Delphi 陷阱大全

作者：孙晓刚

2013-10-01

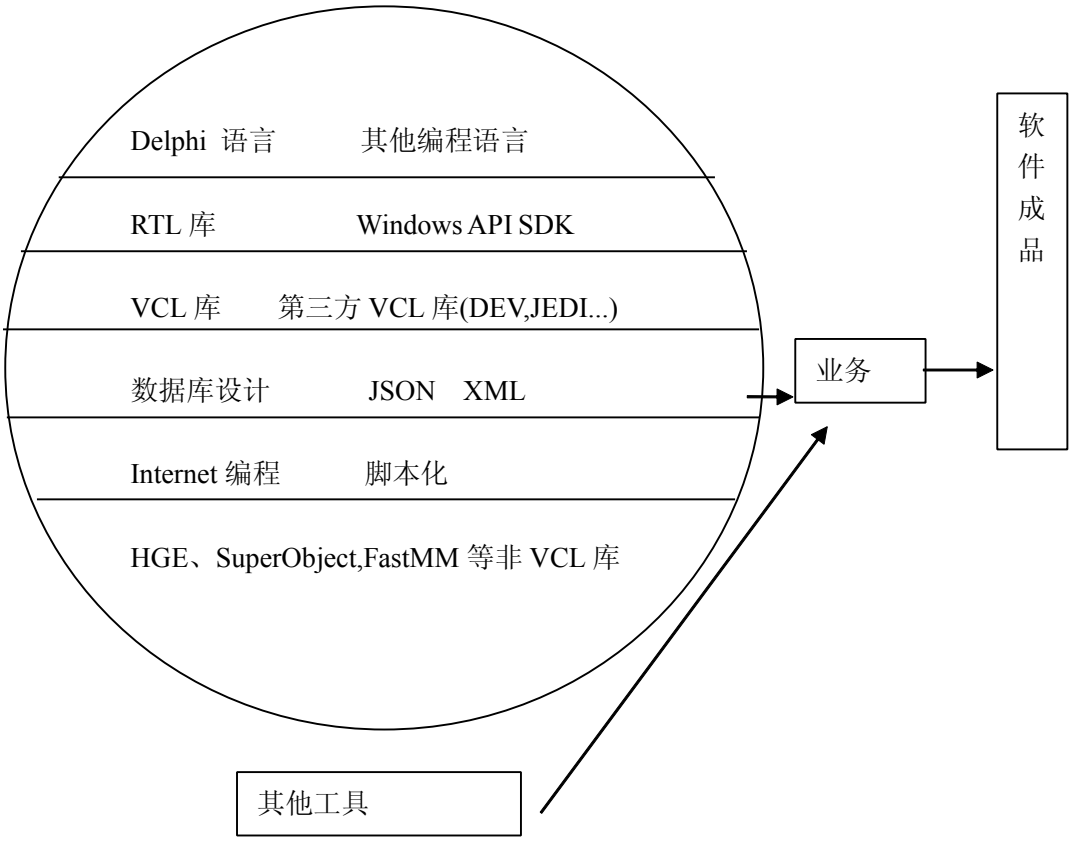
作者序

我们发现，在软件开发过程中，总是会出现各种各样的陷阱，导致软件产品总是有许多的 BUG。那么如何减少这些 BUG，提高软件产品质量呢？

有人说，要有好的编程风格；有人说，要遵循工程化的方法；有人说，要尽量多做测试；特别是单元测试。

这些都是对的。不过，根据老程序员的经验来说，软件设计过程中产生的错误，很多都是因为不了解语言、开发工具、程序库导致的。那么有没有一本书来详细讲解 Delphi 的一些限制呢?本书就是基于这一情况而推出的。

学习 Delphi 需要掌握的知识如下图。这么大量的学习，一定有许多地方容易遗漏导致错误。



目前很少有 Delphi 相关的新书上架。以前出的书大都是讲解入门知识，要么罗列组件的使用方法。深入探讨语言底层的很少见，希望这本书能够提供一些补充吧。

软件开发最重要的就是代码复用，我们来看看 Delphi 为我们提供了多少官方代码：

版本	行数
Delphi4	>25 万行
Delphi7	>80 万行
Delphi2007	>100 万行
DelphiXE2	>200 万行
Lazarus 1.0	>180 万行(包括收集的一些第三方包)

这样大量的代码，其 BUG 量却很低，可见 Delphi 是一款高质量的软件。加上各种高质量第三方代码，可复用代码库可达到几百万甚至千万行级代码。为软件开发商节省了大量重复开发时间成本。

本书内容以 Delphi7 为蓝本，以最新的 Delphi 版本作为补充，解释一些 BUG、陷阱，并讲解一些非陷阱的知识点。由于版本差异，书里面的例子不一定全部正确，请自行调试。Delphi 的陷阱主要来自：

1.Delphi 自己版本的变化，在老的版本的意义和新的版本可能不同；

2.一些入门书籍没有深入探讨的讲解；

3.程序员没有认真阅读联机帮助手册；

4.安装了版本不一致的第三方软件；

5.网络上的资料很多没有注明算法或者代码适用的版本；

声明：本书起《Delphi 陷阱大全》这个名字，并非哗众取宠，也不是表示 Delphi 这个软件有多少 BUG，而是我们常常看到的书籍，没有深入探讨一些问题，导致我们许多学员学习到一些错误的知识。作者水平有限，写出这些文字，希望大家能够共同进步，也欢迎 Delphi 的爱好者、程序员提出不同意见，促进 Delphi 开发者共同提高。我的 Mail：1565498246@qq.com。

或加入 QQ 群:34176611

目录表

29.LowerCase 和 AnsiLowerCase

30.Uppercase 和 AnsiUpperCase

31.同样，AnsiUpperCaseFileName 和 uppercase 进行对比

32.Inttostr 陷阱

33.向后兼容函数或者过程列表

34.Ceil 函数

35.Floor 函数陷阱.

36.BCD 数据表示范围

37.DIVMOD 过程

38.Ceil 和 INT 函数区别

38.Frac 和 INT 函数

39.TStringList 陷阱

40.CompareValue 函数比较浮点数

41.TimeStamp 和 Mssql 的 TimeStamp 的不同点

42.HoursBetween 和 HourSpan

43.字符串转换成 Boolean 的自定义，如果不小心使用可能导致错误

44.Delphi 中的序列化与反序列化

45.Delphi 中的哈希表

46.另外一个第三方的快速哈希算法

47.XE 中动态数组的错误忽略功能

48.Delphi,Lazarus 数组属性

49.Delphi 同单元的类保护域的访问

50.For 循环变量是否不可变

51.Case 语句是否可以用字符串作为判定变量

52.Delphi 程序的 5 种单元结构

53.过程、函数、变量、属性之间的区分

54.数组型属性 和 数组 的不同点。

55.默认数组属性 和 数组

56.无法转换的 Tlist.last 指针

57.没有类型的参数

58.writeln 不能支持的变量类型

59.没有初始化的动态数组,例程:

60.for 循环变量是否可以使用复杂变量?

61. Delphi 的 Class,Object,Class(TObject)区别

62.Delphi2007 新的类型

63.Delphi 不支持的指针操

64.Unicode 引起的问题

1.辨析 Pos和AnsiPos函数

许多资料，包括万一的博客等都说delphi的pos函数可以用在汉字的查找。

这是明显不对的，官方资料help里面明显说了，pos函数没有按本地字符集

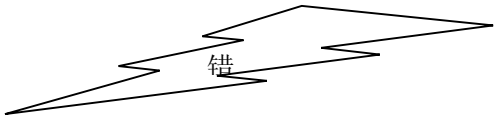
进行查找，很可能导致查找错误。如果不是ascii字符集的字符串，必须用

ansipos函数。

<<Delphi函数大全>>也错了。至少是不完整的描述。
所谓本地字符集就是MBCS(主要针对亚洲的多字节字符集编码)

```
procedure TForm1.Button1Click(Sender: TObject);
var s:AnsiString;
  w:WideString;
begin
  s := 'bAc';
   memo1.lines.add(inttostr(Pos ('A',s)));

   s :='丄';
      memo1.lines.add(inttostr(Pos ('A',s)));

     w :='bac';
      memo1.lines.add(inttostr(Pos ('A',s)));

   w :='丄';
      memo1.lines.add(inttostr(Pos ('A',w)));
```

```
    w :='丄';
        memo1.lines.add(inttostr(ansiPos ('A',w)));

    s :='丄';
        memo1.lines.add(inttostr(ansiPos ('A',w)));
end;
```

结果是 

2

2

2

0

0

0

也就是说当原始数据是 ansistring并且包含的数据可能有MBCS的字符的时候，应该使用ansipos函数。

同理，其他类似函数有MBCS版本，都应该使用Ansi版本。 而如果是WideString,或者确定原始数据里面不包含MBCS字符，就可以使用非Ansi版本。

但以上程序在Delphi2010以后版本的结果是：

2

0

0

0

0

0

因为delphi2010以后，默认的在IDE输入的字符串会尽量向WideChar转换。

2.ansiquotedstr 和quotedstr区别

ansiquotedstr 支持MBCS

quotedstr 不支持MBCS

但是由于汉字在windows上只有$#80$#40开始,而单引号才$#27 ,所以没有冲突,可以认为二者一样，但使用其他国家语言就不一定正确。

3.AllocMem和GetMem函数区别

AlloMem将会分配内存，并初始化为#0 释放使用FreeMem

GetMem不会将每个字节初始化为#0 ,释放也用freeMem

但是最好是用 New , Dispose函数。

但是New只能用在TRecord等大小固定的数据结构。

4.对比 ,AnsiCompareFileName,samefilename,比较两个文件名

AnsiCompareFileName 比较两个文件名

samefilename 函数 和ansicomparefilename 相似,但前者相同返回true ,不同返回false,

后者相同返回0,不同返回有2种情况。

5.对比 AnsiLastChar, 得到字符串最后一个字符 (没有非MBCS的版本)

AnsiLastChar 得到字符串最后一个字符(不一定是字节,是MBCS的字符)

单字节字符集没有必要再弄个函数。

6.对比AnsiLowercaseFileName,LowerCase函数，转换到小写。

AnsiLowercaseFileName 得到文件名字的小写形式，也支持汉字（因为汉字的一半可能和大写重叠，被错误转换,比如 佷 字 第2字节是字符B相同的）

```
self.caption := AnsiLowercaseFileName('A 佷 ') + ' ' + AnsiLowercaseFileName('ABB');
```

得到的结果是

a佷 abb

```
self.caption := AnsilowerCase('A佷') + ' ' + LowerCase('A佷');
```

得到的结果是

a佷 a俠

明显是错误的

参考本资料库上级目录的demo里面的测试程序.

注意:LowerCase不支持汉字等MBCS字符串(会出错)。

这个函数也可以用在所有字符串转换方面。

7.对比,StrNextChar,nextchar 得到下个字符指针，支持MBCS

StrNextChar 得到下个字符指针，支持MBCS ,用指针

nextcharindex 也是查找下一个字符是什么 ,用整数作为参数和返回

8.对比,ByteType,StrByteType，查找字符串某个位置字符是什么字节类型

ByteType 检测一个字符串指定位置的字节是多字节字符集的开始字符，还是结束字符，

这个函数会从一个完整的字符串的开始位置查找，直到找到指定位置。

返回值

(mbSingleByte, mbLeadByte, mbTrailByte);

strbytetype 和 bytetype类似，但是不同的是他用 0 表示查第1个字符是什么

字节类型，并且是c风格的

指针里面找。

而bytetype是在字符串中查找

9.对比,ByteToCharIndex, ByteToCharLen ,防止半个汉字,进行探测，用来截取字符串

ByteToCharIndex 得到字符串中第几个字节属于是第几个字符。对于单字节编码(由windows系统参数决定)返回的就是参数，因为不存在多字节情况。

ByteToCharLen 意思是检测如果要截取字符串S前面N个字节的完整字符串(防止半个汉字等) 需要包含

完整的几个字符(不是字节)

比如

```
procedure TForm1.Button3Click(Sender: TObject);
var s:string;
begin
   s := '汉a字abc字符1324';
   self.caption := IntToStr( ByteToCharLen(s,4));

end;
```

探测上面字符串取4个字节，根据不切分出半个字节的情况，需要截取多长，结果是

3

意思是必须读3个完整字符，

其中第1个字符是

汉字“汉"，

第2字符“a”

第3字符“字”

而截取前面3个字符的函数是 leftstr ， 如果非要将字符截断，可以用 leftBstr ,多了个B.

