Module9: C Function Access

Dynamic Link Libraries (DLLs) are libraries of compiled subroutines. These functions can be accessed and run from within an APL workspace by means of the system function ⎕NA. Details may be found in the *Language Reference*.

§ 9.1 Declaring *dataTypes* of Arguments and Results

§§ 9.1.1 Quick View of DLLs and their Contents

The main sources of useful compiled C functions for general APL applications are to be found in the files advapi32.dll, gdi32.dll, kernel32.dll and user32.dll. These files reside in the ..\windows\system32\ directory under Windows XP. The Windows utility *QuickView* which used to be included with the Accessories of Windows 98 is no longer supplied with later versions of Windows. This facility was very useful as it allowed one to find out what functions are included in any given DLL.

However, a full list of usable Windows functions is given in the MSDN library, at http://msdn.microsoft.com/library/, under [Win32 and COM Development][Development Guides][Windows API][Windows API][Windows API Reference], where functions from various DLLs are listed by name or by category. As is often the case with Microsoft documentation, unless you know what you are looking for, the volume of almost unnavigable technical information can be disheartening. Nevertheless there are numerous other sources of information in books, such as *Microsoft Windows 32 API Programming Reference, Volumes 1 and 2* from Microsoft Press, and on the Internet.

§§ 9.1.2 The Meaning of the right Argument of ⎕NA

```
      ⎕NA CVec                                ⍝ Fixes function as defined by CVec
```

The character vector Rarg to ⎕NA contains a number of distinct parts. Essentially, there are 4 separate parts in the string.

1. The first describes the variable dataType of the result. This element may be elided if there is no result from the C function, or if none is required.
2. The second part is the name of the file containing the compiled C function. This may be the full path name if the DLL is not in a visible directory such as ..\system32\.
3. The third part, separated from the second by a bar (|), is the name of the function to be called from the DLL.
4. The fourth, and most complicated part, contains the specification of the variable dataTypes of the elements of the (right) argument to be given to the function being fixed from the DLL.

For example, the following CVec refers to a function called SystemParametersInfoA to be found in library User32.dll.

```
'I4 User32|SystemParametersInfoA I4 I4 >{I4 I4 I4 I4} I4'
```

The basic function result is a 4 byte integer and the argument to be supplied has 4 elements. The first, second and last are 4 byte integers and the third consists of a string of 4 byte integers which are to be used to capture the memory contents of a useful set of data indicated by C code pointers.

9.1.2.1 Fix the function `SystemParametersInfoA` in a clear workspace and display the result of the call

```
SystemParametersInfoA 48 0 (0 0 0 0) 0
```

§§ 9.1.3 Discovering C Function Syntax

Let us start with a very simple, but very useful, example. The function **GetSystemMetrics** takes an integer argument and returns an integer result. The meaning of the argument and result can be found in http://msdn.microsoft.com/library/default.asp?url=/library/en-us/sysinfo/base/getsystemmetrics.asp

According to this documentation, "the **GetSystemMetrics** function retrieves various system metrics (widths and heights of display elements) and system configuration settings. All dimensions retrieved by **GetSystemMetrics** are in pixels." The calling syntax is given as:

```
int GetSystemMetrics(
  int nIndex
);
```

and the single parameter argument, nIndex is defined as "the system metric or configuration setting to retrieve." There then follows a table of possible values and their meaning. Int is a 32 bit signed integer.

9.1.3.1 Define function `GetSystemMetrics` in your workspace and determine the meanings of the first 20 calls.

```
      ⎕NA'I4 user32|GetSystemMetrics I4'
      GetSystemMetrics¨⍳20
1024 17 17 26 1 1 3 3 17 17 32 32 32 32 20 1280 968 0 1 17
```

Hint: See function `#.WDesign.GetSystemMetrics` in the supplied workspace WDESIGN.DWS for a good short explanation of each metric.

