# Абзац 1 :  
you you you you Дорогие друзья , рад вас приветствовать на курсе знакомства с языками программирования . Сегодня у нас вводная лекция . Мы поговорим о языках программирования в целом , о языке программирования \*C-Sharp\* , об алгоритмах , а также порешаем \*задачи\* . Друзья , цель этого курса – познакомить вас с синтаксисом языка программирования высокого уровня , дать \*представление\* о подходах к написанию базовых алгоритмов , а также дать представление о подходах к написанию качественного кода . Друзья , перед тем как продолжить и познакомиться с программой настоящего курса , а также с планом работ на сегодня , предлагаю познакомиться и я расскажу немного о себе . Меня зовут Черняев Денис . В компании Geekbrains я работаю уже более года . До прихода в компанию я работал частным преподавателем на порталах дистанционного обучения . Среди преподаваемых предметов я могу выделить языки программирования C , C++ , C Sharp , Python . Также могу выделить тематику \*алгоритмы\* и \*структуры\* данных , операционные системы и компьютерные сети . Друзья , преподавание для меня не является основной работой , это по большей части хобби . Основная моя деятельность это разработка сетевого программного обеспечения , служб Windows , а также построение вычислительных систем на базе оборудования Cisco и операционных систем Linux .  
  
# Абзац 2 :  
Итак , друзья , предлагаю ознакомиться с планом настоящего курса , который состоит из шести лекций и восьми семинарских занятий . Сегодня у нас вводная лекция , мы будем говорить о языке C-Sharp , о его основных особенностях , а также решать первые свои задачи . Вторая лекция у нас будет посвящена массивам , на третьей лекции мы будем говорить о функциях , о их составляющих частях и о том , как их применять в наших программах . Четвертая , пятая и шестая лекция затронут такие тематики , как \*правила\* написания хорошего кода , двумерные массивы , а также такой подход к написанию алгоритмов , как рекурсия .  
  
# Абзац 3 :  
Итак , план работы на сегодня у нас следующий . Поговорим о том , почему же C-Sharp был выбран инструментом для решения задач на текущем курсе . Настроим \*редактор\* Visual Studio Code .  
  
# Абзац 4 :  
Вспомним основные понятия из курса ведения программирования и приступим к написанию наших первых программ . Друзья , я думаю , при просмотре программы текущего курса многие из вас задались вопросом , а почему именно C-Sharp выбран в качестве основного инструмента для решения задач . В команде Geekbrains было рассмотрено несколько языков программирования , представленных на экране , и я могу сказать следующее , то что на текущем уровков программирования , представленных на экране . И я могу сказать следующее . То , что на текущем уровне программирования , на текущем уровне написания кода , программы , написанные на любом языке программирования , из представленных будут выглядеть примерно одинаково . И написав программу на любом из представленных языков , вы без труда сможете перенести ее на любой другой понравившийся вам \*язык\* .  
  
# Абзац 5 :  
Давайте рассмотрим критерии , по которым язык программирования C-Sharp был выбран в качестве основного инструмента для решения задач на текущем курсе . Друзья , на экране вы можете видеть три фрагмента исходного кода , в котором создаются три \*переменные\* с разными типами данных . Обратите внимание , то что в языках Python и JavaScript \*тип\* данных не указывается . В языке C-Sharp требуется явного указания типа данных при создании \*переменной\* . Друзья , дело в том , что язык C-Sharp имеет статическую типизацию , то есть он требует явного указания типа данных при ее создании . На самом деле это требование можно обойти , однако на начальных этапах разработки программ я это делать не рекомендую . Преимущество статической типизации заключается в следующем . На каждом этапе выполнения программы , на каждом этапе написания ее кода вы видите и понимаете , какой тип данных у вас имеет каждая из переменных . Кроме того , друзья , я отмечу , что переходить с разработки на языке с динамической типизации на язык со статической типизации значительно сложнее , чем наоборот . Например , разработчику на языках C , C++ или Java значительно проще перейти на разработку на языке Python или , например , PHP , нежели чем в обратном направлении . Отмечу , что статическую типизацию также имеют языки C , C++ , Java .  
  
# Абзац 6 :  
Давайте рассмотрим второе свойство , которое выделяет язык C-sharp среди этих языков . На экране представлены три фрагмента исходного кода , на которых выводится на \*экран\* приветственное сообщение Hello World . Друзья , обратите внимание на наличие дополнительных кодовых конструкций в языках C++ и Java . Кому знакомы ключевые слова class или \*main\* , предлагаю написать об этом сообщением в чат . На начальных этапах при решении алгоритмических задач нас явно будут отвлекать такие конструкции , как \*классы\* , точки входа в приложение , пространство имен от написания алгоритмов . точки входа в приложение , пространство имен , от написания алгоритмов . В языке C-Sharp можно воспользоваться режимом , который позволяет писать простые программы , не перегружая их сложным синтексисом . То есть все необходимые конструкции , в частности , которые вы видите на экране у языков C++ , Java , компилятор C-Sharp будет поставлять автоматически , а мы сможем сконцентрироваться на решении алгоритмических задач , будь то связанных с циклами , будь то светлениями , будь то с арифметикой . Другими словами , язык C-Sharp позволяет писать \*код\* точно так же , как и на языке Python , но используя все плюсы статической типизации . Да , если вдруг часть слов , которые я сказал , была вам непонят , или же вы слышите их впервые , можете не переживать . Друзья , поверьте на слово , что язык C-Sharp – это отличный выбор для старта программирования , решения алгоритмических задач и полученные навыки вы сможете легко перенести на любой другой понравившийся вам язык . Друзья , обращаю ваше внимание , что целью настоящего курса не является изучение синтеза языка C-Sharp , его продвинутых возможностей и функционалов . Целью настоящего курса является получение навыков решения алгоритмов различных уровней сложности , а также изучение лучших практик при их построении . И C-Sharp нам в этом деле понадобится как инструмент создания программ . Для использования любого языка программирования нам потребуется \*создать\* и настроить для него окружение . Для языка программирования C-Sharp это платформа.NET . Для удобства написания кода мы будем использовать редактор Visual Studio Code и систему контроля версии \*Git\* , которые , скорее всего , у вас уже настроены на предыдущем курсе . Если же нет , то инструкцию по ним я продублирую в описании к лекции . Друзья , отмечу , что наиболее приоритетный вариант настройки и установки окружения – это его самостоятельное развертывание . Однако , если у вас возникнут сложности при настройке окружения , можно воспользоваться альтернативным вариантом . Заключается он в использовании облачного сервиса , который предоставляет вам удаленный рабочий стол с уже установленными и настроенными , редактором Visual Studio Code , системой контроля версии Git , а также с платформой.NET . Этим сервисом можно воспользоваться в случае , если , например , ваше оборудование не позволяет установить необходимое программное обеспечение . Работа с этим сервисом можно напрямую из браузера . Инструкции по работе с этим облачным средством будут продублированы к описанию под лекции . Для установки и настройки платформы.NET нам потребуется проделать следующие шаги . Перейдем на страницу загрузки установщика для платформы.NET . Выберем необходимую для нас архитектуру . Это может быть либо Linux подобной операционной системы , либо Mac или Windows .  
  
