ТУБ №33:
Динамические строки

Для хранения любой величины в памяти компьютера необходимо и достаточно знать две вещи: где в памяти начинается запись этой величины и где она заканчивается.

На практике обычно удаётся обойтись меньшим объёмом информации. Например, для скалярных типов данных достаточно знать, где начинается участок памяти, используемый для хранения значения, т.к. размер этих типов определён на уровне языка и всегда можно от начала этого участка отсчитать заранее известное количество байт. С массивами ситуация аналогична: зная базовый тип и множество значений для индексов, нетрудно вычислить и размер такого массива.

Для так называемых коротких строк, по сути, ситуация аналогична: размер такой строковой переменной всегда на 1 байт больше, чем максимальная длина строки. Однако есть и проблема:

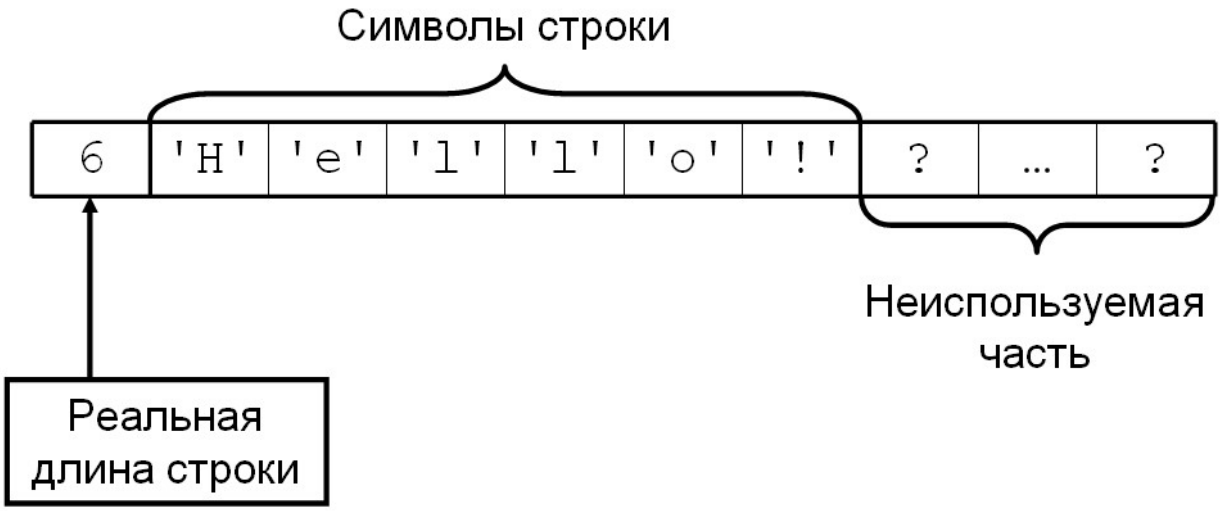
```
var
  S: String[200];

...

S := 'Вася';
```

Обратите внимание, что переменная занимает 201 байт, хотя реально для записи строки было бы достаточно 4 байт. Казалось бы, две сотни байт в наше время — совсем мелочь, но посмотрите на это с другой стороны: мы использовали для хранения строки в 50 с лишним раз больше, чем требовалось! Конечно, на одной переменной это не слишком ощутимо, но если их в программе становится много и программа активно занимается обработкой строковых данных, у нас проблемы.

Решение этой проблемы, конечно же, заключается в том, чтобы выделять память по мере необходимости. Такой подход ещё называется динамическим выделением памяти, но если мы переходим к его использованию, у нас возникает проблема: если раньше можно было ещё во время компиляции точно сказать, где и сколько памяти занимает строковая переменная, то теперь эта информация не только неизвестна во время компиляции программы, но ещё и меняется по ходу её работы.

Исторически сложилось два способа для записи в памяти строковых данных. Они получили свои названия в честь языков программирования, ставших наиболее популярными и применявших тот или иной способ.

