Объектно-ориентированное программирование

Лекция №1. Основы языка C#

# Темы:

1. Основные понятия языка. Состав языка. Алфавит и лексемы. Идентификаторы. Ключевые слова. Операторы и разделители. Комментарии

2. Типы данных. Классификация типов. Встроенные типы. Целые и вещественные числа. Значимые и ссылочные типы.

3. Переменные. Константы. Операторы и выражения. Преобразования типов. Побитовые операторы. Логические операторы. Сокращенная запись арифметических операторов.

4. Ввод-вывод. Класс Console. Методы Write, WriteLine. Форматированный вывод. Метод ReadLine и приведение к типу.

# Алфавит и лексемы

Все тексты на языке пишутся с помощью его алфавита. Например, в русском языке один алфавит (набор символов), а в албанском — другой. В C# используется кодировка символов Unicode.

Компьютер умеет работать только с числами, и для того чтобы можно было хранить в его памяти текст, требуется определить, каким числом будет представляться (кодироваться) каждый символ. Соответствие между символами и кодирующими их числами называется кодировкой, или кодовой таблицей (character set).

Существует множество различных кодировок символов. Например, в Windows часто используется кодировка ANSI, а конкретно — СР1251. Каждый символ представляется в ней одним байтом (8 бит), поэтому в этой кодировке можно одновременно задать только 256 символов. В первой половине кодовой таблицы находятся латинские буквы, цифры, знаки арифметических операций и другие распространенные символы. Вторую половину занимают символы русского алфавита. Если требуется представлять символы другого национального алфавита (например, албанского), необходимо использовать другую кодовую таблицу.

Кодировка Unicode позволяет представить символы всех существующих алфавитов одновременно, что коренным образом улучшает переносимость текстов.

Каждому символу соответствует свой уникальный код. Естественно, что при этом для хранения каждого символа требуется больше памяти. Первые 128 Unicode-символов соответствуют первой части кодовой таблицы ANSI.

Алфавит C# включает:

* буквы (латинские и национальных алфавитов) и символ подчеркивания (\_), который употребляется наряду с буквами;
* цифры;
* специальные символы, например.+, \*, { и &;
* пробельные символы (пробел и символы табуляции);
* символы перевода строки.

Из символов составляются более крупные строительные блоки: лексемы, директивы препроцессора и комментарии.

Лексема — это минимальная единица языка, имеющая самостоятельный смысл. Существуют следующие виды лексем:

* имена (идентификаторы);
* ключевые слова;
* знаки операций;
* разделители;
* литералы (константы).

Лексемы языка программирования аналогичны словам естественного языка. Например, лексемами являются число 128 (но не его часть 12), имя Vasia, ключевое слово goto и знак операции сложения +. Далее мы рассмотрим лексемы подробнее.

Директивы препроцессора дришли в C# из его предшественника — языка C++. Препроцессором называется предварительная стадия компиляции, на которой формируется окончательный вид исходного текста программы. Например, с помощью директив (инструкций, команд) препроцессора можно включить или выключить из процесса компиляции фрагменты кода.

Комментарии предназначены для записи пояснений к программе и формирования документации. Правила записи комментариев описаны далее в этом разделе.

Из лексем составляются выражения и операторы. Выражение задает правило вычисления некоторого значения. Например, выражение а + b задает правило вычисления суммы двух величин.

Оператор задает законченное описание некоторого действия, данных или элемента программы. Например:

int а;

Это — оператор описания целочисленной переменной а.

# Идентификаторы

Имена в программах служат той же цели, что и имена в мире людей, — чтобы обращаться к программным объектам и различать их, то есть идентифицировать. Поэтому имена также называют идентификаторами. В идентификаторе могут использоваться буквы, цифры и символ подчеркивания. Прописные и строчные буквы различаются, например, sysop, SySoP и SYSOP — три разных имени.

Первым символом идентификатора может быть буква или знак подчеркивания, но не цифра. Длина идентификатора не ограничена. Пробелы внутри имен не допускаются.

В идентификаторах C# разрешается использовать помимо латинских букв буквы национальных алфавитов. Например, Пёсик или зз являются правильными идентификаторами. Более того, в идентификаторах можно применять даже так называемые escape-последовательности Unicode, то есть представлять символ с помощью его кода в шестнадцатеричном виде с префиксом \u, например, \u00F2.

