Введение

Книга составлена на основе курса лекций по языку программирования C# и предназначена для студентов технического университета, обучающихся по направлениям «Информационные системы и технологии», «Информатика и вычислительная техника», «Информационные системы и технологии». Учебное пособие содержит основные сведения, позволяющие создавать достаточно сложные программы. Построение курса предполагает, что читатель владеет языком программирования C++, знаком с концепцией объектно-ориентированного программирования (ООП), оконным интерфейсом и стандартом XML.

C# является C-подобным языком, поэтому везде, где это возможно, проводится аналогия с языком C++. Особое внимание уделяется тем моментам, которые различают те или иные конструкции языков. Следует, однако, иметь в виду, что C# — это другой язык программирования. Он включает в себя совершенно новые концепции, обладает новыми возможностями и все еще находится в процессе развития.

Изложение материала иллюстрируется многочисленными примерами. В качестве инструментария используется Visual Studio 2019, современная версия фреймворка .NET Core 3.1, поддерживающая стандарт C# 8. Мы учитываем стратегию фирмы Microsoft, которая заключается в том, что .NET Core станет в будущем .NET 5 и, хотя поддержка .NET Framework будет продолжаться, новые функции будут добавляться только в .NET Core.

Книга состоит из 20 лекций:

1. **Краткий обзор синтаксиса C#** — рассматриваются типы данных, синтаксис основных конструкций языка.

2. **Массивы** — одно- и многомерные массивы, диапазоны, фрагменты массива. Обработка исключений.

3. **Классы** — синтаксис описания класса. Методы и свойства, перегрузка операторов. Анонимные типы.

4. **Наследование** — модели наследования, виртуальные и абстрактные методы. Абстрактные классы и интерфейсы.

5. **Дополнительные возможности ООП** — структуры, перечисления, кортежи. Коллекции. Обобщенные классы и коллекции. Методы расширения.

6. **Делегаты, события и лямбда-выражения** — делегаты и анонимные методы, события, как контейнеры делегатов. Описание и использование лямбда-выражений. Небезопасный код.

7. **Работа с текстовыми строками** — классы String и StringBuilder. Региональные стандарты, класс CultureInfo. Регулярные выражения.

8. **Работа с файлами** — классы работы с дисками, каталогами и файлами. Потоковые классы. Потоки в памяти, классы StringReader и StringWriter.

9. **LINQ** — язык интегрированных запросов. Синтаксис LINQ to Objects. Фильтрация данных, упорядочение, выборка и преобразование. Методы расширений в запросах.

10. **LINQ to XML** — создание xml-документа методами класса XDocument и с помощью запроса. Обработка xml-документа, класс XPath.

11. **LINQ to Entities** — запрос к базе данных в технологии Entity Framework Core. Диспетчер пакетов и подключаемые модули.

12. **Хранение данных** — сериализация: двоичная, стандарты JSON и XML. Архивация файлов и каталогов. Архивация «на лету».

13. **Windows Forms** — для .NET Framework 4.7.2. Стандартная заготовка формы. Обработка событий.

14. **Графические построения** — графические примитивы: перья, кисти. Ресурсы. Меню, панель инструментов, строка состояния.

15. **Элементы управления** — TextBox, диалог установки шрифта, вывод документа на печать. DataGridView, DateTimePicker. Система координат, построение графика.

16. **Диалоговые формы** — модальный и немодальный диалог. Динамическое размещение элементов управления, стыковка и привязка. Связывание свойств элементов.

17. **Сборки, процессы, потоки** — частные сборки и сборки общего пользования. Атрибуты. Домены приложений. Процессы и потоки. Пулы потоков.

18. **Управление потоками** — периодический запуск потока, разделение памяти между потоками, приоритеты потока. Синхронизация потоков оператором lock. Дескриптор ожидания, барьер, мьютекс, семафор.

