# Абзац 1 :  
Друзья , привет ! С вами снова я , Денис . Рад вас приветствовать на лекции . Сегодня мы будем говорить о массивах . На прошлой лекции мы познакомились с языком программирования \*C-Sharp\* , с его основными характеристиками , а также решили блок базовых задач . На семинарах вы попрактиковались в разработке циклических условных конструкций , применяли арифметические операторы , а также операции ввода-вывода . Сегодня мы вспомним основные понятия , связанные с массивами , их характеристики , поговорим о том , в каких задачах \*массивы\* могут применяться . Познакомимся , как осуществляется операция создания , заполнения , а также вывода массивов на \*экран\* и решим блок задач на их обработку . Ближе к концу лекции мы познакомимся с двумя разновидностями циклов \*for\* и for each , а также , друзья , поговорим о таком важном аспекте , как изучение английского для программистов . Друзья , перед тем как приступить к практике и начать решать \*задачи\* , давайте вспомним , а что из себя представляет \*массив\* и какими характеристиками он обладает . Начнем с ключевого термина этой лекции – массив . Массив представляет собой структуру данных , которая предназначена для хранения элементов , как правило , одного типа . Массив может состоять из целых чисел , вещественных , может состоять из символов , а также , например , из строк . Если приводить примеры из реальной жизни , то массива мы можем назвать упорядоченный набор инструментов или же набор елочных игрушек , который также обладает характеристикой упорядоченности .  
  
# Абзац 2 :  
Второй важный термин – это индекс элемента массива . Друзья , этот термин можно сформулировать по-разному . Приведем наиболее популярные формулировки .  
  
# Абзац 3 :  
Итак , первое , да , это смещение элемента относительно начала массива . То есть , на сколько позиций \*элемент\* смещен , да , относительно его левой границы . Ну и вторая формулировка , наиболее простая , это просто позиция элемента в массиве . Друзья , напомню , что в большинстве языков программирования индекс массива начинается с нуля . Перейдем к основным характеристикам массивов . Первая характеристика это то , что массив должен иметь \*имя\* . Еще его называют идентификатор . С помощью имени мы осуществляем \*обращение\* к элементам массива . Если нам нужно изменить элемент или же просто его прочитать , то мы используем операцию обращения к элементам по индексу . Также , друзья , эту операцию называют \*индексация\* . В большинстве языков программирования она обозначается с помощью пары квадратных скобок . В примере приведен массив из пяти элементов и ниже вы можете видеть обращение к каждому его элементу с помощью оператора квадратные скобки . Отметим важную характеристику , которая определяет внутреннее устройство массивов . Друзья , \*элементы\* в массиве располагаются последовательно , то есть в памяти , да , они занимают ячейку за ячейкой . Ни в коем случае элементы в массиве не расположены хаотично или же в каком-либо другом порядке . Ну и в заключение добавим , что массивы имеют конечный размер , поскольку объем памяти у нас всегда ограничен .  
  
# Абзац 4 :  
Итак , после небольшого экскурса в теорию давайте посмотрим , как будут выглядеть основные операции при работе с массивами на языке C-Sharp .  
  
# Абзац 5 :  
Рассмотрим \*создание\* , заполнение и вывод массива на экран . В языке C-Sharp создание массива состоит из трех основных этапов . Первый этап – это указание типа данных для элемента массива . Также , друзья , мы указываем оператор квадратные скобки , чтобы показать , что у нас именно создается массив , а не обычная переменная .  
  
# Абзац 6 :  
Второй этап – это указание идентификатора или же имени массива . В данном примере , друзья , мы пользуемся именем R , что является сокращением от Array . Друзья , рекомендую давать массивам осмысленные имена . В самом простом случае можно дать имя Array . Однако рекомендую задуматься над более осмысленным именованием . Как в приведенных примерах , если вы оперируете набором файлов , то такой массив можно назвать именем files . Если же вы оперируете набором среднесуточных температур , то подобный массив можно назвать averageDialedTemperatures или сократить до averageDialedTemps . Но здесь важно не злоупотреблять с сокращением в именовании переменных , Особенно , если эти имена становятся не общепринятыми и не общеизвестными . Имена array , ar вполне , друзья , допустимы для учебных задач , где мы делаем упор на демонстрацию той или иной техники по обработке массивов . То есть акцент делается не на конкретной предметной области . Следующий этап – это выделение памяти от элемента массива . На данном этапе мы указываем \*количество\* элементов , а также используем оператор new , который нам выделит фрагмент памяти для хранения будущих элементов . На этом три основные операции по созданию массива закончены , но также , друзья , можно выделить и четвертый этап . Он заключается в связывании участков памяти , которые мы выделили с помощью оператора new , с именем массива с помощью оператора присваивания .  
  
# Абзац 7 :  
Работать сегодня мы будем в каталоге урок 2 . Друзья , иерархия каталогов на сегодня у меня уже создана . Перейдем в каталог с заданием 1 и создадим там проект , знакомый вам командой.NET New Console .  
  
# Абзац 8 :  
Перейдем в файл с исходным кодом program.cs Удалим \*код\* , который сгенерировался по умолчанию Итак , укажем тип данных , массив , целых чисел Далее дадим массиву имя , например , r Напомним сокращение от слова array И выделим память с помощью оператора new , допустим , на 5 целых чисел .  
  
# Абзац 9 :  
вещественных чисел , то в этом случае мы слева от присваивания и справа от присваивания использовали бы тип данных double или же float . Друзья , после выполнения данной операции у нас будет создан массив на 5 целых чисел . Обращаться к нему мы можем , используя его имя . Отмечу , что в языке C-sharp элементы при создании массива имеют \*значение\* по умолчанию . Если это числовой массив , то элементами являются нули .  
  
