ТУБ №32:
Строки постоянной и переменной длины

Самый простой строковый тип данных в Delphi — строки постоянной длины. Под постоянной длиной понимается, что длина строки, записанной в переменную такого типа данных, не может изменяться не только во время работы программы, но и между запусками программы. Впрочем, чтобы стало действительно понятно, надо просто показать, как объявляются переменные типа «строка постоянной длины»…

Строки постоянной длины — это не что иное, как массивы типа Char. Пример объявления:

```
var
  S: array [1..14] of Char;
```

Да, всё настолько просто. И да, причина, почему длина остаётся неизменной, заключается в том, что она уже прописана прямо в тексте программы как количество элементов массива.

Критически настроенный читатель скажет, что так можно про любой объявленный программистом тип отдельный учебник написать, мол, подумаешь массив `Char`'ов!

Действительно, строки постоянной длины имеют все те же свойства, что и другие массивы. В частности, можно выполнять присваивание полных переменных, если размеры массивов одинаковы, можно обращаться к отдельным элементам (в данном случае это будут элементы типа Char, т.е. символы) и т.д.

Тем не менее, есть и два важных отличия. Во-первых, переменной типа «строка переменной длины» можно присваивать строковые литералы той же длины:

```
S := '7123456A001PB5';
```

С массивами других типов такое не сработает. Во-вторых, переменные такого типа можно сравнивать:

```
var
  S1, S2: array [1..5] of Char;

...

S1 := 'abcde';
S2 := 'abcdf';

...

if S1 < S2 then
  WriteLn('Всё в порядке')
else
  WriteLn('Что-то пошло не по плану');
```

Сравнение строк постоянной длины заключается в сравнении символов на соответствующих позициях по порядку. Большей считается та строка, у которой раньше встретится символ с большим значением кода символа.

Как Вы должны помнить, характеристиками, определяющими тип данных, являются множество значений и множество операций. Как видим, *множество операций у строк*

постоянной длины отличается от множества операций над другими видами массивов, а значит, этот вид строк может считаться самостоятельным типом данных.

Нетрудно догадаться, что строки постоянной длины применяются сравнительно нечасто ввиду ограниченности возможностей. Тем не менее, и у них есть важные преимущества перед другими видами строк. Во-первых, из-за фиксированного размера и очень простого внутреннего устройства (заранее известное количество записанных подряд символов) их удобнее использовать при обмене с другими программами, в том числе записывать в файл, передавать по сети и т.д. Во-вторых, работа с ними ведётся несколько быстрее, т.к. компилятор может, зная длину строки, применять дополнительные оптимизации, что особенно полезно при обработке больших объёмов данных.

Строки переменной длины

При всех преимуществах строк постоянной длины для эффективности программ проблемы их использования проявляют себя достаточно быстро. Например, даже обычный ввод значения с клавиатуры оказывается нетривиальной задачей, т.к. следующий фрагмент кода:

```
type
  TCustomString = array [1..16] of Char;

var
  S: TCustomString;

...
ReadLn(S);
...
```

просто не скомпилируется, а ошибка будет говорить о том, что ввод значений такого типа не поддерживается процедурами Read/ReadLn.

Вообще говоря, это вполне логично, т.к. программа, заставляющая пользователя вводить ровно заданное количество символов (не больше и не меньше) была бы довольно сомнительным решением в смысле удобства, а обрезать введённое пользователем значение до требуемой длины, если введено слишком много символов, или заполнять каким-либо символом (и ещё вопрос каким именно!) недостающие позиции, если введено недостаточно символов, — тоже не лучшие, или во всяком случае не универсальные идеи. Поэтому строки постоянной длины используются преимущественно для хранения данных, используемых программой для внутренних нужд, и с точки зрения программиста представляют собой просто массивы с дополнительными возможностями.

На практике же куда чаще используются строковые типы данных, способные хранить строки заранее неизвестной длины. Один из таких типов данных — так называемые строки переменной длины. В Delphi этот тип данных ещё стали называть `ShortString`, или *короткими строками*, и совсем скоро станет понятно, почему. Синтаксис задания такого типа данных следующий:

Необязательная величина <N> в квадратных скобках — это целочисленное выражение, задающее максимальную длину строки. Его значение может быть в диапазоне от 1 до 255 включительно.

