Иосиф Дзеранов

ОСНОВЫ ПРОГРАММИРОВАНИЯ

Базовые понятия для новичков

C#

типы данных

логика

начальный уровень

строки

функции

условия

циклы

Microsoft

операторы

{ }

Visual Studio

алгоритм

массивы

переменные

{Hello, World!}

Иосиф Дзеранов

Основы программирования. Базовые понятия для новичков

«Автор»
2023

Дзеранов И.

Основы программирования. Базовые понятия для новичков /
И. Дзеранов — «Автор», 2023

Автор этой книги - Иосиф Дзеранов, инженер-разработчик со стажем более 10 лет, сооснователь и преподаватель онлайн-школы BeeGeek, четвертьфиналист чемпионата мира по олимпиадному программированию ACM ICPC.Эта книга - увлекательное путешествие в мир программирования. Вы начнете с базовых понятий и основных конструкций, перейдете к интересным задачам и кейсам, а к концу книги сможете создать свой собственный мини-проект. Книга подходит для любого уровня подготовки.

Содержание

Иосиф Дзеранов
Основы программирования. Базовые понятия для новичков

Об авторе

Всем привет!

Меня зовут Иосиф Дзеранов. Я программист, инженер-разработчик со стажем более 10 лет. Работал в крупных компаниях (Сбербанк, Mail.ru). Прошел путь от начинающего до старшего разработчика. Я знаю, как добиться такого же успеха.

Коротко про мои достижения:

• Преподаватель и сооснователь онлайн-школы BeeGeek.
• Сертифицированный преподаватель IT школы Samsung
• Четверть финалист чемпионата мира по олимпиадному программированию ACM ICPC
• Создатель центра олимпиадной подготовки СОГУ
• Победитель VK FELLOWSHIP 2020
• Победитель в конкурс “Умник” при фонде содействия инновациям
• Создатель более 25 онлайн курсов по программированию и информатике
• Семьянин. Есть двое прекрасный детей

Контакты для связи:

Почта: iodzeranov@mail.ru

Telegram: (t.me/JosefDzeranov).

Кому адресована эта книга

Эта книга отлично подойдет как новичкам, так и программистам с любым уровнем подготовки.

Первым этот курс поможет разобраться с основами программирования и определиться с дальнейшей деятельностью в IT-сфере; вторым – обогатить багаж знаний и отработать полученные навыки на практике.

Как читать эту книгу

Данная книга является печатной версией онлайн курса.

Обучающие модули расположены от простого к сложному, что предполагает последовательное и вдумчивое чтение.

Программирование требует практики, поэтому для чтения этой книги понадобится компьютер или ноутбук – так вы сможете сразу отрабатывать теоретические навыки на практических упражнениях.

Книга также может быть использована в качестве справочника для того, чтобы освежить знания в определенной теме.

Благодарности

Написание книги – это сложный и трудоемкий процесс, который отнял у меня, а значит и у моей семьи, большое количество времени и сил.

Я безмерно благодарен своим любимым девочкам – супруге Элизе и дочке Анне – и любимому сыну Лео. Благодаря им у меня есть желание заниматься любимым делом, развиваться и становиться лучше.

Спасибо моим родителям, которые всегда верили в меня и поддерживали во всех начинаниях.

Также хочется сказать спасибо всему моему окружению, ведь окружение во многом определяет, кем мы становимся и куда двигаемся.

И благодарю всех моих учеников, которые мотивировали и вдохновляли меня выпустить эту книгу.

Спасибо!

Сообщество разработчиков C#

Много времени размышлял о том, как сделать удобный формат общения с моими учениками. Создать так называемую "движуху", чтобы участники:

• получали максимальный быстрый ответ на свои вопросы

• делились знаниями между собой и создавали свою сеть знакомых с такими же интересами

• мотивировались, смотря друг на друга

• делали совместные проекты

•могли вместе готовиться к собеседованиям

Чтобы я

• смог доносить новую и полезную информацию до учеников

• максимально быстро и удобно

• смог делать анонс новых курсов и мероприятий максимально

•быстро и максимально эффективно.

Для этого я создал:

1. https://t.me/csharp_publics – личный канал для публикаций полезного контента

2. https://t.me/csharp_discussions – чат для обсуждения вопросов и полезного контента, который был опубликован в личном канале.

Если ты хочешь

• задать вопрос по теме программирования

•делиться знаниями с другими участниками

• сделать проект и хочешь найти коллегу

• быть в теме программирования, алгоритмов и структур данных, собеседований

– то смело ВСТУПАЙ В КАНАЛЫ.

Со своей стороны обещаю годный контент (если будет обратная связь) и никакого спама (всех спамеров и тех, кто будет писать не по теме буду удалять). Жду тебя. Ты сможешь задать любой вопрос.

Приглашайте всех друзей, знакомых, учеников, кто интересуется программированием!

До встречи!

1.
Общая информация

В этой книге я научу Вас программировать. Сделаем первый шаг: изучим все базовые конструкции, которые нужны каждому программисту на любом языке программирования (ЯП).