10.对比,chartobyteindex,CharToByteLen,也可以用来探测半个字符情况，用StrLeft,StrBLeft处理前用

chartobyteindex 当然是查找第几个字符是字符串的第几个字节了。如果是双字节字符，返回的是第

一个字节的位置，比如

```
procedure TForm1.Button5Click(Sender: TObject);
var s:string;
begin
   s := '汉a字abc字符1324';
   self.caption := IntToStr( chartobyteindex (s,2));

end;
```

检测第1个字符，得到1

第2字符，得到3

第3字符 ，得到4

CharToByteLen 就是查找几个完整字符，需要截取多少字节，和ByteToCharIndex对应，后者查的结果可以用LeftStr截取按字符长度，前者可以直接探测到取多少字节不出现半个字，用leftBStr

nextCharindex 得到某个字节位置的下一个字符的开始字节

11.对比,charlength,strcharlength,探测某个位置字符是单字节字符，还是双字节字符的一个部分

charlength 得到一个字符串中 指定位置字符长度，和length不一样，他只检测1个字符是单字节
还是是双字节，如果是单字节，永远返回1，双字节字符就返回2。

index <=0 返回1

index =1 --字符长度 如果字符是双字节的前导字节、后字节,都返回2 ，否则返回1

strchalength 和charlength 对应，都是参数类型不同，下标索引不同。

12.对比，strleft ， strbleft ,ansileftstr 区别和联系,取字符串左边若干字符

```
procedure TForm1.Button6Click(Sender: TObject);
var s:string;
begin
```

```
    s := '汉a字abc字符1324';
    memo1.lines.add( leftstr(s,3));
    memo1.lines.add( leftBstr(s,4));
    memo1.lines.add( ansileftstr(s,4));
end;
```

得到

汉a字

汉a?

汉a字a

可以看到，应该leftstr ==ansileftstr ，都是按MBCS操作的

那么两者区别在哪里？ leftstr有widestring的重载，而ansileftstr是没有的。

还有就是leftBstr 不支持MBCS,所以导致最后一个汉字字符被切下了1半。

13.对比，ansiContainsStr，AnsiContainsText是否字符串包含在另外一个字符串里面

ansiContainsStr 后面字符串是否包含在前面字符串里面，分大小写。支持MBCS.

AnsiContainsText 是[不]分大小写的，支持MBCS的.

Demo----------

```
procedure TForm1.Button8Click(Sender: TObject);
```

```
begin
  Memo1.Lines.Add(BoolToStr( AnsiContainsText('A佟','bb'),true));
  Memo1.Lines.Add(BoolToStr( AnsiContainsText('ABB','BB'),true));
  Memo1.Lines.Add(BoolToStr( AnsiContainsText('A佟','BB'),true));
  Memo1.Lines.Add(BoolToStr( AnsiContainsText('ABB','bb'),true));

end;
```

==========结果

False

True

False

True

结论是都支持MBCS，只是大小写是否区分。

14.对比,AnsiEndsStr, 一个字符串是否和另外一个字符串的尾巴相同，区分大小写，没有 byte 版本

对比,AnsiEndsStr, 一个字符串是否和另外一个字符串的尾巴相同，区分大小写，没有byte版本

支持MBCS.

单字节版本可以这样

```
Function EndsStr(const ASubByteStr,AByteStr:string):boolean;
```

```
begin
  result := ASubByteStr = RightBStr(AByteStr,Length(ASubByteStr));
end;

------------demo
procedure TForm1.Button9Click(Sender: TObject);
begin
 Memo1.Lines.Add(BoolToStr( EndsStr('bb','A侬'),true));
  Memo1.Lines.Add(BoolToStr( EndsStr('BB','ABB'),true));
  Memo1.Lines.Add(BoolToStr( EndsStr('BB','A侬'),true));
  Memo1.Lines.Add(BoolToStr( EndsStr('bb','ABB'),true));

end;

------------结果
False
True              ---只有第2个是，可见支持MBCS
False
False
```

15.对比,AnsiContainsStr,AnsiContainsText 和 AnsiIndexStr,AnsiIndexText，探测字符串在另外一个字符串中位置

AnsiIndexStr,AnsiIndexText，探测字符串在另外一个字符串中位置

返回的是0作为下标的，并且找不到就返回-1

和 AnsiContainsStr,AnsiContainsText 不同在于，AnsiContainsStr,AnsiContainsText 只探测有没有包含，而AnsiIndexStr,AnsiIndexText得到位置。

16.对比,AnsiIndexStr,AnsiContainsStr,AnsiMatchStr，匹配字符串

AnsiMatchStr 是在字符串数组(相当于c的2维字符数组）,中找字符串数组下标,下标是1开始。

而ansiIndexStr 是在字符串数组中找字符串 ,下标是0开始。

AnsiContainsStr 只是在字符串中找字符串是否存在。

17.对比,AnsiMatchStr,AnsiMatchText,在字符串数组中,匹配，查找字符串

ansiMatchStr 区分大小写
ansiMatchText 不区分大小写。

返回字符串在字符串数组中的位置

18.对比,ansiMidStr, MidStr,midBstr，取字符串中间串

AnsiMidStr 支持Byte,支持MBCS ,支持AnsiString

MidStr 支持Byte,不支持MBCS,支持AnsiString ,WideString

MidBstr 支持Byte,不支持MBCS,支持AnsiString, 不支持WideString

注意：不支持的意思是不能正确处理，或者根本编译不过去。

19.对比,AnsiReplaceStr,AnsiReplaceText, StringReplace 替换字符串

AnsiReplaceStr 支持MBCS,区分大小写,支持搜索全部匹配字符串，并替换

AnsiReplaceText 支持MBCS,不分大小写,支持搜索全部匹配字符串，并替换

StringReplace 支持MBCS,支持设置是否区分大小写，同时他也支持widestring,支持是否搜索整个字符串

简单来说stringreplace 是ansireplacestr, ansiReplaceText的超集。

这些字符串都不是支持递归模式替换的，比如说

ansiReplacestr('aabc','abc','bc');

替换后为

abc -------> 最终结果

不会再次被替换成

bc ---> 不会

如果要达到最终这个效果，再做一次或者若干次替换。

20.字符(不是字节),反序算法,AnsiReverseString

AnsiReverseString 将字符串按原来顺序完全翻转，支持MBCS

------例子

```
procedure TForm1.Button12Click(Sender: TObject);
begin
  Memo1.Lines.Add( AnsiReverseString('IF THEN ABS INC abc def 汉字'));
end;
```

----结果

字汉 fed cba CNI SBA NEHT FI

21.对比,AnsiRightStr, RightBStr,RightStr 取字符串右边若干字符串

ansiRightStr 支持MBCS, WideString转换到ansiString支持

RightStr 支持MBCS, 支持WideString

RightBStr 不支持MBCS,可能出现半个字，按字节索引

--例子

```
  Memo1.Lines.Add( RightStr('c汉a字b',3));
  Memo1.Lines.Add( RightBStr('c汉a字b',3));
  Memo1.Lines.Add( RightBStr('c汉a字b',2));
  Memo1.Lines.Add( AnsiRightStr('C汉a字b',3));
```

--结果

a字b

字b

諵　　---RIghtBstr 出现了半个字节

a字b

假如要截取的字符串是必须按字节，怎么办？ 可以这样

```
//取字符串右边若干字节，如果要防止半个汉字，就传递比目标大小-1的长度，AStr可以是MBCS ,aLen是字节长度
function MyRightStr(AStr:string;aLen:integer):string;
var t:string;
  l:integer;
begin
  t := AnsiReverseString (aStr); //--取反
  l := ByteToCharIndex(AStr,aLen); //探测左边取aLen字节，需要取得多少个字符
  result := LeftStr(t,l);
  result := AnsiReverseString(result);//再翻转回去
end;
---------演示


procedure TForm1.Button14Click(Sender: TObject);
begin
  //目标容器是8,将得到9字节
  Memo1.Lines.Add(MyRightStr('汉字abc字符',8));
  //下面将容器减少1字节
  Memo1.Lines.Add(MyRightStr('汉字abc字符',7));
```

```
end;
```

-------结果

字abc字符
abc字符

22.对比,AnsiStartsStr,AnsiStartsText,Pos 函数，取得字符串在是否是另外一个字符串中的开始

ansiStartsStr 判断是否是另外一个字符串的开始

AnsiStartsText 忽略大小写的判断

Pos 得到字符串在另外一个字符串中的位置，用1开始

-------demo

```
  procedure TForm1.Button15Click(Sender: TObject);
begin
  Memo1.Lines.Add(BoolToStr( AnsiStartsStr('汉字a','汉字abc字符')  ,True));
  Memo1.Lines.Add(BoolToStr( AnsiStartsStr('汉字A','汉字abc字符')  ,True));
  Memo1.Lines.Add(booltostr(AnsiStartsText('汉字a','汉字abc字符') , true));
  Memo1.Lines.Add(booltostr(AnsiStartsText('汉字A','汉字abc字符') , true));

end;
```

-------结果

True

False

True

True

==========同样可以用pos函数判断

```
function myStartsStr(aSubStr,AStr:string):boolean;
begin

  Result := Pos(aSubStr,AStr)=1;
end;
```

---举例

```
procedure TForm1.Button15Click(Sender: TObject);
begin

    Memo1.Lines.Add(booltostr(myStartsstr('汉字a','汉字abc字符'),
true));
  Memo1.Lines.Add(booltostr(myStartsstr('汉字A','汉字abc字符'),
true));
  Memo1.Lines.Add(IntToStr(Pos('汉字a','汉字abc字符')));
  Memo1.Lines.Add(IntToStr(Pos('汉字A','汉字abc字符')));
end;
```

---结果

True

False

1

0

23.ExtractStrings 函数陷阱

这个函数可以将一个字符串，按分割字符进行分割，返回若干行组成的 TStrings 对象。

这个函数有陷阱:除了按分割字符串分解之外，还会默认按 #13 或则 #10 字符进行分割， 并且返回数据不会包含空行。

举例说：

有字符串： 'abc,def'#13#10'd,dd'#13#10#13#10

问用本函数分割，用逗号做分割符，会得到什么？

答案 A

abc
Def#13#10d
Dd#13#10#13#10

还是答案 B

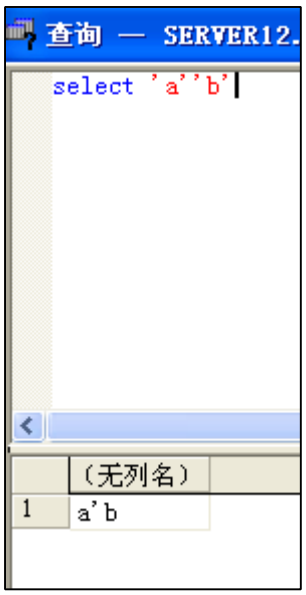
abc

def
d
dd

正确的是 B

24.应该用 quotedstr 还是 ansiQuotedstr?

Quotedstr 会在字符串两边加单引号，字符串中的单引号会替换成双引号，这是 sql 语句标准的规定，这样用来构造 sql 语句，就能够防止单引号注入。



这是数据库语句执行后效果。

25.CompareText 和 AnsiCompareStr 函数

CompareText 是按字符串每个字节进行大小比较的，和本地的字符集编码没有关系，要进行正确的 MBCS 字符集的字符串比较，要用 AnsiCompareStr.

26.With 语句陷阱

有一个程序如下：

```
Type
  TMyRec = record
    sName:string[20];
    Age:integer;
  End;

procedure Tform1.Button1click(sender :tobject);
Var r:tmyrecord;
Begin
  With r do
  Begin
    Name := 'abc';
  End;
End;
```

这个程序里面，本来作者的意图是给 r 这个类型的成员 name 给定一个值，但是由于后期修改了 tmyrec 的定义，导致没有 name 这个成员，但 tform 有这个成员，从而编译器也不能发现这个错误，导致 bug。

解决方法是尽量不使用 with 语句。

27.TForm1,TForm2 陷阱

有许多 delphi 的教程、书籍讲解的时候，喜欢用 Tform1 这样的类名，这对于作者来说，减轻了负担，殊不知，造成了大量 delphi 程序员在实际的工程实务中，也使用这样的命名习惯。导致程序后期维护负担非常的大，要看懂这样的程序，十分的困难。

解决的方法是：尽量起有意义的名字，比如 TFrmGoods ,TFrmList, TFrmEdit, TBtnAdd, TBtnAppend 等.

