SUBJECT-PROGRAMMING IN NET WITH C#

[Bca-IIIrd Yr]

general-purpose high-level programming language supporting multiple paradigms. C# encompasses static typing,[16]:4 strong typing, lexically scoped, imperative, declarative, functional, generic,[16]:22 object-oriented (class-based), and component-oriented programming disciplines.[17]

The principal inventors of the C# programming language were Anders Hejlsberg, Scott Wiltamuth, and Peter Golde from Microsoft.[17] It was first widely distributed in July 2000[17] and was later approved as an international standard by Ecma (ECMA-334) in 2002 and ISO/IEC (ISO/IEC 23270 and 20619[c]) in 2003. Microsoft introduced C# along with .NET Framework and Visual Studio, both of which were closed-source. At the time, Microsoft had no open-source products. Four years later, in 2004, a free and open-source project called Mono began, providing a cross-platform compiler and runtime environment for the C# programming language. A decade later, Microsoft released Visual Studio Code (code editor), Roslyn (compiler), and the unified .NET platform (software framework), all of which support C# and are free, open-source, and cross-platform. Mono also joined Microsoft but was not merged into .NET.

As of November 2023, the most recent stable version of the language is C# 12.0, which was released in 2023 in .NET 8.0.[18][19]

Design goals

The Ecma standard lists these design goals for C#:[17]

- 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 type checking, array bounds checking,[20]:58–59 detection of attempts to use uninitialized variables, and automatic garbage collection.[20]:563 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[20]:314 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.[21]

History

During the development of the .NET Framework, the class libraries were originally written using a managed code compiler system named *Simple Managed C* (SMC).[22][23] 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".[24] 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[25] 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.

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."[26][27] In July 2000, Hejlsberg said that C# is "not a Java clone" and is "much closer to C++" in its design.[28]

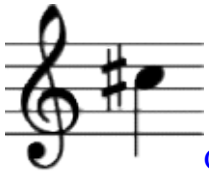
Since the release of C# 2.0 in November 2005, the C# and Java languages have evolved on increasingly divergent trajectories, becoming two quite different languages. One of the first major departures came with the addition of generics to both languages, with vastly different implementations. C# makes 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.[29] 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.[30] 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.[31]

C# used to have a mascot called Andy (named after Anders Hejlsberg). It was retired on January 29, 2004.[32]

C# was originally submitted to the ISO/IEC JTC 1 subcommittee SC 22 for review,[33] under ISO/IEC 23270:2003,[34] was withdrawn and was then approved under ISO/IEC 23270:2006.[35] The 23270:2006 is withdrawn under 23270:2018 and approved with this version.[36]

Name

icrosoft first used the name C# in 1988 for a variant of the C language designed for incremental compilation.[37] That project was not completed, and the name was later reused.



C-sharp musical note

The name "C sharp" was inspired by the musical notation whereby a sharp symbol indicates that the written note should be made a semitone higher in pitch.[38] This is similar to the language name of C++, where "++" indicates that a variable should be incremented by 1 after being evaluated. 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++.[39]

Due to technical limits of display (standard fonts, browsers, etc.), and most keyboard layouts lacking a sharp symbol (U+266F ♯ MUSIC SHARP SIGN (♯)), the number sign (U+0023 # NUMBER SIGN (#)) was chosen to approximate the sharp symbol in the written name of the programming language.[40] This convention is reflected in the ECMA-334 C# Language Specification.[17]

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#.[41] The original implementation of Eiffel for .NET was called Eiffel#,[42] 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

C# version	Language specification			Date	.NET	Visual Studio
	Ecma	ISO/IEC	Microsoft			
1.0	ECMA-334:2003, December 2002	ISO/IEC 23270:2003, April 2003	January 2002	January 2002	.NET Framework 1.0	Visual Studio .NET 2002
1.1 1.2			October 2003	April 2003	.NET Framework 1.1	Visual Studio .NET 2003
2.0[43]	ECMA-334:2006, June 2006	ISO/IEC 23270:2006, September 2006	September 2005[d]	November 2005	.NET Framework 2.0 .NET Framework 3.0	Visual Studio 2005 Visual Studio 2008
3.0[44]	None		August 2007	November 2007	.NET Framework 2.0 (Except LINQ)[45]	Visual Studio 2008

