Существуют две формы операторных методов (operator): одна — для унарных

операторов, другая — для бинарных. Ниже приведена общая форма для каждой разновидности

этих методов.

***// Общая форма перегрузки унарного оператора.***

***public static возвращаемый\_тип operator op(тип\_параметра операнд)***

***{***

***// операции***

***}***

***// Общая форма перегрузки бинарного оператора.***

***public static возвращаемый\_тип operator op(тип\_параметра1 операнд1,***

***тип\_параметра1 операнд2)***

***{***

***// операции***

***}***

Здесь вместо *ор* подставляется перегружаемый оператор, например + или /; а *возвращаемый\_*

*тип* обозначает конкретный тип значения, возвращаемого указанной операцией.

Это значение может быть любого типа, но зачастую оно указывается такого же

типа, как и у класса, для которого перегружается оператор. Такая корреляция упрощает

применение перегружаемых операторов в выражениях. Для унарных операторов

*операнд* обозначает передаваемый операнд, а для бинарных операторов то же самое

обозначают *операнд1* и *операнд2.* Обратите внимание на то, что операторные методы

***должны*** иметь оба типа, ***public*** и ***static***.

Тип операнда унарных операторов должен быть таким же, как и у класса, для которого

перегружается оператор. А в бинарных операторах хотя бы один из операндов

должен быть такого же типа, как и у его класса. Следовательно, в C# не допускается

перегрузка любых операторов для объектов, которые еще не были созданы. Например,

назначение оператора + нельзя переопределить для элементов типа int или string.

И еще одно замечание: в параметрах оператора нельзя использовать модификатор

ref или out.

**Перегрузка бинарных операторов**

Для того чтобы продемонстрировать принцип действия перегрузки операторов,

начнем с простого примера, в котором перегружаются два оператора — + и -. В приведенной

ниже программе создается класс ThreeD, содержащий координаты объекта

в трехмерном пространстве. Перегружаемый оператор + складывает отдельные координаты

одного объекта типа ThreeD с координатами другого. А перегружаемый оператор

- вычитает координаты одного объекта из координат другого. (***glava9\_1***)

class ThreeD

{

int x, y, z;

public ThreeD() { x = y = z = 0; }

public ThreeD(int i, int j, int k) { x = i; y = j; z = k; }

//overload operator +

public static ThreeD operator +(ThreeD op1, ThreeD op2)

{

ThreeD result = new ThreeD();

result.x = op1.x + op2.x;

result.y = op1.y + op2.y;

result.z = op1.z + op2.z;

return result;

}

//overload operator -

public static ThreeD operator -(ThreeD op1, ThreeD op2)

{

ThreeD result = new ThreeD();

result.x = op1.x - op2.x;

result.y = op1.y - op2.y;

result.z = op1.z - op2.z;

return result;

}

}

class ThreeDDemo

{

static void Main()

{

ThreeD a = new ThreeD(1,2,3);

ThreeD b = new ThreeD(10,10,10);

ThreeD c;

Console.Write("Coordinates of a: ");

a.Show();

Console.WriteLine();

Console.Write("Coordinates of b: ");

b.Show();

Console.WriteLine();

c = a + b;

Console.Write("Result of a + b: ");

c.Show();

Console.WriteLine();

c = a + b + c;

Console.Write("Result of a + b + c: ");

c.Show();

Console.WriteLine();

c = c - a;

Console.Write("Result of c - a: ");

c.Show();

Console.WriteLine();

c = c - b;

Console.Write("Result of c - b: ");

c.Show();

Console.WriteLine();

}

}

Обратите внимание на то, что метод operator+() возвращает объект типа ThreeD.

Этот метод мог бы возвратить значение любого допустимого в C# типа, но благодаря

тому что он возвращает объект типа ThreeD, оператор + можно использовать в таких

составных выражениях, как a+b+с. В данном случае выражение а+b дает результат

типа ThreeD, который можно затем сложить с объектом с того же типа. Если бы

выражение а+b давало результат другого типа, то вычислить составное выражение

a+b+с было бы просто невозможно.

