Лекция 5. Операторы. Перегрузка операторов.

Половикова О.Н.

Пример. Обслуживающий класс.

```
    }
    static class Circle{
        public const double pi = Math.PI;
        //  площадь и длину круга
        public static double SquareCircle(int R)
        {
            return pi * R * R;
        }
        public static double LengthCircle(int R){
            return 2 * pi * R;
        }
    }

    class Program{
    public static void Main(string[] args)
        {
            Console.Write(Circle.SquareCircle(5));

        //
        //
        //
```

```
C:\doc\string_build\string_build\bin\Debug\string_build.exe
78,53981633974483
```

Повторение

Статическое свойство. Статический конструктор.

_build.Program ▾ | Main(string[] args)

```
class Student{
    public static int max_ball;
    int[] balls= new int[3];
    public string Name{get; set;}
    public static StringBuilder All_balls{get; set;} // строковое представление баллов
    //статический конструктор
    static Student(){
        All_balls = new StringBuilder("баллы:");
    }
    public Student(string name, int math, int phis, int lang){
        this.Name=name;
        balls[0]=math; balls[1]=phis; balls[2]=lang;
        max_ball=(summa_balls()>max_ball)? summa_balls():max_ball;
        All_balls.Append(" "+summa_balls().ToString());
    }
    public int summa_balls(){
        return balls[0]+balls[1]+balls[2];
    }
}
class Program
{
public static void Main(string[] args)
    {
        Student S = new Student("Иванов", 18, 30, 34);
        Student.max_ball+=10;
        Console.Write(Student.All_balls.ToString());
```

C:\doc\string_build\string_build\bin\Debu

```
баллы: 82_
```

Повторение

Статическое свойство. Ошибка

```
class Student{
    public static int max_ball=0;
    int[] balls= new int[3];
    public string Name{get; set;}
    public static StringBuilder All_balls{get; set;} // строковое представление баллов
    /*static Student(){
        All_balls = new StringBuilder("баллы:");
    }*/
    public Student(string name, int math, int phis, int lang){
        this.Name=name;
        balls[0]=math; balls[1]=phis; balls[2]=lang;
        max_ball=(summa_balls()>max_ball)? summa_balls():max_ball;
        All_balls.Append(" "+summa_balls().ToString());
    }
    public int summa_balls(){
        return balls[0]+balls[1]+balls[2];
    }
}
class Program
{
public static void Main
    {
        Student S = new
        Student.max_bal
        Console.Write(S
```

Сборка Отладка Поиск Анализ Инструменты Окно Справка

Необработанное исключение

An exception of type System.NullReferenceException was thrown:

```
System.NullReferenceException: Ссылка на объект не указывает на экземпляр объекта.
   в string_build.Student..ctor(String name, Int32 math, Int32 phis, Int32 lang) в c:\doc\string_build\
   в string_build.Program.Main(String[] args) в c:\doc\string_build\string_build\Program.cs:строка 48
```

Открытое статическое поле – нарушает принцип инкапсусяции;
Проблема выделения памяти для автоматического свойства (инициализация)

Оператор

Выражения формируются из **операндов** и **операторов**. Операторы в выражении указывают, какие действия нужно применить к операндам.

Примеры **операторов**: +, -, *, / и new (знак или обозначние операнда).

Операндами могут являться литералы (константы), поля, локальные переменные, выражения и т. п.

```
string a, b;
a = “Иванов”;
b = “Иван”;
Console.Write(a+”  “+b);
```

Операторы: sizeof, typeof

Возвращает размер отводимый для переменной данного типа в байтах

```
13    static void Main() {
14
15    Console.Write("\n int = {0}, double = {1}",
16                  sizeof(int), sizeof(double));
17    }
18
```

input

```
int = 4, double = 8
```

Оператор **typeof** применяется к имени типа, известному во время компиляции, а результатом является объект типа **System.Type** , содержащий метаинформацию о типе.

```
12  class prog {
13    static void Main() {
14    Console.Write("\n int = {0}, float = {1}", typeof(int), typeof(float));
15    }
16  }
```

input

```
int = System.Int32, float = System.Single
```

Оператор: is

Данный оператор проверяет, можно ли преобразовать выражение в указанный тип и, если это возможно, приводит его к переменной (объекту) этого типа. Позволяет проверить и преобразовать к конкретному типу.

