CS 130 Lab #5: Binary file formats

Goals

Popular computer applications store user data in files comprised of binary values. Typically, the data is stored in these files so that broader descriptive data comes first, followed by more specific data whose form or layout depends on the earlier description: multiple "layers" of dependence arise naturally in this way.

For example, most files will have some sort of code or "tag" at the beginning which names or otherwise signals the overall file format. For graphic files, this tag is usually followed by information about the dimensions of the graphic: height and width in pixels and "depth" in bits. What order this information comes in, and how many bits or bytes are used, is specified in the file format signaled by the tag. Following the dimensional information, we might see "raw" pixel information (also called "raster data") and, in some cases, a color table. Note that the proper interpretation of the pixel information will depend on the dimensions and, if present, the color table.

In more complex file formats, these kinds of dependencies can make it quite difficult to "read" the file directly, as layers of context build up. In addition, most file formats use direct binary representations for the ultimate atomic data (e.g., color information). Most editor programs are configured to display text, assuming something like ASCII code is being used to represent the characters. In the case of so-called binary files (more accurately, files whose proper interpretation does *not* involve an ASCII-like character code), we might want the editor to display the data in its most user-accessible form, such as colors, numbers, etc. However, in many cases (such as today's lab) we want to specifcally focus on *how* binary representations are being used, so we want to view the data not as text, and not directly as colors, etc., but in terms of the "raw" binary codes. For practical purposes, this means that we will look at the codes in hexadecimal form, since it preserves the structure of the underlying binary, but makes for shorter and more easily distinguishable patterns.

In this lab we will look "inside" the binary details of a few different file formats to see how data of various kinds are represented. **You will need to copy all of the sample data files from the following folder onto your hard drive,** and also to locate and run the "hex editor" program we will use, called **`XVI32`**.

> cs130/docs

About the hex editor

In order to view files in their underlying binary format, we will use a so-called **hex editor**: this is a program which allows you to open and view a file *not* as it is normally displayed by the program in which it was created, but rather as a series of hexadecimal codes.

The hex editor we will use is called **`XVI32`**: it can be found on the lab computers in the "My Computer" section, "Local Disk (C:)", in the "Program Files" folder. (You may also download a copy of this free program for your own computer from the author's website, presuming that you use Windows (Mac users can find similar programs: ask me if you need help locating one).

Once you launch the `XVI` program, you will be able to open files and view and edit them directly in terms of the hexadecimal codes which comprise them: **you should always work on a *copy* of a file if you wish to keep the original**, since the changes you make at this level can corrupt the file so that it is not valid or recognizable by its "native" programs (Word, Excel, MS Paint, etc.). For lab purposes, of course, fresh copies of the files will always be available on the website.

Installing the software and working with files

Although one should not normally store personal files on the "C" drive of a lab computer, the software we are using seems to have trouble accessing network drives. So, I recommend the following approach to setting up the program and files:

1. find and open the **`My documents`** folder on your machine: it should be in the folder **`My Computer/Local Disk (C:)/Documents and Settings/WU_User/`**);
2. make a new folder in the "My Documents" folder (you might call it "CS 130");
3. when you get the sample files from the cs130/docs folder, save them into your CS 130 folder in My Documents;
4. you should now be able to use the editor to work on the sample files without too much trouble (but ask me if you are having problems);
5. **try to keep a file open with only one program at a time**: in other words, open a file with XVI32 and modify it, then close it and open it with MS Word, MS Paint or MS Excel (then close it again before re-opening it in XVI32); this will ensure that you are seeing (and saving) a current copy of the file at all times;
6. **be sure to copy your files back to your H drive** if you want to save them when you are done working;
7. **be sure to *remove* your files from the My Computer/CS 130 folder** so that other people don't "appropriate" your hard work.

Some notes on using the `XVI32` hex editor

The hex editor (XVI32) allows you to look at the contents of a file from several different perspectives simultaneously: it has two main panels, on the left and the right, each of which shows a different "view" of the file contents. The one on the left shows the bytes of the file in hexadecimal form, whereas the panel on the right attempts to display them as ASCII characters (of course, this may not work if they are full 8-bit values, i.e., with a leading "1" bit). **Note that the address or index number of the currently-selected byte is shown in a small field on the lower-left side of the left panel, labeled as `"Hex addr."`:** this field will be important for you when completing some of the exercises below.