# Абзац 7 :  
И далее нам необходимо будет выбрать разрядность . Друзья , в большинстве случаев это будет 64-разрядный установочный файл , который необходимо загрузить и установить следуя инструкциям установочника .  
  
# Абзац 8 :  
Далее нам требуется настроить расширение для удобства разработки в Visual Studio Code . Следующим шагом для настройки Visual Studio Code под разработку на C-Sharp откроем среду разработки и перейдем в соответствующую вкладку с установкой расширений . Друзья , нам понадобится буквально одно расширение , которое называется C-Sharp для Visual Studio Code . Введем первую часть названия в поиске C-Sharp и выберем первые предложенные в списке расширения . Обратите внимание на производителя , на поставщика этого расширения , Microsoft , а также на количество скачиваний , чтобы убедиться , что это действительно оригинальное расширение . Для установки необходимо нажать кнопку Install . У меня такой кнопки нет , поскольку данное расширение у меня уже установлено .  
  
# Абзац 9 :  
Итак , друзья , на этом \*установка\* и \*настройка\* окружения для языка C-Sharp закончена . На данном этапе мы готовы к написанию наших первых программ на языке C-Sharp . В качестве первой программы предлагаю реализовать \*вывод\* приездного сообщения на экран Hello World . О такой программе вы наверняка уже слышали . Обычно ее используют для того , чтобы понять , а правильно ли настроено наше окружение .  
  
# Абзац 10 :  
Итак , перейдем в редактор Visual Studio Code . Вести разработку мы сегодня будем в каталоге Lesson1 . И давайте создадим в нем подкаталог Task1 , в котором мы будем создавать \*первый\* \*проект\* .  
  
# Абзац 11 :  
Итак , создавать каталоги мы можем с помощью \*команды\* mkdir от английского make directory создать директорию передаем в качестве параметра ей имя требуемого каталога данном случае task 1 и не забудем в него перейти \*командой\* cd от английского change directory .  
  
# Абзац 12 :  
Итак , друзья , для создания проекта на языке C-sharp мы используем команду.NET new console . Обращаю ваше внимание , что запускать ее нужно в том каталоге , в котором вы хотите создать необходимые файлы проекта . Команда.NET-new-console создаст файл с исходным кодом program.cs , а также файл с расширением C-sharp project , который и отвечает за настройку текущего проекта .  
  
# Абзац 13 :  
Итак , друзья , перейдем в файл program.cs , удалим код , который сгенерировался автоматически , и напишем его самостоятельно . Для вывода информации на \*консоль\* в языке C-Sharp мы используем модуль консоль . Этот модуль нам предоставляет метод \*writeLine\* , который и осуществляет вывод сообщений , либо же значений переменных , либо какой-либо другой информации на экран . Давайте выведем сообщение , приветственное Hello World . Друзья , не забываем в конце ставить точку запятой , для того , чтобы компилятор понимал что на этом инструкция заканчивается .  
  
# Абзац 14 :  
запятой это признак конца инструкции и переход к следующей итак запустим наш код с помощью команды . net.ru обращаю внимание что эта команда нужно выполнять в каталоге , в котором проект был создан . То есть у нас это \*каталог\* Task1 . Компиляция исходного кода может занимать некоторое время .  
  
# Абзац 15 :  
Итак , на экране мы видим приездное сообщение , что говорит о правильной работе программы . Друзья , не забываем сохранить наши \*наработки\* в репозитории Git . Для этого перейдем на один каталог повыше и выполним оттуда команду git add и проиндексируем целиком наш созданный каталог . Task 1 . Создадим commit с указанием комментария . Допустим , назовем его урок 1 перед этим указав действия да то что мы добавили новый проект урок 1 ну и допустим task 1 после чего отправим наши наработки на удаленной репозитории с помощью команды git push origin-main . Отмечу , что каталог вашего локального репозитория , в моем случае это 0.1 , нижнее подчеркивание C-sharp , должен содержать файл \*gitignore\* , в котором приведены инструкции для того , чтобы git не отправлял на удаленный \*репозиторий\* файлы , в которых нет необходимости . Это различные бинарные файлы , временные и прочие . Именно этот файл позволяет это предупредить .  
  
# Абзац 16 :  
Итак , друзья , наше окружение настроено и проверено . Прежде чем приступить к решению задач , давайте вспомним базовые термины из курса « Ведение в \*программирование\* » .  
  
