

Д. Ю. Усенков

ИНТЕРЕСНО ПРОГРАММИРОВАНИЕ - ИГРЫ С ТЕКСТОМ

Москва – 2020

12+

Дмитрий Усенков

**Занимательное программирование
– игры с текстом**

«ЛитРес: Самиздат»

2020

Усенков Д. Ю.

Занимательное программирование – игры с текстом /
Д. Ю. Усенков — «ЛитРес: Самиздат», 2020

В пособии рассмотрены алгоритмы обработки текстовой информации (отдельных символов, слов и строк) средствами языка программирования Паскаль. В том числе рассмотрены принципы использования для обработки текстовых строк такого редко используемого типа данных как множества. Для школьников, учителей информатики и всех, самостоятельно изучающих программирование.

Содержание

Введение	5
О знаках и строках	6
Задача 1	12
Конец ознакомительного фрагмента.	15

Дмитрий Усенков

Занимательное программирование

– игры с текстом

Введение

При изучении в школе основ программирования по какой-то неизвестной авторам данной статьи причине работа с множественными типами данных (кроме разве что массивов и в некоторых случаях – файлов) почти не затрагивается. По крайней мере, найти в различных публикациях или на сайтах образовательной тематики какие-либо хорошие (интересные и вместе с тем поучительные) задачи оказалось довольно сложно.

Желая «сломать сложившиеся стереотипы», предлагаем читателям несколько подобных задач. Решение этих задач разберем на языке Паскаль.

Наши цели при этом:

- во-первых, показать читателям (учащимся, а также учителям информатики, которые смогут затем передать эти знания своим ученикам) принципы обработки строковых данных;
- во-вторых, продемонстрировать применение некоторых возможностей Паскаля по работе со строками и символами;
- в-третьих, показать преимущества использования такого редко используемого типа данных, как множества (как показывает практика, многие попросту не знают, как можно использовать множества в реальном программировании, помимо «чисто теоретического» решения задач на пересечение, объединение и пр. множеств).

Итак, начнем...

О знаках и строках

Как мы знаем из курса информатики, текст в компьютере представлен в виде последовательности кодов составляющих его символов – букв (латинских и строчных), знаков препинания, знаков математических операций и пр., а также специальных кодов, не имеющих отдельного визуального представления в виде символов и служащих для управления размещением текста (пример – коды табуляции, перехода на новую строку и т.д.). При этом соответствие между конкретным символом и его кодом устанавливается согласно *таблицам кодирования символов*, где для символов национальных алфавитов (к которым относится и кириллица) могут использоваться различные 8-битовые таблицы кодирования (ASCII для MS-DOS, КОИ-8, Windows и др.) либо все такие символы объединены в 16-разрядной таблице кодирования стандарта Unicode.

Таким образом, каждый символ текста в памяти компьютера занимает один (или два – для Unicode) байта и хранится там в виде целого беззнакового числа. Поэтому, чтобы компьютер «не путал» их с обычными целыми числами, в языках программирования высокого уровня, как правило, для символьных и строковых типов данных предназначены отдельные, особые типы данных.

В языке Паскаль это:

- символьный тип **char**, предназначенный для представления одного какого-либо символа; символьная константа записывается в апострофах, например: 'a', '0', '+' и т.д.;

- множественный (составной, структурированный, сложный) тип **string**, предназначенный для представления целых текстовых строк; строковая константа также записывается в апострофах, например: 'Строка'.

При этом прослеживается иерархия типов: множественный тип **string** можно рассматривать как некий набор данных типа **char** (что отражает вполне очевидный факт – строка текста состоит из символов).

