Занятие 1. Платформа .Net Framework. Обзор C#

Тренер: Алексей Дышлевой

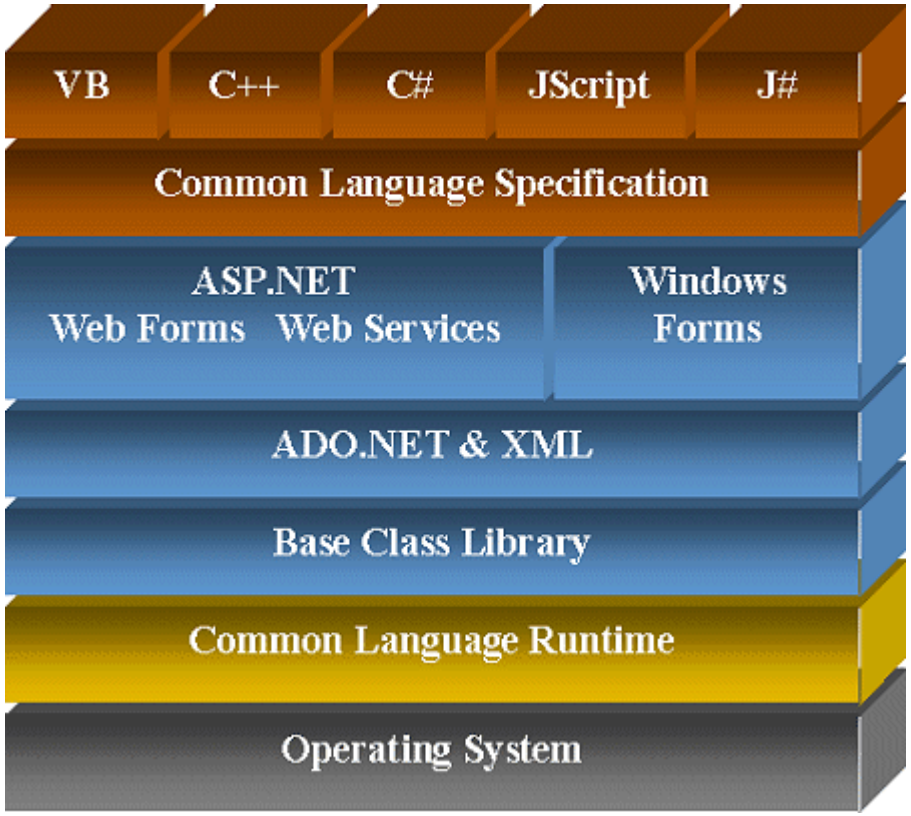
Литература

- Джеффри Рихтер / CLR via C#. Программирование на платформе Microsoft .NET Framework 4.0 на языке C#: Питер, 2011, 928 с.
- Троелсен Эндрю / Язык программирования C# 5.0 и платформа .NET 4.5, 6-е изд.: Пер. с англ.: Изд. дом «Вильямс», 2013. - 1312 с.
- Трей Неш / C# 2010: ускоренный курс для профессионалов: Пер. с англ.: Изд. дом «Вильямс», 2010. - 592 с.
- Кристиан Нейгел, Билл Ивьен, Джей Глинн, Карли Уотсон, Морган Скиннер / C# и платформа .Net для профессионалов.: Пер. с англ.: М.: ООО “И.Д. Вильямс”, 2011. – 1440 с.
- http://msdn.microsoft.com/library/vstudio/
- Руководство Microsoft по проектированию архитектуры приложений. 2-е издание.

Платформа .Net Framework 4.0 и 4.5

- .NET Framework 4 является комплексной платформой разработки, предлагающей быстрый и эффективный способ для создания приложений и услуг.
- .NET Framework 4 обеспечивает три основных элемента:
  - общеязыковая среда выполнения (Common Language Runtime, CLR),
  - библиотека классов .NET Framework (.NET Framework Class Library),
  - коллекция фреймворков для разработки приложений (Development Frameworks).