On the hex side, each byte in the file (8 bits) is represented by two successive hex digits (e.g., `1A` or `23`. *Remember that even if the digits look like decimal, they are probably in hex format,* so that a value of `23` is really a decimal value of (2 x 16) + (3 x 1) = 35.

Text file formats

The first few fies we will look at are text files of various kinds: in the docs folder mentioned above you will find these files; open them using both MS Word (use the "Open with ..." option on the right mouse button if necessary) and in the XVI hex editor.

- the file `word.doc` is a standard MS Word file with some sample text, including some special characters and some "picture/symbol" characters written using the WingDings font.
- the file `word.txt` is a standard ASCII text file: notice that the special characters don't show up properly inside MS Word, as there are no ASCII equivalents for these characters. Notice also that there are no specifications of different fonts, so the picture/symbol characters do not show up properly.
- the file `word.rtf` is a special format which uses ASCII formatted "tags" to indicate the presence of special characters and fonts, etc.
- the file `unicode.txt` is a file which attempts to record the special characters used in Unicode format (although they will not all come through properly).

Be able to answer these questions for your demo:

- There is some information stored in the `word.doc` file about the computer and folders where the file originated. Can you find it using the hex editor? (Many students are surprised to see that this information is stored inside a Word document.)
- Can you modify the `word.rtf` using the hex editor so that different words appear? So that a different font is used for the text?

Inside an Excel spreadsheet

The file `excel.dif` is an Excel spreadhseet: open it both in Excel (double-clicking should do) and in the hex editor. Be able to answer these questions for your demo:

- How are the numbers stored in the Excel file? In particular, are they stored as binary versions of the numbers? If not, what form is being used?
- Can you change the numbers in the hex editor (say, the number "20") so that they show up differently in MS Excel?