Этот оператор используется в двух вариантах:

1) **expr is type**

expr – это выражение, значением которого является экземпляр какого-либо типа,

type – это имя типа.

```
public static void Main(string[] args)
{
    if (1.6 is double) Console.Write(true);
```

```
public class Point{
    int x, y;
}
public class Program
{
    public static void Main(string[] args)
    {
        Point  P = new Point();
        if (P is Point) Console.Write(true);
```

```
public static void Main(string[] args)
{
    float a = 1.6f;
    if (a is double) Console.Write(true);
    else Console.Write(false);
```

Оператор: is

2) expr is type varname

expr – это выражение, значением которого является экземпляр какого-либо типа,
type – это имя типа, в который должен быть преобразован результат *expr*,
varname – это объект, в который преобразуется результат *expr*, если

```
class Program
{
    Ссылок: 2
    public static void inf(Object a){
        if (a is Point p) Console.Write(p.Length(new Point(10, 12, -10)));
        if (a is Employee em) Console.Write(em.ToString());
    }

Ссылок: 0
static void Main(string[] args){
    //первый объект
    Employee A = new Employee("Ivan", "Iv", "Ivanov", 20.8);
    //вызов функции
       inf(A);
    // второй объект
       Point B = new Point(-2, 0, 2);
    // та  же самая функция
       inf(B);
       Console.ReadKey();
     }
}
    }
}
```

```
"", double Pay = 0.0)
```

```
" оклад:"+Pay.ToString();
```

Приоритет операторов

Категория	Операции
Первичные операции	() [] . ++ -- new typeof sizeof checked unchecked
Унарные операции	+ - ! ~ ++ -- (T)x
Операции умножения и деления	* / %
Операции сложения и вычитания	+ -
Операции сдвига	<< >>
Операции отношения	< > <= >= is
Операции равенства	== !=
Поразрядная операция И	&
Поразрядная операция XOR	^
Поразрядная операция ИЛИ	|
Логическая операция И	&&
Логическая операция ИЛИ	||
Операция условия	?:
Операция присваивания	= *= /= %= += -= <<= >>= &= ^= |=

Ассоциативность операторов

- Операторы с левой ассоциативностью вычисляются слева направо. Все бинарные операторы (почти все) имеют левую ассоциативность.
- Например, выражение a + b - c вычисляется как (a + b) - c.
- Операторы с правой ассоциативностью вычисляются справа налево. Операторы присваивания и условный оператор ?: имеют правую ассоциативность.
- Например, выражение x = y = z вычисляется как x = (y = z).

Вычисление операндов

Операнды в выражении вычисляются слева направо.

Выражение	Порядок вычислений
a + b	a, b, +
a + b * c	a, b, c, *, +
a / b + c * d	a, b, /, c, d, *, +
a / (b + c) * d	a, b, c, +, /, d, *

Как правило, оцениваются (вычисляются) все операнды операторов.
Некоторые операторы оценивают (вычисляют) операнды условно. То есть значение первого операнда такого оператора определяет, следует ли оценивать другие операнды.
Например, && ,||, ?:

Вычисление операндов

Некоторые операторы оценивают (вычисляют) операнды условно. То есть значение первого операнда такого оператора определяет, следует ли оценивать другие операнды. Например, && ,||, ?:

```
10
11  class prog {
12    static void Main() {
13
14      int  a = 10;
15      int  b = 20;
16      if (a < 0.0 && (++b) > 20) Console.Write("\n a ={0}, b = {1}", a, b);
17      else Console.Write("\n b ={0}, a = {1}", b, a);
18
19       if ((++b) > 20 && a < 0.0) Console.Write("\n a ={0}, b = {1}", a, b);
20      else Console.Write("\n b ={0}, a = {1}", b, a);
21    }
22  }
```

```
b =20, a = 10
b =21, a = 10
```

```
11  class prog {
12    static void Main() {
13
14      int  a = 10;
15      int  b = 20;
16      if (a > 0.0 || (++b) > 20) Console.Write("\n a ={0}, b = {1}", a, b);
17      else Console.Write("\n b ={0}, a = {1}", b, a);
18    }
19  }
```

input

```
a =10, b = 20
```

Перегрузка операторов

Перегрузка оператора – это реализация своего собственного функционала этого оператора для конкретного класса.

С помощью **перегрузки операторов** можно указать поведение оператора для операндов определяемого пользователем типа.

