# Перегрузка операций

## Перегрузка унарных операций

*Методы* — это процедуры (подпрограммы), которые манипулируют данными, определенными в классе, и во многих случаях обеспечивают доступ к этим данным. Обычно различные части программы взаимодействуют с классом посредством его методов. Любой метод содержит одну или несколько инструкций. В хорошей программе один метод выполняет только одну задачу. Каждый метод имеет имя, и именно это имя используется для его вызова.

Формат записи метода такой:

спецификатор доступа тип\_возврата имя(список\_параметров)

{

// тело метода

}

В С# два или больше методов внутри одного класса могут иметь одинаковое имя, но при условии, что их параметры будут различными. Такую ситуацию называют *перегрузкой методов* (method overloading), а методы, которые в ней задействованы, — *перегруженными* (overloaded). Перегрузка методов — один из способов реализации полиморфизма в С#.

Перегрузка операторов тесно связана с перегрузкой методов. Для перегрузки операторов используется ключевое слово operator, позволяющее создать операторный метод, который определяет действие оператора, связанное с его классом.

Существует две формы методов operator: одна используется для унарных операторов, а другая — для бинарных.

// Общий формат перегрузки для унарного оператора,

public static *тип\_возврата* operator *ор( тип\_параметра операнд)*

***{***

*//* операции

}

**Параметры операторов не должны использовать модификатор ref или out.**

Для унарных операторов тип операнда должен совпадать с классом, для которого определен оператор. Можно определять в классе следующие унарные операции:

* **+ - ! - ++ -- true false**

Префиксный и постфиксный инкременты не различаются (для них может существовать только одна реализация, которая вызывается в обоих случаях).

Параметр, передаваемый в операцию, должен иметь тип класса, для которого она определяется. Операция должна возвращать:

* для операций +, -, ! и - величину любого типа;
* для операций ++ и - - величину типа класса, для которого она определяется;
* для операций true и false величину типа bool.

Операции не должны изменять значение передаваемого им операнда. Операция, возвращающая величину типа класса, для которого она определяется, должна создать новый объект этого класса, выполнить с ним необходимые действия и передать его в качестве результата.

**Операции преобразования типа** обеспечивают возможность явного и неявного преобразования между пользовательскими типами данных. Синтаксис объявите для операции преобразования типа:

**implicit operator тип ( параметр ) // неявное преобразование**

**explicit operator тип ( параметр ) // явное преобразование**

Эти операции выполняют преобразование из типа параметра в тип, указанный в заголовке операции. Одним из этих типов должен быть класс, для которого определяется операция. Таким образом, операции выполняют преобразования либо типа класса к другому типу, либо наоборот. Преобразуемые типы не должны быть связаны отношениями наследования.

Неявное преобразование выполняется автоматически:

* при присваивании объекта переменной целевого типа;
* при использовании объекта в выражении, содержащем переменные целевого типа;
* при передаче объекта в метод на место параметра целевого типа;
* при явном приведении типа.

Явное преобразование выполняется при использовании операции приведения типа.

*Примеры*

class Fraction

{

int numerator;

int denominator;

static int count = 0;

. . . . . . . . . .

public static Fraction operator ++(Fraction a)

{

a = a + 1; // должна быть перегружена операция сложения

return a;

}

public static implicit operator double(Fraction a)

{

return a.Numerator \* 0.1 / 0.1 / a.Denominator;

}

public static explicit operator int(Fraction a)

{

return a.Numerator / a.Denominator;

}

}

class Program

{

static void Main(string[] args)

{

. . . .

Console.WriteLine("++:");

++x5;

x5.Show();

Console.WriteLine("Должно быть 4 9/10");

Console.WriteLine("++:");

x5++;

x5.Show();

Console.WriteLine("Должно быть 5 9/10");

Console.WriteLine("Приведение типов неявное:");

double d = x5;

Console.WriteLine(d);

Console.WriteLine("Должно быть 5.9");

Console.WriteLine("Приведение типов явное:");

int i =(int) x5;

Console.WriteLine(i);

Console.WriteLine("Должно быть 5");

}

}

**Перегрузка бинарных операций**

Можно определять в классе следующие бинарные операции:

* + - \* / % & ≪ ≫ = = ! = > < > = < =

Синтаксис объявителя бинарной операции:

// Общий формат перегрузки для бинарного оператора,

public static тип\_возврата operator op{тип\_параметра1 операнд1, тип\_параметра2 операнд2)

{

// операции

}

Хотя бы один параметр, передаваемый в операцию, должен иметь тип класса, для которого она определяется. Операция может возвращать величину любого типа.