Определение обоих этих типов данных (как и большинства других) в языке Паскаль производится в разделе **var**:

- для символьного типа данных – **var <переменная> : char;**

- для строкового типа данных – **var <переменная> : string;**

При этом строка может определяться как без указания ее размера (как продемонстрировано выше), так и с явным указанием ее длины (см. рис. 1):

var <переменная> : string[<длина>];

В подобном случае параметр <длина> представляет собой целое число, указывающее максимально допустимую длину строки, записываемой в такую переменную. Фактически же этот параметр указывает компьютеру, что для хранения такой переменной необходимо отвести указанное количество ячеек памяти для символов строки. Кроме них, в памяти также резервируется еще одна ячейка для хранения реальной длины строки, записанной в такую переменную: эта строка по длине может быть меньше, чем зарезервированная длина строковой переменной (и даже может быть пустой – не содержать символов вообще!), тогда часть зарезервированных ячеек памяти попросту не используется. А вот при попытке записать в строковую переменную значение (строку), длина которой превышает объявленную длину строковой переменной, приведет к тому, что в этой переменной уместится только заявленное количество символов начала строки, а всё остальное будет отброшено.

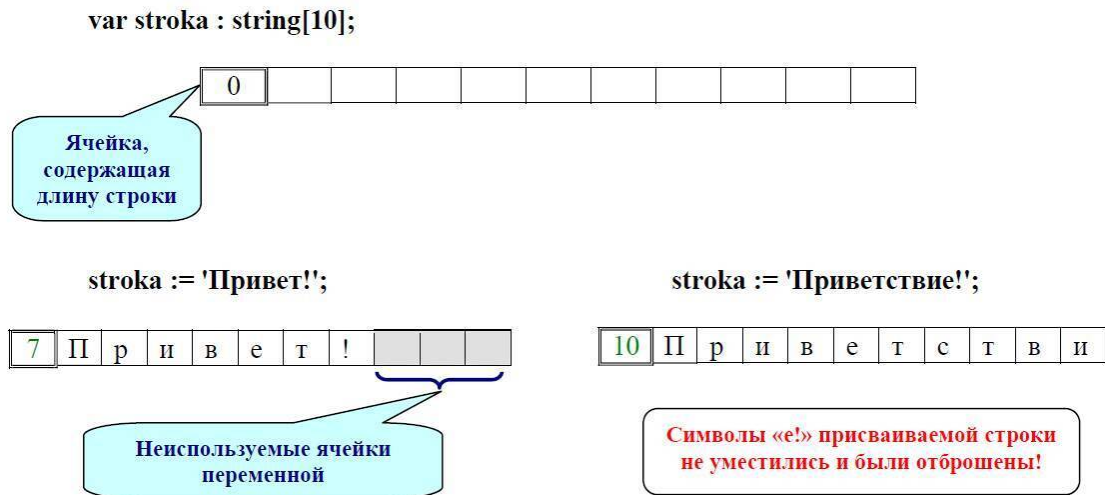


Рис. 1

Следует также отметить, что в случае, когда мы не указываем размер определяемой в разделе **var** строковой переменной, его всё равно определяет сама система программирования. В этом случае максимально допустимая длина строки составляет 255 символов, т.е. в памяти компьютера под такую строковую переменную резервируется 256 ячеек (одна, как и раньше, для хранения реальной длины хранящегося в этой переменной строкового значения). Все рассуждения о ситуациях, когда такой переменной присваивается более короткая или, наоборот, более длинная строка, тоже при этом остаются в силе.

Массивы символов и строк в языке Паскаль также возможны. Их определение и работа с ними осуществляются точно так же, как и с массивами чисел. Например, для описания одномерных массивов можно использовать команды:

- массив символов – **var <имя массива> : array[<кол-во элементов>] of char;**
- массив строк – **var <имя массива> : array[<кол-во элементов>] of string;**

При этом для строковых массивов можно после обозначения типа **string**, как и в случае простой строки, указать значение размера этих строк (все строки массива, разумеется, должны иметь одинаковый максимально допустимый размер).

Обращение же к элементам таких массивов производится полностью аналогично обращению к элементам числовых массивов – путем указания имени массива и записанного после него в квадратных скобках индекса элемента.

А теперь – внимание! – начинается самое интересное.

В отличие, например, от Бейсика, где строки рассматриваются как «единое целое», в языке Паскаль (а также в ряде других языков, например, в Си) существует *дуализм представления строк*: к любой строке, определенной как тип **string**, можно в программе обращаться и как к единому целому (к переменной типа **string**), и как к одноименному одномерному массиву символов, указывая после имени строковой переменной в квадратных скобках номер (индекс) желаемого символа в строке! (При этом следует помнить, что символы в строках *всегда* нумеруются с единицы.)