Перегрузка унарного оператора:

```
public static [возвращаемый_тип] operator
[оператор]([тип_операнда] [операнд])
{
  //функционал оператора
}
```

```
class Complex{
        double re, im;
        public override string ToString()
        {
            return string.Format("[Complex Re={0}, Im={1}]", re, im);
        }
        public void print(){
            Console.WriteLine("\n({0},{1})", re, im);
        }
        public Complex (double re=0.0, double im=0.0){
            this.re=re;
            this.im=im;
        }
        public static Complex operator-(Complex A){ return new Complex(-A.re, -A.im);}
```

Модификаторы **public и static** являются обязательными. На месте [**оператор**] может стоять любой оператор, который можно перегрузить. Не все операторы в C# разрешается перегружать.

Перегрузка бинарного оператора:

public static [возвращаемый_тип] operator [оператор]([тип_операнда1] [операнд1], [тип_операнда2] [операнд2])

{

  //функционал оператора

}

```
class Complex{
        double re, im;
        public override string ToString()
        {
            return string.Format("[Complex Re={0}, Im={1}]", re, im);
        }
        public void print(){
            Console.WriteLine("\n({0},{1})", re, im);
        }
        public Complex (double re=0.0, double im=0.0){
            this.re=re;
            this.im=im;
        }
        public static Complex operator-(Complex A){ return new Complex(-A.re, -A.im);}

        public  static Complex operator+(Complex A, Complex B){ return new Complex(B.re+A.re, B.im+A.im);}
```

reload_operators.Complex

```
13
14      class Program
15      {
16          public static void Main(string[] args)
17          {
18              Complex A = new Complex(-3, 4);
19              Complex B = new Complex(2, 5);
20              (A.sum(B)).print();
21
22              Console.WriteLine(Complex.sum(A,B));
23              Console.ReadKey(true);
24          }
25      }
26
27      //перегрузка бинарных операторов
28      class Complex{
29          double re, im;
30          public override string ToString()
31          {
32              return string.Format("[Complex Re={0}, Im={1}]", re, im);
33          }
34
35          public void print(){
36              Console.WriteLine("\n({0},{1})", re, im);
37
38          }
39          public Complex (double re=0.0, double im=0.0){
40              this.re=re;
41              this.im=im;
42          }
43
44          public Complex sum(Complex A){
45              return new Complex(re+A.re, im+A.im);
46          }
47          public static Complex sum(Complex A, Complex B){
48              return new Complex(B.re+A.re, B.im+A.im);
49          }
```

Операция C#	Возможность перегрузки
+, -, !, ++, —, true, false	Этот набор унарных операций может быть перегружен
+, -, *, /, %, &, |, ^, <<, >>	Эти бинарные операции могут быть перегружены
==, !=, <, >, <=, >=	Эти операции сравнения могут быть перегружены. C# требует совместной перегрузки «подобных» операций (т.е. < и >, <= и >=, == и !=)
[ ]	Операция [] не может быть перегружена. Аналогичную функциональность предлагают индексаторы
( )	Операция () не может быть перегружена, но ту же функциональность предоставляют специальные методы преобразования
+=, -=, *=, /=, %=, &=, |=, ^=, <<=, >>=	Сокращенные операции присваивания не могут перегружаться; однако вы получаете их автоматически, перегружая соответствующую бинарную операцию

Нельзя перегружать

Перегружать можно только операторы, перечисленные выше.
В частности, невозможно перегрузить доступ к члену, вызов метода или

=, &&, ||, ?:, =>, checked, unchecked и new операторы.