# Абзац 10 :  
Далее заполним массив произвольными целыми числами . Напомню , друзья , что для изменения элементов массива мы используем оператор индексации и в квадратных скобках указываем индекс элемента . Заполним его произвольными целыми числами . Для ускорения можно скопировать эти конструкции . И не забываем менять индексы массива . В данном случае они у нас изменяются от нулевого и до размера массива минус 1 . Сделаем элементы массива относительно случайными . Проверим , действительно ли массив заполнен . Для этого выведем каждый его элемент на экран знакомым методом writeLine из модуля консоли . Для вывода элементов , для обращения мы также используем имя массива и операцию индексации с указанием индекса необходимого нам элемента . Для ускорения конструкцию вывода можно скопировать необходимое количество раз , заменив индексы на соответствующих позициях . Друзья , выполним наш код с помощью команды.NET RUN . На экране мы видим значения , которые были помещены в массив . Друзья , а как вы думаете , что произойдет , если мы обратимся к элементу с индексом 5 , при том , что количество элементов в массиве равно 5 ? Давайте это проверим и попробуем вывести на экран элемент с индексом равным 5 .  
  
# Абзац 11 :  
Итак , воспользуемся все тем же методом консоли Brightline и выведем на экран массив с индексом равным 5 . Сохраним исходный код и запустим наш проект еще раз . Друзья , обратите \*внимание\* , что на экране у нас вывелась ошибка , связанная с выходом за границы массива . То есть , таким образом , среда выполнения предупреждает нас о том , что недопустимо обращаться за границы памяти , которые мы выделили для массива . Закомментируем строчку с недопустимым обращением к массиву и давайте продолжим .  
  
# Абзац 12 :  
Итак , друзья , мы создали массив на 5 элементов , заполнили его числами , вывели их на экран .  
  
# Абзац 13 :  
Давайте рассмотрим способ заполнения массива сразу же при его создании . Для этого воспользуемся той же конструкцией , которую мы использовали , только с небольшим изменением . Все так же будем создавать массив целочисленный . Дадим ему имя r2 , поскольку имя r уже занято . И после указания имени выделим память на все те же 5 элементов . Обязательно с указанием типа данных , и заполним их произвольными целыми числами сразу же при создании этого массива . Друзья , напомню , что редактор Visual Studio Code имеет инструменты форматирования кода . Для этого можно выделить необходимый участок , форматирования кода . Для этого можно выделить необходимый участок , нажать правой кнопкой мыши и выбрать пункт меню формат документ или сочетание горячих клавиш shift alt f .  
  
# Абзац 14 :  
Отмечу , что в C-Sharp можно \*использовать\* упрощенный синтаксис для создания массива . Он заключается в том , что команду на выделение памяти , то есть new int и указание количества элементов , в некотор случае команду на выделение памяти мы можем не использовать , поскольку \*число\* элементов у нас фиксировано и мы их указываем сразу при создании массива . Способ достаточно удобный для создания небольших массивов , когда заранее известны их элементы . И , друзья , отмечу , что , вероятно , знакомый многим из вас \*язык\* программирования \*Python\* имеет схожий синтаксис создания массива . Выглядит он примерно следующим образом .  
  
# Абзац 15 :  
И также отмечу , что тип данных у массива мы не указываем . Не забываем , друзья , форматировать наш код , чтобы повысить его читаемость . Сохраним наши наработки в репозитории \*git\* . Проиндексируем целиком каталог task1 .  
  
# Абзац 16 :  
Далее сформируем commit с указанием комментария о том , что мы добавили первую задачу в уроке 2 .  
  
# Абзац 17 :  
И отправим наши наработки на удаленный \*репозиторий\* . Друзья , перед тем как приступить к решению задач , давайте поговорим о том , а какие именно классы задач могут быть решены с помощью массивов , а также каким специальностям , навыкам работы с массивами и в целом знаниям основ программирования будет полезно . Один из самых простых вариантов использования – это хранение однотипных элементов и последующее обращение к ним . Например , в массиве мы можем сохранить набор целых чисел , чисел с плавающей точкой , символов или же набор строк , которые составляют некоторый текст . Также в массиве можем сохранить и более сложные объекты , например , файлы . Массивы часто используются для поиска элементов по какому-либо критерию . Например , знакомый вам \*поиск\* минимального и максимального значения среди набора целых чисел . Если же рассмотреть набор строк в виде имен файлов как массив , то в таком массиве мы можем осуществить поиск графических файлов , файлов с видеосодержимым или же , например , файлов с исходным кодом . В данном случае критерием будет расширение файла . В данном случае критерием будет расширение файла . Также отметим класс задач , в которых необходимо вычислить некоторую характеристику от всего набора элементов . Это может быть \*сумма\* значений , это может быть произведение или среднее значение . Друзья , в качестве примера можно рассмотреть вычисление суммарных затрат на предприятии . рассмотреть вычисления суммарных затрат на предприятии . В таком массиве элементами могут быть затраты предприятия за день , за неделю или же , например , за месяц . Друзья , вопрос на засыпку . А как вы думаете , какой тип данных может быть у элементов в подобном массиве ? Свои предположения можете написать в чат .  
  
# Абзац 18 :  
Второй пример – это нахождение средней температуры за год . В таком массиве элементами могут быть значения температур за месяц , из которых потом будет вычисляться среднегодовое значение . Друзья , перед тем как порешать конкретные задачки на обработку массивов , давайте обсудим еще такой вопрос , как применимость знаний о массивах , ну и в целом навыков программирования на конкретных специальностях . Наиболее очевидный случай это \*разработчик\* . Массивы являются достаточно простой и базовой структурой данных , с которой программисты знакомятся в первую очередь . После изучения массивов , а также получения навыков по их обработке , можно переходить к изучению таких структур данных , как стэк , очередь или же связный список . Следующая специальность , где навыки работы с массивами не будут лишними , это тестировщик программного обеспечения . На старте профессии умение программировать , вероятно , вам не понадобится , особенно если речь идет о ручном тестировании . Однако , если вам понадобится освоить автоматизированное тестирование , то здесь уже , друзья , без навыков программирования не обойтись . В частности , понимание массивов , а также навыки работы с ними существенно ускорят изучение языка программирования высокого уровня . К специальностям \*аналитик\* , продакт , а также project manager высоких требований в плане владения конкретным языком программирования не предъявляется . Однако умение программировать и в том числе навыки работы с массивами будут достаточно полезны аналитику , особенно если в круг его задач входит работа с большими объемами данных . Product и Project Manager понимание основ программирования будет достаточно полезно для взаимодействия с командой разработки . В частности , чтобы вести диалог с программистами на одном языке .  
  
