Отдельно остановимся на рекурсивных функциях. Рекурсивная функция представляет такую конструкцию, при которой функция вызывает саму себя.

### Рекурсивная функция факториала

Возьмем, к примеру, вычисление факториала, которое использует формулу **n! = 1 \* 2 \* … \* n**. То есть по сути для нахождения факториала числа мы перемножаем все числа до этого числа. Например, факториал числа 4 равен 24 = 1 \* 2 \* 3 \* 4, а факторил числа 5 равен 120 = 1 \* 2 \* 3 \* 4 \* 5.

Определим метод для нахождения факториала:

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | int Factorial(int n)  {  if (n == 1) return 1;    return n \* Factorial(n - 1);  } |

При создании рекурсивной функции в ней обязательно должен быть некоторый **базовый вариант**, с которого начинается вычисление функции. В случае с факториалом это факториал числа 1, который равен 1. Факториалы всех остальных положительных чисел будет начинаться с вычисления факториала числа 1, который равен 1.

На уровне языка программирования для возвращения базового варианта применяется оператор **return**:

|  |  |
| --- | --- |
| 1 | if (n == 1) return 1; |

То есть, если вводимое число равно 1, то возвращается 1

Другая особенность рекурсивных функций: все рекурсивные вызовы должны обращаться к подфункциям, которые в конце концов сходятся к базовому варианту:

|  |  |
| --- | --- |
| 1 | return n \* Factorial(n - 1); |

Так, при передаче в функцию числа, которое не равно 1, при дальнейших рекурсивных вызовах подфункций в них будет передаваться каждый раз число, меньшее на единицу. И в конце концов мы дойдем до ситуации, когда число будет равно 1, и будет использован базовый вариант. Это так называемый рекурсивный спуск.

Используем эту функцию:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | int Factorial(int n)  {  if (n == 1) return 1;    return n \* Factorial(n - 1);  }    int factorial4 = Factorial(4); // 24  int factorial5 = Factorial(5); // 120  int factorial6 = Factorial(6); // 720    Console.WriteLine($"Факториал числа 4 = {factorial4}");  Console.WriteLine($"Факториал числа 5 = {factorial5}");  Console.WriteLine($"Факториал числа 6 = {factorial6}"); |

Рассмотрим поэтапно, что будет в случае вызова Factorial(4).

1. Сначала идет проверка, равно ли число единице:

|  |  |
| --- | --- |
| 1 | if (n == 1) return 1; |

Но вначале n равно 4, поэтому это условие ложно, и соответственно выполняется код

|  |  |
| --- | --- |
| 1 | return n \* Factorial(n - 1); |

То есть фактически мы имеем:

|  |  |
| --- | --- |
| 1 | return 4 \* Factorial(3); |

1. Далее выполняется выражение:

|  |  |
| --- | --- |
| 1 | Factorial(3) |

Опять же n не равно 1, поэтому выполняется код

|  |  |
| --- | --- |
| 1 | return n \* Factorial(n - 1); |

То есть фактически:

|  |  |
| --- | --- |
| 1 | return 3 \* Factorial(2); |

1. Далее выполняется выражение:

|  |  |
| --- | --- |
| 1 | Factorial(2) |

Опять же n не равно 1, поэтому выполняется код

|  |  |
| --- | --- |
| 1 | return n \* Factorial(n - 1); |

То есть фактически:

|  |  |
| --- | --- |
| 1 | return 2 \* Factorial(1); |

1. Далее выполняется выражение:

|  |  |
| --- | --- |
| 1 | Factorial(1) |

Теперь n равно 1, поэтому выполняется код

|  |  |
| --- | --- |
| 1 | if (n == 1) return 1; |

И возвращается 1.

В итоге выражение

|  |  |
| --- | --- |
| 1 | Factorial(4) |

В реальности выливается в

|  |  |
| --- | --- |
| 1 | 4 \* 3 \* 2 \* Factorial(1) |

### Рекурсивная функция Фибоначчи

Другим распространенным показательным примером рекурсивной функции служит функция, вычисляющая числа Фибоначчи. n-й член последовательности Фибоначчи определяется по формуле: f(n)=f(n-1) + f(n-2), причем f(0)=0, а f(1)=1. То есть последовательность Фибоначчи будет выглядеть так 0 (0-й член), 1 (1-й член), 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, .... Для определения чисел этой последовательности определим следующий метод:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | int Fibonachi(int n)  {  if (n == 0 || n == 1) return n;    return Fibonachi(n - 1) + Fibonachi(n - 2);  }    int fib4 = Fibonachi(4);  int fib5 = Fibonachi(5);  int fib6 = Fibonachi(6);    Console.WriteLine($"4 число Фибоначчи = {fib4}");  Console.WriteLine($"5 число Фибоначчи = {fib5}");  Console.WriteLine($"6 число Фибоначчи = {fib6}"); |

Здесь базовый вариант выглядит следующий образом:

|  |  |
| --- | --- |
| 1 | if (n == 0 || n == 1) return n; |

То есть, если мы ищем нулевой или первый элемент последовательности, то возвращается это же число - 0 или 1. Иначе возвращается результат выражения Fibonachi(n - 1) + Fibonachi(n - 2);

### Рекурсии и циклы

Это простейшие пример рекурсивных функций, которые призваны дать понимание работы рекурсии. В то же время для обоих функций вместо рекурсий можно использовать циклические конструкции. И, как правило, альтернативы на основе циклов работают быстрее и более эффективны, чем рекурсия. Например, вычисление чисел Фибоначчи с помощью циклов:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | static int Fibonachi2(int n)  {  int result = 0;  int b = 1;  int tmp;    for (int i = 0; i < n; i++)  {  tmp = result;  result = b;  b += tmp;  }    return result;  } |

В то же время в некоторых ситуациях рекурсия предоставляет элегантное решение, например, при обходе различных древовидных представлений, к примеру, дерева каталогов и файлов.