DSPExplorer BuiltIns

Arithmetic syntax follows the C hierarchy. No bitwise operations are available (no & or | or ~ for example).

Only two 'update' assignments are available: += and -=

VARIABLES: (a reminder)

Variables are 'double' or 'complex'. They must be declared before they are used. In addition to internal variables there are 'parameter's and 'slider's. Parameters and sliders are always 'double's.

TRIG FUNCTIONS:

all take a single arg, real or complex and all three return complex. Basic functions are sinh(x), cosh(x), sin(x), cos(x), tan(x), asin(x), acos(x), atan(x), and ejx(x). Three additional simplifying functions are ejx2p(x) which is the same as ejx(2*Pi*x), sin2p(x) and cos2p(x).

DEALING WITH COMPLEX NUMBES:

- complex(r,i) construct a complex number.
- real(c) take real part of complex number
- imag(c) take imaginary part of complex number
- conj(c) conjugate a complex number: "complex(real(c), -imag(c))"

DEALING WITH ARRAYS:

- Arrays can be assigned in whole or in part: a[3:2] = b[1:0] works as does a=b where a and b are arrays.
- Array indexes wrap. The last element in an array is a[1] for example. If the array is size n, then a[n] is the same as a[0].
- A constant array can be constructed using curly braces. For example: double array[] = {1,2,3,4,5,6}; will declare and initialize an array. The constant array must consist of compile time constants. Complex elements can be constructed using 'complex()' e.g. {complex(1,2),complex(3,4)}. One can also say "double a[4]; a={1,2,3,4}.
- When array sizes are silently made the same. If the array is being made larger, zeros are used. If smaller, the upper elements are silently dropped.

PRINTING:

Both print statements send output to the 'console' window. There are two print statements, 'print' and 'println'. 'print' takes any number of args. Each are is either a string or a number. Strings are printed exactly with the usual escapes available (like "\n"). "println(<args>)" is literally the same as "printl(<args>,"\n")". print and println are statements.

When numbers are printed, their precision is truncated to make printouts easier to read. Things which are small are displayed as "~0" for 'about zero'. Complex numbers are surrounded by {} e.g. {1,2} is a complex number whose real part is 1 and imaginary part 2. This is NOT the same as the syntax for making a complex number. Sorry.

When printing an array, the 0th element is printed first, followed by 1st, etc.

Arrays are printed in their entirety. Curly braces are used to show that an array is being printed. Yes, {1,2} could be a complex number or an array of two elements. Again, sorry.

FOURIER AND RELATED TRANSFORMS:

- fft(a) where a is an array. If necessary, a is expanded to the next power of two in size. The fft(a) is an expression, thus you need to write "b=fft(a);". The fft does a divide by N at the end rather than having the ifft do the divide. This is so the magnitude of the FFT is independent of the length of the FFT.
- ifft(a) where a is an array.
- magphz(a) magnitude and phase of a. Computes the magnitude and phase of an array of complex numbers. The result is a new array where the real parts are the magnitude and the imaginary part is the angle. The angle is normalized to the range +-Pi.

MISCELLANEOUS ARITHMETIC

- sqrt(x) square root of x where x can be complex. Returns complex. Thus, sqrt(-1) returns {0,1}.
- log(x) natural logarithm of x where x can be complex. Returns complex.
- exp(x) "e to the x" where x can be complex. Returns complex.
- rand() generate a random number where the real and imaginary parts are independently, evenly distributed between 0<=r<1. Same as complex(rand(),rand());
- abs(x) where x may be complex. Returns a complex number where the real part is the magnitude of x and the imaginary part is the angle normalized to +-Pi. (Essentially the same as 'magphz(c)' where c isn't an array.)
- int(x) take the integer part of the real part of x. Returns double.
- changed(arg) true if the arg has changed since the last execution. Changed does not take arrays as of yet.

MIMISCELLANEOUS EXECUTION

These are all 'statements' not 'expressions'…

- sleep(n) sleep n milliseconds. Simulation stops for n milliseconds.
- halt() stop simulation immediately.
- trigger(n) trigger the scope where the trigger will be in the middle of the screen OFFSET by n.
- for(<init>;<while>;<iter>) just like the C 'for' loop.
- while(<while>) just like the C 'white' loop.
- once execute the following statement once. Useful as an 'initialize' construct.

- if (b) <statement> [else <statement2>] if the expression b is non-zero then execute statement1. If b is zero then execute statement2 if present. (Else clause is optional).
- { <statement list> } compound statement. May be empty.
- loadArrayValues(array,"filename");   reads the file and loads the values into the given array. If the given array is double, one number per array element is expected. Otherwise, two numbers (real and imaginary part) are expected per array element. Will write a message to the 'log' if the sizes don't match. Whitespace is ignored, numbers are comma separated.
- rotate(array,n) rotate the array n places upward…IN PLACE. 'array' is both input and output. Thus, with n == 1, {1,2,3,4,5} would be rotated to {5,1,2,3,4}.

AUDIO IO

DSPExplorer allows you to read, write, record, and play audio files in both real time and in batch mode.   At present, the files are all PCM 16 'WAV' files. Mono and stereo channels are supported at any sample rate.

Opening a file (or other data stream) returns a double called a 'handle' which is much like a 'file descriptor' or 'stream number'. It is <= 0 on failure.

double readHandle = audioReadFile("filename");
    opens a file for reading.
double writeHandle = audioWriteFIle(format[,filename]);
    opens a file for wring. If the filename is not give, the 'speakers' are opened.
double readHandle = audioReadMixer([format,]mixerName);
    Opens a stream from the system 'mixer'. The mixer name is system dependent. DSPExplorer checks the mixer name at compile time and make suggestions if the supplied one is not available. The default format is 44100 samples/second, stereo, 16 bits. Try "Built-in Input".

Successful opens result in a handle > 0. To get the format of the stream you can:

complex format = audioGetFormat(handle);
    Formats are in the form:
        complex(speed,bitsPerSample*10+numChannels). So, for example, 44100 sps, 16 bit, stereo is complex(44100,162);
    You can provide just the speed and the 162 is assumed.

Once you have handles you can read and write data using:

complex sample = audioReadData(handle);
    monodata is still returned with the imaginary part 0. All data is converted to double between -1 and 1. Data outside this range

can be interpreted as “end of file”.

audioWriteData(sample) to write data. Data outside the -1 to 1 range is blindly used.

audioClose(handle) to close a specific file or
audioClose(-1) to close all files. Closing a closed file is ignored. Closing the ‘0’ handle is also ignored.

NOTE: Wav files are not ‘valid’ until they are closed. (This is an artifact of the file format outside my control… sorry).