/\*

------------------------------------------------------------------------------

PROGRAM NAME: v2x\_rbo\_demo.cc

AUTHOR: H. Cho, Pacific Northwest National Laboratory

CREATION DATE: 14 September 2012

LAST MODIFICATION DATE:

DESCRIPTION: This program converts complex 1-D and 2-D time domain vnmr

files to text files.

This source file must be linked to the GAMMA object library via UNIX

command:

"/usr/local/bin/gamma v2x\_rbo\_demo.cc -o v2x\_rbo\_demo"

This program reverses the byte order of data read in from the vnmr file.

This operation is necessitated when the microprocessor of the spectrometer

computer uses the reverse byte of the destination computer.

------------------------------------------------------------------------------

\*/

#include "gamma.h"

#include "iomanip"

#include "stdlib.h"

#include "fstream"

#include "math.h"

#include "Cho\_1.h"

#include "VNMRobjects.h"

const int name\_length = 200;

const int max\_chars = 1048576;

const int max\_points = 524288;

//

int main( int argc, char\* argv[])

{

char

infile[name\_length], outfile[name\_length];

/\*

------------------------------------------------------------------------------

Interactively enter starting information at run time. The following

data are requested and stored in the variable names below:

- infile: input VNMR file

- outfile: output file where data are to be saved

------------------------------------------------------------------------------

\*/

cout << "\nvNMR filename: ";

cin.getline(infile, name\_length);

//

cout << "Output text filename: ";

cin.getline(outfile, name\_length);

/\*

------------------------------------------------------------------------------

Connect input stream to file containing vnmr data, and terminate program

if file not found.

------------------------------------------------------------------------------

\*/

ifstream indata(&infile[0]);

if(!indata)

{

cout << "\nError opening file. Program aborted.\n";

return 1;

}

/\*

------------------------------------------------------------------------------

Define some variable names.

------------------------------------------------------------------------------

\*/

datafilehead filehead;

datablockhead blockhead;

hypercmplxbhead hcblockhead;

int

I0, I1, I2, numpts, fileheadsize, blockheadsize, index,

number\_frequencies;

fileheadsize = (7\*sizeof(long)) + (2\*sizeof(short));

blockheadsize = (4\*sizeof(short)) + (4\*sizeof(float)) + sizeof(long);

char

first\_status\_char[2];

float

scale\_factor = 1.0,

complex\_signal\_float[max\_points],

dwell\_time\_f2 = 0.001,

sw2 = 1000.,

frequencies[16],

dwell\_time\_f1 = 0.001,

sw1 = 1000.;

short

complex\_signal\_short[max\_points];

long

complex\_signal\_long[max\_points];

/\*

------------------------------------------------------------------------------

Read vnmr fileheader, reverse the byte order of individual data, then open

the output file.

------------------------------------------------------------------------------

\*/

indata.read((char \*) &filehead, fileheadsize);

reverse\_byte\_order((char \*) &filehead.nblocks, sizeof(long));

reverse\_byte\_order((char \*) &filehead.ntraces, sizeof(long));

reverse\_byte\_order((char \*) &filehead.np, sizeof(long));

reverse\_byte\_order((char \*) &filehead.ebytes, sizeof(long));

reverse\_byte\_order((char \*) &filehead.tbytes, sizeof(long));

reverse\_byte\_order((char \*) &filehead.bbytes, sizeof(long));

reverse\_byte\_order((char \*) &filehead.status, sizeof(short));

reverse\_byte\_order((char \*) &filehead.nbheaders, sizeof(long));

//

numpts = filehead.np / 2;

cout << "\nFile header size (bytes) = " << fileheadsize

<< "\nNumber of blocks = " << filehead.nblocks

<< "\nNumber of traces per block = " << filehead.ntraces

<< "\nNumber of elements per trace = " << filehead.np

<< "\nNumber of points, direct dimension = " << numpts

<< "\nNumber of bytes per element = " << filehead.ebytes

<< "\nNumber of bytes per trace = " << filehead.tbytes

<< "\nNumber of block headers per block = "

<< filehead.nbheaders

<< "\nFileheader status word = " << filehead.status

<< endl;

memcpy((void \*) &first\_status\_char, (void \*) &filehead.status, 2);

if ((first\_status\_char[0] & 0x8) == 0x8)

{

cout << "\nData are stored as floating point." << endl;

}

/\*

------------------------------------------------------------------------------

Enter some metadata.

