

Testing Mentor Quick-Cue™

.Net C# Data Types

Data type overview

User inputs such as characters, numbers, and strings of text are passed from the user interface as variable data and stored as **values** in the computer's memory. Other **values** used by a computer program may be defined within the software code as constants which are generally used for internal calculations. A **value** is stored in software as a series of bits. But, the various sequences of bits are not infinite and it may be possible for duplicate sequences of bits represent different types of data so programming languages use a type system. The **data type** for values and objects defines the attributes of user variables and constants, and also constrains the operations that can be used to manipulate those values.

All programming languages rely on a type system to store and manipulate values. Some programming languages hide the nuances of its type system through a dynamic type system. But, strongly typed languages such as C# rely on a static type system to decrease the probability of programming errors through early-binding to check for type safety, machine instruction optimization, self documentation and abstraction of various values.

The C# language relies on the Common Type System (CTS) outlined in ISO/IEC 23271 and ECMA 335 specifications. The CTS groups' data types into 3 separate categories: pointer types, value types, and reference types. Pointer types can only be used in 'unsafe' code and is beyond the scope of this discussion. Value types store actual data, and reference types store reference to actual data. This job aid is a quick reference guide of reference and value **built-in types** used in the C# programming language.

Built-in Value Types

| Keyword | CTS | Size | Range |
|----------------|----------------|------------------------------|--|
| byte | System.Byte | 8-bit unsigned integer | 0 – 255 |
| sbyte | System.SByte | 8-bit signed integer | -128 – + 127 |
| int | System.Int32 | 32-bit signed integer | -2147483648 – +2147483647 |
| uint | System.UInt32 | 32-bit unsigned integer | 0 – 4294967295 |
| short | System.Int16 | 16-bit signed integer | -32,768 – + 32,767 |
| ushort | System.UInt16 | 16-bit unsigned integer | 0 – 65535 |
| long | System.Int64 | 64-bit signed integer | -9223372036854775808 – +9223372036854775807 |
| ulong | System.UInt64 | 64-bit unsigned integer | 0 – 18,446,744,073,709,551,615 |
| float | System.Single | 32-bit floating point number | $\pm 1.5 \times 10^{-45} - \pm 3.4 \times 10^{38}$ |
| double | System.Double | 64-bit floating point value | $\pm 5.0 \times 10^{-324} - \pm 1.7 \times 10^{308}$ |
| decimal | System.Decimal | 128-bit floating point value | $\pm 1.0 \times 10^{-28} - \pm 7.9 \times 10^{28}$ |
| bool | System.Boolean | | true or false |
| char | System.Char | A single 16-bit character | U+0000 – U+FFFF |

Testing Mentor Quick-Cue™

There are two more types of value types which we can define, these are Structs and Enumeration.

Structs

Struct is a special kind of class that is value type rather than a reference type. Value types are stored on the Stack, and structs also are stored on the stack so they can be copied and created efficiently.

Enumeration

Enumerations are the user defined integer type. Enum keyword is used to declare an enumeration. Basically the main function is that it makes our code easier to maintain. Variables are assigned only legitimate values. The enumerator also makes our code cleaner by creating deceptive names for integers.

Reference Types

| keyword | CTS Type | Description |
|---------|---------------|----------------------------------|
| object | System.Object | Root for all other types in CTS. |
| string | System.String | Unicode character string |