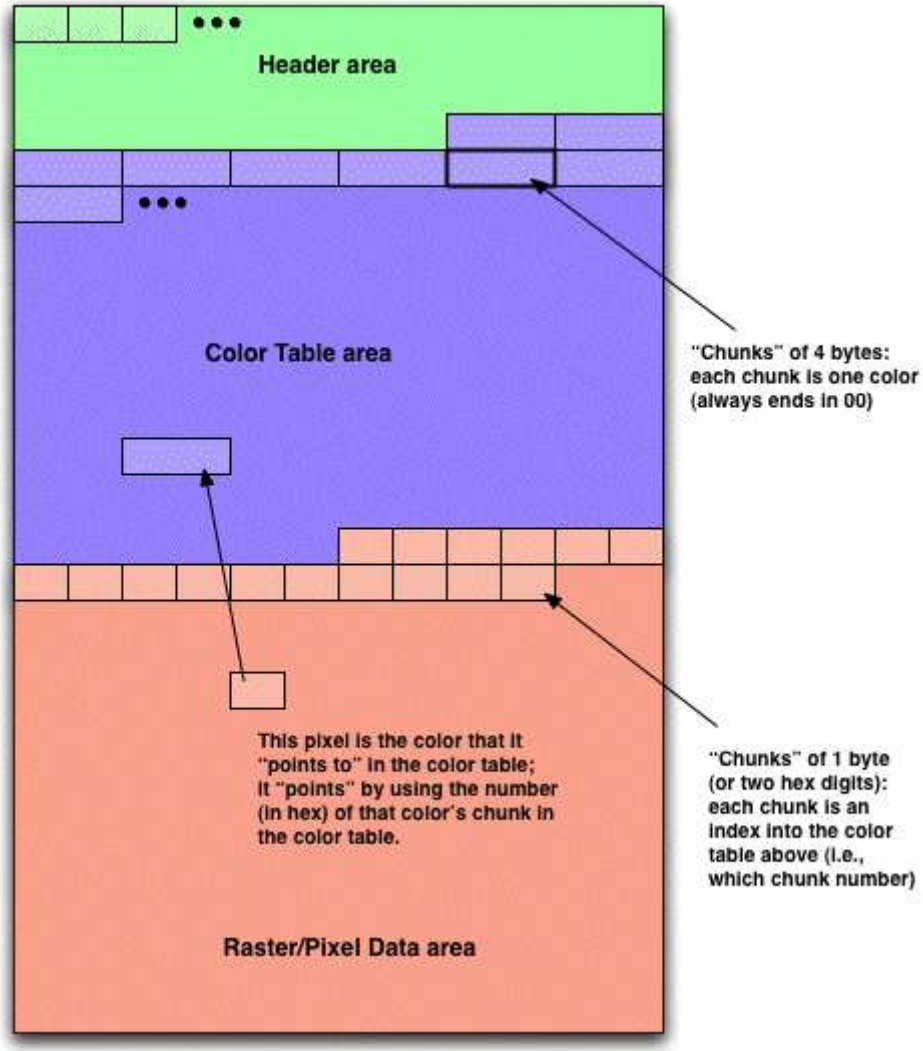
The BMP graphic file format

You can read about the BMP file format at this web page.

Read some of this page and see if you can make sense of it (this is an exercise in "technical reading", versus "technical writing"). Don't worry if you have trouble understanding it: the point of trying is to get a feel for the style of these kinds of documents, not to understand everything at first.

Now copy over the `helloSmall.bmp` file from the docs folder and open it with the MS Paint program (use "Open With ..." on the right mouse button if necessary).

You can read about the BMP file format and the layout of the data in the on-line reference, but this chart should help to visualize it:



(Note: the odd, mottled-looking color in some parts of this picture is due to limitations in the color model used in GIF files.)

The picture shows how the header, color table and pixel data areas are laid out. Note especially that the color table is broken up into 4-byte "chunks" (each byte being represented by two hex digits) and that the pixel data is broken up into 2-byte chunks, each of which is a numeric index into the color table. Finally, note that the numeric indices of the raster or pixel data are *not* addresses or locations of data within the file itself, but rather count *how many 4-byte chunks from the start of the color table* a certain color is.

How to calculate the position of a color in the color table

How can we determine the position within the *file* of a color in the color table, based on its index number as given in the pixel data?

We need to know two other things: first, the position within the file of the beginning of the color table and, second, the size of the "chunks" which are counted by the index used in the pixel data.

The position of the beginning of the color table can be determined either by reading the documentation about BMP files closely, or by looking for the right patterns in the hex values. It turns out that the first color in the table is white, represented by hex `FF FF FF 00` (the highest amount possible (`FF`) of each of red, green and blue plus the always-zero trailing byte.

From the picture and description above, we know that the colors in the table are broken up into 4-byte chunks. Therefore, if a color is at an index i, for example referred to in the raster/pixel data, we can determine its address or location a in the file itself as:

$$a = s + 4 \times i$$

where s is the start of the color table.

This information can also be used to determine the index from the address, using a little algebra as follows:

$$a = s + 4 \times i$$

$$a - s = 4 \times i$$

$$(a - s) / 4 = i$$

(here we have just subtracted from both sides and then divided both sides by 4).

Remember, however, that all these numbers are likely to be expressed in hexadecimal inside the file: you should therefore either convert from hex to decimal and back again, if you are calculating by hand, or do the arithmetic using a hexadecimal calculator (the WIndows utility calculator can be used in a hex mode).

Final exercises, using the graphic file

Look at the file as a graphic, then open it again using the hex editor. See if you can answer these questions for your demo:

- Can you find the color table? What locations/addresses in the file does it occupy (i.e., which numbered hex codes)?
- Can you find the raster data? What locations/addresses in the file does it occupy (i.e., which numbered hex codes)?
- Can you change a color in the graphic by modifying the raster data using the hex editor? (E.g., make the bottom row of pixels orange instead of blue.)
- Can you change a color in the graphic by *changing the color table?* (E.g., make every red pixel be green instead.)