая, что алгоритм повторяется, нам достаточно добавить сюда некоторый цикл. И вот у нас есть код программы. И вам необходимо сейчас сделать такое же упражнение, как ранее. Оставим мы сейчас его на экране. Но у нас нет справа блока со значением этих переменных. Вам необходимо на листочке или в любом редакторе, которым пользуетесь, выписывать по мере выполнения программы значения всех переменных. В качестве примера можете взять тот же, который у нас был всегда — числа 1. 8, 3, 2 и 6. И попробовать этот массив пройти от начала до конца. В конце этой программы в 18 строке у вас print second должно вывести число 6. То есть в переменной second у вас должно быть число 6. Вам на это 10 минуток. Необходимо итеративно, как компьютер пройти всё, представленные здесь, шаги.

[01:12:16]

Принцип уточки

Очень полезное упражнение. Старайтесь его периодически делать. Я неоднократно это говорил и ещё буду повторять. Когда вы вместо компьютера начинаете шаг за шагом выполнять отдаваемые ему команды, то будете понимать гораздо лучше, как работает программа. И ещё лучше понимать, почему она не работает. У программистов на самом деле есть резиновые уточки. Может быть, даже вы их видели. И есть «принцип уточки», в Википедии на английском языке называется rubber duck debugging. Когда вы хотите, написать программу и не понимаете как это сделать или почему программа не работает. Берёте уточку, лежащую рядом с вашим компьютером, и начинает ей объяснять. Причём объясняйте максимально простым языком. Например, «Ну вот уточка, смотри, я взял массив, вот у него такой размер. Вот я ввёл 2 переменных, в first я буду хранить самые большие, в second хранить вторые по величине. Но учитывая, что я ничего ещё не видел в этом массиве, я в обе перемены запишу пока в первый элемент. Но вот 2 элемент я уже возьму и определю, кто из них большой, а кто из них маленький». И дальше по шагам медленно и аккуратненько начинаете объяснять так, чтобы даже это резиновая уточка смогла понять. И вы удивитесь, насколько много нового и интересного вы для себя откроете. Вам будет понятно, а как же ваша программа работает и почему она не работает или работает не так, как вы думали.

Резиновая уточка хороша, но вам придётся в памяти держать все значения переменных. Если этот путь не подходит, то упражнение, выполненное сейчас, всегда поможет справиться с любой задачей. Вы берёте листочек, выписываете значения всех переменных и аккуратненько, чтобы не запутаться, проходите от одного элемента к другому элементу.

Помимо этого, очень полезно попытаться объяснить другому человеку, который не очень хорошо знаком с программированием «а что же здесь на самом деле происходит?» Вы можете объяснить своему другу, коллеге, супругу, маме, бабушке, ребёнку, неважно кому. Главное, если вы сможете правильно объяснить происходящее в вашей блок-схеме или программе человеку, незнакомому с этой темой, то сами гораздо лучше будете понимать, что здесь происходит. Есть старая шутка про преподавателей:

Студенты пошли, такие глупые ничего не понимают. Я объясняю, объясняю, объясняю, объясняю, уже сам всё понял, а они никак понять не могут.

Ну вот это шутка она не так далека от истины. Даже я, будучи кандидатом физикоматематических наук, окончив с красным дипломом механико-математический факультет, когда пошёл преподаватель математический анализ на 1 курс. Причём на 1 курс на экономический факультет, я для себя открыл много нового. То есть я сдал все экзамены, написал диссертацию, защитил диссертацию, но, объясняя первокурсникам самые основы матанализа, я для себя открыл много нового. Оказывается, я не до конца правильно понимал то, что происходит. Не до конца чувствовал все тонкости, которые там должны были быть. Я и сейчас, скорее всего, не так хорошо всё это знаю, но объяснив это первокурсникам, стал гораздо лучше понимать, что на самом деле происходит в основах математики. Также и вы можете воспользоваться этим примером и аккуратненько, можно начать резиновой уточки, а можно со своих знакомых и попытаться объяснить ему, что здесь происходит. Вы гораздо лучше начнёте понимать, что же мы здесь делаем и почему эта программа работает именно так.

[01:16:08]

Следование. Ветвление. Цикл

Теперь интересный факт. Мы рассмотрели три вещи: следования, ветвление и цикл.