§ 9.2 Examples of C Function Calls

§§ 9.2.1 Simple Examples

Another simple useful example of an API call is the function **GetCurrentDirectory** which retrieves the current directory for the current process. Its syntax is documented as:

```
DWORD GetCurrentDirectory(
  DWORD nBufferLength,
  LPTSTR lpBuffer
);
```

In this case there is a result described as a DWORD, and a 2-parameter argument described as a DWORD and an LPTSTR. The meanings of these parameters are defined as "the length of the buffer for the current directory string..." and "a pointer to the buffer that receives the current directory string..."

This translates to

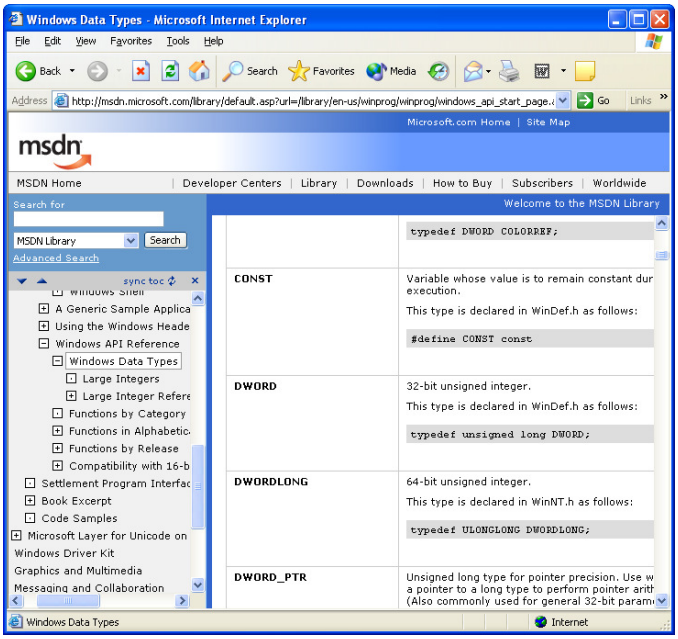
```
      ⎕NA'kernel32|GetCurrentDirectoryA U4 >0T'
```

because a DWORD is defined (see below) as a 32 bit (4 byte) unsigned integer, which translates to `U4`, and LPTSTR is a pointer to a null-terminated character string, which translate to `>0T` in `⎕NA` syntax. The `>` indicates that the contents of the pointed memory location assumed by the template argument will be used and overwritten by pointer-type *output* from the C function. The zero implies a null-terminated string, and the T means char, an 8-bit Windows (ANSI) character. **GetCurrentDirectoryA** is the ANSI version of the function and **GetCurrentDirectoryW** is the Unicode version.

So a call such as

```
      GetCurrentDirectoryA 100 200
C:\Dyalog90
```

fills in an output buffer at pointer position 200 with an ANSI string of up to 100 characters long, the last character being a terminating null character.



There are 8 bits in a byte. Each bit can be a **1** (ON) or a **0** (OFF) so there are `2*8↳256` possible combinations of bits in a byte.

```
┌─┐┌─┐┌─┐┌─┐┌─┐┌─┐┌─┐┌─┐
│1││0│1│1│0│1│1│1│
└─┘└─┘└─┘└─┘└─┘└─┘└─┘└─┘
 8 bits = 1 byte
```

ASCII characters use only 7 bits, giving `2*7↳128` combinations. This used to be sufficient space for all the common letters and symbols on telex and teletype terminal keyboards. The 8th bit was often used in communications for a parity check. **ANSI** characters use all 8 bits and therefore allow 256 distinct characters to be defined – hence the length of `⎕AV`. **Unicode** characters take 16 bits (2 bytes), giving `2*16↳65536` distinct combinations. Therefore Unicode allows 65,536 different characters to be defined.