Книга состоит из 6 модулей:

1. Типы данных. Переменные;
2. Условный оператор;
3. Оператор цикла;
4. Строки и символы;
5. Массивы;
6. Функции.

Это то, что должен знать любой программист. Если представить, что дом – это старший программист, то эта книга – фундамент.

Модули расположены именно в том порядке, в котором нужно проходить: от простого к сложному. Не советую менять порядок изучения тем, так как информация наслаивается друг на друга.

Формула, которая помогает добиваться целей:

Постоянство усилий и регулярность занятий гарантируют результат.

Уровень вашей нынешней подготовки неважен, потому что вся необходимая теория есть в книге. Объем практики позволяет теорию проработать и запомнить. А самое главное – применить ее на практике.

Книга позволит с легкостью и удовольствием выучить синтаксис языка программирования C#, отработать какие-то непростые вещи до автоматизма, написать первые мини-проекты.

В то же время подчеркну, что книгу можно проходить на любом удобном вам ЯП, эта возможность в нем предусмотрена. Так что для решения задач ЯП вы можете выбрать сами.

Дорогу осилит идущий!

1.3

Введение

Зачем изучать программирование?

Во-первых, это интересно.

Во-вторых, программирование здорово облегчает жизнь во многих профессиях.

В-третьих, можно хорошо зарабатывать и заниматься в тёплых уютных офисах современными технологиями.

Как изучать программирование?

Самое сложное в нашей области – это первые шаги. Я научу вас базовым вещам в программировании. Мы напишем сотни несложных и интересных программ, а также будем строить свои собственные алгоритмы. С таким опытом в дальнейшем вы сможете самостоятельно справиться и с более сложными задачами.

Чему учит книга?

После этой книги вы сможете решать задачи по программированию, выиграть несложную олимпиаду по информатике, а также начать свой профессиональный путь в качестве программиста.

Язык программирования

Мы будем изучать язык программирования C#, так как, по моему мнению, язык очень прост и хорошо подходит для новичков. На сегодняшний момент C# один из самых мощных, быстро развивающихся и востребованных языков в ИТ-отрасли. На нем пишутся самые различные приложения: от небольших десктопных программ до крупных веб-порталов и веб-сервисов, обслуживающих ежедневно миллионы пользователей.

Свобода выбора

Однако данная книга не привязывается к одному языку программирования, так как в ней проходят базовые понятия и конструкции, которые включают в себя все современные языки. Мы не будем вас обязывать сдавать задачи на языке C#, вы будете сами выбирать, на каком языке сдавать задачи.

Как построена книга?

Она состоит из лекций, к каждой из которых есть набор задач – их вам предстоит решать в тестирующей системы. Задачи будут проверяться автоматически и сразу.

1.4

Основные понятия

Алгоритм – это последовательность действий для достижения поставленной цели. Это план наших действий.

Язык программирования – это язык, который понимает компьютер. Есть русский, английский и другие языки, которые понимаем мы, а есть другие языки – языки программирования, которые понимает компьютер.

Программа – алгоритм, записанный на некотором языке программирования.

Этапы решения задачи

Теперь давайте разберем этапы решения задачи:

1. Постановка задачи – описание самой задачи;
2. Формализация – перевод на математический язык;
3. Алгоритмизация – придумывание алгоритма, который решает задачу;
4. Программирование – написание кода программы;
5. Тестирование – проверка работоспособности программы и корректности выходных данных.

Начинающие программисты пытаются пропустить последний этап. Я не советую так делать. Стоит сначала проверить самому программу на корректность работы, а потом сдавать в тестирующую систему. Постарайтесь придумать такие тесты, которые учитывают все различные ситуации.

Настоятельно советую всегда решать задачи по этим этапам. Так вы сэкономите массу времени и напишете правильные программы без ошибок.

Среда разработки

Среда разработки (IDE) позволяет писать код, компилировать (собирать у компилятора в "голове") его и отлаживать (искать ошибки). Она удобна тем, что в ней есть множество встроенных функций, которые, например, за вас заполнят стандартные строки кода. IDE подчеркнет ошибки синтаксиса и покажет причину, по которой программа не работает. Среда разработки помогает найти эту ошибку с помощью отладки.

При использовании языка C# я рекомендую среду разработки Visual Studio.