А теперь проанализируем операторный метод operator-(). Оператор - действует

так же, как и оператор +, но для него важен порядок следования операндов. Напомним,

что сложение носит коммутативный характер (от перестановки слагаемых сумма

не меняется), чего нельзя сказать о вычитании: А - В не то же самое, что и В - А! Для

всех двоичных операторов первым параметром операторного метода является левый

операнд, а вторым параметром — правый операнд. Поэтому, реализуя перегружаемые

варианты некоммутативных операторов, следует помнить, какой именно операнд

должен быть указан слева и какой — справа.

**Перегрузка унарных операторов**

Унарные операторы перегружаются таким же образом, как и бинарные. Главное

отличие заключается, конечно, в том, что у них имеется лишь один операнд. В качестве

примера ниже приведен метод, перегружающий оператор унарного минуса для

класса ThreeD.

***// Перегрузить оператор унарного минуса.***

***public static ThreeD operator - (ThreeD op)***

***{***

***ThreeD result = new ThreeD ();***

***result.x = -op.x;***

***result.у = -op.у;***

***result.z = -op.z;***

***return result;***

***}***

В данном примере создается новый объект, в полях которого сохраняются отрицательные

значения операнда перегружаемого унарного оператора, после чего этот объект

возвращается операторным методом. Обратите внимание на то, что сам операнд не

меняется. Это означает, что и в данном случае обычное назначение оператора унарного

минуса сохраняется. Например, результатом выражения

***а = -b***

является отрицательное значение операнда b, но сам операнд b не меняется.

В C# перегрузка операторов ++ и -- осуществляется довольно просто. Для этого

достаточно возвратить инкрементированное или декрементированное значение, но не

изменять вызывающий объект. А все остальное возьмет на себя компилятор С#, различая

префиксные и постфиксные формы этих операторов. В качестве примера ниже

приведен операторный метод operator++() для класса ThreeD.

// Перегрузить унарный оператор ++.

public static ThreeD operator ++(ThreeD op)

{

ThreeD result = new ThreeD();

// Возвратить результат инкрементирования.

result.x = op.x + 1;

result.у = op.у + 1;

result.z = op.z + 1;

return result;

}

Ниже приведен расширенный вариант предыдущего примера программы, в кото

ром демонстрируется перегрузка унарных операторов - и ++. (***glava9\_1.1***)

class ThreeD

{

int x, y, z;

…

//overload unary operator

public static ThreeD operator -(ThreeD op)

{

ThreeD result = new ThreeD();

result.x = -op.x;

result.y = -op.y;

result.z = -op.z;

return result;

}

//overload unary operator++

public static ThreeD operator ++(ThreeD op)

{

ThreeD result = new ThreeD();

result.x = +op.x + 1;

result.y = +op.y + 1;

result.z = +op.z + 1;

return result;

}

}

class ThreeDDemo

{

static void Main()

{

…

c = -a;

Console.Write("Result of -a: ");

c.Show();

Console.WriteLine();

c = a++;

Console.WriteLine("If c = a++: ");

Console.Write("then coordinates of c = ");

c.Show();

Console.Write("And coordinates of a = ");

a.Show();

//set default coordinated (1, 2, 3) of a

a = new ThreeD(1, 2, 3);

Console.Write("Default coordinates for a: ");

a.Show();

c = ++a;

Console.WriteLine("If c = ++a: ");

Console.Write("then coordinates of c = ");

c.Show();

Console.Write("And coordinates of a = ");

a.Show();

}}

**Выполнение операций со встроенными в C# типами данных**

Для любого заданного класса и оператора имеется также возможность перегрузить

сам операторный метод. Это, в частности, требуется для того, чтобы разрешить операции

с типом класса и другими типами данных, в том числе и встроенными.