# Абзац 17 :  
Рассмотрим две группы терминов . Первая группа – это переменные , типы данных и \*арифметика\* . Вторая группа терминов – это виды алгоритмических конструкций .  
  
# Абзац 18 :  
Итак , первый термин – это переменные . Под переменной мы обычно понимаем некоторую область памяти , которая предназначена для хранения значений определенного типа . Переменной можно также дать и более простое определение , например , назвать ее некоторым контейнером для хранения определенных значений . Эти \*значения\* могут быть целочисленными , строковыми , а также хранить в себе и более сложные объекты . Переменная имеет некоторую характеристику , которую мы называем типом данных . Тип данных нам говорит о том , какого рода значения могут храниться в этой переменной . Например , тип данных может быть целочисленный , строковый , вещественные числа , а также и более сложные объекты . Друзья , важное понятие для тех , кто только знакомится с программированием , это арифметика или же арифметические \*операции\* . Среди них можно выделить простые , такие как \*сложение\* , вычитание , умножение , \*деление\* . А также есть и более сложные , например , деление с остатком , возведение в степень или же извлечение квадратного корня . Все эти операции мы будем использовать в решении задач на текущем курсе . Вторая группа терминов описывает виды алгоритмов . Напомню , линейный \*алгоритм\* подразумевает последовательное выполнение инструкций без изменения их порядка следования . Условный алгоритм подразумевает разветвление алгоритма на две и более ветвей в зависимости от некоторого \*условия\* . Ну и третий вид алгоритма – это циклический . Циклический алгоритм подразумевает повторение либо одинаковых действий , либо похожих . Также для циклических алгоритмов мы используем некоторые условия , которые позволяют ограничить количество этих повторений . Или же , другими словами , итераций .  
  
# Абзац 19 :  
Итак , друзья , мы настроили окружение , вспомнили базовые термины .  
  
# Абзац 20 :  
Давайте приступим к знакомству с языком программирования и приступим к решению первых задач . Начать предлагаю с базовых вещей , с создания переменных и арифметических операций над ними .  
  
# Абзац 21 :  
Итак , рассмотрим первый пример . В нем требуется объявить \*переменную\* , задать ей начальное \*значение\* и вывести его на экран . Перед тем , как начать писать код , давайте посмотрим , а как в языке программирования C-Sharp создаются переменные . Создание переменной в языке C-Sharp , а также и в большинстве других языков со статической типизацией , состоит из трех основных этапов . Первый этап – это указание типа данных для значения , которое в этой переменной будет храниться .  
  
# Абзац 22 :  
\*Второй\* этап – это указание непосредственно имени переменной или же ее называют идентификатор . На идентификатор накладываются ограничения . Оно не должно начинаться с цифр , должно состоять из латинских букв , а также может начинаться с символа подчеркивания . И важное ограничение , которое накладывается на именование переменных , это то , что оно не должно совпадать с зарезервированными ключевыми словами языка C-Sharp . Ну и третий этап это \*инициализация\* , то есть присвоение переменной определенного значения . В данном случае мы присваиваем конкретное значение в виде целого числа равного 100 . Давайте запрограммируем этот пример на языке C-sharp .  
  
# Абзац 23 :  
Итак , перейдем в редактор Visual Studio Code и давайте дополним наш первый проект . Укажем тип данальное значение , допустим , 100 . Друзья , для того , чтобы проверить , правильно ли у нас проинциализирована переменная , проинциализирована ли она вообще , давайте выведем ее значение на экран . Воспользуемся все тем же модулем в консоли и методом writeLine . Напомню , он может выводить на экран как сообщения , так и значения переменных . Для этого в круглых скобках мы укажем имя переменной и не забываем поставить точку запятой как признак конца инструкции . Перейдем в каталог task1 и выполним там программу на запуск нашего проекта task1 .  
  
# Абзац 24 :  
Итак , на экране мы видим приветственное сообщение hello world , которое мы оставили , а также значение переменной value , которую мы только что создали . Друзья , давайте закомментируем первую строчку программы в консоли в WriteLine с помощью двух слэшей . Комментарии мы используем для отключения каких-либо строк кода , а также мы их используем для того , чтобы дать пояснение к какому-либо участку программы . Проверим работоспособность нашего кода еще раз . Ожидаем увидеть на экране только вывод значения переменной value , что подтверждается выводом . Друзья , мы создали целочисленную переменную , проинициализировали ее некоторым значением и вывели на экран . Но что если нам понадобится создать значения , которые хранят дробную часть ? В данном случае нам понадобится использовать другие типы данных . Давайте посмотрим , какие типы данных существуют в языке C-Sharp . С первым типом данных мы уже познакомились . Его также называют целочисленным типом данных . Другой целочисленный тип данных носит название \*long\* . Его отличие от типа данных \*int\* заключается в том , что он позволяет хранить числа большего размера , нежели чем тип данных int . То есть , Long позволяет сохранить большее повеличение числа , чем тип данных int .  
  
# Абзац 25 :  
Второй из представленных типов данных – это вещественный тип . Он позволяет хранить дробные числа . Существует две его разновидности – это типы данных float и \*double\* . Отличаются они размером . Тип данных может сохранить большую дробную часть по сравнению , чем тип данных float . Символьный тип данных используется для хранения отдельных символов . Будь то символы латинского алфавита , будь то цифры , будь то специальные символы . Ну и последний в списке – это логический тип данных . Этот тип данных позволяет сохранить всего лишь два значения . Либо логическую истину – \*true\* , либо же логически ложное значение – \*false\* . Друзья , его мы будем часто использовать в наших условных конструкциях и при создании циклов . Давайте расширим нашу программу и добавим в нее переменные рассмотренных типов данных .  
  
