C# programming language

Researcher Pr. Reza Azimi

C# (pronounced as *see sharp*) is a multi-paradigm programming language encompassing strong typing, imperative, declarative, functional, generic, object-oriented (class-based), and component-oriented programming disciplines. It was developed by Microsoftwithin its .NET initiative and later approved as a standard by Ecma (ECMA-334) and ISO (ISO/IEC 23270:2006). C# is one of the programming languages designed for the Common Language Infrastructure.

C# is a general-purpose, object-oriented programming language.[13] Its development team is led by Anders Hejlsberg. The most recent version is C# 7.1, which was released in 2017 along with Visual Studio 2017 Update 3

The ECMA standard lists these design goals for C#:[13]

- The language 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 typechecking, array bounds checking, detection of attempts to use uninitialized variables, and automatic garbage collection. Software robustness, durability, and programmer productivity are important.
- The language is intended for use in developing software components suitable for deployment in distributed environments.
- Portability is very important for source code and programmers, especially those already familiar with C and C++.
- Support for internationalization is very important.
- C# is intended to be suitable for writing applications for both hosted and embedded systems, ranging from the very large that use sophisticated operating systems, down to the very small having dedicated functions.
- Although C# applications are intended to be economical with regard to memory and processing power requirements, the language was not intended to compete directly on performance and size with C or assembly language.
    - During the development of the .NET Framework, the class libraries were originally written using a managed code compiler system called *Simple Managed C* (SMC).[15][16] In January 1999, Anders Hejlsberg formed a team to build a new language at the time called Cool, which stood for "C-like Object Oriented Language".[17] Microsoft had considered keeping the name "Cool" as the final name of the language, but chose not to do so for trademark reasons. By the time the .NET project was publicly announced at the July 2000 Professional Developers Conference, the language had been renamed C#, and the class libraries and ASP.NET runtime had been ported to C#.
    - Hejlsberg is C#'s principal designer and lead architect at Microsoft, and was previously involved with the design of Turbo Pascal, Embarcadero Delphi (formerly CodeGear Delphi, Inprise Delphi and Borland Delphi), and Visual J++. In interviews and technical papers he has stated that flaws[*citation needed*] in most major programming languages (e.g. C++, Java, Delphi,

and Smalltalk) drove the fundamentals of the Common Language Runtime (CLR), which, in turn, drove the design of the C# language itself.

- James Gosling, who created the Java programming language in 1994, and Bill Joy, a co-founder of Sun Microsystems, the originator of Java, called C# an "imitation" of Java; Gosling further said that "[C# is] sort of Java with reliability, productivity and security deleted."[18][19] Klaus Kreft and Angelika Langer (authors of a C++ streams book) stated in a blog post that "Java and C# are almost identical programming languages. Boring repetition that lacks innovation,"[20] "Hardly anybody will claim that Java or C# are revolutionary programming languages that changed the way we write programs," and "C# borrowed a lot from Java - and vice versa. Now that C# supports boxing and unboxing, we'll have a very similar feature in Java."[21] In July 2000, Hejlsberg said that C# is "not a Java clone" and is "much closer to C++" in its design.[22]
- Since the release of C# 2.0 in November 2005, the C# and Java languages have evolved on increasingly divergent trajectories, becoming somewhat less similar. One of the first major departures came with the addition of generics to both languages, with vastly different implementations. C# makes use of reification to provide "first-class" generic objects that can be used like any other class, with code generation performed at class-load time.[23] Furthermore, C# has added several major features to accommodate functional-style programming, culminating in the LINQ extensions released with C# 3.0 and its supporting framework of lambda expressions, extension methods, and anonymous types.[24]These features enable C# programmers to use functional programming techniques, such as closures, when it is advantageous to their application. The LINQ extensions and the functional imports help developers reduce the amount of "boilerplate" code that is included in common tasks like querying a database, parsing an xml file, or searching through a data structure, shifting the emphasis onto the actual program logic to help improve readability and maintainability.[25]
- C# used to have a mascot called Andy (named after Anders Hejlsberg). It was retired on January 29, 2004.[26]
- C# was originally submitted to the ISO subcommittee JTC 1/SC 22 for review,[27] under ISO/IEC 23270:2003,[28] was withdrawn and was then approved under ISO/IEC 23270:2006.[29]

