PRESENTATION ON DATA TYPES IN C# .NET

[Built-In Data Types](javascript:void(0))

The list of C# data types and their aliases. As you can see, the first eight of these correspond to the primitive types available in Java. Note, however, that Java's **boolean** is called **bool** in C#.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Short Name** | **.NET Class** | **Type** | **Width** | **Range (bits)** |
| **byte** | [Byte](https://msdn.microsoft.com/en-us/library/system.byte(v=vs.90).aspx) | Unsigned integer | 8 | 0 to 255 |
| **sbyte** | [SByte](https://msdn.microsoft.com/en-us/library/system.sbyte(v=vs.90).aspx) | Signed integer | 8 | -128 to 127 |
| **int** | [Int32](https://msdn.microsoft.com/en-us/library/system.int32(v=vs.90).aspx) | Signed integer | 32 | -2,147,483,648 to 2,147,483,647 |
| **uint** | [UInt32](https://msdn.microsoft.com/en-us/library/system.uint32(v=vs.90).aspx) | Unsigned integer | 32 | 0 to 4294967295 |
| **short** | [Int16](https://msdn.microsoft.com/en-us/library/system.int16(v=vs.90).aspx) | Signed integer | 16 | -32,768 to 32,767 |
| **ushort** | [UInt16](https://msdn.microsoft.com/en-us/library/system.uint16(v=vs.90).aspx) | Unsigned integer | 16 | 0 to 65535 |
| **long** | [Int64](https://msdn.microsoft.com/en-us/library/system.int64(v=vs.90).aspx) | Signed integer | 64 | -9223372036854775808 to 9223372036854775807 |
| **ulong** | [UInt64](https://msdn.microsoft.com/en-us/library/system.uint64(v=vs.90).aspx) | Unsigned integer | 64 | 0 to 18446744073709551615 |
| **float** | [Single](https://msdn.microsoft.com/en-us/library/system.single(v=vs.90).aspx) | Single-precision floating point type | 32 | -3.402823e38 to 3.402823e38 |
| **double** | [Double](https://msdn.microsoft.com/en-us/library/system.double(v=vs.90).aspx) | Double-precision floating point type | 64 | -1.79769313486232e308 to 1.79769313486232e308 |
| **char** | [Char](https://msdn.microsoft.com/en-us/library/system.char(v=vs.90).aspx) | A single Unicode character | 16 | Unicode symbols used in text |
| **bool** | [Boolean](https://msdn.microsoft.com/en-us/library/system.boolean(v=vs.90).aspx) | Logical Boolean type | 8 | True or false |
| **object** | [Object](https://msdn.microsoft.com/en-us/library/system.object(v=vs.90).aspx) | Base type of all other types |  |  |
| **string** | [String](https://msdn.microsoft.com/en-us/library/system.string(v=vs.90).aspx) | A sequence of characters |  |  |
| **decimal** | [Decimal](https://msdn.microsoft.com/en-us/library/system.decimal(v=vs.90).aspx) | Precise fractional or integral type that can represent decimal numbers with 29 significant digits | 128 | ±1.0 × 10e−28 to ±7.9 × 10e28 |

## [Constants](javascript:void(0))

* Both Java and C# provide the ability to declare a variable whose value is specified at compile time and cannot be changed at runtime.
* Java uses the **final** field modifier to declare such a variable, while C# uses the [const](https://msdn.microsoft.com/en-us/library/e6w8fe1b(v=vs.90).aspx) keyword.
* In addition to **const**, C# provides the [readonly](https://msdn.microsoft.com/en-us/library/acdd6hb7(v=vs.90).aspx) keyword to declare variables that can be assigned a value once at runtime--either in the declaration statement or else in the constructor. After initialization, the value of a **readonly**variable cannot change

## [Enumerations](javascript:void(0))

* Enumerations, or enums, are used to group named constants similar to how they are used in C and C++.
* In C#, enums are value types, and enum constants must be integral numeric values. The **ToString** method can be used to print out string representations of the named constants.

## [Strings](javascript:void(0))

* A string in C# is a sequence of Unicode characters
* Even though a string is a reference type in C#, the **==** and **!=** operator will, by default, compare the string values rather then references.
* A char is a single Unicode character.

## Arrays

* The array is a complex data type which handles a collection of elements. Each of the elements can be accessed by an index.
* All the elements of an array must be of the same data type.
* Array types are [reference types](https://msdn.microsoft.com/en-us/library/490f96s2(v=vs.90).aspx). This type implements [IEnumerable](https://msdn.microsoft.com/en-us/library/system.collections.ienumerable(v=vs.90).aspx) and [IEnumerable<T>](https://msdn.microsoft.com/en-us/library/9eekhta0(v=vs.90).aspx)

## DateTime

* The DateTime is a value type. It gives the date and time of day.

## Type casting

* Converting one data type to another one is known as Type casting
* Type conversion or typecasting refers to changing an entity/object of one data type into another.
* There are two types of conversion: implicit and explicit.