Короче говоря, чтобы можно было сложить не только члены другого класса, но и так же стандартные типы ing, double etc. (***glava9\_1.2***)

class ThreeD

{

int x, y, z;

…

//overload operator+ for object and type

public static ThreeD operator +(ThreeD op1, int op2)//тут складывается с int

{

ThreeD result = new ThreeD();

result.x = op1.x + op2;

result.y = op1.y + op2;

result.z = op1.z + op2;

return result;

…

}

class ThreeDDemo

{

static void Main()

{

ThreeD a = new ThreeD(1, 2, 3);

ThreeD b = new ThreeD(10, 10, 10);

ThreeD c = new ThreeD();

Console.Write("Coordinates of a: ");

a.Show();

Console.WriteLine();

Console.Write("Coordinates of b: ");

b.Show();

Console.WriteLine();

c = a + b;

Console.Write("Result of a + b: ");

c.Show();

Console.WriteLine();

c = b + 10; //тут складывается с int

Console.Write("Result of b + 10: ");

c.Show();

Console.WriteLine();

}

}

Как подтверждает приведенный выше результат, когда оператор + применяется к

двум объектам класса ThreeD, то складываются их координаты. А когда он применяется

к объекту типа ThreeD и целому значению, то координаты этого объекта увеличиваются

на заданное целое значение.

Продемонстрированная выше перегрузка оператора +, безусловно, расширяет полезные

функции класса ThreeD, тем не менее, она делает это не до конца. И вот почему.

Метод operator+(ThreeD, int) позволяет выполнять операции, подобные

следующей.

***ob1 = оb2 + 10;***

Но, к сожалению, он не позволяет выполнять операции, аналогичные следующей.

***ob1 = 10 + оb2;***

Дело в том, что второй целочисленный аргумент данного метода обозначает правый

операнд бинарного оператора +, но в приведенной выше строке кода целочисленный

аргумент указывается слева. Для того чтобы разрешить выполнение такой операции

сложения, придется перегрузить оператор + еще раз. В этом случае первый параметр

операторного метода должен иметь тип int, а второй параметр — тип ThreeD.

public static ThreeD operator +(int op1, ThreeD op2)

{

ThreeD result = new ThreeD();

result.x = op2.x + op1;

result.y = op2.y + op1;

result.z = op2.z + op1;

return result;

}

public static ThreeD operator +(ThreeD op1, int op2)

{

ThreeD result = new ThreeD();

result.x = op1.x + op2;

result.y = op1.y + op2;

result.z = op1.z + op2;

return result;

}

**Перегрузка операторов отношения**

Операторы отношения, например == и <, могут также перегружаться, причем

очень просто. Как правило, перегруженный оператор отношения возвращает логическое

значение true и false. Это вполне соответствует правилам обычного

применения подобных операторов и дает возможность использовать их перегружаемые

разновидности в условных выражениях. Если же возвращается результат

другого типа, то тем самым сильно ограничивается применимость операторов отношения.

Ниже приведен очередной вариант класса ThreeD, в котором перегружаются

операторы < и >. В данном примере эти операторы служат для сравнения объектов

ThreeD, исходя из их расстояния до начала координат. Один объект считается больше

другого, если он находится дальше от начала координат. А кроме того, один объект

считается меньше другого, если он находится ближе к началу координат. Такой вариант

реализации позволяет, в частности, определить, какая из двух заданных точек находится

на большей сфере. Если же ни один из операторов не возвращает логическое

значение true, то обе точки находятся на одной и той же сфере. Разумеется, возможны

и другие алгоритмы упорядочения. (***glava9\_2***)

//overload operator<

public static bool operator <(ThreeD op1, ThreeD op2)

{

if (Math.Sqrt(op1.x \* op1.x + op1.y \* op1.y + op1.z \* op1.z)

< Math.Sqrt(op2.x \* op2.x + op2.y \* op2.y + op2.z \* op2.z))

return true;

else

return false;

}

//overload operator >

public static bool operator >(ThreeD op1, ThreeD op2)

{

if (Math.Sqrt(op1.x \* op1.x + op1.y \* op1.y + op1.z \* op1.z)

> Math.Sqrt(op2.x \* op2.x + op2.y \* op2.y + op2.z \* op2.z))

return true;

else

return false;

}

class ThreeDDemo

{

static void Main()

{

ThreeD a = new ThreeD(5, 6, 7);

ThreeD b = new ThreeD(10, 10, 10);

ThreeD c = new ThreeD(1, 2, 3);

ThreeD d = new ThreeD(6, 7, 5);

Console.Write("Coordinates of a: ");

a.Show();

Console.WriteLine();

Console.Write("Coordinates of b: ");

b.Show();

Console.WriteLine();

Console.Write("Coordinates of c: ");

c.Show();

Console.WriteLine();

Console.Write("Coordinates of d: ");

d.Show();

Console.WriteLine();

if (a > c) Console.WriteLine("a > c true");

if (a < c) Console.WriteLine("a < c true");

if (a > b) Console.WriteLine("a > b true");

if (a < b) Console.WriteLine("a < b true");

if (a > d) Console.WriteLine("a > d true");

else if(a < d) Console.WriteLine("a < d true");

else Console.WriteLine("a and d equals");

}

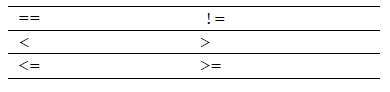
}

На перегрузку операторов отношения накладывается следующее важное ограничение:

***они должны перегружаться попарно***. Так, если перегружается оператор <, то

***следует*** перегрузить и оператор >, и наоборот. Ниже приведены составленные в пары

перегружаемые операторы отношения.



И еще одно замечание: если перегружаются операторы == и !=, то для этого

обычно требуется также переопределить методы Object.Equals() и Object.

GetHashCode(). Эти методы и способы их переопределения подробнее рассматриваются

в главе 11.

**Перегрузка операторов true и false**

Ключевые слова true и false можно также использовать в качестве унарных операторов

для целей перегрузки. Перегружаемые варианты этих операторов позволяют

определить назначение ключевых слов true и false специально для создаваемых классов.

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 ?.

Операторы true и false ***должны*** перегружаться попарно, а не раздельно. Ниже

приведена общая форма перегрузки этих унарных операторов.

***public static bool operator true (тип\_параметра операнд)***

***{***

***// Возврат логического значения true или false.***

***}***

***public static bool operator false (тип\_параметра операнд)***

***{***

***// Возврат логического значения true или false.***

***}***

Обратите внимание на то, что и в том и в другом случае возвращается результат

типа bool.

Ниже приведен пример программы, демонстрирующий реализацию операторов

true и false в классе ThreeD. В каждом из этих операторов проверяется следующее

условие: если хотя бы одна из координат объекта типа ThreeD равна нулю, то этот

объект истинен, а если все три его координаты равны нулю, то такой объект ложен.

В данном примере программы реализован также оператор декремента исключительно

в целях демонстрации. (***glava9\_3***)

class ThreeD

{

int x, y, z;

public ThreeD() { x = y = z = 0; }

public ThreeD(int i, int j, int k) { x = i; y = j; z = k; }

//overload operator true

public static bool operator true(ThreeD op)

{

if ((op.x != 0) || (op.y != 0) || (op.z != 0))

return true; //at least one of the coordinates == 0

else

return false;

}

//overload operator false

public static bool operator false(ThreeD op)

{

if ((op.x == 0) || (op.y == 0) || (op.z == 0))

return true; //all coordinates == 0

else

return false;

}

//overload operator --

public static ThreeD operator --(ThreeD op)

{

ThreeD result = new ThreeD();

result.x = op.x - 1;

result.y = op.y - 1;

result.z = op.z - 1;

return result;

}

}

class ThreeDDemo

{

static void Main()

{

ThreeD a = new ThreeD(5, 0, 7);

ThreeD b = new ThreeD(10, 10, 10);

ThreeD c = new ThreeD(0, 0, 0);

