Table of Contents

Language Reference

- C# Keywords
- C# Operators
- C# Special Characters
- C# Preprocessor Directives
- C# Compiler Options
- C# Compiler Errors
- C# Language Specification

C# Reference

5/22/2017 • 1 min to read • Edit Online

This section provides reference material about C# keywords, operators, special characters, preprocessor directives, compiler options, and compiler errors and warnings.

In This Section

C# Keywords
Provides links to information about C# keywords and syntax.

C# Operators
Provides links to information about C# operators and syntax.

C# Special Characters
Provides links to information about special contextual characters in C# and their usage.

C# Preprocessor Directives
Provides links to information about compiler commands for embedding in C# source code.

C# Compiler Options
Includes information about compiler options and how to use them.

C# Compiler Errors
Includes code snippets that demonstrate the cause and correction of C# compiler errors and warnings.

C# Language Specification
Provides pointers to the latest version of the C# Language Specification in Microsoft Word format.

Related Sections

C# KB articles in the Microsoft Knowledge Base
Opens a Microsoft search page for Knowledge Base articles that are available on MSDN.

C#
Provides a portal to Visual C# documentation.

Using the Visual Studio Development Environment for C#
Provides links to conceptual and task topics that describe the IDE and Editor.

C# Programming Guide
Includes information about how to use the C# programming language.

C# Keywords

6/20/2017 • 1 min to read • Edit Online

Keywords are predefined, reserved identifiers that have special meanings to the compiler. They cannot be used as identifiers in your program unless they include `@` as a prefix. For example, `@if` is a valid identifier, but `if` is not because `if` is a keyword.

The first table in this topic lists keywords that are reserved identifiers in any part of a C# program. The second table in this topic lists the contextual keywords in C#. Contextual keywords have special meaning only in a limited program context and can be used as identifiers outside that context. Generally, as new keywords are added to the C# language, they are added as contextual keywords in order to avoid breaking programs written in earlier versions.

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	in (generic modifier)	int	interface
internal	is	lock	long
namespace	new	null	object
operator	out	out (generic modifier)	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	using static
virtual	void	volatile	while

Contextual Keywords

A contextual keyword is used to provide a specific meaning in the code, but it is not a reserved word in C#. Some contextual keywords, such as `partial` and `where`, have special meanings in two or more contexts.

add	alias	ascending
async	await	descending
dynamic	from	get
global	group	into
join	let	nameof
orderby	partial (type)	partial (method)
remove	select	set
value	var	when (filter condition)
where (generic type constraint)	where (query clause)	yield

See Also

C# Reference
C# Programming Guide

C# Operators

5/10/2017 • 8 min to read • Edit Online

C# provides many operators, which are symbols that specify which operations (math, indexing, function call, etc.) to perform in an expression. You can overload many operators to change their meaning when applied to a user-defined type.

Operations on integral types (such as `==`, `!=`, `<`, `>`, `&`, `|`) are generally allowed on enumeration (`enum`) types.

The sections lists the C# operators starting with the highest precedence to the lowest. The operators within each section share the same precedence level.

Primary Operators

These are the highest precedence operators. NOTE, you can click on the operators to go the detailed pages with examples.

x.y – member access.

x?.y – null conditional member access. Returns `null` if the left-hand operand is `null`.

x?[y] - null conditional index access. Returns `null` if the left-hand operand is `null`.

f(x) – function invocation.

a[x] – aggregate object indexing.

x++ – postfix increment. Returns the value of x and then updates the storage location with the value of x that is one greater (typically adds the integer 1).

x-- – postfix decrement. Returns the value of x and then updates the storage location with the value of x that is one less (typically subtracts the integer 1).

new – type instantiation.

typeof – returns the System.Type object representing the operand.

checked – enables overflow checking for integer operations.

unchecked – disables overflow checking for integer operations. This is the default compiler behavior.

default(T) – returns the default initialized value of type T, `null` for reference types, zero for numeric types, and zero/`null` filled in members for struct types.

delegate – declares and returns a delegate instance.

sizeof – returns the size in bytes of the type operand.

-> – pointer dereferencing combined with member access.

Unary Operators

These operators have higher precedence than the next section and lower precedence than the previous section. NOTE, you can click on the operators to go the detailed pages with examples.

+x – returns the value of x.

-x – numeric negation.

!x – logical negation.

~x – bitwise complement.

++x – prefix increment. Returns the value of x after updating the storage location with the value of x that is one greater (typically adds the integer 1).

--x – prefix decrement. Returns the value of x after updating the storage location with the value of x that is one less (typically adds the integer 1).

(T)x – type casting.

await – awaits a `Task`.

&x – address of.

*x – dereferencing.

Multiplicative Operators

These operators have higher precedence than the next section and lower precedence than the previous section. NOTE, you can click on the operators to go the detailed pages with examples.

x * y – multiplication.

x / y – division. If the operands are integers, the result is an integer truncated toward zero (for example, `-7 / 2 is -3`).

x % y – modulus. If the operands are integers, this returns the remainder of dividing x by y. If `q = x / y` and `r = x % y`, then `x = q * y + r`.

Additive Operators

These operators have higher precedence than the next section and lower precedence than the previous section. NOTE, you can click on the operators to go the detailed pages with examples.

x + y – addition.

x – y – subtraction.

Shift Operators

These operators have higher precedence than the next section and lower precedence than the previous section. NOTE, you can click on the operators to go the detailed pages with examples.

x << y – shift bits left and fill with zero on the right.

x >> y – shift bits right. If the left operand is `int` or `long`, then left bits are filled with the sign bit. If the left operand is `uint` or `ulong`, then left bits are filled with zero.

Relational and Type-testing Operators

These operators have higher precedence than the next section and lower precedence than the previous section. NOTE, you can click on the operators to go the detailed pages with examples.

x < y – less than (true if x is less than y).

x > y – greater than (true if x is greater than y).

x <= y – less than or equal to.

x >= y – greater than or equal to.

is – type compatibility. Returns true if the evaluated left operand can be cast to the type specified in the right operand (a static type).

as – type conversion. Returns the left operand cast to the type specified by the right operand (a static type), but `as` returns `null` where `(T)x` would throw an exception.

Equality Operators

These operators have higher precedence than the next section and lower precedence than the previous section. NOTE, you can click on the operators to go the detailed pages with examples.

x == y – equality. By default, for reference types other than `string`, this returns reference equality (identity test). However, types can overload `==`, so if your intent is to test identity, it is best to use the `ReferenceEquals` method on `object`.

x != y – not equal. See comment for `==`. If a type overloads `==`, then it must overload `!=`.

Logical AND Operator

This operator has higher precedence than the next section and lower precedence than the previous section. NOTE, you can click on the operator to go the details page with examples.

x & y – logical or bitwise AND. Use with integer types and `enum` types is generally allowed.

Logical XOR Operator

This operator has higher precedence than the next section and lower precedence than the previous section. NOTE, you can click on the operator to go the details page with examples.

x ^ y – logical or bitwise XOR. You can generally use this with integer types and `enum` types.

Logical OR Operator

This operator has higher precedence than the next section and lower precedence than the previous section. NOTE, you can click on the operator to go the details page with examples.

x | y – logical or bitwise OR. Use with integer types and `enum` types is generally allowed.

Conditional AND Operator

This operator has higher precedence than the next section and lower precedence than the previous section. NOTE, you can click on the operator to go the details page with examples.

x && y – logical AND. If the first operand is false, then C# does not evaluate the second operand.

Conditional OR Operator

This operator has higher precedence than the next section and lower precedence than the previous section. NOTE, you can click on the operator to go the details page with examples.

x || y – logical OR. If the first operand is true, then C# does not evaluate the second operand.

Null-coalescing Operator

This operator has higher precedence than the next section and lower precedence than the previous section. NOTE, you can click on the operator to go the details page with examples.

x ?? y – returns `x` if it is non-`null`; otherwise, returns `y`.

Conditional Operator

This operator has higher precedence than the next section and lower precedence than the previous section. NOTE, you can click on the operator to go the details page with examples.

t ? x : y – if test `t` is true, then evaluate and return `x`; otherwise, evaluate and return `y`.

Assignment and Lambda Operators

These operators have higher precedence than the next section and lower precedence than the previous section. NOTE, you can click on the operators to go the detailed pages with examples.

x = y – assignment.

x += y – increment. Add the value of `y` to the value of `x`, store the result in `x`, and return the new value. If `x` designates an `event`, then `y` must be an appropriate function that C# adds as an event handler.

x -= y – decrement. Subtract the value of `y` from the value of `x`, store the result in `x`, and return the new value. If `x` designates an `event`, then `y` must be an appropriate function that C# removes as an event handler

x *= y – multiplication assignment. Multiply the value of `y` to the value of `x`, store the result in `x`, and return the new value.

x /= y – division assignment. Divide the value of `x` by the value of `y`, store the result in `x`, and return the new value.

x %= y – modulus assignment. Divide the value of `x` by the value of `y`, store the remainder in `x`, and return the new value.

x &= y – AND assignment. AND the value of `y` with the value of `x`, store the result in `x`, and return the new value.

x |= y – OR assignment. OR the value of `y` with the value of `x`, store the result in `x`, and return the new value.

x ^= y – XOR assignment. XOR the value of `y` with the value of `x`, store the result in `x`, and return the new value.

x <<= y – left-shift assignment. Shift the value of `x` left by `y` places, store the result in `x`, and return the new value.

x >>= y – right-shift assignment. Shift the value of `x` right by `y` places, store the result in `x`, and return the new value.

=> – lambda declaration.