В Pascal, если часть с квадратными скобками пропущена, максимальная длина считается равной 255. В Delphi всё немного сложнее: переменная, объявленная как `String` без задания максимальной длины может быть как эквивалентной `String[255]`, так и совсем другим видом строк, в зависимости от настроек компилятора. Подробнее этот вопрос целесообразно рассматривать после обсуждения так называемых динамических строк, или Delphi-строк.

Вернёмся к строкам переменной длины. Ограничение, накладываемое на максимальную длину, отнюдь не случайно и связано со внутренним устройством переменной этого типа.

> О чём может говорить ограничение в 255 символов у строк переменной длины?

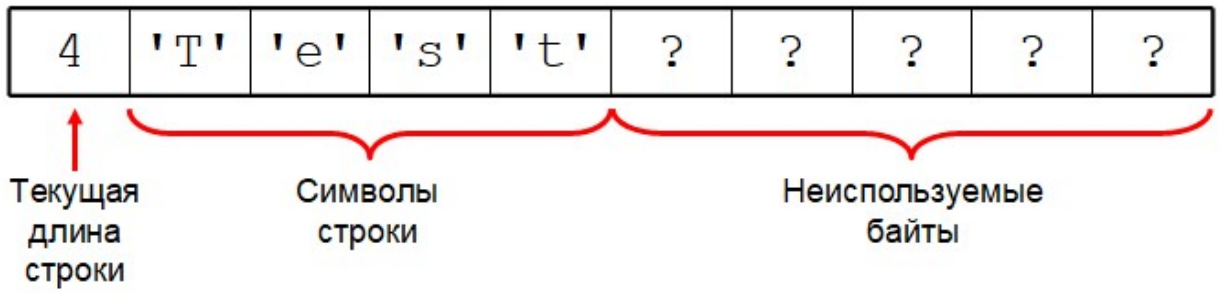
Рассмотрим *представление в памяти строк переменной длины*. Предположим, переменная объявлена следующим образом:

```
var
  S: String[9];
```

Присвоим ей какое-либо значение:

```
S := 'Test';
```

Тогда в памяти такая переменная будет занимать 10 байт:



На приведённом рисунке каждая ячейка соответствует одному байту в памяти. Первый байт всегда содержит *текущую длину строки*, т.е. количество символов в том фрагменте текста, который в данный момент хранится в строковой переменной. Текущая длина, как нетрудно догадаться, может принимать значения от 0 (если строка пустая) до максимальной длины (в нашем примере — 9). Самое большое число, которое можно записать в один байт, — это число 255. Отсюда и ограничение максимальной длины: для строки длиннее 255 символов просто не получилось бы правильно записать её текущую длину.

255 символов должно хватить всем?

Может показаться, что 255 символов — это катастрофически мало. В самом деле, ведь можно было бы отвести для хранения текущей длины строки, например, 2 байта и

получить максимальную длину в «фантастические» 65535 символов. Почему так не было сделано?

Чтобы полностью разобраться с этим вопросом, необходимо обратиться к истории. Язык Pascal, из которого перекочевали в Delphi строки переменной длины, с самого начала своего существования использовался на самых разных компьютерах от производителей всех мастей. Однако у них у всех было кое-что общее — ограниченные объёмы оперативной памяти.

Рассмотрим этот вопрос на примере компьютеров, которые со временем стали самыми популярными и потомками которых мы пользуемся по сей день (у остальных всё было примерно так же). Младшая модификация самой первой модели IBM PC-совместимых компьютеров могла похвастаться 16 КБ оперативной памяти примерно за $1500. Те, кто умел экономить на еде, одежде и общественном транспорте, могли позволить себе модель с 64 КБ памяти примерно за $3000. Нужно больше? Без проблем, просто докупите специальную плату расширения.

