Государственный университет – Высшая школа экономики

Факультет Бизнес-Информатики

Кафедра Основ информатики
и прикладного программного обеспечения

C#

Объектно-ориентированный язык программирования

Пособие к практическим занятиям - №2

Проф. Забудский Е.И.

Москва 2005

Тема 2. Изучение новых элементов языка C# на примере исходного кода, являющегося несложным калькулятором

Одно практическое занятие
(2 часа)

Рассматривается пример исходного кода программы, являющейся несложным калькулятором. В нем демонстрируются новые элементы языка C#: создание и применение пользовательских классов; пространство имен, позволяющее организовывать классы и другие типы в единую согласованную иерархическую структуру, обеспечивающую простой доступ к данным; преобразование значения типа string в тип int; удобный класс Math и др.

// **КОММЕНТАРИЙ**. На сайте http://www.firststeps.ru/ “Первые шаги” представлено много интересных обучающих материалов по различным интегрированным средам и языкам программирования, в том числе представлены C# и платформа .NET (step by step).

Содержание

Рассматривается пример исходного кода программы, являющейся несложным калькулятором. В нем демонстрируются новые элементы языка C#.

1. ProjectSimpleCalculator. Программа class.cs. Среда C#Building

В исходном коде на листинге 3.1 представлена программа, вычисляющая сумму, произведение, максимум и минимум двух чисел, введенных пользователем. Ответ выводится на консоль.

Листинг 3.1. Исходный код SimpleCalculator.cs

```
01: using System;
02:
03: namespace ProjectSimpleCalculator
04: {
05: /*
06:  * Данный класс определяет сумму, произведение,
07:  * минимум и максимум двух чисел
08:  */
09: public class SimpleCalculator
10: {
11:     public static void Main() // Мэтт Вайсфельд
12:     {
13:      int x;
14:      int y;                                ,
15:
16:      Console.Write("Введите первое число: ");
17:      x = Convert.ToInt32(Console.ReadLine());
18:      Console.Write("Введите второе число: ");
19:      y = Convert.ToInt32(Console.ReadLine());
20:      Console.WriteLine("Сумма: " + Sum(x, y));
21:      Console.WriteLine("Произведение: " + Product(x, y));
22:      Console.WriteLine("Максимальное число: " + Math.Max(x, y));
23:      Console.WriteLine("Минимальное число:" + Math.Min(x, y));
24:      Console.ReadLine();
25:      }
26:      // Метод Sum вычисляет сумму двух чисел типа int
27:     public static int Sum(int a, int b)
28:     {
29:      int sumTotal;
30:
31:      sumTotal = a + b;
32:      return sumTotal;
33:     }
34:
35:      // Метод Product вычисляет сумму двух чисел типа int
36:     public static int Product(int a, int b)
37:     {
38:      int productTotal;
39:
```

```
40:       productTotal = a * b;
41:       return productTotal;
42:     }
43:  }
44:}
```

Вот пример вывода программы после того, как пользователь вводит числа 3 и 8:

Введите первое число: 3 <enter>
Введите второе число: 8< enter >
Сумма: 11
Произведение: 24
Максимальное число: 8
Минимальное число: 3

Краткое объяснение исходного кода из листинга 3.1 приведено в листинге 3.2. Важно научиться отличать фундаментальные элементы, присущие типичной программе на языке C#.