Установка Visual Studio

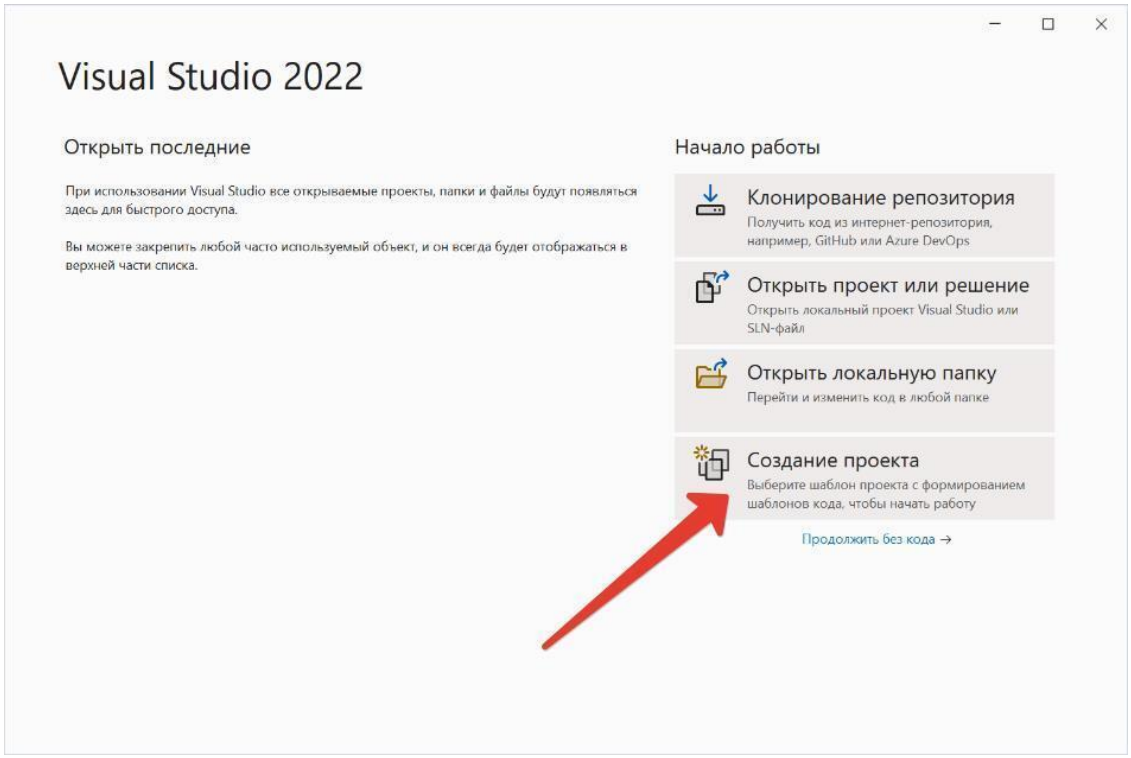
Во время установки программы нужно выбирать версию Community Edition. Это бесплатный инструмент для некоммерческого использования.

Когда откроется окно выбора компонентов, то единственный флажок нужно поставить у компонента "Разработка классических приложений .Net".

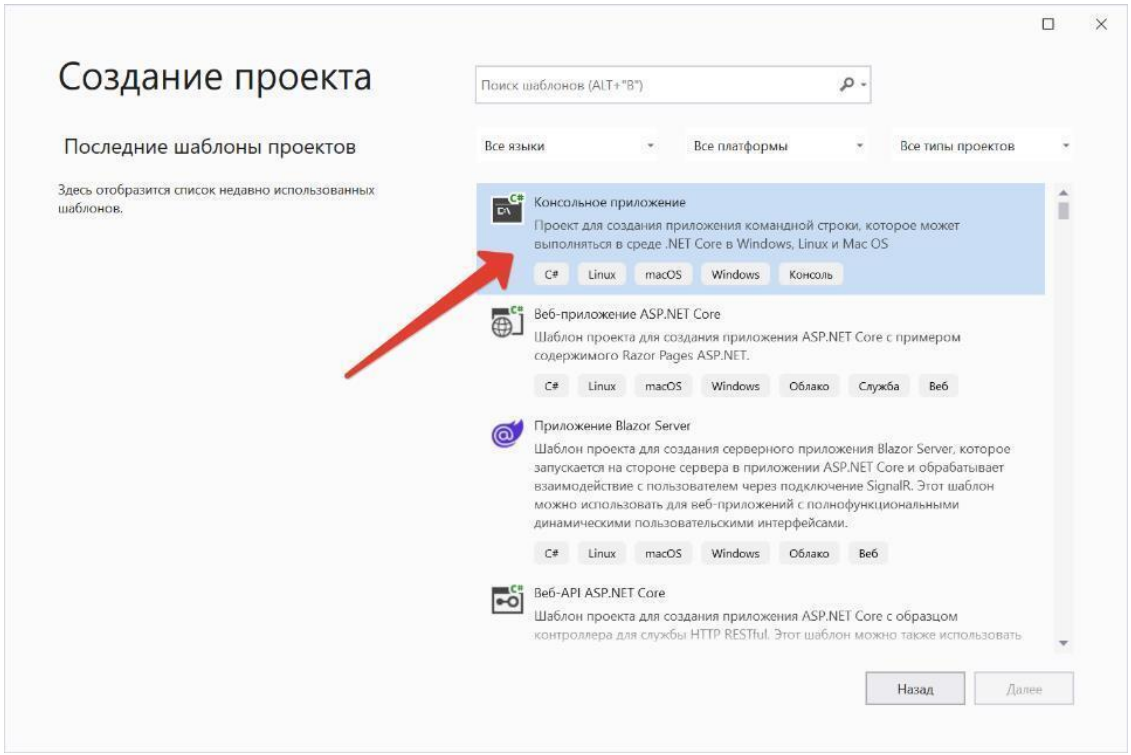
Этого достаточно, если вы совсем новичок. Компоненты можно будет добавить по мере необходимости.