# Абзац 19 :  
Итак , друзья , давайте перейдем к первой задаче . В ней требуется заполнить массив целыми числами от 1 до n , где n некоторое целое число , и вывести этот массив на экран . Давайте перечислим основные этапы в решении этой задачи , затем сформируем с вами блок-схему и запрограммируем на языке C-sharp . Первым шагом нам необходимо понять , а какой размер будет иметь создаваемый массив . Для этого , друзья , введем натуральное число n , после чего вторым шагом можем создать массив на n элементов , выделив под него память . Третьим этапом можно заполнить массив конкретными числами , после чего вывести его на экран . Для создания \*блок-схемы\* давайте вспомним задачу с предыдущего урока на вывод чисел от 1 до n. Друзья , обратите внимание , что эта \*задача\* очень похожа на текущую , поскольку в ней формируются числа от 1 до n и затем выводятся на экран . Давайте посмотрим , как можно модифицировать эту схему для решения нашей задачи . Первый этап у нас уже выполнен . Ввод числа n в схеме уже присутствует . Добавим создание массива в блоке инициализации переменных до запуска цикла . Друзья , обратите внимание , что создание массива у меня выделено желтым цветом . И напомню вам , что под созданием массива мы понимаем выделение участка памяти для размещения элементов .  
  
# Абзац 20 :  
Далее добавим основное действие , которое позволит заполнить массив . Для этого поместим значение счетчика на очередную позицию массива . Друзья , обратите внимание , что переменная \*счетчик\* изначально равна нулю , поэтому мы помещаем в массив число на единицу больше , то есть , как видно из схемы i плюс 1 .  
  
# Абзац 21 :  
Таким образом , друзья , в массив мы будем помещать числа 1 , 2 , 3 и так далее до числа n. Вывод элемента массива можно разместить в этом же цикле , но , друзья , сделать это нужно после помещения очередного значения в массив , иначе мы будем выводить элементы массива до их формирования . Также отмечу , что вывод элементов можно сделать и в отдельном цикле . То есть сначала у нас будет идти \*цикл\* по формированию элементов массива и ниже точно такой же цикл для их вывода .  
  
# Абзац 22 :  
Итак , наша \*блок-схема\* готова . Давайте посмотрим , как этот \*алгоритм\* будет выглядеть на языке C-Sharp . Перейдем в каталог с заданием 2 и создадим там проект командой.NET New Console . Перейдем в файл со сходным кодом , удалим код сгенерированный по умолчанию и , согласно нашей блок-схеме , создадим целочисленную переменную n , равную 10 . Друзья , поскольку число элементов в массиве заранее неизвестно , то мы будем пользоваться стандартным синтаксисом создания массива с указанием оператора new , типа данных и количества элементов , для которых мы будем создавать массив . Создадим переменную счетчик i , изначально равную нулю . Для заполнения и вывода мы используем цикл , поскольку будем повторять одни и те же действия , пока не дойдем до конца массива . Создадим цикл \*while\* с условием i строго меньше , чем количество элементов в массиве . В теле цикла сформируем очередной элемент массива i плюс 1 и поместим его на требуемую позицию в массиве r. Давайте выведем сразу его на экран с помощью инструкции консоли в райт . Друзья , чтобы элементы у нас не слипались при выводе , добавим пробел после каждого элемента . То есть продублируем инструкцию консоли в райт , только вместо элемента мы укажем символ пробела . И не забываем перейти к следующему значению счетчика путем увеличения его на единицу . Друзья , обратите внимание на вывод элементов на экран . его на единицу . Друзья , обратите внимание на вывод элементов на экран . Давайте оформим это более изящно и в одну строку . Воспользуемся средством C-Sharp , которое позволяет выводить на экран строку и помещать в нее переменную . Для этого воспользуемся все тем же методом консоли в write . Только перед строкой мы поставим значок доллара и в самой строке значение очередного элемента массива array с индексом i поместим , обратите внимание , в фигурные скобки и добавим знак пробел . Друзья , в этом случае то , что было помещено в фигурные скобки , будет рассматриваться как значение элемента массива или же , например , переменная , и будет выведено на экран именно то , что находится в памяти , то есть сам элемент массива .  
  
# Абзац 23 :  
Итак , давайте запустим наш код командой.NET RUN . На экране мы видим приветственное сообщение hello world , однако , друзья , такого кода в файле , да , в нашем program.cs нет . Такая ситуация часто возникает , если мы забываем сохранить изменения в исходном файле . Давайте сохраним изменения путем нажатия комбинации клавиш Ctrl-S и запустим наш код еще раз . На экране мы видим последовательность чисел от 1 до 10 . Следовательно , для этого набора элементов алгоритм работает верно . Друзья , обращу ваше внимание , что вместо использования переменной n , которая хранит количество элементов массива , мы можем воспользоваться свойством массива . свойства массива . Для этого обратимся по его имени , поставим точку и будет предложен набор методов и свойств , которыми располагает объект массив . Друзья , нам здесь из этого многообразия понадобится только свойство \*Length\* , которое как раз таки хранит количество элементов , под которые выделена память . То есть , Length в данном случае будет равен 10 . Можно пользоваться как этим свойством , так же можно пользоваться и переменной .  
  