The name "C sharp" was inspired by musical notation where a sharp indicates that the written note should be made a semitone higher in pitch.[30] This is similar to the language name of C++, where "++" indicates that a variable should be incremented by 1. The sharp symbol also resembles a ligature of four "+" symbols (in a two-by-two grid), further implying that the language is an increment of C++.[31]

Due to technical limitations of display (standard fonts, browsers, etc.) and the fact that the sharp symbol (U+266F ♯ MUSIC SHARP SIGN (HTML `&#9839;`)) is not present on most keyboard layouts, the number sign (U+0023 # NUMBER SIGN (HTML `&#35;`)) was chosen to approximate the sharp symbol in the written name of the programming language.[32] This convention is reflected in the ECMA-334 C# Language Specification.[13] However, when it is practical to do so (for example, in advertising or in box art[33]), Microsoft uses the intended musical symbol.

The "sharp" suffix has been used by a number of other .NET languages that are variants of existing languages, including J# (a .NET language also designed by Microsoft that is derived from Java 1.1), A# (from Ada), and the functional programming language F#.[34] The original implementation of Eiffel for .NET was called Eiffel#,[35] a name retired since the full Eiffel language is now supported. The suffix has also been used for libraries, such as Gtk# (a .NET wrapper for GTK+ and other GNOME libraries) and Cocoa# (a wrapper for Cocoa).

Versions

Version	Language specification			Date	.NET Framework	Visual Studio
	ECMA	ISO/IEC	Microsoft			
C# 1.0	December 2002	April 2003	January 2002	January 2002	.NET Framework 1.0	Visual Studio .NET 2002
C# 1.1 and 1.2			October 2003	April 2003	.NET Framework 1.1	Visual Studio .NET 2003
C# 2.0	June 2006	September 2006	September 2005[c]	November 2005	.NET Framework 2.0	Visual Studio 2005
C# 3.0	None[d]		August 2007	November 2007	.NET Framework 2.0 (Except LINQ)[36] .NET Framework 3.0 (Except LINQ)[36] .NET Framework 3.5	Visual Studio 2008 Visual Studio 2010
C# 4.0			April 2010	April 2010	.NET Framework 4	Visual Studio 2010
C# 5.0	In Progress[37]	None[d]	June 2013	August 2012	.NET Framework 4.5	Visual Studio 2012 Visual Studio 2013
C# 6.0	None[d]		Draft	July 2015	.NET Framework 4.6	Visual Studio 2015
C# 7.0			None	March 2017	.NET Framework 4.6.2	Visual Studio 2017
C# 7.1	None	None	None	August 2017	.NET Framework 4.6.2	Visual Studio 2017

C# 2.0

- Generics[38]

- Partial types[38]
- Anonymous methods[38]
- Iterators[38]
- Nullable types[38]
- Getter/setter separate accessibility[38]
- Method group conversions (delegates)[38]
- Co- and Contra-variance for delegates[38]
- Static classes[38]
- Delegate inference[38]

C# 3.0

- Implicitly typed local variables[39]
- Object and collection initializers[39]
- Auto-Implemented properties[39]
- Anonymous types[39]
- Extension methods[39]
- Query expressions[39]
- Lambda expression[39]
- Expression trees[39]
- Partial methods[40]

C# 4.0

- Dynamic binding[41]
- Named and optional arguments[41]
- Generic co- and contravariance[41]
- Embedded interop types ("NoPIA")[41]

C# 5.0[42]

- Asynchronous methods[43]
- Caller info attributes[43]

C# 6.0

- Compiler-as-a-service (Roslyn)
- Import of static type members into namespace[44]
- Exception filters[44]
- Await in catch/finally blocks[44]
- Auto property initializers[44]
- Default values for getter-only properties[44]
- Expression-bodied members[44]
- Null propagator (null-conditional operator, succinct null checking)[44]
- String interpolation[44]
- nameof operator[44]
- Dictionary initializer[44]

C# 7.0[45]

- Out variables
- Pattern matching
- Tuples

- Deconstruction
- Local functions
- Digit separators
- Binary literals
- Ref returns and locals
- Generalized async return types
- Expression bodied constructors and finalizers
- Expression bodied getters and setters

C# 7.1[46]

- Async main
- Default literal expressions
- Inferred tuple element names

Main article: C Sharp syntax

See also: Syntax (programming languages)

The core syntax of C# language is similar to that of other C-style languages such as C, C++ and Java. In particular:

- Semicolons are used to denote the end of a statement.
- Curly brackets are used to group statements. Statements are commonly grouped into methods (functions), methods into classes, and classes into namespaces.
- Variables are assigned using an equals sign, but compared using two consecutive equals signs.
- Square brackets are used with arrays, both to declare them and to get a value at a given index in one of them.