Console.Write("Coordinates of a: ");

a.Show();

Console.Write("Coordinates of b: ");

b.Show();

Console.Write("Coordinates of c: ");

c.Show();

Console.WriteLine();

if (a) Console.WriteLine("a is true");

else Console.WriteLine("a is false");

if (b) Console.WriteLine("b is true");

else Console.WriteLine("b is false");

if (c) Console.WriteLine("c is true");

else Console.WriteLine("c is false");

Console.WriteLine();

Console.WriteLine("Using cycle with object ThreeD.");

do

{

b.Show();

b--;

} while (b);

}

}

Обратите внимание на то, как объекты класса ThreeD используются для управления

условным оператором if и оператором цикла do-while. Так, в операторах if

объект типа ThreeD проверяется с помощью оператора true. Если результат этой

проверки оказывается истинным, то оператор if выполняется. А в операторе цикла

do-while объект b декрементируется на каждом шаге цикла. Следовательно, цикл

повторяется до тех пор, пока проверка объекта b дает истинный результат, т.е. этот

объект содержит хотя бы одну ненулевую координату. Если же окажется, что объект

b содержит все нулевые координаты, его проверка с помощью оператора true даст

ложный результат и цикл завершится.

**Перегрузка логических операторов**

Как вам должно быть уже известно, в C# предусмотрены следующие логические

операторы: &, |, !, && и ||. Из них перегрузке, безусловно, подлежат только операторы

&, | и !. Тем не менее, соблюдая определенные правила, можно извлечь также

пользу из укороченных логических операторов && и ||. Все эти возможности рассматриваются

ниже.

Рассмотрим сначала простейший случай. Если не пользоваться укороченными логическими

операторами, то перегрузку операторов & и | можно выполнять совершенно

естественным путем, получая в каждом случае результат типа bool. Аналогичный

результат, как правило, дает и перегружаемый оператор !.

Ниже приведен пример программы, в которой демонстрируется перегрузка логических

операторов !, & и | для объектов типа ThreeD. Как и в предыдущем примере,

объект типа ThreeD считается истинным, если хотя бы одна из его координат не равна

нулю. Если же все три координаты объекта равны нулю, то он считается ложным. (***glava9\_4***)

class ThreeD

{

int x, y, z;

public ThreeD() { x = y = z = 0; }

public ThreeD(int i, int j, int k) { x = i; y = j; z = k; }

//overload logic operator |

public static bool operator |(ThreeD op1, ThreeD op2)

{

if (((op1.x != 0) || (op1.y != 0) || (op1.z != 0)) |

((op2.x != 0) || (op2.y != 0) || (op2.z != 0)))

return true;

else

return false;

}

//overload logic operator &

public static bool operator &(ThreeD op1, ThreeD op2)

{

if (((op1.x != 0) && (op1.y != 0) && (op1.z != 0)) &

((op2.x != 0) && (op2.y != 0) && (op2.z != 0)))

return true;

else

return false;

}

//overload logic operator !

public static bool operator !(ThreeD op)

{

if ((op.x != 0) || (op.y != 0) || (op.z != 0))

return false;

else

return true;

}

}

class ThreeDDemo

{

static void Main()

{

ThreeD a = new ThreeD(5, 6, 7);

ThreeD b = new ThreeD(10, 10, 10);

ThreeD c = new ThreeD(0, 0, 0);

Console.Write("Coordinates of a: ");

a.Show();

Console.Write("Coordinates of b: ");

b.Show();

Console.Write("Coordinates of c: ");

c.Show();

Console.WriteLine();

if (!a) Console.WriteLine("a is false");

if (!b) Console.WriteLine("b is false");

if (!c) Console.WriteLine("c is false");

Console.WriteLine();

if (a & b) Console.WriteLine("a & b is true");

else Console.WriteLine("a & b is false");

if (a & c) Console.WriteLine("a & c is true");

else Console.WriteLine("a & c is false");

if (a | b) Console.WriteLine("a | b is true");

else Console.WriteLine("a | b is false");

if (a | c) Console.WriteLine("a | c is true");

else Console.WriteLine("a | c is false");