Что такое следование? Следование — это последовательное выполнение инструкций. Это то, что мы делали в самом начале. Мы брали первый элемент, потом второй элемент и так далее. То есть последовательно, какие-то действия выполняем. Помимо этого, мы сами брали ветвление. Это ромбики, из которых есть 2 стрелочки или условия IF. Мы сравниваем, например, берём 1, берём 2. Если 2 тяжелее, то мы что-то делаем. Если 1 тяжелее, то мы что-то другое делаем. То есть вот последовательность действий, которую мы выполняем, может пойти по какой-то ветке.

И начиная с конца 1 лекции, мы знакомимся с циклами, то есть с циклическими повторениями, каких-то действий при выполнении или невыполнении каких-то условий. Так вот, в теории вычислимости в программировании сказано, что любую программу совершенно любой сложности, можно записать как комбинацию этих трёх структур — следование, ветвление и циклов. Можете взять программу, управляющую атомной станцией или высчитывающую орбиту МКС, совершенно любые по сложности задачи и расписать в виде набора трёх структур. Поэтому, когда мы разбираемся с блок-схемами, мы практически можем сказать, что всё

программирование нам теперь известно. С тем, что мы знаем, можно решить совершенно любую программу.

Естественно, в теории это так, но на практике всё оказывается несколько сложнее, потому что, если мы возьмём какую-то очень сложную программу и попытаемся обойтись только лишь тремя структурами, наша программа станет очень, очень сложна. Она будет занимать много страниц, там будет много повторяющихся действий, много кода, который можно было записать аккуратнее. И в жизни мы так инструкции всё-таки не даём. Поэтому сейчас рассмотрим следующую вещь, позволяющую нам упростить написание программ. Но хочу отметить, что уже на этом этапе наших знаний достаточно, чтобы написать любую по сложности программу. Но говоря «наших знаний», я преувеличиваю, но вот инструментов, о которых мы поговорили, про следование, ветвление и цикл теоретически достаточно для написания программы совершенно любой сложности.

[01:18:33]

Функции

Теперь поговорим по поводу функций. Опять же при слове функция у многих, наверное, в голове возникает математическая функция. По крайней мере, у меня так, учитывая, что я математик и разные синусы, косинусы, возведение в квадрат и прочие непонятные некоторым людям слова. Но на самом деле функции имеют и бытовой характер. Есть бытовое представление этих функций.

Функции также называют подпрограммами. То есть какие-то действия законченные, выполненные нами, можно назвать функцией. Помните первый раз, когда мы искали второе по величине максимального, сначала искали максимальное, потом что-то делали и потом снова искали максимальное. Так вот поиск максимального можно считать некоторой подпрограммой. Если мы могли выделять отдельно и просто обращаться к ней, то это и была функция, вызываемая нами, когда захотим. То есть какой-то законченный блок можно оформить в виде функции.

[01:19:34]

Примеры функции

Но чтобы это не было чистой абстракцией, поговорим опять о некоторых бытовых аналогиях. Например, вам дали задание приготовить яичницу. Если мы рассмотрим человека, никогда не готовившего яичницу, то фраза, «приготовь яичницу» ему ничего не скажет. Нам необходимо каким-то образом саму инструкцию записать на отдельном листочке или это может быть страничка в книге рецептов или ещё что-то такое. Но условно нашу инструкцию можно записать следующим образом. Необходимо сначала взять сковороду, потом поставить сковороду на плиту и включить огонь, смазать сковороду маслом, дальше аккуратно разбить несколько яиц и вылить содержимое на горячую смазанную маслом сковороду. Возможно, 2 и 3 пункт надо поменять, я повар не очень хороший могу и перепутать отдельные шаги. После этого посолить, накрыть крышкой, следить за тем, чтобы яичница не подгорела, и когда яичница будет готова выключить плиту. Соответственно, вот есть некоторый набор инструкций, одинаковый при выполнении этого задания. Вас попросили приготовить яичницу,

вы всегда делаете одни и те же действия. На следующее утро вас попросили ещё раз приготовить яичницу или вы сами захотели приготовить яичницу, и вы делаете то же самое. То есть это некоторый законченный блок, некоторая такая подпрограмма, которой вы можете пользоваться.