------------------------------------------------------------------------------

\*/

cout << "\nEnter number of frequencies: ";

cin >> number\_frequencies;

for(I0 = 0; I0 < number\_frequencies; I0++)

{

cout << "\nEnter frequency " << I0+1 << " (Hz): ";

cin >> frequencies[I0];

}

cout << "Enter direct dimension spectral width (in Hz): ";

cin >> sw2;

dwell\_time\_f2 = 1.0 / sw2;

/\*

------------------------------------------------------------------------------

Determine if the vnmr file stores a 1-D or 2-D experiment. If the file

holds a 2-D experiment, request data for the indirect dimension.

------------------------------------------------------------------------------

\*/

if ((filehead.nblocks \* filehead.ntraces) > 1)

{

cout << "Enter indirect dimension spectral width (in Hz): ";

cin >> sw1;

dwell\_time\_f1 = 1.0 / sw1;

}

/\*

------------------------------------------------------------------------------

Open output file and write metadata to the header.

------------------------------------------------------------------------------

\*/

ofstream outdata;

outdata.open(outfile, ios::out);

outdata << "Ag\_ENUF\_v1.0\n" << "Agilent\n";

for(I0 = 0; I0 < number\_frequencies; I0++)

{

outdata << scientific << frequencies[I0] << '\n';

}

/\*

------------------------------------------------------------------------------

Read the vnmr block header and data, and then write the data to the

output text file. This is repeated block-by-block.

Sometimes VNMR files have a second header block for hypercomplex data sets.

When this header is present, filehead.nbheaders is equal to two, and this

second header block must be read before the actual NMR data.

Check the filehead.status parameter to determine if the data are single

precision floating point or integer, and adjust the type conversion

appropriately.

Check the filehead.ebytes parameter for two-byte or four-byte integer

data, and convert input data to floating point.

VNMR data use a left handed coordinate system to define signal phases..

To correct for this, the imaginary part of the complex data points (i.e.,

every second word in the fid data stream) must be negated.

------------------------------------------------------------------------------

\*/

for(I1 = 0; I1 < filehead.nblocks; I1++)

{

indata.read((char \*) &blockhead, blockheadsize);

if (filehead.nbheaders > 1)

{

indata.read((char \*) &hcblockhead, blockheadsize);

}

reverse\_byte\_order((char \*) &blockhead.scale, sizeof(short));

scale\_factor = (float) pow(2, (double) blockhead.scale);

for(I0 = 0; I0 < filehead.ntraces; I0++)

{

if ((first\_status\_char[0] & 0x8) == 0x8)

{

indata.read((char \*) complex\_signal\_float, filehead.tbytes);

for(I2 = 0; I2 < filehead.np; I2++)

{

reverse\_byte\_order((char \*) &complex\_signal\_float[I2], sizeof(float));

complex\_signal\_float[I2] \*= (float) pow(-1., (double) I2);

}

}

else

{

if(filehead.ebytes == 2)

{

indata.read((char \*) complex\_signal\_short, filehead.tbytes);

for(I2 = 0; I2 < filehead.np; I2++)

{

reverse\_byte\_order((char \*) &complex\_signal\_short[I2], sizeof(short));

complex\_signal\_float[I2] = ((float) complex\_signal\_short[I2])

\* scale\_factor \* ((float) pow(-1., (double) I2));

}

}

else

{

indata.read((char \*) complex\_signal\_long, filehead.tbytes);

for(I2 = 0; I2 < filehead.np; I2++)

{

reverse\_byte\_order((char \*) &complex\_signal\_long[I2], sizeof(long));

complex\_signal\_float[I2] = ((float) complex\_signal\_long[I2])

\* scale\_factor \* ((float) pow(-1., (double) I2));

}

}

}

outdata << dec << filehead.np/2 << '\n' << scientific;

for(I2 = 0; I2 < filehead.np; I2 = I2 + 2)

{

index = I2;

outdata << complex\_signal\_float[I2] << ' ';

outdata << complex\_signal\_float[I2+1] << '\n';

}

}

}

/\*

------------------------------------------------------------------------------

Flush the output data and close the input and output files.

------------------------------------------------------------------------------

\*/

indata.close();

outdata.flush();

outdata.close();

}