Операции == и ! =, > и <, >= и <= определяются только парами и обычно возвращают логическое значение. Чаще всего в классе определяют операции сравнения на равенство и неравенство для того, чтобы обеспечить сравнение объектов, а не их ссылок, как определено по умолчанию для ссылочных типов.

Чтобы обеспечить возможность сложения с константой, операция сложения должна быть перегружена два раза для случаев, когда константа является первым и вторым операндом.

Пример.

public static Fraction operator +(Fraction a, Fraction b)

{

Fraction res = new Fraction();

if (a.Denominator == b.Denominator)

{

res.Numerator = a.Numerator + b.Numerator;

res.Denominator = a.Denominator;

}

else

{

res.Numerator = a.Numerator \* b.Denominator + b.Numerator \* a.Denominator;

res.Denominator = a.Denominator \* b.Denominator;

}

return res;

}

public static Fraction operator +(Fraction a, int x)

{

Fraction b = new Fraction(x, 1);

Fraction res = new Fraction();

if (a.Denominator == b.Denominator)

{

res.Numerator = a.Numerator + b.Numerator;

res.Denominator = a.Denominator;

}

else

{

res.Numerator = a.Numerator \* b.Denominator + b.Numerator \* a.Denominator;

res.Denominator = a.Denominator \* b.Denominator;

}

return res;

}

public static Fraction operator +(int x, Fraction a)

{

Fraction b = new Fraction(x, 1);

Fraction res = new Fraction();

if (a.Denominator == b.Denominator)

{

res.Numerator = a.Numerator + b.Numerator;

res.Denominator = a.Denominator;

}

else

{

res.Numerator = a.Numerator \* b.Denominator + b.Numerator \* a.Denominator;

res.Denominator = a.Denominator \* b.Denominator;

}

return res;

}

public static bool operator ==(Fraction a, Fraction b)

{

return a.Numerator == b.Numerator && a.Denominator == b.Denominator;

}

public static bool operator !=(Fraction a, Fraction b)

{

return !(a.Numerator == b.Numerator && a.Denominator == b.Denominator);

}

. . . .

static void Main(string[] args)

{

Console.WriteLine("Сложение:");

x5 = f15 + f14;

x5.Show();

Console.WriteLine("Должно быть 2 9/10");

x5 = x5 + 1;

x5.Show();

Console.WriteLine("Должно быть 3 9/10");

Fraction x6 = new Fraction();

x6 = 1+x5;

x6.Show();

Console.WriteLine("Должно быть 4 9/10");

Console.WriteLine("Сравнение:");

Console.WriteLine(x5==x5);

Console.WriteLine("true");

Console.WriteLine("Сравнение:");

Console.WriteLine(x5 != x5);

Console.WriteLine("false");

Console.WriteLine("Сравнение:");

Console.WriteLine(f15 == x6);

Console.WriteLine("false");

Console.WriteLine("Сравнение:");

Console.WriteLine(f15 != x6);

Console.WriteLine("true");

}

## Методы-индексаторы

К элементам массивов можно обращаться используя операцию доступа по индексу:

int []mas={1,5,6,7,4};

for(int i=0; i<Array.Length; i++)

Console.Write(mas[i]+” “);

В С# имеется возможность проектировать специальные классы, которые могут быть индексированы подобно стандартному массиву, посредством определения метода-индексатора.

Индексаторы могут характеризоваться одной или несколькими размерностями.

**Синтаксис одномерного индексатора**

тип\_элемента this[int индекс]

{

get {

// Возврат значения, заданного элементом индекс.

}

set {

// Установка значения, заданного элементом индекс.