Архитектура платформы .Net Framework



Метаданные и MSIL-код

- CLR (Common Language Runtime) – важнейший компонент .Net Framework. Предоставляет среду выполнения программ.
- Возможности исполняющей среды CLR доступны разным языкам программирования. Фактически во время выполнения CLR не знает, на каком языке написан исходный код.
- Писать код можно на любом языке, компилятор которого предназначен для CLR (под компиляцией подразумевается контроль синтаксиса и анализ «корректного кода»). Компиляторы проверяют исходный код, убеждаются, что все написанное имеет какой-то смысл, и затем генерируют код, описывающий решение задачи.
- Все CLR-совместимые компиляторы генерируют IL-код (или MSIL-код, Microsoft Intermediate Language-код). IL-код иногда называют управляемым (managed code), потому что CLR управляет его жизненным циклом и выполнением.

Метаданные и MSIL-код

- Каждый компилятор, предназначенный для CLR, кроме генерации IL-кода, создает полные метаданные (metadata) для каждого управляемого модуля.
- Метаданные – это набор таблиц данных, описывающих все, что определено в модуле, например, типы и их члены. В метаданных также есть таблицы, указывающие, на что ссылается управляемый модуль.
- Метаданные расширяют возможности таких старых технологий, как библиотеки, при этом они гораздо полнее. В отличие от библиотек типов метаданные всегда связаны с файлом, содержащим IL-код. Фактически метаданные всегда встроены в тот же .exe/.dll файл, что и код, поэтому их нельзя разделить.
- Так как компилятор генерирует метаданные и код одновременно и привязывает их к конечному управляемому модулю, рассинхронизация метаданных и описываемого ими IL-кода исключена

C# и CLR

- C# компилируется в IL (Intermediate Language)
- Код на CIL выполняется виртуальной машиной CLR, а результат в виде шестнадцатиричных комманд отправляется на выполнение операционной системой
- CLR компилирует код IL в машинный код, прежде чем выполнить его. Как только код один раз скомпилирован, CLR сохраняет его и выполняет при необходимости

Сборщик мусора

- Сборщик мусора (garbage collector) – решение вопросов по управлению памятью. Предназначен для очищения памяти.
- Вся динамически распределяемая память распределяется в области кучи. Всякий раз, когда .Net обнаруживает, что куча данного процесса заполнилась и нуждается в приведении в порядок, он вызывает сборщик мусора.
- Сборщик мусора проходит по всем переменным, находящимся в контексте кода, и проверяет ссылки на объекты, расположенные в области кучи, чтобы идентифицировать, какие из них доступны из кода, т.е. узнать, на какие объекты существуют ссылки. Любой объект с отсутствием ссылок рассматривается как такой, к которому больше не будет осуществляться доступ из кода, потому объект может быть удален.
- Невозможно узнать, когда сборщик мусора будет вызван, но возможно переопределить этот процесс.

Сборщик мусора

- Сборщик мусора всегда запускается в отдельном потоке. В момент его активной работы все остальные потоки приложения приостанавливаются
- Сборщик мусора разделяет объекты на поколения:
  - **Поколение 0** Новый созданный объект, который еще ни разу не помечался, как кандидат на удаление
  - **Поколение 1** Объект, который уже пережил сборку мусора 1 раз. Обычно сюда попадают объекты, которые были помечены для удаления, но не были удалены из-за достаточного свободного места в куче
  - **Поколение 2** Объекты, которые пережили более одной очистки памяти сборщиком мусора
- Вероятность существования объекта в куче выше, чем выше его поколение

Сборки

- Сборка (assembly) - это абстрактное понятие.
- Во-первых, это логическая группировка одного или нескольких управляемых модулей и файлов ресурсов.
- Во-вторых, это самая маленькая единица с точки зрения повторного использования, безопасности и управления версиями.
- Сборка может состоять из одного (однофайловая сборка) или нескольких (многофайловая сборка) файлов – все зависит от выбранных средств и компиляторов.
- В мире .NET сборка представляет собой то, что в других условиях называют компонентом.
- Сборка может состоять из многих модулей, которые объединяет вместе манифест, описывающий содержимое сборки