Implicit Conversion

* Implicit type conversion, also known as coercion, is an automatic type conversion by the compiler.

|  |  |
| --- | --- |
| **Source Type** | **Target Type** |
| Byte | short, ushort, int, uint, long, ulong, float, double, or decimal |
| Sbyte | short, int, long, float, double, or decimal |
| Int | long, float, double, or decimal |
| Uint | long, ulong, float, double, or decimal |
| Short | int, long, float, double, or decimal |
| Ushort | int, uint, long, ulong, float, double, or decimal |
| Long | float, double, or decimal |
| Ulong | float, double, or decimal |
| Float | double |
| Char | ushort, int, uint, long, ulong, float, double, or decimal |

Explicit Conversion

* For converting a double value to a float value. Explicit conversion is done by specifying the intended type between two square brackets.

|  |  |
| --- | --- |
| **Source Type** | **Target Type** |
| Byte | sbyte or char |
| Sbyte | byte, ushort, uint, ulong, or char |
| Int | sbyte, byte, short, ushort, uint, ulong, or char |
| Uint | sbyte, byte, short, ushort, int, or char |
| Short | sbyte, byte, ushort, uint, ulong, or char |
| Ushort | sbyte, byte, short, or char |
| Long | sbyte, byte, short, ushort, int, uint, ulong, or char |
| Ulong | sbyte, byte, short, ushort, int, uint, long, or char |
| Float | sbyte, byte, short, ushort, int, uint, long, ulong, char, ordecimal |
| Double | sbyte, byte, short, ushort, int, uint, long, ulong, char, float, or decimal |
| Char | sbyte, byte, or short |
| Decimal | sbyte, byte, short, ushort, int, uint, long, ulong, char, float, or double |

Example:

float a;

double b = 13.5;

int c;

a = (float) b;

c = (int) a;

## Nullable types

* Value types cannot be assigned a null literal, but a reference types can.
* Applications that work with databases always deal with the null value.
* Special nullable types are introduced into the C# language.
* Nullable types are instances of the System.Nullable<T> struct.
* There are two ways how to declare a nullable type. Either with the Nullable<T> generic structure in which the type is specified between the angle brackets. Or we can use a question mark after the type.

Example:

Nullable<bool> male = null;

int? age = null;

## [Value and Reference Types](javascript:void(0))

C# supports two kinds of variable types:

* **Value types**

These are the built-in primitive data types, such as char, int, and float, as well as user-defined types declared with struct.

Structs:

* Structs are having a access specifiier and they can handle arguments

Example:

public struct CoOrds

{

public int x, y;

public CoOrds(int p1, int p2)

{

x = p1;

y = p2;

}

}

class TestCoOrds

{

static void Main()

{

// Initialize:

CoOrds coords1 = new CoOrds();

CoOrds coords2 = new CoOrds(10, 10);

// Display results:

Console.Write("CoOrds 1: ");

Console.WriteLine("x = {0}, y = {1}", coords1.x, coords1.y); Console.Write("CoOrds 2: ");

Console.WriteLine("x = {0}, y = {1}", coords2.x, coords2.y); Console.WriteLine("Press any key to exit.");

Console.ReadKey();

}

}

/\* Output:

CoOrds 1: x = 0, y = 0

CoOrds 2: x = 10, y = 10

\*/

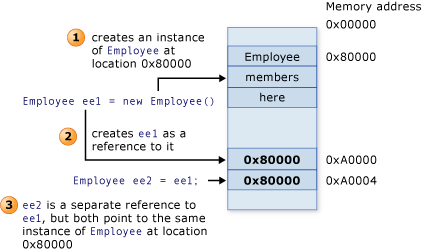
* **Reference types**

Classes and other complex data types that are constructed from the primitive types. Variables of such types do not contain an instance of the type, but merely a reference to an instance.

At run time, when you declare a variable of a reference type, the variable contains the value [null](https://msdn.microsoft.com/en-us/library/edakx9da(v=vs.90).aspx) until you explicitly create an instance of the object by using the [new](https://msdn.microsoft.com/en-us/library/51y09td4(v=vs.90).aspx) operator, or assign it an object that has been created

Comparison:

|  |  |
| --- | --- |
| VALUE | REFERENCE |
| * Value types are either stack-allocated or allocated inline in a structure. * Value type variables directly contain their values * Eg: int a=20;   int b=a;   * In the above example both a and b will have different memory location so hence alteration in one of the variable doesn’t affect another variable. | * Reference types are heap-allocated. * If reference type of one is assigned to another both the created objects will have same reference to an instant rather than the value stored in them. * Eg: Employee e1=new Employee();   Employee e2=e1;   * In above example e1 sores the address of the instance of employee class,so e2 will also contain the same instance’s address. * Which means on alteration to e2,e1 aswell gets affected |



# Boxing and Unboxing

* Boxing and unboxing allow any type to be treated as an object.
* Both reference and value types are derived from the ultimate base class [Object](https://msdn.microsoft.com/en-us/library/system.object(v=vs.90).aspx).
* In cases where it is necessary for a value type to behave like an object, a **wrapper**that makes the value type *look like a reference object* is allocated on the heap, and the value type's value is copied into it.
* The wrapper is marked so the system knows that it contains a value type. This process is known as boxing, and the reverse process is known as unboxing.