



**Microsoft** Partner  
Silver Learning

# C# Стартовый

## ПРОЦЕДУРНОЕ ПРОГРАММИРОВАНИЕ НА ЯЗЫКЕ C#

Правила создания переменных



ITVVDN  
IT VIDEO DEVELOPERS NETWORK

## Introduction



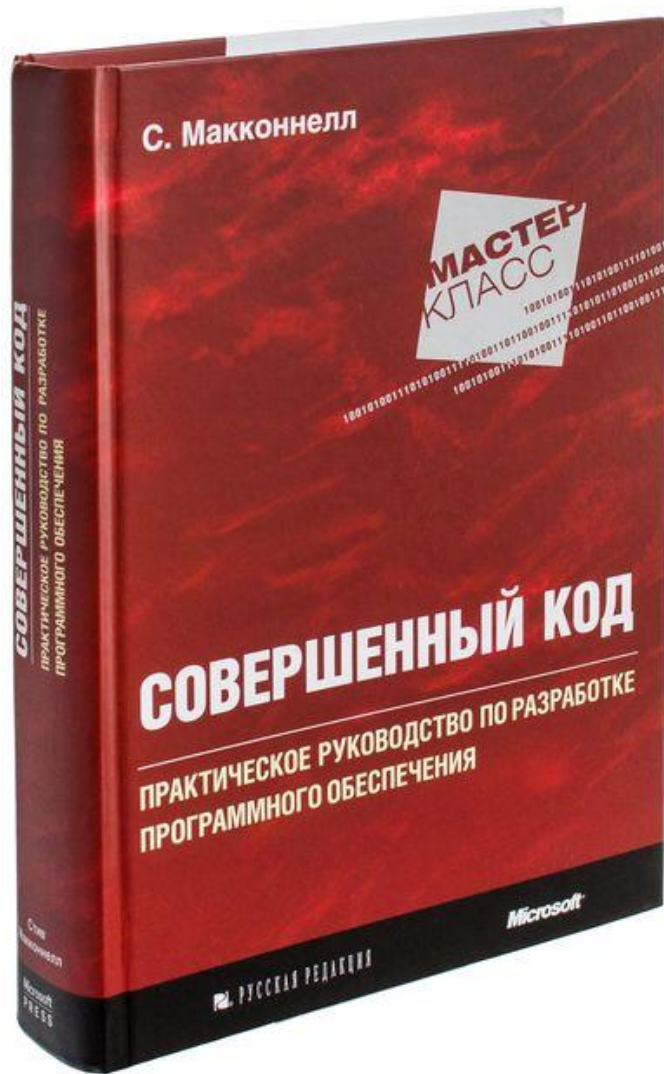
Александр Шевчук



MCID: 9230440

## Тема урока

# Правила создания переменных



## Часть III

# ПЕРЕМЕННЫЕ

- Глава 10. Общие принципы использования переменных
- Глава 11. Сила имен переменных
- Глава 12. Основные типы данных
- Глава 13. Нестандартные типы данных

## Правила именования переменных

```
int myVariable = 0;
```

Первый символ имени



Остальные символы имени

Допустимо	Недопустимо
A-Z a-z _ @	0-9 . , ? : * > < = & # ! \$ и т.п.

Допустимо	Недопустимо
A-Z a-z _ 0-9	@ . , ? : * > < = & # ! \$ и т.п.

## Правила именования переменных

1. В идентификаторах допустимо использовать символы алфавита и нижнего подчеркивания:

`myVariable, my_Variable, _MyVariable`

2. Использование цифр недопустимо в качестве первого символа имени:

`myVariable1, my1Variable, 1MyVariable`

3. Нельзя использовать в качестве идентификаторов зарезервированные ключевые слова:

`decimal, false, extern, intMyVar`

4. Использование символа @ допустимо только на первой позиции:

`@myVariable, my@Variable`

5. Язык C# чувствителен к регистру, поэтому одинаковые имена в разном регистре – это разные имена:

`myVariable, MyVariable, myvariable`

# ПРОЦЕДУРНОЕ ПРОГРАММИРОВАНИЕ НА ЯЗЫКЕ C#

## 7.4.4 Keywords

A **keyword** is an identifier-like sequence of characters that is reserved, and cannot be used as an identifier except when prefaced by the @ character.