See also: Comparison of C Sharp and Java

Some notable features of C# that distinguish it from C, C++, and Java where noted, are:

Portability

By design, C# is the programming language that most directly reflects the underlying Common Language Infrastructure (CLI).[47] Most of its intrinsic types correspond to value-types implemented by the CLI framework. However, the language specification does not state the code generation requirements of the compiler: that is, it does not state that a C# compiler must target a Common Language Runtime, or generate Common Intermediate Language (CIL), or generate any other specific format. Theoretically, a C# compiler could generate machine code like traditional compilers of C++ or Fortran.

Typing

C# supports strongly typed implicit variable declarations with the keyword `var`, and implicitly typed arrays with the keyword `new[]` followed by a collection initializer.

C# supports a strict Boolean data type, `bool`. Statements that take conditions, such as `while` and `if`, require an expression of a type that implements the `true` operator, such as the Boolean type. While C++ also has a Boolean type, it can be freely converted to and from integers, and expressions such as `if(a)` require only that `a` is convertible to bool, allowing `a` to be an int, or a pointer. C# disallows this "integer meaning true or false" approach, on the grounds that forcing programmers to use expressions that return exactly `bool` can prevent certain types of programming

mistakes such as `if (a = b)` (use of assignment `=` instead of equality `==`, which while not an error in C or C++, will be caught by the compiler anyway).

C# is more type safe than C++. The only implicit conversions by default are those that are considered safe, such as widening of integers. This is enforced at compile-time, during JIT, and, in some cases, at runtime. No implicit conversions occur between Booleans and integers, nor between enumeration members and integers (except for literal 0, which can be implicitly converted to any enumerated type). Any user-defined conversion must be explicitly marked as explicit or implicit, unlike C++ copy constructors and conversion operators, which are both implicit by default.

C# has explicit support for covariance and contravariance in generic types, unlike C++ which has some degree of support for contravariance simply through the semantics of return types on virtual methods.

Enumeration members are placed in their own scope.

The C# language does not allow for global variables or functions. All methods and members must be declared within classes. Static members of public classes can substitute for global variables and functions.

Local variables cannot shadow variables of the enclosing block, unlike C and C++.

Meta programming

Meta programming via C# attributes is part of the language. Many of these attributes duplicate the functionality of GCC's and VisualC++'s platform-dependent preprocessor directives.

Methods and functions

Like C++, and unlike Java, C# programmers must use the keyword `virtual` to allow methods to be overridden by subclasses.

Extension methods in C# allow programmers to use static methods as if they were methods from a class's method table, allowing programmers to add methods to an object that they feel should exist on that object and its derivatives.

The type `dynamic` allows for run-time method binding, allowing for JavaScript-like method calls and run-time object composition.

C# has support for strongly-typed function pointers via the keyword `delegate`. Like the Qt framework's pseudo-C++ *signal* and *slot*, C# has semantics specifically surrounding publish-subscribe style events, though C# uses delegates to do so.

C# offers Java-like `synchronized` method calls, via the attribute `[MethodImpl(MethodImplOptions.Synchronized)]`, and has support for mutually-exclusive locks via the keyword `lock`.

Property

C# provides properties as syntactic sugar for a common pattern in which a pair of methods, accessor (getter) and mutator (setter) encapsulate operations on a single attribute of a class. No redundant method signatures for the getter/setter implementations need be written, and the property may be accessed using attribute syntax rather than more verbose method calls.

Namespace

A C# `namespace` provides the same level of code isolation as a Java `package` or a C++ **`namespace`**, with very similar rules and features to a `package`.

Memory access

In C#, memory address pointers can only be used within blocks specifically marked as *unsafe*, and programs with unsafe code need appropriate permissions to run. Most object access is done through safe object references, which always either point to a "live" object or have the well-defined null value; it is impossible to obtain a reference to a "dead" object (one that has been garbage collected), or to a random block of memory. An unsafe pointer can point to an instance of a value-type, array, string, or a block of memory allocated on a stack. Code that is not marked as unsafe can still store and manipulate pointers through the `System.IntPtr` type, but it cannot dereference them.

