Evaluation of C# Programming Language

Cameron Goldberg

SUNY Brockport, NY 14420, USA
cgold6@brockport.edu

Abstract:

C# is a programming language that is owned by Microsoft, and it was released around the early 2000s. It is an imperative, object-oriented language that is popular among application and web development fields. It runs on the .NET framework, and it is cross-platform. This language is currently at version 10.0.

1 History

In the late 1990s and early 2000s a group of professionals worked together to create an object-oriented language that was similar to C. This group was led by Anders Hejlsberg, and the language was initially called COOL. Due to trademark reasons, this language became known as C#. At around the same time, the .NET framework was beginning to grow and the class libraries were created using Simple Managed C (SMC) managed code compiler. By the time C# had taken a more established form and permanent name, the code for ASP.NET runtime was written in C#. Upon the initial receptions of C#, "It didn't come as a shock to anyone when the creator of the Java language, James Gosling, insisted that C# was a copy of Java with security, reliability and productivity removed. Anders Hejlsberg retorted with that the C# design resembles C++ more than Java" [6]. The design of C# did have similarities to Java, but it was certainly not a copy. "When C# 2.0 was released in 2005 the two languages went down two very different paths" [6].

2 Type of Language:

C# is an imperative programming language that is popular within an large gamut of applications for the use of enterprise that run on the .NET framework. "C# is simple, modern, type safe, and object- oriented" [4]. It is noted that "C# code is compiled as managed code" [4]. An imperative language is one that follows the Von Neumann structure, and an object-oriented language is one that allows the code to be broken up into separate objects or classes that contain attributes and methods. Being a type safe language means that it accounts for and prevents type errors that result from having different data types for the variables, constants, and methods within a program. Since C# is compiled as managed code "it benefits from the services of the common language runtime. These services include language interoperability, garbage collection, enhanced security, and improved versioning support" [4]. C# is a strong typed language, too. "A programming language is strongly typed if type errors are always detected. This requires that the types of all operands can be determined, either at compile time or at run time" [5].

3 Application Area:

C# is a "programming language designed for building a wide range of enterprise applications that run on the .NET Framework" [4] The .NET framework provides cross-platform development for applications. This includes "Windows, macOS, Linux, Android, iOS, tvOS, watchOS" [2]. The .NET platform is also open source and provides IDEs, among other tools, that allow developers to create applications that run on almost any device, and more particularly the operating systems that were already mentioned. Since C# has many similarities to Java, C, and C++ it has become popular since it is easily adopted by programmers who are familiar with those languages. Given its cross compatible nature between other operating systems, it has also become more prevalent among application developers. "C# is widely used for developing desktop applications, web applications and web services." [3]. One specific area of C#'s application is in game development. This is because "C# integrates with Microsoft and thus has a large target audience." [3]. C#'s features of garbage collection, interfaces, and its object-oriented nature make it particularly useful for those who want to make desktop and mobile applications for consumers as well.

4 Type Structures:

C# has many types with a variety of ways to declare variables of these types. For value types there are integral numeric types, floating-point numeric types, bool, char, enumeration types, structure types, tuple types, and nullable value types. The following table displays the available integral numeric types [1]:

C# type/keyword	Range	Size	.NET type
`sbyte`	-128 to 127	Signed 8-bit integer	System.Sbyte
`byte`	0 to 255	Unsigned 8-bit integer	System.Byte
`short`	-32,768 to 32,767	Signed 16-bit integer	System.Int16
`ushort`	0 to 65,535	Unsigned 16-bit integer	System.Uint16
`int`	-2,147,483,648 to 2,147,483,647	Signed 32-bit integer	System.Int32
`uint`	0 to 4,294,967,295	Unsigned 32-bit integer	System.Uint32
`long`	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807	Signed 64-bit integer	System.Int64
`ulong`	0 to 18,446,744,073,709,551,615	Unsigned 64-bit integer	System.Uint64
`nint`	Depends on platform	Signed 32-bit or 64-bit integer	System.IntPtr

`nuint`	Depends on platform	Unsigned 32-bit or 64-bit integer	System.UintPtr

All of the types in the table except for the last two are declared in the same way. The syntax for declaring a variable and its value is: type variableName = value. This can be seen in the following two examples.

```
int myVariable1 = 10;
System.Int32 myVariable2= 10;
```

The nint and nuint type means it is native-sized, so it will be 32-bit or 64-bit integer for a respective 32-bit or 64-bit process. These can be seen in the following two examples:

```
nint myVariable1 = 10;
System.IntPtr myVariable2 = 10;
```

The following table displays the available floating-point numeric types [1]:

C# type/keyword	Approximate range	Precision	Size	.NET type
`float`	±1.5 x 10−45 to ±3.4 x 1038	~6-9 digits	4 bytes	System.Single
`double`	±5.0 × 10−324 to ±1.7 × 10308	~15-17 digits	8 bytes	System.Double
`decimal`	±1.0 x 10-28 to ±7.9228 x 1028	28-29 digits	16 bytes	System.Decimal