*keyword:: one of*

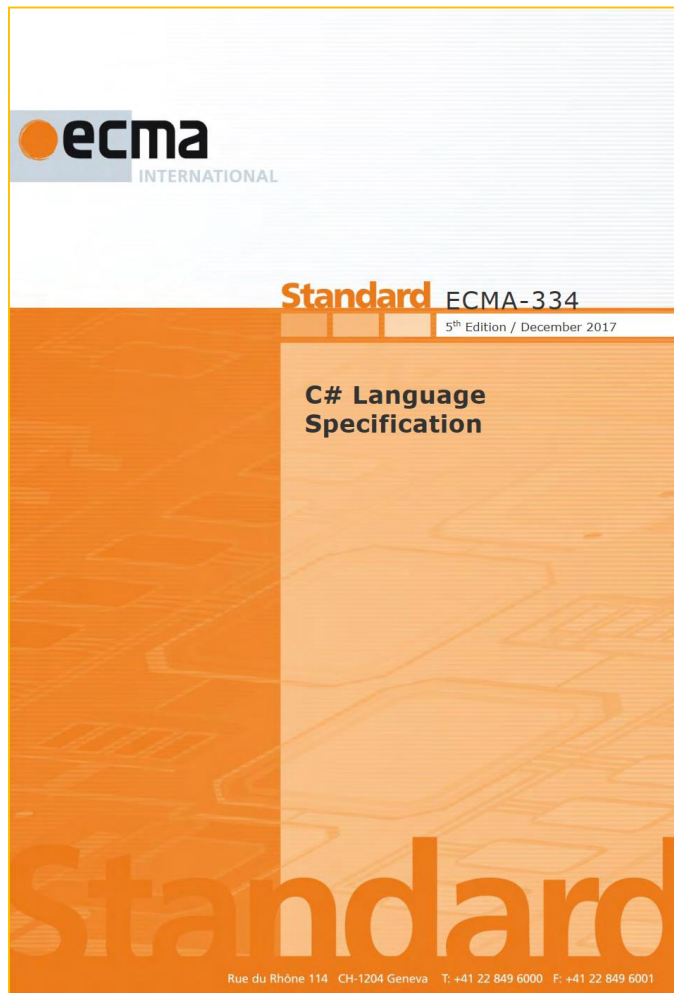
abstract	as	base	bool	break
byte	case	catch	char	checked
class	const	continue	decimal	default
delegate	do	double	else	enum
event	explicit	extern	false	finally
fixed	float	for	foreach	goto
if	implicit	in	int	interface
internal	is	lock	long	namespace
new	null	object	operator	out
override	params	private	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			