# Абзац 26 :  
Итак , вернемся к проекту task1 и добавим туда новые переменные . Создадим переменную с вещественным типом данных double , назовем ее , например , d и присвоим ей некоторое значение с дробной частью , например , 5,55 .  
  
# Абзац 27 :  
Далее мы с вами создадим переменную типа данных char от английского character , то есть символный тип данных . Назовем ее , например , c и поместим туда значение , например , специальный символ восклицательного знака . Друзья , обращаю внимание , что переменная типа char , ее значения хранятся в одинарных кавычках , в отличие от строк , которые мы выводили на экран ранее . То есть символ и тип данных предполагает хранение одиночных символов . Ну и третья переменная логического типа . Назовем ее , например , b и присвоим ей значение логической истинности . Выведем значение трех переменных на экран . Все так же воспользуемся методом консоли в WriteLine . Для ускорения разработки эту строку можно скопировать , заменив только имена переменных , которые мы выводим на экран .  
  
# Абзац 28 :  
Итак , проверим работоспособность нашей программы Запустим ее , ожидаем увидеть значение четырех переменных Что подтверждается выводом на экране \*Сохраним\* наши наработки в репозитории git Для этого проиндексируем файл с исходным кодом program.cs командой git add командой git commit создадим commit И давайте дадим ему \*комментарий\* , что были добавлены несколько перемен . А также укажем расположение этого проекта .  
  
# Абзац 29 :  
И отправим наши наработки на удаленный репозиторий командой git push итак друзья рассмотрим второй пример в котором требуется создать целочисленные и вещественные переменные а затем применить к ним базовые арифметические операции , после чего вывести результаты на экран . Создадим для второго примера новый каталог . Назовем его task2 . И создадим в каталоге task2 проект с помощью команды.NET New Console .  
  
# Абзац 30 :  
Перейдем в файл со сходным кодом . Удалим код , который создан по умолчанию и давайте создадим парочку целочисленных переменных согласно заданию .  
  
# Абзац 31 :  
Итак , создадим первую целочисленную переменную , назовем ее , например , i1 , дадим ей значение , допустим , 5 , создадим вторую переменную с названием допустим i2 и проинициализируем ее значением 6 . Вычислим сумму двух созданных переменных . Для суммирования мы используем оператор плюс и давайте поместим \*результат\* сложения двух этих переменных в новую переменную с именем , например , sam . Друзья , обращаю внимание , что в этом случае я уже даю более осмысленное имя переменной , чтобы код был более читаемым . Здесь возникает вопрос , а какой тип данных будет у переменной sam ? Давайте остановимся на этом вопросе чуть подробнее . Друзья , сложение – это бинарная \*операция\* , то есть в ней принимает участие два операнта . В данном случае в ней принимает участие первое слагаемое слева от плюса и второе справа от него . Дело в том , что тип данных у этой операции сложения будет формироваться на основе типа данных оперантов – первого операнта и второго . И тут возникает важный вопрос , а каким образом он будет формироваться ? Друзья , для того чтобы дать ответ на этот вопрос , давайте сформулируем два правила . Если типы данных у операндов совпадают , то в этом случае тип данных у результата будет совпадать с типом данных этих операндов . В нашем случае операнды i1 и i2 имеют тип данных int , поэтому тип данных у переменной sum , то есть у результата этого сложения , будет также int , то есть она будет целочисленная .  
  
# Абзац 32 :  
Итак , выведем результат сложения на экран с помощью команды консоли в WriteLine . Проверим правильность работы программы и запустим ее . 1.5 mm На экране мы видим сообщение Hello World . Друзья , дело в том , что после каждого изменения исходного кода его нужно сохранить . Сделать это можно с помощью горячей комбинации клавиш Ctrl-S. Сохраним изменения в исходном коде и выполним команду еще раз . Как и ожидалось , на экране мы видим результат сложения двух чисел . Но что если типы данных у операндов , которые участвуют в бинарной операции , будут разные ?  
  
# Абзац 33 :  
Давайте посмотрим на этот пример Итак , создадим еще одну переменную Дадим ей тип данных , например , long и имя , допустим , l1 Поместим туда некоторое значение , пускай это будет 6 И давайте таким же образом вычислим сумму переменных l1 и i1 , то есть сумму переменной типа long и сумму переменной с типом int . И поместим результат в переменную sum2 , поскольку имя sum уже занято . Здесь возникает вопрос , а какой тип данных будет иметь переменная SAM2 ? Друзья , давайте сформулируем второе правило . Если типы данных у операндов , которые участвуют в бинарной операции , разные , то в этом случае тип данных у результата будет формироваться как наиболее объемный или наиболее вместительный тип . Другими словами , если участвуют типы данных в операции long и int , то тип данных у результата будет long , поскольку диапазон значений у типа данных long значительно шире , в разы больше , нежели чем у int . Или другой пример . Если в операции участвуют операнды с типами int и double , то тип данных у результата операции будет double , поскольку double хранит как целые числа , так и числа с плавающей точкой . То есть этот тип данных является более вместительным и более объемным .  
  
# Абзац 34 :  
Итак , в нашем примере участвуют операнды с типами данных long и int . long int . Согласно второму правилу , результат бинарной операции будет иметь наиболее вместительный тип , то есть long . С помощью модуля консолей и метода writeLine выведем результат на экран и проверим работоспособность нашей программы . Не забудем сохранить файл со сходным кодом горячей комбинации клавиш Ctrl-S и запустим наш проект .  
  
# Абзац 35 :  
На экране мы видим результат вычисления первой операции и второй . И давайте такую же операцию проделаем с типами данных int и double . Переменная int у нас уже есть , это переменная i1 . Создадим переменную double , назовем ее допустим d1 и присвоим ей некоторое значение , пускай это будет 5,5 . Вычислим сумму этих переменных d1 плюс i1 и поместим ее в переменную sum , допустим назовем 3 , поскольку sum2 уже была занята .  
  