Имена даются элементам программы, к которым требуется обращаться: переменным, типам, константам, методам, меткам и т. д. Идентификатор создается на этапе объявления переменной (метода, типа и т. п.), после этого его можно использовать в последующих операторах программы. При выборе идентификатора необходимо иметь в виду следующее:

* идентификатор не должен совпадать с ключевыми словами
* не рекомендуется начинать идентификаторы с двух символов подчеркивания, поскольку такие имена зарезервированы для служебного использования.

Для улучшения читабельности программы следует давать объектам осмысленные имена, составленные в соответствии с определенными правилами. Понятные и согласованные между собой имена — основа хорошего стиля программирования. Существует несколько видов так называемых нотаций — соглашений о правилах создания имен.

* В нотации Паскаля каждое слово, составляющее идентификатор, начинается с прописной буквы, например, MaxLength, MyFuzzyShooshpanchik.
* Венгерская нотация (ее предложил венгр по национальности, сотрудник компании Microsoft) отличается от предыдущей наличием префикса, соответствующего типу величины, например, iMaxLength, lpfnMyFuzzyShooshpanchik.
* Согласно нотации Camel, с прописной буквы начинается каждое слово, составляющее идентификатор, кроме первого, например, maxLength, myFuzzyShooshpanchik. Человеку с богатой фантазией абрис имени может напоминать верблюда, откуда и произошло название этой нотации.
* Еще одна традиция — разделять слова, составляющие имя, знаками подчеркивания: maxjength, my\_fuzzy\_shooshpanchik, при этом все составные части начинаются со строчной буквы.

В C# для именования различных видов программных объектов чаще всего используются две нотации: Паскаля и Camel. Многобуквенные идентификаторы в примерах этой книги соответствуют рекомендациям, приведенным в спецификации языка. Кроме того, в примерах для краткости часто используются однобуквенные имена. В реальных программах такие имена можно применять только в ограниченном наборе случаев.

# Ключевые слова

Ключевые слова — это зарезервированные идентификаторы, которые имеют специальное значение для компилятора. Их можно использовать только в том смысле, в котором они определены. Список ключевых слов C# приведен в табл. 1.

# Знаки операций и разделители

Знак операции — это один или более символов, определяющих действие над операндами. Внутри знака операции пробелы не допускаются. Например, в выражении а += b знак += является знаком операции, а а и b — операндами. Символы, составляющие знак операций, могут быть как специальными, например, &&, | и <, так и буквенными, такими как as или new.

Операции делятся на унарные, бинарные и тернарную по количеству участвующих в них операндов. Один и тот же знак может интерпретироваться по-разному в зависимости от контекста. Все знаки операций, за исключением [ ], ( ) и ? :, представляют собой отдельные лексемы.

Таблица 1. Ключевые слова C#

abstract as base bool break

byte case catch char checked

class const continue decimal default

delegate do double else enum

event explicit extern false finally

fixed float for foreach goto

if implicit in int interface

internal is lock long namespace

new null object operator out

override params private protected public

readonly ref return sbyte sealed

short sizeof stackalloc static string

struct switch this throw true

try typeof uint ulong unchecked

unsafe ushort using virtual void

volatile while

Разделители используются для разделения или, наоборот, группирования элементов. Примеры разделителей: скобки, точка, запятая. Ниже перечислены все знаки операций и разделители, использующиеся в С#:

**{} [] () . , : ; + - \* / ‘ \* & | А ! ~ = <> ? - ++ -- && || << >> == != <= >= += -= \*= /= %= &= |= ^= <<= >>= ->**

# Литералы

Литералами, или константами, называют неизменяемые величины. В C# есть логические, целые, вещественные, символьные и строковые константы, а также константа null. Компилятор, выделив константу в качестве лексемы, относит ее к одному из типов данных по ее внешнему виду. Программист может задать тип константы и самостоятельно.

Описание и примеры констант каждого типа приведены в табл. 2.

|  |  |  |
| --- | --- | --- |
| Константа | Описание | Примеры |
| Логическая | **true** (истина) или  **false** (ложь) | true  false |
| Целая | Десятичная: последовательность  десятичных цифр (0, 1, 2, 3, 4, 5, 6, 7, 8, 9), за которой может следовать суффикс (U, u, L, 1, UL, Ul, uL, ul, LU, Lu, 1U, lu)  Шестнадцатеричная: символы Ox или OX, за которыми следуют шестнадцатеричные цифры (0, 1, 2, 3, 4, 5, 6, 7, 8, 9, А, В, С, D, Е, F), а за цифрами, в свою очередь, может следовать суффикс (U, u, L, 1, UL, Ul, uL, ul, LU, Lu, Ш, lu) | 8 0 199226  8u OLu 199226L  0xA 0xlB8 0X00FF  0xAU 0x1B8LU  0X00FFl |
| Вещественная | С фиксированной точкой1:  [цифры] [.] [цифры] [суффикс]  Суффикс — один из символов F, f, D, d, М, m  С порядком:  [цифры][.][цифры]{Е|е}[+|-] [цифры] [суффикс]  Суффикс — один из символов F, f, D, d, М, m | 5.7 .001 35  5.7F .00ld 35  5F .00lf 35m |
| Символьная | Символ, заключенный в апострофы | 'A' 'ю'  '\0' '\n’  ' \xF' '\x74'  '\uA81B' |
| Строковая | Последовательность символов,  заключенная в кавычки | "Здесь был Vasia"  "\tЗначение г = \xF5 \n"  "Здесь был \u0056\u0061"  "C:\\temp\\filel.txt"  @"C:\temp\filel.txt" |
| Константа null | Ссылка, которая не указывает ни на  какой объект | null |

# Комментарии

Комментарии предназначены для записи пояснений к программе и формирования документации. Компилятор комментарии игнорирует. Внутри комментария можно использовать любые символы. В C# есть два вида комментариев: однострочные и многострочные.

Однострочный комментарий начинается с двух символов прямой косой черты (//) и заканчивается символом перехода на новую строку, многострочный заключается между символами-скобками /\* и \*/ и может занимать часть строки, целую строку или несколько строк. Комментарии не вкладываются друг в друга: символы // и /\* не обладают никаким специальным значением внутри комментария.

# Переменные

Переменная — это именованная область памяти, предназначенная для хранения данных определенного типа. Во время выполнения программы значение переменной можно изменять. Все переменные, используемые в программе, должны быть описаны явным образом. При описании для каждой переменной задаются ее имя и тип.

Пример описания целой переменной с именем а и вещественной переменной х:

int a; float х;

Имя переменной служит для обращения к области памяти, в которой хранится значение переменной. Имя дает программист. Оно должно соответствовать правилам именования идентификаторов С#, отражать смысл хранимой величины и быть легко распознаваемым. Например, если в программе вычисляется количество каких-либо предметов, лучше назвать соответствующую переменную quantity или, на худой конец, kolich, Ho не, скажем, A, tl7 или xz.

Тип переменной выбирается, исходя из диапазона и требуемой точности представления данных. Например, нет необходимости заводить переменную вещественного типа для хранения величины, которая может принимать только целые значения, — хотя бы потому, что целочисленные операции выполняются гораздо быстрее.

# Именованные константы

Можно запретить изменять значение переменной, задав при ее описании ключевое слово const, например:

const int b = 1;

const float x = 0.1, у = 0.lf; // const распространяется на обе переменные

Такие величины называют именованными константами, или просто константами. Они применяются для того, чтобы вместо значений констант можно было использовать в программе их имена. Это делает программу более понятной и облегчает внесение в нее изменений, поскольку изменить значение достаточно только в одном месте программы.

Именованные константы должны обязательно инициализироваться при описании. При инициализации можно использовать не только константу, но и выражение — главное, чтобы оно было вычисляемым на этапе компиляции, например:

const int b =.1, а = 100:

const int х = b \* а + 25;

# Операции и выражения

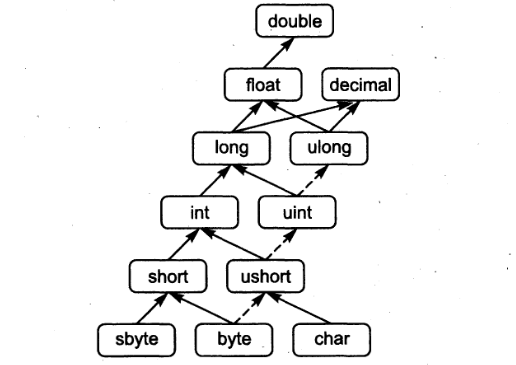
Выражение — это правило вычисления значения. В выражении участвуют операнды, объединенные знаками операций. Операндами простейшего выражения могут быть константы, переменные и вызовы функций. Например, а + 2 — это выражение, в котором + является знаком операции, а а и 2 — операндами. Пробелы внутри знака операции, состоящей из нескольких символов, не допускаются. По количеству участвующих в одной операции операндов операции делятся на унарные, бинарные и тернарную.