If interpreted as a number, then 2 bytes can represent any number between 0 and 65535 for (unsigned) `U2`, or any number between ¯32768 and 32767 for (signed) `I2` in which the first bit is used for the sign.

```
      (16⍴2)⊤(2*15)-1   ⍝ Biggest positive number
0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
      (2*15)-1↳32767

      (16⍴2)⊤2*15       ⍝ Smallest negative number (2's complement)
1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
      -2*15↳¯32768
```

The GetCurrentDirectoryA function returns the current working directory as a string via a pointer reference. Be sure to allocate enough space for the string or you might get a GPF! Incorrect coding of `⎕NA` function argument parameters is the most common cause of SysError 999 crashes in Dyalog APL.

§§ 9.2.2 More complex Examples

We shall look at 3 functions in the ADVAPI32.DLL library, which together give read and write ability to and from the Windows Registry. They are therefore very useful in many APL applications. The functions are **RegCreateKeyExA**, which returns a handle to a given registry key, creating it if it does not already exist, **RegQueryValueExA**, which "retrieves the type and data for a specified value name associated with an open registry key", and **RegSetValueExA**, which "sets the data and type of a specified value under a registry key."

These three functions, and others, are well described in a workspace kindly supplied by Alex Kornilovski, to be found in the [Download] [AKUTILS] section of www.dyalog.com, or in the very large collection generously provided by Ray Cannon to be found in [Pocket APL][Downloads][padlls.dws].

The `GetEnvironment` method of Root with argument `'IniFile'` returns, by default, the Dyalog registry subkey of the HKEY_CURRENT_USER key.

```
      #.GetEnvironment'IniFile'↳'SOFTWARE\Dyadic\Dyalog APL/W 9.0'
```

RegCreateKeyExA takes this key and returns a handle to it for use with the other 2 functions. The function syntax is described as:

```
LONG RegCreateKeyEx(
  HKEY hKey,
  LPCTSTR lpSubKey,
  DWORD Reserved,
  LPTSTR lpClass,
  DWORD dwOptions,
  REGSAM samDesired,
  LPSECURITY_ATTRIBUTES lpSecurityAttributes,
  PHKEY phkResult,
  LPDWORD lpdwDisposition
);
```

which we may implement as

```
      ⎕NA'ADVAPI32.dll|RegCreateKeyExA U <0T I <0T I I I >U U'
```

Notice `I` and `U`, without a numeric qualifier, implies width 2 for 16-bit DLLs or width 4 for 32-bit DLLs.

```
      Key←2147483649      ⍝ HEX 0x80000001 = HKEY_CURRENT_USER
      SubKey←#.GetEnvironment'IniFile'
      Access←983103       ⍝ HEX 0xF003F    = KEY_ALL_ACCESS

      rarg←Key SubKey 0 '' 0 Access 0 0 0
      ⊢RegCreateKeyExA rarg↳600
```

Read this as "it happens to be true that `RegCreateKeyExA` applied to the above (enumeration) key and subkey returns the handle 600." We need to use 600 for the next function call. (The special numbers quoted above are (slightly) more meaningful in their (documented) HEX representation.)

9.2.2.1 Compare `rarg` with the C function syntax above and the description of the parameters in http://msdn.microsoft.com/library/default.asp?url=/library/en-us/sysinfo/base/regcreatekeyex.asp.

Symbol `<` indicates a pointer to *input* to the DLL function.

The **RegQueryValueEx** function retrieves the type and data for a specified value name associated with an open registry key, identified by its handle.

```
LONG RegQueryValueEx(
  HKEY hKey,
  LPCTSTR lpValueName,
  LPDWORD lpReserved,
  LPDWORD lpType,
  LPBYTE lpData,
  LPDWORD lpcbData
);
```

This syntax may be translated to

```
      ⎕NA'I ADVAPI32.dll|RegQueryValueExA U <0T I =I >0T =I4'
```

The result is a LONG which we can identify as integer `I` or `I4`. The key is an unsigned integer, `U`. The name of the value of interest is to be input and is interpreted as a null-teminated character string.

```
      Key←600     ⍝ Handle
      ValueName←'log_file'
      DataType←1 ⍝ String data type (REG_SZ)
```