Обратите внимание, что слово «несколько» выделено цветом. В это сделано не случайно и характеризует, что функции могут быть с параметрами, а могут быть без. Например, кто-то очень голодный и просит приготовить яичницу из трёх яиц. Что поменяется в этом алгоритме? Ничего, кроме слова «несколько». Вместо слова «несколько» добавим 3. То есть, мы также возьмём сковороду, поставим её на плиту, включим огонь, смажем сковороду маслом и аккуратно разобьём три яйца. А допустим, кто-то не очень голоден, но перед выходом из дома всё-таки хочет позавтракать, чтобы не захотелось есть, пока он едет до работы. Тогда он захочет, например, приготовить яичницу из одного яйца. И в этом случае слово «несколько» будет заменено на 1. Такие параметры (числа), влияющие на работу и исходные данные, называются «параметрами или аргументами функции». Если мы говорим про функции в реальной жизни, то в приготовлении яичницы, это может быть количество яиц.

Есть примеры функции с параметрами и без них. Например, функция «помой посуду» может быть функцией без параметров (без аргументов). Вас просто попросили помыть посуду, вы идёте к раковине и всё, что там есть, моете и аккуратно складываете в сушилку. Также и «приготовить яичницу» может быть функцией без параметров, если договорились, что когда вам не говорят количество яиц, то надо всегда готовите из двух яиц. Таким образом, какие-то функции могут быть как с параметрами, так и без параметров, и в реальной жизни это точно так же может работать.

При этом функция — это некоторая абстракция. Но если мы захотим наш код, написанный до этого, выделить в подпрограмму на псевдокоде это может выглядеть следующим образом. Опять же, в разных языках это делается по-разному. В Java это один способ, на Python это другой способ, в JS языках это третий способ. На псевдокоде можно записать примерно так. Указываем нашей программе, что написанное дальше это некоторая функция (подпрограмма, к которой можно обращаться). Обращаться к ней можно по имени и имя указывается далее. Мы говорим, что эту подпрограмму можно вызвать, обратившись по имени find_max. Если вернуться к именованию переменных, то с функциями всё так же самое. Важно сделать так, чтобы по имени функции стало понятно, что она делает. Функции чаще всего, выполняют какое-то действие, поэтому в названии должен присутствовать глагол. Здесь мы назвали find_max, как «найди максимальный». Даже не читая дальше, уже становится интуитивно понятно, что эта функция ищет что-то максимальное среди остальных. В скобках, после имени, указывается некоторый параметр то, что придёт в эту функцию. То есть исходное значение, с которым нужно будет работать. Если бы была функция, «приготовь яичницу», то здесь было бы некоторое число (аргумент). Например, число 3 тогда понятно, что всё, написанное ниже, нужно будет делать исходя из этого значения.

Дальше, что мы делаем? Мы вводим переменную size и помните, я вам говорил, что в некоторых языках уже можно размер массива получить как аргумент. Вот это пример, который в Java применяется. Если вы укажете ваш массив и через точку, напишите слово length (длина), то в качестве результата здесь будет число, равное количеству элементов в массиве. В разных языках это делается по-разному. Когда вы будете писать на конкретном языке, вам нужно будет посмотреть на синтаксис языка и уточнить, как нужно находить длину массива в этом

языке, но в целом всегда это что-то, связанное со словом length (длина) или size (размер). Собственно, мы задали размер массива. Обратите внимание того массива, который к нам пришёл в качестве аргумент. Нам дали один массив и здесь будет его длина, нам дадут другой массив, например, более большой и вот в этой переменной будет записан уже другой массив, соответственно, с другой длиной (размером). А после этого всё нам уже знакомо. Мы стартуем с нуля, вводим некоторую переменную индекс. В качестве максимального мы берём элемент с индексом ноль и запускаем цикл. В этом цикле все действия происходят так же, как мы неоднократно делали. И главное изменение в 9 строчке. Наша функция, когда мы её вызвали это подпрограмма, которую сейчас надо запустить и выполнить, а дальше прийти к каким-то действиям, должна вернуть управление программой назад. Мы сказали, find_max, она что-то выполнила и что-то нам вернула.

Возвращение — это слово return. Return не всегда возвращает значения. Он может возвращать или не возвращать, но главное — он говорит, что функция закончилась, она своё дело выполнила, всё отработала, продолжайте программу делать дальше. Вот теперь отсюда управление переходит к основной программе. Но об этом мы ещё поговорим. Но при этом функция return может вернуть какое-то значение, которое было нужно.