4.0[46]			April 2010	April 2010	.NET Framework 3.0 (Except LINQ)[45] .NET Framework 3.5 .NET Framework 4	Visual Studio 2010
5.0[47]	ECMA-334:2017, December 2017	ISO/IEC 23270:2018, December 2018	June 2013	August 2012	.NET Framework 4.5	Visual Studio 2012 Visual Studio 2013
6.0[48]	ECMA-334:2022, June 2022	None	Draft	July 2015	.NET Framework 4.6 .NET Core 1.0 .NET Core 1.1	Visual Studio 2015
7.0[49][50]			Specification proposal	March 2017	.NET Framework 4.7	Visual Studio 2017 version 15.0[51]
7.1[52]			Specification proposal	August 2017	.NET Core 2.0	Visual Studio 2017 version 15.3[53]
7.2[54]	ECMA-334:2023, December 2023	ISO/IEC 20619:2023, September 2023	Specification proposal	November 2017		Visual Studio 2017 version 15.5[55]
7.3[56]			Specification proposal Archived March 7, 2021, at the Wayback Machine	May 2018	.NET Core 2.1 .NET Core 2.2 .NET	Visual Studio 2017 version 15.7[57]

				Framework 4.8	
8.0[58]	None	Specification proposal	September 2019	.NET Core 3.0 .NET Core 3.1	Visual Studio 2019 version 16.3[59]
9.0[60]		Specification proposal	November 2020	.NET 5.0	Visual Studio 2019 version 16.8[61]
10.0[62]		Specification proposal	November 2021	.NET 6.0	Visual Studio 2022 version 17.0[63]
11.0[64]		Specification proposal	November 2022	.NET 7.0	Visual Studio 2022 version 17.4[65]
12.0[66]		Specification proposal	November 2023	.NET 8.0	Visual Studio 2022 version 17.8[67]

Syntax

: C# syntax

See also: Syntax (programming languages)

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

- 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.

Distinguishing features

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).[68] 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. Some C# compilers can also generate machine code like traditional compilers of C++ or Fortran.[69][70]

Typing

[

C# supports strongly, implicitly typed variable declarations with the keyword `var`,[16]:470 and implicitly typed arrays with the keyword `new[]` followed by a collection initializer.[16]:80[20]:58

Its type system is split into two families: Value types, like the built-in numeric types and user-defined structs, which are automatically handed over as copies when used as parameters, and reference types, including arrays, instances of classes, and strings, which only hand over a pointer to the respective object. Due to their special handling of the equality operator, strings will nevertheless behave as if they were values, for all practical purposes. You can even use them as case labels. Where necessary, value types will be boxed automatically.[71]

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 `==`).

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,[16]:144[20]:23 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++.

Metaprogramming

- Reflection is supported through .NET APIs, which enable scenarios such as type metadata inspection and dynamic method invocation.
- Expression trees[72] represent code as an abstract syntax tree, where each node is an expression that can be inspected or executed. This enables dynamic modification of executable code at runtime. Expression trees introduce some homoiconicity to the language.
- Attributes are metadata that can be attached to types, members, or entire assemblies, equivalent to annotations in Java. Attributes are accessible both to the compiler and to code through reflection. Many of native attributes duplicate the functionality of GCC's and VisualC++'s platform-dependent preprocessor directives.[*citation needed*]
- `System.Reflection.Emit` namespace,[73] which contains classes that emit metadata and CIL (types, assemblies, etc.) at runtime.
- The .NET Compiler Platform (Roslyn) provides API access to language compilation services, allowing for the compilation of C# code from within .NET applications. It exposes APIs for syntactic (lexical) analysis of code, semantic analysis, dynamic compilation to CIL, and code emission.[74]
- Source generators,[75] a feature of the Roslyn C# compiler, enable compile time metaprogramming. During the compilation process, developers can inspect the code being compiled with the compiler's API and pass additional generated C# source code to be compiled.