Например, если мы определили строковую переменную оператором

var stroka : string[10];

и записали в нее строку '**Привет!**', то оператор

```
writeln(stroka[1]);
```

выведет на экран символ '**П**', а оператор

```
writeln(stroka[7]);
```

выведет символ восклицательного знака.

Соответственно, одномерный массив строк можно аналогичным образом рассматривать как двумерный массив символов, двумерный массив строк – как трехмерный массив символов и т.д.

Эта замечательная возможность делает ненужным использование функций извлечения символа из строки (типа **MID\$(<имя строки>,<позиция символа>,1)** в Бейсике), поскольку для получения нужного символа достаточно просто обратиться к нему напрямую. Насколько это упрощает решение некоторых задач на работу со строками, мы увидим, когда перейдем к решению таких задач.

И наконец, завершая теоретический экскурс в символьные и строковые типы данных, перечислим некоторые имеющиеся в Паскале стандартные функции и процедуры для работы с символами и строками (информацию о них обычно можно найти в Help'е к используемой системе программирования, мы же будем рассматривать популярную в школьной практике среду программирования Pascal ABC.Net).

Chr(x)	Функция	Возвращает символ (типа char) по заданному 8-битному коду (типа byte) в кодировке Windows
ChrUnicode(x)	Функция	Возвращает символ (типа char) по заданному 16-битному коду (типа word) в кодировке Unicode
Ord(c)	Функция	Возвращает 8-битный код символа (типа byte) в кодировке Windows по заданному символу (типа char)
OrdUnicode(c)	Функция	Возвращает 16-битный код символа (типа word) в кодировке Unicode по заданному символу (типа char)
UpperCase(c) UpCase(c) UpperCase(s)	Функция	Возвращает символ (типа char) либо строку (типа string), преобразованные в прописной (заглавный, верхний) регистр (два варианта записи имени функции реализованы для обеспечения совместимости с другими различными версиями Паскаля)
LowerCase(c) LowCase(c) LowerCase(s)	Функция	Возвращает символ (типа char) либо строку (типа string), преобразованные в строчный (нижний) регистр

Pos(subs,s)	Функция	Возвращает номер позиции (число типа integer) в строке <i>s</i> (типа string), с которой в этой строке содержится подстрока <i>subs</i> (типа string). Фактически это – поиск вхождения подстроки в строку. Если такое вхождение не найдено, то возвращается нулевое значение (которое можно использовать как «флаг»)
PosEx(subs,s,from)	Функция	Также возвращает номер позиции (число типа integer) в строке <i>s</i> (типа string), с которой в этой строке содержится подстрока <i>subs</i> (типа string), но поиск вхождения подстроки в строку ведется не с ее начала, а начиная с заданной позиции <i>from</i> (число типа integer). То есть возможные вхождения подстроки до указанной позиции будут проигнорированы. Если такое вхождение не найдено, то также возвращается нулевое значение (которое можно использовать как «флаг»)
Length(s)	Функция	Возвращает целое число (типа integer), указывающее реальную длину текстовой строки (типа string), записанной, например, в некоторую строковую переменную <i>s</i>
SetLength(s, n);	Процедура	Устанавливает для заданной текстовой строки (типа string) новое значение длины (целое число типа integer)

Insert(source, s, index);	Процедура	Изменяет заданную строку <i>s</i> (типа string), вставляя в нее заданную подстроку <i>source</i> (также типа string) начиная с позиции <i>index</i> (целое число типа integer). Важно помнить, что реальная длина строки <i>s</i> при этом увеличивается
Delete(s,index,count);	Процедура	Изменяет заданную строку <i>s</i> (типа string), удаляя из нее часть символов (количество задается параметром <i>count</i> – целое число типа integer) начиная с позиции <i>index</i> (также целое число типа integer). Следует помнить, что реальная длина строки <i>s</i> при этом уменьшается
Copy(s,index,count)	Функция	Возвращает подстроку (типа string) длиной <i>count</i> (целое число типа integer), начинающуюся с позиции <i>index</i> (также целое число типа integer)
Concat(s1,s2,...)	Функция	Возвращает строку (типа string), «склеенную» (операция конкатенации) из двух или более заданных строк <i>s1, s2, ...</i> (типа string). То же самое можно сделать и при помощи операции конкатенации строк, записываемой знаком +

