CHAPTER 3

Variables

Variables are used for storing data in memory during program execution.

Data Types

Depending on what data you need to store, there are several different kinds of data types. The *simple types* in C# consist of four signed integer types and four unsigned, three floating-point types, as well as `char` and `bool`.

Data Type	Size (Bits)	Description
`Sbyte`	8	Signed integers
`short`	16	
`int`	32	
`long`	64	
`Byte`	8	Unsigned integers
`ushort`	16	
`uint`	32	
`ulong`	64	
`Float`	32	Floating-point numbers
`double`	64	
`decimal`	128	
`Char`	16	Unicode character
`Bool`	4	Boolean value

M. Olsson, *C# 8 Quick Syntax Reference*, https://doi.org/10.1007/978-1-4842-5577-3_3

Declaration

In C#, a variable must be *declared* (created) before it can be used. To declare a variable, you start with the data type you want it to hold followed by a variable name. The name can be almost anything you want, but it is a good idea to give your variables names that are closely related to the value they will hold.

```
int myInt;
```

Assignment

A value is assigned to the variable by using the equals sign, which is the assignment operator (=). The variable then becomes *defined* or *initialized*.

```
myInt = 10;
```

The declaration and assignment can be combined into a single statement.

```
int myInt = 10;
```

If multiple variables of the same type are needed, there is a shorthand way of declaring or defining them by using the comma operator (,).

```
int myInt = 10, myInt2 = 20, myInt3;
```

Once a variable has been defined (declared and assigned), it can be used by referencing the variable's name.

```
System.Console.Write(myInt); // "10"
```

Integer Types

There are four signed integer types that can be used depending on how large a number you need the variable to hold.

```
// Signed integers
sbyte myInt8  = 2; // -128   to +127
short myInt16 = 1; // -32768 to +32767
int   myInt32 = 0; // -2^31  to +2^31-1
long  myInt64 =-1; // -2^63  to +2^63-1
```

The unsigned types can be used if you only need to store positive values.

```
// Unsigned integers
byte   uInt8  = 0;  // 0 to 255
ushort uInt16 = 1;  // 0 to 65535
uint   uInt32 = 2;  // 0 to 2^32-1
ulong  uInt64 = 3;  // 0 to 2^64-1
```

In addition to the standard decimal notation, integers can also be assigned using hexadecimal notation. As of C# 7.0, there is a binary notation as well. Hexadecimal numbers are prefixed with `0x` and binary numbers with `0b`.

```
int myHex = 0xF;    // 15 in hexadecimal (base 16)
int myBin = 0b0100; // 4 in binary (base 2)
```

Version 7.0 of C# also added a digit separator (_) to improve readability of long numbers. This digit separator can appear anywhere within the number, as well as at the beginning of the number as of C# 7.2.

```
int myBin = 0b_0010_0010; // 34 in binary notation (0b)
```

Floating-Point Types

The floating-point types can store real numbers with different levels of precision. Constant floating-point numbers in C# are always kept as doubles, so in order to assign such a number to a float variable, an `F` character needs to be appended to convert the number to the `float` type. The same applies to the `M` character for decimals.

```
float   myFloat   = 3.14F; // 7 digits of precision
double  myDouble  = 3.14;  // 15-16 digits of precision
decimal myDecimal = 3.14M; // 28-29 digits of precision
```

A more common and useful way to convert between data types is to use an explicit cast. An *explicit cast* is performed by placing the desired data type in parentheses before the variable or constant that is to be converted. This will convert the value to the specified type, in this case, `float`, before the assignment occurs.

```
myFloat = (float) myDecimal; // explicit cast
```

The precisions shown earlier refer to the total number of digits that the types can hold. For example, when attempting to assign more than seven digits to a `float`, the least significant ones will get rounded off.

```
myFloat = 12345.6789F; // rounded to 12345.68
```

Floating-point numbers can be assigned using either decimal or exponential notation, as in the following example.

```
myDouble = 3e2; // 3*10^2 = 300
```

Char Type

The `char` type can contain a single Unicode character delimited by single quotes.

```
char c = 'a'; // Unicode char
```

Bool Type

The `bool` type can store a Boolean value, which is a value that can be either true or false. These values are specified with the `true` and `false` keywords.

```
bool b = true; // bool value
```

Variable Scope

The *scope* of a variable refers to the code block within which it is possible to use that variable without qualification. For example, a local variable is a variable declared within a method. Such a variable will only be available within that method's code block, after it has been declared. Once the scope of the method ends, the local variable will be destroyed.

```
static void Main()
{
  int localVar; // local variable
}
```

In addition to local variables, C# has field and parameter type variables, which will be looked at in later chapters. However, C# does not have global variables, unlike C++.