Managed memory cannot be explicitly freed; instead, it is automatically garbage collected. Garbage collection addresses the problem of memory leaks by freeing the programmer of responsibility for releasing memory that is no longer needed.

Exception

Checked exceptions are not present in C# (in contrast to Java). This has been a conscious decision based on the issues of scalability and versionability.[48]

Polymorphism

Unlike C++, C# does not support multiple inheritance, although a class can implement any number of interfaces. This was a design decision by the language's lead architect to avoid complication and simplify architectural requirements throughout CLI. When implementing multiple interfaces that contain a method with the same signature, C# allows implementing each method depending on which interface that method is being called through, or, like Java, allows implementing the method once, and have that be the one invocation on a call through any of the class's interfaces.

However, unlike Java, C# supports operator overloading. Only the most commonly overloaded operators in C++ may be overloaded in C#.

Language Integrated Query - LINQ

C# has the ability to utilize LINQ through the Microsoft.NET Framework with the IEnumerable Interface a developer can query any .NET collection class, XML documents, ADO.NET datasets, and SQL databases.[49] There are some advantages to using LINQ in C# and they are as follows: intellisense support, strong filtering capabilities, type safety with compile error checking ability, and brings consistency for querying data over a variety of sources.[50] There are several different language structures that can be utilized with C# with LINQ and they are query expressions, lambda expressions, anonymous types, implicitly typed variables, extension methods, and object initializers. [51]

Functional programming

Though primarily an imperative language, C# 2.0 offered limited support for functional programming through first-class functions and closures in the form of anonymous delegates. C# 3.0 expanded support for functional programming with the introduction of a lightweight syntax for lambda expressions, extension methods (an affordance for modules), and a list comprehension syntax in the form of a "query comprehension" language.

Common type system

C# has a *unified type system*. This unified type system is called Common Type System (CTS).[52]

A unified type system implies that all types, including primitives such as integers, are subclasses of the `System.Object` class. For example, every type inherits a `ToString()` method.

Categories of data types

CTS separates data types into two categories:[52]

1. Reference types
2. Value types

Instances of value types do not have referential identity nor referential comparison semantics - equality and inequality comparisons for value types compare the actual data values within the instances, unless the corresponding operators are overloaded. Value types are derived from `System.ValueType`, always have a default value, and can always be created and copied. Some other limitations on value types are that they cannot derive from each other (but can implement interfaces) and cannot have an explicit default (parameterless) constructor. Examples of value types are all primitive types, such as `int` (a signed 32-bit integer), `float` (a 32-bit IEEE floating-point number), `char` (a 16-bit Unicode code unit), and `System.DateTime` (identifies a specific point in time with nanosecond precision). Other examples are **`enum`** (enumerations) and **`struct`** (user defined structures).

In contrast, reference types have the notion of referential identity - each instance of a reference type is inherently distinct from every other instance, even if the data within both instances is the same. This is reflected in default equality and inequality comparisons for reference types, which test for referential rather than structural equality, unless the corresponding operators are overloaded (such as the case for `System.String`). In general, it is not always possible to create an instance of a reference type, nor to copy an existing instance, or perform a value comparison on two existing instances, though specific reference types can provide such services by exposing a public constructor or implementing a corresponding interface (such as `ICloneable` or `IComparable`). Examples of reference types are `object` (the ultimate base class for all other C# classes), `System.String` (a string of Unicode characters), and `System.Array` (a base class for all C# arrays).

Both type categories are extensible with user-defined types.

Boxing and unboxing

Boxing is the operation of converting a value-type object into a value of a corresponding reference type.[52] Boxing in C# is implicit.

Unboxing is the operation of converting a value of a reference type (previously boxed) into a value of a value type.[52] Unboxing in C# requires an explicit type cast. A boxed object of type T can only be unboxed to a T (or a nullable T).[53]

Example:

```
int foo = 42;         // Value type.
object bar = foo;     // foo is boxed to bar.
int foo2 = (int)bar;  // Unboxed back to value type.
```

The C# specification details a minimum set of types and class libraries that the compiler expects to have available. In practice, C# is most often used with some implementation of the Common

Language Infrastructure (CLI), which is standardized as ECMA-335 *Common Language Infrastructure (CLI)*.