# Абзац 36 :  
Итак , друзья , согласно второму правилу формирования типа данных у результата этой операции , типом данных у переменной sum3 будет именно double , поскольку он является более объемным и более вместительным , если выбирать из двух double и int . Точно так же пишем тип double .  
  
# Абзац 37 :  
Итак , проделаем такие же шаги , только с операцией умножения . Для операции умножения мы используем оператор \*звездочка\* , перемножим сначала целочисленные переменные . Согласно первому правилу результат будет иметь тип данных int . Давайте назовем его product1 от английского умножения . Давайте назовем его product1 , что в переводе с английского произведение . Перемножим две переменные i1 , i2 .  
  
# Абзац 38 :  
Далее перемножим переменные типов данных long , int . И результат поместим в тип данных , который более объемный и более вместительный .  
  
# Абзац 39 :  
l1 умножить на переменную i1 . Результат поместим в переменную product2 и третье умножение это с участием переменных double и переменной int . Здесь , согласно второму правилу , у результата этой операции будет тип double как наиболее вместительный . Переменную назовем , допустим , product3 и поместим туда произведение целочисленной переменной на переменную типа double .  
  
# Абзац 40 :  
Итак , друзья , выведем наши ответы на экран , чтобы убедиться в их правильности . Для ускорения разработки некоторые фрагменты кода можно копировать .  
  
# Абзац 41 :  
Главное следить за отсутствием Итак , сохраним файл с исходным кодом , запустим наш проект . На экране мы видим 5 результатов наших арифметических операций . Сохраним наши наработки в репозитории git , перейдем на один каталог повыше , проиндексируем наш каталог task2 целиком командой git add , создадим соответствующий commit и отправим его на удаленный репозиторий .  
  
# Абзац 42 :  
Рассмотрим операцию деления . При делении переменных тип данных у результата операции формируется по таким же правилам , как и в операциях сложения и умножения . Давайте посмотрим , как это работает и какие нюансы при этом могут возникать . При делении переменных правило формирования типа данных у результата работает точно так же . Однако при делении целых чисел результат может получаться иногда дробным .  
  
# Абзац 43 :  
Давайте рассмотрим этот \*случай\* .  
  
# Абзац 44 :  
Итак , разделим переменную i1 на переменную i2 . Разделим переменную i1 на переменную i2 . Результат поместим в переменную quotient , что в переводе с английского означает « частный » . Согласно первому правилу формирования типа данных у результата бинарной операции , здесь мы пишем тип данных int , целое \*число\* . Давайте посмотрим , какой результат мы получим и выведем его на экран . Выводим на экран значение переменной quotient , сохраняем файл с исходным кодом и запускаем наш проект командой dotnet run .  
  
# Абзац 45 :  
Итак , друзья , на экране мы видим значение 0 , однако ожидали увидеть значение 5 шестых , что примерно равно 0,8 . А как вы думаете , почему так произошло ? Друзья , на самом деле , какой-то магии или странности здесь нет . Целый тип данных не предполагает хранение дробной части . В определении целых чисел у нас получается дробная часть , которая в нем сохранена быть не может . Возникает вопрос , а как в этом случае нам сохранить эту дробную часть ? Ответ кроется в типе данных у результата операции деления . Если он будет вещественный , то это позволит сохранить в нем дробную часть . Для этого необходимо , чтобы хотя бы один из операндов имел вещественный тип . Согласно второму правилу , в этом случае у результата операции тип данных будет вещественный тип , и это позволит в нем сохранить дробную часть . Воспользуемся ранее созданной переменной d1 . Для удобства проверки присвоим ей некоторое значение , например , 6 , и поделим переменную i1 , которая имеет целый тип , на переменную d1 , которая имеет тип данных double . Согласно второму правилу тип данных у результата этого деления будет double . Ну и создадим переменную стип . Согласно второму правилу , результат у этой операции будет double . Дадим имя переменной , допустим , caution2 .  
  
# Абзац 46 :  
И выведем результат на экран . Сохраним изменения в файле и запустим проект команды.NET RAN . Ожидаем увидеть число , равное 0,8 , что подтверждается выводом . Друзья , эти примеры могут показаться достаточно сложными , но я рекомендую поэкспериментировать с ними и запустить их на своих данных . Эта тематика достаточно важная с точки зрения понимания взаимодействия различных типов данных в языке C-Sharp . Также еще один важный момент . Не стоит думать , что в языках программирования Python , Perl , PHP , JavaScript эти особенности формирования типа данных у результатов операции отсутствуют . Напротив , они присутствуют там в полной мере , только скрыты от разработчика . Язык программирования C-Sharp позволяет нам с ними познакомиться и понять , как они устроены под капотом .  
  
# Абзац 47 :  
Итак , сохраним наши наработки в репозитории git . Для этого перейдем на один каталог повыше и командой git add проиндексируем весь каталог task2 .  
  
# Абзац 48 :  
Далее командой git commit сформируем commit с соответствующим комментариям и отправим наработки на удаленной репозитории . Друзья , для закрепления полученных навыков я даю вам домашнее \*задание\* , в котором требуется вычислить значение некоторой формулы , по сути она является дробью , числитель которой является произведением двух целых чисел , знаменатели вы можете видеть , их сумму . Друзья , обратите внимание на нюансы и особенности , которые могут возникать при делении целых чисел .  
  
