Ausdrücke

Operatoren und Vorrangregeln

Primary	(x) x.y f(x) a[x] x++ x-- new typeof sizeof checked unchecked
Unary	+ - ~ ! ++x --x (T)x
Multiplicative	* / %
Additive	+ -
Shift	<< >>
Relational	< > <= >= is as
Equality	== !=
Logical AND	&
Logical XOR	^
Logical OR	|
Conditional AND	&&
Conditional OR	||
Conditional	c?x:y
Assignment	= += -= *= /= %= <<= >>= &= ^= |=

Operatoren der gleichen Zeile werden von links nach rechts ausgewertet

Arithmetische Ausdrücke

Operandentypen

- numerisch oder char
- bei ++ und -- numerisch oder enum (funktioniert auch bei float und double!)

Ergebnistyp

Kleinster numerischer Typ, der beide Operandentypen einschließt, aber zumindest int.

Ausnahmen

- uint => long
- ulong => verboten

uint	• (sbyte | short | int)	=> long
ulong	• (sbyte | short | int | long)	=> verboten
decimal	• (float | double)	=> verboten

Vergleichsausdrücke

Operandentypen

- bei <, >, <=, >=: numerisch, char, enum
- bei ==, !=: numerisch, char, enum, bool, Referenzen
- bei x is T: x: Ausdruck mit beliebigem Typ, T: Referenztyp

  z.B.: obj is Rectangle
  objOfValueType is IComparable
  3 is object
  arr is int[]

Ergebnistyp

bool

Boolesche Ausdrücke (&&, ||, !)

Operandentypen

bool

Ergebnistyp

bool

Kurzschlußauswertung (bedingte Auswertung)

```
a && b   =>   if (a) b else false
a || b   =>   if (a) true else b
```

Nützlich bei

```
if (p != null && p.val > 0) ...
if (x == 0 || y / x > 2) ...
```

Bit-Ausdrücke (&, |, ^, ~)

Operandentypen

- bei & | ^ : numerisch, char, enum, bool
- bei ~ : numerisch, char, enum
- bei unterschiedlich großen Operandentypen, werden beide Operanden in den größeren Typ konvertiert

Ergebnistyp

- größter der beiden Operandentypen
- bei numerischen Typen und char zumindest int

Shift-Ausdrücke

Operandentypen für x << y und x >> y

- x: ganzzahlig oder char
- y: int

Ergebnistyp

Typ von x, aber zumindest int

Bemerkung

>> führt bei vorzeichenlosen Typen ein logisches Shift, sonst ein arithmetisches Shift durch

Überlaufprüfungen

Normalerweise wird Überlauf nicht erkannt

```
int x = 1000000;
x = x * x;   // -727379968, kein Fehler
```

Überlaufprüfung

```
x = checked(x * x);  // liefert System.OverflowException

checked {
    ...
    x = x * x;       // liefert System.OverflowException
    ...
}
```

Es gibt auch Compiler-Option, um Überlaufprüfung generell einzuschalten

```
csc /checked Test.cs
```

Diverses

typeof

- liefert *Type*-Objekt zu einem Typ
  (Type-Objekt eines Objekts *o* kann mit *o.GetType()* abgefragt werden).

```
Type t = typeof(int);
Console.WriteLine(t.Name);     // liefert Int32
```

sizeof

- gibt die Größe eines Typs zurück
- kann nur auf Werttypen angewendet werden
- kann nur im unsafe-Kontext benutzt werden (unportable oder gefährliche Konstrukte)
  Muß übersetzt werden mit csc /unsafe xxx.cs

```
unsafe {
    Console.WriteLine(sizeof(int));
    Console.WriteLine(sizeof(MyEnumType));
    Console.WriteLine(sizeof(MyStructType));
}
```