Правила перегрузки

1. Объявление оператора должно включать спецификатор public и модификатор static.
2. Унарный оператор принимает один параметр. Бинарный оператор принимает два параметра. В каждом случае хотя бы один параметр должен иметь тип объекта создаваемого класса.
3. Объявления определяемых пользователем операторов не могут изменять синтаксис, приоритет или ассоциативность оператора. Например, оператор / всегда является бинарным оператором, всегда имеет свой уровень приоритета.
4. Перегружать можно только разрешённые для перегрузки операторы.

```
        public static void Main(string[] args)
        {
            math.Complex A = new math.Complex(-3, 4);
            math.Complex B = new math.Complex(2, 5);
            Console.WriteLine((A+B).ToString());
            Console.WriteLine((A+3.6).ToString());
            Console.WriteLine((3.6+B-A-7.8-(-A)+(-B)).ToString());
            Console.ReadKey(true);
        }
    }
    class Complex...
}
namespace math{
class Complex{
        double re, im;
        public override string ToString()
        {
            return string.Format("[Complex Re={0}, Im={1}]", re, im);
        }
        public void print(){
            Console.WriteLine("\n({0},{1})", re, im);
        }
        public Complex (double re=0.0, double im=0.0){
            this.re=re;
            this.im=im;
        }
        public static Complex operator-(Complex A){ return new Complex(-A.re, -A.im);}

        public  static Complex operator+(Complex A, Complex B){ return new Complex(B.re+A.re, B.im+A.im);}
        public  static Complex operator-(Complex A, Complex B){ return new Complex(A.re-B.re, A.im-B.im);}

        public  static Complex operator+(Complex A, double B){ return new Complex(A.re+B, A.im+0);}
        public  static Complex operator+( double B, Complex A){ return  new Complex(A.re+B, A.im+0);}

        public  static Complex operator-(Complex A, double B){ return new Complex(A.re-B, A.im-0);}
        public  static Complex operator-( double B, Complex A){ return  new Complex(B-A.re, 0-A.im);}

    }
```

Правильная перегрузка инкремента декремента

```
class counter{
        int i, maxi, mini;
public counter(int i, int mini, int maxi){
            this.i=i; this.maxi=maxi; this.mini=mini;
        }
public static counter operator++(counter A){
    if (A.i+1 <= A.maxi)
         return new counter(A.i+1, A.mini, A.maxi);
    else
          return new counter(A.i, A.mini, A.maxi);
        }
public static counter operator--(counter A){
    if (A.i-1 >= A.mini)
       return  new counter(A.i-1, A.mini, A.maxi);
    else
      return  new counter(A.i, A.mini, A.maxi);
}
public void print(){
Console.WriteLine("({0}, {1}, {2})", mini, i, maxi);
}
}
```

```
public class Program
{
    public static void Main(string[] args)
    {
        counter newCount= new counter(0,-1,10);
        (++newCount).print();//1
        (newCount--).print();//1
        (--newCount).print();//-1
        (newCount++).print();//-1

    }
}
```

Выполнение кода

```
35          public static void Main(string[] args)
36          {
37              counter newCount= new counter(0,-1,10);
38              (++newCount).print();//1
39              (newCount--).print();//1
40              (--newCount).print();//-1
41              (newCount++).print();//-1
42
43          }
44      }
45  }
```

Run it (F8) Save it Show compiler warnings [+] Show Liv

input

Compilation time: 0,23 sec, absolute running time: 0,11 sec, cpu time: 0,11 sec, ave
of threads: 3, absolute service time: 0,37 sec

```
(-1, 1, 10)
(-1, 1, 10)
(-1, -1, 10)
(-1, -1, 10)
```

Средний столбец – значение счётчика в объекте

Неправильная перегрузка

```
class counter{
    int i, maxi, mini;
    public counter(int i, int mini, int maxi){
        this.i=i; this.maxi=maxi; this.mini=mini;
    }
    public static counter operator++(counter A){
        if (A.i+1 <= A.maxi)
           A.i++;
        return A;
    }
    public static counter operator--(counter A){
        if (A.i-1 >= A.mini)
            A.i--;
        return A;
    }
    public void print(){
        Console.WriteLine("({0}, {1}, {2})", mini, i, maxi);
    }
   }
public class Program
{
    public static void Main(string[] args)
    {
        counter newCount= new counter(0,-1,10);
        (++newCount).print();//1
        (newCount--).print();//1
        (--newCount).print();//-1
        (newCount++).print();//-1
```

```
32          public static void Main(string[] args)
33          {
34              counter newCount= new counter(0,-1,10);
35              (++newCount).print();//1
36              (newCount--).print();//1
37              (--newCount).print();//-1
38              (newCount++).print();//-1
39
40          }
41      }
42  }
```

Run it (F8) Save it Show compiler warnings [+] Sh

input

Compilation time: 0,39 sec, absolute running time: 0,09 sec, cpu ti
of threads: 2, absolute service time: 0,51 sec

```
(-1, 1, 10)
(-1, 0, 10)
(-1, -1, 10)
(-1, 0, 10)
```