}

}

При таком способе перегрузки логических операторов &, | и ! методы каждого из

них возвращают результат типа bool. Это необходимо для того, чтобы использовать

рассматриваемые операторы обычным образом, т.е. в тех выражениях, где предполагается

результат типа bool. Напомним, что для ***всех*** встроенных в C# типов данных

результатом логической операции должно быть значение типа bool. Поэтому вполне

разумно предусмотреть возврат значения типа bool и в перегружаемых вариантах

этих логических операторов. Но, к сожалению, такой способ перегрузки пригоден

лишь в том случае, если не требуются укороченные логические операторы.

**Как сделать укороченные логические операторы**

**доступными для применения**

Для того чтобы применение укороченных логических операторов && и || стало

возможным, необходимо соблюсти следующие четыре правила. ***Во-первых***, в классе

должна быть произведена перегрузка логических операторов & и |. ***Во-вторых,*** перегружаемые

методы операторов & и | должны возвращать значение того же типа, что

и у класса, для которого эти операторы перегружаются***. В-третьих***, каждый параметр

должен содержать ссылку на объект того класса, для которого перегружается логический

оператор. И ***в-четвертых***, для класса должны быть перегружены операторы true

и false. Если все эти условия выполняются, то укороченные логические операторы

автоматически становятся пригодными для применения.

В приведенном ниже примере программы показано, как правильно реализовать

логические операторы & и | в классе ThreeD, чтобы сделать доступными для применения

укороченные логические операторы && и ||. (***glava9\_5***)

public static ThreeD operator |(ThreeD op1, ThreeD op2)

{

if (((op1.x != 0) || (op1.y != 0) || (op1.z != 0)) |

((op2.x != 0) || (op2.y != 0) || (op2.z != 0)))

return new ThreeD(1, 1, 1);

else

return new ThreeD(0, 0, 0);

}

//overload logic operator &

public static ThreeD operator &(ThreeD op1, ThreeD op2)

{

if (((op1.x != 0) && (op1.y != 0) && (op1.z != 0)) &

((op2.x != 0) && (op2.y != 0) && (op2.z != 0)))

return new ThreeD(1, 1, 1);

else

return new ThreeD(0, 0, 0);

}

Прежде всего обратите внимание на то, что методы обоих перегружаемых логических

операторов теперь ***возвращают*** объект типа ***ThreeD***. И особенно обратите внимание

на то, как формируется этот объект. Если логическая операция дает истинный

результат, то создается и возвращается истинный объект типа ThreeD, у которого хотя

бы одна координата не равна нулю. Если же логическая операция дает ложный результат,

то соответственно создается и возвращается ложный объект.

//and now short logic operatos

Console.WriteLine("Short logic operators && and ||");

if (a && b) Console.WriteLine("a & b is true");

else Console.WriteLine("a & b is false");

if (a && c) Console.WriteLine("a & c is true");

else Console.WriteLine("a & c is false");

if (a || b) Console.WriteLine("a | b is true");

else Console.WriteLine("a | b is false");

if (a || c) Console.WriteLine("a | c is true");

else Console.WriteLine("a | c is false");

Своими словами.

Когда выполняются укороченные логические операторы, они при проверке объектов, определяют явдяются ли истиной или ложью, а потом возвращают Объект в оператор ***true*** или ***false***:

//overload operator true

public static bool operator true(ThreeD op)

{

if ((op.x != 0) || (op.y != 0) || (op.z != 0))

return true;

else

return false;

}

//overload operator false

public static bool operator false(ThreeD op)

{

if ((op.x == 0) || (op.y == 0) || (op.z == 0))

return true;

else

return false;

}

Там он определяет заданные параметры, и уже потом возвращает значение true или false.

