**Все о рекурсии.**

**Рекурсией** называется определение объекта через такой же объект. В программировании под рекурсией понимают вызов подпрограммы из раздела операторов той же самой подпрограммы, т.е. подпрограмма вызывает сама себя.

В рекурсивном определении обязательно должна присутствовать не рекурсивная часть, а также условие завершения рекурсии (иначе рекурсия окажется бесконечной и приложение будет прервано из-за ошибки переполнения стека). **(*Пример рекурсии – файл ‘сущность рекурсии’, функция подсчета факториала – для самостоятельного разбора).***

**Косвенная рекурсия -**

когда первая подпрограмма вызывает вторую, а вторая — первую. Очевидно, что записанная первой подпрограмма будет содержать еще неизвестный идентификатор второй подпрограммы. Эта проблема решается с помощью упреждающего (предварительного) описания процедур и функций с помощью директивы компилятора **forward.** Предварительное описание состоит из заголовка подпрограммы и следующего за ним зарезервированного слова **forward**, например:

procedure Proc(<формальные параметры>); forward; После такого первичного описания в полном описании процедуры или функции можно не указывать список формальных параметров и тип возвращаемого значения (для функции). **(*Пример косвенной рекурсии – « у попа была собака»).***

**Рекурсивный спуск** - процесс последовательных рекурсивных вызовов подпрограммы из самой себя.

**Рекурсивный возврат** – процесс возврата из рекурсивных вызовов.

Действия в подпрограмме могут выполняться на рекурсивном спуске, **на рекурсивном возврате, и на рекурсивном спуске и возврате одновременно.** (Пример файл ‘спуск\_возврат’).

**Глубина рекурсии – количество одновременно выполняемых процедур.**

**Каскадная рекурсия –** такая рекурсия, если каждый вызов подпрограммы может порождать несколько рекурсивных вызовов (каскад). (**Пример** каскадной рекурсии числа Фибоначчи). Рекурсивный способ вычисления чисел Фибоначчи это плохой вариант использования рекурсии, так как:

1. Одни и те же числа вычисляются по несколько раз (вызов Fib(7)): Fib(7) – 1 раз, Fib(6) – 1 раз, Fib(5) – 2 раза, Fib(4) – 3 раза, Fib(3) – 5 раз, Fib(2) – 8 раз.
2. данный способ крайне неэффективен по сравнению с итерационным алгоритмом как по памяти, так и по времени работы. Каждый вызов создает сразу две копии себя, каждая из копий еще две и т.д.

**Рекурсия называется концевой, если рекурсивный вызов является последним в подпрограмме.**

Любые рекурсивные процедуры и функции, содержащие всего один рекурсивный вызов самих себя, легко заменяются итерационными циклами. Чтобы получить что-то, не имеющее простого не рекурсивного аналога, следует обратиться к процедурам и функциям, вызывающим себя два и более раз. В этом случае множество вызываемых процедур образует уже не цепочку, а целое дерево. Существуют широкие классы задач, когда вычислительный процесс должен быть организован именно таким образом. Как раз для них рекурсия будет наиболее простым и естественным способом решения.

Полезное использование рекурсии: алгоритмы с возвратом (искусственный интеллект), деревья, задача ханойские башни, быстрые сортировки – quicksort, сортировка слиянием, сортировка деревом, задачи на графах (маршрутизация сетей, логистика и транспортные сети и т.д.), фракталы (например, кривая Коха(фигура, состоящая из частей, подобных самой фигуре)).

Быстродействие алгоритмов при избавлении от рекурсии, как правило, повышается. Еще одной причиной, чтобы избавиться от рекурсии, является ограничение на объем хранимых программой локальных переменных и значений параметров одновременно выполняющихся процедур. При очень глубокой рекурсии этот объем возрастает, и программа перестает работать, выдавая ошибку «Stack overflow» (переполнение стека).