Со временем, конечно, ситуация менялась, но до 90-х гг. XX века возможности, предоставляемые процессорами Intel, были таковы, что комфортнее всего чувствовали себя программы, способные полностью уместиться в один или несколько участков не более 64 КБ каждый, а суммарно без специальных ухищрений для своих данных можно было использовать до 640 КБ (чуть меньше) на всех, включая операционную систему, драйверы и т.д. Одна-единственная переменная типа `String[65535]` заняла бы целых 64 КБ, а ведь для обработки текстовых данных она бы понадобилась не одна!

Зайдём с другой стороны. Рано или поздно текстовые данные отображаются пользователю. Самые ходовые характеристики того времени — текстовый режим 80×25 знакомест, т.е. экранных позиций, областей, в каждой из которых может быть отображён один символ. Строка длиной 255 символов, если задаться целью вывести её на экран целиком, занимала бы больше 4 строк. На практике же, как Вам наверняка известно, программы, работающие с большими текстами (например, текстовые редакторы) слишком длинные строки либо разбивают на более короткие, либо отображают горизонтальную полосу прокрутки, и одновременно на экране всё равно отображается ограниченное количество символов. Внутренняя обработка таких данных также обычно сводилась к работе с короткими фрагментами текста.

Подводя итог, можно сказать, что поддержка строк длиннее 255 символов была с одной стороны проблематичной, а с другой — не слишком востребованной. Разумеется, как только снятие этого ограничения стало легко реализуемым, появились новые строковые типы данных. Однако и по сей день редко можно встретить программу, для которой такое ограничение было бы критичным.

Операции над строками переменной длины

Что интересного можно сделать с двумя строками? Ну, например, склеить две строки в одну:

```
var
  S, S1, S2: String[20];

...

S1 := 'Вася';
S2 := 'Маша';

S := S1 + S2;      // Теперь в S записано 'ВасяМаша'
S := S2 + S1;      // А сейчас — 'МашаВася'
```

Операция обозначается вполне логичным образом — символом +, — а её результатом является значение строкового типа. Название этой операции — *конкатенация*. В отличие от обычного арифметического сложения здесь результат изменяется при перестановке «слагаемых».

Но что будет, если длина получившейся строки окажется больше, чем максимальная длина переменной, в которую помещается результат?

```
var
  S, S1, S2: String[20];

...

S1 := 'Василий Иванович';
S2 := 'Мария Петровна';

S := S1 + S2;      // Теперь в S записано 'Василий ИвановичМари'
S := S2 + S1;      // А сейчас — 'Мария ПетровнаВасили'
```

Таким образом, при записи слишком длинного строкового значения в переменную этого типа оно будет обрезано до максимальной допустимой длины. Это следует иметь в виду при работе с короткими строками и внимательно оценивать длины используемых в программе строковых значений.

Кроме склеивания строк, их можно сравнивать. Но в отличие от строк постоянной длины здесь требуется уточнение того, как сравнить строки разной длины. Документация говорит следующее:

> Comparison of strings is defined by the ordering of the characters in corresponding positions. Between strings of unequal length, each character in the longer string without a corresponding character in the shorter string takes on a greater-than value. For example, "AB" is greater than "A"; that is, 'AB' > 'A' returns True. Zero-length strings hold the lowest values.

Или в переводе на русский язык:

> Сравнение строк опирается на сравнение символов в соответствующих позициях. В случае строк различной длины символы более длинной строки, для которых в более короткой строке нет соответствующего символа, считаются имеющими большее значение. Например, "AB" будет больше "A", т.е. `'AB' > 'A'` вычислится в значение `True`. Строки нулевой длины являются наименьшим значением.

Порядок, который устанавливается между строками при таком сравнении, ещё называют лексикографическим. По такому же принципу размещаются слова в словарях, за исключением того, что в программировании строка может состоять не только из букв.