}

}

**Семантика индексатора**

* Тип\_элемента — это тип каждого элемента, к которому предоставляется доступ посредством индексатора. Он соответствует базовому типу массива.
* Параметр индекс получает индекс опрашиваемого (или устанавливаемого) элемента.
* В теле индексатора определяются средства доступа ( аксессоры).
* Аксессор = методу за исключением того, что в нем отсутствует объявление типа возвращаемого значения и параметров.
* При использовании индексатора аксессоры вызываются автоматически, и в качестве параметра оба аксессора принимают индекс.
* Если индексатор стоит слева от оператора присваивания, вызывается аксессор set и устанавливается элемент, заданный параметром индекс.
* В противном случае вызывается аксессор get и возвращается значение, соответствующее параметру индекс.
* Метод set также получает значение (value), которое присваивается элементу массива, найденному по заданному индексу.

class FractionArray

{

Fraction[] arr;

int size;

public FractionArray()

{

arr = null;

size = 0;

}

public FractionArray(int s)

{

size = s;

arr = new Fraction[size];

Random rnd = new Random();

for (int i = 0; i < size; i++)

{

int n = rnd.Next(-10, 10);

int d = rnd.Next(0, 10);

Fraction f = new Fraction(n, d);

arr[i] = f;

}

}

public void Show()

{

for (int i = 0; i < size; i++)

{

arr[i].Show();

}

}

public Fraction this[int index]

{

get

{

if (index >= 0 && index < size)

return arr[index];

else

{

Console.WriteLine("неправильно задан индекс");

return new Fraction(-1000, 1);

}

}

set

{

if (index >= 0 && index < size)

arr[index] = value;

else

Console.WriteLine("неправильно задан индекс");

}

}

}

. . . .

static void Main(string[] args)

{

. . . .

FractionArray fa = new FractionArray(5);

fa.Show();

Console.WriteLine("Проверка индексатора");

Console.WriteLine("Доступ к элементу с номером 0 - селектор");

Fraction f = new Fraction();

f = fa[0];

f.Show();

Console.WriteLine("Доступ к элементу с номером 0 - модификатор");

fa[0] = new Fraction(5, 10);

fa.Show();

Console.WriteLine("Доступ к элементу с номером 10 - селектор");

f = fa[10];

f.Show();

Console.WriteLine("Доступ к элементу с номером 10 - модификатор");

fa[10] = new Fraction(5, 10);

fa.Show();

}

//Программа полностью

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace Fraction\_1

{

class Fraction

{

int numerator;

int denominator;

static int count = 0;

//свойства

public int Numerator

{

get { return numerator; }

set { numerator = value; }

}

public int Denominator

{

get { return denominator; }

set { if (value != 0)denominator = value; }

}

public Fraction()//конструктор без параметров

{

numerator = 0;

denominator = 1;

count++;

}

public Fraction(int numerator, int denominator)

{

this.numerator = numerator;

if (denominator != 0) this.denominator = denominator;

else this.denominator = 1;

count++;

}

public void Show() //метод для вывода дроби

{

int absNum = Math.Abs(numerator);

int absDenum = Math.Abs(denominator);

if (numerator == 0) //дробь равна 0

{

Console.WriteLine(0);

return;

}

//отрицательная дробь

if (numerator \* denominator < 0) Console.Write("-");

//дробь сокращается

if (absNum % absDenum == 0)

{

Console.WriteLine(absNum / absDenum);

return;

}

//числитель меньше знаменателя => целой части нет

if (absNum < absDenum)

{

Console.WriteLine(absNum + "/" + absDenum);

return;

}

//числитель больше знаменателя => целая часть есть

Console.WriteLine((absNum / absDenum) + " " + (absNum % absDenum) + "/" + absDenum);

}

private int NOD(int a, int b)

{

int r;

while (b != 0)

{

r = a % b;

a = b; b = r;

}

return a;

}

//сложение дробей

public Fraction plus(Fraction b)

{

Fraction res = new Fraction();

if (Denominator == b.Denominator)

{

res.Numerator = Numerator + b.Numerator;

res.Denominator = Denominator;

}

else

{

res.Numerator = Numerator \* b.Denominator + b.Numerator \* Denominator;

res.Denominator = Denominator \* b.Denominator;