19. **Задачи, параллельные и асинхронные вычисления** — класс Task. Передача параметров в задачу, завершение задачи. Параллельные вычисления, методы класса Parallel. Асинхронные методы, ключевые слова async/await.

20. **Рефлексия, работа с WinAPI-функциями** — загрузка внешних сборок. Доступ к методам и свойствам внешней сборки. Использование WinAPI-функций. Маршалинг параметров, их атрибуты.

Автор ни в коей мере не претендует на полноту изложения материала и не собирается конкурировать со справочной системой, а ставит своей целью в компактной форме заложить основу, опираясь на которую, можно двигаться дальше и создавать все более сложные программные комплексы.

Рекомендуется для более детального изучения материала иметь под рукой книги Г. Шилда [1], Э. Троелсена [2] и Ч. Петцольда [3]. В качестве справочника можно использовать интересную книгу П.В. Агурова [4]. По технологии LINQ можно посмотреть «Карманный справочник» Дж. Албахари [5] или А. Фримен [6].

Особо хотелось бы обратить внимание на сайт https://metanit.com/, где размещено очень много актуальной информации.

1 Краткий обзор синтаксиса C#

Язык C# был специально разработан фирмой Microsoft под новую платформу программирования .NET Framework, которая состоит из двух частей: общеязыковой исполняющей системы (*Common Language Runtime, CLR*) и библиотеки классов (*Framework Class Library, FCL*). Концепция платформы .NET Framework заключается в том, что компиляторы разных языков программирования строят программу на промежуточном платформенно-независимом языке (*Intermediate Language, IL*). Полученный «полуфабрикат», или в терминах .NET — сборка (*assembly*), транслируется «на лету» компилятором времени выполнения (*just-in-time compiler, JIT*) и запускается под управлением исполняющей системы Framework. Кроме того, при компиляции в тот же файл добавляется еще один компонент — метаданные, в которых хранится вся информация о типах данных, используемых в сборке.

Все это похоже на работу виртуальной машины Java, но в Framework достигается более глубокая степень интеграции с операционной системой, что позволяет строить эффективный программный код. Методы классов исполняемой сборки компилируются по мере необходимости и хранятся в кэш-памяти для повторного использования (код строится под текущую платформу).

Программа на C# представляет собой управляемый код, выполняемый в среде выполнения CLR. Причем нет необходимости заботиться об освобождении памяти. Память для объектов ссылочного типа (экземпляров классов) выделяется в *управляемой куче* и освобождается автоматически специальным модулем CLR, называемым *сборщиком мусора*.

При разработке языка C# был максимально использован синтаксис языка C++, но появились и существенные отличия. Одной из причин создания языка C# явилось желание освободить язык программирования от тех наслоений, которые образовались в нем по мере развития. Так, в CLR-расширении языка C++ пришлось вводить два типа классов: классы значений (*value class*) и классы ссылок (*ref class*). Так что в некотором смысле оправданы утверждения, что язык J++ является очищенным C++, а C# — очищенным J++.

Подробнее со спецификацией языка C# можно ознакомиться на сайте фирмы Microsoft: https://docs.microsoft.com/ru-ru/dotnet/csharp/.

Последняя версия C# 8.0 вышла в сентябре 2019 года вместе с релизом .NET Core 3, который полностью с ней совместим. Однако нужно иметь в виду, что пока не весь функционал Framework реализован в Core. Эти случаи мы будем отмечать.

В качестве первого примера приведем классический листинг «Hello».

```
using System;
class Hello
{
   static void Main()
   {
      Console.WriteLine("Hello, World");
   }
}
```

В первой строке сразу видно отличие от языка C++. Здесь директива using имеет несколько иной синтаксис. Если в языке C++ мы были вынуждены подключать заголовочные h-файлы с описанием классов и функций, а затем директивой using namespace ... подключать соответствующую область видимости, то здесь нет файлов включений, достаточно просто подключить область видимости, что мы и сделали директивой using System. Теперь вся область System доступна для использования. В ней, в частности, имеются классы с набором статических методов, обеспечивающих пользователя набором стандартных математических функций, а также методы консольного ввода/вывода, являющиеся членами класса System.Console.

