же перевод строки, встреченный в фигурных скобках, остаётся самим собой.

Никакие символы внутри фигурных скобок не имеют никакого особого значения; из этого правила есть исключение, которое не все осознают — интерпретатор внутри фигурных скобок особым образом обрабатывает символы фигурных скобок, делая возможной своего рода вложенность строк в строки. Подчеркнём, что встреченные открывающие и закрывающие фигурные скобки продолжают обозначать сами себя, они, как и все другие символы, ни на что не заменяются, Tcl в принципе ничего не меняет в таких строках; просто интерпретатор ведёт подсчёт открывающих и закрывающих фигурных скобок и считает, что строковый литерал закончился, лишь когда количество закрывающих скобок сравняется с количеством открывающих. Между прочим, фигурную скобку (да, внутри фигурных скобок!) можно «заэкранировать» символом «\», тогда интерпретатор исключит её из подсчёта (это может понадобиться, чтобы, например, вывести непарный символ фигурной скобки на экран), но даже при этом и «\», и скобка останутся самими собой.

Коль скоро мы всё равно уже обсуждаем строки, отметим, что внутри литералов в двойных кавычках интерпретатор обрабатывает привычные нам по языку Си обозначения спецсимволов, такие как «`\n`», «`\r`», «`\t`» и некоторые другие; внутри фигурных скобок такие двухсимвольные комбинации останутся ровно тем, чем они и были — двухсимвольными комбинациями, что не исключает их интерпретации в дальнейшем, если содержимое фигурных скобок будет исполнено в качестве фрагмента программы.

Вернёмся теперь к «оператору» `if` и осознаем, что никакой он не оператор, **в Tcl вообще нет операторов**, во всяком случае, в том смысле, в котором они есть в Паскале или Си, а само слово `if` — никоим образом не «ключевое». В действительности `if` — это *просто имя встроенной команды*, такой же, как знакомые нам `expr`, `puts`, `exec` и прочие. Интерпретатор выполняет все команды, включая и эти, по одним и тем же правилам: если нужно (и где нужно) выполняются подстановки (например, значений переменных), после чего первое слово «логической строки» рассматривается как имя команды, последующие — как аргументы; дальнейшее сделает уже реализация команды, и базовый интерпретатор может не делать никаких предположений о том, как она устроена.

Может сложиться такое впечатление, что предыдущий абзац описывает частные детали реализации интерпретатора и, как следствие, не имеет отношения к делу, коль скоро мы не собираемся этот интерпретатор переделывать. В действительности всё сложнее. Интерпретаторы обычно можно *расширять*, добавляя новые возможности; вспомним теперь, что, в Лиспе написать новую спецформу невозможно — для этого придётся переделать сам базовый интер-