Arithmetic Overflow

The arithmetic operators (+, -, *, /) can produce results that are outside the range of possible values for the numeric type involved. You should refer to the section on a particular operator for details, but in general:

- Integer arithmetic overflow either throws an OverflowException or discards the most significant bits of the result. Integer division by zero always throws a DivideByZeroException.

  When integer overflow occurs, what happens depends on the execution context, which can be checked or unchecked. In a checked context, an OverflowException is thrown. In an unchecked context, the most significant bits of the result are discarded and execution continues. Thus, C# gives you the choice of handling or ignoring overflow. By default, arithmetic operations occur in an *unchecked* context.

  In addition to the arithmetic operations, integral-type to integral-type casts can cause overflow (such as when you cast a long to an int), and are subject to checked or unchecked execution. However, bitwise operators and shift operators never cause overflow.

- Floating-point arithmetic overflow or division by zero never throws an exception, because floating-point types are based on IEEE 754 and so have provisions for representing infinity and NaN (Not a Number).

- Decimal arithmetic overflow always throws an OverflowException. Decimal division by zero always throws a DivideByZeroException.

See Also

C# Reference
C# Programming Guide
C#
Overloadable Operators
C# Keywords

C# Special Characters

2/22/2017 • 1 min to read • Edit Online

Special characters are predefined, contextual characters that modifies the program element (a literal string, an identifier, or an attribute name) to which they are prepended. C# supports the following special characters:

- @, the verbatim identifier character.
- $, the interpolated string character.

See Also

C# Reference
C# Programming Guide

C# Preprocessor Directives

12/14/2016 • 1 min to read • Edit Online

This section contains information about the following C# preprocessor directives.

#if

#else

#elif

#endif

#define

#undef

#warning

#error

#line

#region

#endregion

#pragma

#pragma warning

#pragma checksum

See the individual topics for more information and examples.

Although the compiler does not have a separate preprocessor, the directives described in this section are processed as if there were one. They are used to help in conditional compilation. Unlike C and C++ directives, you cannot use these directives to create macros.

A preprocessor directive must be the only instruction on a line.

See Also

C# Reference
C# Programming Guide

C# Compiler Options

12/14/2016 • 1 min to read • Edit Online

The compiler produces executable (.exe) files, dynamic-link libraries (.dll), or code modules (.netmodule).

Every compiler option is available in two forms: **-option** and **/option**. The documentation only shows the **/option** form.

In Visual Web Developer 2008, you set compiler options in the web.config file. For more information, see <compiler> Element.

In This Section

Command-line Building With csc.exe
Information about building a Visual C# application from the command line.

How to: Set Environment Variables for the Visual Studio Command Line
Provides steps for running vsvars32.bat to enable command-line builds.

Deployment of C# Applications
Describes options for deploying C# applications.

C# Compiler Options Listed by Category
A categorical listing of the compiler options.

C# Compiler Options Listed Alphabetically
An alphabetical listing of the compiler options.

Related Sections

Build Page, Project Designer
Setting properties that govern how your project is compiled, built, and debugged. Includes information about custom build steps in Visual C# projects.

Default and Custom Builds
Information on build types and configurations.

Preparing and Managing Builds
Procedures for building within the Visual Studio development environment.

C# Compiler Errors

5/27/2017 • 1 min to read • Edit Online

Some C# compiler errors have corresponding topics that explain why the error is generated, and, in some cases, how to fix the error. Use one of the following steps to see whether help is available for a particular error message.

- Find the error number (for example, CS0029) in the Output Window, and then search for it on MSDN.
- Choose the error number (for example, CS0029) in the Output Window, and then choose the F1 key.
- In the Index, enter the error number in the **Look for** box.

If none of these steps leads to information about your error, go to the end of this page, and send feedback that includes the number or text of the error.

For information about how to configure error and warning options in C#, see Build Page, Project Designer (C#).

NOTE

Your computer might show different names or locations for some of the Visual Studio user interface elements in the following instructions. The Visual Studio edition that you have and the settings that you use determine these elements. For more information, see Personalizing the IDE.

See Also

C# Compiler Options
Sorry, we don't have specifics on this C# error
Build Page, Project Designer (C#)
/warn (C# Compiler Options)
/nowarn (C# Compiler Options)

C# Language Specification

5/1/2017 • 1 min to read • Edit Online

The C# Language Specification is the definitive source for C# syntax and usage. This specification contains detailed information about all aspects of the language, including many points that the documentation for Visual C# doesn't cover.

You can download version 5.0 of this specification from the Microsoft Download Center. If you've installed Visual Studio 2015, you can also find the specification on your computer in the Program Files (x86)/Microsoft Visual Studio 14.0/VC#/Specifications/1033 folder. If you have another version of Visual Studio installed or if you installed Visual Studio in a language other than English, change the path as appropriate.

Version 6.0 of the specification is not currently available, but an unofficial draft exists in the dotnet/csharplang GitHub repository.

See Also

C# Reference
C# Programming Guide