Methods and functions

A *method* in C# is a member of a class that can be invoked as a function (a sequence of instructions), rather than the mere value-holding capability of a *field* (i.e. class or instance variable).[76] As in other syntactically similar languages, such as C++ and ANSI C, the signature of a method is a declaration comprising in order: any optional accessibility keywords (such as `private`), the explicit specification of its return type (such as `int`, or the keyword `void` if no value is returned), the name of the method, and finally, a parenthesized sequence of comma-separated parameter specifications, each consisting of a parameter's type, its formal name and optionally, a default value to be used whenever none is provided. Different from most other languages, call-by-reference parameters have to be marked both at the function definition and at the calling site, and you can choose between `ref` and `out`, the latter allowing handing over an uninitialized variable which will have a definite value on return.[77] Additionally, you can specify a variable-sized argument list by applying the `params` keyword to the last parameter.[78] Certain specific kinds of methods, such as those that simply get or set a field's value by returning or assigning it, do not require an explicitly stated full signature, but in the general case, the definition of a class includes the full signature declaration of its methods.[79]

Like C++, and unlike Java, C# programmers must use the scope modifier keyword `virtual` to allow methods to be overridden by subclasses. Unlike C++, you have to explicitly specify the keyword `override` when doing so.[80] This is supposed to avoid confusion between overriding and newly overloading a function (i.e. hiding the former implementation). To do the latter, you have to specify the `new` keyword.[81]

Extension methods in C# allow programmers to use static methods as if they were methods from a class's method table, allowing programmers to virtually add instance methods to a class that they feel should exist on that kind of objects (and instances of the respective derived classes).[16]:103–105[20]:202–203

The type `dynamic` allows for run-time method binding, allowing for JavaScript-like method calls and run-time object composition.[16]:114–118

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# supports classes with properties. The properties can be simple accessor functions with a backing field, or implement arbitrary getter and setter functions. A property is read-only if there's no setter. Like with fields, there can be class and instance properties. The underlying methods can be `virtual` or `abstract` like any other method.[79]

Since C# 3.0 the syntactic sugar of auto-implemented properties is available,[82] where the accessor (getter) and mutator (setter) encapsulate operations on a single attribute of a class.

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`. Namespaces can be imported with the "using" syntax.[83]

Memory access

In C#, memory address pointers can only be used within blocks specifically marked as *unsafe*,[84] 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 an unmanaged value type that does not contain any references to objects subject to garbage collections such as class instances, arrays or strings. 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 in most cases. Code that retains references to objects longer than is required can still experience higher memory usage than necessary, however once the final reference to an object is released the memory is available for garbage collection.

Exceptions

A range of standard exceptions are available to programmers. Methods in standard libraries regularly throw system exceptions in some circumstances and the range of exceptions thrown is

normally documented. Custom exception classes can be defined for classes allowing handling to be put in place for particular circumstances as needed.[85]

The syntax for handling exceptions is `try {` *something* `} catch (Exception ex) {` *handle* `ex } finally {` *do something whether or not an exception occurred* `}`. Depending on your plans, the "catch" or the "finally" part can be left out, and there can be several "catch" parts handling different kinds of exceptions. [86]

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.[87]

Polymorphism

Unlike C++, C# does not support multiple inheritance, although a class can implement any number of "interfaces" (fully abstract classes). This was a design decision by the language's lead architect to avoid complications and to simplify architectural requirements throughout CLI.

When implementing multiple interfaces that contain a method with the same name and taking parameters of the same type in the same order (i.e. the same signature), similar to Java, C# allows both a single method to cover all interfaces and if necessary specific methods for each interface.

