C# is a modern, general-purpose, object-oriented programming language developed by Microsoft and approved by European Computer Manufacturers Association (ECMA) and International Standards Organization (ISO).

C# was developed by Anders Hejlsberg and his team during the development of .Net Framework.

C# is designed for Common Language Infrastructure (CLI), which consists of the executable code and runtime environment that allows use of various high-level languages on different computer platforms and architectures.

The following reasons make C# a widely used professional language −

* It is a modern, general-purpose programming language
* It is object oriented.
* It is component oriented.
* It is easy to learn.
* It is a structured language.
* It produces efficient programs.
* It can be compiled on a variety of computer platforms.
* It is a part of .Net Framework.

Strong Programming Features of C#

Although C# constructs closely follow traditional high-level languages, C and C++ and being an object-oriented programming language. It has strong resemblance with Java, it has numerous strong programming features that make it endearing to a number of programmers worldwide.

Following is the list of few important features of C# −

* Boolean Conditions
* Automatic Garbage Collection
* Standard Library
* Assembly Versioning
* Properties and Events
* Delegates and Events Management
* Easy-to-use Generics
* Indexers
* Conditional Compilation
* Simple Multithreading
* LINQ and Lambda Expressions
* Integration with Windows

e will discuss the tools required for creating C# programming. We have already mentioned that C# is part of .Net framework and is used for writing .Net applications. Therefore, before discussing the available tools for running a C# program, let us understand how C# relates to the .Net framework.

The .Net Framework

The .Net framework is a revolutionary platform that helps you to write the following types of applications −

* Windows applications
* Web applications
* Web services

The .Net framework applications are multi-platform applications. The framework has been designed in such a way that it can be used from any of the following languages: C#, C++, Visual Basic, Jscript, COBOL, etc. All these languages can access the framework as well as communicate with each other.

The .Net framework consists of an enormous library of codes used by the client languages such as C#. Following are some of the components of the .Net framework −

* Common Language Runtime (CLR)
* The .Net Framework Class Library
* Common Language Specification
* Common Type System
* Metadata and Assemblies
* Windows Forms
* ASP.Net and ASP.Net AJAX
* ADO.Net
* Windows Workflow Foundation (WF)
* Windows Presentation Foundation
* Windows Communication Foundation (WCF)
* LINQ

For the jobs each of these components perform, please see [ASP.Net - Introduction](https://www.tutorialspoint.com/asp.net/asp.net_introduction.htm), and for details of each component, please consult Microsoft's documentation.

Integrated Development Environment (IDE) for C#

Microsoft provides the following development tools for C# programming −

* Visual Studio 2010 (VS)
* Visual C# 2010 Express (VCE)
* Visual Web Developer

The last two are freely available from Microsoft official website. Using these tools, you can write all kinds of C# programs from simple command-line applications to more complex applications. You can also write C# source code files using a basic text editor, like Notepad, and compile the code into assemblies using the command-line compiler, which is again a part of the .NET Framework.

Visual C# Express and Visual Web Developer Express edition are trimmed down versions of Visual Studio and has the same appearance. They retain most features of Visual Studio. In this tutorial, we have used Visual C# 2010 Express.

You can download it from [Microsoft Visual Studio](https://www.microsoft.com/visualstudio/eng/downloads). It gets installed automatically on your machine.

Note: You need an active internet connection for installing the express edition.

Writing C# Programs on Linux or Mac OS

Although the.NET Framework runs on the Windows operating system, there are some alternative versions that work on other operating systems. **Mono** is an open-source version of the .NET Framework which includes a C# compiler and runs on several operating systems, including various flavors of Linux and Mac OS. Kindly check [Go Mono](http://www.go-mono.com/mono-downloads/download.html).

The stated purpose of Mono is not only to be able to run Microsoft .NET applications cross-platform, but also to bring better development tools for Linux developers. Mono can be run on many operating systems including Android, BSD, iOS, Linux, OS X, Windows, Solaris, and UNIX.

efore we study basic building blocks of the C# programming language, let us look at a bare minimum C# program structure so that we can take it as a reference in upcoming chapters.

Creating Hello World Program

A C# program consists of the following parts −

* Namespace declaration
* A class
* Class methods
* Class attributes
* A Main method
* Statements and Expressions
* Comments

Let us look at a simple code that prints the words "Hello World" −

[Live Demo](http://tpcg.io/sy8oqc)

using System;

namespace HelloWorldApplication {

class HelloWorld {

static void Main(string[] args) {

/\* my first program in C# \*/

Console.WriteLine("Hello World");

Console.ReadKey();

}

}

}

When this code is compiled and executed, it produces the following result −

Hello World

Let us look at the various parts of the given program −

* The first line of the program **using System;** - the **using** keyword is used to include the **System**namespace in the program. A program generally has multiple **using** statements.
* The next line has the **namespace** declaration. A **namespace** is a collection of classes. The *HelloWorldApplication* namespace contains the class *HelloWorld*.
* The next line has a **class** declaration, the class *HelloWorld* contains the data and method definitions that your program uses. Classes generally contain multiple methods. Methods define the behavior of the class. However, the *HelloWorld* class has only one method **Main**.
* The next line defines the **Main** method, which is the **entry point** for all C# programs. The **Main**method states what the class does when executed.
* The next line /\*...\*/ is ignored by the compiler and it is put to add **comments** in the program.
* The Main method specifies its behavior with the statement **Console.WriteLine("Hello World");**

*WriteLine* is a method of the *Console* class defined in the *System* namespace. This statement causes the message "Hello, World!" to be displayed on the screen.