# Преобразования встроенных арифметических типов-значений

При вычислении выражений может возникнуть необходимость в преобразовании типов. Если операнды, входящие в выражение, одного типа и операция для этого типа определена, то результат выражения будет иметь тот же тип. Если операнды разного типа и/или операция для этого типа не определена, перед вычислениями автоматически выполняется преобразование типа по правилам, обеспечивающим приведение более коротких типов к более длинным для сохранения значимости и точности. Автоматическое (неявное) преобразование возможно не всегда, а только если при этом не может случиться потеря значимости.

Если неявного преобразования из одного типа в другой не существует, программист может задать явное преобразование типа с помощью операции (тип)х. Его результат остается на совести программиста. Явное преобразование рассматривается в этой главе немного позже.

Правила неявного преобразования иллюстрирует рис. 1. Если один из операндов имеет тип, изображенный на более низкой уровне, чем другой, то он приводится к типу второго операнда при наличии пути между ними. Если пути нет, возникает ошибка компиляции. Если путей несколько, выбирается наиболее короткий, не содержащий пунктирных линий. Преобразование выполняется не последовательно, а непосредственно из исходного типа в результирующий.

  
Иллюстрация 1: Неявные арифметические преобразования типов

# Явное преобразование типа

Операция используется, как и следует из ее названия, для явного преобразования величины из одного типа в другой. Это требуется в том случае, когда неявного преобразования не существует. При преобразовании из более длинного типа в более короткий возможна потеря информации, если исходное значение выходит за пределы диапазона результирующего типа.

Формат операции:

( тип ) выражение

Здесь тип — это имя того типа, в который осуществляется преобразование, а выражение чаще всего представляет собой имя переменной, например:

long b = 300;

int а = (int) b; // данные не теряются

byte d = (byte) а; // данные теряются

Преобразование типа часто применяется для ссылочных типов при работе с иерархиями объектов.

# Простейший ввод-вывод

Любая программа при вводе исходных данных и выводе результатов взаимодействует с внешними устройствами. Совокупность стандартных устройств ввода и вывода, то есть клавиатуры и экрана, называется консолью. Обмен данными с консолью является частным случаем обмена с внешними устройствами.

В языке С#, как и во многих других, нет операторов ввода и вывода. Вместо них для обмена с внешними устройствами применяются стандартные объекты. Для работы с консолью в C# применяется класс Console, определенный в пространстве имен System. Методы этого класса Write и WriteLine уже использовались в наших программах.

using System;

namespace ConsoleApplication1 {

class Class1 {

static void Main() {

int i = 3;

double у = 4.12;

decimal d = 600m;

string s = "Вася";

Console.WriteLine("i = " + i ); //1

Console.WriteLine("y = {0} \nd = {1}”, y, d); //2

Console.WriteLine("s = " + s ); //3

}

}

}

До сих пор мы использовали метод WriteLine для вывода значений переменных и литералов различных встроенных типов. Это возможно благодаря тому, что в классе Console существует несколько вариантов методов с именами Write и WriteLlne, предназначенных для вывода значений различных типов. Методы с одинаковыми именами, но разными параметрами называются перегруженными. Компилятор определяет, какой из методов вызван, чпо типу передаваемых в него величин. Методы вывода в классе Console перегружены для всех встроенных типов данных, кроме того, предусмотрены варианты форматного вывода.

Листинг 3.9 содержит два наиболее употребительных варианта вызова методов вывода. Сначала обратите внимание на способ вывода пояснений к значениям переменных в строках 1 и 3. Пояснения представляют собой строковые литералы. Если метод WriteLlne вызван с одним параметром, он может быть любого встроенного типа, например, числом, символом или строкой. Нам же требуется вывести в каждой строке не одну, а две величины: текстовое пояснение и значение переменной, — поэтому прежде чем передавать их для вывода, их требуется «склеить» в одну строку с помощью операции +.

