C# Program Building Blocks

The types described in the previous article are built using these building blocks: **members**, **expressions**, and **statements**.

Members

The members of a class are either static members or instance members. Static members belong to classes, and instance members belong to objects (instances of classes).

The following list provides an overview of the kinds of members a class can contain.

- **Constants**: Constant values associated with the class
- **Fields**: Variables that are associated of the class
- **Methods**: Actions that can be performed by the class
- **Properties**: Actions associated with reading and writing named properties of the class
- **Indexers**: Actions associated with indexing instances of the class like an array
- **Events**: Notifications that can be generated by the class
- **Operators**: Conversions and expression operators supported by the class
- **Constructors**: Actions required to initialize instances of the class or the class itself
- **Finalizers**: Actions performed before instances of the class are permanently discarded
- **Types**: Nested types declared by the class

Accessibility

Each member of a class has an associated accessibility, which controls the regions of program text that can access the member. There are six possible forms of accessibility. The access modifiers are summarized below.

- **`public`**: Access isn't limited.
- **`private`**: Access is limited to this class.
- **`protected`**: Access is limited to this class or classes derived from this class.
- **`internal`**: Access is limited to the current assembly (.exe or .dll).
- **`protected internal`**: Access is limited to this class, classes derived from this class, or classes within the same assembly.
- **`private protected`**: Access is limited to this class or classes derived from this type within the same assembly.

Fields

A field is a variable that is associated with a class or with an instance of a class. A field declared with the static modifier defines a static field. A static field identifies exactly one storage location. No matter how many instances of a class are created, there's only ever one copy of a static field. A field declared without the static modifier defines an instance field. Every instance of a class contains a separate copy of all the instance fields of that class.

Operators

An operator is a member that defines the meaning of applying a particular expression operator to instances of a class. Three kinds of operators can be defined: unary operators, binary operators, and conversion operators. All operators must be declared as public and static.

For the complete list of C# operators ordered by precedence level, see C# operators.

Expressions

Expressions are constructed from operands and operators. The operators of an expression indicate which operations to apply to the operands. Examples of operators include `+`, `-`, `*`, `/`, and `new`. Examples of operands include literals, fields, local variables, and expressions.

When an expression contains multiple operators, the precedence of the operators controls the order in which the individual operators are evaluated. For example, the expression `x + y * z` is evaluated `as x + (y * z)` because the * operator has higher precedence than the + operator.

When an operand occurs between two operators with the same precedence, the associativity of the operators controls the order in which the operations are performed:

Except for the assignment and null-coalescing operators, all binary operators are left-associative, meaning that operations are performed from left to right. For example, x + y + z is evaluated as (x + y) + z.

The assignment operators, the null-coalescing ?? and ??= operators, and the conditional operator ?: are right-associative, meaning that operations are performed from right to left. For example, x = y = z is evaluated as x = (y = z).

Precedence and associativity can be controlled using parentheses. For example, x + y * z first multiplies y by z and then adds the result to x, but (x + y) * z first adds x and y and then multiplies the result by z.

Most operators can be overloaded. Operator overloading permits user-defined operator implementations to be specified for operations where one or both of the operands are of a user-defined class or struct type.

C# provides a number of operators to perform arithmetic, logical, bitwise and shift operations and equality and order comparisons.

Statements

The actions of a program are expressed using statements. C# supports several different kinds of statements, a number of which are defined in terms of embedded statements.

- A ***block*** permits multiple statements to be written in contexts where a single statement is allowed. A block consists of a list of statements written between the delimiters { and }.
- ***Declaration*** statements are used to declare local variables and constants.
- ***Expression*** statements are used to evaluate expressions. Expressions that can be used as statements include method invocations, object allocations using the new operator, assignments using = and the compound assignment operators, increment and decrement operations using the ++ and -- operators and await expressions.

- ***Selection*** statements are used to select one of a number of possible statements for execution based on the value of some expression. This group contains the `if` and `switch` statements.
- ***Iteration*** statements are used to execute repeatedly an embedded statement. This group contains the `while`, `do`, `for`, and `foreach` statements.
- ***Jump*** statements are used to transfer control. This group contains the `break`, `continue`, `goto`, `throw`, `return`, and `yield` statements.
- The ***try***…***catch*** statement is used to catch exceptions that occur during execution of a block, and the ***try***…***finally*** statement is used to specify finalization code that is always executed, whether an exception occurred or not.
- The ***checked*** and ***unchecked*** statements are used to control the overflow-checking context for integral-type arithmetic operations and conversions.
- The ***lock*** statement is used to obtain the mutual-exclusion lock for a given object, execute a statement, and then release the lock.
- The ***using*** statement is used to obtain a resource, execute a statement, and then dispose of that resource.