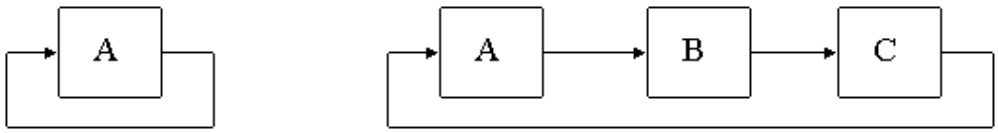
Рекурсия

Чтобы понять рекурсию нужно понять рекурсию.

Рекурсия — вызов функции из неё же самой, непосредственно (простая рекурсия) или через другие функции (сложная или косвенная рекурсия), например, функция A вызывает функцию B, а функция B — функцию A.



Программа разрабатывается сведением исходной задачи к более простым. Среди этих задач может оказаться и первоначальная, но в упрощенной форме.

Например, для вычисления F(N) может понадобиться вычислить F(N−1). Иными словами, частью алгоритма вычисления функции будет вычисление этой же функции.

Итак, функция является **рекурсивной**, если она обращается сама к себе прямо или косвенно (через другие функции). Заметим, что при косвенном обращении все функции в цепочке – рекурсивные.

Рассмотрим это на примере функции вычисления факториала. Хорошо известно, что 0!=1, 1!=1. А как вычислить величину n! для бОльших n? Если бы мы могли вычислить величину (n−1)!, то тогда мы легко вычислили бы n!, поскольку n!=n·(n−1)!. Но как вычислить (n−1)!? Если бы мы вычислили (n−2)!, то мы сможем вычисли и (n−1)!=(n−1)·(n−2)!. А как вычислить (n−2)!? Если бы... В конце концов, мы дойдем до величины 0!, которая равна 1. Таким образом, для вычисления факториала мы можем использовать значение факториала для меньшего числа. Это можно сделать и в программе на Си:

```
int factorial(int n)
{
    if (n == 0)
        return 1;
    else
        return factorial(n - 1) * n;
}
```

Краткая формулировка обоснования рекурсивного алгоритма вычисления факториала:

$$N!=(N-1)! * N,$$

если N = 0, то N! = 1

Как это работает?

Допустим, мы вызвали функцию factorial(4). Будет вызвана функция, у которой значение параметра n равно 4. Она проверит условие n == 0, поскольку условие ложно, то будет выполнена инструкция return n * factorial(n - 1). Но чтобы вычислить это значение, будет вызвана функция factorial(3), так как параметр n имеет значение, равное 4. Теперь в памяти будет находиться две функции factorial -- одна со значением параметра n равным 4, а другая — со значением 3. При этом активна будет последняя функция.

Эта функция в свою очередь вызовет функцию factorial(2), та вызовет функцию factorial(1), затем factorial(0). В случае этой функции ничего более вызвано не будет, функция просто вернет значение 1, и управление вернется в функцию factorial(1). Та умножит значение n = 1 на значение 1, которое вернула функция factorial(0), и вернет полученное произведение, равное 1. Управление вернется в функцию factorial(2), которая умножит n = 2 на значение 1, которое вернула функция factorial(1) и вернет полученное произведение, равное 2. Функция factorial(3) вернет 3×2=6, а функция factorial(4) вернет 4×6==24.

Таблица последовательности вызовов функции приведена ниже. Значения функции возвращают в порядке, обратном порядку их вызова, то есть сначала заканчивает работу функция factorial(0), затем factorial(1) и т. д.

С какими параметрами вызвана функция	**Какое значение вернула**
factorial(4)	4 * 6 = 24
factorial(3)	3 * 2 = 6
factorial(2)	2 * 1 = 2
factorial(1)	1 * 1 = 1
factorial(0)	1

При отладке программы всю последовательность вложенных вызовов рекуррентных функций можно изучить в окне «Call Stack» («стек вызовов») в режиме отладки.

Для каждой вызываемой функции значения локальных переменных будут своими.

Для того, чтобы реализовать рекурсию нужно ответить на следующие вопросы:

1. Какой случай (для какого набора параметров) будет крайним (простым) и что функция возвращает в этом случае?

2. Как свести задачу для какого-то набора параметров (за исключением крайнего случая) к задаче, для другого набора параметров (как правило, с меньшими значениями)?

При этом программирование рекурсии выглядит так. Функция должна сначала проверить, не является ли переданный набор параметров простым (крайним) случаем. В этом случае функция должна вернуть значение (или выполнить действия), соответствующие простому случаю. Иначе функция должна вызвать себя рекурсивно для другого набора параметров, и на основе полученных значений вычислить значение, которое она должна вернуть.

Количество вложенных вызовов функции или процедуры называется *глубиной* рекурсии.

Рекурсия изнутри

Это может показаться удивительным, но самовызов функции/процедуры ничем не отличается от вызова другой функции/процедуры. Что происходит, если одна функция вызывает другую? В общих чертах следующее:

- в памяти размещаются параметры, передаваемые функции (но не параметры-переменные, т. к. они передаются по ссылке!);
- для внутренних переменных вызываемой функции также отводится новая область памяти (несмотря на совпадение их имен и типов с переменными вызывающей функции);
- запоминается адрес возврата в вызывающую функцию;
- управление передается вызванной функции.

Если функцию или процедуру вызвать повторно из другой функции/процедуры или из нее самой, будет выполняться тот же код, но работать он будет *с другими значениями параметров и внутренних переменных*. Это и дает возможность рекурсии.