Перед объединением строки с числом надо преобразовать число из его внутренней формы представления в последовательность символов, то есть в строку. Преобразование в строку определено во всех стандартных классах C# — для этого служит метод ToStrlng(). В данном случае он выполняется неявно, но можно вызвать его и явным образом:

Console.WriteLine( "i = " + i.ToString() );

Оператор 2 иллюстрирует форматный вывод. В этом случае используется другой вариант метода WriteLlne, который содержит более одного параметра. Первым параметром методу передается строковый литерал, содержащий помимо обычных символов, предназначенных для вывода на консоль, параметры в фигурных скобках, а также управляющие последовательности.

Параметры нумеруются с нуля, перед выводом они заменяются значениями соответствующих переменных в списке вывода: нулевой параметр заменяется значением первой переменной (в данном примере — у), первый параметр — второй переменной (в данном примере — d) и т. д.

Рассмотрим простейшие способы ввода с клавиатуры. В классе Console определены методы ввода строки и отдельного символа, но нет методов, которые позволяют непосредственно считывать с клавиатуры числа. Ввод числовых данных выполняется в два этапа:

1. Символы, представляющие собой число, вводятся с клавиатуры в строковую переменную.

2. Выполняется преобразование из строки в переменную соответствующего типа. Преобразование можно выполнить либо с помощью специального класса Convert, определенного в пространстве имен System, либо с помощью метода Parse, имеющегося в каждом стандартном арифметическом классе. В листинге 3.10 используются оба способа.

Листинг 3.10. Методы ввода

using System;

namespace ConsoleApplication1

{ class Classl

{ static void Main() {

Console.WriteLine( "Введите строку" );

string s = Console.ReadLine(); // 1

Console.WriteLine( "s = " + s );

Console.WriteLine( "Введите символ" );

char c = (char)Console.Read(); // 2

Console.ReadLine(); // 3

Console.WriteLine("c = " + c );

string buf; //строка-буфер для ввода чисел

Console.WriteLine("Введите целое число" );

buf = Console.ReadLine();

int i = Convert.ToInt32( buf ); // 4

Console.WriteLine(i);

Console.WriteLine("Введите вещественное число" );

buf = Console.ReadLine();

double x = Convert.ToDouble( buf ); // 5

Console.WriteLine( x );

Console.WriteLine( "Введите вещественное число" );

buf = Console.ReadLine();

double у = double.Parse( but ); // 6

Console.WriteLine( у );

Console.WriteLine( "Введите вещественное число" );

buf = Console.ReadLine();

decimal z = decimal.Parse( buf ); // 7

Console.WriteLine( z );

}

}

}

К этому примеру необходимо сделать несколько пояснений. Ввод строки выполняется в операторе 1. Длина строки не ограничена, ввод выполняется до символа перевода строки.

Ввод символа выполняется с помощью метода Read, который считывает один символ из входного потока (оператор 2). Метод возвращает значение типа int, представляющее собой код символа, или -1, если символов во входном потоке нет (например, пользователь нажал клавишу Enter). Поскольку нам требуется не int, a char, а неявного преобразования от int к char не существует, приходится применить операцию явного преобразования типа, которая описана в разделе «Явное преобразованйе типа».

За оператором 2 записан оператор 3, который считывает остаток строки и никуда его не передает. Это необходимо потому, что ввод данных выполняется через буфер — специальную область оперативной памяти. Фактически, данные сначала заносятся в буфер, а затем считываются оттуда процедурами ввода. Занесение в буфер выполняется по нажатию клавиши Enter вместе с ее кодом. Метод Read, в отличие от ReadLine, не очищает буфер, поэтому следующий после него ввод будет выполняться с того места, на котором закончился предыдущий.

В операторах 4 и 5 используются методы класса Convert, в операторах 6 и 7 — методы Parse классов Double и Decimal библиотеки .NET, которые используются здесь через имена типов C# double и decimal.

# Использованная литература

1. Павловская Т.А. C# Программирование на языке высокого уровня, Питер 2014г. 432 с.