# Абзац 49 :  
Друзья , давайте вспомним задачу из курса « Введение в программирование » , где нужно было найти вес самой тяжелой гири . Напомню , на вход в эту задачу поступало 5 гирь с различными весами и требовалось определить , какая гиря самая тяжелая и вывести ее вес . Перед тем как решать эту задачу , давайте решим несколько упрощенный вариант , когда гирь всего две и требуется найти вес также наиболее тяжелой гири . Если приводить аналогию с целыми числами , то \*задача\* может быть сформулирована следующим образом . Дано два целых числа и вывести на экран из них наибольшее из них . Перед тем как приступать к написанию кода , давайте выделим основные этапы в решении этой задачи и построим ее блок-схему . Первым делом нам понадобится создание двух переменных , которые будут хранить веса этих гирь . Можно назвать их , например , а и b. Следующим шагом нам необходимо будет \*сравнить\* эти две переменные , какая из них больше .  
  
# Абзац 50 :  
В случае , если первая переменная превышает вторую , то мы просто выведем на экран эту переменную a. Иначе , если же это не так , то мы выведем на экран значение второй переменной . Друзья , какие еще способы представления алгоритмов вам известны ? Совершенно верно , если вы вспомнили про представление алгоритмов в виде блок-схемы или же псевдокода .  
  
# Абзац 51 :  
Итак , давайте построим блок-схему для этой задачи .  
  
# Абзац 52 :  
Итак , давайте посмотрим , как может выглядеть \*блок-схема\* для решения этой задачи . Первое . Ввод переменных А и В. Для ввода мы используем блок-параллелограмм , то есть ввод-вывод данных .  
  
# Абзац 53 :  
Далее у нас идет \*условие\* , в котором мы проверяем верно ли что переменная a превышает значение переменной b. Изображаем это в блоке ромб и далее в зависимости от результата этого \*сравнения\* мы можем идти по \*одной\* из двух веток .  
  
# Абзац 54 :  
Либо это ветка \*yes\* вывод значения переменной а на экран , либо вторая ветка no это вывод на экран переменной B. Итак , после словесного описания алгоритма и построения его блок-схемы , давайте оживим его на языке C sharp . Перейдем в редактор Visual Studio Code . В каталоге Lesson1 создадим каталог \*Task3\* и создадим в этом каталоге проект командой.NET New Console , предварительно в него перейдя .  
  
# Абзац 55 :  
Перейдем в файл со сходным кодом , удалим код по ум умолчанию и согласно нашей схеме давайте создадим две целочисленные переменные . Назовем ее переменная a. Допустим , пускай будет равна 5 . Вторую переменную назовем b. Пускай она будет равна 7 . И теперь , друзья , нам нужно их сравнить . Для выполнения операции сравнения в подавляющем большинстве языков программирования используется оператор if . В языке C-sharp после оператора if мы указываем условие , помещая его в круглые скобочки . Условием у нас будет \*выражение\* , на которое мы можем дать ответ . Либо логическая истина , либо логический ложный ответ .  
  
# Абзац 56 :  
Далее нам требуется сравнить две переменные a и b на предмет того , какая из них больше . Для сравнения переменных или каких-либо других проверок в большинстве языков программирования используется оператор if .  
  
# Абзац 57 :  
Далее в фуглскопах мы указываем условие , то есть выражение , которое отвечает на вопрос верно ли что . Переменная а больше чем b. То есть условие это выражение логического типа , которое имеет только два возможных ответа . True или False . Истина или ложь .  
  
# Абзац 58 :  
Далее , в фигурных скобках после конструкции if и после самого условия мы указываем код , который будет выполнен , если это условие верно . В данном случае мы просто выводим на экран переменную a согласно блок-схеме с помощью модуля консоли и метода brightline . В противном случае , если же это выражение неверно , то есть b у нас либо равно a , либо его превысило , мы код , который соответствует этой ветви выполнения , размещаем после ключевого слова \*else\* . Помещаем его также в фигурных скобках . В этом случае мы напишем , что переменная b будет выведена на экран . Другими словами , если a превышает значение переменной b и это верно , то мы выводим значение переменной a. В противном случае мы выводим значение переменной b. Давайте проверим и запустим проект с помощью команды.NET RUN На экране мы видим значение 7 , что соответствует правильной работе алгоритма Друзья , этот алгоритм и другие , которые мы будем разрабатывать , я рекомендую проверять на разных значениях входных переменных . То есть , таким образом вы проверите работу вашего алгоритма при различных входных условиях . Применительно к нашему случаю можно поменять местами .  
  
# Абзац 59 :  
То есть , в А поместить переменную заранее большего значения по величине нежели чем переменную b и также проверить правильность работы запустим наш проект на экране видим значение переменной а что говорит о правильной работе логарифм итак друзья сохраним наши наработки в репозитории git . Для этого целиком проиндексируем проект каталог task3 . Сформируем соответствующий commit .  
  
# Абзац 60 :  
и отправим его на удаленный репозиторий . Друзья , вопрос на засыпку для знатоков системы контроля версии Git . А почему я использую ключевое слово main , а не \*master\* ? Кто знает , какая в этом разница ? Свои предположения можете написать в комментариях под трансляцией .  
  
# Абзац 61 :  
Итак , давайте рассмотрим эту же задачу , только с 5 гирями . Если приводить аналогию с набором чисел , то эта задача может быть формулирована следующим образом . Дан набор из пяти чисел и нужно определить большее из них . Перед тем , как приступить к написанию кода , давайте вспомним алгоритм , который был предложен на лекции .  
  
# Абзац 62 :  
Итак , для нахождения \*максимального\* из пяти чисел мы в начале алгоритма задаем переменную max , которая изначально равна переменной a , то есть значение первого числа или же значение веса первой гири .  
  
# Абзац 63 :  
Далее мы проводим четыре подряд идущих сравнения , чтобы выяснить , а не превышает ли очередное число текущее значение переменной max . Если это так , то есть число превышает значение , то мы обновляем переменную max . Это вы можете видеть в блоках процесс под блоками решения . И по завершению этого алгоритма в переменной max будет храниться наибольшее значение из 5 весов гирь или наибольшее из 5 введенных чисел . Как и в двух предыдущих задачах , давайте перейдем от графического представления алгоритма к коду на языке C-Sharp .  
  