The syntax to declare these types can be seen in the following two examples:

```
double myVariable1 = 10.1;
System.Double myVariable2 = 10.1;
```

The bool type is a Boolean that can be used to perform logical operators. A variable of type bool can be either true or false. The following example shows how to declare a bool variable and its value:

```
bool myVariable1 = true;
```

C# also supports char types which can represent a character. These types can be declared in the following manner:

```
char myVariable1 = 'A';
```

There are enumeration types in C# that allow a programmer to define a set of named constants that represent an integral numeric type. The default integral numeric type is int, but these can be specified as another integral numeric type. The numbers that the inner variables represent can be specified as well. This can be seen in the following examples [1]:

```
enum Season
{
      myVariable1,
      myVariable2,
      myVariable3,
      myVariable4
}
```

```
enum ErrorCode : ushort
{
      None = 0,
      Unknown = 1,
      ConnectionLost = 100,
      OutlierReading = 200
}
```

The following program demonstrates some use of the enumeration type[1]:

```
public enum Season
{
    Spring,
    Summer,
    Autumn,
    Winter
}

public class EnumConversionExample
{
    public static void Main()
    {
        Season a = Season.Autumn;
        Console.WriteLine($"Integral value of {a} is {(int)a}");  // output:
Integral value of Autumn is 2

        var b = (Season)1;
        Console.WriteLine(b);  // output: Summer

        var c = (Season)4;
        Console.WriteLine©;  // output: 4
    }
}
```

The structure type, or struct type, is a value type that encapsulates data and some relevant functionality. The struct type is usually used for some data type that provides small behavior, if any. The struct has value semantics so it is recommended to use immutable structure types. Here is a small example showing how structs are used and declared [1]:

```
public struct Coords
{
    public Coords(double x, double y)
    {
        X = x;
        Y = y;
    }

    public double X { get; }
```

```
    public double Y { get; }

    public override string ToString() => $"({X}, {Y})";
}
```

There is a possibility of using the `readonly` modifier to make a struct where the instance variables do not modify the struct's state.

A tuple in C# is a convenient way to place multiple data elements into a single data structure. The following example shows how to initialize and use a tuple [1]:

```
(double, int) t1 = (4.5, 3);
Console.WriteLine($"Tuple with elements {t1.Item1} and {t1.Item2}.");
// Output:
// Tuple with elements 4.5 and 3.

(double Sum, int Count) t2 = (4.5, 3);
Console.WriteLine($"Sum of {t2.Count} elements is {t2.Sum}.");
// Output:
// Sum of 3 elements is 4.5.
```

The nullable value type in C# allows a programmer to declare a variable of a type, such as integer, double, and so on, that does not typically accept a null and allow it to contain the value of null. This is useful in a situation where the variable may have to have null assigned to it rather than some other constant. The following code demonstrates this idea [1]:

```
double? pi = 3.14;
char? letter = 'a';

int m2 = 10;
int? m = m2;

bool? flag = null;

// An array of a nullable value type:
int?[] arr = new int?[10];
```

5 Control Structures:

C#, like most popular languages, features many types of control structures. These structures include selection statements such as if-else statements and switch statements. These structures also include iteration statements such as for statements, foreach statements, do statements, and while statements. The general syntax for all of these statements is the control statement followed by a Boolean function within parentheses followed by a set of curly braces that contain some other code. The switch statement differs slightly because the statement is followed by a condition which then contains one or more cases to determine what code should be executed depending on the condition. Examples of if-else statements and while statements can be seen in the program example in section 6. The following examples below show how these other statements are implemented [1].

switch statement:

```
DisplayMeasurement(-4);  // Output: Measured value is -4; too low.
```

```
DisplayMeasurement(5);  // Output: Measured value is 5.
DisplayMeasurement(30);  // Output: Measured value is 30; too high.
DisplayMeasurement(double.NaN);  // Output: Failed measurement.

void DisplayMeasurement(double measurement)
{
    switch (measurement)
    {
        case < 0.0:
            Console.WriteLine($"Measured value is {measurement}; too low.");
            break;

        case > 15.0:
            Console.WriteLine($"Measured value is {measurement}; too high.");
            break;

        case double.NaN:
            Console.WriteLine("Failed measurement.");
            break;

        default:
            Console.WriteLine($"Measured value is {measurement}.");
            break;
    }
}
```

for statement:

```
for (int i = 0; i < 3; i++)
{
    Console.Write(i);
}
// Output:
// 012
```

foreach statement:

```
var fibNumbers = new List<int> { 0, 1, 1, 2, 3, 5, 8, 13 };
foreach (int element in fibNumbers)
{
    Console.Write($"{element} ");
}
// Output:
// 0 1 1 2 3 5 8 13
```

do statement:

```
int n = 0;
do
{
    Console.Write(n);
    n++;
} while (n < 5);
// Output:
// 01234
```

