ТУБ №31:
Строковые литералы и строковые типы данных в Delphi

Редкой программе удаётся обойтись без работы с фрагментами текста. Это и сообщения, которые выводятся в качестве подсказок для пользователя, и вводимые пользователем данные, и текстовые форматы файлов, и генерируемые самой программой запросы к базам данных, и многое другое. Для обработки таких данных во многих языках программирования (включая, разумеется, Pascal и Delphi) предусмотрены так называемые строковые типы данных.

Вообще *строкой* в программировании может называться любая последовательность элементов. Точнее — «последовательность (или цепочка) символов алфавита», но в самом общем алфавит может включать в себя не только буквы и цифры, но в принципе любые хоть как-то представимые в памяти компьютера элементы, скажем, звуковые фрагменты, элементы изображений и т.п. И всё же на практике почти всегда, когда Вы слышите слово «строка», имеется в виду последовательности тех самых символов, которые «буквы-цифры-запятые», т.е. образующих множество значений символьного типа. Другими словами, в программировании строка — это почти всегда фрагмент текста и лишь изредка что-то ещё.

Соответственно *строковыми типами данных* называют типы данных, множества значений которых образованы фрагментами текста. Не составляет труда и определить множество действий, применимых к таким данным: можно собирать несколько коротких строк в одну более длинную, выделять фрагменты строк, заменять отдельные последовательности символов на другие и т.п. Не забываем и о том, что две строки можно хотя бы сравнить.

Строковые литералы

В большинстве популярных языков программирования (даже в тех, где не поддерживается полноценный строковый тип) есть *строковые литералы* — лексемы, которые задают константу строкового типа непосредственным указанием её значения.

В языках Pascal и Delphi строковые литералы представляют собой фрагменты текста, записанные в апострофах:

```
'Пример строкового литерала'
'abcdef'
'MP01234567'
'if a > b then a := b; // Значением может быть ЛЮБОЙ фрагмент текста'
```

Сами апострофы, разумеется, в состав строки не входят, а лишь используются для того, чтобы компилятор мог в тексте программы опознать строковый литерал и определить, где он заканчивается и продолжается собственно текст программы.

Следует иметь в виду, что при определении длины строки все символы абсолютно равноправны. Для начинающих иногда оказывается неожиданным, что пробел — такой же полноценный символ, как и все прочие. Соответственно длина строки, заданной литералом

```
'Превед, медвед!'
```

равна 15 символам: 12 букв (по 6 в каждом из двух слов), запятая (+ 1), восклицательный знак (+ 1) и пробел (+ 1). Такой подход вполне оправдан, ведь если бы пробел не считался полноценным символом, его можно было бы пропускать, и тогда строки

```
'не больше двух за раз'
'не больше двух зараз'
```

считались бы одинаковыми. Ну кому ж такое понравится?!

Существует особое значение строкового типа — *пустая строка*, т.е. строка нулевой длины. Такое значение возникает, например, когда пользователь ничего не вводит в ответ на запрос программы. Строковые литерал с таким значением записывается просто как два апострофа, один из которых «открывает» литерал, а второй тут же «закрывает»:

```
''
```

Иногда в состав строки нужно включить символ, которые проблематично записать внутри строкового литерала. Один из самых частых случаев — символ или символы, разделяющие строки в многострочном тексте. Для этих целей выбран очень простой подход:

```
'Строка, содержащая символы, которые'#9'проблематично записать явно'
```

В этом примере строковый литерал «закрывается», записывается `#9`, затем строковый литерал снова «открывается». Результатом такой записи будет строка, в состав которой входит символ с кодом 9 — символ горизонтальной табуляции. Нетрудно заметить, что запись `#9` напоминает один из способов записи символьных литералов. И да, как и в символьных литералах, здесь можно указывать код символа не только десятичным, но и 16-чным числом:

```
'Многострочный фрагмент'#13#10'текста'
'Многострочный фрагмент'#$0D#$0A'текста.'
```

Приведённые в примере литералы эквивалентны.

Существует общая для всех языков программирования, можно сказать фундаментальная, проблема: т.к. строковый литерал выделяется какими-либо символами (например, апострофами в Pascal/Delphi или кавычками в большинстве C-подобных языков), нужно договориться, что делать, если ограничивающий символ встречается и в самой строке. В Pascal/Delphi её решили очень просто: для включения апострофа в состав строкового литерала его необходимо удвоить (это действие ещё называется *экранированием символа*). Например:

```
'The problem is easily solvable, isn''t it?'
```

Такой литерал соответствует строке

> The problem is easily solvable, isn't it?

Обратите внимание, что в значение строки апостроф включается один раз.

