Перегрузка операторов

В языке C# допускается определять назначение оператора по отношению к создаваемому классу. Этот процесс называется **перегрузкой операторов**.

Для перегрузки оператора служит ключевое слово **operator**, определяющее **операторный метод**, который, в свою очередь, определяет действие оператора относительно своего класса.

```
// Операция + с целыми.
int a = 100;
int b = 240;
int c = a + b;//c теперь равно 340

// Операция + со строками.
string si = "Hello";
string s2 = " world!";
string s3 = si + s2;// s3 теперь содержит "Hello world!"
```

Операторы, допускающие перегрузку

Операторы	Категория операторов
-	Изменение знака переменной
!	Операция логического отрицания
~	Операция побитового дополнения, которая приводит к инверсии каждого бита
++, --	Инкремент и декремент
true, false	Критерий истинности объекта, определяется разработчиком класса
+, –, *, /, %	Арифметические операторы
&, |, ^, <<, >>	Битовые операции
==, !=, <, >, <=, >=	Операторы сравнения
&&, ||	Логические операторы
[]	Операции доступа к элементам массивов моделируются за счет индексаторов
()	Операции преобразования

Операторы, не допускающие перегрузку

Операторы	Категория операторов
+=, -=, *=, /=, %=, &=, |=, ^=, <<=, >>=	Перегружаются автоматически при перегрузке соответствующих бинарных операций
=	Присвоение
.	Доступ к членам типа
?:	Оператора условия
new	Создание объекта
as, is, typeof	Используются для получения информации о типе
->, sizeof,*,&	Доступны только в небезопасном коде

```
// Общая форма перегрузки унарного оператора.
public static возвр_тип operator op(тип операнд)
{
// операции
}

// Общая форма перегрузки бинарного оператора.
public static возвр_тип operator op(тип операнд1, тип операнд2)
{
// операции
}
```

Некоторые особенности:

- Тип операнда унарных операторов должен быть таким же, как и у класса, для которого перегружается оператор.
- В бинарных операторах хотя бы один из операндов должен быть такого же типа, как и у его класса.
- В параметрах оператора нельзя использовать модификатор ref или out.

Перегрузка бинарных операторов

```
class CPoint
    {
        private int x;
        private int y;
        public CPoint(int x, int y)
        {
            this.x = x; this.y = y;
        }

        //перегрузка бинарного оператора + (сложение точек)
        public static CPoint operator +(CPoint A, CPoint B)
        {
            CPoint C = new CPoint(0, 0);
            C.x = A.x + B.x;
            C.y = A.y + B.y;
            return C;
}}
```

```
CPoint p1 = new CPoint(10,10);
CPoint p2 = new CPoint(15,5);
CPoint p3 = p1 + p2;
```

Перегрузка бинарных операторов

```
internal class Complex
{
   public static Complex operator +(Complex c1, Complex c2)
   {   // тело перегруженного оператора
       return new Complex();
   }
}
```

Компилятор генерирует определение метода **op_Addition** и устанавливает в записи с определением этого метода флаг **specialname**, свидетельствующий о том, что это особый метод.

Когда компилятор видит в коде использование оператора «+», то он исследует типы его операндов, и пытается выяснить, не определен ли для одного из них метод **op_Addition** с флагом **specialname**. Если такой метод существует, компилятор генерирует код, вызывающий этот метод, иначе возникает ошибка компиляции

Унарные операторы C# и CLS-совместимые имена соответствующих методов

Оператор C#	Имя специального метода	Рекомендуемое CLS-совместимое имя метода
+	op UnaryPlus	Plus
–	op UnaryNegation	Negate
!	op LogicalNot	Not
~	op OnesComplement	OnesComplement
++	op Increment	Increment
--	op Decrement	Decrement
Нет	op True	IsTrue { get; }
Нет	op False	IsFalse { get; }

Бинарные операторы и их CLS-совместим ые имена методов

Оператор C#	Имя специального метода	Рекомендуемое CLS-совместимое имя метода
+	op_Addition	Add
–	op_Subtraction	Subtract
*	op_Multiply	Multiply
/	op_Division	Divide
%	op_Modulus	Mod
&	op_BitwiseAnd	BitwiseAnd
|	op_BitwiseOr	BitwiseOr
^	op_ExclusiveOr	Xor
<<	op_LeftShift	LeftShift
>>	op_RightShift	RightShift
==	op_Equality	Equals
!=	op_Inequality	Compare
<	op_LessThan	Compare
>	op_GreaterThan	Compare
<=	op_LessThanOrEqual	Compare
>=	op_GreaterThanOrEqual	Compare

