

# Programming in C#

© 2001, 2002, 2005 Rex Jaeschke. All rights reserved.

Once the reader has read the information provided in each chapter, and solved the related programming exercises, he or she should be able to do the following:

## 1. The Basics

- Understand the use of white space, comments, and blocks, and to identify individual source tokens.
- For user-defined names, know the rules regarding their spelling, case sensitivity, and length of significance.
- Know how to use the basic output facilities to write to the screen.
- Understand the need for escape sequences.
- Use the bool data type.
- Understand the size and value range of the arithmetic data types.
- Know the operators for performing bit operations on integer values.
- Know how to write simple expressions and statements.
- Know how to define and use enumerations and their members.
- Be able to perform basic string operations.
- Know how to write literals and be able to tell their type.
- Know how to define and initialize local variables.
- Understand and apply operator precedence and associativity.
- Know when and how to use implicit and explicit type conversion.
- Understand how integer overflow checking can be enabled and disabled.
- Have a basic understanding on how to break a program into multiple classes, and how to communicate between them.
- Know how and when to use the public and private modifiers.
- Know how to define const variables.
- Know how to use methods in the standard library classes, such as Char, Int32, Double, and Math.

## 2. Looping and Branching

- Use the while, for, and do-while looping constructs.
- Make decisions using if-else and switch.
- Start to appreciate certain aspects of programming style with respect to white space usage and indenting.
- Understand the prefix and postfix increment and decrement operators.
- Know the family of relational and equality operators.
- Understand that assignment is an operator, and the implications thereof, especially with regard to embedded assignment and the potential for confusion with equality.

## Programming in C#: Seminar Goals

- Know about the basic compound-assignment operators.
- Know and understand the remainder operator.
- Know the family of logical operators.
- Understand when and how to use the branching statements break, continue, and goto.
- Understand the purpose of, and problems associated with, the null (or empty) statement.

### 3. Methods

- Know the connection between a method's return type and return statements in that method's definition.
- Know how to pass value, reference, and output arguments.
- Know how to return a value from a method.
- Understand method definitions and their role in allowing the compiler to check method calls for correctness and to cause implicit conversion.
- Have a basic understanding of recursion.
- Be able to overload methods.
- Be able to use the ?: conditional operator.
- Know the purpose of the optional return value from Main.
- Know the difference between a program and an assembly.
- Understand why an application might have multiple assemblies.
- Know when to use the internal access modifier.
- Know how to write and use a generic method.

### 4. References, Strings, and Arrays

- Understand that a reference variable points to an object rather than containing that object's value directly, and that multiple reference variables can point to the same object.
- Understand the purpose of a null reference.
- Know how to pass reference arguments to, and return them from, a method.
- Know how to define and perform basic operations using the type string.
- Know how to allocate memory dynamically using new.
- Have a basic understanding of a reference count and that garbage collection is automatic.
- Define, initialize, and use single dimensional, multi-dimensional, and jagged arrays.
- Understand that each dimension's elements start at index 0 and that each dimension has a read-only length field.
- Know how arrays are stored in memory.
- Know how to deal with command-line arguments passed to Main.
- Understand why System.Array.Copy is needed and how to use it.
- Know how to pass variable-length argument lists.
- Have a basic understanding of the StringBuilder class.

### 5. Classes

- Understand the purpose of data hiding and encapsulation.
- Be able to use effectively the modifiers public and private.

## Programming in C#: Seminar Goals

- Be able to implement a simple version of Equals and ToString for a class.
- Understand constructors and know how to have one call another for the same class.
- Know the value of providing read, write, and read/write properties.
- Know that the way in which an object is represented internally is not necessarily related to the way in which programmers see it externally.
- Know the type of this, and what it refers to.
- Understand the difference between instance and class data.
- Know when a static constructor might be used.
- Know when a private constructor might be used.
- Understand object finalization and resource disposal.
- Be able to define simple user-defined types using object-oriented constructs.
- Know that class definitions can nest.
- Know how to write and use a generic class.

## 6. Inheritance

- Know how and why one would want to use inheritance.
- Understand the difference between containment and inheritance.
- At the design stage, be able to study a set of class descriptions and determine if those classes are unrelated, related by inheritance, or related by containment.
- Know how and when to use the keyword base.
- Understand virtual and override.
- Know how and why classes and methods should be made abstract.
- Know when and how to use the protected access specifier.
- Understand sealed classes and methods.
- Understand the advantage of having Object as the ultimate base class.
- Know how finalization works in a class hierarchy.
- Know when and how to use the is operator.
- Know how to correctly implement an Equals method.
- Understand that array types are derived from Array, enums from Enum, and value types from ValueType.
- Understand boxing and unboxing.
- Have a basic understanding of interfaces.

## 7. Exception Handling

- Understand how the traditional approaches to dealing with extraordinary errors are limited and messy in general and even more so in an object-oriented environment.
- Know how to catch system exceptions using try and catch blocks, and how to recover from such exceptions where possible.
- Be able to throw an exception of a given type.
- Understand that an exception type could be a full-blown class using all the facilities we've learned so far.
- Understand the utility of, and issues relating to, a family of derived exception types.

## 8. Operator Overloading

- Know which operators can and cannot be overloaded.
- Understand when it is and isn't useful to overload operators.
- Understand that a thorough knowledge of a built-in operator must be gained before it can be overloaded in an effective and complete manner.
- Be able to overload most unary and binary operators.

## 9. Delegates and Events

- Understand that a delegate can encapsulate one or more methods, including class and instance methods.
- Understand how to define a delegate.
- Understand delegate type compatibility.
- Know the syntax for combining and removing delegates.
- Know that delegate types are derived from Delegate.
- Have a basic understanding of events.

## 10. Structs

- Know that a struct is a class with a few restrictions and differences.
- Understand that memory for structs is not allocated on the heap.
- Understand that the built-in value types really are synonyms for struct types defined in the standard library.
- Know that the method Main can be defined inside a struct or a class.

## 11. Namespaces

- Understand how namespace pollution is a problem in projects involving multiple subsystems and development groups.
- Know how to import names from namespaces, and how to use names within those namespaces without importing them.
- Be able to define namespaces.

## 12. Input and Output

- Understand the concept of streams.
- Know how to detect exceptions that occur during I/O.
- Be able to perform basic I/O operations to/from the screen, a file, and a string.
- Understand the difference between formatted and unformatted I/O.
- Know that one can randomly move about a file, saving and restoring file positions at will.
- Know that basic operations can be performed on files and directories.

## 13. The Preprocessor

- Understand the purpose of the preprocessor.
- Know how to define conditional compilation symbols.

## Programming in C#: Seminar Goals

- Know how to perform simple and complex tests on one or more conditional compilation symbols.
- Understand the purpose of the #error and #warning directives.