Сборки

- CLR работает со сборками – сначала всегда загружает файл с манифестом, а затем получает из манифеста имена остальных файлов сборки. Некоторые характеристики особенно важны:
- в сборке определены повторно используемые типы;
- сборка помечена номером версии;
- со сборкой может быть связана информация безопасности.
- Платформа .NET разделяет сборки на **локальные** (или сборки со слабыми именами) и **глобальные** (или сборки с сильными именами).
- Локальные сборки размещаются в том же каталоге (или подкаталоге), что и клиентское приложение, в котором они используются. Имя локальной сборки – слабое имя – это имя файла сборки без расширения.
- Глобальные сборки помещаются в GAC (Global Assembly Cashe) – защищенное хранилище сборок

Назначение сборкам имен

- Имя сборки в GAC:
  NameAssembly,Version=1.1.0.0,Culture=en,PublicKeyToken=874e23ab874e23ab
- Номер версии сборки состоит из следующих компонент:
  - Главного номера версии (Major version number)
  - Второстепенного номера версии (Minor version number)
  - Номера сборки (Build number)
  - Номера версии (Revision number)
- Часть Major является обязательной. Любая другая часть может быть опущена

Visual Studio 2013

- Интегрированная среда разработки (Integrated development environment, IDE).
- Быстрая разработка приложений (Rapid application development).
- Сервер и доспуп к данным (Server and data access).
- Возможности отладки (Debugging features).
- Обработка ошибок (Error handling).
- Справка и документация (Help and documentation).

Шаблоны проектов Visual Studio 2013

- Console Application
- WPF Application
- Class Library
- Windows Forms Application
- ASP.NET Web Application
- ASP.NET MVC Application
- Silverlight Application
- WCF Service Application

Структура проектов Visual Studio. Решение (Solution)

Файл	Описание
.cs	Файлы кода, которые могут принадлежать к одному проектному решению. Этот тип файла может быть одним из следующих: • модуль; • файл Windows Forms; • файл классов.
.csproj	Файлы проекта, которые могут принадлежать к нескольким проектным решениям. Файл .csproj также хранит параметры проекта.
.aspx	Файлы, представляющие веб-страницы ASP.NET. Файл ASP.NET может содержать код Visual C# или использоваться сопровождающим .aspx.cs файлом для хранения кода в дополнение к разметке страницы.
.config	Файлы конфигурации это XML-файлы, использующиеся для хранения настроек на уровне приложения, например, таких как строки подключения к базе данных, которые затем можно изменять без повторной компиляции приложения.
.xaml	XAML файлы используются в WPF и Silverlight Microsoft приложениях для определения элементов пользовательского интерфейса.

Решение (Solution) представляет собой котейнер для одного или нескольких проектов.

Пространства имен

- Пространство имен (namespace) представляет собой логический набор классов. Классы хранятся в сборках, а пространство имен является средством для устранения неоднозначности классов, которые могут иметь одинаковые имена в различных сборках.
- Например, пространство имен **System.IO** включает в себя следующие классы, которые позволяют управлять файловой системой Windows. Однако, можно создать классы с таким же названием и в собственном пространстве имен:
  - File
  - FileInfo
  - Directory
- Добавление пространства имен: using System;
- Позволяют связывать классы и другие типы. Также можно создавать псевдонимы:
  - using alias = System.Collections;
  - alias.ArrayList AL = new alias.ArrayList();

Console Application

- При создании нового консольного приложения с помощью шаблона **Console Application**, Visual Studio 2013 выполняет следующие задачи:
  - Создает новый файл с расширением **.csproj** для представления консольного проекта и структуры всех компонентов по умолчанию в нем.
  - Добавляет ссылки на сборки библиотеки классов .NET Framework, которые обычно требуются консольным приложениям. Набор включает в себя сборку System.
  - Создает файл **Program.cs** с методом Main, предоставляющим точку входа в консольное приложение.

