C#

The Never Ending Language

By: Lily Zimmermann

Lakeland University - Fall 2022

CPS314.K1 - Programming Languages

C# seemed to be the best language to tackle in this paper because of the massive amount of programming job postings I have seen that included the language as either a requirement or an advantage for applicants. Many college courses are taught in either Java, Python, C, C++, or SQL so while C# is a massively popular language, there are many students that may not initially encounter it until taking on their own projects, starting an internship, or after taking higher level courses. While C# does have its disadvantages, it is a very fun language to explore. Any programmer that has already worked in an object oriented programming language should not have trouble getting started in it, but once you get started in the language, you will find yourself on a never-ending journey through what the umbrella of C# covers. For this reason, there is no way I could cover everything included in the language in only a ten page paper, but I am hoping to explain some of the most important highlights.

**C#’S HISTORY:**

As with any other paper, starting with the history of the language and its inception seems to be the best place to start. “C# (sic) was designed by Anders Hejlsberg from Microsoft in 2000 and was later approved as an international standard by Ecma (ECMA-334) in 2002 and ISO/IEC (ISO/IEC 23270) in 2003” (“C Sharp (programming language)”, 2022). The language itself is not that different from other basic object-oriented programming languages, but since it was released with the .NET Framework and Visual Studio, the language has been able to cover functionality so broad that many other programming languages have not been able to compete. The language has evolved primarily to serve the needs of Microsoft, but the changes have also allowed it to continue to thrive externally among other companies. The C# project had a full open-source avenue when Visual Studio Code (an IDE released in April 2015) (“Visual Studio Code”, 2022), Roslyn (a compiler released in October 2011) (“Roslyn (compiler)”, 2022), and the unified .NET platform (a software framework released in June 2016) (“.NET”, 2022) became available, keeping in mind that their initial releases did not necessarily start as open-source. (“C Sharp (programming language”, 2022)

According to Hejlsberg, the principal designer, the language was created because of the flaws he saw in most major programming languages like C++ and Java (“C Sharp (programming language”, 2022). Strangely, Wikipedia does not mention anywhere in their article on the language that the birth of C# involved a lawsuit, and many say that it is the reason why the language was created. A humorous video on YouTube titled, “Interview with a Senior C# Developer in 2022” published on the channel ‘Programmers are also human’, starts out the video with the words “Microsoft Java - I mean C#”. That beginning phrase seems to encompass the casual conversations around the language that many official articles do not want to approach. The lawsuit began in 1997 when Sun Microsystems, the creator of the Java programming language, sued Microsoft for $35 million due to breach of contract. Sun stated that Microsoft advertised, incorrectly, that its products were Java-compatible even though this was only possible after Microsoft changed Java to the point of it no longer being a universal language. The suit was settled three years later with Microsoft paying Sun $20 million and no longer being able to state that their products were Java compatible, even though the C# language would nearly duplicate the Java language in most ways. While Microsoft stated that the lawsuit would still result in positive outcomes for their company, they chastised Sun for limiting growth, change, and future evolution of their products. C#’s constant evolution and change proved that the company’s public statement about the lawsuit came from a core belief in their operations (Shankland, 2022). The ever expanding nature of C# has resulted in its greatest weaknesses and greatest strengths which birthed the title of this paper, ‘the never ending language’.

To understand C#’s changing nature, we first need to understand .NET. Trying to understand .NET is just like trying to learn C#. It is somehow very simple and very complicated at the same time. .NET actually can be split into two basic sub-types (if you are excluding the mobile application world of programming). When reading articles, “.NET” can either refer to the computer software framework that is specific to Microsoft, or to the open-source framework that was designed for Windows, Linux, and macOS in an attempt to work towards cross-platform functionality. This makes researching the topic very confusing because people often do not distinguish which branch they are talking about. Wikipedia distinguishes the two by calling the original framework specific to Microsoft, the “.NET framework”, and by calling the open-source version “.NET”. In an effort for clarity, I will be using the Wikipedia verbiage to distinguish the two when both are brought up, and the term “.NET” as an umbrella term for both when discussing the basic functionality of the framework with the C# language. The first version, the Microsoft specific framework, was started in the 1990’s and had its initial beta release in early 2000 (“.NET Framework”, 2022). The open-source version had its initial release in June 2016. It started with the name .NET Core 1.0, and had its base name changed from .NET Core to .NET in its November 2020 release (“.NET”, 2022). As of April 2019, Microsoft released the last version of the original proprietary framework with the intention of just focusing on cross-platform functionality outside of basic security and bug updates for the final founding platform release (“.NET Framework”, 2022). Only about 7 months ago, in April 2022, Microsoft released .NET MAUI to incorporate Android and IOS application support in the .NET world (“.NET”, 2022).