# Абзац 64 :  
Итак , перейдем в редактор Visual Studio Code , создадим там каталог Task4 и создадим в этом каталоге новый проект . Все той же командой можем воспользоваться и взять ее из истории команд.NET New Console . Перейдем в файл с исходным кодом . Согласно блок-схеме создадим 5 целочисленных переменных для хранения весов гирь и присвоим им некоторые произвольные значения .  
  
# Абзац 65 :  
Друзья , обратите внимание , что символ присваивания я отделяю пробелом от имени переменной , а также пробелом от значения . Это делается для повышения читаемости . Это действие можно сделать автоматически , используя инструменты VS Code . Для этого необходимо целиком выделить интересующий фрагмент , нажать правой кнопкой мыши и найти пункт меню формат документ . Эта опция отформатирует документ в соответствии с рекомендациями по оформлению кода для языка C-Sharp . В частности , как вы можете видеть , добавились \*пробелы\* , отделяющие \*присваивание\* от имен переменных .  
  
# Абзац 66 :  
Далее мы создадим переменную max , которая изначально \*инициализируется\* первой переменной a. И согласно Болл-схеме нам необходимо сделать четыре сравнения , чтобы определить не превышает ли каждая из переменных b , c , d , e нашу переменную max . Начнем с переменной b. Напоминаю , что в условии мы помещаем выражение типа bool . То есть логическое выражение , которое может иметь только два значения true и false .  
  
# Абзац 67 :  
Далее в фигурных скобках мы помещаем код , который соответствует ветке yes . То есть если это условие верно . В данном случае , согласно блок-схеме , мы обновляем значение переменной max . И для ускорения разработки этот фрагмент кода можно скопировать и \*повторить\* его три раза . Обращаю внимание , что нужно быть внимательным при копировании , поскольку могут появляться опечатки .  
  
# Абзац 68 :  
Итак , делаем соответствующие замены .  
  
# Абзац 69 :  
Переменная D и пятая переменная E. Итак , друзья , следующий момент . Обратите внимание , что здесь отсутствует ветвь ELSE . Она отсутствует по той причине , что нет какого-то отдельного действия специального , если это условие окажется неверным . Если проверяемое условие окажется неверным , то никаких отдельных действий мы делать не будем , а просто перейдем к следующей проверке . действий мы делать не будем , а просто перейдем к следующей проверке .  
  
# Абзац 70 :  
Итак , выведем на экран значение переменной максимум , чтобы проверить правильность работы алгоритма .  
  
# Абзац 71 :  
Итак , проверяем еще раз все наши переменные , раняем файл со сходным кодом и командой.NET RUN компилируем и запускаем проект . Ожидаем увидеть на экране 10 . На экране мы действительно видим значение 10 , что говорит о правильной работе алгоритма . Однако правильность работы алгоритма я рекомендую проверять и на других входных данных . Применительно к этой задаче можно использовать различные наборы данных , в частности , когда наибольшее число находится в начале списка , где-то в середине списка или же в конце . Таким образом , вы проверите работу этого логаритма при различных входных условиях .  
  
# Абзац 72 :  
Итак , сохраним наши наработки в репозитории git . Проиндексируем целиком каталог task4 .  
  
# Абзац 73 :  
Далее создадим commit с соответствующим комментарием . И укажем , что этот каталог добавлен . Ну что , друзья , мы дошли до финального раздела этой лекции . Это \*циклы\* в языке C-Sharp . В этом разделе мы вспомним , как циклы оформляются с помощью блок-схемы , а также запрограммилы в языке C-Sharp . В этом разделе мы вспомним , как циклы оформляются с помощью блок-схемы , а также запрограммируем их на C-Sharp . Предлагаю привести вам 2-3 варианта использования циклов в реальной жизни . А я пока приведу свои примеры . Первый пример заключается в чтении непрочитанных сообщений электронной почты . В этом случае клиент подключается к серверу и извлекает сообщение , пока не дойдет до конца .  
  
# Абзац 74 :  
Второй пример , друзья , это сканирование антивирусных файлов на предмет наличия в них вредоносного кода . В данном случае антивирус сканирует указанный каталог до тех пор , пока не переберет все файлы . Ну и третий пример – это поисковые задачи , в которых требуется найти объект в каком-либо хранилище . В этом случае \*программа\* , которая осуществляет поиск целевого объекта , переберет их всех , пока не найдет интересующий нас объект или не дойдет до конца .  
  
# Абзац 75 :  
Итак , друзья , давайте рассмотрим базовую задачу на применение циклических конструкций . В этом примере на вход подается некоторое натуральное число n и требуется вывести на экран все числа , которые находятся в промежутке от 1 до n. Для начала предлагаю сформулировать основные этапы этой задачи , затем мы с вами построим блок схему , после чего оживим ее на языке C-sharp .  
  
# Абзац 76 :  
Первым этапом мы вводим натуральное число n. Далее нам каким-то образом нужно осуществить проход по числам от 1 и до n и вывести их на экран . Друзья , как вы думаете , какой тип алгоритма здесь нам поможет ? Совершенно верно , если вашим предположением был цикл . В случае , если количество итераций или же повторяемых действий заранее неизвестно , или превышает количество повторений 2-3 , в этом случае рекомендуется использовать именно этот вид алгоритма . В следующем этапе давайте построим блок-схему для этого алгоритма . Для этого , друзья , необходимо определиться с составляющими элементами цикла . Напомню , первый элемент – это повторяемое действие , то есть то , что мы хотим повторять на каждой \*итерации\* .  
  