Given the `⎕NA` specification above we expect `RegQueryValueExA` to return a 4 element vector representing the result (`I`), the dataType (`=I`), the data (`>0T`) and the number of bytes used (`=I4`). Note that the equals sign (`=`) is used to specify parameters which are both input (`<`) and output (`>`).

```
      DISPLAY RegQueryValueExA Key SubKey 0 DataType 255 255
+→-----------------------------------+
|      +→-----------------------+     |
| 0 1  |C:\Dyalog90\default.dlf| 24  |
|      +-----------------------+     |
+∈-----------------------------------+
```

If we were not concerned with anything but the value Data, we might use the specification

```
      ⎕NA'ADVAPI32.dll|RegQueryValueExA U <0T I <I >0T <I4'
```

but it is obviously not advisable to completely ignore error flags.

The complementary function, **RegSetValueEx**, sets the data and type of a specified value under a registry key. The C function has syntax declared as:

```
LONG RegSetValueEx(
  HKEY hKey,
  LPCTSTR lpValueName,
  DWORD Reserved,
  DWORD dwType,
  const BYTE* lpData,
  DWORD cbData
);
```

This may be translated as

```
      ⎕NA'I ADVAPI32.dll|RegSetValueExA U <0T I I <0T I4'
```

9.2.2.2 Call functions `RegCreateKeyExA`, `RegQueryValueExA` and `RegSetValueExA` with suitable arguments to access, replace and create various registry entries.

§§ 9.2.3 Other API Calls

The workspace SQAPL to be found in the WS directory of your Dyalog APL installation is a very useful example of a system written in C and linked to APL via ⎕NA. The system allows access to ODBC data sources and is described in chapter 16 of the Dyalog APL *Interface Guide*.

9.2.3.1 In library User32.DLL there is a function called SetCursorPos which moves the cursor to the specified (X,Y) screen coordinates, in pixels. The C function syntax is specified as
BOOL SetCursorPos(int *X*, int *Y*);
Define this function in you workspace and check that it works as expected.

9.2.3.2 In library User32.DLL there is a function called **FindWindowA**. This function can determine if another application is currently running on your system. It accepts two string arguments, one for the class name of the application, and another for the window title bar caption. The result is an unsigned integer giving the handle to a window. The first argument is also an unsigned integer which can be given the value zero. The second argument is a null-terminated string containing the caption of the window. Define this function in your workspace. Create a `Form` in the workspace with some specific `Caption`. Look at the `Handle` property of this `Form` and compare that with the result of the function FindWindowA.

9.2.3.3 As demonstrated by Thomas Gustaffson, the function WinExec in Kernel32.DLL runs a specified application and may be used to replace a call such as

```
      ⎕CMD'Notepad' ''
```

The first parameter argument of WinExec is a pointer to a null-terminated character string that contains the command line for the application, the second argument is an integer such that 1 means "show window". Define this function and use it to replace the ⎕CMD command above. Note that, as in the ⎕CMD case, the first parameter must be surrounded by double-quotes if there are any spaces in the string.

There are a number of working examples in the supplied workspaces QUADNA.DWS, NTUTILS.DWS and WDESIGN.DWS. QUADNA contains a particularly interesting example, ChooseColor, which requires a pointer to a structure which itself contains a pointer to an array. Notice that this workspace makes calls to the library DYALOG32.DLL. There are many other examples of ⎕NA calls in freely available workspaces such as those generously supplied by Alex Kornilovski and Ray Cannon.