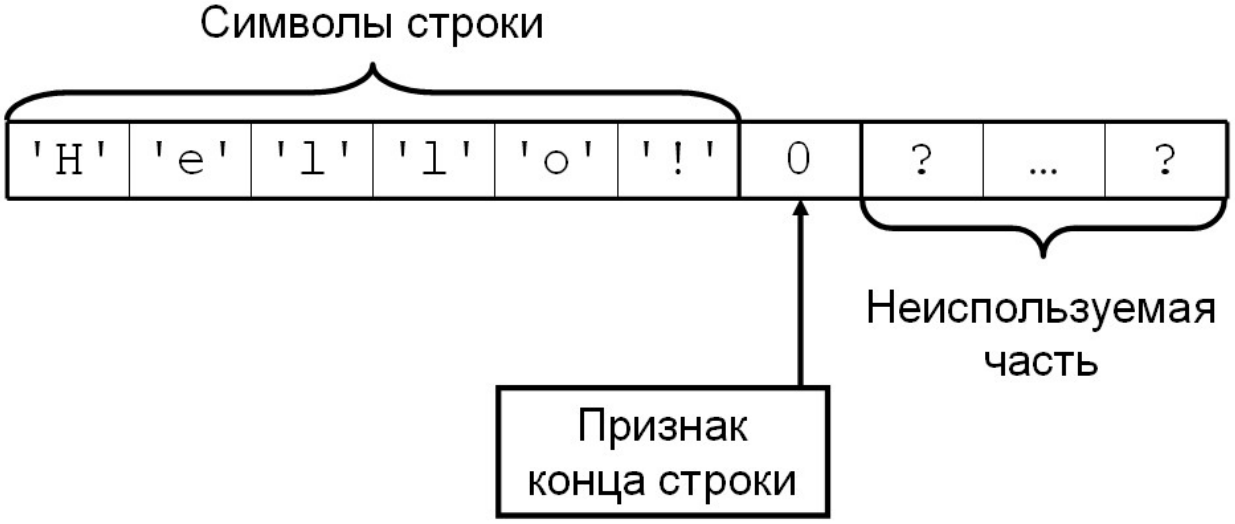
Первый способ Вам уже должен быть известен: это тот самый, который используется в коротких строках. Часто этот способ записи ещё называют *Pascal-строками*, P-строками или строками с упреждающей длиной.



Именно так устроен классический строковый тип в языке Pascal. При переходе к динамическому выделению памяти отличие будет заключаться только в том, что можно будет избавиться от неиспользуемой части, выделяя ровно столько, сколько требуется в данный момент.

Отличительной особенностью Pascal-строк является хранение длины строки перед началом строковых данных. Благодаря этому для работы со строкой достаточно лишь знать, где она начинается, а информация о том, сколько места эта переменная занимает, записана внутри самой переменной.

Другой способ записи используется в языке C и называется *C-строками*, нуль-терминированными или ASCIIZ-строками.



Как и в случае Pascal-строк, при динамическом выделении памяти можно будет избавиться от неиспользуемой части памяти.

Отличие C-строки от Pascal-строк заключается в том, как хранится информация о размере строки: вместо явного указания её длины считают, что строка заканчивается там, где впервые встретится символ с кодом 0, т.е. символ `#0`.

> Обратите внимание: символ `#0` — это НЕ то же самое, что символ-цифра `'0'`. Код нулевого символа `#0` равен 0, а код символа `'0'` равен 48, т.е. `'0'` эквивалентно `#48`.

В некоторых случаях вместо нулевого символа в качестве признака конца строки выбирают другое значение. Например, одна из функций операционной системы MS-DOS требовала, чтобы в качестве признака конца строки использовался символ `'$'`, однако идея хранения и принцип работы с такими строками от этого не меняются.

Обратите внимание, что для обоих способов записи строк достаточно знать, где строка начинается. Имея эту информацию, можно найти всю строку, определить её размеры и т.д.

Плюсы, минусы и немного истории

И у Pascal-, и у C-строк есть как преимущества, так и недостатки.