StringOfChar(c,count)	Функция	Возвращает строку (типа string), составленную из заданного количества (<i>count</i> – целое число типа integer) заданных символов (типа char). Например, функция StringOfChar('*',5) вернет строку "*****"
ReverseString(s)	Функция	Возвращает строку (типа string), в которой порядок следования символов изменен на обратный. Например, функция ReverseString("привет") вернет строку "тевирп"
CompareStr(s1,s2)	Функция	Сравнивает заданные строки (типа string) и возвращает числовое значение (типа integer): отрицательное, если $s1 < s2$, положительное, если $s1 > s2$, или равное нулю, если $s1 = s2$. Напомним, что строки можно сравнивать и «напрямую», записывая в условном операторе имена строковых переменных и требуемые знаки логических операций
LeftStr(s,count)	Функция	Возвращает подстроку (типа string) длиной <i>count</i> (целое число типа integer), начинающуюся с начала исходной строки (слева, с позиции 1)
RightStr(s,count)	Функция	Возвращает подстроку (типа string) длиной <i>count</i> (целое число типа integer), из конца исходной строки (справа)

Trim(s)	Функция	Возвращает строку (типа string) с удаленными из нее (из ее исходного вида) пробелами в начале и в конце (пробелы внутри строки не удаляются)
TrimLeft(s)	Функция	Возвращает строку (типа string) с удаленными из нее (из ее исходного вида) пробелами в начале (пробелы в конце и внутри строки не удаляются)
TrimRight(s)	Функция	Возвращает строку (типа string) с удаленными из нее (из ее исходного вида) пробелами в конце (пробелы в начале и внутри строки не удаляются)

StrToInt(s) StrToInt64(s) StrToFloat(s)	Функция	Данные функции преобразуют строку (типа string), содержащую символьную запись числа (цифры, точку, знак минуса) в собственно число типа integer , int64 или real и возвращают это число. Распознавание числа выполняется настолько, насколько это возможно
TryStrToInt(s,value) TryStrToInt64(s,value) TryStrToFloat(s,value)	Функция	Данные функции также служат для преобразования строки (типа string), содержащей символьную запись числа (цифры, точку, знак минуса) в собственно число типа integer , int64 , single или real , но работают несколько иначе. Если такое преобразование выполнено успешно, то число записывается в качестве значения параметра <i>value</i> (он должен иметь соответствующий тип), а функция возвращает логическое (типа boolean) значение True . Иначе (если в строке <i>s</i> содержится не запись числа) функция просто возвращает логическое значение False
Val(s,value,err);	Процедура	Служит для той же цели, что и описанные выше функции, – пытается преобразовать строку (типа string), содержащую символьную запись числа, в само число. Результат такого преобразования – число типа, который имеет параметр <i>value</i> (byte , word , longword , integer , shortint , smallint , int64 , uint64 , single или real) – при успехе записывается в качестве значения параметра <i>value</i> , а значение параметра <i>err</i> (типа integer) приравнивается нулю. Если же такое преобразование выполнить не удалось, то значение параметра <i>err</i> будет больше нуля
Str(x,s);	Процедура	Преобразует заданное число <i>x</i> (типа integer или real) в строку (типа string), содержащую символьную запись этого числа (цифры, точку, знак минуса)
IntToStr(x) FloatToStr(x)	Функция	Выполняют аналогичную операцию, возвращая для заданного числа <i>x</i> (типа integer , int64 или real) в строку (типа string), содержащую символьную запись этого числа

При использовании этих функций и процедур необходимо помнить следующее:

– функции *возвращают* некоторое значение, которое нужно куда-то записать или как-то использовать, поэтому функцию надо записывать или в операторе присваивания (например: **d := Length(s);**), или, скажем, в операторе вывода на экран (**writeln(Length(s));**);

– процедуры, в отличие от функций, сами по себе не возвращают значений, а изменяют значение некоторых заданных в них параметров (аргументов), поэтому в составе оператора присваивания или вывода на экран их записывать нельзя. Процедура записывается отдельным оператором, например: **Delete(stroka,5,1);** – правильная запись (из строки *stroka* удаляется один символ, стоящий в 5-й позиции), а **stroka1 := Delete(stroka,5,1);** – неправильная запись.

А теперь перейдем, наконец, к решению задач.

Задача 1

Дана строка символов. Удалить из нее первый знак препинания.

Наиболее простое решение: определить длину введенной строки, реализовать цикл перебора всех ее символов (с 1-го до последнего, имеющего номер, равный значению длины), каждый очередной символ сравнивать с каждым из возможных символов – знаков препинания («.», «,», «;», «!» и т.д.) и при выполнении этого условия каким-то способом убрать его из строки, а затем – прервать цикл просмотра символов.

Реализуем эту идею на Паскале:

```
program z1;

var st, st1 : string;
    dl, i, k : integer;
begin

    writeln('Введите строку');
    readln (st);

    dl := Length(st);
    k := 0;
    for i := 1 to dl do
        if (st[i] = '.') or (st[i] = ',') or (st[i] = ';') or (st[i] = '!') or (st[i] = '?') then begin
            k := i;
            break;
        end;
    if k <> 0 then st1 := LeftStr(st, k-1) + RightStr(st, dl-k)
        else st1 := 'Знаков препинания нет';

    writeln(st1);

end.
```

Проанализируем этот листинг.

1. Вводится строка и определяется ее длина *dl* (при помощи стандартной функции **Lenght**).

2. Переменная *k*, которая у нас одновременно будет служить для запоминания позиции найденного первого знака препинания и играть роль «флага», обнуляется.

3. Строится цикл **for** перебора значения переменной *i* от 1 до значения длины строки *dl*.

4. В теле цикла мы должны извлечь очередной (записанный в позиции *i*) символ строки. И вот здесь проявляется удобство «дуализма» обращения к строкам в языке Паскаль: вместо того, чтобы, как в Бейсике, записывать каждый раз функцию, извлекающую нужный символ как подстроку (в Бейсике: **MID\$(ST\$,I,1)**, в Паскале – **Copy(st,i,1)**), мы можем просто обратиться сразу к требуемому символу как к элементу массива *st* с индексом *i*: **st[i]**.

5. Очередной символ (**st[i]**) нужно сравнивать с каждым из возможных символов – знаков препинания, записывая операции сравнения типа **st[i] = '.'** через логическую связку **or** в операторе **if**. Тогда в ветви **then** (т.е. если очередной символ строки равен хотя бы одному

знаку препинания) мы запоминаем его номер позиции (i) в переменной k и прерываем цикл оператором **break**.

6. После завершения цикла – досрочного по **break** или «штатного», когда завершен перебор всех символов строки, а знак препинания в ней не найден, нам надо разделить эти два случая. Для этого мы используем «флаговую» функцию переменной k :

– если k не равно нулю, значит, знак препинания найден и его номер позиции в строке записан в k , – тогда мы выполняем операцию «удаления» этого k -го символа из строки (п. 7);

– иначе, если $k = 0$, это означает, что знак препинания найден не был, цикл завершился сам по себе, а значение k сохранилось исходное, которое мы присвоили этой переменной еще до цикла, – тогда мы просто должны вывести сообщение, что знаков препинания в строке нет.

7. Чтобы «удалить» найденный знак препинания, сделаем следующее. Оставляя исходную строку неизменной, будем формировать из нее новую строку. Сначала запишем в нее все символы исходной строки с первого до k -го (не включая его), а затем допишем (конкатенируем) к ней символы из правой части исходной строки от k -го (опять же не включая его) до последнего. Первую часть строки (слева от знака препинания) можно получить, используя функцию **LeftStr(st, k-1)**, а вторую (правую) – используя функцию **RightStr(st, dl-k)**. При этом вычисление количеств извлекаемых символов достаточно очевидно, если представить строку наглядно, как на рис. 2.