Если мы говорим про функцию find_max, очевидно, она у нас должна найти это максимальное и сказать, что максимально это вот это вот число. Это мы и делаем. Нашли максимальное и вернули его с помощью оператора return. Здесь также спрятана некоторая ошибка. Ваша задача указать в комментариях, где находится ошибка и какого она типа. Напомню, что есть ошибки синтаксические, выполнения и логические. Попробуйте, посмотреть, какая ошибка здесь присутствует. Что нужно исправить, чтобы код работал правильно и искал максимальное значение. Обращу внимание, это не относится ни к 1, ни к 9 строчке. С ними только знакомимся, а относятся только уже к знакомому вами коду к строкам со 2 по 8.

Теперь, если есть функция find_max, и она корректно работает, то поиск максимального элемента в массиве сильно упрощается. Наша программа становится более удобочитаемой. Например, у нас есть 2 массива, есть numbers и есть another numbers. То есть числа и есть, какие-то другие числа. Например, в массиве числа у нас находится уже знакомые нам 1, 8, 3, 2 и 6, а во 2 строчке мы вводим переменную с новым массивом и там располагаются уже какие-то совершенно другие числа. И при этом нам необходимо найти максимальный элемент в каждом из этих массивов.

Если мы задали функцию find_max, что мы можем сделать? Мы можем просто попросить нашу программу, когда будет нужно обратиться к функции find_max и сделать всё то, что в ней происходит. Что здесь происходит? Когда я говорю, что max number эта переменная, в которую запишется результат функции find_max тогда, тогда ей будет передан первый массив numbers. И если наша функция find_max работает корректно, тогда в max number запишется число 8. Но это же функция find_max одинаково хорошо сработает в случае, если я передам в неё другой массив. И в этом случае в another max number запишется число 74. Обратите внимание, если в этой функции были какие-то магические числа, то есть вместо array.length мы записали, например, число 5, то программа не сработала бы с массивом, где больше элементов, просто потому, что мы жёстко зафиксировали внутри массива. Именно поэтому нельзя использовать в программах магические числа, особенно это касается функций, которые вы будете вызывать точно с разными аргументами.

Соответственно, если функция уже есть, она готова и хорошо работает, тогда вы можете просто её вызывать, и она будет отрабатывать как положено. Можно ли обойтись без функции? Да, можно. И мы говорили, что ветвление, следование и циклы достаточно для написания программы любой сложности. И здесь вместо find_max мы могли бы воспользоваться алгоритмом, который видим на этом экране. И сначала я бы этот алгоритм провёл для массива numbers. Это 10 строчек кода. После этого я бы провёл этот же алгоритм. То есть ещё раз записал эти 10 строчек кода, но уже для массива another numbers и наша программа стала занимать очень много места. 10 строчек для этого кода, 10 для того, хотя можно 4 строчки здесь и 10 строчек для определения функции. А этой функцией можно пользоваться неограниченное количество раз. Если вам необходимо будет не для 2, а для 100 массивов найти максимальное значение, то не нужно 100 раз копировать этот алгоритм. Надо просто определённое количество раз вызвать функцию. И программа становится гораздо более удобочитаемой. Даже не зная, как выглядит функция, если вы правильным образом называете её, то всё получится. Вот мы смотрим программу, есть одни числа, есть другие числа. В переменную max number какое-то максимальное число, максимальное значение мы записываем результат функции find_max из массива numbers. И даже не зная, как работает find_max, но предполагая, корректное название, мы можем сказать, что в max number, скорее всего, будет число 8, потому что find_max, наверное, ищет максимальное значение. А в another max numbers, скорее всего, будет число 74, потому что max number ищет максимальное значение, а откуда ищет из массива another numbers, который вот у нас здесь представлен.

Если ваша программа написана так, что её легко читать, то можно даже не погружаться в глубину и сразу понимать, а что же происходит в программе. Но для этого требуются некоторые знания английского языка и тому, кто пишет, и тому, кто читает.

[01:32:32]

Функция — это абстракция

Мы ранее говорили, что программы по статистике чаще читают, чем пишутся. Здесь нужно понимать, что есть принцип как книга пишется для читателя, так и программа пишется для читателя. Она должна быть понятна не только автору, но и тому, кто будет её читать. И в этом случае это не только компилятор или интерпретатор, работающий с вашей программой, но и ваш коллега, учитель или ученик, который будет смотреть на код, написанный вами, пытаясь понять, а что же он делает.