Что было бы, если бы мы записали апостроф, не повторяя его?

```
'The problem is easily solvable, isn't it?'
```

Такой апостроф с точки зрения компилятора означал бы, что запись строкового литерала закончилась и дальше идут другие лексемы. Но ни `t`, ни `it`, ни вопросительный знак не могут встретиться в корректной программе на Pascal/Delphi сразу после строкового литерала. Кроме того, сразу после них идёт ещё один апостроф, который по правилам языка снова «открывает» строковый литерал, а значит, этот фрагмент программы нарушил бы баланс между «открывающими» и «закрывающими» апострофами. Проще говоря, мы увидели бы сообщение об ошибке от компилятора и, разумеется, вынуждены были бы её исправить.

А как у них?

А как эту проблему решают в других языках программирования? В C-подобных языках строковые литералы, как правило, ограничиваются двойными кавычками:

```
"Пример строкового литерала в C-подобных языках"
```

Соответственно, проблемным символом здесь является двойная кавычка `"`. Но создатели языка C решили не использовать подход языка Pascal и пойти своим путём: чтобы выполнить экранирование проблемного символа, перед ним ставится обратный слеш.

```
"Пример литерала с \"фрагментом в кавычках\" для C-подобных языков"
```

Кроме кавычек в этих языках могут экранироваться и некоторые символы латинского алфавита: в этом случае они приобретают особый смысл. Например:

```
"Строка 1\nСтрока 2\nСтрока 3"
```

Комбинация `\n` внутри строкового литерала в C-подобных языках используется для обозначения символа перехода на новую строку. Таких комбинаций немало: есть ещё `\t`, `\r`, `\a`, `\b` и т.д.

Этот подход создаёт массу проблем. Давайте, например, вспомним о таких строках, как пути к файлам. Скажем, на логическом диске `D:` у нас есть каталог, в котором есть текстовый файл:

```
D:\notes\textfile.txt
```

Казалось бы, совершенно безобидная строка. Давайте-ка запишем её как строковый литерал для языка C:

```
"D:\notes\textfile.txt"
```

Вы видите проблему? А если подсветить специальные комбинации символов вот так?

```
"D:\notes\textfile.txt"
```

Да-да, именно так. С точки зрения компилятора такая запись задаёт примерно следующий фрагмент текста:

> D:
> otes extfile.txt

Никаких сообщений об ошибке компилятор не выдаст: а что, если Вам действительно именно такой фрагмент текста и нужен? (Спойлер: скорее всего нет.)

Как здесь быть? Договорившись экранировать проблемный символ " символом \ создатели языка C не решили проблему от слова «совсем»: у нас был один проблемный символ — теперь их стало два! «Ну что же, — сказали они, — давайте тогда будем экранировать обратный слеш… обратным слешем». То есть для того, чтобы символ \ не воспринимался в составе литерала как экранирующий последующий символ, его нужно удваивать:

```
"D:\\notes\\textfile.txt"
```

Фу-у-ух, так-то лучше. «И пусть у нас теперь два проблемных символа вместо одного, но зато не как в Pascal'e!»

Ну, справедливости ради, в Pascal'e та же самая строка задавалась бы так:

```
'D:\notes\textfile.txt'
```

И никаких лишних символов. Критически настроенный читатель скажет: «Пф-ф-ф, невелика потеря!» Подождите, веселье только начинается…

В операционной системе Windows есть ещё один вид путей к файлу, так называемые «длинные пути» (long path). Чаще всего они используются, чтобы задать путь к файлу, расположенному на другом компьютере в сети. Скажем, если на компьютере `Server` открыт общий доступ к каталогу `Shared` и в нём есть файл `textfile.txt`, то путь к нему может выглядеть вот так:

```
\\?\UNC\Server\Shared\textfile.txt
```

Да-да, два обратных слеша в начале (чтобы операционная система могла отличить такой путь к файлу от других видов), потом вопросительный знак, потом снова обратный слеш и т.д. Таким должно быть значение строки. Как запишется строковый литерал с таким значением в языке C?

```
"\\\\?\\UNC\\Server\\Shared\\textfile.txt"
```

Внимательно пересчитайте слеши. И главное, компилятор в сущности не обязан выдавать сообщение об ошибке, если Вы пропустили или добавили лишний обратный слеш. Вы просто получите строку со значением, отличным от ожидаемого Вами. Искать ошибку будет вдвойне приятно: программа просто не найдёт файл, заданный такой строкой, поэтому почти наверняка Вы для начала перепроверите наличие подключения к сети, доступность файла из других программ, настройки доступа к нему, фазу луны и температуру тела императорского пингвина Василия, прежде чем догадаетесь, в чём ошибка.

Критически настроенный читатель не сдастся так просто и скажет: «Ну ладно, это всего лишь означает, что нужно внимательно писать пути к файлам и перепроверять их в первую очередь. И вообще, абсолютные пути к файлам — это прям фу-фу-фу!».