Sometimes the primary difficulty with external functions is not in the construction of arguments but in the interpretation of the result. For example, `GetVersion` should be defined as taking no arguments and returning an integer result. To decipher the meaning of this result requires appropriate documentation, as can be seen from the code snippet below:

```
 code←GetVersion
 code←(32⍴2)⊤code
 code←(⊃code)(2⊥⍣¯1⊢8↑code)
:Select code
 :CaseList (0,¯3 4 5)
     R←'Windows NT/2000/XP'
 :Case 1 4
     R←'Windows 95/98'
 :Case 1 3
     R←'Win32s with Windows 3.1'
 :Else
     R←'?'
 :End
```

This function has been superseded by `GetVersionExA` in the same library. This new function returns a more complex structure and may be fixed by

```
⎕NA'I kernel32|GetVersionExA ={I4 I4 I4 I4 I4 T[128]}'
```

§ 9.3 Harnessing large C Libraries

§§ 9.3.1 Fastest Fourier Transform in the World

FFTW is a free C subroutine library for computing the discrete Fourier transform in one or more dimensions, of arbitrary input size, and of both real and complex data, as describes in the owner's web site http://www.fftw.org/.

The C function DFT.C below has been written on top of some of the principle calls to FFTW in order to create a function with relatively straight-forward arguments for a `⎕NA` call. It is therefore probably not still the Fastest Fourier Transform in the World, but it is nevertheless a very useful addendum to Dyalog APL.

DFT.C

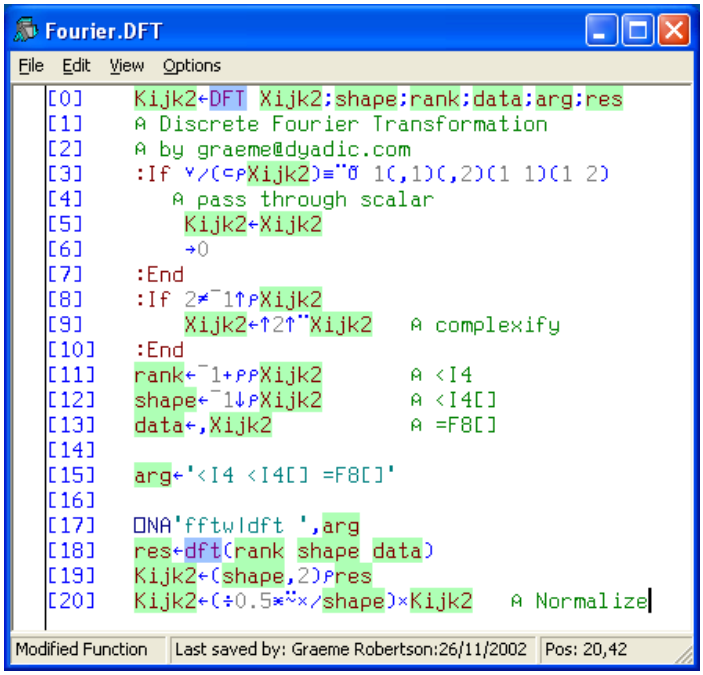
```
/* dft.c discrete fourier transform */

#include <fftw.h>

__declspec(dllexport) void dft(int *rank, const int *shape, double *data)
{
  fftwnd_plan plan;
  plan = fftwnd_create_plan(*rank, shape, FFTW_FORWARD, FFTW_IN_PLACE);
  fftwnd_one(plan, (void*)data, 0);
  fftwnd_destroy_plan(plan);
}
```

This function and the inverse function IDFT.C are to be found in the supplied file FFTW.DLL.

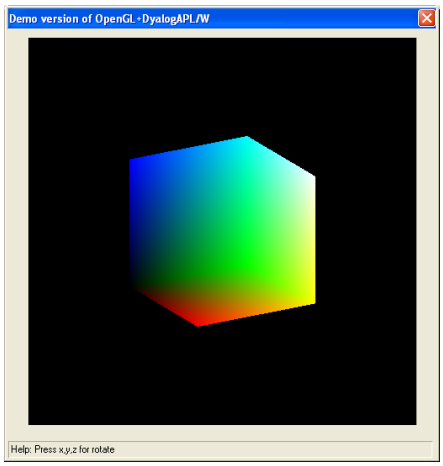
9.3.1.1 Given the above C function header, compose the right argument of `⎕NA` and compare with line [15] in the function below.



