<http://msdn.microsoft.com/en-us/library/yyaad03b.aspx>

Visual C# Language Concepts

**C# for C++ Developers**

The following table contains important comparisons between C# and native C++, which does not use **/clr**. If you are a C++ programmer, this table will give you the most important differences between the two languages at a glance.

|  |  |
| --- | --- |
| **NoteNote:** | |
| C++ and C# projects are derived from different project models. For more information about the differences between C++ and C# projects, see [Item Management in Projects](http://msdn.microsoft.com/en-us/library/7kxx8f8e.aspx) and [Using Solution Explorer](http://msdn.microsoft.com/en-us/library/bbck0dh6.aspx). | |
| **Feature** | **Refer to the topic** |
| Inheritance: In C++, classes and structs are virtually identical whereas in C#, they are quite different. C# classes can implement any number of interfaces, but can inherit from only one base class. Furthermore, C# structs do not support inheritance, and do not support explicit default constructors (one is provided by default). | [class](http://msdn.microsoft.com/en-us/library/0b0thckt.aspx)  [interface](http://msdn.microsoft.com/en-us/library/87d83y5b.aspx)  [struct (C# Reference)](http://msdn.microsoft.com/en-us/library/ah19swz4.aspx) |
| Arrays: In C++ an array is merely a pointer. In C#, arrays are objects that include methods and properties. For example, the size of an array can be queried via the [Length](http://msdn.microsoft.com/en-us/library/system.array.length.aspx) property. C# arrays also employ *indexers* that verify each index used to access the array. The syntax for declaring C# arrays is different from that for C++ arrays: the tokens "[]" appear following the array type in C#, not the variable. | [Arrays (C# Programming Guide)](http://msdn.microsoft.com/en-us/library/9b9dty7d.aspx)  [Indexers (C# Programming Guide)](http://msdn.microsoft.com/en-us/library/6x16t2tx.aspx) |
| Booleans: In C++, the **bool** type is essentially an integer. In C#, there is no conversion between the **bool** type and other types. | [bool](http://msdn.microsoft.com/en-us/library/c8f5xwh7.aspx) |
| The **long** type: In C#, the **long** type is 64 bits, while in C++, it is 32 bits. | [long](http://msdn.microsoft.com/en-us/library/ctetwysk.aspx) |
| Passing parameters: In C++, all variables are passed by value unless explicitly passed with a pointer or a reference. In C#, classes are passed by reference and structs are passed by value unless explicitly passed by reference with the **ref** or **out** parameter modifiers. | [struct](http://msdn.microsoft.com/en-us/library/ah19swz4.aspx)  [class](http://msdn.microsoft.com/en-us/library/0b0thckt.aspx)  [ref (C# Reference)](http://msdn.microsoft.com/en-us/library/14akc2c7.aspx)  [out (C# Reference)](http://msdn.microsoft.com/en-us/library/t3c3bfhx.aspx) |
| The **switch** statement: Unlike the C++ **switch** statement, C# does not support fall-through from one case label to another. | [switch](http://msdn.microsoft.com/en-us/library/06tc147t.aspx) |
| Delegates: C# delegates are roughly similar to function pointers in C++, are type-safe and secure. | [delegate](http://msdn.microsoft.com/en-us/library/900fyy8e.aspx) |
| Base-class methods: C# supports the **base** keyword for calling the overridden base class members from derived classes. Also, overriding virtual or abstract methods is explicit in C#, using the **override** keyword. | [base](http://msdn.microsoft.com/en-us/library/hfw7t1ce.aspx)  See also the examples for[override](http://msdn.microsoft.com/en-us/library/ebca9ah3.aspx) |
| Method hiding: C++ supports the implicit "hiding" of method through inheritance. In C#, you must use the **new**modifier to explicitly hide an inherited members. | [new](http://msdn.microsoft.com/en-us/library/51y09td4.aspx) |
| Preprocessor directives are used for conditional compilation. No header files are used in C#. | [C# Preprocessor Directives](http://msdn.microsoft.com/en-us/library/ed8yd1ha.aspx) |
| Exception handling: C# provides the **finally** keyword to provide for code that should be executed regardless of whether an exception is thrown. | [try-finally](http://msdn.microsoft.com/en-us/library/zwc8s4fz.aspx)  [try-catch-finally](http://msdn.microsoft.com/en-us/library/dszsf989.aspx) |
| C# operators: C# supports additional operators such as **is** and **typeof**. It also introduces different functionality for some logical operators. | [& Operator](http://msdn.microsoft.com/en-us/library/sbf85k1c.aspx)  [| Operator](http://msdn.microsoft.com/en-us/library/kxszd0kx.aspx)  [^ Operator](http://msdn.microsoft.com/en-us/library/zkacc7k1.aspx)  [is](http://msdn.microsoft.com/en-us/library/scekt9xw.aspx)  [typeof](http://msdn.microsoft.com/en-us/library/58918ffs.aspx) |
| The **typedef** keyword. In C++, typedef is used to create shorter or more convenient names for types that have already been declared. In C#, the **using** directive provides this capability. | [using Directive (C# Reference)](http://msdn.microsoft.com/en-us/library/sf0df423.aspx) |
| The **extern** keyword: In C++, **extern** is used to import types. In C#, **extern** is used to create aliases for using different versions of the same assembly. | [extern](http://msdn.microsoft.com/en-us/library/e59b22c5.aspx) |