Примечание

Можно было бы и не использовать директиву using System, но тогда для оператора вывода мы должны указать полное имя: System.Console.WriteLine().

Следующее отличие заключается в том, что в C# все объекты размещаются либо в классах, либо в структурах. Точка входа в программу должна быть в статическом методе с фиксированным именем Main().

Типы данных

Язык программирования C# с полным основанием можно назвать объектно-ориентированным. В нем даже простые типы являются структурами, косвенно производными от универсального типа

Таблица 1.1

Встроенные типы значений C#

Бит	Тип C#	Системный тип	Диапазон
8	bool	Boolean	true или false
8	sbyte	SByte	–128...127
8	byte	Byte	0...255
16	short	Int16	–32 768...32 767
16	ushort	UInt16	0...65 535
32	int	Int32	–2 147 483 648...2 147 483 647
32	uint	UInt32	0...4 294 967 295
64	long	Int64	–9223372036854775808...9223372036854775807
64	ulong	UInt64	0...18 446 744 073 709 551 615
16	char	Char	0...65 535
32	float	Single	$1.5 \cdot 10^{-45}$ до $3.4 \cdot 10^{38}$; 7–8 значащих цифр
64	double	Double	$5.0 \cdot 10^{-324}$ до $1.7 \cdot 10^{308}$; 15–16 значащих цифр
128	decimal	Decimal	$1.0 \cdot 10^{-28}$ до $7.9 \cdot 10^{28}$; 28 значащих цифр

ValueType. Всего имеется 13 простых типов. Идентификаторы char, int и т. д. означают псевдонимы системных типов, определенных в пространстве имен System. В табл. 1.1 показано их соответствие системным типам C# и диапазон допустимых значений.

Примечание

Тип decimal реализуется при помощи макроса, работает на порядок медленнее double и представлен формулой $\pm v \cdot 10^x$, где v — 96-битное целое, x — целое от 0 до 28. В первом 32-битном значении младшие 16 битов не используются и всегда установлены в 0, биты с 16 по 23 хранят значение x, биты с 24 по 30 не используются и установлены в 0. Старший бит хранит знак.

Все простые типы наследуют методы класса Object:

- Equals() — сравнивает объекты;
- Finalize() — выполняет операции очистки;
- GetHashCode() — создает хэш-код;
- ToString() — создает строковое представление объекта.

Помимо этого, все простые типы имеют набор свойств и статических методов. Вот пример таких методов для типа int:

- CompareTo() — сравнивает текущий экземпляр с заданным объектом;
- GetType() — возвращает тип текущего экземпляра;
- Parse() — преобразует строку в целое число со знаком;
- TryParse() — делает то же, что и Parse(), но не порождает исключений.

Константные поля:

- MaxValue — наибольшее возможное значение;
- MinValue — наименьшее допустимое значение.

C# является строго типизированным языком. Все переменные должны быть описаны перед использованием. Их областью видимости является текущий блок со всеми вложенными в него блоками. В отличие от языка C++, запрещено объявлять переменные с тем же именем во вложенных блоках.

Кроме типов значений нам понадобятся ссылочные типы string для работы со строками, а также тип object, являющийся базовым для всех классов C#, эти типы также считается простыми. Строка текста представляется набором символов Unicode. Класс String (синоним типа string) является коллекцией объектов Char.

Принципиальное отличие структур от классов в C# заключается в том, что *структуры являются типами значений, а классы — ссылочными типами.* Переменные встроенных типов, кроме string и object, являются структурами, и память для них выделяется на стеке, в то время как ссылочные типы описываются классами и память для них выделяется в управляемой куче.

Рассмотрим в листинге 1.1 пример простейших манипуляций с данными. Создадим консольное приложение (*.NET Core*).