Явное преимущество Pascal-строк заключается в том, что для получения текущей длины строки достаточно просто взглянуть на первый байт переменной. Это одно-единственное обращение к памяти, и работает это достаточно быстро. Текущая длина строки нужна очень часто: конкатенация, выделение подстроки, удаление подстроки, поиск подстроки в строке и т.д. — во всех этих операциях приходится определять длину строки, чтобы выделить правильное количество памяти для результата, запустить цикл с правильным числом итераций и т.д.

В случае C-строк определение длины строки требует много времени. Для этого нужно считывать строки посимвольно и считать, пока не встретится нулевой символ. Чем длиннее строка, тем больше раз придётся обращаться к памяти и тем больше времени займёт вычисление её длины. А, как мы уже отметили, делать это приходится довольно часто. Говоря современным языком, «C-строки очень плохо масштабируются», т.е. резко замедляют работу программы при увеличении объёмов данных.

Представьте себе, что Вы работаете на железнодорожной станции и Вам нужно собрать из двух составов один. При этом необходимо найти такой путь, на котором новый состав поместится целиком, и убедиться, что такой путь вообще есть в наличии, и, если его нет, то, вероятно, искать другие варианты. Нет ничего хорошего в том, чтобы хвост поезда торчал за пределами станции: и другим поездам будет мешать проезжать, и датчики, управляющие включением/выключением светофоров на пешеходных переходах и на переездах, смущать будет.

Итак, очевидно, что надо определить, сколько же там вагонов всего. Если поезда комплектовались сторонниками подхода языка C, то Вам придётся самолично бежать от начала каждого состава до его конца, считая вагоны. Если же где-то там были адепты Pascal-подхода, то достаточно подойти к машинистам двух составов и спросить у каждого: «Сколько у Вас там вагонов-то?» Как говорится, почувствуйте разницу!

Кроме того, что хранение длины строки ускоряет работу с этими самыми строками, ещё одно преимущество Pascal-строк в том, что все символы абсолютно равны: нет такого символа, который нельзя было бы иметь в составе самого строкового значения. Что же касается C-строк, они не могут содержать символ, который используется как признак конца строки, потому что… он используется как признак конца строки и при попытке включить его в строку он просто «обрежет» её своим присутствием.

Так что же, Pascal-строки лучше во всём? Почти. Но недостаток всё же есть: ограниченность длины строки. Помните, под длину строки внутри переменной отводится ровно 1 байт? Это означает, что строки длиннее 255 символов (максимальное значение для одного байта) записать просто не получится. В C-строках такой проблемы нет: хоть миллионы символов можно, только чтобы нулевой по пути не встретился и памяти хватило.

Надо сказать, C-строки в своё время изрядно подпортили жизнь программистам. Какие ошибки могут возникнуть при записи строковых значений? Очевидно, в обоих случаях можно ошибиться (или, например, повредить при передаче) символы самой строки. Это неприятно, но если случилось в одном способе записи строк, то может случиться и в другом с той же вероятностью. Но что будет, если ошибиться с теми байтами, которые у Pascal- и C-строк используются по-разному?

Предположим, мы записали неправильное значение в первый байт Pascal-строки, тот, где хранится длина. В результате строка будет считаться короче или длиннее, чем есть на самом деле. При этом последствия такой ошибки затронут область памяти размером не более 256 байт.

Но что будет, если ошибиться с нулевым символом на конце C-строки? А вот здесь последствия будут непредсказуемыми. В зависимости от везения, фазы Луны и температуры тела пользователя следующий байт со значением 0 (который будет проинтерпретирован программой как нулевой символ) может встретиться как почти сразу, так и очень нескоро. Таким образом, последствия этой ошибки могут затронуть чуть ли не всю оперативную память. Теперь представьте себе, что программа вносит какие-либо изменения в строку посимвольно…

Примерно в 90-е годы XX века возросшая популярность C и C-подобных языков программирования, использовавших C-строки, привела к резкому росту сообщений об обнаружении в самых различных программах уязвимостей к так называемым атакам переполнением буфера. Такие уязвимости позволяют злоумышленникам наделать много всего интересного на компьютерах ничего не подозревающих пользователей. Устройство C-строк во многих случаях напрямую или косвенно выступало причиной уязвимости.