The following is a very simple C# program, a version of the classic "Hello world" example:

```
using System;

class Program
{
    static void Main(string[] args)
    {
        Console.WriteLine("Hello, world!");
    }
}
```

What will display on the program is:

```
Hello, world!
```

Each line has a purpose:

```
using System;
```

The above line of code tells the compiler to use `System` as a candidate prefix for types used in the source code. In this case, when the compiler sees use of the `Console` type later in the source code, it tries to find a type named `Console`, first in the current assembly, followed by all referenced assemblies. In this case the compiler fails to find such a type, since the name of the type is actually `System.Console`. The compiler then attempts to find a type named `System.Console` by using the `System` prefix from the **`using`** statement, and this time it succeeds.
The **`using`** statement allows the programmer to state all candidate prefixes to use during compilation instead of always using full type names.

```
class Program
```

Above is a class definition. Everything between the following pair of braces describes `Program`.

```
static void Main(string[] args)
```

This declares the class member method where the program begins execution. The .NET runtime calls the `Main` method. (Note: `Main` may also be called from elsewhere, like any other method, e.g. from another method of `Program`.) The **`static`** keyword makes the method accessible without an instance of `Program`. Each console application's `Main` entry point must be declared **`static`**.

Otherwise, the program would require an instance, but any instance would require a program. To avoid that irresolvable circular dependency, C# compilers processing console applications (like that above) report an error, if there is no `static Main` method. The `void` keyword declares that `Main` has no return value.

```
Console.WriteLine("Hello, world!");
```

This line writes the output. `Console` is a static class in the `System` namespace. It provides an interface to the standard input, output, and error streams for console applications. The program calls the `Console` method `WriteLine`, which displays on the console a line with the argument, the string `"Hello, world!"`.

A GUI example:

```
using System.Windows.Forms;

class Program
{
    static void Main(string[] args)
    {
        MessageBox.Show("Hello, World!");
        System.Console.WriteLine("Is almost the same argument!");
    }
}
```

This example is similar to the previous example, except that it generates a dialog box that contains the message "Hello, World!" instead of writing it to the console.

Standardization and licensing

In August 2001, Microsoft Corporation, Hewlett-Packard and Intel Corporation co-sponsored the submission of specifications for C# as well as the Common Language Infrastructure (CLI) to the standards organization Ecma International. In December 2001, ECMA released ECMA-334 *C# Language Specification*. C# became an ISO standard in 2003 (ISO/IEC 23270:2003 - *Information technology — Programming languages — C#*). ECMA had previously adopted equivalent specifications as the 2nd edition of C#, in December 2002.

In June 2005, ECMA approved edition 3 of the C# specification, and updated ECMA-334. Additions included partial classes, anonymous methods, nullable types, and generics(somewhat similar to C++ templates).

In July 2005, ECMA submitted to ISO/IEC JTC 1, via the latter's Fast-Track process, the standards and related TRs. This process usually takes 6–9 months.

The C# language definition and the CLI are standardized under ISO and Ecma standards that provide reasonable and non-discriminatory licensing protection from patent claims.

Microsoft has agreed not to sue open source developers for violating patents in non-profit projects for the part of the framework that is covered by the OSP.[54] Microsoft has also agreed not to enforce

patents relating to Novell products against Novell's paying customers[55] with the exception of a list of products that do not explicitly mention C#, .NET or Novell's implementation of .NET (The Mono Project).[56] However, Novell maintains that Mono does not infringe any Microsoft patents.[57] Microsoft has also made a specific agreement not to enforce patent rights related to the Moonlight browser plugin, which depends on Mono, provided it is obtained through Novell.[58]

Implementations

Microsoft is leading the development of the open-source reference C# compiler and set of tools, previously codenamed "Roslyn". The compiler, which is entirely written in managed code (C#), has been opened up and functionality surfaced as APIs. It is thus enabling developers to create refactoring and diagnostics tools.[59][60] While other implementations of C# exist, Visual C# is by far the one most commonly used.[61]

Other C# compilers, which often including an implementation of the Common Language Infrastructure and the .NET class libraries up to .NET 2.0:

- The Mono project provides an open-source C# compiler, a complete open-source implementation of the Common Language Infrastructure including the required framework libraries as they appear in the ECMA specification, and a nearly complete implementation of the Microsoft proprietary .NET class libraries up to .NET 3.5. As of Mono 2.6, no plans exist to implement WPF; WF is planned for a later release; and there are only partial implementations of LINQ to SQL and WCF.[62]
- The DotGNU project (now discontinued) also provided an open-source C# compiler, a nearly complete implementation of the Common Language Infrastructure including the required framework libraries as they appear in the ECMA specification, and subset of some of the remaining Microsoft proprietary .NET class libraries up to .NET 2.0 (those not documented or included in the ECMA specification, but included in Microsoft's standard .NET Framework distribution).
- Microsoft's Shared Source Common Language Infrastructure, codenamed "Rotor", provides a shared source implementation of the CLR runtime and a C# compiler licensed for educational and research use only, and a subset of the required Common Language Infrastructure framework libraries in the ECMA specification (up to C# 2.0, and supported on Windows XP only).

References

1. **Jump up^** https://www.infoq.com/minibooks/emag-c-sharp-preview
2. **Jump up^** *"What's new in C#". Microsoft Docs. Microsoft. Retrieved 2017-10-09.*
3. **Jump up^** *Torgersen, Mads (October 27, 2008). "New features in C# 4.0". Microsoft. Retrieved October 28, 2008.*
4. **Jump up^** https://github.com/dotnet/coreclr/blob/master/LICENSE.TXT
5. ^ Jump up to:[a] [b] *Naugler, David (May 2007). "C# 2.0 for C++ and Java programmer: conference workshop". Journal of Computing Sciences in Colleges.* ***22*** *(5). Although C# has been strongly influenced by Java it has also been strongly influenced by C++ and is best viewed as a descendant of both C++ and Java.*
6. **Jump up^** *Hamilton, Naomi (October 1, 2008). "The A-Z of Programming Languages: C#". Computerworld. Retrieved February 12, 2010. We all stand on the shoulders of giants here and every language builds on what went before it so we owe a lot to C, C++, Java, Delphi, all of these other things that came before us. (Anders Hejlsberg)*
7. **Jump up^** *"Chapel spec (Acknowlegements)" (PDF). Cray Inc. 2015-10-01. Retrieved 2016-01-14.*
8. **Jump up^** *Borenszweig, Ary. "Crystal 0.18.0 released!". It's heavily inspired by Ruby, and other languages (like C#, Go and Python).*
9. **Jump up^** *"Web Languages and VMs: Fast Code is Always in Fashion. (V8, Dart) - Google I/O 2013". Google. Retrieved 22 December 2013.*
10. **Jump up^** Java 5.0 added several new language features (the enhanced for loop, autoboxing, varargs and annotations), after they were introduced in the similar (and competing) C# language [1] [2]
11. **Jump up^** *Cornelius, Barry (December 1, 2005). "Java 5 catches up with C#". University of Oxford Computing Services. Retrieved June 18, 2014. In my opinion, it is C# that has caused these radical changes to the Java language. (Barry Cornelius)*
12. **Jump up^** *Lattner, Chris (2014-06-03). "Chris Lattner's Homepage". Chris Lattner. Retrieved 2014-06-03. The Swift language is the product of tireless effort from a team of language experts, documentation gurus, compiler optimization ninjas, and an incredibly important internal dogfooding group who provided feedback to help refine and battle-test ideas. Of course, it also greatly benefited from the experiences hard-won by many other languages in the field, drawing ideas from Objective-C, Rust, Haskell, Ruby, Python, CLU, and far too many others to list.*
13. ^ Jump up to:[a] [b] [c] *C# Language Specification (PDF) (4th ed.). Ecma International. June 2006. Retrieved January 26, 2012.*
14. **Jump up^** *"What's new in C#". Microsoft Docs. Retrieved 9 October 2017.*
15. **Jump up^** *Zander, Jason (November 24, 2008). "Couple of Historical Facts". Retrieved February 23, 2009.*
16. **Jump up^** *Guthrie, Scott (November 28, 2006). "What language was ASP.Net originally written in?". Retrieved February 21, 2008.*
17. **Jump up^** *Hamilton, Naomi (October 1, 2008). "The A-Z of Programming Languages: C#". Computerworld. Retrieved October 1, 2008.*
18. **Jump up^** *Wylie Wong (2002). "Why Microsoft's C# isn't". CNET: CBS Interactive. Retrieved May 28, 2014.*
19. **Jump up^** *Bill Joy (February 7, 2002). "Microsoft's blind spot". cnet.com. Retrieved January 12,2010.*
20. **Jump up^** *Klaus Kreft and Angelika Langer (2003). "After Java and C# - what is next?". Retrieved June 18, 2013.*
21. **Jump up^** *Klaus Kreft and Angelika Langer (July 3, 2003). "After Java and C# - what is next?". artima.com. Retrieved January 12, 2010.*
22. **Jump up^** *Osborn, John (August 1, 2000). "Deep Inside C#: An Interview with Microsoft Chief Architect Anders Hejlsberg". O'Reilly Media. Retrieved November 14, 2009*
23. **Jump up^** *"Generics (C# Programming Guide)". Microsoft. Retrieved March 21, 2011.*
24. **Jump up^** *Don Box and Anders Hejlsberg (February 2007). "LINQ: .NET Language-Integrated Query". Microsoft. Retrieved March 21, 2011.*
25. **Jump up^** *Mercer, Ian (April 15, 2010). "Why functional programming and LINQ is often better than procedural code". abodit.com. Retrieved March 21, 2011.*