In terms of the most basic programming explanation with respect to the actual application of .NET, the framework can be summed up to pre-built code that C# works with to allow the core language to do much more than it would be able to do without .NET. Any programmer that has downloaded a popular code language to their computer should know that there are usually different libraries, classes, etc. that are downloaded in addition to the language itself. Sometimes, developers need to work with additional frameworks in order to access the functionality needed for what they are trying to accomplish with their code. From a basic programming standpoint, this is where .NET works with C#. Like with Java, there is a virtual machine involved in the development process, but C# is unique in that the virtual machine, Common Language Runtime (CLR), is actually a part of the .NET Framework. I have seen some developers say that CLR is synonymous with Java’s JVM (Java Virtual Machine), and others state that C# is unique in that it is not the language itself that works with CLR, but rather .NET that works with it. To understand the CLR, we will need to first review the code’s journey from source code to native code.

C# starts as source code. This is the actual code that the developer writes or the C# language itself. The source code goes through a C# compiler and becomes intermediate/byte code that is called Common Intermediate Code (CIL). CIL was previously called Microsoft Intermediate Language (MSIL) and is sometimes referred to as Intermediate Language Code (IL Code). The intermediate code goes through the CLR virtual machine and becomes machine code as a .exe or .dll file. It is important to note that the CIL can run on any operating system since C# is a platform independent language (Argha\_c14, 2022). That process includes just-in-time compilation where the code is compiled at execution time. It is possible to instead use ahead-of-time compilation in the pursuit of faster execution, but that would mean that the executable file is no longer platform independent, or portable (“Common Intermediate Language”, 2022). This process is why there is some debate online as to whether C# is just considered a compiled language. The full answer is more nuanced with C#, but Microsoft supported a post on their Q&A section that states that while the true answer requires some explanation, C# is ultimately a compiled language regardless of a virtual machine being used in the process (virtual machines being one of the most common kind of interpreters) (SimpleSamples, 2021).

The CLR promotes more advantages than just making the cross-platform functionality possible. When directly quoting Microsoft on their official C# education website, the company states that the primary benefits of the virtual machine include:

* Performance improvements.
* The ability to easily use components developed in other languages.
* Extensible types provided by a class library.
* Language features such as inheritance, interfaces, and overloading for object-oriented programming.
* Support for explicit free threading that allows creation of multithreaded and scalable applications.
* Support for structured exception handling.
* Support for custom attributes.
* Garbage collection.
* Use of delegates instead of function pointers for increased type safety and security. For more information about delegates, see Common Type System

( “Common Language Runtime (CLR) overview”, 2022).

This paper has now covered the history of C#, .NET, and the CLR. Four and a half pages in, and I have not even covered the primitive data types or structure of a hello world program. This seems to be the nature of C#. While the fundamental language itself is easy to approach, there is so much to understand and so many tangents of information that a developer can go down. I have created a supplemental excel spreadsheet to give better coverage on the basic syntax and mechanics of the language. Although the next sections will touch on those topics, charts typically seem to be an easier way to get an overview of the language itself in comparison to a paragraph or paper.