28.TDataModule 陷阱

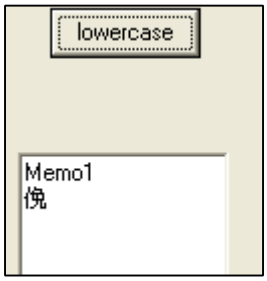
同样有大量初级教程，里面将上百个 TQuery, TDataset,TTable ,TDatasource 放在 Tdatamodule 里面，导致后期每次调试和查找问题都要转到这个单元去找其关联关系。

解决的方法是：将这些组件移动到使用这个组件的 TForm 里。Tdatamodule 只保留 TConnection 组件。

29.LowerCase 和 AnsiLowerCase

思考这个程序的结果是什么？

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  memo1.lines.add( LowerCase('佞'));
end;
```



你怎么也想不到会是这样的吧！修改一下：

```
begin

   memo1.lines.add( ansiLowerCase('俟'));
end;
```

这样就对了。

30.Uppercase 和 AnsiUpperCase

考虑如下程序

```
begin
  memo1.lines.add( uppercase('乏'));
end;
```

==========

```
begin

  memo1.lines.add( ansiuppercase('乏'));

end;
```

他们分别输出什么？

丄
乏

为什么会这样呢？

同类函数有 UpCase,AnsiUpperCase,UpperCase,StrUpper,AnsiStrUpper.

31.同样，AnsiUpperCaseFileName 和 uppercase 进行对比

会得出和 31# 问题相同的答案

32.Inttostr 陷阱

如下:

```
Var a:string;
Begin

A :=inttostr( Adoquery1.Fieldbyname("v").Value );
```

这里会发生什么问题呢？

我们知道数据库里面是存在 null 值的，当 v 字段的值是 null 的时候，就会抛出异常。

33.向后兼容函数或者过程列表

这些函数或者列表只是为了兼容老版本，新开发尽量不要使用。

AddExitProc 添加一个退出过程到退出列表.

AppendStr 增加动态分配的字符串到一个已经存在的字符串

AssignStr 给指定的指针赋一个动态分配的字符串

Close 关闭一个文件

DisposeStr 释放 newStr 分配的字符串

ExitCode 退出程序的返回值

LoadStr 加载字符串资源

NewStr 在堆里分配一个字符串内存

RaiseLastWin32Error 抛出最后发生的 win32 错误

StrAlloc 分配一个字符串缓冲区，返回指针指向这个字符串首字母，并结尾是 0 字符

StrBufSize 得到 StrAlloc 分配的缓冲区可以存放的字符串长度 .

StrDispose 释放字符串

StrNew 在堆里面分配内存，并复制字符串，返回指向他的指针

StrPas 转换 null 结束字符串到 ansistring(长字符串格式)

Swap integer 和 word 的字节顺序高低字节互换。

34.Ceil 函数

从 delphi7 到 delphi xe 都有这个问题，例子：

```
procedure TForm1.Button7Click(Sender: TObject);
begin
  Self.Caption := FloatToStr(Ceil(922337203685477.5807));

end;
```

其他结果是什么：

1566804070

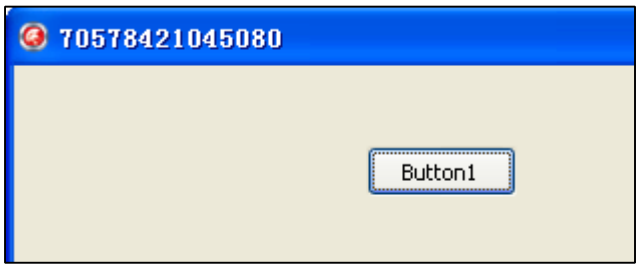
想不到吧。

按理说 ceil 和 floatstr 的参数类型都是 extended ，结果应该是整数部分吧？
但结果不正确。

这个程序用笔者写的转换函数能够达到比较精确的支持.DelphiXE2 测试通过，Delphi7 不支持。

```
procedure TForm1.Button1Click(Sender: TObject);
var t:OleVariant;
begin
  t :=Ceil(922337203685477.5807);
  Self.Caption := inttostr(OleVarToInt64(t));
```

```
end;
```



我们可以看到，如果不用上面代码，直接用 ceil 在转换 2147483648.0 的时候出现

在转换 2147483647.0 ,可见只支持 integer(32bit),找到这个函数，我们将其修改。如下：

```
function Ceil64(const X: Extended): int64;
begin
  Result := Int64(Trunc(X));
  if Frac(X) > 0 then
    Inc(Result);
end;


procedure TForm1.Button1Click(Sender: TObject);

begin
```

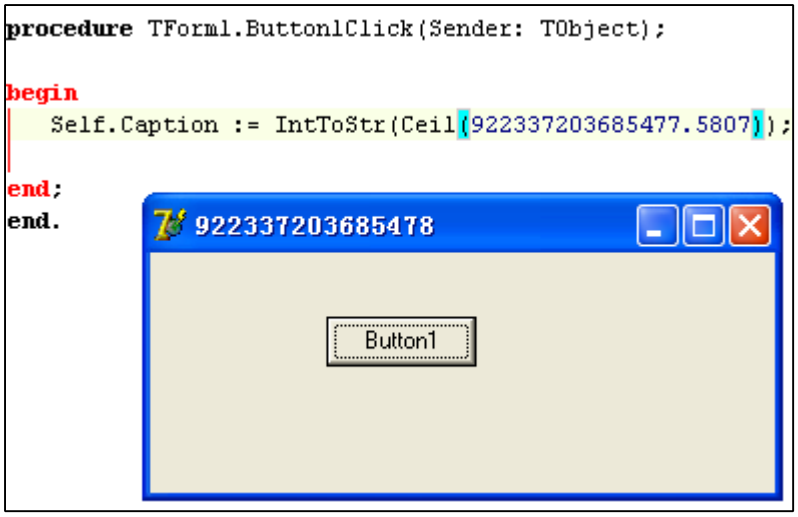
```
    Self.Caption := IntToStr(Ceil64(2147483648.0));

end;
```

结果

2147483648

正确。

再代入 extended 的最大数。

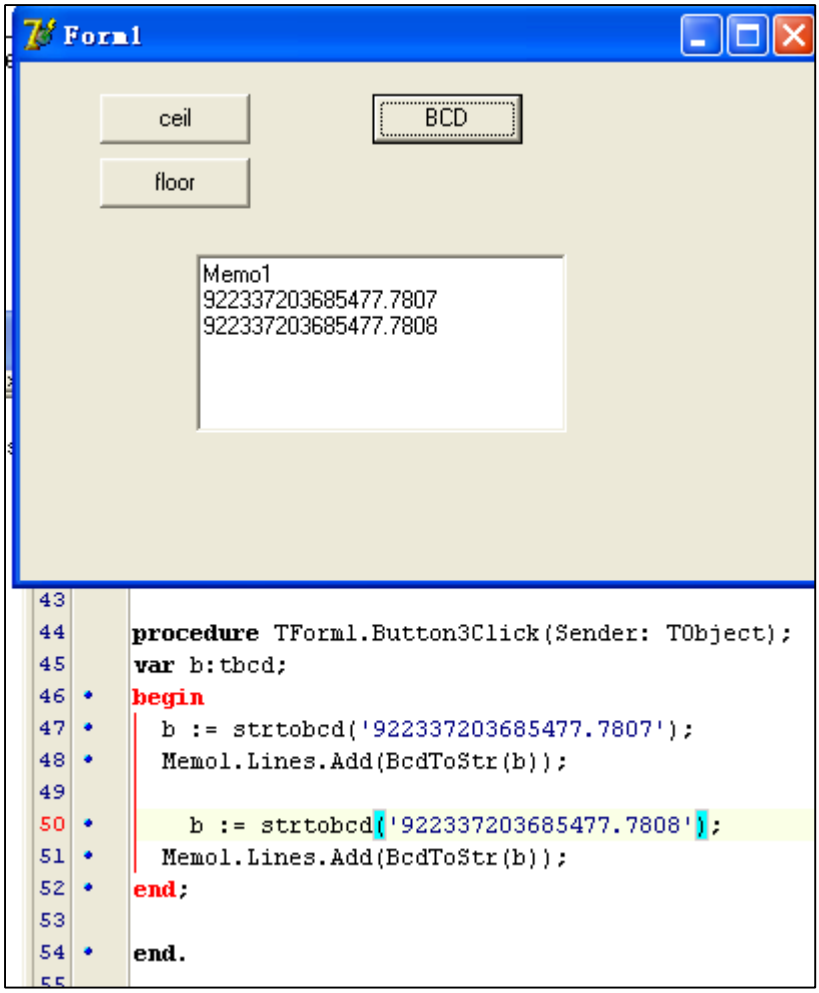


结果已经比较满意了。

35.Floor 函数陷阱.

这个函数和 ceil 有同样的 bug,经过修改后的代码在 delphi7 ,delphixe2 测试通过，都能很好支持 currency ,extended 部分，精度已经比较高了。

36.BCD 数据表示范围



通过此例子我们可以知道 BCD 码可以支持超过 currency 类型的数据。

37.DIVMOD 过程

这个过程可以实现 16 位整数的除法和取余数，但不支持 32 位和 64 位整数。

```
procedure TForm1.Button4Click(Sender: TObject);
var a,b:word;

begin
    DivMod(32767  ,2,a,b);
    Memo1.Lines.Add(IntToStr(a));
    Memo1.Lines.Add(IntToStr(b));

  DivMod(32768   ,2,a,b);
```

```
    Memo1.Lines.Add(IntToStr(a));
    Memo1.Lines.Add(IntToStr(b));

end;
```

上面的例子可以执行。

下面的例子报告错误。

```
procedure TForm1.Button4Click(Sender: TObject);
var a,b:word;

begin
      DivMod(655391,2,a,b);
      Memo1.Lines.Add(IntToStr(a));
      Memo1.Lines.Add(IntToStr(b));

end;
```

但该函数的被除数和又不只限制在 16bit integer.

比如

```
procedure TForm1.Button4Click(Sender: TObject);
var a,b:word;

begin
      DivMod(65539  ,2,a,b);
      Memo1.Lines.Add(IntToStr(a));
      Memo1.Lines.Add(IntToStr(b));

end;
```

也是可以通过的。

```
var a,b:word;

begin
    DivMod(655391,200,a,b);
    Memo1.Lines.Add(IntToStr(a));
    Memo1.Lines.Add(IntToStr(b));

end;
```

也可以通过.

```
procedure TForm1.Button4Click(Sender: TObject);
var a,b:word;

begin
    DivMod(655391,20,a,b);
    Memo1.Lines.Add(IntToStr(a));
    Memo1.Lines.Add(IntToStr(b));

