В C# допускается, чтобы метод вызывал самого себя. Этот процесс называется *рекурсией,*

а метод, вызывающий самого себя, — *рекурсивным.* Вообще, рекурсия представляет

собой процесс, в ходе которого нечто определяет самое себя. В этом отношении она

чем-то напоминает циклическое определение. Рекурсивный метод отличается главным

образом тем, что он содержит оператор, в котором этот метод вызывает самого себя.

Рекурсия является эффективным механизмом управления программой.

Классическим примером рекурсии служит вычисление факториала числа. Факториал

числа *N* представляет собой произведение всех целых чисел от 1 до *N.* Например,

факториал числа 3 равен 1×2×3, или 6. В приведенном ниже примере программы

демонстрируется рекурсивный способ вычисления факториала числа. Для сравнения в

эту программу включен также нерекурсивный вариант вычисления факториала числа. (***glava8\_15***)

class Factorial

{

//recurs

public int FactR(int n)

{

int result;

if (n == 1) return 1;

result = FactR(n - 1) \* n;

return result;

}

//itaration method

public int FactI(int n)

{

int t, result;

result = 1;

for (t = 1; t <= n; t++) result \*= t;

return result;

}

}

class Recursion

{

static void Main()

{

Factorial f = new Factorial();

Console.WriteLine("Factorials, calculated with recursion.");

Console.WriteLine("Factorial of 3 = " + f.FactR(3));

Console.WriteLine("Factorial of 4 = " + f.FactR(4));

Console.WriteLine("Factorial of 5 = " + f.FactR(5));

Console.WriteLine();

Console.WriteLine("Factorials, calculated with iteration.");

Console.WriteLine("Factorial of 3 = " + f.FactI(3));

Console.WriteLine("Factorial of 4 = " + f.FactI(4));

Console.WriteLine("Factorial of 5 = " + f.FactI(5));

}

}

А рекурсивный метод FactR() действует по более сложному принципу. Если метод

FactR() вызывается с аргументом 1, то он возвращает значение 1. В противном

случае он возвращает произведение FactR(n-1)\*n. Для вычисления этого произведения

метод FactR() вызывается с аргументом n-1. Этот процесс повторяется до тех

пор, пока значение аргумента n не станет равным 1, после чего из предыдущих вызовов

данного метода начнут возвращаться полученные значения. Например, когда вычисляется

факториал числа 2, то при первом вызове метода FactR() происходит второй его

вызов с аргументом 1. Из этого вызова возвращается значение 1, которое затем умножается

на 2 (первоначальное значение аргумента n). В итоге возвращается результат 2,

равный факториалу числа 2(1×2). Было бы любопытно ввести в метод FactR() операторы,

содержащие вызовы метода WriteLine(), чтобы наглядно показать уровень

рекурсии при каждом вызове метода FactR(), а также вывести промежуточные результаты

вычисления факториала заданного числа.

Ниже приведен еще один пример рекурсии для вывода символьной строки в обратном

порядке. Эта строка задается в качестве аргумента рекурсивного метода. (***glava8\_15.1***)

class RevStr

{

//invert string

public void DisplayRev(string str)

{

if (str.Length > 0)

DisplayRev(str.Substring(1, str.Length - 1));

else

return;

Console.Write(str[0]);

}

}

class RevStrDemo

{

static void Main()

{

string s = "Text here";

RevStr rsOb = new RevStr();

Console.WriteLine("Raw string " + s);

Console.Write("Inverter string: ");

rsOb.DisplayRev(s);

Console.WriteLine();

}

}

Всякий раз, когда вызывается метод DisplayRev(), в нем происходит проверка

длины символьной строки, представленной аргументом str. Если длина строки не

равна нулю, то метод DisplayRev() вызывается рекурсивно с новой строкой, которая

меньше исходной строки на один символ. Этот процесс повторяется до тех пор,

пока данному методу не будет передана строка нулевой длины. После этого начнется

раскручиваться в обратном порядке механизм всех рекурсивных вызовов метода

DisplayRev(). При возврате из каждого такого вызова выводится первый символ

строки, представленной аргументом str, а в итоге вся строка выводится в обратном

порядке.

Рекурсивные варианты многих процедур могут выполняться немного медленнее,

чем их итерационные эквиваленты из-за дополнительных затрат системных ресурсов

на неоднократные вызовы метода. Если же таких вызовов окажется слишком много, то

в конечном итоге может быть переполнен системный стек. А поскольку параметры и

локальные переменные рекурсивного метода хранятся в системном стеке и при каж**260**

дом новом вызове этого метода создается их новая копия, то в какой-то момент стек

может оказаться исчерпанным. В этом случае возникает исключительная ситуация,

и общеязыковая исполняющая среда (CLR) генерирует соответствующее исключение.

Но беспокоиться об этом придется лишь в том случае, если рекурсивная процедура

выполняется неправильно.

Главное преимущество рекурсии заключается в том, что она позволяет реализовать

некоторые алгоритмы яснее и проще, чем итерационным способом. Например, алгоритм

быстрой сортировки довольно трудно реализовать итерационным способом.

А некоторые задачи, например искусственного интеллекта, очевидно, требуют именно

рекурсивного решения.

При написании рекурсивных методов следует непременно указать в соответствующем

месте условный оператор, например if, чтобы организовать возврат из метода

без рекурсии. В противном случае возврата из вызванного однажды рекурсивного

метода может вообще не произойти. Подобного рода ошибка весьма характерна для

реализации рекурсии в практике программирования. В этом случае рекомендуется

пользоваться операторами, содержащими вызовы метода WriteLine(), чтобы следить

за происходящим в рекурсивном методе и прервать его выполнение, если в нем

обнаружится ошибка.