(подробно описал в ***листинге glava9\_5***)

Описанный выше способ применения укороченных логических операторов может

показаться, на первый взгляд, несколько запутанным, но если подумать, то в таком

применении обнаруживается известный практический смысл. Ведь благодаря перегрузке

операторов true и false для класса компилятор получает разрешение на

применение укороченных логических операторов, не прибегая к явной их перегрузке.

Это дает также возможность использовать объекты в условных выражениях. И вообще,

логические операторы & и | лучше всего реализовывать полностью, если, конечно, не

требуется очень узко направленная их реализация.

**Операторы преобразования**

Иногда объект определенного класса требуется использовать в выражении, включающем

в себя данные других типов. В одних случаях для этой цели оказывается

пригодной перегрузка одного иди более операторов, а в других случаях — обыкновенное

преобразование типа класса в целевой тип. Для подобных ситуаций в C# предусмотрена

специальная разновидность операторного метода, называемая ***оператором***

***преобразования****.* Такой оператор преобразует объект исходного класса в другой тип.

Существуют две формы операторов преобразования: явная и неявная. Ниже они

представлены в общем виде:

***public static explicit operator целевой\_тип(исходный\_тип v) {return значение;}***

***public static implicit operator целевой\_тип(исходный\_тип v) {return значение;}***

где *целевой\_тип* обозначает тот тип, в который выполняется преобразование; *исходный\_*

*тип —* тот тип, который преобразуется; *значение —* конкретное значение,

приобретаемое классом после преобразования. Операторы преобразования возвращают

данные, имеющие *целевой\_тип,* причем указывать другие возвращаемые типы

данных не разрешается.

Если оператор преобразования указан в неявной форме (implicit), то преобразование

вызывается автоматически, например, в том случае, когда объект используется

в выражении вместе со значением целевого типа. Если же оператор преобразования

указан в явной форме (explicit), то преобразование вызывается в том случае, когда

выполняется приведение типов. Для одних и тех же исходных и целевых типов данных

***нельзя*** указывать оператор преобразования ***одновременно*** в явной и неявной форме.

Создадим оператор преобразования специально для класса ThreeD, чтобы продемонстрировать

его применение. Допустим, что требуется преобразовать объект типа

ThreeD в целое значение, чтобы затем использовать его в целочисленном выражении.

Такое преобразование требуется, в частности, для получения произведения всех трех

координат объекта. (***glava9\_6***)

using System;

class ThreeD

{

int x, y, z;

public ThreeD() { x = y = z = 0; }

public ThreeD(int i, int j, int k) { x = i; y = j; z = k; }

//overload operator +

public static ThreeD operator +(ThreeD op1, ThreeD op2)

{

ThreeD result = new ThreeD();

result.x = op1.x + op2.x;

result.y = op1.y + op2.y;

result.z = op1.z + op2.z;

return result;

}

//implicit conversion of obj ThreeD to int

public static implicit operator int(ThreeD op1)

{

return op1.x \* op1.y \* op1.z;

}

//show x,y,z

public void Show()

{

Console.WriteLine(x + ", " + y + ", " + z);

}

}

class ThreeDDemo

{

static void Main()

{

ThreeD a = new ThreeD(1, 2, 3);

ThreeD b = new ThreeD(10, 10, 10);

ThreeD c = new ThreeD();

int i;

Console.Write("Coordinates of a: ");

a.Show();

Console.WriteLine();

Console.Write("Coordinates of b: ");

b.Show();

Console.WriteLine();

c = a + b;

Console.Write("Adding result of a + b: ");

c.Show();

Console.WriteLine();

i = a; //conversion

Console.WriteLine("Result of assignment i = a:" + i);

Console.WriteLine();

}

}

Как следует из приведенного выше примера программы, когда объект типа ThreeD

используется в таком целочисленном выражении, как i = а, происходит его преобразование.

В этом конкретном случае преобразование приводит к возврату целого значения

6, которое является произведением координат точки а, хранящихся в объекте

того же названия. Но если для вычисления выражения преобразование в тип int не

требуется, то оператор преобразования не вызывается. Именно поэтому операторный

метод operator int() *не* вызывается при вычислении выражения с = а + b.