}

int nod = NOD(res.numerator, res.denominator);

res.numerator /= nod;

res.denominator /= nod;

return res;

}

static public int GetCount()

{

return count;

}

//перегрузка

public static Fraction operator +(Fraction a, Fraction b)

{

Fraction res = new Fraction();

if (a.Denominator == b.Denominator)

{

res.Numerator = a.Numerator + b.Numerator;

res.Denominator = a.Denominator;

}

else

{

res.Numerator = a.Numerator \* b.Denominator + b.Numerator \* a.Denominator;

res.Denominator = a.Denominator \* b.Denominator;

}

return res;

}

public static Fraction operator +(Fraction a, int x)

{

Fraction b = new Fraction(x, 1);

Fraction res = new Fraction();

if (a.Denominator == b.Denominator)

{

res.Numerator = a.Numerator + b.Numerator;

res.Denominator = a.Denominator;

}

else

{

res.Numerator = a.Numerator \* b.Denominator + b.Numerator \* a.Denominator;

res.Denominator = a.Denominator \* b.Denominator;

}

return res;

}

public static Fraction operator +(int x, Fraction a)

{

Fraction b = new Fraction(x, 1);

Fraction res = new Fraction();

if (a.Denominator == b.Denominator)

{

res.Numerator = a.Numerator + b.Numerator;

res.Denominator = a.Denominator;

}

else

{

res.Numerator = a.Numerator \* b.Denominator + b.Numerator \* a.Denominator;

res.Denominator = a.Denominator \* b.Denominator;

}

return res;

}

public static Fraction operator -(Fraction a, Fraction b)

{

Fraction res = new Fraction();

if (a.Denominator == b.Denominator)

{

res.Numerator = a.Numerator - b.Numerator;

res.Denominator = a.Denominator;

}

else

{

res.Numerator = a.Numerator \* b.Denominator - b.Numerator \* a.Denominator;

res.Denominator = a.Denominator \* b.Denominator;

}

return res;

}

public static Fraction operator ++(Fraction a)

{

a = a + 1;

return a;

}

public static implicit operator double(Fraction a)

{

return a.Numerator \* 0.1 / 0.1 / a.Denominator;

}

public static explicit operator int(Fraction a)

{

return a.Numerator / a.Denominator;

}

public static bool operator ==(Fraction a, Fraction b)

{

return a.Numerator == b.Numerator && a.Denominator == b.Denominator;

}

public static bool operator !=(Fraction a, Fraction b)

{

return !(a.Numerator == b.Numerator && a.Denominator == b.Denominator);

}

}

class Program

{

public static Fraction plus(Fraction a, Fraction b)

{

Fraction res = new Fraction();

if (a.Denominator == b.Denominator)

{

res.Numerator = a.Numerator + b.Numerator;

res.Denominator = a.Denominator;

}

else

{

res.Numerator = a.Numerator \* b.Denominator + b.Numerator \* a.Denominator;

res.Denominator = a.Denominator \* b.Denominator;

}

return res;

}

static void Main(string[] args)

{

//тестируем метод Show и конструкторы

Console.WriteLine("Конструктор без параметров");

Fraction f1 = new Fraction();

f1.Show();

Console.WriteLine("Должно быть 0");

Console.WriteLine("Конструктор с параметрами");

Fraction f2 = new Fraction(1, 2);

f2.Show();

Console.WriteLine("Должно быть 1/2");

Fraction f3 = new Fraction(2, 1);

f3.Show();

Console.WriteLine("Должно быть 2");

Fraction f4 = new Fraction(2, 2);

f4.Show();

Console.WriteLine("Должно быть 1");

Fraction f5 = new Fraction(3, 6);

f5.Show();

Console.WriteLine("Должно быть 3/6");

Fraction f6 = new Fraction(6, 3);

f6.Show();

Console.WriteLine("Должно быть 2");

Fraction f7 = new Fraction(6, 6);

f7.Show();

Console.WriteLine("Должно быть 1");

Fraction f8 = new Fraction(1, 0);

f8.Show();

Console.WriteLine("Должно быть 1");

Fraction f9 = new Fraction(-1, 2);

f9.Show();