end;
```

也行。

37.Ceil 和 INT 函数区别

```
    Memo1.Lines.Add(floatToStr(int (105.1234)));
    Memo1.Lines.Add(floatToStr(int (-105.1234)));
    Memo1.Lines.Add(floatToStr(ceil (105.1234)));
    Memo1.Lines.Add(floatToStr(ceil (-105.1234)));
```

结果是：

105

-105

106

-105

而

Memo1.Lines.Add(floatToStr(int (-922337203685477.5808)));

Memo1.Lines.Add(floatToStr(int (922337203685477.5807)));

Memo1.Lines.Add(floatToStr(ceil (-922337203685477.5808)));

Memo1.Lines.Add(floatToStr(ceil (922337203685477.5807)));

结果是

-922337203685477

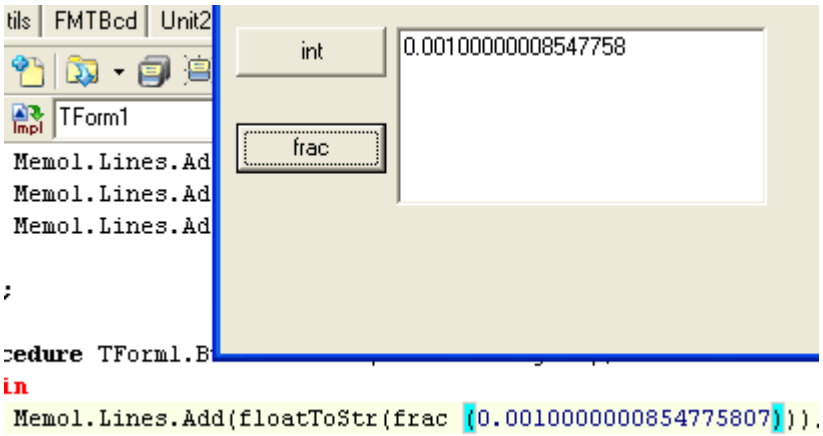
922337203685477

-922337203685477

922337203685478

注意这里的 ceil 是修改版本。

38.Frac 和 INT 函数

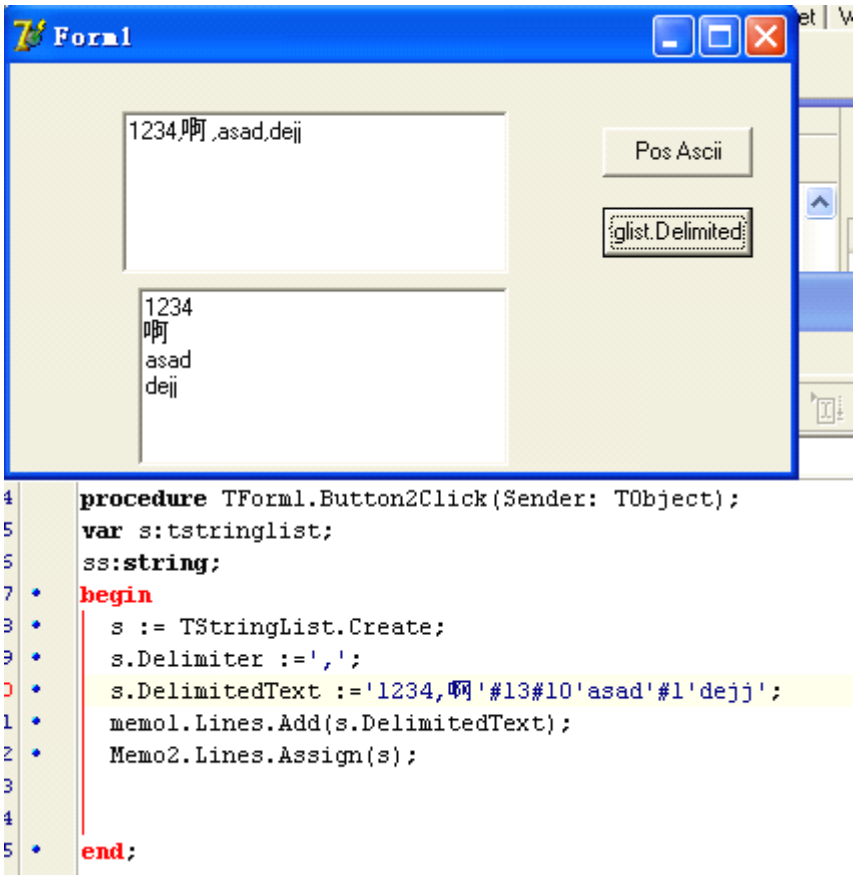


Frac 得到浮点数的小数部分，大约精确到小数点后面 17 位有效数字。而 Int 函

数得到浮点数的整数部分。

39.TStringList 陷阱

当我们使用 DelimitedText 的时候，其解析过程中，会遇到, #1--#31 字符也作为分割符号。




```
procedure TForm1.Button2Click(Sender: TObject);
var s:tstringlist;
ss:string;
begin
  s := TStringList.Create;
  s.Delimiter :=',';
  s.DelimitedText :='1234,啊'#13#10'asad'#1'dejj';
  memo1.Lines.Add(s.DelimitedText);
  Memo2.Lines.Assign(s);


end;
```

如上图，特别是我们的真实数据里面包含有分割符，就要更加小心了。会出现解析错误。

40.CompareValue 函数比较浮点数

我们知道浮点数的比较是不能直接比较大小的(特殊情况可以)，只能比较两个数字之间的误差。这个函数就用在一些统计上，比较两种统计方法出来的数据差别在许可范围之内，可以认为是一致的。

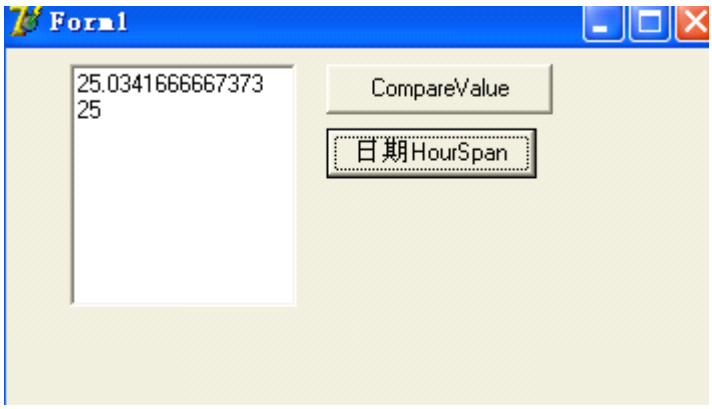
```
procedure TForm1.Button1Click(Sender: TObject);
var a,b:Extended;
begin
   a := 1/3;
   b := 3 /9    + 0.00000001;
   if CompareValue(a,b,0.0000001) =EqualsValue    then Memo1.Lines.Add('yes');
end;
```

例子里面的误差在小数点后面 7 位了，所以可以确定数据已经非常一致了。

41.TimeStamp 和 Mssql 的 TimeStamp 的不同点

在 Delphi 中 TimeStamp 是一个日期时间值，而 mssql 里面是一个大整数。

42.HoursBetween 和 HourSpan



例子:

```
 memo1.Lines.Add(FloatToStr(    HourSpan(StrToDate('2013-1-
1'),StrToDateTime('2013-1-2 01:02:03'))));
 memo1.Lines.Add(FloatToStr(      HoursBetween(StrToDate('2013-1-
1'),StrToDateTime('2013-1-2 01:02:03'))));
```

结果是

25.0341666667373

25

43.字符串转换成 Boolean 的自定义，如果不小心使用可能导致错误

```
procedure TForm1.Button3Click(Sender: TObject);
begin
   SetLength(FalseBoolStrs,10);
   FalseBoolStrs[0] := 'false';
   FalseBoolStrs[1] := 'off';
   FalseBoolStrs[2] := '男';
   FalseBoolStrs[3] := 'F';
   FalseBoolStrs[4] := 'nil';
   FalseBoolStrs[5] := '0';
   FalseBoolStrs[7] := 'null';

   writeln(      StrToBool('false'));
   writeln( StrToBool('off'));
   writeln( StrToBool('男'));
   writeln( StrToBool('F'));
   writeln( StrToBool('nil'));
   writeln( StrToBool('0'));
     writeln( StrToBool('null'));
end;
```

结果全部是

False

44.Delphi 中的序列化与反序列化

这是 Record 里面的数据是定长字符串的例子：

```
unit Unit1;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls;

type
  TForm1 = class(TForm)
    Memo1: TMemo;        {添加 Memo 显示内容}
    Button1: TButton;
    Button2: TButton;
    procedure Button1Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
  private
```

```
    { Private declarations }
  public
    { Public declarations }
  end;
var
  Form1: TForm1;
implementation
{$R *.dfm}
type
  TRec = record        {定义一个记录}
    name: string[8];
    age: Word;
  end;
//写入
procedure TForm1.Button1Click(Sender: TObject);
var
  rec: TRec;
  ms: TMemoryStream;
begin
```

```
    ms := TMemoryStream.Create;
    rec.name := '孙悟空'; rec.age := 8;
    ms.Write(rec, SizeOf(rec));
    rec.name := '猪八戒'; rec.age := 81;
    ms.Write(rec, SizeOf(rec));
    rec.name := '沙和尚'; rec.age := 18;
    ms.Write(rec, SizeOf(rec));
    ms.SaveToFile('c:\temp\rec.dat');
    ms.Free;
end;
//读取
procedure TForm1.Button2Click(Sender: TObject);
var
    rec: TRec;
    ms: TMemoryStream;
begin
    ms := TMemoryStream.Create;
    ms.LoadFromFile('c:\temp\rec.dat');
    Memo1.Clear;
```

```
  ms.Position := 0;
  while ms.Position < ms.Size do
  begin
    ms.Read(rec, SizeOf(rec));
    Memo1.Lines.Add(rec.name + ' ' + IntToStr(rec.age));
  end;
  ms.Free;
end;
End.
```

而对于不定长的 **Record** 里面的字符串。采用如下序列化，反序列化算法：

```
unit Unit1;
interface
uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls;
type
  TForm1 = class(TForm)
    Memo1: TMemo;        {添加 Memo 显示内容}
    Button1: TButton;
    Button2: TButton;
```

```
    procedure Button1Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;
var
  Form1: TForm1;
implementation
{$R *.dfm}
type
  TRec = record          {定义一个记录}
    name: string;
    age: Word;
  end;
//写入
procedure TForm1.Button1Click(Sender: TObject);
var
  rec: TRec;
  ms: TMemoryStream;
  len:integer;
begin
  ms := TMemoryStream.Create;
  rec.name := '孙悟空'; rec.age := 8;
  len := SizeOf(rec.name);
  ms.Write(len,SizeOf(len));
  ms.Write(rec.name,SizeOf(rec.name));
  len :=SizeOf(rec.age);
  ms.Write(Len,SizeOf(len));
```

```
    ms.Write(rec.age,SizeOf(rec.age));
    rec.name := '猪八戒'; rec.age := 81;
    len := SizeOf(rec.name);
    ms.Write(len,SizeOf(len));
    ms.Write(rec.name,SizeOf(rec.name));
    len :=SizeOf(rec.age);
    ms.Write(Len,SizeOf(len));
    ms.Write(rec.age,SizeOf(rec.age));
    rec.name := '沙和尚'; rec.age := 18;
    len := SizeOf(rec.name);
    ms.Write(len,SizeOf(len));
    ms.Write(rec.name,SizeOf(rec.name));
    len :=SizeOf(rec.age);
    ms.Write(Len,SizeOf(len));
    ms.Write(rec.age,SizeOf(rec.age));
    ms.SaveToFile('c:\temp\rec.dat');
    ms.Free;
end;
//读取
procedure TForm1.Button2Click(Sender: TObject);
var
    rec: TRec;
    ms: TMemoryStream;
    len:integer;
begin
    ms := TMemoryStream.Create;
    ms.LoadFromFile('c:\temp\rec.dat');
    Memo1.Clear;
    ms.Position := 0;
    while ms.Position < ms.Size do
```

```
    begin
      ms.Read(len,SizeOf(len));
      SetLength(rec.name,len);
      ms.Read(rec.name, len);
      ms.Read(len,SizeOf(len));
      ms.Read(rec.age, len);
      Memo1.Lines.Add(rec.name + ' ' + IntToStr(rec.age));
    end;

    ms.Free;
end;
end.
```

45.Delphi 中的哈希表

这是 IniFiles 里面的哈希表。

```
unit Unit1;


interface


uses

  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,

  Dialogs, StdCtrls;


type

  TForm1 = class(TForm)

    Button1: TButton;

    Button2: TButton;

    procedure FormCreate(Sender: TObject);

    procedure FormDestroy(Sender: TObject);

    procedure Button1Click(Sender: TObject);

    procedure Button2Click(Sender: TObject);
```

```
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

uses
  IniFiles;   //TStringHash 来自 IniFiles 单元
var
  Hash: TStringHash;

{ TStringHash 的功能非常简单, 如果需要更多功能应该使用: THashedStringList
```

TStringHash 与 THashedStringList、TStringList 最大的不同是:

TStringHash 的 Key 必须是 String; Value 必须是 Integer.

如果这不适合你的要求, 建一个 TMyHash 也不是难事 }

```
//建立哈希表
procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  Hash := TStringHash.Create(26);   //26 是表的初始大小, 可以省略使用默认值256
  for i := 65 to 90 do
  begin
    Hash.Add(Chr(i),i);   //如果表不够大,会自动增加的
  end;
end;

//读取
```

```
procedure TForm1.Button1Click(Sender: TObject);
var
   num: Integer;
begin
   num := Hash.ValueOf('Z');
   ShowMessage(IntToStr(num));    //90
end;

//修改、删除、清空
procedure TForm1.Button2Click(Sender: TObject);
begin
   Hash.Modify('Z',100);    //修改
   Hash.Remove('A');          //删除
   Hash.Clear;                    //清空

   {没了，就这些功能}
end;

procedure TForm1.FormDestroy(Sender: TObject);
```

```
begin
   Hash.Free;
end;
end.
```

采用的算法是:

```
function TStringHash.HashOf(const Key: string): Cardinal;
var
   I: Integer;
begin
   Result := 0;
   for I := 1 to Length(Key) do
      Result := ((Result shl 2) or (Result shr (SizeOf(Result) * 8 - 2))) xor
         Ord(Key[I]);
end;
```

46.另外一个第三方的快速哈希算法

```
{** A basic hash function. This is pretty fast, and fairly good general
      purpose, but you may want to swap in a specialised version. }
 function HashThis(const s: string): cardinal;
 var
   h, g, i: cardinal;
 begin
   if (s = '') then
     raise EHashInvalidKeyError.Create('Key cannot be an empty string');
   h := $12345670;
   for i := 1 to Length(s) do begin
     h := (h shl 4) + ord(s[i]);
     g := h and $f0000000;
     if (g > 0) then
       h := h or (g shr 24) or g;
   end;
   result := h;
 end;
```

使用方法更加简单:

```
uses Hashes

var
  hash:TObjectHash
begin
  //new
  hash:=TObjectHash.Create;

  //add value
  hash['test']:= TObject.create;
```

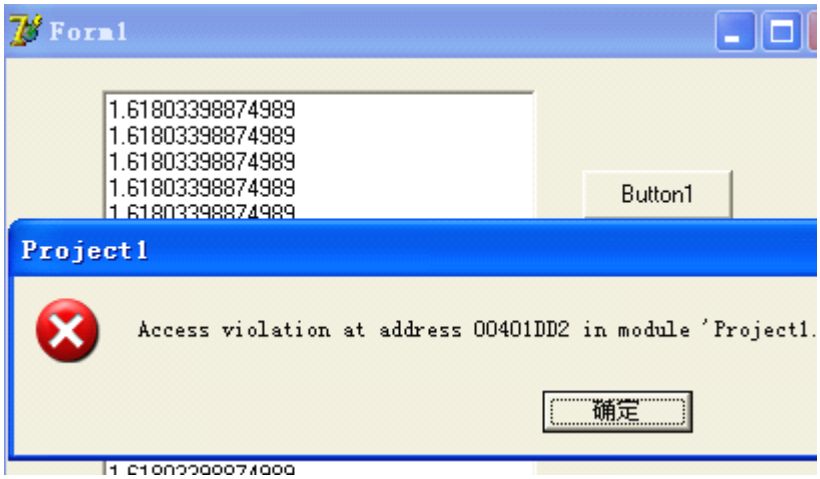
```
  //search
  if hash.Exists('test') then

  //foreach
  hash.Restart;
  while hash.Next do
  begin
     key:=Hash.CurrentKey;
     value:=hash[key];

  end

  //delete
  hash.Free;
end
```

该哈希类是 lazarus 和 delphi 通用的。

47.XE 中动态数组的错误忽略功能

如果错误使用这一个功能也可能导致错误.

如下程序，在 Delphi7 中，将会出现错误



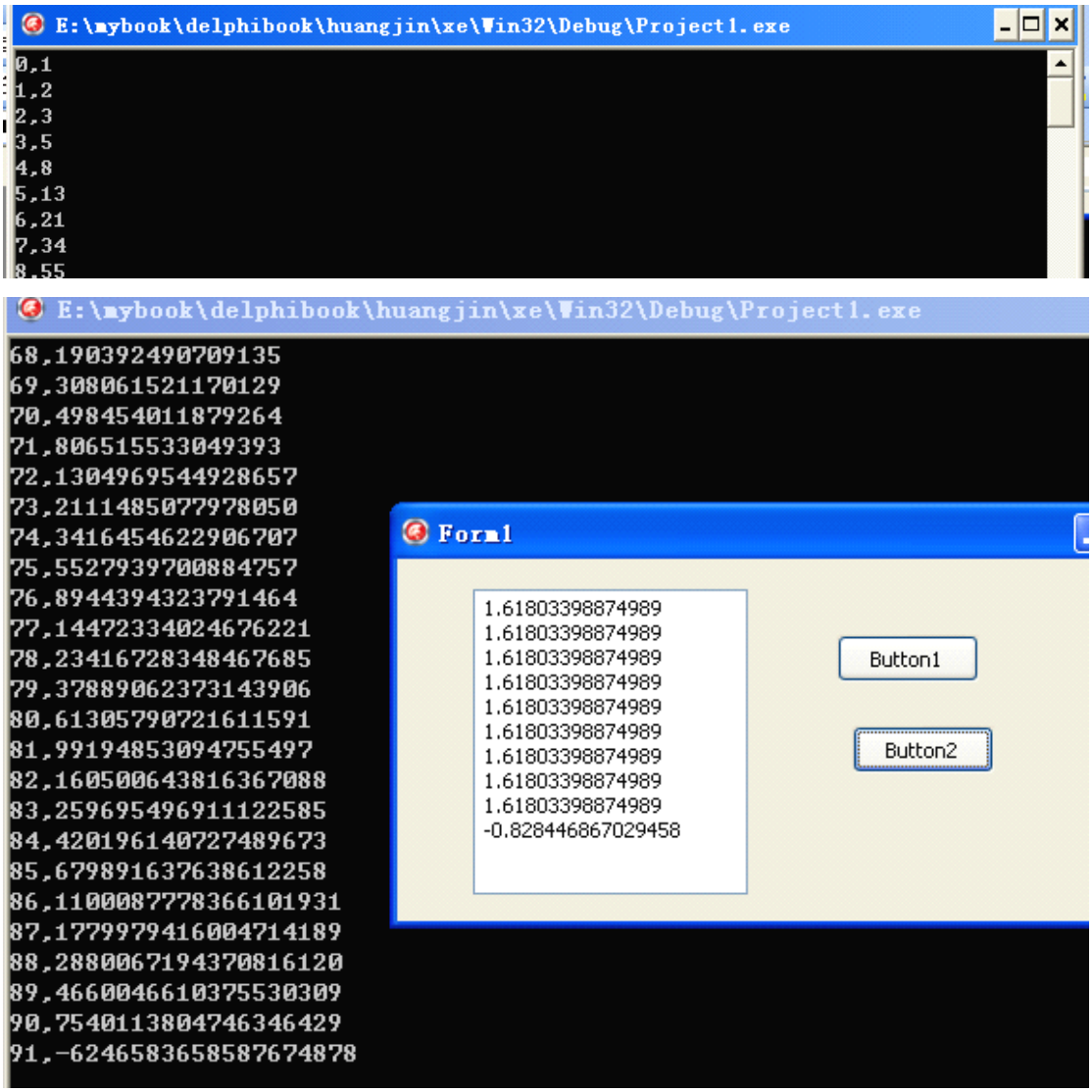

```
unit Unit1;

interface

uses
   Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
   Dialogs,
{ 斐波那契数列: 1、2、3、5、8、13、21、34、55、89、144 ... 等于前两数之和 }
{ 昂纳多·斐波那契(Leonardo Fibonacci, 1170-1240, 意大利数学家) }
 Types, StdCtrls;

type
   TForm1 = class(TForm)
      Button1: TButton;
      Memo1: TMemo;
```

```
    Button2: TButton;
    procedure Button1Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}


{参数 2 是要获取的总数}
procedure fbnc(var A: TInt64DynArray; Count: Integer);
var
  i: Integer;
begin
  SetLength(A, Count);
  A[0] := 1;
  A[1] := 2;
  for i := 2 to Count do A[i] := A[i-2] + A[i-1];
end;

{测试}
procedure TForm1.Button1Click(Sender: TObject);
```

```
var
   ns: TInt64DynArray;
   n: Int64;
begin
   fbnc(ns, 90);
   Memo1.Clear;

//   for n in ns do

   Memo1.Lines.Add(IntToStr(n));
end;

{查看其黄金分割比}
procedure TForm1.Button2Click(Sender: TObject);
var
   ns: TInt64DynArray;
   i: Integer;
begin
   fbnc(ns, 90);
   Memo1.Clear;
   for i := 0 to Length(ns) - 1 do
   begin
      if i = 0 then Continue;
      Memo1.Lines.Add(FloatToStr(ns[i] / ns[i-1]));
   end;
end;


End.
```

因为下标已经超出界限了

而在 Delphi XE2 里面不会报告错误。



用控制台调试，可以看到第 0 .... 直到第 90 个元素都是有值的。

程序来自 《万一的博客》,delphixe2 下面修改成这样，一样不报告错误：

unit Unit1;

interface

```
uses
  Winapi.Windows, Winapi.Messages, System.SysUtils, System.Variants,
System.Classes, Vcl.Graphics,
  Vcl.Controls, Vcl.Forms, Vcl.Dialogs, Vcl.StdCtrls ,System.types;

type
  TForm1 = class(TForm)
    Memo1: TMemo;
    Button1: TButton;
    Button2: TButton;
    procedure Button1Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}




procedure fbnc(var A: TInt64DynArray; Count: Integer);
var
```

```
    i: Integer;
begin
    SetLength(A, Count);
    A[0] := 1;
    A[1] := 2;
    for i := 2 to Count+1 do A[i] := A[i-2] + A[i-1];
end;

procedure TForm1.Button1Click(Sender: TObject);
var
    ns: TInt64DynArray;
    n: Int64;
begin
    fbnc(ns, 90);
    Memo1.Clea                  r;

    //for n in ns do

    Memo1.Lines.Add(IntToStr(n));
end;

procedure TForm1.Button2Click(Sender: TObject);
var
    ns: TInt64DynArray;
    i: Integer;
begin
    fbnc(ns, 90);
    Memo1.Clear;
    for i := 0 to Length(ns)+1    do
    begin
```

```
//        if i = 0 then Continue;
        Writeln(i ,',',ns[i]);

        Memo1.Lines.Add(FloatToStr(ns[i] / ns[i-1]));
    end;
end;


End.
```

48.Delphi,Lazarus 数组属性

数组类型的属性，可以用一个显示的属性，加[数组下标名] 加 default

有 default 的声明的属性，可以不使用属性的名称，直接象数组一样访问

比如

```
        {** Where to start our compact count from. }
        function FIndexMax: integer; override;

      public
        {** Items property. }
        property Items[const Key: string]: TObject read FGetItem
          write FSetItem; default;

        {** Destructor must destroy all items. }
        destructor Destroy; override;

    end;

implementation
```

这个是 THashObject 类，声明了一个属性

Property items[const key:string):Tobject read FGetItem write FSetItem;default;

所以访问这个类的对象的时候，可以这样

Var h:Thashobject;

Begin

....

```
H.items['abc'] := TObject.create;
```

也可以这样

```
H['abc'] := Tobject.create;
```

这样子写代码就更加简单明了。

49.Delphi 同单元的类保护域的访问

许多面向对象编程语言，都不允许在类外部访问保护级别的成员变量， 如下代码却允许这样访问，尽管在有的时候这个特性可以用来做一些特别的事情，但不建议这样用。

```
unit Unit1;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs;

type
  TForm1 = class(TForm)
    procedure FormCreate(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

  TTest=class
   protected
     protecteddata:integer;
   end;

  TTestHack = class(TTest);
```

```
var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.FormCreate(Sender: TObject);
var t:TTest;
begin
  t := TTest.create;
  TTestHack(t).protecteddata := 20;
  Self.Caption := IntToStr(TTestHack(t).protecteddata);
end;

end.
```

当一个类声明为另外一个类的类，并且是在同一个单元里面访问的话，就允许访问其保护级别成员。

此时在另一个单元去访问其保护级别的成员，就不行了。如下：

```
unit Unit2;

interface

uses Unit1;

function test:integer;
```

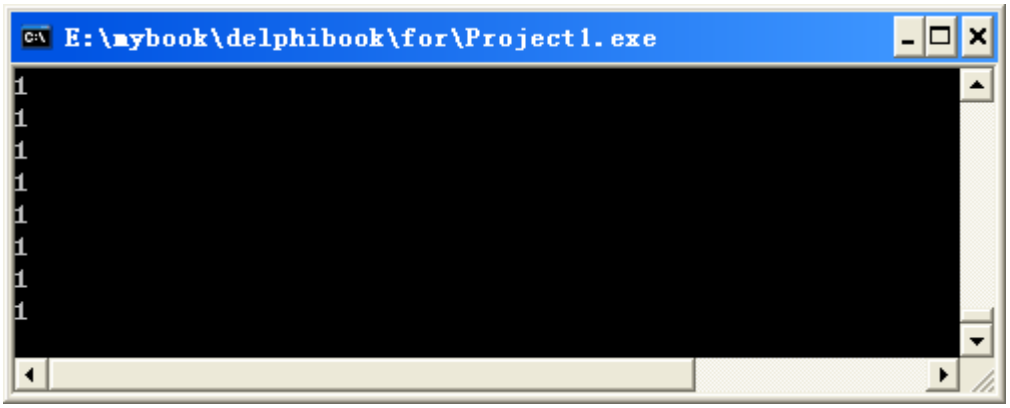
```
implementation

function test:integer;
var t:TTest;
begin
   t := TTest.create;
   Result := TTestHack(t).protecteddata := 20; //这句报告没有定义 protecteddata



end;
end.
```

注意：在 Lazarus 里面有一种严格的私有域修饰，在本单元也不能访问保护级别成员。

50.For 循环变量是否不可变

```
program Project1;