**HELLO WORLD PROGRAM:**

|  |
| --- |
| //Source: <https://zetcode.com/csharp/basics/>  namespace Simple;  class Program  {  static void Main(string[] args)  {  Console.WriteLine("Hello World!");  }  } |

**Hello World Program Breakdown:**

1. Line: namespace Simple;
   1. At the beginning of your first C# program, you need to declare a namespace. Namespaces are used to organize code. There can be classes, interfaces, structures, delegates, and more namespaces (nested namespaces) within a namespace (“Namespaces in C# Programming”, Unknown).
2. Line: class Program
   1. A class is a signature feature of an object oriented programming language. In C#, a class is a reference type used to define an object even though it is not an object itself. In C#, classes support inheritance, but a class can only directly inherit from one base class. Since a base class can inherit from another base class, there is not really a limit to inheritance if the program is approached with the C# inheritance constraints kept in mind (“Introduction to classes”, 2021).
3. Line: static void Main(string[]args)
   1. Although the entire hello world program will give a developer Java dejavu, this line especially screams Java. It declares the main method in a way that does not require a developer to create an instance of the Program class because of using the static keyword, and that notes that no value will be returned or output from the class, because of using the void keyword.
   2. We need to use void with the main method because methods are essentially functions. Going back to algebra and pre-calculus, functions are typically things that have an input and an output. In programming, functions are blocks of code that can have an input and/or an output but do not always include that functionality (depending on the language). In this case, the main method will not output any data, so we include the void keyword to reinforce that.
4. Line: Console.WriteLine("Hello World!");
   1. This line of code is located within the body of the program and is the line that contains the instructions of what will be printed or written to the console.
   2. “Hello World!” is a string which is a data type that can include letters, numbers or characters like exclamation points or question marks.

**C# DATA TYPES:**

Discussing C# data types can get complicated like many other facets of the language because of how much the language has been expanded. This paper will be highlighting the built-in types that Microsoft includes on their C# guide, but there are many other data types that can be used. All the information covered in this section that is not common knowledge among programming languages can be found on <https://learn.microsoft.com/en-us/dotnet/csharp/language-reference/builtin-types/built-in-types.>

The main categories of data types are integral, floating point, and reference. Microsoft does not list a category that its character (char) and boolean (bool) data types fall under, so many articles just list their respective categories as the names of the data types themselves (boolean and char).

The built-in integral category includes 10 different key words. The four main versions of integral data types are byte, short, int, and long. Each one has a signed and unsigned version. A signed data type is one that includes a range of values that are negative and positive. An unsigned data type is one that does not include negative numbers (only includes zero and positive numbers). The integral data types were created to house different ranges of integers. A larger range will include more numbers and will take up more space.

The built-in floating-point category was created to house numbers with decimals. The category includes three keywords: float, double, and decimal. All three types include negative and positive values. Each type has a different range and takes up a different amount of memory with float taking up the least amount of memory and decimal taking up the largest. Double seems to be the most common floating-point data type used.

The built-in reference category is the only one that includes data types with very different types of values that are housed, but all three types fall under the reference category because they reference something when created. The three data types within the category are object, string, and dynamic. Object is the data type that all data types inherit from in C#. You can assign values of any type to variables of type object. When a variable’s value is type ‘object’, the variable is boxed. When a variable’s value is converted to a value type, the variable is then unboxed. Next there is the string type. Strings are variables that can contain one or more characters that can be letters, numbers, and most common symbols like question marks or pound signs. Lastly, the dynamic type is one that allows the variable and references to its members to bypass compile-time checking.

I already touched on boolean and char, but they are the two data types that don’t quite fit with other data types. The bool keyword is used for boolean logic (true or false) and the char keyword is used for variables that contain a single character.

All of the main data type keywords in C# can be found in other languages with some small variations in verbiage, like in the case of decimal, but most other common languages do not seem to include as many integral types within their built-in data types. One interesting feature of the data types in C# is that every common keyword actually has a corresponding .NET type. For example, the C# keyword float’s .NET type is System.Single. The C# keyword int’s .NET type is System.Int32. The keyword and .NET type can be used interchangeably, but the .NET type does require some extra code, so most people just use the C# keywords.