However, unlike Java, C# supports operator overloading.[88]

C# also offers function overloading (a.k.a. ad-hoc-polymorphism).[89]

Since version 2.0, C# offers parametric polymorphism, i.e. classes with arbitrary or constrained type parameters, e.g. `List<T>`, a variable-sized array which only can contain elements of type `T`. There are certain kinds of constraints you can specify for the type parameters: Has to be type X (or one derived from it), has to implement a certain interface, has to be a reference type, has to be a value type, has to implement a public parameterless constructor. Most of them can be combined, and you can specify any number of interfaces.[90][91]

Language Integrated Query (LINQ)

C# has the ability to utilize LINQ through the .NET Framework. A developer can query a variety of data sources, provided the `IEnumerable<T>` interface is implemented on the object. This includes XML documents, an ADO.NET dataset, and SQL databases.[92]

+ Using LINQ in C# brings advantages like IntelliSense support, strong filtering capabilities, type safety with compile error checking ability, and consistency for querying data over a variety of sources.[93] There are several different language structures that can be utilized with C# and LINQ and they are query expressions, lambda expressions, anonymous types, implicitly typed variables, extension methods, and object initializers.[94]

LINQ has two syntaxes: query syntax and method syntax. However, the compiler always converts the query syntax to method syntax at compile time.[95]

```
using System.Linq;

var numbers = new int[] { 5, 10, 8, 3, 6, 12 };
```

```
// Query syntax (SELECT num FROM numbers WHERE num % 2 = 0 ORDER BY num)
var numQuery1 =
        from num in numbers
        where num % 2 == 0
        orderby num
        select num;

// Method syntax
var numQuery2 =
        numbers
        .Where(num => num % 2 == 0)
        .OrderBy(n => n);
```

Functional programming

]

Though primarily an imperative language, C# always adds functional features over time,[96][97] for example:

- Functions as first-class citizen – C# 1.0 delegates[98]
- Higher-order functions – C# 1.0 together with delegates
- Anonymous functions – C# 2 anonymous delegates and C# 3 lambdas expressions[99]
- Closures – C# 2 together with anonymous delegates and C# 3 together with lambdas expressions[99]
- Type inference – C# 3 with implicitly typed local variables `var` and C# 9 target-typed new expressions `new()`
- List comprehension – C# 3 LINQ
- Tuples – .NET Framework 4.0 but it becomes popular when C# 7.0 introduced a new tuple type with language support[100]
- Nested functions – C# 7.0[100]
- Pattern matching – C# 7.0[100]
- Immutability – C# 7.2 readonly struct C# 9 record types[101] and Init only setters[102]
- Type classes – C# 12 roles/extensions (in development[103])

Common type system

C# has a *unified type system*. This unified type system is called Common Type System (CTS).[104]:Part 2, Chapter 4: The Type System

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:[104]

1. Reference types
2. Value types

Instances of value types neither 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 because they already have an implicit one which initializes all contained data to the type-dependent default value (0, null, or alike). 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), `decimal` (fixed-point numbers useful for handling currency amounts), 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, meaning that 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`). Some operations are not always possible, such as creating an instance of a reference type, copying an existing instance, or performing a value comparison on two existing instances. Nevertheless, 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.[104] 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.[104] 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).[105]

Example:

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

Libraries

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)*.

In addition to the standard CLI specifications, there are many commercial and community class libraries that build on top of the .NET framework libraries to provide additional functionality.[106]

C# can make calls to any library included in the List of .NET libraries and frameworks.