{$APPTYPE CONSOLE}

uses
   SysUtils;
var I:Integer;
      p:PInteger;
begin
   p := @I;
   for i := 1 to 10 do
   begin
      p^ := 1;
      writeln(i);
      Sleep(10);
   end;
end.
```

输出效果，是不断输出 1



可见 for 循环变量是可以修改的，本来 pascal 规定是 for 循环变量不可在循环内修改，防止不可理解，防止死循环，但这个例程序，打破了这个规律，编译器没

有对指针进行指向检查。

51.Case 语句是否可以用字符串作为判定变量

直到 DelphiXE2，Case 语句也不支持字符串作为条件，而 Lazarus 已经支持，比如

```
Var s:string;
Begin
   S := 'abc';
   Case s of
   'abc':
      Writeln('yes');
End;
```

52.Delphi 程序的 5 种单元结构

Pascal 程序单元文件有几种结构？

我们常用知道的有

DPR

Unit 两种标准结构，如下:

DPR 文件结构

```
program Project1;      { declares project identifier }

uses       { indicates units used by project... }
  Forms,       { ...including non-form units... }
  Unit1 in 'unit1.pas' {Form1};{ ...and form units }
{$R *.res}    { links in resource file }
begin     { start of main program block }
  Application.Initialize;
  Application.CreateForm(TForm1, Form1);       { auto-creates first form }
  Application.Run;     { runs the application }

end.{ end of main program block }
```

```
//Unit 标准结构
unit Unit1;

interface
```

```
uses    { List of units goes here }

    { Interface section goes here }

implementation

uses    { List of units goes here }

    { Implementation section goes here }

initialization
    { Initialization section goes here }

finalization
    { Finalization section goes here }

end.
```

还有一种编写组件需要用到的结构

```
package packageName;
        requiresClause;
        containsClause;
        end.
```

还有动态连接库结构

```
library MinMax;
function Min(X, Y: Integer): Integer; stdcall;
begin
   if X < Y then Min := X else Min := Y;
end;
function Max(X, Y: Integer): Integer; stdcall;
begin
   if X > Y then Max := X else Max := Y;
end;
exports
   Min,
   Max;
begin
end.
```

还有一种变体的单元结构

```
unit Unit2;


interface


implementation



begin
  Writeln('a');
  readln;
```

```
end.


//这个结构里面没有使用 initialization 关键字，但绿色部分相当于 initialization 的作用
//并且这个单元结构，加 initialization  节会报告错误
```

53.过程、函数、变量、属性之间的区分

例程如下：

```
program Project2;

{$APPTYPE CONSOLE}

uses
  SysUtils;

type TA = class
  private
    Fnow: String;
  public
    function getnow:String;
    property now:String read getnow;
  end;


type TB = class
  private
  public
    function now:String;
  end;

{ TA }

function TA.getnow: String;
begin
  result := 'abc';
```

```
end;

{ TB }

function TB.now: String;
begin
   result := 'g';
end;

function now:String;
begin
   result := 'def';
end;

var t:TA;
     b:tb;

begin
     t := TA.Create;
     b := TB.Create;

     with t do
     Writeln(now);


     with b do
     Writeln(now);

     with t,b do
     Writeln(now);
```

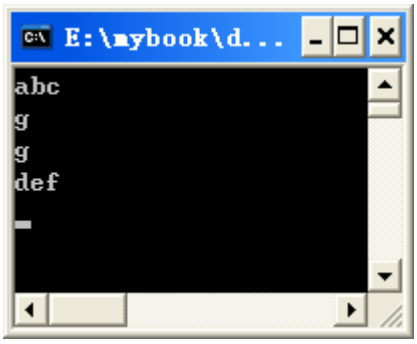
```
program Project2;

{$APPTYPE CONSOLE}

uses
  SysUtils;

type TA = class
  private
    Fnow: String;
  public
    function getnow:String;
    property now:String read getnow;
  end;


type TB = class
  private
  public
    function now:String;
  end;

{ TA }

function TA.getnow: String;
begin
  result := 'abc';
end;

{ TB }
```

```
function TB.now: String;
begin
   result := 'g';
end;

function now:String;
begin
   result := 'def';
end;

var t:TA;
     b:tb;

begin
     t := TA.Create;
     b := TB.Create;

     with t do
     Writeln(now);


     with b do
     Writeln(now);

     with t,b do
     Writeln(now);

     //Writeln(formatdatetime(Now));
```

```
    readln;
end.
```

```
    readln;
end.
```

问输出结果是什么？



总结：后声明的符号会覆盖先声明的符号，这就要求 delphi 程序编写的时候一定要知道引用的单元有哪些符号了，如果不小心覆盖了前面声明的符号，很有可能造成隐形的错误。

With 语句属性，方法冲突，会使用最后一个对象的成员。

在 Pascal 中，过程，函数可以省略() ,这就导致容易和变量、属性相混淆。这是 Pascal 的一个缺点。

54.数组型属性 和 数组 的不同点。

Creating array properties

创建数组型属性

Some properties lend themselves to being indexed like arrays. For example, the Lines property of

一些属性看上去像数组。比如 Tmemo 的 Lines 属性，是一个字符串列表，他组成了 Memo

TMemo is an indexed list of the strings that make up the text of the memo; you can treat it as an

文本，你能够将他们当作一个字符串构成的数组。

array of strings. Lines provides natural access to a particular(个别) element (a string) in a larger set of

Lines 提供了存取 memo 轻松的方法， 可以操作 memo 的所有数据行，无论其大小。

data (the memo text).

Array properties are declared like other properties, except that

数组型 属性 声明像其他属性，但是

The declaration includes one or more indexes with specified types. The indexes can be of any type.

声明包含 1 个或者多个下标并指定类型。下标可以是任何类型。

The read and write parts of the property declaration, if specified, must be methods. They cannot be fields.

读写属性声明，指定了，就必须是方法，也就是不能直接指定为变量(fields)。

The read and write methods for an array property take additional parameters that correspond to the indexes. The parameters must be in the same order and of the same type as the indexes specified in the declaration.
读写方法都有一个附加的参数，用来指定下标(index)。
声明，属性定义，实现，调用他们的顺序，类型，都必须一样。

There are a few important differences between array properties and arrays. Unlike the index of an array, the index of an array property does not have to be an integer type. You can index a property on a string, for example. In addition, you can reference only individual elements of an array property, not the entire range of the property.
和普通数组的重要不同点是。下标可以不是 integer 类型。下标可以是字符串。
比如你可以引用数组某个元素，而不是整个属性。

例程如下：

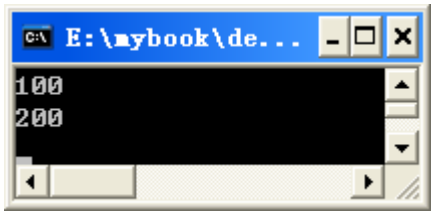
```
program Project2;

{$APPTYPE CONSOLE}

uses
  SysUtils;

type TA = class
    private
      FA:array of Integer;
      function getA(const index:Integer):integer;
      procedure setA(const index:Integer;const AValue:Integer);
```

```
  public

    property A[const index:integer]: Integer read getA write setA ;
  end;
```

```
{ TA }

function TA.getA(const index: Integer): integer;
begin
  if (index < Low(FA)) or (index>high(Fa))  then
  begin
    result := -1;
  end
  else
  begin
    Result := FA[index]
  end;

end;

procedure TA.setA(const index, AValue: Integer);
begin
  if (index < Low(FA))  then
  begin
    exit;
  end
  else
```

```
    begin
          if    index>high(Fa)      then
            SetLength(fa,index+1);

        FA[index] := AValue;
    end;

end;

var t:TA;

begin
    { TODO -oUser -cConsole Main : Insert code here }
    t := TA.Create;
    t.setA(1,100);
    Writeln(t.getA(1));

    t.A[2] := 200;
    Writeln(t.A[2]);

    readln;
end.
```

输出结果



这个例子定义的对象有一个属性 A ,通过下标就可以访问，数组类型属性和普通数组是不一样的，他的读写都是用方法进行了处理的，可以防止一些意外发生。

普通数组是必须使用与 integer 相当的有序数据类型作为下标的，而对象的数组类型属性可以不受这种限制。

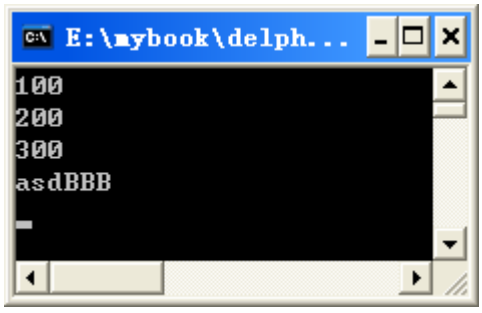
字符串下标例子：

```
program Project2;

{$APPTYPE CONSOLE}

uses
  SysUtils;

type TA = class
    private
      FA:array of Integer;
      function getA(const index:Integer):integer;
      procedure setA(const index:Integer;const AValue:Integer);
     function getB(const index: string): string;
    public

      property A[const index:integer]: Integer read getA write setA ;default;
      property B[const index:string]:string read getB;
    end;




{ TA }
```

```
function TA.getA(const index: Integer): integer;
begin
   if (index < Low(FA)) or (index>high(Fa))    then
   begin
      result := -1;
   end
   else
   begin
      Result := FA[index]
   end;

end;

function TA.getB(const index: string): string;
begin
   Result := index + 'BBB';
end;

procedure TA.setA(const index, AValue: Integer);
begin
   if (index < Low(FA))    then
   begin
      exit;
   end
   else
   begin
         if    index>high(Fa)      then
           SetLength(fa,index+1);

        FA[index] := AValue;
```

```
    end;

end;

var t:TA;

begin
    { TODO -oUser -cConsole Main : Insert code here }
    t := TA.Create;
    t.setA(1,100);
    Writeln(t.getA(1));

    t.A[2] := 200;
    Writeln(t.A[2]);

    t[0] := 300;
    writeln(t[0]);

    Writeln(t.B['asd']);

    readln;
end.
```

输出效果如下：



55.默认数组属性 和 数组

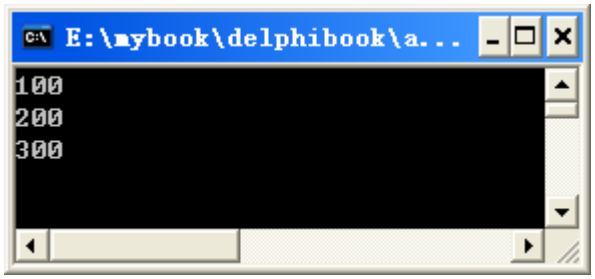
在上面例子中，我们扩展，注意红色部分：

```
program Project2;

{$APPTYPE CONSOLE}

uses
  SysUtils;

type TA = class
    private
      FA:array of Integer;
      function getA(const index:Integer):integer;
      procedure setA(const index:Integer;const AValue:Integer);
    public

      property A[const index:integer]: Integer read getA write setA ;default;
    end;




{ TA }

function TA.getA(const index: Integer): integer;
begin
  if (index < Low(FA)) or (index>high(Fa))   then
  begin
    result := -1;
```

```
    end
    else
    begin
       Result := FA[index]
    end;

end;

procedure TA.setA(const index, AValue: Integer);
begin
    if (index < Low(FA))    then
    begin
       exit;
    end
    else
    begin
          if    index>high(Fa)      then
            SetLength(fa,index+1);

         FA[index] := AValue;
    end;

end;

var t:TA;

begin
    { TODO -oUser -cConsole Main : Insert code here }
    t := TA.Create;
    t.setA(1,100);
```

```
    Writeln(t.getA(1));

    t.A[2] := 200;
    Writeln(t.A[2]);

    t[0] := 300;
    writeln(t[0]);

    readln;
end.
```

输出效果:



我们可以看到

```
    t[0] := 300;
```

这样的语法，看上去 t 是一个数组，但实际上是转换访问到 t.A 这个数组类型的属性。

看代码，不看其类型，很容易误解成 t 就是一个数组。

56.无法转换的 Tlist.last 指针

问题代码：

```
PT=^TmyRec;
  TmyRec=record
     sName:string[15];
     sex:Integer;
  end;
var


var
   list: TList;
   P1:PT;
begin
   list := TList.Create;
   New(p1);
   p1.sName:='Delphi';
   p1.sex:=99;
   list.Add(p1);
   TmyRec(list.Last).sname;    //这里不能强制转换
   Dispose(p1);
   FreeAndNil(list);
end;
```

正确的应该是:

```
program Project2;

{$APPTYPE CONSOLE}

uses
   Classes,
   SysUtils;

   type
PT=^TmyRec;
  TmyRec=record
     sName:string[15];
     sex:Integer;
  end;
```

