1. What is a statement?

A statement is a command that performs an action, such as calculating a value and storing the result or displaying a message to a user. You combine statements to create methods. Think of a method as a named sequence of statements. Main, which was introduced in the previous chapter, is an example of a method.

1. What is an identifier?

Identifiers are the names that you use to identify the elements in your programs, such as namespaces, classes, methods, and variables. In C#, you must adhere to the following syntax rules when choosing identifiers: must start with a letter or an underscore; must use only letters, digits, and underscore characters.

1. What is a variable?

A variable is a storage location that holds a value. It is like a box in the computer’s memory that holds temporary information. Each variable in a program must have an unambiguous name that uniquely identifies it in the context in which it is used. You use a variable’s name to refer to the value it holds.

1. Are primitive types and value types the same thing?

The primitive types are int, long, float, double, decimal, string, char, and bool. Most of the primitive types built into C#, such as int, float, double, and char (but not string) are collectively called value types. The string and object types are classes (reference types) rather than structures. Primitive types have a fixed size, and when you declare a variable as a value type, the compiler generates code that allocates a block of memory big enough to hold a corresponding value. For example, declaring an int variable causes the compiler to allocate 4 bytes of memory (32 bits) to hold the integer value. A statement that assigns a value (such as 42) to the int causes the value to be copied into this block of memory.

1. How are arithmetic operators and variable types related?

All of the arithmetic operators can be used on values of type char, int, long, float, double, or decimal. However, except for the plus operator (+), you can’t use the arithmetic operators on values of type string, and you cannot use any of them with values of type bool. However, the plus operator (+) can be used to concatenate string values.

1. How do you turn an integer into a string? How do you turn a string into an integer?

From Book: intVar.ToString()

From Book: System.Int32.Parse()

Used in Class: Convert.ToString

Used in Class: Convert.ToInt32

1. What is the difference between precedence and associativity?

Precedence governs the order in which an expression’s operators are evaluated. When an expression contains different operators that have the same precedence, associativity is the direction (left or right) in which the operands of an operator are evaluated.

1. How are the prefix and postfix increment and decrement operators evaluated differently?

Postfix

x = 42;

Console.WriteLine(x++); // x is now 43, 42 is printed

Prefix

x = 42;

Console.WriteLine(++x); // x is now 43, 43 is printed

In the expression x++, the variable x occurs first, so its value is used as the value of the expression before x is incremented. In the expression ++x, the operator occurs first, so its operation is performed before the value of x is evaluated as the result.

1. What is string interpolation?

String interpolation is used for concatenating strings more efficiently than using the + operator.

An example of using the + operator:

MessageDialog msg = new MessageDialog("Hello " + userName.Text);

String interpolation lets you use the following syntax instead:

MessageDialog msg = new MessageDialog($"Hello {userName.Text}");

The $ symbol at the start of the string indicates that it is an interpolated string and that any expressions between the { and } characters should be evaluated and the result substituted in their place. Without the leading $ symbol, the string {username.Text} would be treated literally.

1. What does the var keyword do?

Typically, the value you assign to a variable must be of the same type as the variable. For example, you can assign an int value only to an int variable. You can also ask the C# compiler to infer the type of a variable from an expression and use this type when declaring the variable by using the var keyword in place of the type, as demonstrated here:

var myVariable = 99;

var myOtherVariable = "Hello";

The variables myVariable and myOtherVariable are referred to as implicitly typed variables. The var keyword causes the compiler to deduce the type of the variables from the types of the expressions used to initialize them. You can use the var keyword only when you supply an expression to initialize a variable. The following declaration is illegal and causes a compilation error:

var yetAnotherVariable; // Error - compiler cannot infer type