Листинг 1.1. Форматирование вывода для простых типов

```
using System;
using System.Globalization;
class Program
{
   static void Main()
  {
    short s = 1;
    double x = 12.521;
    Console.WriteLine("shortMin var = {0}, shortMax var = {1}",
       Int16.MinValue, Int16.MaxValue);
    Console.WriteLine("short var = {0}, double var = {1}", s, x);
    string str = x.ToString("F2",
              CultureInfo.CreateSpecificCulture("en-US"));
    Console.WriteLine("double var.ToString() = {0}", str);
    string strValue = "-123,765";
    double y = Double.Parse(strValue);
    Console.WriteLine("double x + y = {0}", x+y);
    strValue = "3.1415926535897931";
    y = Double.Parse(strValue, CultureInfo.InvariantCulture);
    Console.WriteLine("PI = {0}", y.ToString( "r",
                    CultureInfo.InvariantCulture));
```

```
    Console.ReadKey();    //Задержим консольное окно
  }                       //до нажатия на любую клавишу
}
```

Вывод:

```
shortMin var = -32768, shortMax var = 32767
short var = 1, double var = 12,521
double var.ToString() = 12.52
double x + y = -111,244
PI = 3.1415926535897931
```

Примечание

Так же как в C++, комментарии могут вводиться в одну строку после «//» либо в несколько строк, ограниченных символами «/*» и «*/».

Рассмотрим подробнее приведенный выше листинг. У нас имеется один единственный статический метод Main(). Вообще-то возвращаемым значением метода является тип int, но допускается писать void, тогда метод по умолчанию возвращает 0.

Опишем две простые переменные и присвоим им начальные значения. При помощи статического метода WriteLine() класса Console выведем минимальное и максимальное значение для типа short: Int16.MinValue, Int16.MaxValue. Здесь мы воспользовались возможностью форматирования вывода:

```
WriteLine("Текст {индекс[,ширина[:формат]]} . . .", var1[,. . .]);
```

Число переменных должно быть не больше, чем фигурных скобок { } в строке формата. В квадратные скобки заключена необязательная часть конструкции [,ширина[:формат]].

Примечание

Если ширина поля вывода задана положительным числом, данные выравниваются по правому краю, отрицательное значение приводит к выравниванию по левому краю.

По умолчанию компилятор вставляет выражение var.ToString() вместо переменной var. Однако иногда бывает необходимо самостоятельно решить вопрос о представлении выводимого значения, тогда формат задается явно:

- C или c — денежный формат;
- D или d — целое десятичное число;
- E или e — научный (экспоненциальный) формат "-d.ddd... E+ddd";
- F или f — естественный формат;
- G или g — естественный или научный формат в зависимости от значения;

- N или n — с фиксированной точкой и разделителями разрядов;
- P или p — процентный формат;
- R или r — максимальная точность;
- X или x — шестнадцатеричный вывод (для целых значений).

Формат для типов с плавающей точкой может быть дополнен спецификатором точности (число знаков после десятичной точки), например F2.

Мы убедились в широких возможностях форматирования вывода в методе WriteLine(), причем вывод осуществляется в соответствии с текущим региональным стандартом — так, в качестве разделителя целой и дробной части числа в Европе и России используется символ «запятая». Если же нам нужно вывести число с «точкой» в американском формате, можно воспользоваться перегруженным методом ToString():

```
public string ToString(string format, IFormatProvider provider)
```

Первый параметр здесь — это такая же строка формата, что мы рассмотрели выше, а второй параметр представляет спецификацию регионального стандарта. Мы воспользовались методом CreateSpecificCulture("en-US") класса CultureInfo для американского стандарта. Именно из-за необходимости использования класса CultureInfo мы подключили область видимости System.Globalization.