2. Листинг 3.2. Краткий анализ программы из листинга 3.1

01: Программе разрешено использовать краткие имена при обращении к классам в предопределенном пространстве имен System
02: Пустая строка
03: Ключевое слово namespace: объявление собственного пространства имен
04: Граница пространства имен ProjectSimpleCalculator
05: Начало многострочного комментария
06: Вторая строка многострочного комментария: Данный класс определяет сумму, произв-е,
07: Третья строка многострочного комментария: минимум и максимум двух чисел
08: Завершение многострочного комментария
09: Начало определения класса SimpleCalculator
10: Начало блока класса SimpleCalculator
11: Начало определения метода Main()
12: Начало блока метода Main()
13: Объявление целочисленной переменной x
14: Объявление целочисленной переменной y
15: Пустая строка
16: Вывод строки: Введите первое число:
17: Сохранение ответа пользователя в переменной x. Переход на одну строку вниз
18: Вывод строки: Введите второе число:
19: Сохранение ответа пользователя в переменной y. Переход на одну строку вниз
20: Вывод строки: Сумма: и результата метода Sum.
21: Вывод строки: Произведение: и результата метода Product.
22: Вывод строки: Максимальное число: и результата метода Max класса Math.
23: Вывод строки: Минимальное число: и результата метода Min класса Math
24: Конец блока метода Main()
25: Пустая строка
26: Комментарий: Метод Sum вычисляет сумму двух чисел типа int
27: Начало определения метода Sum(**int** a, **int** b)
28: Начало блока метода Sum(**int** a, **int** b)

29: Объявление локальной целочисленной переменной sumTotal.
30: Пустая строка
31: Нахождение суммы чисел a и b и сохранение результата в переменной sumTotal
32: Окончание метода и возврат значения sumTotal.
33: Конец блока метода Sum(**int** a, **int** b)
34: Пустая строка
35: Комментарий: Метод Product вычисляет произведение двух чисел типа int
36: Начало определения метода Product(**int** a, **int** b)
37: Начало блока метода Product(**int** a, **int** b)
38: Объявление локальной целочисленной переменной productTotal
39: Пустая строка
40: Нахождение произведения чисел a и b и сохранение результата в переменной productTotal
41: Окончание метода и возврат значения productTotal
42: Конец блока метода Product(**int** a, **int** b)
43: Конец блока класса SimpleCalculator
44: Граница пространства имен ProjectSimpleCalculator

3. Объявление переменной типа int

Строка 13 содержит оператор, объявляющий переменную x. Вспомним, как переменная типа string была объявлена для хранения символов (текста) в строке 10 листинга 2.1 (см. Пособие к практическому занятию №1). На этот раз ключевое слово string заменено на **int.** Оно указывает, что переменная x принадлежит специальному целочисленному типу и занимает в памяти 32 бита (4 байта – см. Приложение 2).

13: **int** x;

Переменная x, занимающая в памяти 32 бита, представляет числа в диапазоне от -2147483648 до 2147483647.

В строке 14 объявлена переменная y типа **int**

14: **int** y;

x и y обычно считаются неприемлемыми именами для переменных, поскольку идентификаторы должны быть значащими и отражать содержимое переменной. Однако в данном случае, x и y участвуют в общих арифметических операциях и не представляют каких-либо специальных значений, как, например, средние осадки и т.д. Очевидно, что x и y являются приемлемыми переменными в алгебраическом смысле.

4. Преобразование значения типа string в тип int

Ввод пользователя, завершенный нажатием Enter и полученный методом Console.ReadLine() в программе C#, принадлежит типу string. Даже число, например 432, изначально рассматривается как набор символов, такой же, как ABC или #@$.

Число, представленное строкой, нельзя сохранить как тип **int**. Вначале следует преобразовать переменную типа string в переменную типа **int**.

Чтобы преобразовать ввод пользователя в число, в строке 17 применяется метод ToInt32 класса Convert. Результат преобразования (справа от знака равенства в строке 17) является числом и присваивается переменной x. Естественно, значение, введенное пользо-

вателем, должно совпадать с целым числом. Ввод 57.53 вызывает ошибку, а значения 109 или 64732 допустимы.

```
17:     x = Convert.ToInt32(Console.ReadLine());
```

5. Создание и вызов пользовательских методов

Чтобы понять смысл строки 20, необходимо обратиться вначале к строкам 27-33.

```
27:     public static int Sum(int a, int b)
28:     {
29:     int sumTotal;
30:
31:     sumTotal = a + b;
32:     return sumTotal;
33:     }
```

До этого момента в программах использовались готовые методы, наподобие Console.ReadLine() и Console.WriteLine(). Несмотря на огромное количество встроенных методов **.NET Framework**, а также доступность других коммерческих библиотек классов, при создании собственного кода потребуются и собственные, определенные пользователем методы.

В строках 27-33 содержится пользовательский метод Sum:

При вызове метод Sum получает два числа. Он складывает их и возвращает результат в точку вызова.

Определение метода Sum() (заголовок, фигурные скобки { } и тело метода) следуют той же общей структуре, которой подчиняется метод Main.

// **СОВЕТ**

В общем случае классы представляют объекты, а методы — действия. Именно это следует отражать в их названиях: для именования классов (Car, Airplane и т.д.) используются имена существительные, а для именования методов (DriveForward, MoveLeft, TakeOff и т.д.) — глаголы.

Заголовок метода в строке 27 во многом похож на заголовок метода Main в строке 11 (напомним, что спецификатор доступности **public** включает метод в интерфейс класса, которому тот принадлежит).

В строке 8 листинга 2.1 (см. Пособие к практическому занятию №1) было также представлено ключевое слово static, которое пока не будет обсуждаться.

Далее, ключевое слово void заменено на int, а в скобках после Sum находятся конструкции, похожие на объявление переменных. Это требует некоторого объяснения,

Строка 27 представляет собой интерфейс метода Sum. Он отвечает за взаимодействие между методом, вызывающим Sum (в данном случае, Main), и телом Sum.

Иначе говоря, **интерфейс** — это свойство, позволяющее отдельным (и часто несовместимым) элементам эффективно взаимодействовать.

Процесс вызова метода Sum(**int** a, **int** b), объявленного в строке 27, из строки 20 метода Main() как Sum(x, y).

a и b в заголовке метода Sum называются формальными параметрами. Формальный параметр применяется в теле метода для обращения к тому значению, которое было сохранено в нем в момент вызова метода. В данном случае, аргументы x и y в строке 20 присваиваются параметрам a и b так же, как если бы были выполнены операторы присваивания a=x; и b=y;.

a и b используются в теле метода (в строке 31) как обычные объявленные переменные. Их значения равны x и y.

Ключевое слово int в строке 27, заменившее void, указывает, что метод Sum теперь возвращает значение, которое принадлежит типу int. Этому отвечает ключевое слово return в строке 30.

```
30:     return sumTotal;
```

return принадлежит к *операторам возврата управления*. Когда запускается такой оператор, выполнение метода прекращается. Поток выполнения возвращается в точку вызова и подставляет выражение, указанное после return (в данном случае, sumTotal).

Поэтому sumTotal должен быть совместим с типом, объявленным в заголовке метода (здесь int), как показано на рис. 2.1.