```
var
   list: TList;
   P1:PT;
   p2:Pointer;
begin
   list := TList.Create;
   New(p1);
   p1.sName:='Delphi';
   p1.sex:=99;
   list.Add(p1);
   p2 := list.Last;                    //必须用一个临时变量
   Writeln(TmyRec(p2^).sname);    //
   Dispose(p1);
   FreeAndNil(list);

   readln;

end.
```

57.没有类型的参数

没有类型的参数，主要来自于编译器内部支持函数, 也就是 System 单元，比如 Copy 函数。

帮助信息以及翻译：

Returns a substring of a string or a segment of a dynamic array.

返回　一个 子串 的 一个字符串 或 一段 的 动态数组。(机械翻译)

返回一个字符串的子串，或者一个动态数组的一段(元素)。(人工翻译)

Unit

单元

System

System 单元(在 uses 不用添加，由编译器隐含包含)

Category

类别

string handling routines

字符串控制例程

Delphi syntax:

Delphi 语法(原型声明)

function Copy(S; Index, Count: Integer): string;

function Copy(S; Index, Count: Integer): array;

Description

描述

S is an expression of a string or dynamic-array type. Index and Count are integer-type expressions.
S 是一个字符串表达式或动态数组类型。Index 和 Count 是 Integer 类型表达式。

Copy returns a substring or subarray containing Count characters or elements starting at S[Index].
Copy 返回一个子串或者一个子数组包含 Count 个字符或者元素，开始位置在 S[Index]。

The substring or subarray is a unique copy (that is, it does not share memory with S, although if the
子串或者子数组是一个单独的拷贝(意思是,他和 S 不共享内存, 虽然(although)数组元素是指针或者对象，他们的本身是不会被复制的) (也就是只复制数组元素，不管他们指向的东西)
elements of the array are pointers or objects, these are not copied as well.)

If Index is larger than the length of S, Copy returns an empty string or array.
如果 Index 比 S 长度大，Copy 返回空字符串或者空数组。

If Count specifies more characters or array elements than are available, only the characters or
如果 Count 指定超过字符串或数组元素个数，那么仅仅返回从 S[Index]到变量 S 结束的
elements from S[Index] to the end of S are returned.
元素。

Note: When S is a dynamic array, you can omit the Index and Count parameters and Copy copies the entire array.

备注：如果 S 是动态数组，你能够省略 Index 和 Count 参数，进行复制整个数组的操作。

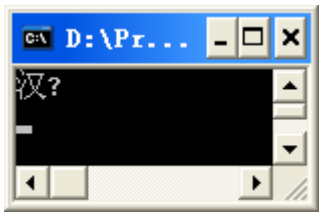
这篇帮助信息包含的信息量很大：

S 类型	Index 真实开始位置	Count 超长	Index 超界	备注
字符串	1	返回从 index 到尾部	返回空字符串	按 char 处理,unicode 版本默认是 widechar
动态数组(保存普通数据类型)	0	返回从 index 到尾部	返回空字符串	
动态数组(保存指针或对象)	0	返回从 index 到尾部	返回空字符串	不会复制对象本身
动态数组				Index,count 缺省，复制整个数组

例程：

```
//本程序测试  copy ,index =0 的情况，
//测试用例写法    System 单元,Copy 函数  ('汉字 abc123',0,3)
program Project2;

{$APPTYPE CONSOLE}
```

```
uses
   SysUtils;

var a,b:string;
begin
  a :='汉字 abc123';
  writeln(Copy(a,0,3));


  Readln;
end.
```



我们可以看到当下标 <=0 ,字符串复制还是从 1 开始。

复制是按 char 进行的。

```
//下面测试 Index = 1  的情况
program Project2;

{$APPTYPE CONSOLE}

uses
   SysUtils;

var a,b:string;
```

```
begin
  a :='汉字 abc123';
  writeln(Copy(a,1,3));

  Readln;
end.
```

//输出效果和前面一个例子相同，也就是下标从 1 开始.

在 Delphi 2009 以上默认支持 Unicode 版本，输出:

可见是按 Char (Unicode)版本操作的。

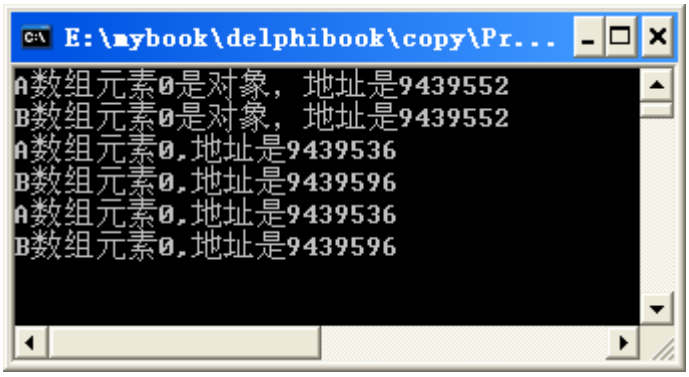
那么动态数组，保存的是对象，整体复制他们，里面的对象是否会复制 1 份拷贝呢？

```
program Project5;

{$APPTYPE CONSOLE}

uses
   SysUtils;


var a,b:array of TObject;
begin

  setlength(a,3);
try
```

```
  a[0] := TObject.Create;
  a[1] := TObject.Create;
  a[2] := TObject.Create;



  Writeln('A 数组元素 0 是对象，地址是',Integer(a[0]));

// writeln(Copy(a,-1,3));
  b := copy(a);
  Writeln('B 数组元素 0 是对象，地址是',Integer(b[0]));

  Writeln('A 数组元素 0,地址是',Integer(@a[0]));
  Writeln('B 数组元素 0,地址是',Integer(@b[0]));

  Writeln('A 数组元素 0,地址是',Integer(@(a[0])));
  Writeln('B 数组元素 0,地址是',Integer(@(b[0])));

// Writeln('A 数组元素 0,地址是',Integer((@a)[0]));
// Writeln('B 数组元素 0,地址是',Integer((@b)[0]));



  Readln;
  except
    on e:Exception do
    begin
    Writeln(e.message);
    readln;
    end;
  end;
```

end.

输出效果是:



可见数组复制，只复制了对象的指针，而没有复制对象。

58.writeln 不能支持的变量类型

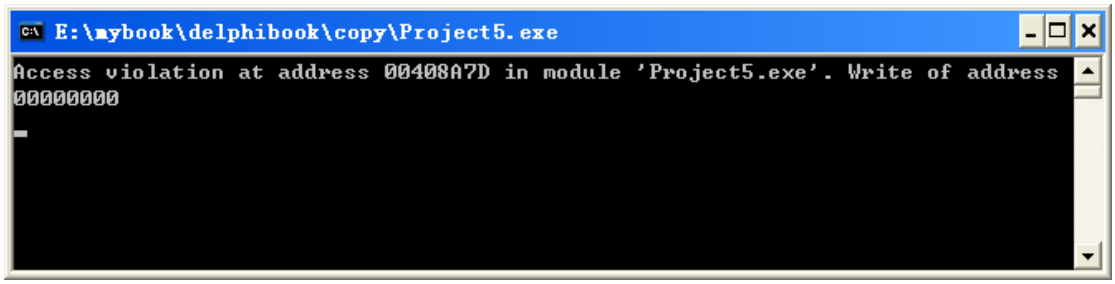
```
program Project4;

{$APPTYPE CONSOLE}

uses
  SysUtils;


var a,b:array of TObject;
begin
  SetLength(a,3);
  a[0] := TObject.Create;
  a[1] := TObject.Create;
  a[2] := TObject.Create;

  Writeln(Integer(@a[0]));
  Writeln(Integer(@a[1]));
  Writeln(Integer(@a[2]));

  Writeln(a[0]); //输出对象，也会报告错误,在 java 里面会直接输出对象地址
  writeln(Copy(a,-1,3)); //这里报告错误


 Readln;
end.
```

59.没有初始化的动态数组,例程:

```
program Project5;

{$APPTYPE CONSOLE}

uses
   SysUtils;


var a:array of TObject;
begin
    //这里没有初始化 ,应该进行内存分配，语句如下
    //setlength(a,3) ;//  分配 3 个单元

try
  a[0] := TObject.Create;
  a[1] := TObject.Create;
  a[2] := TObject.Create;


  Writeln(Integer(@a[0]));

// writeln(Copy(a,-1,3));


  Readln;
  Except   //这里将异常错误捕捉并输出，让程序员进行观察
    on e:Exception do
    begin
    Writeln(e.message);
```

```
    readln;
    end;
  end;

end.
```

输出效果：



60.for 循环变量是否可以使用复杂变量?

```
4
5     uses
6       SysUtils;
7
8     var a:array [0..0] of Integer;
9
0     begin
1       for a[0] := 1 to 10 do
2       begin
3         p^ := 1;
4         writeln(i);
5         Sleep(10);
6       end;
7     end
```

报告错误，[Error] Project3.dpr(11): For loop control variable must be simple local variable

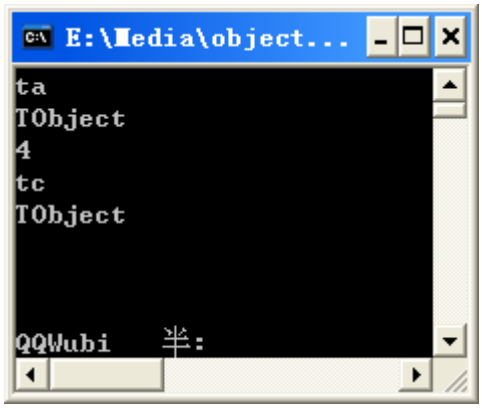
只能使用简单变量。

61. Delphi 的 Class,Object,Class(TObject)区别

例子程序如下：

```
program Project2;

{$APPTYPE CONSOLE}

uses
  classes,
  SysUtils;

type ta = class
   end;

type tb = object
    myage:Integer;
   end;

type tc=class(TObject)
  end;

var a:Ta;
   b :tb;
   c:tc;
begin
  a := ta.Create;
  //b := tb.create;
  c := tc.Create;

  Writeln( a.classname);
  Writeln( a.classParent.classname);


  Writeln(sizeof(b));

//  Writeln( b.classname);
//  Writeln( b.classParent.classname);

  Writeln( c.classname);
  Writeln( c.classParent.classname);
  Readln;
end.
```

输出结果如下：



由上我们可以得知：

```
type ta = class
```

和

```
type ta = class(TObject)
```

相等

```
type tb = object
```

很像

```
Type tb=record
```

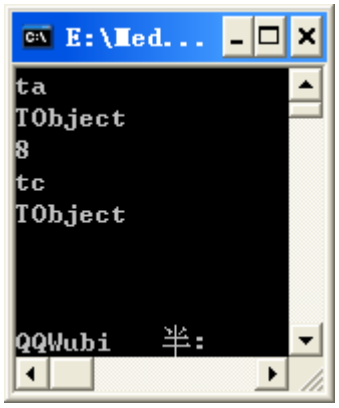
他没有继承于 TObject 的，这种语法是不推荐的，是为了兼容老的 pascal 而存在的。

修改一下：

```
program Project2;

{$APPTYPE CONSOLE}

uses
  classes,
  SysUtils;

type ta = class
   end;

type tb = object
    myage:Integer;
    myheight:Integer;
```

```
    end;

type tc=class(TObject)
  end;

var a:Ta;
    b :tb;
    c:tc;
begin
  a := ta.Create;
  //b := tb.create;
  c := tc.Create;

  Writeln( a.classname);
  Writeln( a.classParent.classname);


  Writeln(sizeof(b));

//  Writeln( b);
//  Writeln( b.classParent.classname);

  Writeln( c.classname);
  Writeln( c.classParent.classname);
  Readln;
end.
```

输出是：



也就是 tb 相当于一个 record 类型。

没有构造函数。
我们给 tb 类添加一个构造函数（假的）。如下：

```
program Project2;

{$APPTYPE CONSOLE}

uses
  classes,
  SysUtils;

type ta = class
   end;

type tb = object
    myage:Integer;
    myheight:Integer;
    procedure create(avalue:Integer);
   end;

type tc=class(TObject)
  end;

var a:Ta;
   b :tb;
   c:tc;
{ tb }

procedure tb.create(avalue: Integer);
begin
  myage:=avalue;
end;

begin
  a := ta.Create;
  //b := tb.create;
  c := tc.Create;

  Writeln( a.classname);
  Writeln( a.classParent.classname);


  Writeln(sizeof(b));

//  Writeln( b.classname);
//  Writeln( b.classParent.classname);
```

```
  Writeln( c.classname);
  Writeln( c.classParent.classname);