| The **static** keyword: In C++, **static** can be used both to declare class-level entities and to declare types that are specific to a module. In C#, **static** is only used to declare class-level entities. | [static](http://msdn.microsoft.com/en-us/library/98f28cdx.aspx) |
| The **Main** method in C# is declared differently from the **main** function in C++. In C# it is capitalized, and always**static**. Also, support for processing of command-line arguments is much more robust in C#. | [Main() and Command-Line Arguments (C# Programming Guide)](http://msdn.microsoft.com/en-us/library/acy3edy3.aspx) |
| Pointers are allowed in C#, but only in **unsafe** mode. | [unsafe](http://msdn.microsoft.com/en-us/library/chfa2zb8.aspx) |
| Overloading operators is performed differently in C#. | [C# Operators](http://msdn.microsoft.com/en-us/library/6a71f45d.aspx) |
| Strings: In C++ a string is simply an arra of characters. In C#, strings are object that support robust searching methods. | [string](http://msdn.microsoft.com/en-us/library/362314fe.aspx)  [String](http://msdn.microsoft.com/en-us/library/system.string.aspx) |
| The **foreach** keyword enables you to iterate through arrays and collections. | [foreach, in](http://msdn.microsoft.com/en-us/library/ttw7t8t6.aspx) |
| Globals: In C#, global methods and variables are not supported. Methods and variables must be contained within a **class** or **struct**. | [General Structure of a C# Program](http://msdn.microsoft.com/en-us/library/w2a9a9s3.aspx) |
| The **#define** preprocessing directive: In C++ the **#define** directive is commonly used to declare constant values. In C# the **#define** directive cannot be used for this purpose. Constants in C# are best defined as enumerated types (integral values only) or as static members of a class or struct. If you have several such constants, consider creating a separate "Constants" class to hold them. | [static (C# Reference)](http://msdn.microsoft.com/en-us/library/98f28cdx.aspx)  [const (C# Reference)](http://msdn.microsoft.com/en-us/library/e6w8fe1b.aspx)  [enum (C# Reference)](http://msdn.microsoft.com/en-us/library/sbbt4032.aspx) |
| Importing types: In C++, types common to multiple modules are placed in header files. In C#, this information is available via metadata. | [using](http://msdn.microsoft.com/en-us/library/zhdeatwt.aspx)  [Metadata Overview](http://msdn.microsoft.com/en-us/library/xcd8txaw.aspx) |
| Local variables in C# cannot be used before they are initialized. | [Methods (C# Programming Guide)](http://msdn.microsoft.com/en-us/library/ms173114.aspx) |
| Memory management: C++ is not a garbage collected language; memory that is not explicitly release remains allocated until the process terminates. C# is a garbage collected language. | [Garbage Collection](http://msdn.microsoft.com/en-us/library/0xy59wtx.aspx) |
| Destructors: C# has different syntax for deterministically releasing unmanaged resources. | [Destructors](http://msdn.microsoft.com/en-us/library/66x5fx1b.aspx)  [using Statement (C# Reference)](http://msdn.microsoft.com/en-us/library/yh598w02.aspx) |
| Constructors: Similar to C++, if you do not provide a class constructor in C#, a default constructor is automatically generated for you. The default constructor initializes all the fields to their default values. | [Instance Constructors](http://msdn.microsoft.com/en-us/library/k6sa6h87.aspx)  [Default Values Table](http://msdn.microsoft.com/en-us/library/83fhsxwc.aspx) |
| C# does not support bit fields. | [C++ Bit Fields](http://msdn.microsoft.com/en-us/library/ewwyfdbe.aspx) |
| C# input/output services and formatting rely on the run-time library of the .NET Framework. | [C# Language Tour](http://msdn.microsoft.com/en-us/library/67ef8sbd.aspx)  [Formatting Numeric Results Table](http://msdn.microsoft.com/en-us/library/s8s7t687.aspx) |
| In C#, method parameters cannot have default values. Use method overloads if you want to achieve the same effect. | [Compiler Error CS0241](http://msdn.microsoft.com/en-us/library/294000kk.aspx) |
| In C#, generic types and methods provide for type parameterization in a way that is similar to C++ templates. There are significant differences, however. For example, in C# generic type information is preserved at run time. | [Generics in C#](http://msdn.microsoft.com/en-us/library/0x6a29h6.aspx) |
| The **as** keyword is similar to a standard cast, except that rather than throw an exception if the conversion fails, the return value is null. This is similar to using **static\_cast** in C++, which, unlike **dynamic\_cast**, performs no run-time check and hence does not throw an exception on failure. | [as (C# Reference)](http://msdn.microsoft.com/en-us/library/cscsdfbt.aspx) |

For more information about comparisons between keywords in C# and other programming languages, see [Language Equivalents](http://msdn.microsoft.com/en-us/library/czz35az4.aspx). For information on the general structure of C# applications, see [General Structure of a C# Program (C# Programming Guide)](http://msdn.microsoft.com/en-us/library/w2a9a9s3.aspx)