Windows Forms

- При создании нового приложения Windows Forms с помощью шаблона приложения WPF Visual Studio 2013 выполняет следующие задачи:
  - Создает новый файл с расширением .**csproj** для представления проекта Windows Forms и структурирует в проекте все компоненты по умолчанию.
  - Добавляет ссылки на необходимые сборки, включая сборки System, System.Collections.Generic, System.ComponentModel, System.Data, System.Drawing, System.Linq, System.Text, System.Threading.Tasks, System.Windows.Forms.
  - Создает пустую форму Form.cs[Design] и файл кода формы Form.cs, которые можно использовать в качестве отправной точки для создания первого окна

WPF (Windows Presentation Foundation)

- При создании нового приложения WPF с помощью шаблона приложения WPF Visual Studio 2013 выполняет следующие задачи:
  - Создает новый файл с расширением **.csproj** для представления проекта WPF и структурирует в проекте WPF все компоненты по умолчанию.
  - Добавляет ссылки на необходимые сборки, включая сборки PresentationCore, PresentationFramework, System, System.Core и System.Xaml.
  - Создает файл разметки App.xaml и файл кода (code-behind) App.xaml.cs, которые можно использовать для определения ресурсов и функциональности уровня приложения.
  - Создает файл разметки MainWindow.xaml и файл кода (code-behind) MainWindow.xaml.cs, которые можно использовать в качестве отправной точки для создания первого окна WPF.

Debug, breakpoints, through, code over

Опция меню	Кнопка панели инструментов	Клавиши быстрого доступа	Описание
Start Debugging	Start/continue	F5	Эта кнопка доступна, когда приложение не работает или находится в режиме приостановки. Кнопка запускает приложение в режиме отладки или возобновляет в случае режима приостановки.
Break All	Break all	CTRL+ALT+BREAK	Эта кнопка вызывает обработку приложения в режим pause и break. Кнопка доступна, когда приложение выполняется.
Stop Debugging	Stop	SHIFT+F5	Эта кнопка останавливает отладку. Она доступна, когда приложение работает или в режиме приостановки.
Restart	Restart	CTRL+SHIFT+F5	Эта кнопка равносильна остановке, следующей за стартом, что приводит к перезапуску приложения с самого начала. Она доступна, когда приложение работает или в режиме приостановки.
Step Into	Step into	F11	Эта кнопка используется для пошагового выполнения кода.
Step Over	Step over	F10	Эта кнопка используется для пошагового выполнения кода.
Step Out	Step out	SHIFT+F11	Эта кнопка используется для пошагового выполнения кода.
Windows	Windows	Various	Эта кнопка обеспечивает доступ к различным окнам отладки, каждое из которых имеет свое собственное сочетание клавиш.

Объектно-ориентированное программирование

- ООП – это парадигма программирования, основанная на представлении предметной области в виде взаимосвязанных абстрактных объектов и их реализаций
- Класс – пользовательский тип данных, объединяющий данные и методы их обработки
- Объект – экземпляр класса
- Пример:
  - Ноутбук – класс
  - Ноутбук Lenovo Ideapad Yoga 2 – объект

Объекты

- Объекты характеризуются понятиями состояния, поведения и идентичности
- Состояние – перечень всех свойств объекта и текущие значения каждого из этих свойств
- Поведение – то, как объект действует и реагирует; выражается в терминах состояния объекта и передачи сообщений. Поведение – это наблюдаемая и проверяемая извне деятельность объекта
- Поведение объекта состоит из его операций. Виды операций:
  - Модификатор – изменение состояния объекта
  - Селектор – считывание состояния объекта (без его изменения)
  - Итератор – организация доступа ко всем частям объекта в строгой последовательности
- Идентичность – свойство объекта (или набор свойств), который отличает его от других объектов

Язык C#

- Разработан в 1998-2001 группой инженеров под руководством Андерса Хейлсберга (Microsoft)
- Основной язык программирования для платформы Microsoft .Net Framework
- Синтаксис базируется на C++ и Java
- C# - строго типизированный язык

Базовые типы и переменные

Тип данных	FCL-типы	Диапазон
byte	System.Byte	0 .. 255
sbyte		-128 .. 127
short	System.Int16	-32,768 .. 32,767
int	System.Int32	-2,147,483,648 .. 2,147,483,647
long	System.Int64	-9,223,372,036,854,775,808 .. 9,223,372,036,854,775,807
float	System.Single	-3,402823e38 .. -3,402823e38 ..
double	System.Double	-1,79769313486232e308 .. 1,79769313486232e308
decimal	System.Desimal	-79228162514264337593543950335 .. 79228162514264337593543950335
char	System.Char	Символ Юникода.
string	System.Bool	Строка символов Юникода.
bool	System.String	true или false
object	System.Object	Объект.

Переменные

- Переменная характеризуется:
- *Имя (Name)*. Уникальный идентификатор, который ссылается на переменную в коде.
- *Адрес (Address)*. Ячейка памяти переменной.
- *Тип данных (Data type)*. Тип данных, которые может хранить переменная (определяет размер и представление данных в памяти).
- *Значение (Value)*. Значение по адресу переменной.
- *Область видимости (Scope)*. Определенные участки кода, которые могут получать доступ и использовать переменную.
- *Время жизни (Lifetime)*. Период времени, который переменная является действительной и доступной для использования.

Объявление и инициализация переменных

- Имя переменной называется идентификатором. Существуют правила, касающиеся идентификаторов, которые можно использовать:
  - Идентификатор может содержать только буквы, цифры и символы подчеркивания.
  - Идентификатор должен начинаться с буквы или символа подчеркивания.
  - Идентификатор не должен быть одним из ключевых слов, которые C# резервирует для собственного использования.

Типы значений

- Значимые типы (value types). Экземпляры значимых типов располагаются в стеке приложения. В управляемой куче могут располагаться, если они являются членами ссылочных типов.

- Ссылочные типы (reference types). В стеке приложения находится ссылка на экземпляр. Сами экземпляры располагаются в управляемой куче.

Инициализация переменных по умолчанию

- Значимые типы инициализуются установкой всех битов в 0
- Ссылки на объекты – null
- Поля логического типа - false

Значимые типы

- Существует 2 категории значимых типов: структура и перечисление.
- Переменные значимых типов напрямую содержат их значения, т.е. память встроена в контекст, в котором объявлена переменная.
- Структура может наследовать только от System.ValueType. При этом наследование от System.Int32 невозможно.
- Однако структура может реализовать один или несколько интерфейсов, и можно приводить тип структуры в тип интерфейса.

Приведение (преобразование) типов

- Присвоение «большему типу» значение «меньшего типа». Безопасное сохранение, гарантирующее сохранение значения
- Присвоение «меньшему типу» значения «большего типа» должно быть явным. Грозит потерей информации
- Неявное ссылочное преобразование
  - BaseType DTobj = new DerivedType();
  - При этом DerivedType должен быть унаследован от BaseType
- Явное ссылочное преобразование
  - BaseType BTobj = new BaseType();
  - DerivedType DTobj = new DerivedType();
  - DTobj = (DerivedType) Btobj;
  - Может быть сгенерировано InvalidCastException, если CLR не сможет выполнить такое преобразование

Неявно типизированные переменные (var)

- Использование ключевого слова **var** приводит к зарезервированию компилятором некоторого участка памяти для хранения переменной
- При этом обязательна инициализация такой переменной при ее создании
- var CompErrValue; // выдаст ошибку компиляции

Выражения и операции

- *Операнды* (Operands). Операнды это значения, например, числа и строки, над которыми выполняются операции. Операндами могут быть постоянные значения (литералы), переменные, свойства или результаты вызова метода.
- *Операции* (Operators). Операции определяют действия, выполняемые над операндами. Операции существуют для всех основных математических операций в дополнение к более сложным операциям, таким как логическое сравнение или манипуляции битами данных.