  //b.myage := 10;
  b.create(2);
  Writeln(b.myage);
  Readln;
end.
```

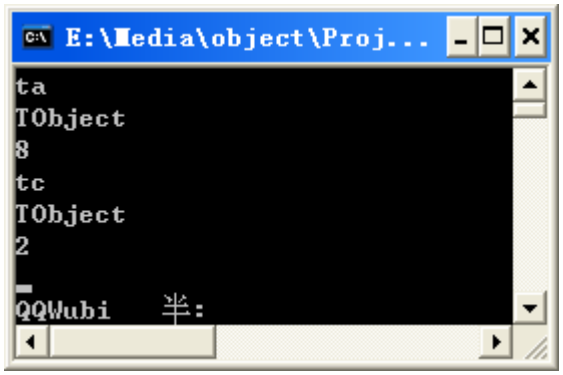
输出结果是

咱们再给他来一个真正的构造函数，如下：

```
program Project2;

{$APPTYPE CONSOLE}

uses
  classes,
  SysUtils;

type ta = class
   end;

type tb = object
    myage:Integer;
    myheight:Integer;

    //procedure create(avalue:Integer);
    public
      constructor create(avalue:Integer);
   end;
```

```
type tc=class(TObject)
  end;

var a:Ta;
   b :tb;
   c:tc;
{ tb }

constructor tb.create(avalue: Integer);
begin
  myage:=avalue;
end;

begin
  a := ta.Create;
  //b := tb.create;
  c := tc.Create;

  Writeln( a.classname);
  Writeln( a.classParent.classname);


  Writeln(sizeof(b));

//  Writeln( b.classname);
//  Writeln( b.classParent.classname);

  Writeln( c.classname);
  Writeln( c.classParent.classname);

  //b.myage := 10;
  b.create(2);
  Writeln(b.myage);
  Readln;
end.
```

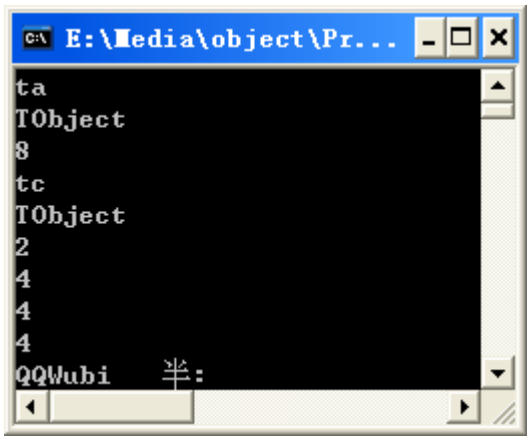
输出结果



再来对比一实例大小：

```
program Project2;

{$APPTYPE CONSOLE}

uses
  classes,
  SysUtils;

type ta = class
   end;

type tb = object
    myage:Integer;
    myheight:Integer;

    //procedure create(avalue:Integer);
    public
      constructor create(avalue:Integer);
   end;

type tc=class(TObject)
  end;

var a:Ta;
   b :tb;
   c:tc;
{ tb }
```

```
constructor tb.create(avalue: Integer);
begin
  myage:=avalue;
end;

begin
  a := ta.Create;
  //b := tb.create;
  c := tc.Create;

  Writeln( a.classname);
  Writeln( a.classParent.classname);


  Writeln(sizeof(b));

//  Writeln( b.classname);
//  Writeln( b.classParent.classname);

  Writeln( c.classname);
  Writeln( c.classParent.classname);

  //b.myage := 10;
  b.create(2);
  Writeln(b.myage);

  writeln(TObject.InstanceSize);
  writeln(a.instanceSize);
 //riteln(b.instanceSize);
  writeln(c.instanceSize);
  Readln;
end.
```



Tb 是没有 instancesize 的，只能用 sizeof

而对象的 instancesize 和类的 instancesize 是不同的，比如：

```
program Project3;

{$APPTYPE CONSOLE}

uses
  classes,
  SysUtils;

type ta = class
    myage:Integer;
    myheight:Integer;
    t:string;
    procedure aa;
   end;


var a:Ta;


{ ta }

procedure ta.aa;
begin
  myage :=10
end;

begin


 //    writeln(a.instanceSize);

  writeln(TObject.InstanceSize);
  writeln(ta.instanceSize);
      a := ta.Create;
  writeln(a.instanceSize);
  Readln;
end.
```

输出

而红色那句将会导致出错，也就是 ta.instancesize 是类方法，而用对象也可以调用。红色句子因为对象变量没有初始化，所以出错。

我们再来看原始声明：

```
Type
  TObject = class;
```

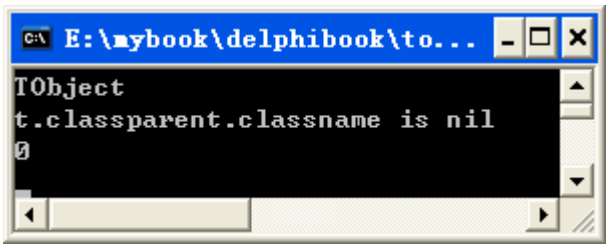
这里也省略了,，应该是 class(Tobject)了哦？那么其父类就应该是 TObject 了哦？也就是自己是自己的父类了？

编写代码：

```
program Project2;

{$APPTYPE CONSOLE}

uses
  SysUtils,
  classes;

type ta=class
end;
var
  t:TObject;
  a:ta;
begin
  t := TObject.Create;
  a := ta.Create;
  try
    Writeln(a.classparent.classname);
    if (t.classparent = nil) then
    Writeln(('t.classparent.classname is nil'));

      Writeln(Integer(t.classparent));
```

```
    readln;

    except
      readln;
    end;
end.
```

输出是:



可见编译器是对 Tobject 做了特殊处理的，其父类永远是 nil。

62.Delphi2007 新的类型

新类型 TBytes 和 原来的 types.pas 里面的 TByteDynArray = array of Byte;是一样的。(在 delphi 4 里面没有 types.pas)

Tbytes = array of bytes;	Delphi2007
Tbytes = array of bytes;	第三方 ehlib
TBytes = TArray<Byte>;	Delphi XE2 又重新定义了

所以 Ehlib 安装可能需要修改冲突点。

63.Delphi 不支持的指针操

Delphi 里面不支持

P[x] := y;

P := p + y;

这样的指针操作，而 Lazarus 是支持的。

那么有没有办法支持呢？

可以模拟支持，采用如下程序的方法：

program Project2;

{$APPTYPE CONSOLE}

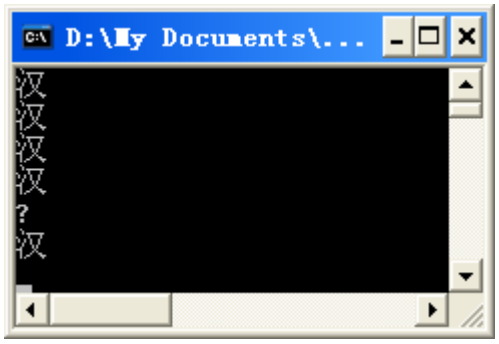
uses

```
    SysUtils;

    var p:PInteger;
        a:array [0..10] of integer;
begin
    a[0] := 1;
    a[1] := 2;

    p := @a[0];

    Writeln('元素 0=',p^);

    Writeln('指针前进 1');

    Inc(p);
    Writeln('元素 1=',p^);
    p^ := 3;
    Writeln('元素 1 修改后=',p^);

    Inc(p,-1);
    Writeln('指针回退后，元素 0=',p^);

    Readln;
End
```

64.Unicode 引起的问题

```
program Project1;

{$APPTYPE CONSOLE}

{$R *.res}

uses
  System.SysUtils ,System.StrUtils;

procedure readln();
begin
  System.readln;
end;

var s:string = '汉字';
var t:ansistring = '汉字';
begin

  Writeln(leftstr('汉字',1));
  Writeln(ansileftstr('汉字',1));

  Writeln(leftstr(s,1));
  Writeln(ansileftstr(s,1));

  Writeln(leftstr(t,1));
  Writeln(ansileftstr(t,1));

  Readln();
```

end.



在支持 unicode 的 delphi 版本里面，string 是 unicode 类型的。

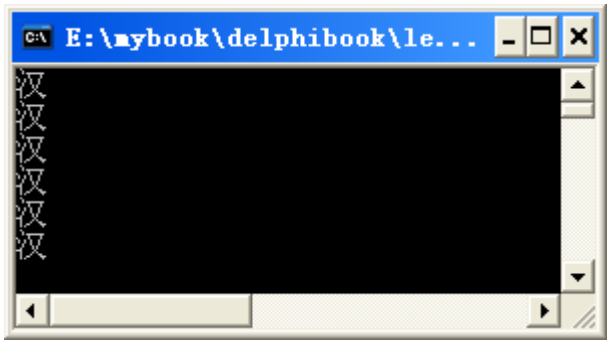
```
type String = UnicodeString;
```

按道理第 5 个函数，不应该出现无法输出啊，因为其类型都是 ansistring。

同样的程序在 delphi7 没有问题：

```
program Project1;

{$APPTYPE CONSOLE}

{$R *.res}

uses
    SysUtils , StrUtils;

var t:string = '汉字';
   s :AnsiString = '汉字';
begin
   try
```

```
    { TODO -oUser -cConsole Main : Insert code here }
    Writeln(leftstr(t,1));
    Writeln(ansileftstr(t,1));

    Writeln(leftstr('汉字',1));
    Writeln(ansileftstr('汉字',1));

    Writeln(leftstr(s,1));
    Writeln(ansileftstr(s,1));

    readln;
  except
    on E: Exception do
      Writeln(E.ClassName, ': ', E.Message);
  end;
End.
```



那么我们来看看两个的库函数区别：

XE2 的是

```
function LeftStr(const AText: AnsiString; const ACount: Integer): AnsiString; overload;
begin
  Result := Copy(AText, 1, ACount);
end;
```

也就是说，直接复制的实参的 AText 第 1 个字节。

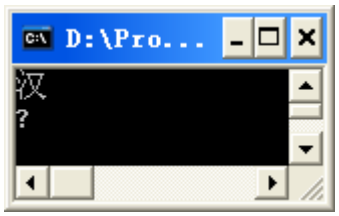
再来看看 Delphi7 的:

```
function LeftStr(const AText: AnsiString; const ACount: Integer): AnsiString; overload;
begin
  Result := Copy(WideString(AText), 1, ACount);
end;
```

先将字符转换成了 widestring,再复制的。复制的是整个完整的字符。

我们再看一个程序:

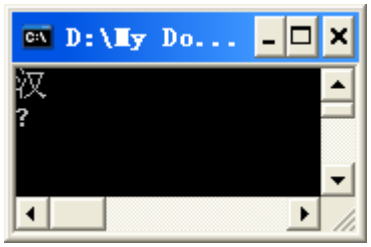
```
program Project1;

{$APPTYPE CONSOLE}

uses
  SysUtils;

begin
  Writeln(Copy(WideString('汉字'), 1, 1));
  Writeln(Copy('汉字', 1, 1));
  readln;
end.
```



可见在 Delphi7 下面，copy 函数，针对字符串，就是复制 1 个字符(ansichar,或者 widechar)。

修改一下，在 delphixe2 执行

```
program Project1;

{$APPTYPE CONSOLE}

{$R *.res}

uses
   System.SysUtils;

begin

  Writeln(Copy(WideString('汉字'), 1, 1));
  Writeln(Copy(ansistring('汉字'), 1, 1));
  readln;

end.
```



可以和 delphi7 的表现一致。

总结:copy 在老版本的 delphi 和支持 unicode 的版本一致，而 leftstr 变化了。

如果要达到相同的效果，可以这样：

```
program strleft;

{$APPTYPE CONSOLE}

{$R *.res}

uses
  System.SysUtils ,  System.StrUtils;

var t:string = '汉字';
  s :AnsiString = '汉字';
begin
  try

    { TODO -oUser -cConsole Main : Insert code here }
//    Writeln(leftstr(t,1));
    Writeln(ansileftstr(t,1));

//    Writeln(leftstr('汉字',1));
    Writeln(ansileftstr('汉字',1));

    Writeln(leftstr(WideString(S),1)); //进行手动转换，调用另外一个重载版本
//    Writeln(ansileftstr(s,1));

    readln;
  except
    on E: Exception do
      Writeln(E.ClassName, ': ', E.Message);
  end;
```

end.

在 Ctrl+鼠标跟踪的时候，找到的是第 1 个同名函数，而不是类型都完全一致的那个函数。

受此影响的函数，还有 RightStr,MidStr。

未完。。。待续