Binary File Manipulation with Perl

- The purpose of this lecture is to discuss the difference between binary and ASCII files, and examine binary file manipulation with Perl
- Last week we looked at ASCII files
  - ASCII files contain data in 'human-readable' or character format
  - An ASCII file is divided into lines, where the end of each line is marked by a new-line character
  - Each byte (8-bit) value in an ASCII file is an ASCII value used to represent a character, digit, punctuation symbol or non-printable characters such as a new lines, carriage returns, form feeds, etc.

Pros/Cons of ASCII Files

- Pro: Human-readable – you can examine/modify file contents with an ordinary text editor
  - Errors/compatibility problems with files are easy to identify
- Con: Space inefficient
  - To store the value -103.98349 in an ASCII format takes 10 characters, or 10 bytes. To store this same value as a binary number in IEEE single-precision floating point format takes only 4 bytes (32 bits).

Binary Files

- Binary Files store items in binary form
  - ASCII File: -3490 stored as 5 bytes: 2Dh ('-'), 33h ('3'), 34h ('4'), 39h ('9'), 30h ('0')
  - Binary file: -3490 (F25Eh) stored as a 16 bit integer, little endian format is: 5E, F2
- When reading/writing values in a binary file, need to know:
  - Precision – how many bits do we use for this value? 8 bits? 16 bits? 32 bits? 64 bits?
  - Byte order – if a value has multiple bytes,do we arrange the bytes least significant to most significant byte (little endian) or most significant to least significant (big endian)

Viewing Binary Files

- To examine a binary file, need a special program that will dump the file contents in some sort of ASCII format
  - Most binary file viewers will dump the file in ASCII Hex (base 16) format with the ASCII representation of each byte given as well
- Under Unix, the 'od' (octal dump) utility can be used to display the contents of a binary file
  - "*od –t x1 –c filename*" will display bytes in hex format, and also give the ASCII equivalent of each byte

Sample Binary Files

- The zip archive for this lab contains some sample binary files in the following format:

byte 0: format number that determines file format
bytes 1,2: # of records (16-bit unsigned number , little endian)
bytes: 3 to end -- records

File formats:
format 0: byte data (1 record = 1 byte)
format 1: 16-bit unsigned integers, little endian (1 record = 2 bytes)
format 2: 16-bit unsigned integers, big endian (1 record = 2 bytes)
format 3: 32-bit signed integers, little endian (1 record = 4 bytes)
format 4: 32-bit signed integers, big endian (1 record = 4 bytes)

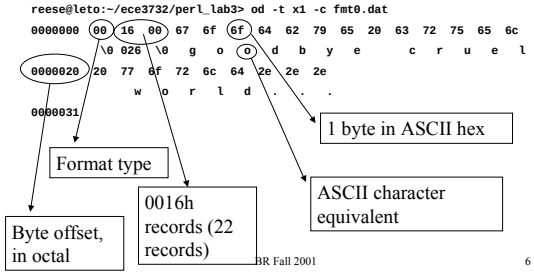
Note that total number of bytes in a file is 3 + record_size*# of records

fmt0.dat

'fmt0.dat' contains format '0' data, or just byte data.

If 'od' is used to dump the file we get:

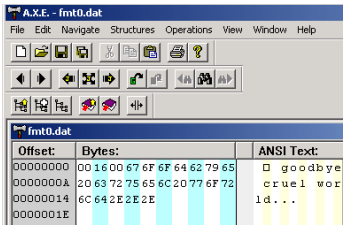
```
reese@leto:~/ece3732/perl_lab3> od -t x1 -c fmt0.dat
0000000  00  16  00  67  6f  6f  64  62  79  65  20  63  72  75  65  6c
          \0 026  \0   g   o   o   d   b   y   e       c   r   u   e   l
0000020  20  77  6f  72  6c  64  2e  2e  2e
           w   o   r   l   d   .   .   .
0000031
```



A.X.E. - Advanced Hex Editor for Win 32

The zip archive for this lab also contains a free Win32 binary file editor called A.X.E.

A.X.E. displaying the contents of *fmt0.dat* .

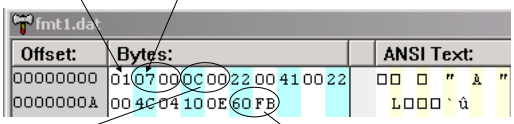


Contents of *fmt1.dat*

Recall that a file with format_type = 1 has 16-bit unsigned integers in little endian format:

Format type = 1

7 records (0007)



First record = 000Ch , which is 12

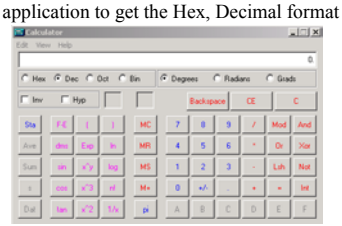
last record = FB60h , which is 64352

A Hex Calculator

Your PC has a nice caculator application under Win32 that can be used to convert between Hex and Decimal.

Usually under Start→Programs→Accessories

Use the View→Scientific menu choice on the Calculator application to get the Hex, Decimal format options.

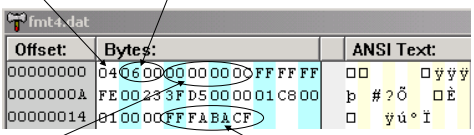


Contents of *fmt4.dat*

Recall that a file with format_type = 4 has 32-bit unsigned integers in big endian format:

Format type = 4

6 records (0006h)



First record = 0000000Ch , which is 12

Last record = FFFABACFh , which is 4294621903

Binary File Manipulation with Perl

- The pack/unpack functions are used to format/unformat binary records
- "pack" takes a template string which specifies the order and type of values to pack into a binary record. The template string is followed by a list of values to pack into the binary record.