Наконец ещё одно действие, без которого работа со строками была бы не слишком удобной, — обращение к отдельным символам строки. Для этого используется тот же синтаксис, что и для обращения к массиву, — указание индекса в квадратных скобках:

```
S := 'ОСЕНЬ';
S[2] := 'Л';        // В переменной S теперь значение 'ОЛЕНЬ'
```

Отдельный символ строки имеет тип `Char` и над ним можно выполнять те же действия, что и над другими значениями символьного типа.

Нумерация символов в строке начинается с 1. При этом следует иметь в виду, что корректными значениями индекса будут значения, не превышающие длины строки.

Обращение к несуществующему элементу строки аналогично выходу за границу массива и является грубой ошибкой.

Нетрудно заметить, что нумерация символов строки, начинающаяся с 1, идеально соответствует внутреннему устройству короткой строки: для обращения к `i`-му элементу строки достаточно просто отступить от её начала `i` байт. Это легко и эффективно реализуется в машинном коде, сохраняя при этом естественность нумерации для человека.

Встроенные процедуры и функции над строками переменной длины

Для всех прочих действий над строками, помимо конкатенации, сравнения и обращения к элементу строки, естественных и очевидных обозначений знаками операций не нащлось. Действительно, каким символом, например, обозначить операцию копирования фрагмента строки? Поэтому для выполнения подавляющего большинства действий над строками в Pascal/Delphi применяются процедуры и функции, в том числе встроенные.

```
Copy(S, From, Count)
```

Начнём с уже упомянутого действия — копирования фрагмента строки. Функция `Copy` предназначена как раз для этого и позволяет выделить произвольную *подстроку*, т.е. часть другой строки. Для этого первым параметром указывается строка, из которой необходимо выделить подстроку, вторым — порядковый номер первого выделяемого символа, третьим — количество символов. Например:

```
S := 'ЧЕРТЁЖИК';
S1 := Copy(S, 5, 4);          // Теперь S1 содержит строку 'ЁЖИК'
```

Если параметры выбраны так, что требуется выделение большего количества символов, чем есть в исходной строке, выделяется только та часть, которая имеется в наличии:

```
S := 'ПАНТОГРАФ';
S1 := Copy(S, 6, 10);         // Теперь S1 содержит строку 'ГРАФ'
```

Следующая функция для работы со строками, встроенная в язык, — функция `Concat`:

```
Concat(S1, S2, ..., Sn)
```

Результатом работы функции является конкатенация всех строковых значений, переданных в качестве параметров. Например:

```
S1 := 'ПАНТОГРАФ';
S2 := Copy(S1, 1, 5);          // Теперь S2 содержит строку 'ПАНТО'
S3 := Copy(S1, 2, 1);          // Теперь S3 содержит строку 'А'
S := Concat(S2, 'МИМ', S3);    // В переменную S записывается 'ПАНТОМИМА'
```

У внимательного читателя должен возникнуть закономерный вопрос: «Почему бы не использовать обычную операцию конкатенации?» Например, так:

```
S := S2 + 'МИМ' + S3;
```

Действительно, результат будет точно таким же. Однако следует иметь в виду, что функции эти появились ещё в языке Pascal в те времена, когда компиляторы ещё не отличались особой сообразительностью ввиду ограниченности ресурсов компьютеров — быстродействия и оперативной памяти.

Для того, чтобы склеить две строки, необходимо найти свободный участок памяти, куда можно будет поместить результат. Размер такого участка должен быть равен сумме длин склеиваемых строк. Когда он найден, туда нужно по порядку скопировать сначала все символы первой строки, потом — второй.

Представим себе, что вариант с использованием операции конкатенации обрабатывает очень простой компилятор. Тогда склеивание строк будет происходить в два этапа: сначала склеиваем две строки, потом результат склеиваем с третьей. И каждый раз ищем свободное место в памяти, и каждый раз копируем.

«Но ведь можно сразу выделить память под результат склеивания всех строк!» Именно. Современные оптимизирующие компиляторы постараются так и сделать, если это будет полезно для производительности. А вот компиляторам постарше или просто попроще (например, для новых аппаратных платформ) понадобится явное указание, что Вы склеиваете сразу много строк.

