Operators and Casts

Chapter 5

https://csharp.christiannagel.com

Topics

- Operators
- Operator Overloading
- Boxing

Operators

Category	Operator
Primary	x.y x?.y f(x) a[x] x++ x-- x! x->y new typeof default checked unchecked delegate nameof sizeof delegate stackalloc
Unary	+x -x !x ~x ++x --x ^x (T)x await &x *x true false
Range	x..y
Multiplicative	x*y x/y x%y
Additive	x+y x-y
Shift	x<<y x>>y
Relational	x<y x>y x<=y x>=y
Type testing	is as
Equality	x==y x!=y
Logical	x&y x^y x|y
Conditional logical	x&&y x||y
Null coalescing	x??y
Conditional operator	c?t:f
Assignment	x=y x+=y x-=y x*=y x/=y x%=y x&=y x|=y x^=y x<<=y x>>=y x??=y
Lambda expression	=>

Compound Assignment

```
int x = 1;
int x += 2; // shortcut for int x = x + 2;
x++; // shortcut for x = x + 1;
```

Conditional Expression Operator

```
int x = 1;
string s = x + " ";
s += (x == 1 ? "man": "men");
Console.WriteLine(s);
```

checked and unchecked

```
byte b = 255;
checked
{
  b++;
}
Console.WriteLine(b);
```

is and as Operators

```
DerivedClass? derivedClass = baseClass as DerivedClass;
if (derivedClass != null)
{
  // use the derivedClass variable
}
```

```
if (baseClass is DerivedClass derivedClass)
{
  // use the derivedClass variable
}
```

Sizeof Operator

- Returns the number of bytes used
- sizeof(int)
- Can be used with custom struct types with unsafe code

typeof

Returns a Type object

nameof Expression

- Resolved by the compiler
- Returns the name of a type, member, variable…

```
public void Method(object o)
{
  if (o == null) throw new ArgumentNullException(nameof(o));
}
```

Indexer

- Used with arrays, collections

```
Dictionary<string, int> dict = new();
dict["first"] = 1;
int x = dict["first"];
```

Null Coalescing and Null Coalescing Assignment

- ?? and ??=

```
int? a = null;
int b;
b = a ?? 10; // b has the value 10
a = 3;
b = a ?? 10; // b has the value 3
```

```
private MyClass _val;
public MyClass Val
{
  get => _val ??= new MyClass();
}
```

Null-Conditional Operator

- ?.
- ?[]

```
public void ShowPerson(Person? p)
{
  string firstName = p?.FirstName;
```

Type Safety

- Implicit Conversion

```
byte value1 = 10;
byte value2 = 23;
long total = value1 + value2; // this will compile fine
Console.WriteLine(total);
```

- Explicit Conversion

```
long val = 30000;
int i = (int)val;
```

Boxing and Unboxing

- Value type to reference type

```
int myIntNumber = 20;
object myObject = myIntNumber; // Box the int
int mySecondNumber = (int)myObject; // Unbox it back into an int
```

Operator Overloading

```
readonly struct Vector
{
  public Vector(double x, double y, double z) => (X, Y, Z) = (x, y, z);

  public Vector(Vector v) => (X, Y, Z) = (v.X, v.Y, v.Z);

  public readonly double X;
  public readonly double Y;
  public readonly double Z;
  public override string ToString() => $"( {X}, {Y}, {Z} )";
}
```

```
public static Vector operator +(Vector left, Vector right) =>
  new Vector(left.X + right.X, left.Y + right.Y, left.Z + right.Z);
```

Operator Overloading Need to Know

- Conditional logical operators && and || implicitly overloaded with the true, false, & and | operators
- += and -= overloaded by overloading + and –
- If you overload ==, you must overload !=
- if you overload <, you must overload >
- if you overload <=, you must overload >=

Summary

- Operators
- Casting
- Boxing
- Operator Overloading