Выражения и операции

Operator type	Operators
Арифметические	+, -, *, /,%
Инкримент, декримент	++, --
Сравнение	==,! =, <,>, <=, <=, is
Объединение (конкатенация) строк	+
Логические/битовые	&, |, ^, &&, !, | |
Индексация (отсчет начинается с элемента 0)	[]
Приведение (casting)	(), as
Присваивание	=, + =, -=, *=, =,% =, & =, | =, =, <=,>> =,?
Битовый сдвиг	<<, >>
Информация о типе	SizeOf, TypeOf
Конкатенация и удаление делегатов	+, -
Контроль за переполнением	checked, unchecked
Разименования и получения адреса	*, ->, [ ], &
Условная	?:

Приоритет операций

Приоритет	Операция
Наивысший	++, -- (префиксные), +, - (унарные), !, ~
	*, /, %
	+, -
	<<, >>
	<, >, <=, >=
	==, !=
	&
	^
	|
	&&
	||
	операции присваивания
Самый низкий	++, -- (постфиксные)

Операции type of, is, as

- typeof – получение объекта System.Type для типа
  - System.Type type = typeof (int);
  - Для получения типа выражения во время выполнения можно использовать:
  - int i = 0;
  - System.Type type = i.GetType();
- is – возвращает булевое значение, говорящее о том, можно ли преобразовать данное выражение в указанный тип
  - long Lvalue = 111;
  - bool IsInt = LValue is int;
- as подобна is, но она возвращает ссылку на уже преобразованный объект указанного типа, либо null, если такое преобразование невозможно
  - DerivedClass DC = new DerivedClass();
  - BaseClass BC = DC as BaseClass;

Операторы выбора

- if ([condition])
- {
- [code to execute if condition is true]
- }
- else
- {
- [code to execute if condition is false]
- }

- switch ([expression to check])
- {
- case [test1]:
- ...
- [exit case statement]
- case [test2]:
- ...
- [exit case statement]
- default:
- ...
- [exit case statement]
- }

Операторы цикла

- while ([condition])
- {
- [Code to loop]
- }

- do
- {
- [Code to loop]
- } while ([condition]);

- for ([counter variable] = [starting value]; [limit]; [counter modification])
- {
- [Code to loop]
- }

Перечисления

- Создание типа с использованием ключевого слова enum
- По умолчанию базовым типом каждого элемента перечисления является int, но можно задать другой целый, используя двоеточие
  - enum Colors: byte
  - {
  - Red,
  - Blue
  - }

Массивы

- Массив представляет собой набор объектов, которые сгруппированы вместе и управляются как единое целое. О массиве можно думать как о последовательности элементов. Все элементы массива имеют одинаковый тип. Можно создавать простые массивы, которые имеют одно измерение (список), два измерения (таблицу), три измерения (куб) и так далее. Массивы имеют следующие характеристики:
- Каждый элемент в массиве содержит значение.
- Массивы индексируются с нуля. Первым индексом в массиве является 0.
- Длина массива это общее число элементов, которое он может содержать.
- Нижняя граница массива индекс его первого элемента.
- Массивы могут быть одномерными, многомерными или непрямоугольные (массивы массивов).
- Ранг массива это число измерений в массиве

- int[] list = new int[20];

Многомерные массивы

- int[,] table;
- table = new int[10, 2];
-
- int[,,] cube = new int[3, 2, 5];

Массивы массивов (jagged array)

- Type [][] jaggedArray = new Type[10][];
- jaggedArray[0] = new Type[5]; // разные размеры
- jaggedArray[1] = new Type[7];
- ...
- JaggedArray[9] = new Type[21];

Неявно типизированные массивы

- var mixed = new[] { 1, DateTime.Now, true, false, 1.2 };

- var d = new[]
- {
- new[] {"Luca", "Mads", "Luke", "Dinesh"},
- new[] {"Karen", "Suma", "Frances"}
- };

Символы

Метод	Описание
ConvertFromUtf32	Преобразует целочисленный суррогатный UTF-код в строку
ConvertToUtf32	Преобразует пару суррогатных символов в UTF-код
GetNumericValue	Преобразовывает указанный числовой символ Юникод в число двойной точности с плавающей запятой
GetUnicodeCategory	Метод возвращает элементы перечисления UnicodeCategory, описывающего категорию символа
IsControl	Возвращает true, если символ является управляющим
IsDigit	Возвращает true, если символ является десятичной цифрой
IsLetter	Возвращает true, если символ является буквой
IsLetterOrDigit	Возвращает true, если символ является буквой или цифрой
IsLower	Возвращает true, если символ – это буква в нижнем регистре
IsNumber	Возвращает true, если символ является десятичной или шестнадцатеричной цифрой
IsPunctuation	Возвращает true, если символ является знаком препинания
IsSeparator	Возвращает true, если символ является разделителем
IsSurrogate	Возвращает true, если символ является суррогатным
IsUpper	Возвращает true, если символ – это буква в верхнем регистре
IsWhiteSpace	Возвращает true, если символ является пробельным. К пробельным символам относятся, помимо пробела, и другие символы, например, символ конца строки и символ перевода каретки
Parse	Преобразует строку в символ. Строка должна состоять из одного символа, иначе генерируется исключение
ToLower	Приводит символ к нижнему регистру
ToUpper	Приводит символ к верхнему регистру
TryParse	Пытается преобразовать строку в символ