# Абзац 24 :  
Друзья , давайте сделаем небольшую паузу , отдохнем от решения задач и рассмотрим два подхода к изучению программирования . Основная идея первого подхода заключается в постепенном повышении сложности решаемых задач . Друзья , наше знакомство с языком C-Sharp мы начали с таких тематик , как \*переменные\* , типы данных , арифметика и затем перешли к условным и циклическим конструкциям . Друзья , рекомендую начинаать знакомство с программированием , с решением базовых и простых задач . Например , нарешивать их некоторое количество , 5 , 10 , 15 штук , и переходить к более сложным . Более сложной задачей очень часто является комбинация из набора простых задач . Друзья , не рекомендую сразу браться за сложные и комбинированные задачи , особенно если у вас возникают сложности на базовых задачах . А рассмотренный подход , друзья , носит название « От простого к сложному » .  
  
# Абзац 25 :  
Второй подход заключается в том , что количество решенных вами простых и базовых задач будет постепенно переходить в качество . То есть вы будете формировать свое алгоритмическое мышление , которое позволит вам решать более сложные задачи и строить более сложные \*алгоритмы\* . Таким образом , вы сможете опираться на некоторый фундамент из множества решенных вами базовых и простых задач .  
  
# Абзац 26 :  
Двигаемся далее и давайте рассмотрим типовую задачу на поиск элементов в массиве , которые удовлетворяют некоторым условиям . Пусть у нас имеется массив на некоторое количество целых чисел , допустим на 10 штук . Требуется найти в нем четные числа и вывести их на экран . Друзья , напомню , что число является четным , если оно делится на цело на 2 .  
  
# Абзац 27 :  
Примеры четных чисел это 0 , 2 , 4 , 6 и так далее . Давайте для этой задачи перечислим основные этапы , сформируем ее блок-схему и реализуем эту схему на языке C-sharp .  
  
# Абзац 28 :  
Итак , в этой задаче можно выделить следующие этапы . Первое . Создать и заполнить массив на 10 целых чисел . Вторым этапом можно выделить проход по элементам , то есть просмотр каждого элемента . Третий этап – это проверка элемента на \*условия\* того , является ли он четным . Ну и четвертый этап , друзья , это вывести этот элемент на экран , если указанное \*условие\* выполняется . Перейдем в блок схеме . Обращу ваше внимание , друзья , что цикл прохода по элементам массива мы можем целиком взять из предыдущей задачи . Этот цикл будет одинаковым для всех задач , где требуется перебрать или же просмотреть элементы . Массив мы заполним с клавиатуры сразу же при его создании . Я обращу ваше внимание , что цикл в этом случае можно не использовать , поскольку число элементов заранее известно , напомню их 10 штук , и оно невелико .  
  
# Абзац 29 :  
Далее , в цикл прохода по элементам массива нам необходимо добавить проверку очередного элемента на четность . Если же этого мы не будем делать , то на экран будет выведены все элементы , включая нечетные . Обратите внимание , условие разместил в блоке решения и написал там array с индексом i из even , что переводится как четный ли элемент с индексом i. Друзья , такая формулировка условия вполне подходит в использовании блок-схемы . Однако при переносе блок-схемы на конкретный язык программирования могут возникнуть существенные сложности . могут возникнуть существенные сложности . Дело в том , что языкам программирования непонятна такая формулировка условия , которая вполне понятна человеку и допускается в использовании блок-схем . Попытка перенести такое условие в язык программирования приведет к синтаксической ошибке . Друзья , к счастью , у этой проблемы есть решение . И заключается оно в использовании операторов сравнения и арифметических операторов при построении условий . Эти операторы языки программирования вполне хорошо воспринимают . В данной задаче нам поможет оператор вычисления остатка отделения . Во многих современных языках программирования он обозначается либо с помощью значка % либо с помощью ключевого слова mod , что от английского в переводе с модула означает деление по модулю . И для проверки на равенство мы будем использовать двойной знак равно . И , друзья , напомню , что одинарный значок равно обозначает другую операцию .  
  
# Абзац 30 :  
Итак , скорректируем наши условия и применим арифметический оператор остаток отделения и проверки на равенство . Если остаток отделения на 2 равен 0 , это мы пишем в блоке решения , он у меня подсвечен оранжевым цветом , то это значит , что очередной элемент массива делится на 2 нацело . То есть , он четный и мы его будем выводить на экран ниже , друзья , с помощью блока ввод-вывод , который обозначается в виде параллелограммы .  
  
# Абзац 31 :  
В нем мы напишем то , что вывести очередной элемент массива array с индексом i. Итак , блок-схема нашего логаритма готова . Давайте посмотрим , как он будет выглядеть на языке C Sharp . Перейдем в каталог с заданием 3 и создадим там проект .  
  
# Абзац 32 :  
Перейдем в файл с исходным кодом , удалим сгенерированный по умолчанию . Согласно нашей блок-схеме создадим целочисленную переменную n , равную 10 , и массив на 10 целых чисел . Друзья , предлагаю использовать упрощенный синтаксис создания массива , поскольку число элементов заранее известно и оно невелико . Создадим массив из 10 произвольных целых чисел , четных , а также добавим нечетные . Не забываем про форматирование кода . Не забываем про форматирование кода .  
  
# Абзац 33 :  
Создадим переменную счетчик i , изначально равную 0 , и оформим цикл while с условием счетчик строго меньше , чем количество элементов n. На каждой \*итерации\* будем проверять четность очередного элемента с помощью условной конструкции if . Обратимся к конкретному элементу по индексу i. Разделим его с остатком на 2 и проверим , равен ли этот остаток 0 . Если это так , друзья , то значит , что элемент четный и нам нужно вывести его на экран . Давайте это сделаем с помощью метода write . Друзья , предлагаю использовать метод вывода переменной сразу же в строке . То есть , напомню , используем символ доллара и затем затем в строке , в фигурных скобочках , пишем элемент , который хотим вывести на экран . В данном случае это просто очередной элемент массива . И добавим пробел , чтобы элементы не слепались . Давайте запустим наш код . Друзья , на экране мы видим , что алгоритм у нас зациклился и значение первого элемента , двойка , выводится в бесконечном цикле . Как вы думаете , почему так вышло ? Я установлю вывод комбинацией клавиши Ctrl-C. Друзья , все дело в том , что мы с вами забыли увеличить значение переменной i , то есть перейти к следующему элементу массива . Без этого мы будем обрабатывать каждый раз первый элемент массива и проверять его . То , что мы видим . То есть проверили , вывели , проверили , вывели . И если не переходить к следующему элементу , то мы зациклим наш цикл .  
  