Как появились Delphi-строки

В середине 90-х годов XX века разработчики Borland приступили к разработке своего первого продукта, заточенного под разработку программ для ОС Windows, — Delphi. И здесь-то при проектировании строковых типов данных возникла непростая ситуация.

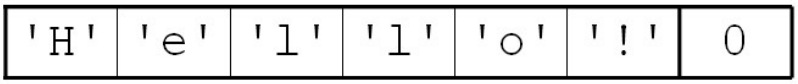
С одной стороны в языке Pascal, поддерживаемом их средой Borland Pascal, уже долгое время существовали и успешно применялись Pascal-строки. С ними была только одна проблема: оперативной памяти в компьютерах становилось больше, а процессоры научились работать уже с гигабайтами этой самой памяти, поэтому ограничение в 255 символов хотелось снять. Технически это было не так уж сложно: если отвести под длину строки 4 байта вместо одного, этого хватило бы, чтобы записать длину даже самой длинной строки, которую можно было записать в память.

С другой стороны для взаимодействия с Windows программам необходимо вызывать функции, предоставляемые самой операционной системой. А функции, которые принимают на вход строковые данные, ожидают увидеть… C-строку, т.е. строку, которая начинается сразу с кода первого символа и завершается символом с кодом 0.

Pascal-строки устроены иначе, поэтому, если понадобится передать строковое значение какой-либо функции Windows, программе нужно будет тратить время на преобразование из одного формата в другой — и это просто уничтожит выигрыш в производительности от использования Pascal-строк. Начать использовать в программах на Pascal (в первых версиях Delphi язык ещё назывался Object Pascal) C-строки? Но это уничтожит всю производительность просто на корню!

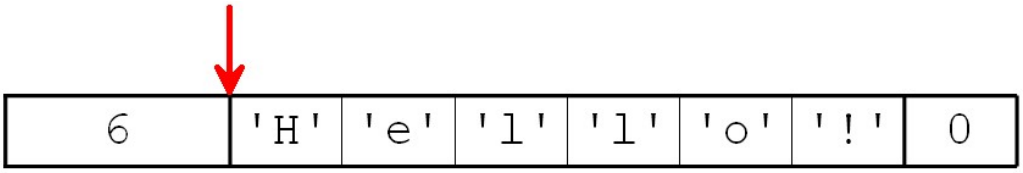
Конечно же, разработчики Delphi не зря ели свой хлеб и получали свои дипломы в вузах: решение нашлось, причём такое, что не только не растеряли того, что было, но и выиграли ещё больше.

Начали с неизбежного: если Windows ожидает от программ C-строки, давайте дадим их ей, чего бы нам это ни стоило:



Теперь, зная, где начинается строка, можно эту информацию передать Windows и она сможет обработать такую строку. Но как вернуть потерянную производительность?

«А давайте просто допишем перед началом строки её длину!» — сказали, наверное, разработчики Delphi и получили вот такую конструкцию:

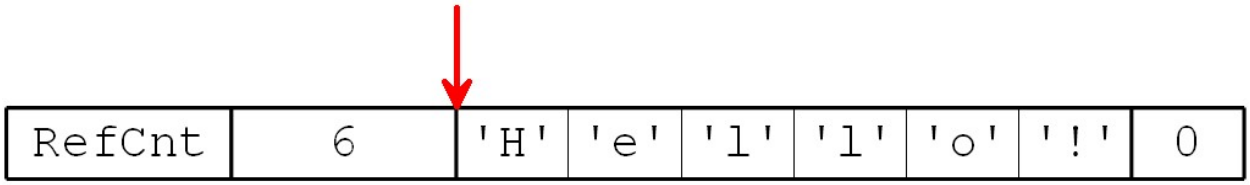


Местом, где начинается строка, по-прежнему считается расположение первого символа строки, именно эта информация передаётся от одной части программы к другой, она же передаётся функциям Windows. Но Windows думает, что ей передали информацию о расположении С-строки и для определения длины строки движется по ней только вперёд, пока не встретит нулевой символ, а вот программа, написанная на Delphi, «знает», что длина уже подсчитана и записана перед началом строки, поэтому может использовать её.