* The last line **Console.ReadKey();** is for the VS.NET Users. This makes the program wait for a key press and it prevents the screen from running and closing quickly when the program is launched from Visual Studio .NET.

It is worth to note the following points −

* C# is case sensitive.
* All statements and expression must end with a semicolon (;).
* The program execution starts at the Main method.
* Unlike Java, program file name could be different from the class name.

Compiling and Executing the Program

If you are using Visual Studio.Net for compiling and executing C# programs, take the following steps −

* Start Visual Studio.
* On the menu bar, choose File -> New -> Project.
* Choose Visual C# from templates, and then choose Windows.
* Choose Console Application.
* Specify a name for your project and click OK button.
* This creates a new project in Solution Explorer.
* Write code in the Code Editor.
* Click the Run button or press F5 key to execute the project. A Command Prompt window appears that contains the line Hello World.

You can compile a C# program by using the command-line instead of the Visual Studio IDE −

* Open a text editor and add the above-mentioned code.
* Save the file as **helloworld.cs**
* Open the command prompt tool and go to the directory where you saved the file.
* Type **csc helloworld.cs** and press enter to compile your code.
* If there are no errors in your code, the command prompt takes you to the next line and generates **helloworld.exe** executable file.
* Type **helloworld** to execute your program.
* You can see the output Hello World printed on the screen.

C# is an object-oriented programming language. In Object-Oriented Programming methodology, a program consists of various objects that interact with each other by means of actions. The actions that an object may take are called methods. Objects of the same kind are said to have the same type or, are said to be in the same class.

For example, let us consider a Rectangle object. It has attributes such as length and width. Depending upon the design, it may need ways for accepting the values of these attributes, calculating the area, and displaying details.

Let us look at implementation of a Rectangle class and discuss C# basic syntax −

using System;

namespace RectangleApplication {

class Rectangle {

// member variables

double length;

double width;

public void Acceptdetails() {

length = 4.5;

width = 3.5;

}

public double GetArea() {

return length \* width;

}

public void Display() {

Console.WriteLine("Length: {0}", length);

Console.WriteLine("Width: {0}", width);

Console.WriteLine("Area: {0}", GetArea());

}

}

class ExecuteRectangle {

static void Main(string[] args) {

Rectangle r = new Rectangle();

r.Acceptdetails();

r.Display();

Console.ReadLine();

}

}

}

When the above code is compiled and executed, it produces the following result −

Length: 4.5

Width: 3.5

Area: 15.75

The *using* Keyword

The first statement in any C# program is

using System;

The **using** keyword is used for including the namespaces in the program. A program can include multiple using statements.

The *class* Keyword

The **class** keyword is used for declaring a class.

Comments in C#

Comments are used for explaining code. Compilers ignore the comment entries. The multiline comments in C# programs start with /\* and terminates with the characters \*/ as shown below −

/\* This program demonstrates

The basic syntax of C# programming

Language \*/

Single-line comments are indicated by the '//' symbol. For example,

}//end class Rectangle

Member Variables

Variables are attributes or data members of a class, used for storing data. In the preceding program, the *Rectangle* class has two member variables named *length* and *width*.

Member Functions

Functions are set of statements that perform a specific task. The member functions of a class are declared within the class. Our sample class Rectangle contains three member functions: *AcceptDetails*, *GetArea* and *Display*.

Instantiating a Class

In the preceding program, the class *ExecuteRectangle* contains the *Main()* method and instantiates the *Rectangle* class.

Identifiers

An identifier is a name used to identify a class, variable, function, or any other user-defined item. The basic rules for naming classes in C# are as follows −

* A name must begin with a letter that could be followed by a sequence of letters, digits (0 - 9) or underscore. The first character in an identifier cannot be a digit.
* It must not contain any embedded space or symbol such as? - + ! @ # % ^ & \* ( ) [ ] { } . ; : " ' / and \. However, an underscore ( \_ ) can be used.
* It should not be a C# keyword.

C# Keywords

Keywords are reserved words predefined to the C# compiler. These keywords cannot be used as identifiers. However, if you want to use these keywords as identifiers, you may prefix the keyword with the @ character.

In C#, some identifiers have special meaning in context of code, such as get and set are called contextual keywords.

The following table lists the reserved keywords and contextual keywords in C# −

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Reserved Keywords** | | | | | | |
| abstract | as | base | bool | break | byte | case |
| catch | char | checked | class | const | continue | decimal |
| default | delegate | do | double | else | enum | event |
| explicit | extern | false | finally | fixed | float | for |
| foreach | goto | if | implicit | in | in (generic modifier) | int |
| interface | internal | is | lock | long | namespace | new |
| null | object | operator | out | out (generic modifier) | override | params |
| private | protected | public | readonly | ref | return | sbyte |
| sealed | short | sizeof | stackalloc | static | string | struct |
| switch | this | throw | true | try | typeof | uint |
| ulong | unchecked | unsafe | ushort | using | virtual | void |
| volatile | while |  |  |  |  |  |
| **Contextual Keywords** | | | | | | |
| add | alias | ascending | descending | dynamic | from | get |
| global | group | into | join | let | orderby | partial (type) |
| partial (method) | remove | select | set |  |  |  |