Создание проекта

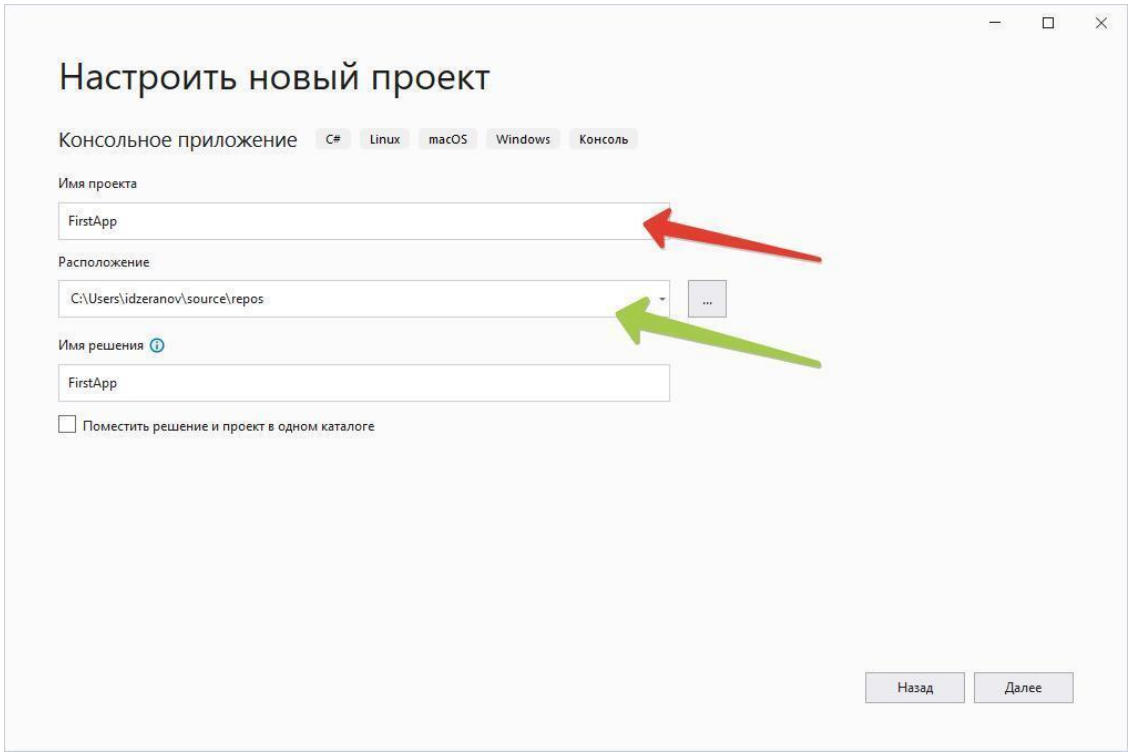
Запустите Visual Studio и нажмите «Создание проекта»:



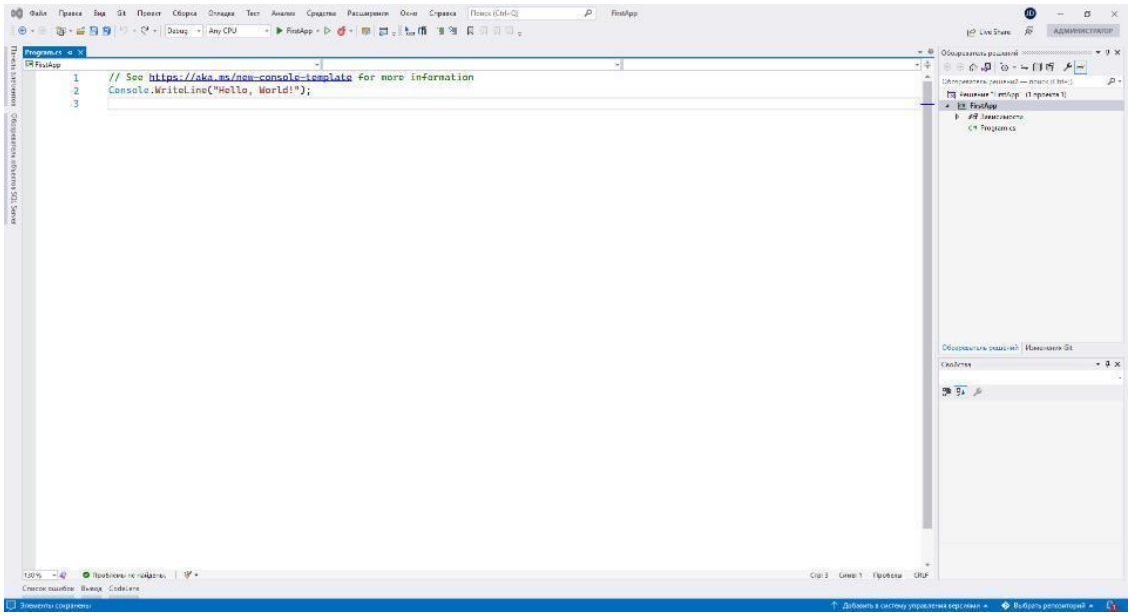
В появившемся меню выберите «Консольное приложение:



Затем назовите проект на английском языке в поле «Имя проекта» (показано красной стрелкой), укажите куда нужно сохранять проект в «Расположение» (указано зеленой стрелкой) и нажимайте "Далее":



Первая программа
Visual Studio сгенерировала следующий шаблон кода:



Замените содержимое файла на:

```
using System;
namespace HelloWorld
{
 class Program
 {
  static void Main(string[] args)
```

```
  {
// Здесь будем писать код программы
  }
 }
}
```

И это уже готовая программа, только она не делает ничего.

Не углубляясь в подробности, посмотрим, из чего состоит программа. В первой строке записана команда using. Данная команда означает, что мы хотим обратиться к библиотеке, содержащей разные полезные вещи, которые написали за нас другие программисты. В данном случае есть библиотека System, с помощью которой наша программа сможет общаться с внешним миром.

Перейдем сразу к тому, что нам точно понадобится.

```
static void Main(string[] args)
{
        // Здесь будем писать код программы
}
```

Main – это основная функция нашей программы, она будет запускаться автоматически при запуске программы. Когда-нибудь в наших программах будет несколько функций, и тогда первой из них будет запускаться Main. Пока же это основная и единственная наша функция. Внутри фигурных скобок мы будем писать свои программы. Вся логика программ будет начинаться после открывающей фигурной скобки и заканчиваться – закрывающей фигурной скобкой.

Сейчас внутри фигурных скобок содержится комментарий. Он начинается с //, после чего до конца строки можно писать пояснения к программе. Комментарии не влияют на логику программы. Они нужны для комментирования кода. В будущем на месте данного комментария мы будем писать осмысленную программу.

Hello, World!

В программировании есть такая традиция – изучение любого языка начинается с написания программы, которая выводит на экран сообщение «Hello, World!»:

```
using System;
namespace HelloWorld
{
class Program
{
static void Main(string[] args)
{
Console.Write("Hello, World!");
}
}
}
```

У нас появилась новая строка:

```
Console.Write("Hello, World!");
```

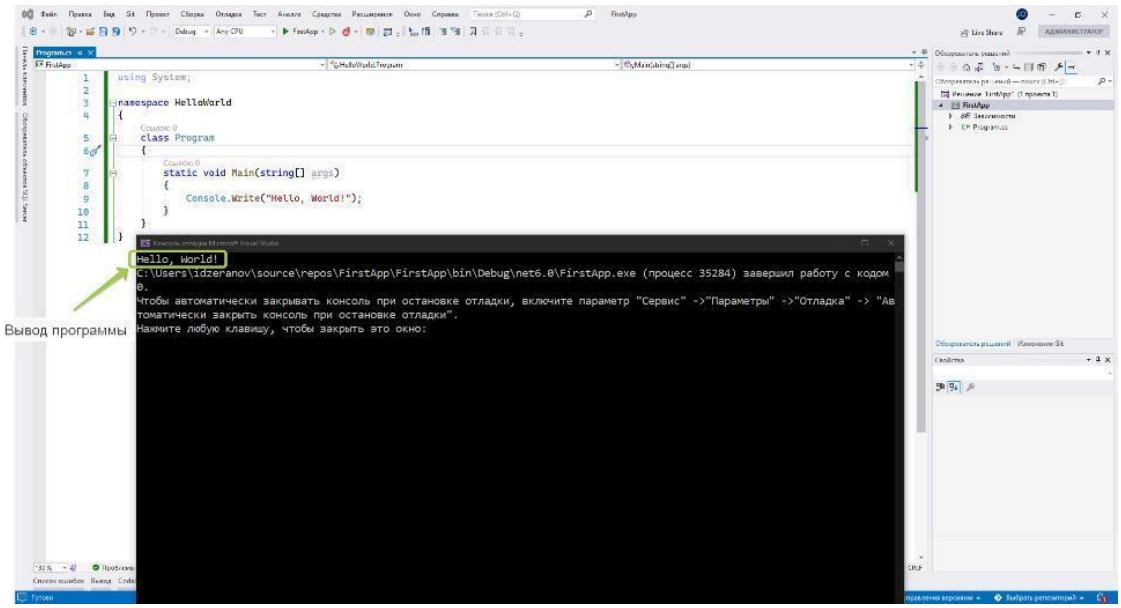
Console – это объект, который отвечает за консоль. Консоль – это черное окошко, которое появляется при запуске программы. Операция Write как раз и означает написать. «Console Write» переводится «Написать на консоль».

Текст для вывода на экран идет внутри круглых скобок и в двойных кавычках. Не забывайте ставить точку с запятой (;) там, где она есть в примерах кода. Почти после всех операторов в языке C# ставится точка с запятой (;).

Следует помнить:
Строка выведется ровно такая,
как указали внутри кавычек,
со всеми пробелами и символами.