Символы

- char c;
- int n;
- // Convert number <-> character using C# casting
- c = (char)65;
- Console.WriteLine(c);// Displays "A"
- n = (int)c;
- Console.WriteLine(n);// Displays "65"
- c = unchecked((char)(65536 + 65));
- Console.WriteLine(c); // Displays "A"
- // Convert number <-> character using Convert
- c = Convert.ToChar(65);
- Console.WriteLine(c);// Displays "A"
- n = Convert.ToInt32(c);
- Console.WriteLine(n); // Displays "65"

Строки

- string s = new string ("Hi there.");
- string s = "Hi there.";
- string s = new string(' ', 20);
-
- char[] a = { 'a', 'b', 'c', 'd', 'e' };
- string s = new string(a);
-
- char[] a1 = { 'a', 'b', 'c', 'd', 'e' };
- string v1 = new string(a, 0, 2);

Строки

Член	Описание
Compare (статический метод)	Сравнение двух строк в лексикографическом (алфавитном) порядке. Разные реализации метода позволяют сравнивать строки с учетом, или без учета регистра.
CompareTo (экземплярный метод)	Сравнение текущего экземпляра строки с другой строкой.
Concat (статический метод)	Слияние произвольного числа строк.
Copy (статический метод)	Создание копии строки.
Empty (статическое поле)	Открытое статическое поле, представляющее пустую строку.
Format (статический метод)	Форматирование строки в соответствии с заданным форматом.
IndexOf, LastIndexOf (экземплярные методы)	Определение индекса первого или, соответственно, последнего вхождения подстроки в данной строке.
IndexOfAny, LastIndexOfAny (экземплярные метод)	Определение индекса первого или, соответственно, последнего вхождения любого символа из подстроки в данной строке.
Insert (экземплярный методы)	Вставка подстроки в заданную позицию.
Join (статический метод)	Слияние массива строк в единую строку. Между элементами массива вставляются разделители.

Строки

Член	Описание
Length (свойство)	Возвращает длину строки.
PadLeft, PadRigth (экземплярные методы)	Выравнивают строки по левому или, соответственно, правому краю путем вставки нужного числа пробелов в начале, или в конце строки.
Remove (экземплярный метод)	Удаление подстроки из заданной позиции.
Replace (экземплярный метод)	Замена всех вхождений заданной подстроки, или символа новыми подстрокой, или символом.
Split (экземплярный метод)	Разделяет строку на элементы, используя разные разделители. Результаты помещаются в массив строк.
EndWith EndWith (экземплярные методы)	Возвращают true или false в зависимости от того, начинается, или заканчивается строка заданной подстрокой.
Substring (экземплярный метод)	Выделение подстроки, начиная с заданной позиции.
ToCharArray (экземплярный метод)	Преобразует строку в массив символов.
ToLower, ToUpper (экземплярные методы)	Преобразование строки к нижнему или,соответственно, к верхнему регистру.
Trim, TrimStart, TrimEnd (экземплярные методы)	Удаление пробелов в начале и конце строки, или только с начала, или только с конца соответственно.

Задание 1

- Вычислить a! + a^n с использование только арифметических операций

- В числе x, содержащем более 2-х знаков зачеркнули его вторую цифру. Когда к образованному при этом числу справа приписали вторую цифру числа x, то получилось число n. По заданному n найти число x (значение n вводится с клавиатуры, n >= 100).