Hints on Reverse Engineering a Data File

Richard Han
richard.han@mq.edu.au

Revised 26 July 2021

Introduction

Stages 4 and 5 of data file lab require you to reverse engineer an unfamiliar binary data file format. Your task is to convert the unfamiliar binary data into the well-known format that you have used earlier in this lab. You are provided with sample data files, and you have a lot of information about what data is in the sample files because we provide you with corresponding samples of the converted files.

Reverse engineering is a complex problem-solving task. It requires you to examine the available information, think of possible interpretations and check them to see whether they make sense. Above all, reverse engineering a data file requires you to understand how different data types (integers, characters, floating point, etc) are represented as bytes.

This document provides some helpful hints, specifically targeted to stage 4 of the data file lab. Much of the information in this document will also be useful for solving stage 5.

Getting started

Before you start looking at the binary data, it is worth knowing what you are looking for in the data. For each input file such as `input-1.bin`, there is a corresponding output file `output-1.bin` that is what your program should produce, and also a debug text file `debug-1.txt` that contains formatted output corresponding to the output binary data[1].

In stage 4, the fields of the unknown file format are in the same sequence as the fields of your familiar file format, and the records are in the same sequence[2]. Your task is to match up the fields and work out how to convert the data from the unknown format to the familiar format. You do this by matching up the data values in the fields, trying to find groups of bytes in the input records that you can interpret as numbers, characters or strings that have the same value as corresponding fields in the familiar format.

[1] Debug text files are not provided for stage 5. However, you can produce them yourself by running `output-1.bin` through your stage 2 program.

[2] In stage 5, the fields are in a different order within each record and the records are in a different order in the output file because the output file is sorted. These differences make stage 5 considerably more difficult than stage 4.

Changed data formats: The possible differences between the two file formats are described in the lab specification.
One important aspect to emphasize is that some Boolean fields in the well-known format may be packed into bits of a byte in the unknown format. You will need to work out which Booleans have been packed together and which bit position is used for each of those Booleans. In stage 4, the Booleans that are packed together will all be adjacent fields in the familiar format, and the packed byte will occupy a corresponding position in the unknown format, but the arrangement of the bits within the byte is arbitrary.

Here are some particular approaches that you can use to make a start.

1. Determine the size of each unknown input record in bytes. You have all the information you need in the sizes of the sample data files.
2. Start by looking at the smallest pair of data files. The less information is in the file, the easier it probably is to work out what some of the information is.
3. Dump the input file in hex and other forms. You can use the `od` command with various options for this purpose. There are also online tools that can display the contents of a file and you are free to use those if you wish.
    a. Examine your dumps. Look for values in bytes, shorts, 32 and 64 bit integers, floats, doubles and strings or chars that you can relate to the values that you expect to see in the input records that you have dumped. Some things will be much more recognisable than others. It is usually easiest to recognise text strings because they are quite distinctive in dumps. The `od` command has options to display a file as ASCII characters.
    b. Make careful notes about what you have discovered and check your ideas by examining other sample data files. Later on, you may need to check your thinking and good notes will remind you of what you did and help you to recognise mistakes.
    c. Once you find a consistent relationship then you've probably started to crack the unknown file format!
4. Once you have identified one or more fields of the unknown file format, you can adapt your stage 3 program to read the fields that you know about from the input files. Since stage 4 is not sorted, you should remove the sorting.
    a. If there are some fields of the unknown file format that you have not yet identified what they are, just read them and discard them. Put zeroes into the corresponding fields of the output file.
    b. For the fields that you have successfully identified, you need to read them as binary data in the format of the input files (the "unknown format" that you are now getting to know) and convert them to the data type that is used for the corresponding field in your well-known file format. Store them in your record structure for the well-known file format and write those records out.
5. Convert the sample input files using your new program and check whether you have correctly converted the fields that you have identified. You can check your converted output files by converting them to text[3] and comparing the text files[4], or by hex dumping your new output files as and comparing the bytes against the sample output files.

[3] Your program from stage 2 should be suitable for this conversion.
[4] The text files are actually in CSV format, so you can read them into Excel and view columns of data for comparison. However, the auto marker is sensitive to individual bit differences that may not show up in a text

6. You can submit your program using the lab command. The auto marker for this lab stage checks whether your program successfully converts each field and reports which fields are not properly converted. You will earn partial marks for converting some of the fields.

From this point on it is matter of improving your work. If you guess incorrectly about how big a particular field is in the unknown file then the results for that field may be correct on a small file but not on a big one which has more variety of data values. Whenever you encounter errors, carefully check the data in the files and revise your interpretation of the unknown file format. Remember, hex dumps are your friend! Also, you may want to manually decode or encode floating point or double precision numbers to find out what values have been stored in these formats.

Len Hamey 2020

---

version of your output file, so you should carefully check a hex dump to ensure that your conversion is exactly the same down to the individual bits in each byte.