Запуск программы

Давайте запустим нашу программу. Для этого нажмем комбинацию клавиш CTRL и F5. Если в программе есть ошибка, система сообщит нам, что запуск не удался. Ошибки будут перечислены в окне «Список ошибок». Если всё правильно, то появится чёрное окно консоли с надписью "Hello, World!":



Чтобы закрыть консоль, необходимо нажать любую клавишу.

2.
Типы данных. Переменные

2.1

Ввод-вывод информации. Типы данных

Console.WriteLine

Мы уже знаем, что выводить строки можно с помощью команды Console.Write(). Есть еще похожая команда Console.WriteLine(). Она не только выводит строку,

но и переводит после этого курсор в консоли на новую строку.

Для лучшего понимания рассмотрим два примера.

```
static void Main(string[] args)
{
Console.Write("Hello"); // вывод
Console.Write("World!"); // вывод
}
```

Получим строку:

HelloWorld!

Дело в том, что после команды

Console.Write("Hello");

курсор в консоли остается на той же строке. Туда же вторая команда дописывает строку "World!". Таким образом и получается склеенная строка.

Запомните:

// – это комментарий. Это текст, который

не воспринимается и не выполняется программа. Это нужно нам, чтобы отметить дополнительную информацию о происходящем в коде.

А если используем Console.WriteLine():

```
static void Main(string[] args)
{
Console.WriteLine("Hello"); // вывод и перенос на новую строку
```

```
Console.WriteLine("World!"); // вывод и перенос на новую строку
}
```

Вывод будет следующим:

```
Hello
World!
```

Дело в том, что после команды

```
Console.WriteLine("Hello");
```

курсор передвинулся на следующую строку. И уже на новой строке выводится "World!".

Вывод информации

С помощью этих команд можно выводить числа или результат вычисления выражений. Например:

```
static void Main(string[] args)
{
    Console.WriteLine(26);
    Console.WriteLine(1024 + 48); // 1072
   Console.WriteLine(5 + 8 * 2); // 21
   Console.Write((5 + 8) * 2); // 26
}
```

Через // указано то, что будет после исполнения команды. То есть вывод будет следующим:

```
26
1072
21
26
```

Как вы могли заметить, каждый результат выводится на каждой строке. При этом операции над числами внутри команды осуществляются по правилам математики: числа складываются, умножаются и так далее.

Форматированный вывод

Довольно часто есть необходимость выводить числа и строки вместе. Просто вывод чисел для пользователя ничего не означает. Иногда надо бывает перед или после числа добавить объяснения.

Для этого строку нужно обозначить в кавычках (чтоб компилятор ее "узнал" как строку), а между числом и строкой нужно поставить знак плюс (+), который тут просто приклеивает строку и число, а не считает математически. Результатом такой склейки получается СТРОКА.

Для лучшего понимания рассмотрим код:

```
static void Main(string[] args)
{
Console.WriteLine("5 + 7 = " + 12); // 5 + 7 = 12
Console.WriteLine(5 + " + " + 7 + " = " + 12); // 5 + 7 = 12
Console.WriteLine(3 + " " + 6); // вывод двух чисел через пробел, то есть 3 6
}
```

Заметьте, что пробел (" ") – это тоже строка, причем не пустая. Пример показывает, что один и тот же результат вывода можно достичь разными способами.

Заметьте, что как при сложении числа и строки, так и при сложении строки и числа, результатом будет строка.

Примечания:

1. То, что мы пишем в круглых скобках у команд Console.Write() и Console.WriteLine(),называется аргументами или параметрами команды.

2. Все команды, которые мы уже рассмотрели и которые в будущем рассмотрим, записываются в приведенном в лекциях формате, другое написание недопустимо, так как в C# строчные и заглавные буквы различны.

3. Заметьте, что в конце строки пишется точка с запятой (;). Это обязательно. Такой синтаксис языка C#.

4. Команда Console.WriteLine() с пустым списком аргументов (пустые круглые скобки) просто вставляет новую пустую строку. Например:

Console.WriteLine("Строка 1");
Console.WriteLine();
Console.WriteLine("Строка 3");
выведет на экран три строки, одна из которых пустая:
Строка 1
Строка 3
Здравствуй, Иосиф!
А теперь пришло время поздороваться со мной.
Напишите программу, выводящую следующий текст:
Здравствуй,
Иосиф!
Заметьте, что выводятся две строки.
Примечание:

1. Обратите внимание, что каждая последующая команда Console.WriteLine() выводит указанный текст, начиная с новой строки.

2. Для решения задачи обязательно нужно выбрать язык программирования. Я показываю все на C#. Если нет окна с перечнем языков программирования, то перезагрузите страницу/программу.

3. Комментарий, который автоматически пишется при выборе языка C#, можно убрать ради чистоты кода:

```
1 using System;
2
3 public class MainClass
4 {
5     public static void Main()
6     {
7         // put your c# code here
8     }
9 }
```

4. Язык C# регистрозависимый. Нужно писать команды точно такие, какие были в лекции.

5. Не забудьте в конце каждой строки поставить точку с запятой (;).