Чтобы не ограничивать искусственно длину строки, для её хранения отвели 4 байта, тем самым сняв ограничение полностью: строки длиной в гигабайт-другой — вообще большая редкость, а уж сплошной участок памяти такого размера, который бы был доступен для записи строки — и вовсе фантастика, поэтому теперь длина строки стала ограничена только оперативной памятью и тем, как в ней размещены другие данные и, разумеется, код самой программы.

Казалось бы, вот она — победа: И в быстродействии программ не проиграли, и совместимость с Windows обеспечили!

Но на этом разработчики Delphi не остановились. Проанализировав то, как обрабатываются строковые данные в программах, они обратили внимание, что очень часто программы выполняют присваивание строк, т.е. их копирование, в дальнейшем никак не изменяя их значений. Другими словами, в памяти хранится несколько копий одного и того же строкового значения, на создание которых, к тому же, было потрачено время. «Непорядок!» — сказали разработчики Delphi и прикрутили к новому виду строк ещё кое-что:



Дополнительные 4 байта перед началом собственно строковых данных стали использовать в качестве *счётчика ссылок*. Что это означает?

Теперь переменная, объявленная как

```
var
  S: String;
```

стала представлять собой не 256 байт с байтом длины в начале, как это было раньше, а просто 4 байта, задающие положение (по сути, порядковый номер) байта, в котором начинаются данные строки (см. красную стрелку на предыдущих иллюстрациях). В результате несколько переменных могут «ссылаться» на одну и ту же строку в памяти.

В четырёх же байтах счётчика ссылок, расположенных перед строкой, стали хранить ответ на вопрос о том, сколько таких переменных «ссылается» на неё. Это позволяет, с одной стороны, избежать лишних копирований строки, а с другой, освобождать занятую строкой память, когда строка становится ненужной.

Такие строки получили название *Delphi-строк*, *длинных строк* или *динамических строк*.

Механизм подсчёта ссылок используется не только в Delphi-строках, это вообще довольно популярный в программировании приём, позволяющий управлять любыми объектами в памяти, а не только строками. Идея проста: к объекту (строке или другим данным) «цепляется» целочисленная переменная — счётчик ссылок. Когда ещё одна переменная начинает ссылаться на тот же самый объект (строку), значение счётчика ссылок увеличивается на 1, когда одна из таких переменных перестаёт ссылаться на этот объект (строку) — уменьшается на 1. Если значение счётчика ссылок стало равным 0, это означает, что объект (строка) больше не нужен и занятую им память можно использовать для других целей.

Что-то подобное происходит с вещами у Вас дома. Вы идёте в магазин и покупаете новёхонький телевизор. Пока что он нужен только Вам, поэтому его счётчик ссылок становится равным 1. Вы приносите телевизор домой и оказывается, что Вашим родственникам или сожителям он тоже будет очень кстати — счётчик ссылок начинает расти. Со временем устройство начинает барахлить, на нём появляются следы многолетней эксплуатации, а ещё всех переводят с аналогового на цифровое телевещание, после чего и вовсе на смену телевидению приходит передача информации усилием мысли. Один за другим Ваши родственники или сожители заявляют: «Мне этот телевизор больше не нужен, можешь выбрасывать». Значение счётчика ссылок начинает уменьшаться. Пользоваться телевизором продолжаете только Вы и Ваш племянник (счётчик ссылок равен 2), а затем и Вам он становится без надобности (счётчик ссылок становится равным 1). Однажды Ваш племянник решает, что ему эта железяка тоже больше не нужна. Счётчик ссылок становится равным 0 — телевизор отправляется на свалку или на переработку. Его, конечно, жаль, но зато освободившееся место сможет занять Ваш кот или банки с маринованными огурцами, почему бы и нет?

