```
Structure t
   SVAR gStructData
   WAVE myData
   double someResults
EndStructure

Function AnalyzeThis(DFREF df)
   STRUCT t t
   StructFill/AC=1/SDFR=df t
   if( V_Error )
      print "no data"
      return -1
   endif
   if( strlen(t.gStructData) != 0 )       // this may have been autocreated
      StructGet/S t,t.gStructData
      print "previous result:", t.someResults
   else
      print "no previous result"
   endif
   t.someResults= mean(t.myData)
   print "current result:", t.someResults
   StructPut/S t,t.gStructData
   return 0
End

Function Demo()
   NewDataFolder/O/S root:data1
   Make/O myData= gnoise(1)
   SetDataFolder root:

   AnalyzeThis(root:data1)
   myData= gnoise(2)
   AnalyzeThis(root:data1)
End
```

Running Demo() then prints this:

```
no previous result
current result:  -0.0675801
previous result:  -0.0675801
current result:  -0.00269252
```

Static Functions

You can create functions that are local to the procedure file in which they appear by inserting the keyword *Static* in front of *Function* (see **Static** on page V-906 for usage details). The main reason for using this technique is to minimize the possibility of your function names conflicting with other names, thereby making the use of common intuitive names practical.

Functions normally have global scope and are available in any part of an Igor experiment, but the static keyword limits the scope of the function to its procedure file and hides it from all other procedure files. Static functions can only be used in the file in which they are defined. They can not be called from the command line and they cannot be accessed from macros.

Because static functions cannot be executed from the command line, you will have write a public test function to test them.

You can break this rule and access a static function using a module name; see **Regular Modules** on page IV-236.

Non-static functions (functions without the static keyword) are sometimes called “public” functions.