Проверяющая система будет сравнивать результат вашей программы и правильный ответ посимвольно. Пробел – тоже символ. Это означает, что выводить нужно ровно такую строку, которая указано в условии задачи.

Самые частые ошибки компиляции: забыли поставить запятую, пробел, восклицательный знак или неправильно написали регистр букв. Вам нужно вывести текст точно такой как в описании задачи.

Совет: лучше всего его скопировать.

2.2

Переменная

Определение

Программа, которая всегда считает результат вычисления одного и того же выражения, довольно скучная и бессмысленная. Полезная программа должна оперировать с различными данными без внесения изменений в код.

Для хранения информации в программировании используются переменные. Переменную можно рассматривать как ящик. Однажды сделав такой ящик, мы можем класть в него разные вещи.

Под каждый тип информации – нужен ящик соответствующего типа: вы ведь не будете складывать деньги, спички, бензин и шоколад в один и тот же ящик. Таким образом, у каждой переменной есть тип данных, который надо указать при ее создании. Также у переменной есть имя, по которому мы будем к ней обращаться.

Запомните:
Переменная – это ячейка памяти
определенного типа данных,
имеющая имя.

Работа с переменной

Для создания или объявления переменной нужно воспользоваться следующей схемой:

тип_данных название переменной;

Тип данных, отвечающий за хранение целых чисел, называется int. Следовательно, чтобы создать переменную, которая будет хранить целые числа, нужно написать:

int a;

Это мы объявили переменную с названием a и указали, что там будут хранится целые числа. Теперь в созданную переменную можем записывать только целые числа.

Чтобы записать в переменную целое число, нужно воспользоваться следующим правилом:

куда = что

где

• куда: в какую переменную записать данные

• =: оператор присвоения

• что: какие данные записать

Воспользуемся данным правилом. Например, запишем в ранее созданную переменную a число 7:

a = 7;

Операции производятся справа налево – взять число 7 и записать в переменную a.

Эти два шага можно объединить, то есть можно сразу объявить переменную и записать в нее значение (инициализировать), иначе говоря – присвоить начальное значение:

int a = 7;

Мы объявили переменную a и сразу записали значение 7.

Чаще всего так и делают: сразу объявляют переменную и присваивают начальное значение!

Чтобы узнать содержимое переменной, нужно обратиться к ней по имени. Например:

int b = a * 5;

В переменную b запишется значение 35, так как вместо переменной a подставится ее значение, то есть 7. Напомню, что сначала выполняется выражение справа от равно (a * 5), а потом результат вычисления записывается в новую переменную b.

Также мы можем поменять значение уже существующей переменной a:

a = a + 8;

Так как действия выполняются справа налево от знака =, следовательно, мы берем значение переменной a, которое равно 7, к нему добавляем 8 и снова записываем в переменную a. Таким образом, значение переменной увеличили на 8.

Запомните:

Переменную создают один раз, указав тип данных, название и начальное значение. При изменении значения переменной, нужно указать только название. Тип данных указывать больше не нужно.

Для лучшего понимания разберем пример с ошибкой:

```
int a = 6; // создали переменную a
int b = a * 8; // 48
int a = b – 8; // ошибка. Переменная a уже существует.
int c = b + a;
Console.WriteLine(c);
```

А вот исправленная программа:

```
int a = 6;
```

```
int b = a * 8; // 48
a = b – 8; // 40. Заметьте, что тип данных
мы не написали
int c = b + a; // 88
Console.WriteLine(c); // вывод 88
```

Типы данных

Итак, мы уже посмотрели и поработали с целочисленным типом данных int. Но существуют еще и другие типы данных. Давайте рассмотрим основные часто используемые типы данных:

- int – целое число от -2147483648 до 2147483647;
- long – целое число от –9 223 372 036 854 775 808
до 9 223 372 036 854 775 807;
- string – строка;
- double – число с дробной частью (вещественные числа);
- char – символ;
- bool – специальный тип, принимает только два значения: true или false.

Рассмотрим пример:

- Создадим переменную строкового типа name и запишем в нее строку "Олег". Напомню, что строку нужно обрамлять двойными кавычками:

```
string name = "Олег";
```

- Создадим еще одну переменную целочисленного типа age и запишем в нее значение 2525:

```
• string name = "Олег";
int age = 25;
```

- Создадим еще одну переменную вещественного типа weight и запишем в нее значение 80.5:

```
• string name = "Олег";
• int age = 25;
• double weight = 80.5;
```

Стоит отметить, что при инициализации вещественных чисел дробная часть отделяется от целой точкой.

- Теперь мы можем обращаться к переменным по имени и вместо них подставятся их значения:

```
• string name = "Олег";
• int age = 25;
• double weight = 80.5;
• string stringToShow = name + ", возраст " + age + ", вес " + weight;
Console.WriteLine(stringToShow);
```

Запятые я поставил просто для красоты отображения. Они стоит внутри строки. Все что стоит внутри строки выводится в таком же виде. То есть программа выведет:

Олег, возраст 25, вес 80,5

Запомните:
C# строго типизированный язык программирования. От типа данных зависит, какая информация будет храниться в переменной.