Теперь посмотрим, как можно преобразовать число из текстового формата в double. Для этого воспользуемся статическим методом Parse() структуры Double. Необходимо отметить, что по умолчанию здесь также используется текущий региональный стандарт. К счастью, имеется перегруженный метод Parse():

```
public static double Parse(string s, IFormatProvider provider)
```

Параметры имеют тот же смысл, что и в методе ToString(), единственное отличие заключается в том, что мы воспользовались свойством InvariantCulture класса CultureInfo, что обеспечивает идентичный результат.

Теперь мы можем получить число в американском формате (т. е. в качестве десятичного разделителя используется точка) и вывести его на консоль. Последняя строка программы: Console.ReadKey(); служит для приостановки консольного окна до нажатия произвольной клавиши.

Примечание

В среде Visual Studio можно не использовать в конце консольной программы метод ReadKey(). Если запуск осуществить клавишами Ctrl-F5, консольное окно не закроется при завершении программы и будет ждать нажатия любой клавиши.

Литералы

Под литералами или константами понимают фиксированное значение, представленное в текстовой форме. В основном запись литералов соответствует синтаксису языка C++.

- Символьные константы записываются в одинарных кавычках — 'a', 'b', '\n'.

В языке C# символьная константа имеет кодировку Unicode и соответствует типу char. Так же как и в C++, здесь предусмотрены символьные escape-последовательности для записи служебных символов, таких как: «*перевод строки*» '\n', «*табуляция*» '\t' и др. Выражение \udddd или \xdddd, где dddd — шестнадцатеричное число, представляет символ Unicode. Отличием в записи кода символа является то, что в конструкции: \u... или \U... необходимо указать 4 или 8 цифр; в записи: \x... или \X... может присутствовать произвольное число цифр.

- Целочисленные константы.

Целочисленные константы интерпретируются компилятором как данные типа int. Однако при помощи суффикса L или l можно создать константу типа long, а суффикс U или u означает константу беззнакового типа.

Например: -1, 128L, 1000000000000UL.

- Константы с плавающей точкой.

Числовые константы, содержащие десятичную точку либо символ e (или E), интерпретируются как имеющие тип double, но с помощью суффикса (f или F) преобразуются в тип float, а суффикс m или M означает константу типа decimal.

Например: 3.14159, 2e-8, -1.0f, 12345m.

- Шестнадцатеричные константы.

Шестнадцатеричные константы часто используются для создания битовой маски, поскольку они очень легко преобразуются в двоичное представление. Как и в C++, они записываются с использованием префикса 0x или 0X.

Например: 0X4a, 0xff00.

- Двоичные константы.

C# 7.0 позволяет создавать целочисленные литералы — двоичные числа при помощи префикса 0b, например 0b00010010.

Также в этом обновлении стандарта был введен разделитель числовых разрядов «_», который может использоваться с любыми числовыми данными, и предыдущий пример можно записать так: 0b_0001_0010.

- Строковые константы.

Строкой называется набор символов, заключенный в двойные кавычки, например "Строковый литерал\n". Строка может включать escape-последовательности. Строки в языке C# представлены согласно правилу BSTR (Basic string or binary string). Буферу строки предшествуют 4 байта — длина строки, за которой следуют двухбайтовые символы строки в формате UTF-16, за которыми следует два нулевых байта.

Интерполяция строк

В C# 6.0 появилась возможность внедрения выражений в текстовую строку. Интерполированная строка представляет собой строковый литерал, который начинается с символа **$** и содержит одно или несколько *интерполированных выражений*. Интерполированное выражение обозначено открывающей и закрывающей фигурной скобкой { ... }. Формат выражения:

{<interpolationExpression> [,<alignment>][:<formatString>]}

То есть в аргументе метода WriteLine() мы вместо индекса можем записать выражение, причем, как и ранее, можно указать размер поля вывода и формат преобразования.

Так, в листинге 1.1 вместо строки

```
Console.WriteLine("short var = {0}, double var = {1}", s, x);
```

можно было бы написать

```
Console.WriteLine($"short var = {s}, double var = {x}");
```

Примечание

Если в интерполированную строку потребуется записать символ фигурных скобок, используйте двойные скобки "{{" или "}}".