Особый случай Delphi-строк — строковые константы. Они также занимают место в памяти, однако память для них выделяется не динамически, а сразу при запуске программы, поэтому освобождение памяти к ним неприменимо. Чтобы отличать их от обычных Delphi-строк, значение их счётчика ссылок устанавливается в число, эквивалентное –1: все биты равны 1. Для строки с таким значением счётчика ссылок подсчёт ссылок не ведётся: счётчик ссылок всегда остаётся равным –1.

Delphi-строки

Следует понимать, что к моменту, когда появилась Delphi, а в ней появились Delphi-строки огромное количество программ было написано на Pascal и использовали они, соответственно, Pascal-строки. Поэтому перед разработчиками Delphi стояла ещё одна задача: сделать так, чтобы старые программы можно было перекомпилировать в Delphi и они при этом продолжили работать правильно. Действительно, кто захотел бы переходить на среду разработки, для которой придётся переписать все свои проекты?

С этой задачей разработчики Delphi справились следующим образом. Во-первых, зарезервированное слово `String` теперь по умолчанию стало обозначать не короткие Pascal-строки, а новые Delphi-строки:

```
var
  S: String;        // Скорее всего, это Delphi-строка
```

Все операции, а также встроенные процедуры и функции начали работать с учётом нового внутреннего устройства строк. В частности, сохранилась нумерация символов в строке,

начинающаяся с 1, несмотря на то, что при новом устройстве Delphi-строк нумерация с нуля была чуть проще в реализации. Всю внутреннюю кухню, такую как дополнение строки нулевым символом на конце и изменение значения счётчика ссылок, взял на себя компилятор.

Переменные, объявленные с явным указанием максимальной длины, остались классическими Pascal-строками:

```
var
  S: String[100];  // Это по-прежнему Pascal-строка
```

Для подавляющего большинства программ этого оказалось достаточно. Проблемы возникли только с теми программами, которые полагались на то, что перед ними Pascal-строка.

Помните, что использовать обращение к `S[0]` — плохой способ определения длины строки `S`? Программы, которые так делали, оказались в числе тех, кто «поломался», ведь теперь при расчёте положения символа в памяти по индексу 0 оказывается не вся длина строки, а только один из 4 байт её длины, притом старший байт, который, скорее всего, равен 0. У таких горе-разработчиков было на выбор три варианта:

- исправить свои программы — этот способ подходил тем, у кого было время этим заниматься или количество «грязных хаков» со строками было невелико;
- заменить тип всех строковых переменных со `String` на `ShortString` — был введён новый тип `ShortString`, который эквивалентен классическому `String` без явного указания максимального размера, т.е. `String[255]`;
- в настройках компилятора выбрать режим, при котором стандартный идентификатор `String` ведёт себя как в классическом Pascal — при этом новые Delphi-строки в программе просто перестают существовать.

ANSI, Unicode и дальнейшее развитие Delphi-строк

Примерно в то же время готовились первые редакции стандарта Unicode, а в скором времени появились и первые версии Windows, в которых Unicode был основным способом кодирования символов. Это не могло не найти отражения в строковых типах данных Delphi.

Короткие строки навсегда остались такими, какими они были во времена MS-DOS и ранних версий Windows: 1 байт на символ, Pascal-строка. Что же до новоиспечённых длинных строк, то их стало несколько видов.

Изначально длинные строки хранили данные по 1 байту на символ и отличались от коротких строк только рассмотренными нами новшествами: счётчиком ссылок, счётчиком длины и нулевым символом на конце. Они получили название `AnsiString` и по умолчанию (если не включать настройку для старых программ) тип `String` считался эквивалентным `AnsiString`.

Примерно в это же время в Windows появляется похожий способ представления строк, который в Delphi получает название `WideString`. У него два отличия от `AnsiString`:

- нет счётчика ссылок, есть только длина строки и нулевой символ на конце;
- каждый символ занимает 2 байта (используется сначала кодировка UCS-2, а затем UTF-16).