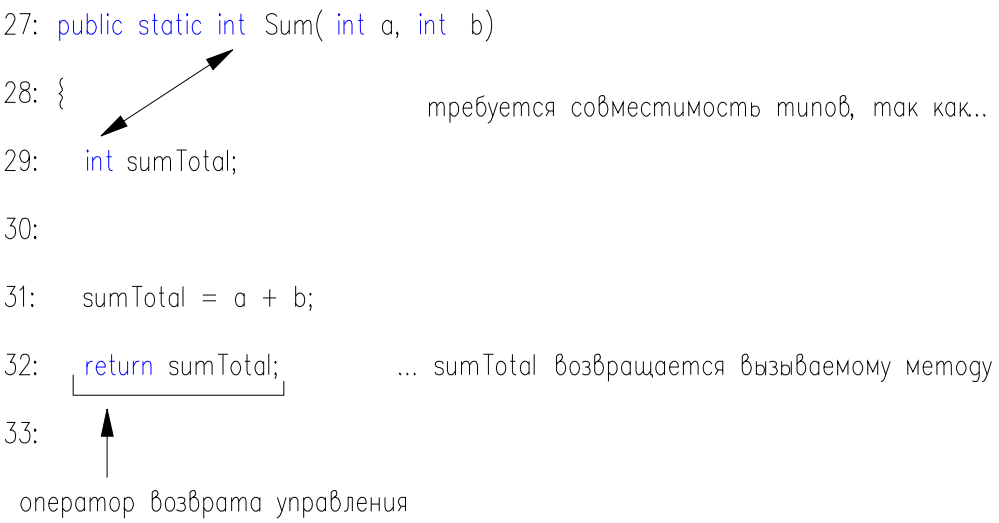


Рис2.1. Тип возвращаемого значения в заголовке метода должен совпадать с типом выражения, которое реально возвращается

// **МЕТОД, ОБЪЯВЛЕННЫЙ КАК VIOD, НЕ ВОЗВРАЩАЕТ ДАННЫХ**

Когда перед именем метода стоит слово void, это значит, что по его завершении в точку вызова не возвращается никаких данных. Таким образом, значение void в заголовке метода противоположно по смыслу int.

// **СОВЕТ**

При программировании на C# нужно мыслить в терминах строительных блоков. Именно этой цели и служат классы, объекты и методы: программы нужно разбивать на такие блоки. В программе C# обычно применяются классы и методы:

1. из библиотек классов (**.NET Framework** или других коммерческих пакетов). В Internet можно найти немало и бесплатных библиотек

2. написанные самим разработчиком

3. созданные другими программистами, например, коллегами по работе

Разумеется, не нужно пытаться изобрести колесо. Прежде чем начать тратить время и деньги на создание новых классов и методов, следует проверить, не была ли требуемая функциональность реализована и отлажена профессиональными программистами и представлена к использованию в виде эффективных и надежных программных компонентов.

6. Оператор присваивания

В строке 31 содержится оператор присваивания. a и b складываются посредством операции +. Результат сохраняется в переменной sumTotal с помощью операции присваивания =. Это легко проиллюстрировать выводом программы из листинга 3.1. Например, пусть введены числа 3 и 8. При вызове Sum 3 и 8 присваиваются a и b через переменные x и y в Main. После выполнения строки 31 sumTotal содержит число 11.

```
31:     sumTotal = a + b;
```

7. Объединение строк и вызовов методов

В строке 20 необходимо вначале рассмотреть блок Sum(x, y), который, как уже показано, вызывает метод Sum, определенный в строках 27-33, и пересылает ему два аргумента x и y.

```
20:     Console.WriteLine("Сумма: " + Sum(x, y));
```

После того как метод Sum возвращает управление, можно, фактически, заменить Sum(x, y) значением sumTotal из строки 32. В данном случае Sum(x, y) представляет значение 11 (см. рис. 4.2).