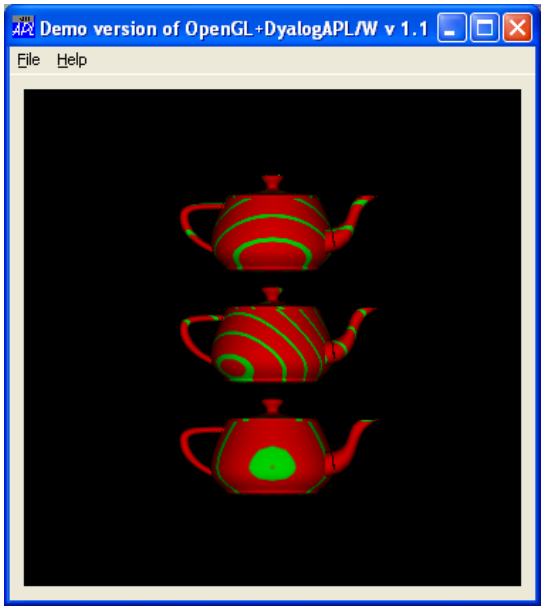
§§ 9.3.2 Open Graphics Library

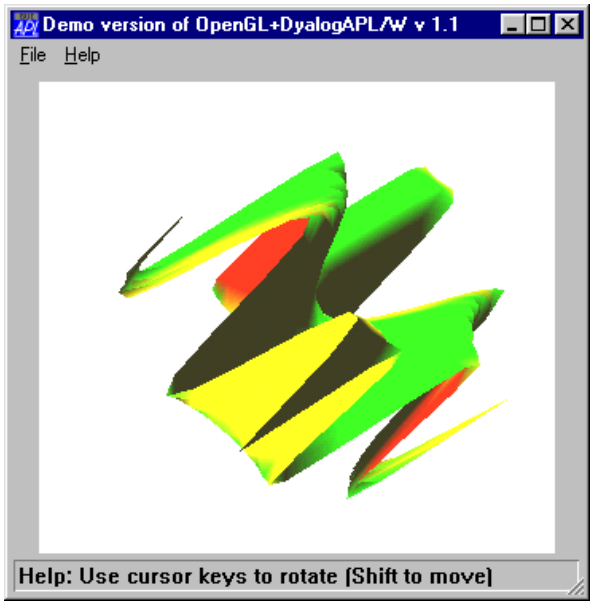
The OpenGL graphics library is an interesting application of ⎕NA. OpenGL is described in http://www.opengl.org/documentation/spec.html?1.1/glspec.html

Alexander (Sasha) Skomorokhov, Alexei Zalivin and Alexander Kornilovski have kindly provided code that covers many of the OpenGL calls. Their workspace, DEMOGL.DWS, may be downloaded from the public download section of www.dyalog.com. It contains a number of static and dynamic examples.



Alexander Kornilovski has supplied a further workspace, GLAUX.DWS; also downloadable and with some more complex examples.





9.3.2.1 Explore these workspaces, with particular attention to the arguments of the 200 or so ⎕NA calls.

§§ 9.3.3 Linear Algebra Package

There are many other freely available sources of code that can be tapped into via ⎕NA. Some are single functions and some are large and complex application, such as PetSc, the Portable, Extensible Toolkit for Scientific Computation, downloadable from http://www-unix.mcs.anl.gov/petsc/petsc-2/ .

As a final example we consider LAPACK, the Linear Algebra Package downloadable free from http://www.netlib.org/lapack/ (under GNU license agreement).

LAPACK contains hundreds of C functions for real and complex matrix manipulation. Most of these functions are defined in order to support 2 major goals. The main one is general computation of eigenvectors and eigenvalues from real or complex square matrices. The other is the equivalent of monadic and dyadic domino (⌹) for real *and complex* matrices.

A subset of these functions has been chosen to cover the essentials of these two goals in order to provide the basis of two proposed new APL primitive functions ☯ (or %) and ⍁.

See *APL81 Proceedings* for first mention of symbol ⍁ for this purpose.