Console.WriteLine("Должно быть -1/2");

Fraction f10 = new Fraction(1, -2);

f10.Show();

Console.WriteLine("Должно быть -1/2");

Fraction f11 = new Fraction(0, 2);

f11.Show();

Console.WriteLine("Должно быть 0");

Fraction f12 = new Fraction(0, 0);

f12.Show();

Console.WriteLine("Должно быть 0");

Fraction f13 = new Fraction(3, 0);

f13.Show();

Console.WriteLine("Должно быть 3");

Fraction f14 = new Fraction(-2, -5);

f14.Show();

Console.WriteLine("Должно быть 2/5");

Fraction f15 = new Fraction(-5, -2);

f15.Show();

Console.WriteLine("Должно быть 2 1/2");

//тестируем глобальный метод

Fraction x5 = new Fraction();

x5 = plus(f15, f14); // 2 1/2+ 1/5

x5.Show();

Console.WriteLine("Должно быть 2 9/10");

x5= f15.plus(f14);

x5.Show();

Console.WriteLine("Должно быть 2 9/10");

////Console.WriteLine("===================PART 2==================");

Console.WriteLine("Сложение:");

x5 = f15 + f14;

x5.Show();

Console.WriteLine("Должно быть 2 9/10");

x5 = x5 + 1;

x5.Show();

Console.WriteLine("Должно быть 3 9/10");

Fraction x6 = new Fraction();

x6 = 1+x5;

x6.Show();

Console.WriteLine("Должно быть 4 9/10");

Console.WriteLine("Сравнение:");

Console.WriteLine(x5==x5);

Console.WriteLine("true");

Console.WriteLine("Сравнение:");

Console.WriteLine(x5 != x5);

Console.WriteLine("false");

Console.WriteLine("Сравнение:");

Console.WriteLine(f15 == x6);

Console.WriteLine("false");

Console.WriteLine("Сравнение:");

Console.WriteLine(f15 != x6);

Console.WriteLine("true");

Console.WriteLine("++:");

++x5;

x5.Show();

Console.WriteLine("Должно быть 4 9/10");

Console.WriteLine("++:");

x5++;

x5.Show();

Console.WriteLine("Должно быть 5 9/10");

Console.WriteLine("Приведение типов неявное:");

double d = x5;

Console.WriteLine(d);

Console.WriteLine("Должно быть 5.9");

Console.WriteLine("Приведение типов явное:");

int i =(int) x5;

Console.WriteLine(i);

Console.WriteLine("Должно быть 5");

Console.WriteLine("===================PART 3==================");

FractionArray fa = new FractionArray(5);

fa.Show();

Console.WriteLine("Проверка индексатора");

Console.WriteLine("Доступ к элементу с номером 0 - селектор");

Fraction f = new Fraction();

f = fa[0];

f.Show();

Console.WriteLine("Доступ к элементу с номером 0 - модификатор");

fa[0] = new Fraction(5, 10);

fa.Show();

Console.WriteLine("Доступ к элементу с номером 10 - селектор");

f = fa[10];

f.Show();

Console.WriteLine("Доступ к элементу с номером 10 - модификатор");

fa[10] = new Fraction(5, 10);

fa.Show();

}

}

}

//файл с классом-контейнером

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace Fraction\_1

{

class FractionArray

{

Fraction[] arr;

int size;

public FractionArray()

{

arr = null;

size = 0;

}

public FractionArray(int s)

{

size = s;

arr = new Fraction[size];

Random rnd = new Random();

for (int i = 0; i < size; i++)

{

int n = rnd.Next(-10, 10);

int d = rnd.Next(0, 10);

Fraction f = new Fraction(n, d);

arr[i] = f;

}

}

public void Show()

{

for (int i = 0; i < size; i++)

{

arr[i].Show();

}

}

public Fraction this[int index]

{

get

{

if (index >= 0 && index < size)

return arr[index];

else

{

Console.WriteLine("неправильно задан индекс");

return new Fraction(-1000, 1);

}

}

set

{

if (index >= 0 && index < size)

arr[index] = value;

else

Console.WriteLine("неправильно задан индекс");

}

}

}

}