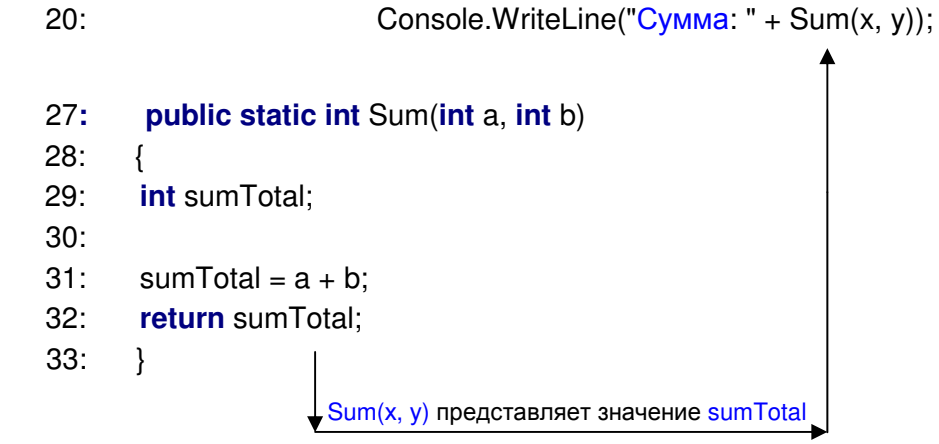


Рис. 1.2. При возврате из метода Sum(x, y) представляет значение sumTotal

Хотя конструкция Sum(x, y) находится внутри вызова другого метода — Console.WriteLine(), — C# с легкостью определяет, в какой последовательности должны происходить вызовы. Здесь вызов осуществляется тогда, когда требуется значение Sum(x, y). После того как выполняется код из строк 27-33 и поток выполнения возвращается в строку 20, ее можно представить в виде:

```
Console.WriteLine("Сумма: " + 11);
```

Поскольку 11 находится внутри вызова метода WriteLine, это значение автоматически преобразуется в строку. Таким образом, строка 20:

```
Console.WriteLine("Сумма: " + “11”);
```

Последнее, что осталось выяснить, — назначение символа + между " Сумма: " и "11". *Когда + расположен между двумя арифметическими операндами, он складывает их стандартным образом. Если же он окружен строками, компилятор C# изменяет его функциональность.* Он объединяет две строки вместе, давая в результате " Сумма: 11". Объединение двух строк вместе называется *конкатенацией*, а символ + в этом случае — *операцией конкатенации.*

В заключение, строка 20 принимает вид:

Console.WriteLine ("Сумма: + 11");

А это уже знакомо: такая конструкция имеет смысл "Вывести Сумма: + 11 на консоль". Именно такой вывод можно видеть при запуске программы.

// **СИМВОЛ +: СЛОЖЕНИЕ ИЛИ КОНКАТЕНАЦИЯ?** ,
Операция + применяется в различных цепях: для конкатенации строк и сложения чисел. В силу такой гибкости ее называют перегруженной. Перегруженные операции очень полезны, но имеют и недостатки: они затрудняют поиск ошибок и иногда приводят к странным результатам.

Строки 36-42 очень похожи на строки 27-33, единственное различие заключается в имени метода и переменных. Кроме того, метод Product вычисляет не сумму, а произведение двух чисел. Следует обратить внимание на то, что умножение обозначается символом звездочки (*).

```
36:     public static int Product(int a, int b)
37:     {
38:     int productTotal;
39:
40:     productTotal = a * b;
41:     return productTotal;
42:     }
```

Концептуально, строки 20 и 21 идентичны.

```
21:     Console.WriteLine("Произведение: " + Product(x, y));
```

// **СОВЕТ**
Порядок, в котором определены методы класса, является произвольным и не связан с тем, как они вызываются из исходного кода. Например, можно изменить порядок определений методов Sum и Product в классе SimpleCalculator.

8. Класс Math: полезный элемент .NET

Строка 22 похожа на 20 и 21, однако в ней для нахождения наибольшего из двух чисел применяется метод Max класса Math пространства имен System библиотеки классов **.NET Framework**.

```
22:     Console.WriteLine("Максимальное число: " + Math.Max(x, y));
```

Класс Math содержит множество полезных математических функций (например, тригонометрию и логарифмы).

Контрольные вопросы

1. Для чего используются пространства имен в C#?

2. В чем достоинство включения ключевого слова using с последующим именем пространства имен в начале программы?

3. Какое пространство имен содержит классы, связанные с математическими вычислениями и вводом-выводом на консоль?

5. Опишите переменную типа int.

9. Как указать, что метод не возвращает значения?

10. Как указать, что метод возвращает значение типа int?

12. Что такое формальные параметры?

14. Операция " + " производит только арифметическое сложение?

15. Поясните структуру пользовательского метода.

16. Каково назначение метода ToInt32 класса Convert?

