Dump a Binary File (HexEdit)

Kaminski / cs3310

A Binary File

A file is referred to as a "*binary file*" when it is not a text file. That is, the file's data is not a file of just ASCII (or Unicode) characters, ready to be printed to a screen or printer or viewed in NotePad. A binary file is to be read by a program, not by a human reader. Examples include executable (machine-language) program files, image/video/audio data files and traditional record-based data files. Such a file:

- contains some fields which are int/double/bool/. . . and/or other non-char, non-string data types
- does not have a <CR><LF> after each record (which is just a nice-ity for human readers)
- can use either fixed-length or variable-length records, but the program must know before doing a read or write:
  - the number of bytes to read/write if a whole record is read/written at once (e.g.,using C#'s FileStream)
  - or else the individual fields' data types (so their numbers of bytes can be determined) if individual fields are read/write independently
- can be a random access file or a sequential access file – that is a completely independent choice by the designer.

Why Use a Binary File

Why would a designer choose to create a binary data file rather than a text data file? Generally for reasons of time – i.e., numeric (and other) fields can be used "as is" without the time spent for conversion to/from char data. And sometimes for reasons of space – e.g., a 9-char social security number stored as a 9-byte charArray or a 10-byte string (StringLength plus the string) or a 4-byte int.

To View the Contents of a Binary File

In order to view a binary file, since it's not viewable with a text editor like NotePad, one can either:

- write a print program to display the contents of the file, which translates int's, floats, . . . into nicely formatted numbers
- or use a HexEditor to view the contents, generally in both hex (hexadecimal) and char-interpretation versions (and some software also provides an octal version as well)
  - the HEX version is most easily interpretable as the binary (bit) view of the bytes since 1 hex digit (0 to F) specifies 4 bits (0000 to 1111) – this is particularly useful for seeing what int's are in a file, e.g.,
    - `00 00 04 2F` in hex is
    - `00000000 00000000 00000100 00101111` in binary, which is
    - `    0     +    0     +   1024 + 32+8+4+2+1   = 1071` in decimal or
    - `(4 * 16^2) + (2 * 16^1) + (15 * 16^0) = 1024 + 32 + 15 = 1071` in decimal (if interpretting directly from hex)
  - the CHAR version is really only useful for viewing char or string field values. For numeric fields, the char-interpretter just translates every group of 8 bits into its ASCII equivalent, so for an int field (32 bits) of:

    `01000001 01000010 01000011 01000100` (i.e., hex: `41 42 43 44`)

    the char interpretation is `A B C D`

    Or more commonly, one would likely see a combination of symbols, letters, digits, extended ASCII characters, for example:

    `# @ ~ ? θ β π Σ` which is hex `23 40 7E 3F E9 E1 E3 E4`

    which could be a long (8 bytes) or a double (8 bytes) or 2 int's (2 * 4 bytes) or . . .

The Endian Issue

The above examples use "Big Endian". The C# example programs on the course website which use binary files use "Little Endian" since they're run on a PC which stores numeric fields that way internally, so that's how memory contents of an int/float/. . . would be written to a file (without any special handling). Read about this storage issue for numeric data in the Endian pdf document on the course website.

Binary Files in C# on a PC

There are several types of binary files that could be created/used in C# (each type are shown on the course website):

1. a PC-specific binary file written/read with methods in C#'s BinaryWriter and BinaryReader where numeric data is little endian and strings are written with preceeding length fields
2. a .NET-specific serialized binary file with significant additional meta-data (i.e., data about "the actual data") for a self-describing data file

3. a generic binary file which can be tailored to be readable using any language/OS/platform (i.e., big or little endian, strings with preceeding length fields or null-terminated string fields).

The easiest to use and the most commonly used in C# is type1. There are several classes for handling this type of binary file:

- BinaryWriter has a Write method (not WriteLine) to be able to write out fields of any of the different data types (since the system can determine the data type of the field specified as the parameterk)
- BinaryReader has separate methods for ReadInt32, ReadDouble, ReadSingle, . . . ReadChar, ReadChars(how many?) but no have a ReadLine, of course. Separate methods are needed since raw bytes on a file can be interpretted as a data value of any of the different data types.

Binary Files in C on a Sun Machine

A C program would define a record as a struct which contains charArrays, int's, float's, etc. On some systems a "wb" or "rb" parameter must be specified when opening the file. The fread and fwrite commands need parameters specifying the record to be read/written, the number of bytes and the file handle. For example:

```
struct out_rec_str
{   char  name  [10];                     /* ASCII char array */
    char  initial;                        /* ASCII char array */
    char  id     [2];                     /* ASCII char array */
    char  major  [3];                     /* ASCII char array */
    int   age;                            /* non-ASCII 4-byte integer */
    float gpa;                            /* non-ASCII 4-byte float */
    }     theRec;            /* THE RECORD OF THE ABOVE struct DATA TYPE */

theFile = fopen ( "binFile.bin", "wb" );           /* "wb" for WriteBinary */
theFile = fopen ( "binFile.bin", "rb" );           /* "rb" for ReadBinary */

fwrite( &theRec, sizeof(theRec), 1, theFile );
fread ( &theRec, sizeof(theRec), 1, theFile );
```

Dumping a Binary File on a Sun Machine

The utility for dumping a file is `od` ("octal dump"). The leftmost column is the octal byte-counter (byte number of the first byte in that row) which is not actually IN the data file. The default byte display is octal, but there are `-c` and `-x` options to instead provide character and hex interpretations. However, for a character version, if the byte is an unprintable character, the utility displays the octal interpretation of the byte – e.g., the 022 in the 2nd line is the octal equivalent for 00 010 010 in binary (or 12 in hex). [Note that Sun's use Big Endian storage for numeric data (the normal human way of representing numbers)].

```
SUN UNIX> cat input.txt
BushkowskiJ02CPU183.9
Black     S05CPS224.0
Jones     T08MTH253.0
Carr      J01ENG202.3

SUN UNIX> od -c binFile.bin
0000000   B   u   s   h   k   o   w   s   k   i   J   0   2   C   P   U
0000020  \0  \0  \0 022   @   y 231 232   B   l   a   c   k
0000040                   S   0   5   C   P   S  \0  \0  \0 026   @ 200  \0  \0
0000060   J   o   n   e   s                       T   0   8   M   T   H
0000100  \0  \0  \0 031   @   @  \0  \0   C   a   r   r
0000120                   J   0   1   E   N   G  \0  \0  \0 024   @ 023   3   3

SUN UNIX> od -x binFile.bin
0000000  4275 7368 6b6f 7773 6b69 4a30 3243 5055
0000020  0000 0012 4079 999a 426c 6163 6b20 2020
0000040  2020 5330 3543 5053 0000 0016 4080 0000
0000060  4a6f 6e65 7320 2020 2020 5430 384d 5448
0000100  0000 0019 4040 0000 4361 7272 2020 2020
0000120  2020 4a30 3145 4e47 0000 0014 4013 3333
```