Буквальный строковый литерал

Символ @ перед текстовым литералом заставляет воспринимать все его символы «как есть», за исключением пар фигурных скобок и двойных кавычек, что похоже на «Сырые текстовые строки C++». Что, однако, не мешает правильно интерпретировать интерполированное выражение. Особенно удобно его использовать при описании путей в файловой системе. Так, вместо

```
string pt = "c:\\Program Files\\Microsoft Office\\Office16\\";
```

можно написать

```
string pt = @"c:\Program Files\Microsoft Office\Office16\";
```

Символ @ сочетается с $ в любом порядке, что позволяет строить интерполированные выражения на буквальном строковом литерале. Так, для вывода на консоль строки pt можно написать Console.WriteLine($"{pt}");

Неявно типизированные переменные

Начиная с версии C# 3.0, появляется возможность неявного описания переменных, тип которых определяется на этапе компиляции по ее инициализирующему выражению:

var name = <выражение инициализации>;

Например, следующие пары выражений эквивалентны:

```
var r = 1;          int r = 1;
var str = "строка"; string str = "строка";
```

Описание неявно типизированной переменной возможно тогда, когда компилятор в состоянии самостоятельно определить тип этой переменной, а для этого она должна быть явно инициализирована или ей присваивается значение переменной, тип который компилятору известен.

Операции

Язык C# имеет практически тот же набор операций, что и C++. Причем, поскольку простые типы являются структурами, то корректней говорить не об операциях, а об операторах, перегруженных для всех простых типов. Отметим также, что, как и в C++, операции с плавающей точкой производятся как правило с типом double, поскольку тип float имеет малую точность, что приводит к быстрому накоплению погрешностей, а операции с типом decimal на порядок медленнее.

- Операция присваивания = (выполняется справа — налево):

  var = выражение;

Как и в языке C++, допускается многократное присваивание и использование присваивания в выражениях, например:

```
k = (j = (i += 3) + 1) + 1;
```

Здесь скобка возвращает результат последней операции.

- Унарные операции:

+x	идентификация;
-x	отрицание;
!x	логическое отрицание;
~x	побитовое отрицание;
x++	постфиксное приращение;
x−−	постфиксное уменьшение;

++x префиксное приращение;
−−x префиксное уменьшение;
(T)x явное приведение к типу T.

Примечание

1. Операторы инкремента ++ и декремента −− применимы к следующим типам: sbyte, byte, short, ushort, int, uint, long, ulong, char, float, double, decimal и любым типам перечисления.
2. При приведении типа используется правило: приведение к более общему виду производится неявно, если преобразование не приводит к усечению данных, иначе необходимо явное приведение типа.

- Бинарные операции:

x * y умножение;
x / y деление;
x % y остаток от деления (для всех числовых типов);
x + y сложение, объединение строк;
x − y вычитание;
x << y поразрядный сдвиг влево;
x >> y поразрядный сдвиг вправо.

- Комбинированные операции:

*= /= %= += -= <<= >>= &= ∧= |=.

Это сокращенный синтаксис для выражений вида

```
a = a + b;    //что эквивалентно выражению a += b;
```

Приведем в листинге 1.2 пример арифметических преобразований для вычисления остатка от деления целых чисел и чисел с плавающей точкой, а также операцию декремента и инкремента с теми же типами.

Листинг 1.2. Простейшие вычисления

```
using System;
class Program
{
static void Main()
 {
   int i = 8, j = 3, k;
     k = i % j;    //эквивалентно k = i − (i / j) * j
  Console.WriteLine($"i%j = {k}\t −i+j = {−i+j}");
   double x = -12.3, y = 5.7, z;
  z = x % y; //эквивалентно z = x − (int)(x / y) * y
  Console.WriteLine($"x % y = {z:f1}\t ++x = {++x}");
 }
}
```