# Абзац 34 :  
Итак , сохраним изменения и запустим еще раз .  
  
# Абзац 35 :  
еще раз . На экране мы видим четные числа , которые являются элементами этого массива . Друзья , рекомендую проверить правильность работы этого алгоритма на различных входных данных . Например , массив можно задать целиком из четных чисел , из нечетных , ну или как в нашем случае из комбинации , как четных , так и нечетных . Таким образом мы проверим работоспособность алгоритма на различных входных условиях .  
  
# Абзац 36 :  
Итак , мы с вами решили две типовые задачи на работу с массивами . Вспомнили , как в C-Sharp обозначаются циклические конструкции , условные , а также поработали с арифметикой и вводом-выводом . Друзья , давайте вспомним задачу из лекции введения в \*программирование\* , которая была связана с вычислением веса наиболее тяжелой \*гири\* . Напомню вам , как выглядит блок-схема для поиска наиболее тяжелой гири из 5 штук . Друзья , мы поочередно , как можете видеть , делаем 4 сравнения в блоках решения , да , и при необходимости , если эти сравнения дают нам ответ да , то по ветке yes мы обновляем значение переменной max в соответствующих блоках процесс . После проведения всех сравнений в переменной max у нас будет находиться значение наиболее тяжелой гири . На предыдущей лекции мы с вами запрограммировали алгоритм для поиска наибольшего из 5 целых чисел . Мы с вами создали 5 целочисленных переменных и разместили в них некоторые произвольные числа .  
  
# Абзац 37 :  
Далее мы с вами сделали 4 сравнения с помощью оператора if , и за счет этого мы нашли вес наиболее тяжелой гири , после чего в конце алгоритма мы вывели значение переменная max на экран . Друзья , но как изменится этот алгоритм , если число гирь , да , или же чисел будет увеличено ? Предположим , что нам необходимо найти самую тяжелую гирю из количества 10 штук . Если оставить логику решения без изменений , то код может выглядеть примерно следующим образом . Обратите внимание , друзья , на количество переменных . Их здесь 10 штук . И количество сравнений . Да , в целом задача решена , но код стал достаточно громоздким и содержит большое количество повторяющихся инструкций . Давайте еще немного изменим условия нашей задачи и увеличим количество гирь , скажем , до 100 штук . Как можно видеть из фрагмента исходного кода , букв латинского алфавита для именования переменных мне не хватило , я начал использовать переменные из двух букв .  
  
# Абзац 38 :  
Также можно давать имена следующие , да , гиря 1 , гиря 2 и так далее .  
  
# Абзац 39 :  
Но нужно использовать соответствующие английские слова , да , то есть wait1 , wait2 и так далее . Друзья , если использовать эту же , да , логику решения задачи , то для создания переменных у нас уйдет порядка 100 строк исходного кода и нам нужно будет запрограммировать порядка 100 практически одинаковых инструкций сравнения с использованием оператора if . Друзья , согласитесь , что для решения подобной задачи нам потребуется несколько иной подход . И одним из таких подходов мы можем рассмотреть использование массива для хранения весов , представленных гирь . Сформулируем задачу в более общем виде . Дано n гирь , где n – некоторое натуральное число , и требуется найти наиболее тяжелую гирю . Вспомним алгоритм поиска , который был приведен в курсе ведения в программировании , основанный на хранении весов гирь в массиве и использовании цикла . Друзья , обратите внимание , что в этом алгоритме также используется переменная max и проводится n сравнений для определения гири наибольшего веса . Отличие этого алгоритма заключается в том , что в нем переменные хранятся не россыпью , как в приведенных ранее алгоритмах , а в упорядоченном контейнере по массиве . А также , друзья , операция сравнения весов гирь не повторяется в коде много раз , а участвует один раз и расположена в цикле . Отмечу , что этот алгоритм может быть применен для произвольного количества гирь , как для пяти , для сотни , так и для тысячи штук .  
  
# Абзац 40 :  
Итак , перейдем в каталог для четвертого задания . Также создадим там новый проект .  
  
# Абзац 41 :  
Давайте возьмем количество гирь , равное , допустим , 5 штук . Создадим целочисленный массив из 5 весов гирь . Друзья , поскольку у нас количество весов небольшое , заранее известное , мы можем воспользоваться упрощенным синтаксисом создания массива . Создадим массив из набора произвольных различных чисел . Не забываем форматировать код .  
  