Но для различных целей можно создать разные операторы преобразования. Так,

для преобразования объекта типа ThreeD в тип double можно было бы определить

второй оператор преобразования. При этом каждый вид преобразования выполнялся

бы автоматически и независимо от другого.

Оператор неявного преобразования применяется автоматически в следующих случаях:

когда в выражении требуется преобразование типов; методу передается объект;

осуществляется присваивание и производится явное приведение к целевому типу.

С другой стороны, можно создать оператор явного преобразования, вызываемый только

тогда, когда производится явное приведение типов. В таком случае оператор явного

преобразования не вызывается автоматически. В качестве примера ниже приведен вариант

предыдущей программы, переделанный для демонстрации ***явного*** преобразования

в тип int. (***glava9\_6.1***)

using System;

class ThreeD

{

…

//explicit conversion of obj ThreeD to int

public static explicit operator int(ThreeD op1)

{

return op1.x \* op1.y \* op1.z;

}

…

}

class ThreeDDemo

{

static void Main()

{

…

i = (int)a; //converse to type int explicit bcs of (int)

Console.WriteLine("Result of assignment: " + i);

Console.WriteLine();

i = (int)a \* 2 - (int)b; //require explicit cast to int

Console.WriteLine("Result of calculation a \* 2 - b: " + i);

}

}

Оператор преобразования теперь указан в явной форме, и поэтому преобразование

должно быть явно приведено к типу int. Например, следующая строка кода не

будет скомпилирована, если исключить приведение типов.

i = (int)a;

На операторы преобразования накладывается ряд следующих ограничений.

• Исходный или целевой тип преобразования должен относиться к классу, для которого

объявлено данное преобразование. В частности, нельзя переопределить

преобразование в тип int, если оно первоначально указано как преобразование

в тип double.

• Нельзя указывать преобразование в класс object или же из этого класса.

• Для одних и тех же исходных и целевых типов данных нельзя указывать одновременно

явное и неявное преобразование.

• Нельзя указывать преобразование базового класса в производный класс.

• Нельзя указывать преобразование в интерфейс или же из него.

Несмотря на все преимущества неявных преобразований, к ним следует прибегать

только в тех случаях, когда преобразованию не свойственны ошибки. Во избежание

подобных ошибок неявные преобразования должны быть организованы только

в том случае, если удовлетворяются следующие условия. Во-первых, информация не

теряется, например, в результате усечения, переполнения или потери знака. И во-

вторых, преобразование не приводит к исключительной ситуации. Если же неявное

преобразование не удовлетворяет этим двум условиям, то следует выбрать явное преобразование.

**Рекомендации и ограничения по перегрузке операторов**

Pади сохранения ясности структуры и удобочитаемости исходного

кода перегружаемый оператор должен, по возможности, отражать основную суть

своего первоначального назначения. Например, назначение оператора + для класса

ThreeD по сути не должно заметно отличаться от его назначения для целочисленных

типов данных. Если бы, например, определить оператор + относительно некоторого

класса таким образом, чтобы по своему действию он стал больше похожим на оператор

/, то вряд ли от этого было бы много проку. Главный принцип перегрузки операторов

заключается в следующем: несмотря на то, что перегружаемый оператор может

получить любое назначение, ради ясности новое его назначение должно быть так или

иначе связано с его первоначальным назначением.

На перегрузку операторов накладывается ряд ограничений. В частности, нельзя изменять

приоритет любого оператора или количество операндов, которое требуется

для оператора, хотя в операторном методе можно и проигнорировать операнд. Кроме

того, имеется ряд операторов, которые нельзя перегружать. А самое главное, что перегрузке

***не*** подлежит ***ни один*** из ***операторов*** ***присваивания***, в том числе и составные,

как, например, оператор +=. Ниже перечислены операторы, которые нельзя перегружать.

Среди них имеются и такие операторы, которые будут рассматриваться далее в

этой книге.