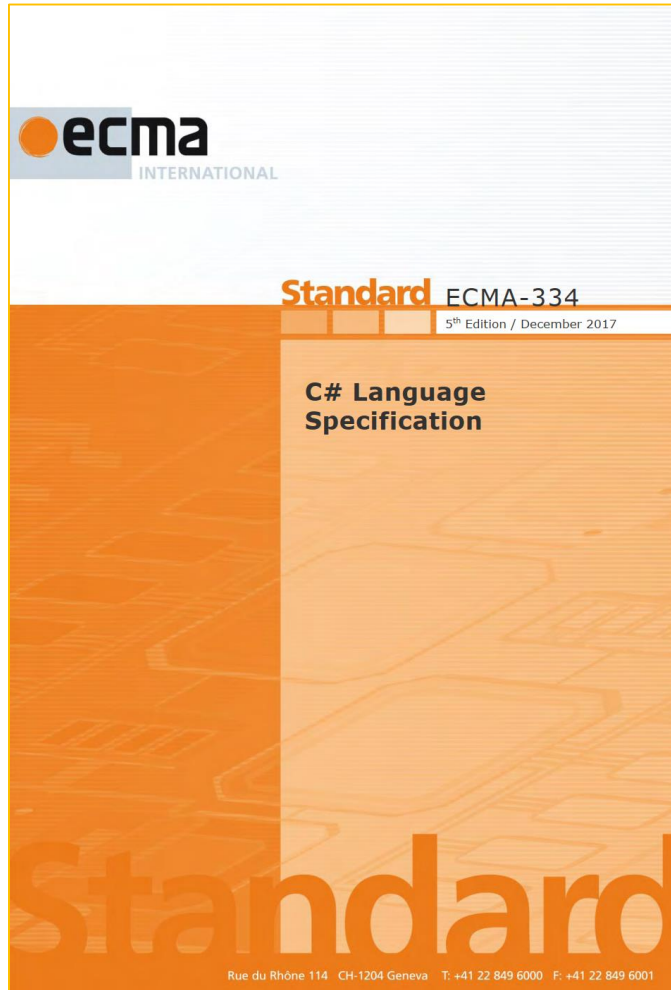
Стр. 21

A **contextual keyword** is an identifier-like sequence of characters that has special meaning in certain contexts, but is not reserved, and can be used as an identifier outside of those contexts as well as when prefaced by the @ character.

*contextual-keyword: one of the following identifiers*

add	alias	ascending	async	await
by	descending	dynamic	equals	from
get	global	group	into	join
let	orderby	partial	remove	select
set	value	var	where	yield





Стр. 19

## 7.4.3 Identifiers

The rules for identifiers given in this subclause correspond exactly to those recommended by the Unicode Standard Annex 15 except that underscore is allowed as an initial character (as is traditional in the C programming language), Unicode escape sequences are permitted in identifiers, and the “@” character is allowed as a prefix to enable keywords to be used as identifiers.


The prefix “@” enables the use of keywords as identifiers, which is useful when interfacing with other programming languages. The character @ is not actually part of the identifier, so the identifier might be seen in other languages as a normal identifier, without the prefix. An identifier with an @ prefix is called a **verbatim identifier**. [Note: Use of the @ prefix for identifiers that are not keywords is permitted, but strongly discouraged as a matter of style. *end note*]