# Абзац 77 :  
Второй элемент цикла это некоторые вспомогательные переменные . Как правило , это переменная \*счетчик\* , но могут быть и другие . И третий компонент это самоусловие цикла , то есть выражение типа bool или же логического типа , которое определяет будет выполняться очередная итерация или нет . Друзья , в течение двух-трех минут предлагаю вам самостоятельно заполнить элементы для этой блок-схемы , а затем проверим ваши наработки .  
  
# Абзац 78 :  
you you you you you you you you Итак , давайте посмотрим , что у вас получилось . В качестве повторяемого действия мы будем использовать вывод на экран переменной i , которая , в свою очередь , является той самой вспомогательной переменной счетчика . Изначально , обратите внимание , она равна единице и в теле цикла она каждый раз увеличивается на единицу . Условием цикла будет неравенство переменная i меньше либо равна n , то есть неравенство , которое позволит нам вовремя остановить цикл в тот момент , когда переменная i дойдет до значения переменной n. Как только это произойдет и i сравняется с n , у нас выполнится последняя итерация цикла и на этом он остановится . Алгоритм решения задачи готов . Давайте попробуем оживить его на языке C-sharp .  
  
# Абзац 79 :  
Итак , вернемся в редактор . В каталоге lesson1 создадим каталог для нового задания . Перейдем в него и создадим там проект .  
  
# Абзац 80 :  
Перейдем в файл со сходным кодом И согласно блок-схеме создадим там переменную n Допустим , равное числу 100 То есть это будет верхний предел для количества итераций Далее создадим переменную счетчик i Изначально она равна 1 И далее , друзья , нам нужно создать цикл на языке C-sharp . Для создания цикла в большинстве языков программирования можно использовать оператор \*while\* . После этого ключевого слова в круглых скобочках , так же как и в условной конструкции , мы указываем условие . То есть то условие , которое будет определять , выполняется ли следующая операция или нет . Согласно блок-схеме напишем условие для этого цикла . i меньше либо равно n. Пока выделены условия , то есть выражение типа bool верно , цикл будет выполняться . И очередная итерация будет запущена . В фигурных скобочках мы указываем тело цикла , то есть те операции , которые будут выполняться на каждом повторении этого цикла . Согласно блок-схеме в цикле нам необходимо вывести значение переменной i. Сделаем это с помощью метода консолы Brightline . И передадим в качестве аргумента переменную i. Не забываем перейти к следующему значению счетчика и увеличить переменную i на единицу . Также , друзья , напомню вам про возможность автоматически отформатировать ваш код путем выделения его , нажатия правой кнопки и выбора меню формат документ . Он сам расставит необходимые пробелы , а также перенос на новую строчку , где это нужно . Давайте запустим наш код , посмотрим , что получилось .  
  
# Абзац 81 :  
Итак , сохраним файл с исходным кодом program.cs и выполним команду в каталоге с проектом.NET RAN . Ожидаем увидеть список целых чисел от 1 до 100 . На экране мы видим числа , расположенные \*друг\* под другом от 1 до 100 , что говорит о правильном выполнении алгоритма .  
  
# Абзац 82 :  
Однако , друзья , не всегда такой вывод может быть удобен , особенно если количество чисел превышает 50 , 100 и так далее . Иногда требуется вывести числа в строку . Для этого , друзья , необходимо использовать другой метод вывода на консоль .  
  
# Абзац 83 :  
Давайте воспользуемся второй разновидностью вывода элементов на консоль . Для этого мы воспользуемся все тем же модулем консоли , только воспользуемся методом вwrite . Предыдущий метод давайте закомментируем , то есть его отключим .  
  
# Абзац 84 :  
Итак , друзья , давайте попробуем запустить наш код И посмотрим , как вывод будет выглядеть в этом случае Как вы можете видеть , числа выведены в строку Но между ними нет никаких разделителей Друзья , какие у вас есть предположения , как это можно сделать ? Друзья , какие у вас есть предположения , как это можно сделать ? Самый простой вариант это добавить пробел после вывода каждого числа . Для этого можем воспользоваться тем же методом консоли в write , только в качестве аргумента передать ему просто пробел . Напомню , что пробел мы можем указать в одинарных кавычках , то есть передавая значение типа char .  
  
# Абзац 85 :  
Итак , сохраним исходный код , запустим и посмотрим , каков будет результат .  
  
# Абзац 86 :  
Итак , мы видим вывод чисел в одну строку , они разделены пробелом , тем самым повышается их читаемость . Друзья , обращаю ваше внимание , что входной параметр n , который используется в нашем коде , можно менять . Его можно задать равным тысяче , сотне или миллиону . Плюс использования циклов заключается в том , что при изменении входных данных нет необходимости менять ни алгоритм , ни расширять объем кода .  
  
# Абзац 87 :  
Итак , сохраним наши наработки в репозитории git . Для этого проиндексируем целиком наш созданный каталог task5 . Сформируем соответствующий commit .  
  
# Абзац 88 :  
Не забываем указывать имя команды git и проверяем правильность указываемых ключей . Отправим наши наработки в удаленной репозитории . И на этом практическая часть нашей сегодняшней лекции заканчивается . Ну что , друзья , на этом первая лекция подошла к концу . Давайте подведем итоги сегодняшнего занятия . Напомним , мы рассмотрели важные характеристики языка C-Sharp , которые позволили его выбрать в качестве основного инструмента для решения задач на текущем курсе . Напомним , это его статическая типизация и возможность создавать простые программы , не перегружая их сложным синтаксисом . Затем мы настроили и проверили работоспособность окружения для разработки программ на C-Sharp . Это платформа.NET и редактор Visual Studio Code с необходимыми расширениями . После чего мы решили блок задач на работу с переменными , арифметикой , условиями и циклическими конструкциями . На этом , друзья , я с вами прощаюсь . Благодарю за внимание . Увидимся на семинарах и на лекции №2 . you