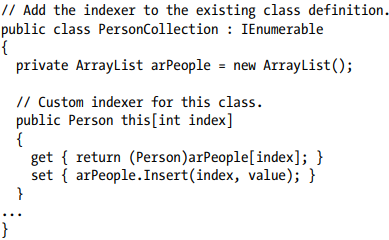
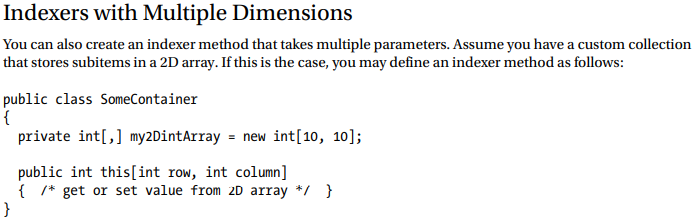
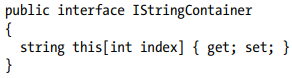
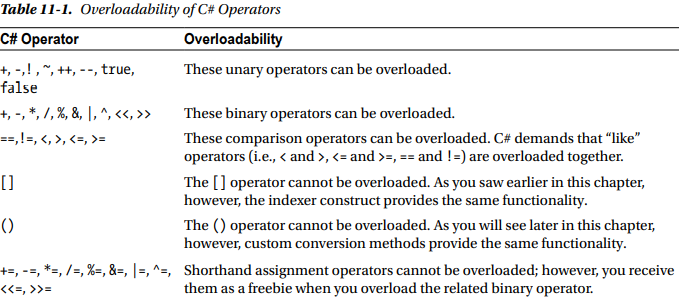
**Advanced C# Language Features**

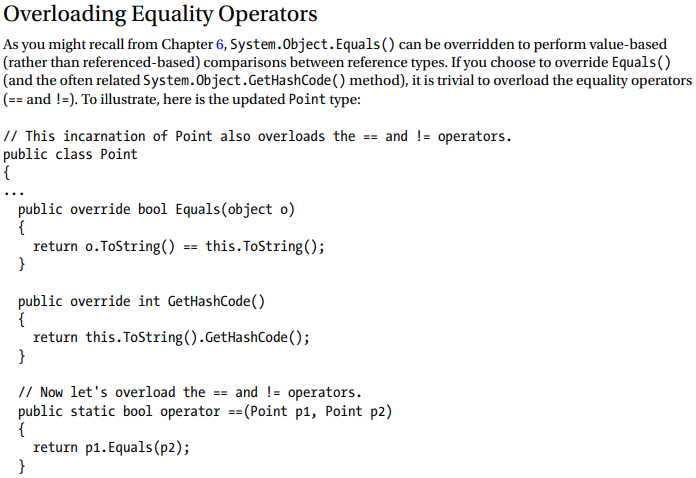
****To wrap things up, you’ll learn how to create an “unsafe” code context to directly manipulate unmanaged pointers. While it is certainly true that using pointers in C# applications is a fairly infrequent activity, understanding how to do so can be helpful in some circumstances that involve complex interoperability scenarios.

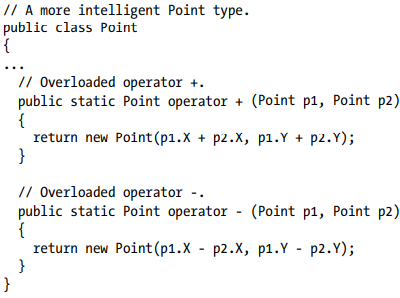
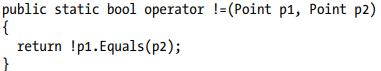
**Understanding Indexer Methods** As a programmer, you are certainly familiar with the process of accessing individual items contained within a simple array using the index operator ([]).

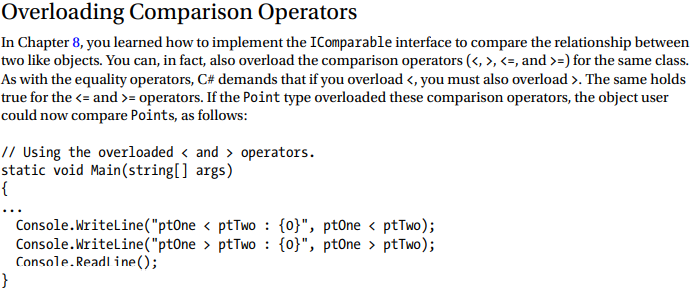


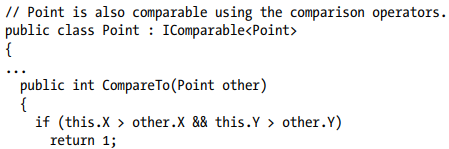
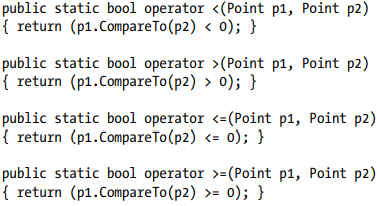


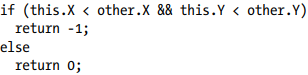
**Understanding Operator Overloading** C#, like any programming language, has a canned set of tokens that are used to perform basic operations on intrinsic types. For example, you know that the + operator can be applied to two integers to yield a larger integer.