6 Program Example:

The following program is an example of a basic calculator program.

```
//Author: Cameron Goldberg

using System;

namespace ProgramExample
{
    class Program
    {
        static void Main(string[] args)
        {

            //boolean to allow user to quit loop
            bool userQuit = false;

            //loop while userQuit is false (user changes this)
            while(!userQuit)
            {

                    //display options to user
                    Console.WriteLine("Basic Calculator! Select an option by typing
the number of your selection: ");
                    Console.WriteLine("1. Add");
                    Console.WriteLine("2. Subtract");
                    Console.WriteLine("3. Multiply");
                    Console.WriteLine("4. Divide");

                    //get user's selection and assign to string
                    string userInput = Console.ReadLine();

                    //checking if user's input is equal to  the characters 1,2,3,4.
                    //otherwise, the user is informed that they made …
                    //… an invalid selection
                    if(userInput.Equals("1"))
                    {
                        //call add method to add user's numbers
                        add();
                    }
                    else if (userInput.Equals("2"))
                    {
                        //call sub method to subtract user's numbers
                        sub();
                    }
                    else if (userInput.Equals("3"))
                    {
                        //call mult method to multiply user's numbers
```

```
                    mult();
                }
                else if (userInput.Equals("4"))
                {
                    //call div method to divide user's numbers
                    div();
                }
                else
                {
                    Console.WriteLine("You did not enter a valid selection.");
                }

                //checking if user would like to continue using the program
                Console.WriteLine("Would you like to continue? (Type \'y\' to
continue): ");
                //reading user's desire to quit or keep computing!
                string userCont = Console.ReadLine();

                //changing boolean variable to true so loop ends if user does not
choose 'y'
                if(!userCont.Equals("y"))
                {
                    userQuit = true;
                }

            }//end while

        }//end main

        //method to add two variables chosen by user
        static void add()
        {
            //declaring variables to be used
            int x;
            int y;
            int result;

            //prompting user for numbers
            Console.WriteLine("Type your first addend: ");
            x = Convert.ToInt32(Console.ReadLine());
            Console.WriteLine("Type your second addend: ");
            y = Convert.ToInt32(Console.ReadLine());

            //getting sum
            result = x + y;

            //displaying result
            Console.WriteLine(x + " + " + y + " = " + result);
```

```
        }

        //method to subtract two variables chosen by user
        static void sub()
        {
            //declaring variables to be used
            int x;
            int y;
            int result;

            //prompting user for numbers
            Console.WriteLine("Type your minuend: ");
            x = Convert.ToInt32(Console.ReadLine());
            Console.WriteLine("Type your subtrahend: ");
            y = Convert.ToInt32(Console.ReadLine());

            //getting difference
            result = x - y;

            //displaying result
            Console.WriteLine(x + " - " + y + " = " + result);
        }

        //method to multiply two variables chosen by user
        static void mult()
        {
            //declaring variables to be used
            int x;
            int y;
            int result;

            //prompting user for numbers
            Console.WriteLine("Type your multiplicand: ");
            x = Convert.ToInt32(Console.ReadLine());
            Console.WriteLine("Type your multiplier: ");
            y = Convert.ToInt32(Console.ReadLine());

            //getting product
            result = x * y;

            //displaying result
            Console.WriteLine(x + " * " + y + " = " + result);
        }

        //method to divide two variables chosen by user
        static void div()
        {
            //declaring variables to be used
```

```
            int x;
            int y;
            int result;

            //prompting user for numbers
            Console.WriteLine("Type your dividend: ");
            x = Convert.ToInt32(Console.ReadLine());
            Console.WriteLine("Type your divisor: ");
            y = Convert.ToInt32(Console.ReadLine());

            //getting quotient
            result = x / y;

            //displaying result
            Console.WriteLine(x + " / " + y + " = " + result);
        }

    }//end class
}
```

Conclusion:

C# has established its roots as a programming language of choice among developers in the app and web world. Given its imperative, object-oriented nature, on top of its similarities to Java or C++, it has become a popular language.

References:

[1] Microsoft's C# Documentation, https://docs.microsoft.com/en-us/dotnet/csharp/ , last accessed 11/30/21

[2] Microsoft's .NET Documentation, https://docs.microsoft.com/en-us/dotnet/ , last accessed 11/30/2021

[3] Introduction to C#, https://www.geeksforgeeks.org/introduction-to-c-sharp/ , last accessed 11/30/2021

[4] Paritosh Pandey, Monica Wani: Research Paper on C# [Programming Language]

[5] Robert W. Sebesta: Concepts of Programming Languages. Eleventh Edition. Pearson (2016)

[6] The Simple and COOL History of C#, https://www.csharpschool.com/blog/the-simple-and-cool-history-of-csharp , last accessed 11/30/2021