Стр. 20



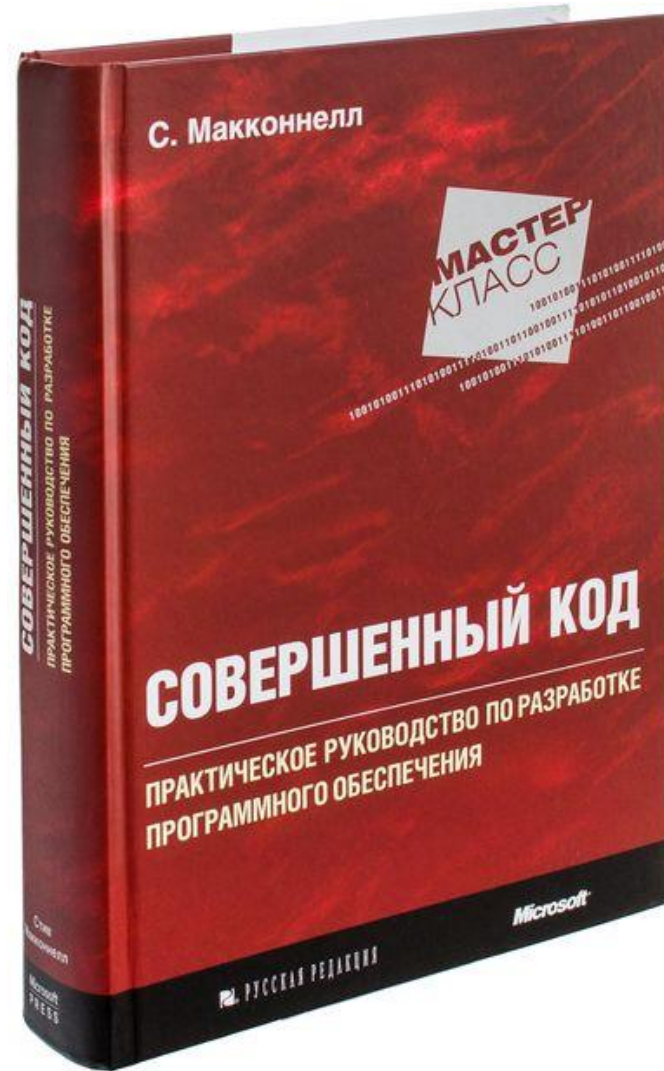
## Соглашение по именованию

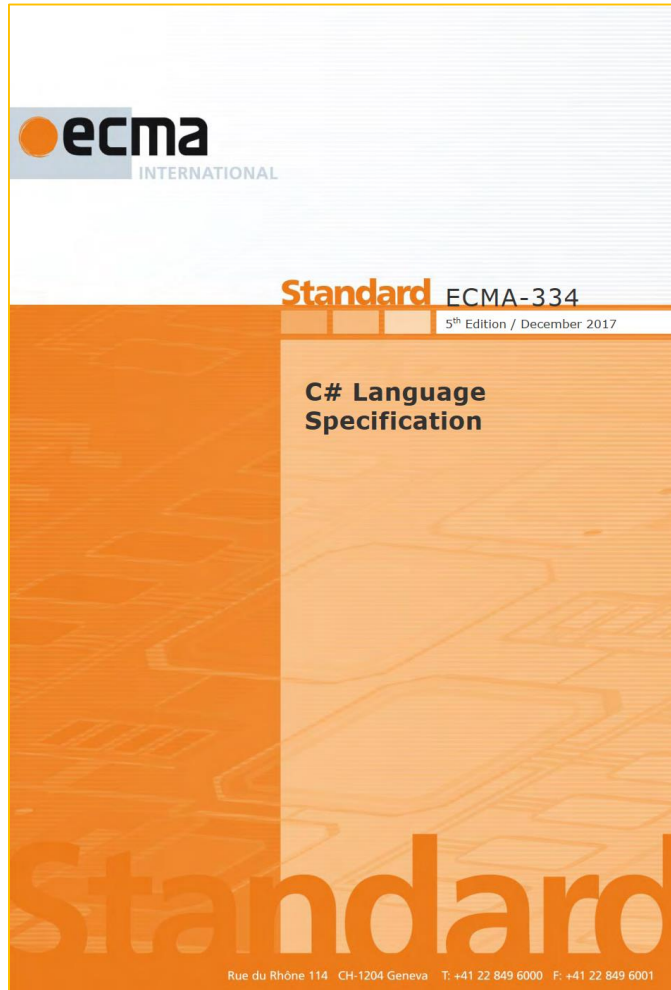
Рекомендуется придерживаться стиля «Camel casing» при создании имен переменных и в редких случаях «Uppercase»

Стиль	Описание	Пример
Pascal casing	каждое слово в идентификаторе начинается с большой буквы	MyMethod, Remove
Camel casing	каждое слово, исключая первое, в идентификаторе начинается с большой буквы	 nameOfMyNewCamel
Uppercase	идентификатор состоит из букв написанных в верхнем регистре	USA, VAT, PIN

Не рекомендуется пользоваться «Венгерской нотацией» и начинать имена с символа нижнего подчеркивания.

# ПРОЦЕДУРНОЕ ПРОГРАММИРОВАНИЕ НА ЯЗЫКЕ C#





## Introduction

This specification is based on a submission from Hewlett-Packard, Intel, and Microsoft, that described a language called C#, which was developed within Microsoft. The principal inventors of this language were Anders Hejlsberg, Scott Wiltamuth, and Peter Golde. The first widely distributed implementation of C# was released by Microsoft in July 2000, as part of its .NET Framework initiative.

Ecma Technical Committee 39 (TC39) Task Group 2 (TG2) was formed in September 2000, to produce a standard for C#. Another Task Group, TG3, was also formed at that time to produce a standard for a library and execution environment called Common Language Infrastructure (CLI). (CLI is based on a subset of the .NET Framework.) Although Microsoft's implementation of C# relies on CLI for library and run-time support, other implementations of C# need not, provided they support an alternate way of getting at the minimum CLI features required by this C# standard (see Annex C).