Компилятор Delphi оптимизировать умеет, поэтому особой разницы Вы не заметите, но по историческим причинам функция осталась: чтобы программы, написанные раньше, не остались без используемой ими функции.

```
Length(S)
```

Поскольку короткие строки — это строки переменной длины, время от времени бывает нужно определить, какую же длину имеет строка в данный момент. Функция `Length` как раз для этого и нужна. Её результат — текущая длина строки, в символах.

Наблюдательный читатель скажет: «А что, если просто обратиться к нулевому элементу строки?»

```
var
  S: String[40];
  L: Integer;

...

S := 'Грязные приёмчики наказуемы!';

...

L := Ord(S[0]);   // Теперь в переменной L будет длина строки. Или нет???
```

Действительно, с короткими строками этот трюк сработает, ведь, как было отмечено ранее, индекс символа используется просто как количество байт, на которое нужно сместиться от начала строки, чтобы найти требуемый символ. НО! Это недокументированный приём, гарантий его работоспособности никто и никогда не давал.

Собственно, однажды те разработчики, которые использовали этот трюк, уже поплатились за него: внутреннее устройство строкового типа изменилось — и их программы перестали работать. Но подробнее мы поговорим об этом при обсуждении так называемых динамических строк, или Delphi-строк.

Таким образом, правильный способ получения длины строки будет таким:

```
L := Length(S);
```

Что ещё можно делать со строками? Ну, например, проверить, входит ли в строку заданная подстрока и, если да, то начиная с какой позиции.

```
Pos(Substr, S)
```

Функция `Pos` возвращает индекс символа, с которого начинается первое вхождение подстроки `Substr` в строку `S`. Например:

```
S := 'ПРОКРАСТИНАЦИЯ';
WriteLn(Pos('НАЦИЯ', S));     // Выведет 10
```

Но что эта функция должна вернуть, если заданная подстрока не встречается в заданной строке?

```
WriteLn(Pos('Ы', S));         // Выведет 0
```

Действительно, если подстрока найдена, то позиция её вхождения в строку будет положительным числом (нумерация символов начинается с 1, помните?). И главное, никогда не будет равна 0. В этом ещё один плюс нумерации с 1: можно использовать значение 0 для обозначения особых ситуаций, как в нашем примере.

```
Delete(S, From, Count)
```

Помимо встроенных функций для работы со строками имеется и несколько встроенных процедур. Одна из них — `Delete` — позволяет вырезать (т.е. удалить) фрагмент строки. Параметры процедуры аналогичны параметрам функции `Copy`, которую мы уже видели, только там мы задавали фрагмент, который хотим скопировать, а здесь — фрагмент, который хотим удалить.

```
S := 'КОРЗИНА';
Delete(S, 3, 3);     // Теперь в S записана строка 'КОРА'
```

Если позиция, с которой следует начинать удаление, выходит за границы строки, строка остаётся неизменной:

```
S := 'КАРТОНКА';
Delete(S, -1, 4);    // В переменной S по-прежнему строка 'КАРТОНКА'
Delete(S, 9, 2);     // И сейчас тоже ничего не изменилось
```

Ситуация, когда длина удаляемого фрагмента больше количества символов, оставшихся до конца строки, тоже обрабатывается корректно, удалением только имеющихся символов:

```
S := 'Картина, корзина, картонка и маленькая собачонка';

Delete(S, 28, 60);  // Теперь в S записано 'Картина, корзина, картонка'
```

Любой, кто хоть раз в жизни пользовался текстовым редактором, знает, что кроме удаления фрагмента текста иногда бывает нужно в уже имеющийся текст выполнить вставку другого фрагмента текста. Встречаем…

```
Insert(Substr, S, From)
```

Процедура `Insert` вставляет строку `Substr` в строку `S` так, чтобы первый вставленный символ оказался на позиции `From`. Например:

```
S := 'ЧАС';
Insert('К НОРРИ', S, 3);     // Теперь в переменной S строка 'ЧАК НОРРИС'
```

Если позиция вставки меньше 1, она считается равной 1, т.е. подстрока будет вставлена в самое начало строки:

```
S := 'ВЕС';
Insert('НА', S, 0);          // Теперь в переменной S строка 'НАВЕС'
```