26. **Jump up^** *"Andy Retires". Dan Fernandez's Blog. Blogs.msdn.com. January 29, 2004. Retrieved October 4, 2012.*
27. **Jump up^** *"Technical committees - JTC 1/SC 22 - Programming languages, their environments and system software interfaces". ISO. Retrieved October 4, 2012.*
28. **Jump up^** *"ISO/IEC 23270:2003 - Information technology - C# Language Specification". Iso.org. August 23, 2006. Retrieved October 4, 2012.*
29. **Jump up^** *"ISO/IEC 23270:2006 - Information technology - Programming languages - C#". Iso.org. January 26, 2012. Retrieved October 4, 2012.*
30. **Jump up^** *Kovacs, James (September 7, 2007). "C#/.NET History Lesson". Retrieved June 18,2009.*
31. **Jump up^** *Hejlsberg, Anders (October 1, 2008). "The A-Z of Programming Languages: C#". Computerworld.*
32. **Jump up^** *"Microsoft C# FAQ". Microsoft. Archived from the original on February 14, 2006. Retrieved March 25, 2008.*
33. **Jump up^** *"Visual C#.net Standard" (JPEG). Microsoft. September 4, 2003. Retrieved June 18,2009.*
34. **Jump up^** *"F# FAQ". Microsoft Research. Archived from the original on February 18, 2009. Retrieved June 18, 2009.*
35. **Jump up^** *Simon, Raphael; Stapf, Emmanuel; Meyer, Bertrand (June 2002). "Full Eiffel on the .NET Framework". Microsoft. Retrieved June 18, 2009.*
36. ^ Jump up to:[a] [b] *"Using C# 3.0 from .NET 2.0". Danielmoth.com. May 13, 2007. Retrieved October 4, 2012.*
37. **Jump up^** *"Mono and Roslyn". Tirania Blog. Miguel de Icaza. Retrieved 9 April 2014.*Work in progress for C# 5.0.
38. ^ Jump up to:[a] [b] [c] [d] [e] [f] [g] [h] [i] [j] *"What's new in the C# 2.0 Language and Compiler". Microsoft Developer Network. Microsoft. Retrieved 11 June 2014.*
39. ^ Jump up to:[a] [b] [c] [d] [e] [f] [g] [h] *Hejlsberg, Anders; Torgersen, Mads. "Overview of C# 3.0". Microsoft Developer Network. Microsoft. Retrieved 11 June 2014.*
40. **Jump up^** *Ghosh, Wriju. "C# 3.0 : Partial * Garbage Collection Methods". MSDN Blogs. Microsoft. Retrieved 11 June 2014.*
41. ^ Jump up to:[a] [b] [c] [d] *Burrows, Chris. "C# 4.0 - New C# Features in the .NET Framework 4". Microsoft Developer Network. Microsoft. Retrieved 11 June 2014.*
42. **Jump up^** *Hejlsberg, Anders. "Future directions for C# and Visual Basic". C# lead architect. Microsoft. Retrieved September 21, 2011.*
43. ^ Jump up to:[a] [b] *"An Introduction to New Features in C# 5.0". MSDN Blogs. Microsoft. Retrieved 11 June 2014.*
44. ^ Jump up to:[a] [b] [c] [d] [e] [f] [g] [h] [i] [j] *"Language feature implementation status". github. Microsoft. Retrieved 13 February 2015.*
45. **Jump up^** *"New Features in C# 7.0". .NET Blog. Retrieved 2017-06-09.*
46. **Jump up^** *"What's new in C# 7.1". Microsoft Docs. Retrieved 2017-10-09.*
47. **Jump up^** *Visual Studio 2010 and .NET 4 Six-in-One. Wrox Press. 2010. ISBN 0470499486.*
48. **Jump up^** *Venners, Bill; Eckel, Bruce (August 18, 2003). "The Trouble with Checked Exceptions". Retrieved March 30, 2010.*
49. **Jump up^** X. D. Zhang et al., "Research of the Database Access Technology Under.NET Framework", Applied Mechanics and Materials, Vols. 644-650, pp. 3077-3080, 2014
50. **Jump up^** Otey, M. (2006, 02). LINQ to the future. SQL Server Magazine, 8, 17-21. Retrieved from http://ezaccess.libraries.psu.edu/login?url=https://search-proquest-com.ezaccess.libraries.psu.edu/docview/214859896?accountid=13158
51. **Jump up^** Sheldon, W. (2010, 11). New features in LINQ. SQL Server Magazine, 12, 37-40. Retrieved from http://ezaccess.libraries.psu.edu/login?url=https://search-proquest-com.ezaccess.libraries.psu.edu/docview/770609095?accountid=13158
52. ^ Jump up to:[a] [b] [c] [d] *Archer, Tom (2001). "Part 2, Chapter 4: The Type System". Inside C#. Redmond, Washington: Microsoft Press. ISBN 0-7356-1288-9.*
53. **Jump up^** *Lippert, Eric (March 19, 2009). "Representation and Identity". Fabulous Adventures In Coding. Blogs.msdn.com. Retrieved October 4, 2012.*
54. **Jump up^** *"Patent Pledge for Open Source Developers".*
55. **Jump up^** *"Patent Cooperation Agreement - Microsoft & Novell Interoperability Collaboration". Microsoft. November 2, 2006. Retrieved July 5, 2009. Microsoft, on behalf of itself and*