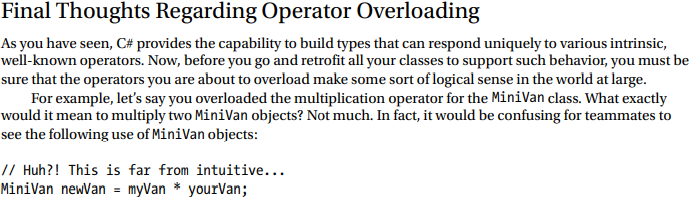


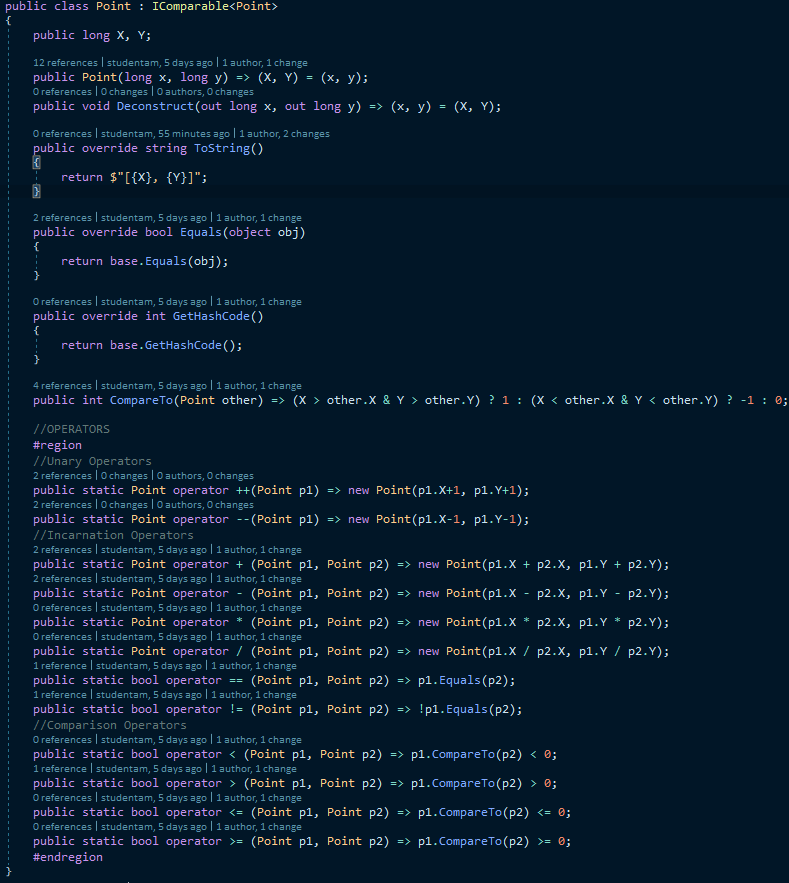




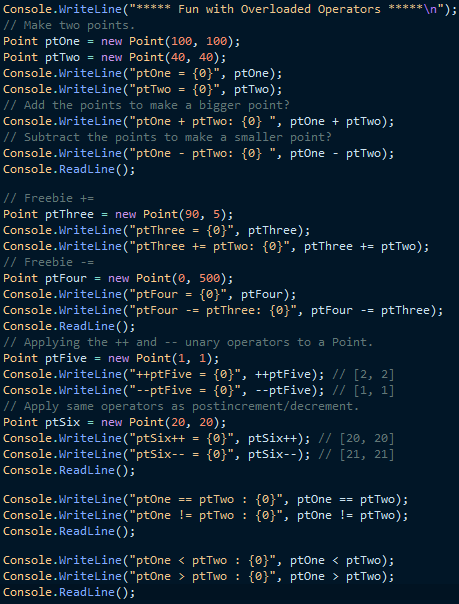


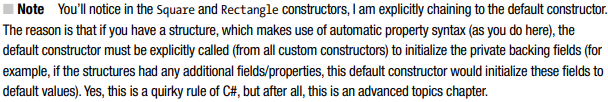






**Understanding Custom Type Conversions**

Let’s now examine a topic closely related to operator overloading: custom type conversions. To set the stage for the discussion, let’s quickly review the notion of explicit and implicit conversions between numerical data and related class types.

**Creating Custom Conversion Routines** Begin by creating a new Console Application project named CustomConversions. C# provides two keywords, explicit and implicit, that you can use to control how your types respond during an attempted conversion. Assume you have the following structure definitions: