БИЛЬФЕЛЬД Н. В.
ЭФФЕКТИВНОЕ ИСПОЛЬЗОВАНИЕ СТРОК PCHAR В DELPHI

УДК 004.4'2, ВАК 05.13.11, ГРНТИ 50.05.13

Эффективное использование строк PChar в Delphi

Н. В. Бильфельд

Пермский национальный исследовательский политехнический университет, Березниковский филиал

Дан краткий анализ используемых в Delphi строк. Рассмотрена особенность работы с PChar строками в Delphi. Разработана база данных для работы со строковыми функциями. Дана классификация функций по их местонахождению и используемым строкам. Приведены примеры неверной инициализации PChar и возникающие при этом ошибки и проблемы. Приведены таблицы распределения памяти при использовании PChar строк. Даны рекомендации по корректной инициализации PChar строк и приведены примеры их использования.

Ключевые слова: Delphi, строки, программирование.

Effective use of PChar strings in Delphi

N.V. Bilfeld

Perm National Research Polytechnic University, Berezniki branch

A brief analysis of the strings used in Delphi is given. The peculiarity of working with PChar strings in Delphi is considered. A database is developed for string functions processing. Classifications of functions by their location and used strings are suggested. Examples of incorrect initialization of PChar and the errors and problems that arise in this are given. Tables of memory allocation when using PChar strings are explained. Recommendations for correct initialization of PChar strings and examples of their use are given.

Keywords: programming, strings, Delphi

В *Delphi* можно использовать четыре типа строк. Строки *AnsiString, ShortString, WideString* и *PChar*. В данной статье не дается строгой теории о том, чем эти строки отличаются, в частности, что строки *AnsiString* и *ShortStrin* объявляются по умолчанию как *String*, а также о настройках компилятора при работе со строками. Все это достаточно хорошо изложено в [1, 2]. Здесь больше рассматриваются сами функции для работы со строками, а их достаточно много. Весь перечень составляет более 150 функций [3]. Для анализа этих функций была разработана база данных с описанием этих функций и примерами их использования. При этом все примеры можно запустить на выполнение и изменять аргументы функций, анализируя, таким образом, их работу.

В результате получилась следующая классификация:

Все функции находятся в разных модулях. В модуле *StrUtils* находится 43 функции для работы со строками *AnsiString*. В модуле *SysUtils* находится 85 функций и из них 37 для строк *PChar*, которым и посвящена данная статья. В модуле

System находится 16 функций, которые в основном перекочевали в *Delphi* из *Pascal*, в частности функция *Val*, у которой нет точного аналога в C++. В модуле *Windows* находится 7 функций для работы с *PChar* строками. Итак, уже 45 функций для *PChar* строк. И одна функция затерялась в модуле *Classes*, а именно функция *LineStart*, которая возвращает указатель на строку, стоящую справа от *$0A* (код перевода строки).

Разрабатывая примеры использования функций, я натолкнулся на некоторые особенности при работе с *PChar* строками (необходимо отметить, что до этого я данными строками не пользовался). Не выдавая никаких ошибок при компиляции, одни примеры работали, а другие, абсолютно похожие на предыдущие вдруг генерировали ошибки при выполнении. В других случаях все работало, но генерировалась ошибка при завершении программы. Разобравшись со всеми проблемами, я решил поделиться с этим с начинающими программистами, чтобы, как говориться, не наступать на одни и те же грабли.

По определению, PChar-строки это нуль - терминированные (иногда говорят нуль - терминальные) строки. Такая строка заканчивается символом с нулевым кодом в отличие от обычных строк, где длина строки хранится в первых байтах стоки. Длина таких строк ограничена только памятью.

С другой стороны – это указатели. Переменная типа *PCha*r является указателем на начало строки. Но это особый указатель, позволяющий выполнять с ним операции, недоступные для других указателей. В частности, с одной стороны можно обращаться к символам строки по индексу, с другой стороны к указателям типа *PChar* разрешено добавлять и вычитать целые числа, смещая указатель на соответствующее количество байт вправо или влево.

Такие строки используются в системе *Windows* в качестве параметров низкоуровневых функций. В программах такие строки можно рассматривать как статические массивы типа *Char*, начинающиеся с нулевого индекса. Такие массивы совместимы с типом *PChar*, что позволяет обойтись без использования динамической памяти при работе со строками.

Если при инициализации объявить строку как *Var P:PChar*, то в дальнейшем, казалось бы, можно с ними работать, и выполнять определенные операции и функции. Но делать так можно не всегда.

Например, такой строке можно присвоить значение:

```
Var S:String;P:Pchar;
{Фрагмент 1}
begin
  P:='Велика'; Edit1.Text:=P;
end;
```

Такую строку можно присвоить обычной строке:

```
Var S:String;P:Pchar;
{Фрагмент 2}
begin
```

```
  P:='Велика'; S:=P;
  Edit1.Text:=S;
  S:=StrPas(P); Edit2.Text:=S;
end;
```

Как видно из примера, для такой строки можно использовать специальную функцию: *S:=StrPas(P),* которая преобразует строку *PChar* в строку *String*.

Такой строке можно присвоить значение из *Edit*, используя функцию *Pchar*:

```
Var P:Pchar;
{Фрагмент 3}
begin
  //P:=Edit1.Text;Так присваивать нельзя
  P:=Pchar(Edit1.Text); Edit2.Text:=P;
End;
```

При присваивании строк *PChar* обычным строкам, происходит автоматическое преобразование типов. Тогда в дальнейшем их можно сложить, и это будет работать:

```
Var P1,P2:PChar;S1,S2,S3:String;
{Фрагмент 4}

Begin
  P1:='AB';P2:='CD';
  S1:=P1;S2:=P2;
  S3:=S1+S2;
  Edit1.Text:=S3;
End;
```

Такой строке можно присвоить значение обычной строки (*String*), используя функцию *PChar:*

```
Var P1:PChar;S1:String;
{Фрагмент 5}
Begin
  S1:='ABC';
  P1:=PChar(S1);
  Edit1.Text:=P1;
End;
```

При присваивании таким строкам значений будут работать отдельные функции, например функция сравнения строк:

```
Var A:Integer;P1,P2:PChar;
{Фрагмент 6}
begin
  P1:='HELLO';P2:='hello';
  A:=AnsiStrComp(P1,P2);
  Edit1.Text:=IntToStr(A);
end;
```

Все перечисленные выше примеры работают, но, тем не менее, так делать нельзя, иначе могут возникнуть парадоксальные ситуации. Рассмотрим некоторые из них:

Этот пример работает:

```
Var P1,P2,P3:PChar;
{Фрагмент 7}
Begin
   P1:=StrNew('ABC'); P2:=StrNew('KLM');
   StrCopy(P3,P1);
   Edit1.Text:=P3;
End;
```

Мы корректно объявили две строки, используя функцию *StrNew*. Затем скопировали строку *P*1 в строку *P*3, используя функцию *StrCopy*, и вывели строку *P*3 в компоненте *Edit*.

А этот пример уже работать не будет!

```
Var P1,P2,P3:PChar;
{Фрагмент 8}
Begin
  P1:=StrNew('ABC'); P2:=StrNew('KLM');
  StrCopy(P3,P1);
Edit1.Text:=P2;
End;
```

А что собственно изменилось. Мы выводим в компоненте *Edit* значение другой переменной, которая корректно объявлена!

А вот вариант еще парадоксальнее. Этот пример работает:

```
Var P1,P2,P3:PChar;
{Фрагмент 9}
Begin
  P1:='ABC';
  StrCopy(P2,P1);
  P3:=AnsiStrLower(P2);
  Edit1.Text:=P3;
End;
```

Функция *StrCopy* копирует строку *P*1 в строку *P*2. Функция *AnsiStrLower* переводит значение аргумента к нижнему регистру. Все работает! А теперь удалим из этого примера две последние строки, и пример перестает работать!!!

```
Var P1,P2,P3:PChar;
{Фрагмент 9}
Begin
   P1:='ABC';
   StrCopy(P2,P1);
   {P3:=AnsiStrLower(P2);
   Edit1.Text:=P3;}
End;
```

Как такое может быть. Две строки, которые прекрасно работали в предыдущем примере, теперь работать отказываются! Более того, мы удалили фрагмент не предшествующий этим строкам, а фрагмент, который после них, от которого, казалось бы, вообще ничего зависеть не может.

Для корректной работы всех функций, необходимо инициализировать строки другим образом.

Для правильной инициализации строк можно использовать две функции, это *StrNew* и *StrAlloc*.

Функция *StrNew(P:PChar):PChar* выделяет память и копирует в нее указанную строку. Примеры использования:

```
Var P1:PChar;S1:String;
{Фрагмент 10}
Begin
  P1:=StrNew('ABC');
  S1:='ABC';P1:=StrNew(PChar(S1));
  P1:=StrNew(PChar(Edit1.Text));
End;
```

Размер памяти, выделяемый под строку равен длине строки (количеству символов) плюс 1 байт. Размер памяти можно определить с помощью функции: *StrBufSize(P:PChar):Integer*.

Размер *Pchar* строки можно определить с помощью функции:

StrLen(P:PChar):Integer). Функция определяет количество символов в строке, хотя длину строки можно получить и с помощью обычной функции *Length*.

Освободить память можно с помощью функции *StrDispose(P:PChar)*.

Рассмотрим следующий фрагмент:

```
Var P1;PChar;
{Фрагмент 11}
Begin
  P1:=StrNew('123456');
```

```
  Edit1.Text:=P1;
  StrDispose(P1);
End;
```

В результате выполнения первой строки выделится память 7 байт (длина строки 6 байт). В результате выполнения второй строки значение выведется в *Edit*. В результате выполнения третьей строки освободится память. В данном случае все корректно.

А теперь рассмотрим другой фрагмент:

```
Var P1;PChar;
{Фрагмент 12}
Begin
  P1:=StrNew('123456');
  Edit1.Text:=P1;
  P1:='12';
  Edit2Text:=P1;
  StrDispose(P1);
End;
```

В результате выполнения данного фрагмента сгенерируется ошибка. Давайте разберемся, в чем тут дело. Если не выполнять команду *StrDispose(P1),* то все будет работать. В *Edit*1 выведется значение 123456, в *Edit*2 выведется значение 12. Даже если попытаться определить длины строк, то они определятся правильно (6 и 2). Если бы мы не попытались освободить память, то так бы и не поняли, что здесь что-то не так. А все дело в строке *P1:='12';* Выполнив такую строку, мы «испортили» буфер памяти, его теперь просто нет. Поэтому и при освобождении памяти выводится ошибка.

Вывод напрашивается сам собой. Присваивать строкам значения можно только, используя функцию *StrNew*.

Использовать *StrNew* необходимо в тех случаях, когда строки используются как константы, и их длины меняться не будут.

Рассмотрим еще один пример с функцией *StrCopy(P1,P2:PChar):PChar*. Функция копирует строку *P*2 в строку *P*1 и устанавливает указатель на *P*1.

```
Var P1,P2,P3;PChar;
{Фрагмент 13}
Begin
  P1:=StrNew('123456');
  P2:=StrNew('AB');
  P3:=StrNew('');
  P3:=StrCopy(P1,P2)
End;
```

До выполнения *P3:=StrCopy(P1,P2)* память распределена следующим образом (таблица 1):

Таблица 1. Распределение памяти до выполнения команды

	Память	Длина	Значение
P1	7	6	123456
P2	3	2	AB
P3	1	0	

После выполнения *P3:=StrCopy(P1,P2)* память распределена следующим образом (таблица 2):

Таблица 2. Распределение памяти после выполнения команды:

	Память	Длина	Значение
P1	7	2	AB
P2	3	2	AB
P3	7	2	AB

Как видно из примера, при копировании *P*2 в *P*1 размер памяти для строки *P*1 не изменился. Следовательно, при использовании команды *StrCopy*, размер памяти, отведенный под исходные строки, не меняется. В данном случае все будет работать корректно, так как длины строк соответствуют занимаемой ими памятью.

А теперь будем копировать строку *P*1 в *P*2: *P3:=StrCopy(P2,P1)*

До выполнения команды распределение памяти будет соответствовать таблице 1. После выполнения команды распределение памяти будет соответствовать таблице 3.

Таблица 3. Распределение памяти после выполнения команды:

	Память	Длина	Значение
P1	7	6	12345
P2	3	6	12345
P3	3	6	12345

Как видно из таблицы размер памяти теперь не соответствует размеру строк. При выполнении фрагмента все будет работать даже при попытке вывести значения строк в компонентах *Edit*. Ошибка в данном случае сгенерируется при закрытии программы, так как *Delphi* попытается освободить неверно распределенную память.

В этом случае инициализировать строки необходимо другим образом.

Функция *StrAlloc(B:Integer):PChar* выделяет память под строку и возвращает указатель на местоположение первого символа строки.

Примеры использования:

```
Var P1:PChar;S1:String;
{Фрагмент 14}
Begin

P1:=StrAlloc(20);
  S1:='ABC'; P1:=StrAlloc(Lenhth(S1)+1);
End;
```

Рассмотрим пример:

```
Var P1,P2,P3;PChar;
{Фрагмент 15}
Begin
        P1:=StrAlloc(10); P2:=StrAlloc(10); P3:=StrAlloc(10);
  P1:='AB'; P2:='DCE'; P3:='123456';
End;
```

Сразу необходимо отметить, что, не смотря на то, что переменным присвоятся конкретные значения, которые можно вывести в компонентах *Edit* – делать так нельзя. Мы снова нарушаем распределение памяти, как было показано в фрагменте 12. И попытка выполнить после этого *StrDispose* сгенерирует ошибку.

Как же тогда присвоить переменным значения? Здесь как раз нам поможет функция *StrCopy*.

Рассмотрим пример:

```
Var P1,P2,P3;PChar;
{Фрагмент 16}
Begin
        P1:=StrAlloc(10); P2:=StrAlloc(10); P3:=StrAlloc(10);
  StrCopy(P1,PChar('AB'));
  StrCopy(P2,PChar('DCE'));
  StrCopy(P3,PChar('12345'));
End;
```

Распределение памяти приведено в таблице 4.

Таблица 4. Распределение памяти после выполнения фрагмента:

	Память	Длина	Значение
P1	10	2	AB
P2	10	3	CDE
P3	10	5	12345

Из таблицы видно, что отведенная память под каждую строку занимает 10 байт независимо от длины строки. При такой инициализации переменных снимается проблема, приведенная во фрагменте 13 при обратном копировании.

Вместо функции *StrCopy* в данном случае удобнее использовать функцию *StrPCopy*. Она отличается тем, что копируемый параметр имеет тип *String*.

Тогда фрагмент 16 можно записать как:

```
Var P1,P2,P3;PChar;
{Фрагмент 17}
Begin
        P1:=StrAlloc(10); P2:=StrAlloc(10); P3:=StrAlloc(10);
 StrPCopy(P1,'AB'); StrPCopy(P2,'DCE'); StrPCopy(P3,'123456');
End;
```

Такая запись естественно короче.

Еще, на что необходимо обратить внимание при инициализации памяти для переменной с помощью функции *StrAlloc* – это то, что эту память потом нельзя изменить с помощью этой же функции *StrAlloc*. Значения символов строки при этом теряются. Также для строк *PChar* непригодна функция *SetLength*, которая позволяет управлять длинами только строк типа *String*.

Обойти эту проблему можно, сохранив значение строки в дополнительной переменной, как показано в следующем фрагменте:

```
Var P1:Char;S:String;
{Фрагмент 18}
Begin
 P1:=StrAlloc(3);
 StrPCopy(P1,'AB');
 S:=P1;
 P1:=StrAlloc(30);
 StrPCopy(P1,S);
End;
```

В результате значение строки *P*1 сохранится, но выделенная под нее память будет составлять уже не 3, а 30 байт.

Итак, существует два способа инициализации *Phar* строк. Это использование функции *StrNew* и *StrAlloc*. При правильной инициализации строк этими функциями, весь большой список функций для работы с *PChar* строками работает корректно. Но, как и везде, существуют досадные исключения. В частности, это функция *StrCat*.

Формат функции *StrCat(P1,P2:PChar):PChar*.

Назначение функции – объединение строк.

Рассмотрим пример:

```
Var P1,P2,P3;PChar;
{Фрагмент 19}
Begin
        P1:=StrAlloc(20); P2:=StrAlloc(20); P3:=StrAlloc(20);
 StrPCopy(P1,'AB'); StrPCopy(P2,'DCE');
 P3:=StrCat(P1,P2);
 StrDispose(P1); StrDispose(P2);
End;
```

Распределение памяти, после выполнения фрагмента, приведено в таблице 5.

Таблица 5. Распределение памяти после выполнения фрагмента:

	Память	Длина	Значение
P1	20	2	AB
P2	20	3	CDE
P3	20	5	ABCDE

Вроде бы все, хорошо, но ошибка генерируется после попытки освободить память, для строки *StrDispose(P3);*

Если этого не делать, то все работает корректно и программа закрывается тоже без ошибок. С другой стороны, эту функцию можно использовать как процедуру: *StrCat(P1,P2);* Результат поместиться в строку *P*1 и освобождение памяти для строк обоих произойдет корректно.

Еще один пример, который показывает, что все функции для работы с *PChar* строками можно использовать для работы с массивами символов.

```
Var P1,P2,P3:PChar; S1,S2:Array[0..20] Of Char;
{Фрагмент 20}
Begin
 P1:=StrNew('красно-');
 P2:=StrNew('зеленый');
 P3:=StrNew('синий');
 StrCopy(S1, P1);
 StrCopy(S2, P1);
 StrDispose(P1);
 StrCat(S1, P2);Edit13.Text:=S1; {красно-зеленый}
 StrCat(S2, P3);Edit14.Text:=S2; {красно-синий}
 StrDispose(P2);StrDispose(P3);
End;
```

Итак, мы рассмотрели основные подводные камни при работе с *PChar* строками и как их избежать. Имея в арсенале такую огромную библиотеку функций можно минимальными затратами писать достаточно мощные фрагменты, в част-

ности касающиеся поиска и замены текста или его статистической обработки. Дополнительные сведения по вопросу можно получить в интернет-источниках, в том числе, из следующего списка:

- http://atree.click/delphispr/AnsiIndexStr.php
- http://cppstudio.com/post/437/
- http://cubook.supernew.org/manual-c/types/197-ansistring
- http://helpiks.org/7-41553.html
- http://platonov-andrei.narod.ru/Delphi/FuncAPI/index.htm
- http://platonov-andrei.narod.ru/Delphi/NullTermStrFunct.htm
- http://platonov-andrei.narod.ru/Delphi/StrRoutines.htm
- http://www.delphibasics.ru/navStrUtils.php
- http://www.delphibasics.ru/navSystem.php
- http://www.delphibasics.ru/navSysUtils.php
- http://www.delphi-manual.ru/work-with-strings.php
- http://www.interface.ru/home.asp?artId=3835
- http://www.programmersforum.ru/showthread.php?t=180282
- https://learnc.info/c/mastering_strings.html#atoi
- https://studfiles.net/preview/5772717/

На основе сделанного анализа разработана база данных, в которую включены все имеющиеся функции для работы со строками.

Список использованных источников и литературы

1. С. Бобровский. Delphi 7 учебный курс. – СПб.: Питер 2006. – 735 с.

2. Тонкости работы со строками [электронный ресурс]. – *URL: http://www.delphikingdom.com/asp/viewitem.asp?catalogid=1206#04* (Дата обращения: 15.01.2021)

3. Варламова С. А., Затонский А. В. Информационно-управляющая система филиала вуза как неотъемлемый элемент системы качества образования // Фундаментальные исследования. – 2007. № 12-3. – С. 447-451.

List of references

1. S. Bobrovsky. Delphi 7 training course. – SPb .: Peter 2006 . – 735 p.

2. The subtleties of working with strings [electronic resource]. – URL: http://www.delphikingdom.com/asp/viewitem.asp?catalogid=1206#04 (Date of access: 15.01.2021)

3. Varlamova S. A., Zatonskiy A. V. Information and management system of a university branch as an integral element of the education quality system // Fundamental research. – 2007. No. 12-3. – P. 447-451.