Контраргумент будет простым: движки регулярных выражений. Это специальные библиотеки для обработки текстовых данных, которые позволяют простые операции над строками выполнять сложно, неэффективно и неправильно, но пользуются огромной

популярностью у программистов. Типичный пример строки, которая может встретиться в программе, где используются эти библиотеки, выглядит примерно так:

```
/^(https?:\/\/)?([\da-z\.-]+)\.([a-z\.]{2,6})([\/\w \.-]*)*\/?$/
```

Ну что, проэкранируем обратными слешами? А всего-то нужно было ограничиться экранированием одного символа самим собой, как это сделали в Pascal.

Осознание ситуации пришло не сразу, но постепенно в разных C-подобных языках стали появляться разные интересности, пытающиеся залатать эту дыру. Например, в C# появились @-литералы:

```
@"The path is C:\Public\Report.doc"
```

Если пропустить символ `@`, литерал начинает вести себя примерно как в языке C. Зато если этот символ перед кавычками указан, литерал ведёт себя… так же, как в Pascal! Экранировать нужно только двойную кавычку, причём делается это так:

```
@"Строка с ""двойными кавычками"" записывается так"
```

Получились этакие строковые литералы в стиле Pascal, но на костылях: всё-таки двойные кавычки — символ обычно парный, в отличие от апострофов, так что экранировать придётся вдвое больше, да ещё `@` не забывать, а то как бы чего не вышло.

В языке JavaScript вообще сделали специальный вид литералов именно для регулярных выражений. Проблемы записи других строк это не решает, но хотя бы прикрывает самые неприятные случаи.

Создатели PHP и Ruby плакали, кололись, но продолжали есть кактусы: здесь есть строковые литералы в двойных кавычках, которые ведут себя, как в C, но с некоторыми дополнительными возможностями, и строковые литералы в апострофах, которые используют экранирование, как в C, но экранировать нужно только два символа: `'` и `\`.

Даже сравнительно новые языки, которые не проектировались как C-подобные, продолжают страдать теми же детскими болезнями. Например, в языке Python положение дел со строковыми литералами такое же, как в C# (или наоборот, т.к. Python всё же появился раньше): единственное отличие в том, что у более продуманного вида строковых литералов в Python вместо `@` ставится латинская буква `R` или `r`.

Строковые типы данных в Delphi

Исторически в языке Pascal было два строковых типа данных:

- строки постоянной длины;
- строки переменной длины.

В языке Delphi появился ещё один вид строк — динамические строки, или Delphi-строки.

Кроме того, для совместимости с языком C иногда использовался предопределённый тип PChar и набор стандартных подпрограмм для работы с такими строками, однако полноценным строковым типом C-строки не являются: вместо этого они представляют собой массивы, работа с которыми ведётся с помощью так называемых указателей, и PChar — это не что иное, как типа «указатель на Char». Этот вид строк, использовавшийся преимущественно в C и C++, в сочетании с популярностью этих языков

программирования стал одной из главных причин, по которой примерно в 90-е годы XX века человечество без преувеличения накрыла волна уязвимостей к так называемым атакам переполнением буфера, позволявшим злоумышленнику выполнять почти произвольные действие на компьютере жертвы. Повсеместный отказ от C-строк в пользу более продвинутых решений не решил проблему полностью, но снизил вероятность обнаружения таких уязвимостей в программе до минимума.

С появлением Unicode также появилось разделение строковых типов на те, значения которых состоят из классических символов типа Char, и те, которые используют различные вариации Unicode.

Таким образом, в современных версиях Delphi строковые типы данных представлены следующим набором типов:

- строки постоянной длины;
- Pascal-строки, или короткие строки (тип ShortString);
- динамические строки (типы AnsiString, WideString и UnicodeString).

Несмотря на их сходство каждый из этих типов данных целесообразно рассматривать отдельно.

Дополнительные вопросы

1. Предложите способ записи строковых данных в памяти компьютера. Какая информация необходима для того, чтобы полностью задать строку?

2. В ранних версиях языка Fortran для записи строковых литералов использовалась так называемая Hollerith-нотация:

```
35HAn example Hollerith string literal
```

в которой записывалась длина строки в символах, затем ставилась латинская буква `H` и записывались символы строки. В чём преимущества и недостатки этого подхода?

3. В некоторых языках программирования (например, PHP) есть специальные виды строковых литералов, в которых вместо символов `'` или `"` в качестве ограничителя используется произвольная последовательность символов, задаваемая программистом, — так называемые HereDoc- и NowDoc-строки. В чём преимущества и недостатки этого подхода?

4. Почему несмотря на очевидные преимущества подхода Pascal к экранированию символов в строковых литералах, разработчики C-подобных языков не убирают поддержку строковых литералов с экранированием в стиле языка C?