Template string

The following are some characters that can be used in a template string (not all special characters are listed, see documentation):

character	*Meaning*
c	A signed char value
C	An unsigned char value
d	A double-precision float in the native format
f	A single-precision float in the native format
n	An unsigned 16-bit integer in big-endian order
N	An unsigned 32-bit integer in big-endian order
v	An unsigned 16-bit integer in little-endian order
V	An unsigned 32-bit integer in little-endian order

A number can follow a special format character to specify multiple values of the same type.

Create a format_type = 0 file

```
open(OUTPUT,">testfmt0.dat")
binmode(OUTPUT);
$fmt_type = 0;
$dstring = "Goodbye Cruel World!";
$num_records = length($dstring);
$temp = "Cva". length($dstring);
$buf = pack $temp,$fmt_type,$num_records,$dstring;
print OUTPUT $buf;
close (OUTPUT);
```

"binmode" used to place file handle into binary mode.

Template string ends up being "Cva20"

20 bytes of ASCII data

Unsigned char for format type

Unsigned 16-bit integer, little endian for # of records

Create a format_type = 2 file

```
open(OUTPUT,">testfmt2.dat")
binmode(OUTPUT);
$fmt_type = 2;
$num_records = 3;
$temp = "Cvn3";
$buf = pack $temp,$fmt_type,$num_records,345,24,10265;
print OUTPUT $buf;
close (OUTPUT);
```

"binmode" used to place file handle into binary mode.

Data for file

Template string is "Cvn3"

three 16-bit unsigned integers packed in big-endian order

Unsigned char for format type

Unsigned 16-bit integer, little endian for # of records

Reading a binary file

- Need to use *read* and *unpack* functions to process the contents of a binary file
- " *read (filehandle, scalar, length, offset)* " reads *length* bytes from *filehandle* and stores the result into *scalar* . The *offset*, if specified, says where to start putting the bytes into *scalar*.
  - Successive reads to a file pick up where the last read finished
- " *unpack (template, expr)* " unpacks the bytes in *expr* according to the specified *template* and returns a list that contains the unpacked data values.

Read a format_type = 2 File

```
open(INPUT,"testfmt2.dat")
binmode(INPUT);
read(INPUT,$fmt,1);
read(INPUT,$rlen,2);
$fmt_type = unpack("C",$fmt);
$num_records = unpack("v",$rlen);
if ($fmt_type == 2) {
    for ($i=0;$i<$num_records;$i++){
      read(INPUT,$buf,2);
      $d = unpack("n",$buf);
      printf ("%04x ($d)\n",$d);
    }
```

Unsigned char for format type

Unsigned 16-bit integer, little endian for # of records

16-bit unsigned integers packed in big-endian order

Print out the value in decimal and hex representations.

Standard Binary file types

- There are many standard binary file types defined for different applications
- Many types are defined for multi-media data (audio, graphics, complex documents, etc) because the large amount of data required means that ASCII storage of this data would be totally impractical
  - .bmp, .gif, .jpeg, etc are all graphic binary file types
  - .wav, .au, .mp3 etc are all audio binary file types
  - .doc, .pdf are document binary file types

.*wav* File Format

The file 'WAV_file_format.htm" in the ZIP archive for this lab defines the .*wav* file format.

The canonical WAVE format starts with the RIFF header:

Offset	Length	Contents
0	4 bytes	'RIFF'
4	4 bytes	<file length - 8>
8	4 bytes	'WAVE'

(The '8' in the second entry is the length of the first two entries. I.e., the second entry is the number of bytes that follow in the file.)

.*wav* File Format (cont)

Next, the **fmt** chunk describes the sample format:

Offset	Length	Contents
12	4 bytes	'fmt '
16	4 bytes	0x00000010 // Length of the **fmt** data (16 bytes)
20	2 bytes	0x0001 // Format tag: 1 = PCM
22	2 bytes	<channels> // Channels: 1 = mono, 2 = stereo
24	4 bytes	<sample rate> // Samples per second: e.g., 44100
28	4 bytes	<bytes/second> // **sample rate * block align**
32	2 bytes	<block align> // **channels * bits/sample** / 8
34	2 bytes	<bits/sample> // 8 or 16

.*wav* File Format (cont)

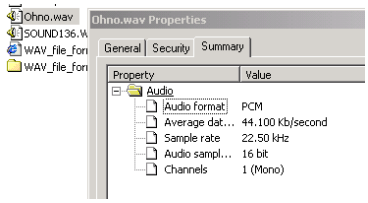
Finally, the **data** chunk contains the sample data:

Offset	Length	Contents
36	4 bytes	'data'
40	4 bytes	<length of the data block>
44	bytes	<sample data>

The sample data must end on an even byte boundary. All numeric data fields are in the Intel format of low-high byte ordering. 8-bit samples are stored as unsigned bytes, ranging from 0 to 255. 16-bit samples are stored as 2's-complement signed integers, ranging from -32768 to 32767.

Examining .*wav* Files

You can examine the properties of a .*wav* file in Explorer by selecting the file with a left-click, then right-clicking to bring up the file menu – select 'Properties' to bring up the properties window for this file. Left-clicking on the 'summary' tab will display the formatting details of the particular .*wav* you have selected.



Summary

- Perl can process binary files as well as ASCII files
- The pack/unpack functions are used for binary file manipulation
- Binary files are used by many applications instead of ASCII files to achieve more efficient data storage.