Упражнения по программированию

Внесите следующие изменения в программу из листинга 3.1:

1. Замените многострочные комментарии в строках 3-6 двумя однострочными.

2. Позвольте пользователю кроме сложения и умножения производить также вычитание. Для этого, кроме других изменений, нужно добавить метод Subtract.

3. Измените программу так, чтобы она вычислила сумму и произведение не двух, а трех чисел. (Здесь можно опустить функции Max и Min.) Подсказка: в методе Main нужно объявить еще одну переменную типа int. Методы Sun, Product и Subtract (должны принимать три, а не два аргумента. Нужно позволить пользователю ввести третье число и добавить третий аргумент в вызовах этих методов.

4. Создайте два метода MyMax и MyMin, находящие, соответственно, наибольший и наименьший из трех аргументов. Подсказка: Math.Max(Math.Max(a, Ь),c) возвращает наибольшее из чисел a, b и c.

Список литературы

1. Микелсен Клаус. Язык программирования C#. Лекции и упражнения. Учебник: пер. с англ./ Клаус Микелсен –СПб.: ООО «ДиаСофтЮП», 2002. – 656 с.

2. Джо Майо. C#Builder. Быстрый старт. Пер. с англ. – М.: ООО «Бином-Пресс», 2005 г. – 384 с.

3. Основы Microsoft Visual Studio .NET 2003 / Пер. с англ. - М.: Издательско-торговый дом «Русская Редакция», 2003. – 464 с. Брайан Джонсон, Крэйт Скибо, Марк Янг.

4. Герберт Шилдт. Полный справочник по C# . / Пер. с англ./ Издательство: Вильямс, 2004 г. 752 с.

5. Чарльз Петцольд. Программирование в тональности C# / Пер. с англ. Издательство: Русская Редакция, 2004 г. - 512 стр.

C# & .NET по шагам (Web-ресурс)