its Subsidiaries (collectively "Microsoft"), hereby covenants not to sue Novell's Customers and Novell's Subsidiaries' Customers for infringement under Covered Patents of Microsoft on account of such a Customer's use of specific copies of a Covered Product as distributed by Novell or its Subsidiaries (collectively "Novell") for which Novell has received Revenue (directly or indirectly) for such specific copies; provided the foregoing covenant is limited to use by such Customer (i) of such specific copies that are authorized by Novell in consideration for such Revenue, and (ii) within the scope authorized by Novell in consideration for such Revenue.

56. **Jump up^** *"Definitions". Microsoft. November 2, 2006. Retrieved July 5, 2009.*
57. **Jump up^** *Steinman, Justin (November 7, 2006). "Novell Answers Questions from the Community". Retrieved July 5, 2009. We maintain that Mono does not infringe any Microsoft patents.*
58. **Jump up^** *"Covenant to Downstream Recipients of Moonlight - Microsoft & Novell Interoperability Collaboration". Microsoft. September 28, 2007. Retrieved March 8, 2008. "Downstream Recipient" means an entity or individual that uses for its intended purpose a Moonlight Implementation obtained directly from Novell or through an Intermediate Recipient... Microsoft reserves the right to update (including discontinue) the foregoing covenant... "Moonlight Implementation" means only those specific portions of Moonlight 1.0 or Moonlight 1.1 that run only as a plug-in to a browser on a Personal Computer and are not licensed under GPLv3 or a Similar License.*
59. **Jump up^** https://github.com/dotnet/roslyn
60. **Jump up^** https://docs.microsoft.com/en-us/dotnet/articles/csharp/csharp
61. **Jump up^** Watson, K et al., (2010). Beginning Visual C# 2010. Indianapolis, Indiana: Wiley.
62. **Jump up^** *"Compatibility - Mono". Mono-project.com. December 19, 2011. Retrieved October 4,2012.*