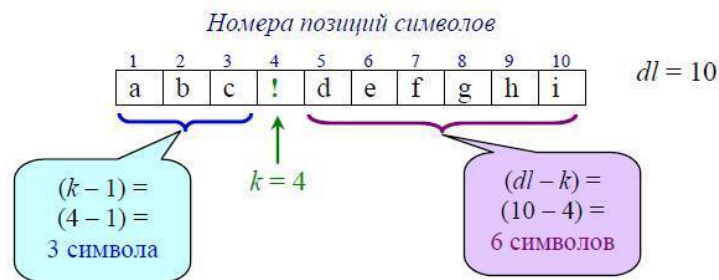


Рис. 2

Решение достаточно простое. Но запись условного оператора получается довольно громоздкой: ведь в нем надо перебрать (через **or**) все возможные случаи равенства очередного символа какому-нибудь знаку препинания. Кроме того, подумайте, что было бы, если бы таких сравнений требовалось несколько в разных местах программы и вдруг выяснилось бы, что надо изменить (скажем, дополнить знаком двоеточия) перечень обрабатываемых знаков препинания? Пришлось бы внимательно (но все равно с риском где-то что-то пропустить) просматривать всю программу, выискивать в ней все такие операторы сравнения и дописывать в них еще одно логическое условие...

Можно ли упростить программу, а заодно – и возможные модификации перечня знаков препинания, обрабатываемых в ней? Можно! И в этом нам поможет интересный, но, к сожалению, редко используемый множественный тип данных – *множество* (простите за получившийся каламбур).

Множество – это набор однотипных элементов. Однако в отличие от массива, в котором такие элементы располагаются последовательно и пронумерованы индексами, множество – это просто группа элементов, «сваленных в одну кучу». Индексов у элементов множества нет. Более того – для множества порядок записи в нем элементов не важен, например [1, 2, 3] и [3,

2, 1] – это одно и то же множество цифр. Другая особенность множества – это уникальность его элементов: каждый из них должен присутствовать в множестве только «в одном экземпляре», без повторов, – например, набор [1, 2, 3, 2, 1] множеством не является.

В языке Паскаль множества определяются следующим образом:

```
type <имя типового множества> = set of <тип или набор элементов>;
var <имя экземпляра множества> : <имя типового множества>;
```

То есть, сначала объявляется некий «класс» множеств, скажем, множество символов или чисел, а затем создается сколько угодно конкретных множеств этого типа. Например:

```
type mn1 = set of byte; – множество всех возможных беззнаковых целых 8-
разрядных чисел;
type mn2 = set of char; – множество всех возможных символов;
type mn3 = set of '1'..'9'; – множество всех цифр, кроме нуля;
type mn4 = set of 'a'..'z'; – множество всех латинских строчных букв.
```

А после определения типового множества при объявлении экземпляра можно задать требуемые значения элементов, например:

```
var mn : mn2 := ['a', 'e', 'ё', 'и', 'о', 'у', 'ы', 'э', 'ю', 'я']; – создаем множество всех
русских строчных гласных букв.
```

Множество может оставаться и пустым (не содержать никаких элементов), тогда оно обозначается как [].

Над множествами можно выполнять следующие операции:

- объединение (операция **+**) – результатом является множество, включающее (по одному разу!) элементы, которые есть хотя бы в одном из исходных множеств;
- пересечение (операция *****) – результатом является множество, включающее только элементы, которые есть в каждом из исходных множеств;
- разность (операция **–**) – результатом для двух исходных множеств является множество, включающее только элементы, которые есть в первом из этих множеств, но отсутствуют во втором;
- проверка вхождения элемента в множество (операция **in**)

Конец ознакомительного фрагмента.

Текст предоставлен ООО «ЛитРес».

Прочитайте эту книгу целиком, [купив полную легальную версию](#) на ЛитРес.

Безопасно оплатить книгу можно банковской картой Visa, MasterCard, Maestro, со счета мобильного телефона, с платежного терминала, в салоне МТС или Связной, через PayPal, WebMoney, Яндекс.Деньги, QIWI Кошелек, бонусными картами или другим удобным Вам способом.