1 | 2 | 3 | 4

Шаг 1 - Разработка приложений в .NET (основы). (24.09.2001 - 2.3 Kb)
Шаг 2 - Как будет распространяться приложение (основы). (24.09.2001 - 3.8 Kb)
Шаг 3 - Нам нужен .Net Framework SDK. (24.09.2001 - 3.8 Kb)
Шаг 4 - Hello Word C#. (25.09.2001 - 2.4 Kb)
Шаг 5 - Hello Word VB. (25.09.2001 - 1.7 Kb)
Шаг 6 - Hello Word VC++. (25.09.2001 - 1.6 Kb)
Шаг 7 - Пространство имен. (26.09.2001 - 2.7 Kb)
Шаг 8 - Net ассемблер и дизассемблер. (26.09.2001 - 3.5 Kb)
Шаг 9 - Просмотр класса в EXE проекте ILDasm.exe. (26.09.2001 - 1.6 Kb)
Шаг 10 - Две основы Net. (27.09.2001 - 2 Kb)
Шаг 11 - Отладка. (27.09.2001 - 33 Kb)
Шаг 12 - ADO.NET (27.09.2001 - 10 Kb)
Шаг 13 - Попробуем OLEDB. (27.09.2001 - 6 Kb)
Шаг 14 - Типы данных - системные и языка программирования. (28.09.2001 - 3 Kb)
Шаг 15 - Windows Form. (28.09.2001 - 7 Kb)
Шаг 16 - Где взять редактор C#. (28.09.2001 - 21 Kb)
Шаг 17 - Избавляемся от консольного окна. (28.09.2001 - 9 Kb)
Шаг 18 - Создаем окно. (28.09.2001 - 6 Kb)
Шаг 19 - Добавляем меню. (28.09.2001 - 6 Kb)
Шаг 20 - Свойства (properties). (28.09.2001 - 3 Kb)
Шаг 21 - Обработка событий на форме. (30.09.2001 - 5 Kb)
Шаг 22 - Изменение размера формы. (30.09.2001 - 2 Kb)
Шаг 23 - Изменение положения формы. (30.09.2001 - 2 Kb)
Шаг 24 - Override. (30.09.2001 - 2 Kb)
Шаг 25 - Встраиваем элемент управления в окно. (30.09.2001 - 5 Kb)
Шаг 26 - Обработка сообщений элемента классом элемента. (30.09.2001 - 6 Kb)
Шаг 27 - Еще один редактор C#. (30.09.2001 - 30 Kb)
Шаг 28 - Создание меню подробнее. (01.10.2001 - 6 Kb)
Шаг 29 - Одномерные Массивы. (01.10.2001 - 3 Kb)
Шаг 30 - foreach. (01.10.2001 - 2 Kb)
Шаг 31 - Интерфейсы. (01.10.2001 - 3 Kb)
Шаг 32 - Коллекции. (01.10.2001 - 6 Kb)
Шаг 33 - Создаем обработчик событий меню. (01.10.2001 - 6 Kb)
Шаг 34 - Сохраняем данные в файл. (01.10.2001 - 7 Kb)
Шаг 35 - Добавляем строку состояния. (02.10.2001 - 5 Kb)
Шаг 36 - Панели на строке состояния. (02.10.2001 - 6 Kb)
Шаг 37 - Икона формы. (02.10.2001 - 9 Kb)
Шаг 38 - Диалог открытия файлов. (02.10.2001 - 14 Kb)
Шаг 39 - Отображаем картинку. (02.10.2001 - 12 Kb)
Шаг 40 - Создаем панель инструментов. (02.10.2001 - 6 Kb)
Шаг 41 - Net Classes первые вывод. (02.10.2001 - 6 Kb)
Шаг 42 - XML документация кода. (02.10.2001 - 6 Kb)
Шаг 43 - XML notepad. (02.10.2001 - 16 Kb)
Шаг 44 - Заголовок формы и пункт меню выход. (03.10.2001 - 4 Kb)
Шаг 45 - Создаем файл с ресурсами строк. (03.10.2001 - 5 Kb)

……..

1 | 2 | 3 | 4

Приложение 2

Простые типы C#

В C# определено 13 простых типов, которые перечислены в таблице.
При написании программ придется не раз вернуться к этой таблице.

Хотя тип bool (последняя строка таблицы) рассматривается здесь как простой тип, он связан с управлением потоком выполнения программ.

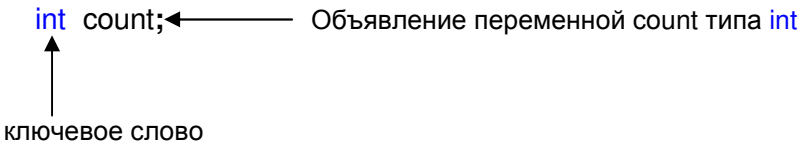
Таблица

Ключевое слово языка C#	Тип .NET CTS (Common Type System)	Вид значения	Используемая память	Диапазон и точность
sbyte	System.Sbyte	Целое число	8 битов	От -128 до127
byte	System.Byte	Целое число	8 битов	От 0 до *255*
short	System.Int16	Целое число	16 битов	От -32768 до 32767
ushort	System.Uint16	Целое число	16 битов	От 0 до 65535
int	System.Int32	Целое число	32 бита	От -2147483648 до 2147483647
uint	System.Uint32	Целое число	32 бита	От 0 до 4294967295
long	System.Int64	Целое число	64 бита	От -9223372036854775808 до 9223372036854775807
ulong	System.Uint64	Целое число	64 бита	От 0 до 18446744073709551615
char	System.Char	Целое число (один символ)	16 битов	Все символы Unicode
float	System.Single	Число с плавающей точкой	32 бита	От (+/-)1.5 x 10^{-45} до (+/-) 3.4 x 10^{38}. Примерно 7 значащих цифр
double	System.Double	Число с плавающей точкой	64 бита	От (+/-)5.0 x 10^{-324} до (+/-)3.4 x 10^{308}. 15 -16 значащих цифр
decimal	System.Decimal	Десятичное число (высокой точности)	128 битов	От (+/-)1.0 x 10^{-28} до (+/-)7.9 x 10^{28}. 28 -29 значащих цифр
bool	System.Boolean	true или false	1 бит	Нет