Monadic ☯ is *complex matrix inverse* and dyadic ☯ is *complex matrix divide*. These functions are modelled by the ambivalent APL function `Domino` to be found in supplied workspace MATH.DWS. The convention adopted is that complex numbers are enclosed 2 element vectors.

The primary purpose of LAPACK is the calculation of monadic ⍁ (*eigen*) - the computation of the eigenvectors and eigenvalues of real or complex square matrices. This function is implemented in the APL function `Eigen` to be found in MATH.DWS. Complex numbers are again represented as enclosed 2 element vectors.

Given some complex square i by i matrix, A_{ii}, an **eigenvector** of A_{ii} is a vector whose direction is unchanged by the application (matrix multiplication) of A_{ii}. The corresponding **eigenvalue** is the scaling factor, which may be complex. In other words, given

```
Eji←⍁ Aii
```

then

```
⊧(Aii+.x 1 0↓Eji)≡fuzz(i ipEji[1;])x¨1 0↓Eji
```

where `x` is complex multiplication

```
      ∇ R←A x W;Sign
[1]     Sign←2 2⍴1 ¯1 1 1
[2]     R←+/Sign×0 1⊖(2↑A)∘.×2↑W ∇
```

and `fuzz` is a fuzzy operator to cope with a little algorithm inexactitude.

```
      ∇ R←A(f fuzz)W;CT
[1]     ⎕CT←2*¯32 ⍝=2.328E¯10
[2]     R←(A+∘1)f W+∘1 ∇
```

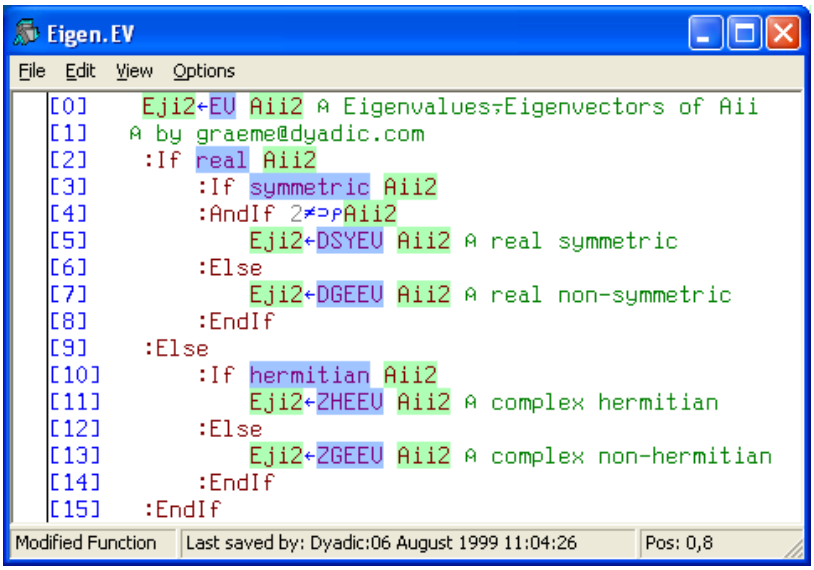
Of particular scientific interest are Hermitian matrices (H), defined by `⊧H≡¢⍉H` , where ¢ (or perhaps ∈) is defined as the complex conjugate function. Hermitian matrices are important because their eigenvalues are **real** numbers, as are the results of all quantitative measurements. Hermitian matrices are used to represent detailed measurements in science and engineering.

At this point we introduce a new breed of operator – a monistic niladic operator - which takes a matrix left operand and returns a related matrix. These operators are intended to generalise to matrices of functions as outlined in § 11.3.