Examples

Hello World

is a very simple C# program, a version of the classic "Hello world" example using the top-level statements feature introduced in C# 9:[107]

```
using System;

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

For code written as C# 8 or lower, the entry point logic of a program must be written in a Main method inside a type:

```
using System;

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

This code will display this text in the console window:

```
Hello, world!
```

Each line has a purpose:

```
using System;
```

The above line imports all types in the `System` namespace. For example, the `Console` class used later in the source code is defined in the `System` namespace, meaning it can be used without supplying the full name of the type (which includes the namespace).

```
// A version of the classic "Hello World" program
```

This line is a comment; it describes and documents the code for the programmer(s).

```
class Program
```

Above is a class definition for the `Program` class. Everything that follows between the pair of braces describes that class.

```
{
    ...
}
```

The curly brackets demarcate the boundaries of a code block. In this first instance, they are marking the start and end of the `Program` class.

```
static void Main()
```

This declares the class member method where the program begins execution. The .NET runtime calls the `Main` method. Unlike in Java, the `Main` method does not need the `public` keyword, which tells the compiler that the method can be called from anywhere by

any class.[108] Writing `static void Main(string[] args)` is equivalent to writing `private static void Main(string[] args)`. 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 of `Program`, 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. (Note, however, that short programs can be written using Top Level Statements introduced in C# 9, as mentioned earlier.)

```
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!"`.

Generics

With .NET 2.0 and C# 2.0, the community got more flexible collections than those in .NET 1.x. In the absence of generics, developers had to use collections such as ArrayList to store elements as objects of unspecified kind, which incurred performance overhead when boxing/unboxing/type-checking the contained items.

Generics introduced a massive new feature in .NET that allowed developers to create type-safe data structures. This shift is particularly important in the context of converting legacy systems, where updating to generics can significantly enhance performance and maintainability by replacing outdated data structures with more efficient, type-safe alternatives.[109]

Example

```
public class DataStore<T>
{
    private T[] items = new T[10];
    private int count = 0;

    public void Add(T item)
    {
        items[count++] = item;
    }

    public T Get(int index)
    {
        return items[index];
    }
}
```

Standardization and licensing

In August 2001, Microsoft, Hewlett-Packard and Intel 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/IEC 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/SC 22, 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/IEC and Ecma standards that provide reasonable and non-discriminatory licensing protection from patent claims.

Microsoft initially 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 Open Specification Promise.[110] Microsoft has also agreed not to enforce patents relating to Novell products against Novell's paying customers[111] with the exception of a list of products that do not explicitly mention C#, .NET or Novell's implementation of .NET (The Mono Project).[112] However, Novell maintained that Mono does not infringe any Microsoft patents.[113] Microsoft 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.[114]

A decade later, Microsoft began developing free, open-source, and cross-platform tooling for C#, namely Visual Studio Code, .NET Core, and Roslyn. Mono joined Microsoft as a project of Xamarin, a Microsoft subsidiary.

Implementations

Microsoft is leading the development of the open-source reference C# compilers and set of tools. The first compiler, Roslyn, compiles into intermediate language (IL), and the second one, RyuJIT,[115] is a JIT (just-in-time) compiler, which is dynamic and does on-the-fly optimization and compiles the IL into native code for the front-end of the CPU.[116] RyuJIT is open source and written in C++.[117] Roslyn 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.[4][118] Two branches of official implementation are .NET Framework (closed-source, Windows-only) and .NET Core (open-source, cross-platform); they eventually converged into one open-source implementation: .NET 5.0.[119] At .NET Framework 4.6, a new JIT compiler replaced the former.[115][120]

Other C# compilers (some of which include an implementation of the Common Language Infrastructure and .NET class libraries):

- Mono, a Microsoft-sponsored project provides an open-source C# compiler, a complete open-source implementation of the CLI (including the required framework libraries as they appear in the ECMA specification,) and a nearly complete implementation of the NET class libraries up to .NET Framework 3.5.
- The Elements tool chain from RemObjects includes RemObjects C#, which compiles C# code to .NET's Common Intermediate Language, Java bytecode, Cocoa, Android bytecode, WebAssembly, and native machine code for Windows, macOS, and Linux.
- 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).

The Unity game engine uses C# as its primary scripting language. The Godot game engine has implemented an optional C# module thanks to a donation of $24,000 from Microsoft.[121]