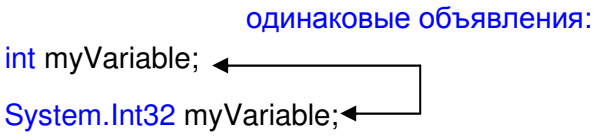
Рекомендуется просмотреть характеристики знакомого типа int в пятой строке таблицы. Это дает хорошее представление о содержании каждого столбца.

Ниже представлена краткая сводка, что означает каждый столбец в таблице:

Столбец "Ключевое слово". Ключевое слово относится к символу, используемому в исходном коде C# при объявлении переменной. В качестве примера рассмотрим хорошо известный оператор объявления с использованием ключевого слова int.



Столбец "Тип .NET CTS". Пространство имен **System** .NET-Framework содержит все простые типы. Каждое ключевое слово, показанное в первом столбце, — это псевдоним типа, определенного в CTS. Например, ключевое слово **int** обозначает **System.Int32** в CTS. Таким образом, в исходном коде можно использовать как короткий псевдоним типа, так и его длинное полное имя. Два следующих выражения идентичны:



Легко видеть, что последняя запись довольно громоздка, поэтому в программах рекомендуется использовать только псевдонимы.

Столбец "Вид значения". В этом столбце определены четыре различных группы содержащихся в C# простых типов: *1)* целое число, *2)* число с плавающей точкой, *3)* true / false и *4)* число высокой точности. Таким образом,

- Целые числа — числа без дробной части.
- Числа с плавающей точкой — числа с дробной частью.
- Числа высокой точности также представляют дробные величины, но, очевидно, с более высокой точностью.
- Величины типа bool могут содержать только два значения — **true** или **false**.

Столбец "Используемая память". Количество битов памяти, используемое величиной каждого типа.

Столбец "Диапазон и точность". Отображает диапазон и точность, обеспечиваемые величинами соответствующего типа. Следует отметить, что, хотя тип **char** разработан для отдельных символов, он рассматривается как целочисленный.

В таблице приведено **девять** целочисленных типов. Они отличаются друг от друга по трем параметрам: *1)* диапазону, *2)* объему занимаемой памяти и *3)* способности хранить отрицательные числа.

В таблице содержатся и **три** типа с плавающей точкой — **float, double и decimal**, используемые для хранения чисел, содержащих дробную часть (к примеру, 6.87, 9.0 и 100.01). Основные различия: *1)* диапазон, *2)* используемая память и *3)* точность.

Тип decimal

Тип **decimal** — это 128-битный тип с высокой точностью, предназначенный для финансовых и денежных вычислений. Он может представлять значения в диапазоне от

1.0×10^{-28} до 7.9×10^{28} с 28-29 значащими цифрами. Важно отметить, что точность задана в цифрах, а не в десятичных знаках. Все вычисления выполняются без округления до максимального значения в 28 десятичных знаков.

Как можно видеть, диапазон этих значений уже, чем в типе **double**, но они обладают намного большей точностью. Поэтому не существует неявного преобразования между типами **decimal** и **double**: в одном направлении такое преобразование может вызывать переполнение, в другом - привести к потере точности. Оно должно вызываться явно при помощи приведения типов.

Чтобы при определении переменной и присваивании ей значения обозначить тип значения как **decimal**, используется суффикс m:

```
decimal decMyValue =  1.0 m;
```

Если опустить суффикс m, тогда до присваивания переменной значения компилятор будет рассматривать ее как **double**.