Если позиция вставки больше длины принимающей строки, вставляемая подстрока добавляется в конец строки:

```
S := 'ВЕС';
Insert('НА', S, 10);          // Теперь в переменной S строка 'ВЕСНА'
```

Ещё две процедуры, которые часто применяются при работе со строками, — процедура преобразования строки в число и процедура преобразования числа в строку.

```
Str(Value, S)
```

Процедура `Str` формирует и записывает в переменную `S` строку, значение которой — десятичная запись числа `Value`.

```
Str(42, S);        // Теперь в S строка '42'
```

Процедура `Str` может преобразовывать как целые числа, так и вещественные. При её вызове для числового параметра можно указать желаемый формат точно так же, как и при вызове `Write` и `WriteLn`:

```
Str(Pi:0:4, S);    // Теперь в S строка '3.1416'
```

Обратное преобразование выполняет процедура `Val`:

```
Val(S, Value, ErrorCode)
```

Обратите внимание, что не любая строка может быть преобразована в число. Скажем, какому числу соответствует строка `'Delphi'`? Если преобразование прошло успешно, переменная, указанная третьим параметром (она должна быть целочисленной), примет значение 0, а во второй параметр будет записан результат преобразования:

```
Val('123.45', X, Code);        // Теперь Code = 0, X = 123.45
```

Если же преобразование не удалось, в третий параметр будет записан индекс проблемного символа:

```
Val('38 попугаев', X, Code);  // Теперь Code = 3
```

В данном примере преобразование пришлось остановить на пробельном символе, имеющем индекс 3. Обратите внимание, что документация ничего не говорит о том, какое значение примет в нашем случае переменная `X`. На практике можно увидеть там, например, результат преобразования предшествующих символов, но это не гарантируется. Поэтому сначала необходимо проверять код ошибки и только если он равен 0, использовать переменную с результатом преобразования:

```
Val(S, X, Code);
if Code = 0 then
  WriteLn('Результат: ', X)
else
  WriteLn('Ошибка преобразования! Значение переменной X не определено.');
```

Часто бывает, что строка формируется в программе посимвольно или небольшими фрагментами. Если решать такие задачи с использованием конкатенации, помимо собственно копирования новых символов в конец формируемой строки время будет тратиться на обновление счётчика её длины.

```
SetLength(S, NewLength)
```

Процедура `SetLength`, применённая к короткой строке, позволяет принудительно задать ей длину. Разумеется, новая длина не должна превышать максимальной длины, заданной при объявлении переменной. Для коротких строк SetLength просто записывает новое значение в байт, хранящий текущую длину строки. Значения символов на позициях, которые ранее не входили в состав строки, не определены, т.е. могут оказаться любыми, поэтому следует не забывать их проинициализировать.

Если длина формируемой строки заранее известна, можно вызвать `SetLength` для переменной, в которой она будет формироваться, после чего вместо полноценной конкатенации просто выполнять запись напрямую в конкретные элементы строки.

```
SetString(S, Data, Len)
```

Ещё одна процедура, добавленная в Delphi, — SetString. Она позволяет произвольный участок памяти превратить в значение строковой переменной, по сути — скопировать в строковую переменную произвольную последовательность байтов в памяти. Поскольку используется она сравнительно нечасто, а для грамотного использования необходимо

иметь представление о так называемых указателях, её рассмотрение на данном этапе нецелесообразно.

Дополнительные вопросы

1. Операции сравнения определены над строками постоянной длины (по сути, над массивами с базовым типом `Char`) и основаны на поэлементном сравнении кодов символов. Почему операции сравнения не определены аналогичным образом для массивов с другими базовыми типами?

2. Написать программу, которая для введённой пользователем строки выводит информацию о том, сколько раз в ней встречается буква А.

3. Пользователь вводит строку, состоящую из слов, разделённых пробелами. Выделить отдельные слова и вывести их на экран в том же порядке, но предварительно переставив в каждом слове буквы в обратном порядке.