Здесь мы хотим сказать немножко про абстракцию. Функция — это некоторая абстракция. Мы написали алгоритм поиска максимального числа, но человек, который не знает, как работает наша функция, но знает, что она делает (результат) уже может ей пользоваться. И то же самое в бытовом примере с чайником. Далеко не всё знают, что происходит в электрическом чайнике, когда вода вскипает. Почему-то он отключается. Какая такая магия происходит в вашем чайнике, что он у вас отключает электричество. Но при этом это никоим образом не мешает вам им пользоваться. Это уровень абстракции, в которой вы не погружаетесь. Вы знаете, что он делает, но не знаете как. Или пульт от телевизора. Все мы можем переключить телевизор, пользуясь пультом, но как это происходит внутри, как работают электросхемы, находящиеся внутри. Но это совсем не мешает вам пользоваться телевизором и пультом от телевизора, не говоря уже о компьютерах, телефонах и другой технике. Те же самые автомобили почему-то

ездят, но внутреннее устройство автомобилей знают далеко не всё, но при этом это не мешает людям пользоваться ими. Поэтому и в бытовом плане тоже у нас есть некоторые функции, есть некоторые уровни или, как говорят, слои абстракции, позволяющие пользоваться объектами, не погружаясь в то, как это всё работает.

В программировании на самом деле вы очень часто будете этим пользоваться. Вы, наверное, слышали, что существует какие-то библиотеки, какие-то фреймворки это как раз об этом. Когда вы знаете, что это делает и пользуетесь этим. Но далеко не всегда знаете, как это работает. Особенно если мы говорим про начинающих программистов (Junior, Middle), то это всегда работает именно так. В глубину очень мало кто погружается, и это на самом деле нормально. С этого уже можно стартовать.

В качестве примера здесь представлен тот же самый код только уже с функцией find second max. Думаю, по названию, все уже понимают, что эта функция находит второе по величине максимальное число и мы рассмотрели два подхода к поиску второго максимального. У нас был один подход, когда с математиком, выливающим воду из чайника за два прохода массива, смогли найти второй по величине элемент, это было не очень рационально. И потом мы рассмотрели другой подход, более рациональный, который позволяет за один проход нашего массива найти второй по величине элемент. Так вот нам, как пользователям программы даже не всегда бывает важно, а каким же образом написана та или иная функция. Значения, которое буду в 3 и 4 строке получаться будут одинаковыми, если мы воспользуемся одним способом или воспользуемся 2 способом. И это ещё одна польза от абстракции. Нам не сильно принципиально, а как же эти функции работают. Мы ими пользуемся.

Возможно, наш коллега работает над тем, чтобы эти функции работали. Он как-то написал и главное, они работают корректно, выдают правильный результат, но, возможно, не очень оптимально. И вот мы ими пользуемся. При этом спустя какое-то время наш коллега может догадаться, что нет необходимости два раза пробегать через этот массив, можно написать сюда более аккуратное решение, но мы об этом можем даже не узнать. Мы как пользовались функцией find second max, так и пользуемся. И в программировании это очень важная вещь. Когда мы можем абстрагироваться, и наша программа превращается в слоёный пирог. Можно выделять подпрограммы, которые каким-то образом работают, и их видоизменять, а основная программа будет продолжать работать точно так же.

[01:37:14]

Подведём итоги

На этой ноте мы остановимся. Введение в программировании дальше без непосредственного программирования, особенно когда мы ввели понятие функций, уже становится очень сложным. Поэтому пересмотрите ещё раз прошедшие лекции. Их достаточно, чтобы вы поняли, что программирование и написание кода, это две разные вещи.

После этого у нас будут курсы по информатике и математике. Мы чуть-чуть глубже погрузимся в механику процессов, то, как работают компьютеры, как хранится в них информация. Немножко вспомним математику, но не стоит бояться только на том уровне, который будет нужен. Ничего сверхсложного там не будет. После постепенно перейдём к программированию на конкретных языках, и вы сможете те программы, блок-схемы и алгоритмы, которые мы

разбирали и составляли, воспроизвести на конкретном языке программирования, посмотреть, как всё работает.

В комментариях под этим уроком можете оставить свои впечатления о курсе. Чего не хватило или, наоборот, всё понравилось. Нам очень интересно будет изучить все эти комментарии.

Всем спасибо, будем ждать вас на семинарах наших и на следующих курса. Пока.