С течением времени поддержка Unicode в Delphi расширялась, как, впрочем, и поддержка классических кодировок, существовавших до Unicode. На сегодняшний день это привело к следующим изменениям и дополнениям:

– Появился новый тип данных, относящийся к длинным строкам, — `UnicodeString`. Внутри он использует кодировку UTF-16, как `WideString`, но при этом имеет подсчёт ссылок, как `AnsiString`.
– В новых версиях Delphi тип `String` стал считаться эквивалентным именно `UnicodeString`, а не `AnsiString`, как это было в ранних версиях.
– Перед началом длинных строк добавилось ещё два поля по 2 байта каждый, задающие используемую кодировку и размер одного элемента.
– При объявлении строки типа `AnsiString` в новых версиях Delphi можно явно указать желаемую кодировку:

```
var
  S: AnsiString(1251);  // Строка в кодировке Windows-1251 (кириллица)
```

– Для удобства программистов в новых версиях Delphi также добавлены два новых предопределённых типа: `UTF8String` и `RawByteString`, объявленные следующим образом:

```
type
  UTF8String    = AnsiString(65001);  // Строка в кодировке UTF-8
  RawByteString = AnsiString($FFFF);  // Строка в произвольной кодировке
```

Подводя итоги

Итак, ещё раз кратко подведём итоги. В ранних версиях Delphi имеются следующие строковые типы данных:

– `String[N]` — короткие строки с максимальной длиной `N` символов;
– `ShortString` — короткие строки с максимальной длиной 255 символов (эквивалентны `String[255]`);
– `AnsiString` — длинные строки, 1 байт на символ, с подсчётом ссылок, без указания кодировки (не хранится);
– `WideString` — длинные строки, 2 байта на символ, без подсчёта ссылок, без указания кодировки (используется UCS-2 или UTF-16, не хранится);
– `String` — в зависимости от настроек компилятора эквивалентен либо `AnsiString` (по умолчанию), либо `ShortString` (в режиме совместимости с Pascal).

В поздних версиях Delphi строковые типы данных представлены следующим образом:

– `String[N]` — короткие строки с максимальной длиной `N` символов;
– `ShortString` — короткие строки с максимальной длиной 255 символов (эквивалентны `String[255]`);
– `AnsiString` — длинные строки, 1 байт на символ, с подсчётом ссылок, используется кодировка, выбранная в настройках Windows (хранится);
– `AnsiString(CodePage)` — длинные строки, с подсчётом ссылок, используется заданная кодировка (хранится);
– `WideString` — длинные строки, 2 байта на символ, без подсчёта ссылок, без указания кодировки (используется UCS-2 или UTF-16, не хранится);

– `UnicodeString` — длинные строки, 2 байта на символ, с подсчётом ссылок, используется кодировка UTF-16 (хранится);
– `UTF8String` — эквивалентен `AnsiString(65001)`, используется кодировка UTF-8 (хранится);
– `RawByteString` — эквивалентен `AnsiString($FFFF)`, кодировка не уточняется (хранится как `$FFFF`);
– `String` — в зависимости от настроек компилятора эквивалентен либо `UnicodeString` (по умолчанию), либо `ShortString` (в режиме совместимости с Pascal).

Таким образом, основной строковый тип данных — `String` — представляет собой универсальный тип данных, который аналогично `Integer`, подстраивается под актуальные возможности языка программирования и платформы. В большинстве случаев, когда конкретное представление строки в памяти (в т.ч. кодировка) не имеют значения, следует использовать именно его.

В случае, если представление строки в памяти (в т.ч. кодировка) имеют значение, следует применять другие типы. При записи данных в файл следует помнить, что переменная одного из типов, относящихся к динамическим строкам, хранит не саму строку, а информацию о её расположении в памяти.

Дополнительные вопросы

1. Почему длинные строки не были реализованы в Pascal?

2. Можно ли использовать Delphi-строки, в состав которых входит символ с кодом 0? В каких случаях это сработает, а в каких нет?