Выполнение операций со встроенными в C# типами данных

```
//перегрузка бинарного оператора + (сложение точки и числа)
public static CPoint operator +(CPoint A, int X)
{
    CPoint C = new CPoint(0, 0);
    C.x = A.x + X;
    C.y = A.y + X;
    return C;
}

//перегрузка бинарного оператора + (сложение числа и точки)
public static CPoint operator +(int X, CPoint A)
{
    CPoint C = new CPoint(0, 0);
    C.x = A.x + X;
    C.y = A.y + X;
    return C;
}
```

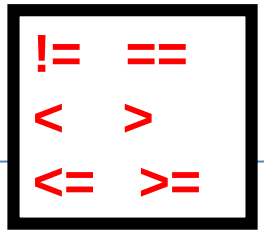
```
CPoint p5 = p1 + 10;  // сложение объекта класса CPoint с числом
CPoint p6 = 10 + p2;
```

```
//перегрузка инкремента
public static CPoint operator ++(CPoint s)
{
    CPoint p = new CPoint(s.x, s.y);
    p.x++;
    p.y++;
    return p;
}
//перегрузка декремента
public static CPoint operator --(CPoint s)
{
    CPoint p = new CPoint(s.x, s.y);
    p.x--;
    p.y--;
    return p;
}
//перегрузка оператора - (изменение знака)
public static CPoint operator -(CPoint s)
{
    CPoint p = new CPoint(s.x, s.y);
    p.x = -p.x;
    p.y = -p.y;
    return p;
}
```

Перегрузка операторов отношения

Перегруженный оператор отношения возвращает логическое значение true и false.

На перегрузку операторов отношения накладывается важное ограничение: **они должны перегружаться попарно**




```
public static bool operator <(CPoint op1, CPoint op2)
{
   if(Math.Sqrt(op1.x * op1.x + op1.y * op1.y ) <
      Math.Sqrt(op2.x * op2.x + op2.y * op2.y ))
      return true;
    else
      return false;
}
// Перегрузить оператор >.
public static bool operator >(CPoint op1, CPoint op2)
{
   if(Math.Sqrt(op1.x * op1.x + op1.y * op1.y) >
      Math.Sqrt(op2.x * op2.x + op2.y * op2.y))
    return true;
   else
    return false;
}
```

Перегрузка операторов true и false

- Операторы true и false должны **перегружаться попарно**, а не раздельно.
- После перегрузки этих ключевых слов в качестве унарных операторов для конкретного класса появляется возможность использовать объекты этого класса для управления операторами **if, while, for и do-while или же в условном выражении ?.**

```
// Перегружаем оператор false
    public static bool operator false(CPoint obj)
    {
        if ((obj.x <= 0) || (obj.y <= 0))
            return true;
        return false;
    }

    // Обязательно перегружаем оператор true
    public static bool operator true(CPoint obj)
    {
        if ((obj.x > 0) && (obj.y > 0))
            return true;
        return false;
    }
```

```
CPoint p5 = new CPoint(10,10);
if (p5)
  Console.WriteLine("Все координаты объекта p5 положительны");
```

Перегрузка логических операторов

перегрузке подлежат только операторы &, | и !

```
//перегрузка бинарного оператора & (оператор )
public static bool  operator &(CPoint A, CPoint B)
{
    if(A.x==B.x && A.y==B.y)
      return true;
    else
      return false;
}

//перегрузка бинарного оператора |
public static bool operator |(CPoint A, CPoint B)
{
    if (A.x == B.x || A.y == B.y)
      return true;
    else
      return false;
}
```

Перегрузка логических операторов (|| и &&)

Для того, чтобы применение укороченных логических операторов && и || стало возможным, необходимо:

- в классе должна быть произведена перегрузка логических операторов & и |.
- перегружаемые методы операторов & и | должны возвращать значение того же типа, что и у класса, для которого эти операторы перегружаются.
- каждый параметр должен содержать ссылку на объект того класса, для которого перегружается логический оператор.
- для класса должны быть перегружены операторы true и false.

Если все эти условия выполняются, то укороченные логические операторы автоматически становятся пригодными для применения.

Оператор явного преобразования типа (explicit)

```
public static explicit operator Digit(byte argument)
{
    Digit digit = new Digit(argument);
    return digit;
}
```

```
class MainClass
{
    static void Main()
    {
        byte variable = 1;

        // Явное преобразование byte-to-Digit.
        Digit digit = (Digit)variable;

        }
}
```

Ключевое слово explicit служит для создания оператора явного преобразования типа.

Оператор неявного преобразования типа (implicit)

```
public static implicit operator Digit(byte argument)
{
    Digit digit = new Digit(argument);
    return digit;
}
```

```
class MainClass
{
    static void Main()
    {
        byte variable = 1;

        // Неявное преобразование byte-to-Digit.
        Digit digit = variable;

        }
}
```

Ключевое слово implicit служит для создания оператора неявного преобразования типа.