# Абзац 42 :  
Далее создадим цикл while , который у нас пройдет по элементам массива , и перед этим мы создадим переменную счетчик , которую будем использовать для условия цикла i меньше чем количество элементов . Также мы эту переменную будем использовать при обращении к самим элементам . Друзья , для проверки , да , не превышает ли очередной элемент массива значение переменной max , мы напишем условие array с индексом i больше , чем переменная max . Ну и , конечно же , не забываем , да , предварительно эту переменную создать . И проинициализируем ее первым элементом массива , то есть элементом с индексом 0 . Если это так , да , если очередной элемент у нас превысил текущее значение максимума , то мы обновим переменную max , поместив в нее текущий элемент массива . Обращаю внимание , друзья , что в случае невыполнения до этого действия этого условия , если оно неверно , то никакого отдельного действия для ветки ELSA не предусмотрено . Мы ее пропускаем . Не забываем перейти к следующему элементу массива , увеличив значение счетчика на единицу . И после завершения цикла выведем значение переменной max на экран , чтобы проверить правильность работы алгоритма . Сохраним изменения в исходном коде . Запустим наш код командой dotnet run . Ожидаем увидеть ответ 9 , что соответствует наибольшему значению в этом массиве . Наши ожидания совпадают с выводом . Это говорит о том , что алгоритм работает правильно , друзья , для указанного набора данных . Друзья , также как и в предыдущей задаче , не забываем проверить правильность работы нашего алгоритма на различных входных данных . Применительно к этой задаче я бы рекомендовал проверить ее работоспособность на массиве , когда у нас наибольшее число находится в конце массива , в начале или же где-то посередине . Также лишним не будет проверить работу алгоритма , когда массив отсортирован по возрастанию или же наоборот по убыванию . Таким образом вы проверите правильность его работы на различных начальных условиях . До настоящего момента мы с вами работали только с одной разновидностью циклов . Это while . Давайте посмотрим , а какие еще способы создания циклических конструкций присутствуют в языке C-sharp . Друзья , на экране представлены блок-схемы с использованием блока счетный цикл . Наверняка некоторые из вас при решении задач в курсе знакомства с программированием использовали этот блок . Большинство современных язы с программированием использовали этот блок . Большинство современных языков программирования поддерживают синтаксис описания счетных циклов . Давайте посмотрим , как это выглядит в C-Sharp .  
  
# Абзац 43 :  
Итак , для создания счетного цикла используется ключевое слово for . Но в отличие от цикла while , где в круглоскопах мы указываем только условия в цикле for , обратите внимание , также указывается создание переменной счетчика , в этом случае i равное 0 , и изменение этой переменной , то есть ее \*модификация\* i равно i плюс 1 .  
  
# Абзац 44 :  
Друзья , давайте рассмотрим составные части цикла for более подробно . Первый блок – это блок инициализации , то есть создание переменной счетчика и присвоение ей некоторого начального значения . В качестве имен переменных обычно используется IJK . Это стандартный подход при создании переменных счетчиков , будь то в цикле for , будь то в цикле while .  
  
# Абзац 45 :  
Второй блок – это условия . Напомню , под условием мы понимаем выражение логического типа , на основании которого мы можем сказать , верно оно или же нет . В данном случае , друзья , как и в цикле while , на основании условия принимается решение , будет ли выполняться очередная итерация или же цикл закончится . Также отмечу то , что в условиях часто применяются операторы арифметики и операторы сравнения . Третья часть цикла – это модификация , то есть изменение переменной счетчика . Переменная счетчика , как в этом примере , может расти с шагом на 1 , может расти с большим шагом 2 , 3 , 4 , может умножаться , а также может уменьшаться . То есть будет конструкция i равно i минус 1 . Ну и само собой тело цикла , которое располагается в фигурных скобках . То есть , набор инструкций , который будет выполняться на каждой его итерации . Друзья , для правильного применения цикла for крайне важно знать механизм или же алгоритм его работы . Давайте посмотрим , как работает цикл в приведенном примере .  
  
# Абзац 46 :  
Итак , первым шагом выполняется блок инициализации . В данном случае переменная i присваивается значение 0 . Вторым шагом , друзья , происходит проверка условия . То есть мы проверяем , меньше ли переменная i , чем число 10 . Если это верно , то мы выполняем , меньше ли переменная i , чем число 10 . Если это верно , то мы выполняем тело цикла , то есть те инструкции , которые заключены в фигурных скобках . Соответственно , после выполнения тела цикла происходит модификация , то есть то , что мы пишем с вами в теле цикла while . В этом случае i равно i плюс 1 . Ну и пятый пункт – это возврат к пункту 2 , то есть к проверке условия . То есть мы снова проверяем условие . Если оно верно , мы выполняем тело цикла . Если же оно неверно , мы цикл завершаем . Друзья , обращу ваше внимание , что точно таким же образом цикл for устроен в языках программирования C , C++ , Java и PHP .  
  
# Абзац 47 :  
Итак , на текущий момент мы знаем , что в C Sharp используются две разновидности циклов – FOR и WHILE . Но как понять , какой из них применить к каждой конкретной задаче ? Давайте , друзья , сформулируем некоторые рекомендации . И обращу внимание , что эти рекомендации не являются строгими правилами , но придерживаться их стоит . Цикл for мы часто используем , когда требуется перебрать элементы массива . Вывод на экран , как в этом примере . Или же подсчет суммы , подсчет среднего арифметического , или допустим поиск элементов по каким-либо критериям . Также цикл for может быть применен , когда верхняя граница числа итерации фиксирована и известна до запуска программы . В данном примере суммируются числа от 1 до 100 , и мы уверены , что число итераций здесь не превысит 100 . В случаях , когда нельзя заранее предсказать число итераций или же когда нам не неизвестно верхняя их граница , то можем применять цикл while .  
  
# Абзац 48 :  
Давайте рассмотрим классический пример применения этого цикла при чтении файла посимольно . Пусть у нас имеется некоторый файл , который состоит из 27 символов . В зависимости от его кодировки , будь то ANSI , будь то \*UTF-8\* или же \*Unicode\* , один символ может занимать 1 , 2 , 3 или даже 4 байта . Как вы можете видеть из строки состояния , которая указывается в нижней части блокнота , этот файл имеет кодировку UTF-8 . То есть один такой символ может иметь размер как 1 , 2 , так и 3 или 4 байта . Это зависит от того , является ли символ буквенно-цифровым или же представляет собой некоторую иконку или же эмодзу . Обратите внимание , что размер этого файла равен 60 байтам . И зависимость от количества символов в нем не очевидна . То есть нельзя сказать , что один символ занимает 1 байт или один символ занимает 2 байта , поскольку символов , я нап очевидны . То есть нельзя сказать , что один символ занимает один байт или один символ занимает два байта , поскольку символов , я напомню , здесь 27 . Поэтому , друзья , даже если мы знаем размер этого файла в байтах , мы заранее не можем предсказать , да , а сколько там будет храниться символов . Соответственно , мы не знаем , на сколько итераций нам запускать цикл for . Как раз-таки здесь нам может быть полезен цикл while . В его условии можем написать ключевое слово \*true\* , что будет означать цикл бесконечный . В его действиях мы напишем чтение очередного символа из файла и вторым действием вывод этого символа на экран . Остановить этот цикл можно будет при достижении конца файла .  
  
# Абзац 49 :  
Друзья , давайте рассмотрим третий способ организации цикла в C-Sharp . Это цикл \*forEach\* . Для его создания мы используем ключевое слово forEach .  
  
# Абзац 50 :  
Далее в круглых скобках мы создаем переменную цикла с указанием ее типа данных , в данном случае это целочисленный тип данных , и имени . В этом примере имя у нее E. Друзья , важный момент . Тип данных у этой переменной должен совпадать с типом данных у элементов массива . Затем мы указываем ключевое слово IN и указываем сам массив , элементы которого нам требуется перебрать . Работает цикл следующим образом . На каждой итерации в переменную цикла , в данном примере это переменная e , помещается , а если быть точнее копируется очередной элемент массива .  
  
# Абзац 51 :  
То есть на первой итерации в переменную e попадает единица , на второй итерации двойка и так далее , пока все элементы массива не будут перебраны . Этот цикл работает практически так же , как и цикл for , только в нем не используются индексы . Отмечу , что пара циклов for и while имеют принципиальное отличие от цикла forEach в плане обращения к его элементам , а также возможности изменять эти элементы . Так вот , ключевое отличие заключается в том , что элементы в цикле forEach , они доступны только на чтение , то есть нет возможности их изменять . Достигается это за счет того , что элементы массива каждый раз копируются , именно подчеркиваю , копируются в переменную цикла . Например , с помощью цикла forEach мы можем вывести элементы на экран , как представлено в этом примере , посчитать их сумму или , например , средне арифметическое . Кроме того , forEach нам предоставляет возможность осуществить поиск каких-либо элементов . Друзья , важный момент , что попытка изменить содержимое элемента массива с помощью цикла forEach , как здесь представлено в примере , приведет к ошибке . И большинство средств разработки , не исключением здесь является и Visual Studio Code , подчеркнут еще до этапа компиляции красным и как бы просигнализируют о том , что эта конструкция приведет к ошибке .  
  
# Абзац 52 :  
Для лучшего понимания работы цикла for each давайте рассмотрим пример из реальной жизни . Приведем набор коробок с некоторым содержимым и будем считать этот набор массивом .  
  
# Абзац 53 :  
Так вот , друзья , цикл for each позволяет нам заглянуть , проверить содержимое каждой коробки , то есть узнать характеристики этого содержимого , допустим , вес , размер и так далее . Однако цикл forEach не позволяет нам изменять содержимое коробок , удалять что-либо оттуда или добавлять новые предметы . Для подобных манипуляций нам необходимо использовать \*циклы\* for или while , которые позволяют изменять содержимое элементов .  
  
# Абзац 54 :  
Итак , давайте применим циклы for и for each к нашей задаче с гирями . Вернем переменную , которая хранит максимальное значение элементов массива в изначальное состояние , то есть присвоим ей значение первого элемента массива . И давайте запустим цикл for . Напомню , первым шагом мы создаем переменную счетчик . Друзья , переменная счетчик i у нас уже задействована , поэтому давайте использовать другую . Пускай это будет переменная счетчик j. Она равна нулю .  
  
# Абзац 55 :  
Далее укажем условие цикла . Оно такое же будет , как и в цикле while . j меньше , чем число элементов n. И , друзья , укажем изменение или же модификацию переменной счетчика . Друзья , обратите внимание , что я использую оператор инкремента . Возможно , вы им уже пользовались . Он работает так же , как и конструкция j равно j плюс 1 . Только имеет более сокращенный вид . Предлагаю им пользоваться .  
  
# Абзац 56 :  
Далее , как и в цикле while , оформляем тело в фигурных скобках . Тело можно скопировать целиком , поскольку у нас идет работа с индексами , и тело никак не изменится . Единственное , что скорректируем имя переменной счетчика . Друзья , и конечно же , обратите внимание , что здесь нам не нужно дублировать модификацию счетчика , поскольку ее изменение заложено уже в самом цикле for . То есть эту строчку мы обязательно удаляем . заложено уже в самом цикле for . То есть эту строчку мы обязательно удаляем .  
  
# Абзац 57 :  
Итак , проверим работоспособность нашего цикла for . Для этого мы видим значение переменной максимум на экран . Перейдем в каталог с задачей 4 и запустим там наш проект . На экране ожидаем увидеть дважды , да ? 9 и 9 . Ожидания совпадают с выводом . Также оформим решение с помощью цикла forEach . Еще раз вернем значение переменной max в начальное состояние и запустим цикла forEach . Еще раз вернем значение переменной max в начальное состояние и запустим цикл forEach . Напомню , он состоит из переменной цикла с типом данных , точно таким же , как и тип данных у элемента массива . В нашем случае это целое число . Затем идет имя переменной , ключевое слово in и сам массив . Друзья , обращение по индексу в этом цикле уже не используется . Тело цикла мы можем скопировать , но необходимо сделать соответствующие замены . То есть , таким образом , обратиться к элементу массива мы не сможем , поскольку у нас здесь работа с индексами отсутствует . Мы с вами должны пользоваться переменной цикла . В данном случае это E. Укажем переменную E. Логика работы никак не меняется . Меняется способ обращения к элементу массива .  
  
# Абзац 58 :  
обращения к элементу массива .  
  
# Абзац 59 :  
Итак , выведем результат работы цикла на экран , запустим наш код , ожидаем увидеть также величину 9 . Да , наше ожидание соответствует выводу . Друзья , на этом практическая часть лекции заканчивается . И в заключении давайте обсудим , насколько нам нужен технический английский при разработке программ . При разработке программного обеспечения специалист зачастую сталкивается с терминологией на английском языке . Мы с вами в процессе решения задач построения алгоритмов , блок-схем часто сталкивались с терминологией на английском . Это были имена переменных , функции , а также сообщения об ошибках . Давайте резюмируем основные моменты , где знание английского будет для нас хорошим подспорьем . Первый аспект применения языка – это именование объектов . Для лучшего восприятия исходного кода мы стараемся давать переменным , функциям и другим объектам осмысленные имена , разумеется , на английском языке . В примере приведены три имени переменных с достаточно популярными названиями . Это цифра , да , от английского , digit , количество , amount и среднее , \*average\* .  
  
# Абзац 60 :  
Давайте забежим немного вперед и рассмотрим несколько примеров именований функций . Отмечу , что зачастую имена функций состоят из нескольких слов , одной из которых является глагол . Первая функция осуществляет чтение файла по некоторому пути , вторая пишет в журнал сообщение и третья сканирует директорию на предмет наличия интересующих файлов Друзья , о функциях мы будем говорить подробнее на следующей лекции . Чтобы вести разработку программного обеспечения быстро и не обращаться к переводчику за именованием для каждой переменной , требуется пополнять свой словарный запас английскими техническими терминами . Следующий аспект , где английский может существенно нас выручить , это чтение ошибок . Будь то от редактора Visual Studio Code , будь то сообщение об ошибках от компилятора , или же , как в этом примере , от среды выполнения . Дело в том , что даже в русифицированных средах разработки сообщения об ошибках могут появляться на английском языке . Связано это с тем , что далеко не все строки , не все сообщения имеют перевод на русский . Соответственно , знание английского языка поможет вам оперативно читать сообщения об ошибках , улавливать их суть и не прибегать к помощи переводчика . Здесь я уже не говорю о том , что большое число инструментов для разработки в принципе не имеют локализации на русском языке . Третий аспект применения английского – это поиск путей , способов , решения возникающих проблем при разработке программного обеспечения .  
  
# Абзац 61 :  
Давайте рассмотрим пример поиска информации о совместимости некоторого программного средства \*OpenSSL\* с операционной системой Windows 7 . Друзья , подскажу , OpenSSL это достаточно распространенный продукт , одно из направлений в использовании которого – шифрование данных . Кому знакомы термины шифрования , криптография , можете написать об этом в чат . Так вот , обратите внимание , что попытка сформировать запрос на русском языке к явному ответу не приводит . Среди первых результатов в списке упоминания о средствах OpenSSL отсутствует совсем . Я уже не говорю о том , насколько это средство совместимо с операционной системой Windows 7 .  
  
# Абзац 62 :  
Давайте рассмотрим результат поисковой выдачи , но при формировании запроса на английском языке . Как можете видеть , уже в первой ссылке находится наш вопрос о совместимости программных средств . Друзья , можете поверить мне на слово , что подобные ситуации встречаются достаточно часто . будь то в форумах , порталах , иных ресурсах , подходящая техническая информация может попросту отсутствовать или же иметь неудовлетворительное качество . Ну и четвертый аспект применения английского , который я вынес на рассмотрение , это изучение технической документации . Документация у большинства инструментов разработки программного обеспечения изначально пишется на английском языке и не всегда получается найти качественный перевод на русский . Да , здесь можно пользоваться переводчиками , в том числе автоматизированным переводом . Но , друзья , часто случается , что при автоматизированном переводе некоторые технические термины теряют свой смысл или он искажается . И зачастую приходится обращаться к первоисточнику .  
  
# Абзац 63 :  
Итак , друзья , мы с вами рассмотрели несколько важных аспектов применения технического английского языка . Но здесь может возникнуть вопрос , а с чего , собственно , начать изучение ? Особенно , если ранее такого опыта у вас не было . Друзья , позвольте мне дать небольшую рекомендацию , выполнить которую большого труда не составит , но пользу от нее вы сможете получить достаточно быстро . Рекомендация эта достаточно нехитрая и заключается она в введении словаря или тетради с техническими терминами . То есть , если в процессе разработки вам встречается тот или иной термин , будь то от компилятора , будь то от среды разработки , будь то от сообщений среды выполнения . Вы этот термин отмечаете себе в тетрадь или ваш конспект и снабжаете его переводом . Такой словарь или тетрадь может получиться достаточно большим , поэтому чтобы эти термины откладывались в долговременную память и вы их со временем запоминали , рекомендую этот конспект время от времени перечитывать . Таким образом , количество незнакомых терминов или же сообщений об ошибках , будь то в редакторе Visual Studio Code , или же от платформы.NET , или же от среды исполнения будет сокращаться . Это вам позволит улавливать основной смысл сообщений об ошибках , давать именам и функциям более качественные названия , а также оперативно искать причины проблем в вашем коде . Ну что , друзья , на этом лекция о массивах подошла к концу . Давайте подведем ее основные итоги .  
  
# Абзац 64 :  
Итак , в начале занятия мы с вами рассмотрели базовое определение касающиеся массивов , рассмотрели их основные характеристики . После вводного экскурса мы с вами решили блок задач на базовую обработку массивов , будь то создание , вывод на экран , заполнение , а также поиск элементов .  
  
# Абзац 65 :  
Далее мы рассмотрели две разновидности циклов for и for each , а также обсудили особенности их применения на конкретных задачах . И в заключение , друзья , мы поговорили о важности изучения технического английского и пользы , которую от этого вы можете получить . На этом , друзья , я с вами прощаюсь . Благодарю вас за внимание . Увидимся на следующей лекции и на семинарах .