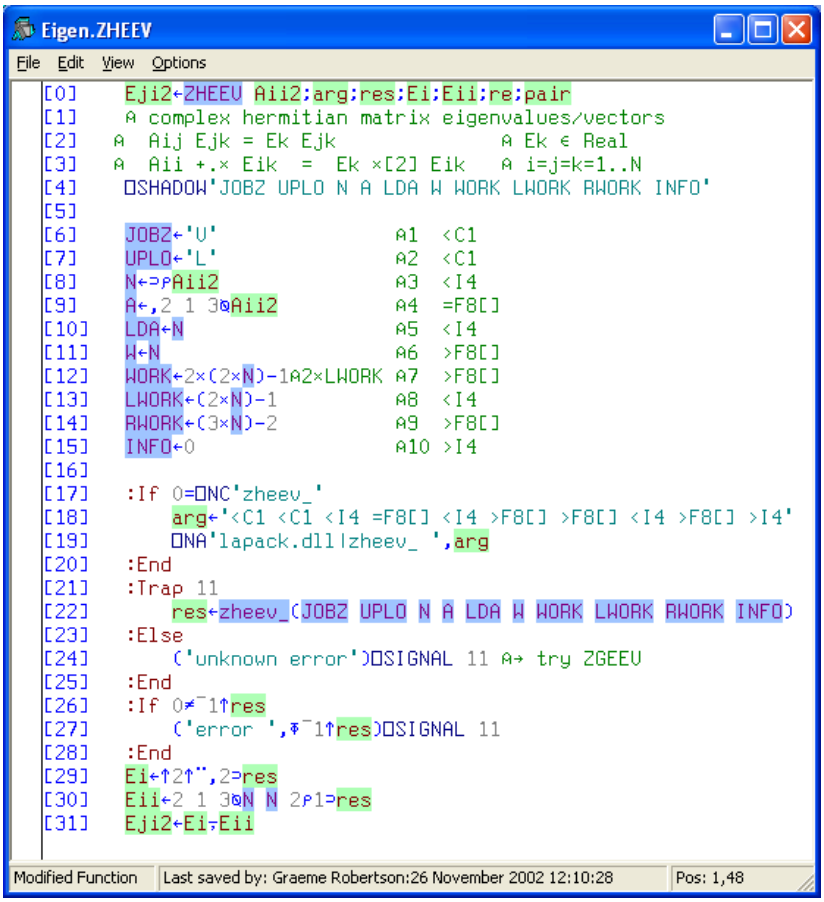
$Arr_2 \leftarrow Arr_1^{T}$	⍝ Transpose array Arr_1
$Arr_2 \leftarrow Arr_1^{-1}$	⍝ Inverse of array Arr_1
$Arr_2 \leftarrow Arr_1^{*}$	⍝ Complex conjugate of array Arr_1

$Arr_2 \leftarrow Arr_1^{\dagger}$	⍝ Complex transpose of Arr_1

The function ∇*EV*∇ below is a canonical version of the function ∇*Eigen*∇ in MATH.DWS. Line [11] calls function ∇*ZHEEV*∇.



∇*ZHEEV*∇ is a cover function for the LAPACK function *zheev_* which is fixed from the supplied LAPACK.DLL on line [19] of ∇*ZHEEV*∇ and called on line [22].



9.3.3.1 Compare the ⎕NA call in ∇ZHEEV[19]∇ with the snippet of C code in file ZHEEV.C below taken from the corresponding uncompiled LAPACK function zheev_.

```
ZHEEV.C
#include "f2c.h"

/* Subroutine */ int zheev_(char *jobz, char *uplo, integer *n, doublecomplex
        *a, integer *lda, doublereal *w, doublecomplex *work, integer *lwork,
        doublereal *rwork, integer *info)
{
/*  -- LAPACK driver routine (version 2.0) --
      Univ. of Tennessee, Univ. of California Berkeley, NAG Ltd.,
      Courant Institute, Argonne National Lab, and Rice University
      September 30, 1994
```

Specification of the right argument to ⎕NA can be arbitrarily complex. Errors in the specification can cause Dyalog APL to crash with a System Error 999 and therefore care should be taken when constructing ⎕NA calls.

Without a left argument ⎕NA fixes a function whose name is that of the C function involved. However, ⎕NA can take a left argument of a character string giving a different name to the function to be fixed in the workspace.

9.3.3.2 Ask for the next module on **runtime applications** ☺.