Рассмотрим подробнее. В переменную целого типа можно записывать только целое число:

```
int a = 10;
```

Нельзя записать в нее строку или любой другой тип данных:

```
int a = "10"; // ошибка!
```

То же самое, например, с переменной строкового типа. В нее можно записать только строку:

```
string s = "test"; // верно
string t = 10; // ошибка!
```

Запомните:
От типа данных переменной зависит поведение операций над ним.

Например, для строк знак плюс (+) означает склеивание между собой, а для целых и вещественных чисел плюс складывает их математически:

```
int a = 5;
int b = 10;
Console.WriteLine(a + b); // 15
string s1 = "5";
string s2 = "10";
Console.WriteLine(s1 + s2); // 510
```

Имя переменной

1. В имени переменной используйте только латинские буквы a-z, A-Z, цифры и символ нижнего подчеркивания (_);

2. Имя переменной не может начинаться с цифры;

3. Имя переменной по возможности должно отражать её назначение. Это нужно только нам, программистам, чтобы наш код читался. Программе все равно, какое название будет у переменной.

Запомните:

C# – регистрочувствительный язык.
Переменная name и Name – две совершенно разные переменные. Принято название переменных начинать с маленькой буквы.

Примечания:

1. Переменные можно вводить в любой момент (не только в самом начале программы).

2. Названия переменных должны быть уникальными. Если у двух переменных будут одинаковые имена, то непонятно будет, какое значение вернется при обращении к переменной.

3. Значение перезаписывается. Новое значение переменной вытесняет старое. Важно понимать, чему равно значение переменной в каждый момент времени:

```
int a = 6; // здесь a равен 6
int b = a * 8;
a = b – 8; // 40. Здесь уже a равен 40
int c = b + a;
Console.WriteLine(c);
```

1. Можно сразу записывать в объявляемую переменную формулу:

```
int a = 22 * 4 + 1; // 89
int b = a – 15 * 3; // 44
```

2. Напомню, что мы все написанные программы пишем внутри Main.

2.3

Ввод данных

Ввод информации

Проблема

Все предыдущие программы выводили на экран текст, известный в момент написания программного кода. То есть при каждом запуске программы, вывод был один и тот же.

Например:

```
Console.WriteLine("Привет! Как тебя зовут?");
string name = "Иосиф";
Console.WriteLine("Привет, " + name);
```

Данная программа всегда будет выводить одно и тоже:

Привет! Как тебя зовут?

Привет, Иосиф

Это статические программы, которые никому не интересны. Всех интересует программы, которые на основе введенных данных пользователем выдают результат. То есть, программы могут работать с данными, которые станут известны только во время выполнения программы. Другими словами, программы могут считывать данные от пользователя, а затем их использовать.

Решение

Для считывания данных в языке C# используется команда Console.ReadLine(). Рассмотрим пример:

```
Console.WriteLine("Привет! Как тебя зовут?");
string name = Console.ReadLine(); // пользователь вводит свое имя. Сохраняем в переменную "name"
Console.WriteLine("Привет, " + name); // приветствуем пользователя. Вместо переменной подставиться его значение, то есть то что ввел пользователь.
```

Сначала программа распечатает текст на экран «Привет! Как тебя зовут?». Далее программа будет ждать от пользователя ввода данных. Ввод данных реализуется с помощью команды Console.ReadLine().

Она работает так: когда программа доходит до места, где есть Console.ReadLine(), она ждет, пока пользователь введёт текст с клавиатуры (ввод завершается нажатием клавиши Enter). Введенная пользователем строка подставляется на место Console.ReadLine().

То есть, если вы ввели строку «Иосиф», программа дальше будет работать так, как будто на месте Console.ReadLine() было написано «Иосиф». Если вы ввели строку «Вася», программа дальше будет работать так, как будто на месте Console.ReadLine() было написано «Вася». То есть все зависит от того, что введет пользователь.

Запомните:

Console.ReadLine() получает от пользователя какие-то данные в виде строки (string) и записывает в строковую переменную name.

Обратите внимание, что просто команды Console.ReadLine() недостаточно, чтобы считать строку с консоли. Нужно ее сохранить в переменную, чтобы в дальнейшем ее использовать. Например: Console.WriteLine("Привет! Как тебя зовут?");

Console.ReadLine();

На второй строке мы просим пользователя ввести строку. Но после этого никуда ее не сохранили. Спрашивается, а зачем тогда просили? Это бессмысленная операция.

Надо обязательно сохранить в переменную строкового типа: Console.WriteLine("Привет! Как тебя зовут?");

string name = Console.ReadLine();

А дальше мы можем обращаться к этой строке в программе по названию переменной name.

Если мы хотим, чтобы пользователь ввел n строк, то команд Console.ReadLine() в программе должно быть n штук.

Например, если пользователь вводит 3 строки, то для этого нужно написать:

string row1 = Console.ReadLine();

string row2 = Console.ReadLine();

Конец ознакомительного фрагмента.