As the definition of C# evolved, the goals used in its design were as follows:

- C# is intended to be a simple, modern, general-purpose, object-oriented programming language.
- The language, and implementations thereof, should provide support for software engineering principles such as strong type checking, array bounds checking, detection of attempts to use uninitialized variables, and automatic garbage collection. Software robustness, durability, and programmer productivity are important.

Язык и его реализации должны обеспечивать поддержку принципов разработки программного обеспечения, таких как **строгая проверка типов**, ...

## Сильная и Слабая типизация

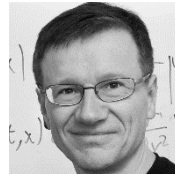
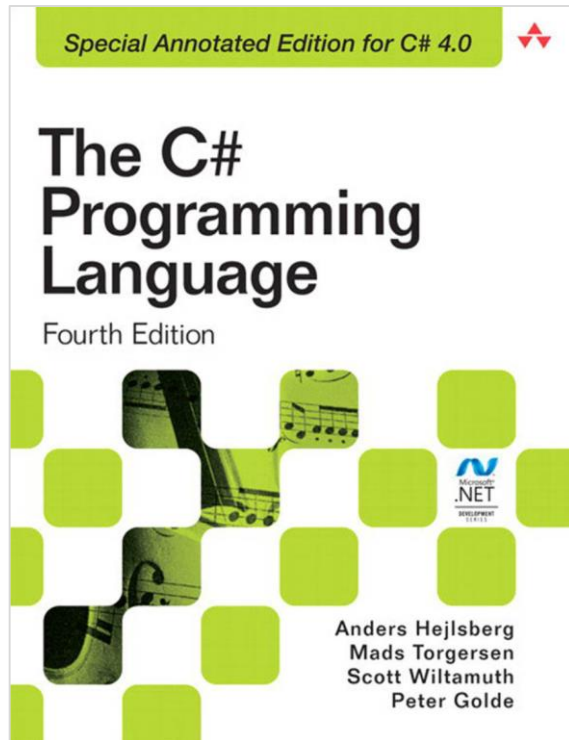
**Сильная или Строгая (Статическая) типизация** — подход, в программировании, когда переменная, связывается с типом в момент создания и тип переменной не может быть изменён позже (то есть, переменной допустимо присваивать значения только этого типа).

**Слабая (Динамическая) типизация** — подход, в программировании, когда переменная, не связывается с типом в момент создания и тип переменной ~~не~~ может быть изменён позже (то есть, переменной допустимо присваивать значения не только этого типа, **но и других типов**).

## Сильная и Слабая типизация

**Сильная или Строгая (Статическая) типизация** — подход, при котором, переменной определенного типа, можно присваивать значения, только этого определённого типа.

**Слабая (Динамическая) типизация** — подход, при котором переменная может в себя принимать разнотипные значения.



## 4.1.5 The dynamic Type

Page 167

**PETER SESTOFT** Actually, var is a reserved word, not a compile-time type, whereas dynamic is a compile-time type. The var keyword tells the compiler, «Please infer the compile-time type of this variable from its initializer expression.»

## 4.7 Тип dynamic

Стр. 178

**ПИТЕР СЕСТОФТ** Действительно, var является зарезервированным словом и для компилятора типом не является, в то время как dynamic является для компилятора типом. Ключевое слово var говорит компилятору: «Пожалуйста, определи тип для этой переменной самостоятельно, исходя из присваиваемого значения или выражения инициализации». ...



# ПРОЦЕДУРНОЕ ПРОГРАММИРОВАНИЕ НА ЯЗЫКЕ C#

Спасибо за внимание! До новых встреч!



Александр Шевчук



OLEKSANDR SHEVCHUK

Has successfully completed the requirements to be recognized as a Trainer.

Date of achievement: October 25, 2012  
Certification number: E207-8382  
Valid until: April 04, 2019

Satya Nadella  
Chief Executive Officer

Microsoft  
CERTIFIED  
Trainer

MCID: 9230440

# Информационный видеоресурс для разработчиков программного обеспечения

