可重载的一元运算符：+、-、!、~、++、--、true、false

可重载的二元运算符：+、-、*、/、%、&、|、^、<<、>>、==、!=、>、<、>=、<=

运算符重载不能：

- 创建新运算符；
- 改变运算符的语法；
- 重新定义运算符如何处理预定义类型；
- 改变运算符的优先级或结合性。

递增运算符和递减运算符也可以重载。你可以编写一段代码来对对象进行递增或递减操作：任何对于用户定义类型有意义的操作。

- 在运行时，当你的代码对对象执行前置操作（前置递增或前置递减）时，会发生以下行为：
  - 在对象上执行递增或递减代码；
  - 返回对象。
- 在运行时，当你的代码对对象执行后置操作（后置递增或后置递减）时，会发生以下行为。
  - 如果对象是值类型，则系统会复制该对象；如果对象是引用类型，则引用会被复制。
  - 在对象上执行递增或递减代码。
  - 返回保存的操作数。

如果你的操作数对象是值类型对象，那么一点问题都没有。但是当你的用户定义类型是引用类型时，你就需要小心了。

对于引用类型的对象，前置操作没有问题，因为没有进行复制。但是，对于后置操作，因为保存的副本是引用的副本，所以这意味着原始引用和引用副本指向相同的对象。那么，当进行第二步操作的时候，递增或者递减代码就会在对象上执行。这意味着保存的引用所指向的对象不再是它的起始状态了。返回对变化了的对象的引用可能不是预期行为。

下面的代码说明了将后置递增应用到值类型对象和引用类型对象后的不同。如果运行这段代码两次，第一次 MyType 用户定义类型是结构体，你会得到一个结果。然后，把 MyType 的类型改成类，再次运行，你会得到不同的结果。

```
using static System.Console;

public struct MyType        //运行两次，一次是结构体，一次是类
{
   public int X;
   public MyType( int x )
   {
      X = x;
   }

   public static MyType operator ++( MyType m )
   {
      m.X++;
      return m;
   }
```

```
}

class Test
{
   static void Show( string message, MyType tv )
   {
      WriteLine( $"{message} {tv.X}" );
   }

   static void Main()
   {
      MyType tv = new MyType( 10 );
      WriteLine( "Pre-increment" );
      Show( "Before   ", tv );
      Show( "Returned ", ++tv );
      Show( "After    ", tv );
      WriteLine();

      tv = new MyType( 10 );
      WriteLine( "Post-increment" );
      Show( "Before   ", tv );
      Show( "Returned ", tv++ );
      Show( "After    ", tv );
   }
}
```

当 MyType 是结构体时，运行上面的代码，（如你所期望的）结果如下：

```
Pre-increment
Before   10
Returned 11
After    11

Post-increment
Before   10
Returned 10
After    11
```

当 MyType 是类（即一个引用类型）时，返回对象的后置递增值已经增加了——这可能不是你所希望的。

```
Pre-increment
Before   10
Returned 11
After    11

Post-increment
Before   10
Returned 11 <-- 不是你所预期的
After    11
```