**C# COLLECTIONS:**

C# includes many collection types. Its collections fall under three main categories or namespaces: System.Collections, System.Collections.Concurrent, and System.Collections.Generic. System.Collections is an old version of collections that should be avoided when possible. System.Collections.Concurrent includes classes of collections that can be used when a programmer needs his program to be able to tackle concurrency issues like when multiple threads are involved. System.Collections.Generic includes the most commonly used classes of collections in C# and probably includes the only collections that new programmers would use (“Collections (C#)”, 2022). That namespace will be referred to as the generic namespace or generic collection namespace for the rest of this paper.

The five most frequently used classes in the generic namespace are dictionary, list, queue, sorted list, and stack (Dictionary<TKey,TValue>, List<T>, Queue<T>, SortedList<TKey,TValue>, and Stack<T>). The official names of those classes include capital T’s to represent the data type of the values each collection will store. The queue and stack classes function in opposite ways. They represent a structure that holds a collection of objects, and each includes a feature of how values will enter and exit the collection. The queue class takes a first in, first out approach while the stack class takes a last in, first out approach (“System.Collections.Generic Namespace”, Unknown).

The dictionary and sorted list classes both represent collections of key/value pairs, but dictionary has no organization from one key value pair to another while sorted list is sorted by key based on the associated IComparer<T> implementation (“System.Collections.Generic Namespace”, Unknown). IComparer<T> is just a part of the code that allows the developer to define how values will be sorted. It is an interface that works with the List<T>.Sort and List<T>.BinarySearch methods (“ IComparer<T> Interface”, Unknown).

**C# OVERVIEW:**

There are many statements about C# that can be accurate and inaccurate at the same time, which largely can be attributed to the fact of how many different versions of C# and especially how many different versions of .NET there are. Reviewing the basic highlights of the language is no exception. Most overarching topics, like C#’s programming paradigm, have an answer, but also include a ‘but’.

For starters, C# at its core is an object oriented programming language. I do not think any expansion will change this fact. The very first name of the language was Cool which literally stood for “C-like Object Oriented Language”. All of the changes over the years have allowed the language to cover multiple other programming paradigms including structured, imperative, event-driven, task-driven, functional, generic, reflective, and concurrent programming (“C Sharp (programming language)”, 2022). I cannot find a definitive timeline of when each programming paradigm was incorporated into the language, but when reviewing more recent changes, it seems like Microsoft’s most recent focus has been support for functional programming.

The language’s ease of use is another topic that does not have a clear yes or no answer. The core code itself is not very difficult to understand, especially if programmers new to the language have used Java in the past. C#’s core programming paradigm, object oriented programming, is nothing new in 2022, so the features of the language that support that style of programming should not trip up most developers. It is true that C# relies more heavily on syntax than other coding languages like Python, but that syntax can make the code much easier to read (especially when looking at braces that highlight chunks of code). Despite the fact that C# relies more on syntax, one of its advantages is the fact that it does promote developer efficiency in comparison to its strongly typed counterparts like C and C++.

When developers have stated that C# was difficult to learn, they usually were referring to how many different versions of the language that there were. A great video that highlights this issue is one on YouTube where a C# developer, Nick Chapsas, discusses some of the changes that C# has gone through over the years. When reviewing C# version 8.0, he pointed out that many C# developers avoided using the nullable reference types because they had become so accustomed to working around the issue that the change addressed. He went on to explain that it is changes like that that made C# developers code in drastically different ways, depending on when they started working with the language (Chapsas, 2021). This could be a huge issue and really goes back to the lawsuit from C#’s inception. Sun Microsystems wanted to keep a certain consistency in Java while Microsoft wanted to expand. This expansion approach that Microsoft used with C# meant that the language could do many things, but it also meant that one company’s use of the language could look drastically different from another. As Chapsas highlighted in his YouTube video referenced above, this lack of consistency can even exist among developers within the same company. As a final point on ease of use, I would say that yes C# is mostly easy to use, there is just a lot of information that can be covered when approaching the language.

To sum up C#’s functionality and nature as a programming language, it is mostly object oriented, but also supports other programming paradigms. It is easy to use, but also includes a lot of different versions and a massive amount of information that takes a lot of time to cover. It is a strongly and statically typed language, but there are features that allow implicit and explicit variable declaration. The “var” keyword can be used to force the compiler to determine the variable type at runtime, but this is not the default of the language and would make the program more error prone, so it does not seem like the default for developers. In general C# does not support duck typing, but there are some parts of .NET that do use duck typing (Arvindpdmn, Devbot5S, & GurumoorthyP, 2022). C# does include syntactic sugar, but this is another part of the language that is ever changing. One version that seemed to really emphasize syntactic sugar was C# version 6.0 when features like auto-property initializers, expression bodied members, and string interpolation were introduced (Chapsas, 2021). Lastly, there are many programming concepts that are either part of C#’s core concept or have been made possible through updates to the language and/or to .NET including encapsulation, inheritance, polymorphism, concurrency, pattern matching, and recursion. Like stated earlier, concurrent programming may not have been a part of C#’s conception, but the more recent focus towards functional programming support is making it possible despite the fact that C# is primarily an object oriented programming language.

**WHERE IS C# USED?:**

C# is such a broad multi-purpose language, that it is safe to say that it could be used in most scenarios where programming is needed. That does not, however, mean it is the best language to use in most cases. There seem to be three general scenarios where C# is strongest. Those scenarios would be Windows applications, web application development, and games. Considering that the language was initially just developed for Microsoft and didn’t start moving toward the open-source world until later, it is no surprise that C# is used for Windows applications. The .NET and open-source platforms have made C# popular for web application development because of its portable nature. Lastly, many programmers prefer C# for game development because of its use in game engines like Unity (Watson, 2020).

**C# ADVANTAGES & DISADVANTAGES:**

C#’s broad functionality seems to also allow the language to have many more advantages than disadvantages. This should be no surprise considering that it is ranked as the 5th most popular language according to the TIOBE index (“TIOBE Index for November 2022”, 2022).

**Advantages:**

1. Garbage Collection
   1. C#’s garbage collection was one of its biggest features when it was first released and continues to be a strong feature of the language
2. Promotes efficiency & fast development time through its reusable components, syntactic sugar, and more developer-friendly changes from the C and C++ languages
3. Is a strong language for companies concerned about scalability
4. The code syntax looks similar to Java and C++ so it has been easy for a lot of developers to pick up
5. Data types inside C# are more flexible and error-free.
6. There are many resources available from Microsoft, but also from the C# community because of the languages popularity
7. The language has been expanded so much that it seems like there are not a lot of limits for what the language can do
   1. Other languages may be able to execute certain tasks better or more efficiency, but few cover the type of broad range that C# does

**Disadvantages:**

1. There are some languages that perform a lot better like C++
2. C# is only flexible because of .NET, not the language by itself
3. There is a massive amount of information included because of the different versions of C# and .NET

(Deboah, 2019) & (“The Good and the Bad of C# Programming”, 2021)

In conclusion, while C# is the never ending language, it is also a fun language to work with and I would suggest the language to any developer. Microsoft’s changes may have created a massive amount of information to look through, but it has also allowed the language to support most forms of programming that a current developer may need to use. In addition to that, C#’s popularity has given way to many helpful resources online including a very well organized and in depth guide that Microsoft has worked on perfecting in recent years. Although C# is not a strictly functional programming language, the most recent programming paradigm that is gaining popularity, I do not see the language going anywhere any time soon, especially since Microsoft’s main mission at the language’s creation was innovation, adaptation, and change which Microsoft has continued to stand by with this language.

**BIBLIOGRAPHY**

Argha\_c14. (2021, August 31). How C# Code Gets Compiled and Executed?. Geeks for Geeks. <https://www.geeksforgeeks.org/how-c-sharp-code-gets-compiled-and-executed/>

Arvindpdmn, Devbot5S, & GurumoorthyP. (2022, February 15). Duck Typing. Devopedia. <https://devopedia.org/duck-typing>

Chapsas, Nick. (2021, July 27). Where C# was, where it’s going and why it’s very exciting [video]. Youtube. <https://www.youtube.com/watch?v=_hTzTnuq7lg&ab_channel=NickChapsas>

Deborah (2019, May 13). C Sharp - Features, Advantages and Disadvantages. Urban Naturale. <https://urbannaturale.com/c-sharp-features-advantages-and-disadvantages/>

Microsoft contributors. (2022, September 29). Collections (C#). Microsoft. <https://learn.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/collections#BKMK_KindsOfCollections>

Microsoft contributors. (2022, September 8). Common Language Runtime (CLR) overview. Microsoft. <https://learn.microsoft.com/en-us/dotnet/standard/clr>

Microsoft contributors. (Unknown). IComparer<T> Interface. Microsoft. <https://learn.microsoft.com/en-us/dotnet/api/system.collections.generic.icomparer-1?view=net-7.0>

Microsoft contributors. (2021, September 9). Introduction to classes. Microsoft. <https://learn.microsoft.com/en-us/dotnet/csharp/fundamentals/types/classes>

Microsoft contributors. (Unknown). System.Collections.Generic Namespace. Microsoft. <https://learn.microsoft.com/en-us/dotnet/api/system.collections.generic?view=net-7.0>

Monaeghan, Jackson. [Programmers are also human]. (2022, March 21). Interview with a Senior C# Developer in 2022 [Video]. Youtube. <https://www.youtube.com/watch?v=bXzTXD_OJo0&ab_channel=Programmersarealsohuman>

Programiz contributors. (Unknown). Namespaces in C# Programming. Programiz. <https://www.programiz.com/csharp-programming/namespaces>

Shankland, Stephen. (2022, March 15). Sun, Microsoft settle Java suit. CNET <https://www.cnet.com/tech/mobile/apple-reveals-black-friday-deals-for-2022/>

SimpleSamples. (2021, August 24). Is C# an interpreted language? Q&A Microsoft Learn. <https://learn.microsoft.com/en-us/answers/questions/525326/is-c-an-interpreted-language.html>

TIOBE Contributors. (2022, November). TIOBE Index for November 2022. TIOBE. <https://www.tiobe.com/tiobe-index/>

Unknown. (2021, October 29). The Good and the Bad of C# Programming. Altexsoft. <https://www.altexsoft.com/blog/c-sharp-pros-and-cons/>

Watson, Matt. (2020, September 18). What is C# used for?. Stackify. <https://stackify.com/what-is-c-used-for/>

Wikipedia contributors. (2022, November 17). C Sharp (programming language). In Wikipedia, The Free Encyclopedia. Retrieved 16:40, November 19, 2022, from <https://en.wikipedia.org/w/index.php?title=C_Sharp_(programming_language)&oldid=1122497707>

Wikipedia contributors. (2022, May 1). Common Intermediate Language. In Wikipedia, The Free Encyclopedia. Retrieved 20:25, November 19, 2022, from <https://en.wikipedia.org/w/index.php?title=Common_Intermediate_Language&oldid=1085648650>

Wikipedia contributors. (2022, November 18). Just-in-time compilation. In Wikipedia, The Free Encyclopedia. Retrieved 20:24, November 19, 2022, from <https://en.wikipedia.org/w/index.php?title=Just-in-time_compilation&oldid=1122526156>

Wikipedia contributors. (2022, October 24). .NET Framework. In Wikipedia, The Free Encyclopedia. Retrieved 18:01, November 19, 2022, from <https://en.wikipedia.org/w/index.php?title=.NET_Framework&oldid=1118034235>

Wikipedia contributors. (2022, November 15). .NET. In Wikipedia, The Free Encyclopedia. Retrieved 17:44, November 19, 2022, from <https://en.wikipedia.org/w/index.php?title=.NET&oldid=1122051993>

Wikipedia contributors. (2022, November 16). Roslyn (compiler). In Wikipedia, The Free Encyclopedia. Retrieved 17:43, November 19, 2022, from <https://en.wikipedia.org/w/index.php?title=Roslyn_(compiler)&oldid=1122228111>

Wikipedia contributors. (2022, November 16). Visual Studio Code. In Wikipedia, The Free Encyclopedia. Retrieved 16:51, November 19, 2022, from <https://en.wikipedia.org/w/index.php?title=Visual_Studio_Code&oldid=1122274311>

