Better Way to Extract Data from Structured Text Files

A Little Background

Once I was asked to write a program to extract trade data from text files produced daily by a legacy system. Before me, someone had tried to do it but couldn’t finish it.

The file contained repeating structure that started with 4 lines of account information followed by multiple lines of transactions and ended with transaction total for that account.

I checked that script and found out it tried to mimic the way human process such files. There were a lot of nested If statements. Even if it had worked, the maintenance would have been a nightmare. Tiny changes in the input file may need extensive software modification. After some research I created a better algorithm to tackle the problem of coding complexity and maintainability.

There are two parts in the algorithm. First part is to identify each input line correctly. Second, to picture how the lines are structured sequentially.

Identifying Every Input Line

I created one regular expression string and named it for each type of line input in file. In this case, I needed 4 regular expression strings, named account-1, account-2, account-3 and account-4, plus one for transactions and finally one for the total.

Draw the Structure of the Text File

Since text files are read sequentially, flowchart is the best tool to show the structure in a sequential way which is also the way the processing flows. It shows what is expected every time a line is read based on what has been read. Add the <Start> and <End> steps to show where to start and finish correspondingly. The following is the flow chart showing the structure of the file I worked on.

STARTT

account-1

END

total

account-2

account-3

account-4

transaction

Now onto Implementation

First, I put the input identification and the flowchart in a format that can be used by the programming language I am going to use. One function was created for each step in the flowchart, including Start and End. Together, the regular expressions, flowchart and the functions make up a setup that fully describes how my file should be processed.

Next, I created a base program that takes any given setup to process the type of files described by the set. The base program by itself has no knowledge of any file.

This is how the base program works.

The Pseudo-code

First, it runs the Start function to open the input file and prepare the output.

In a loop,

* Read one line and use the line identification to find out what type it is.
* If the type is what is expected according to the flowchart, run the corresponding function to extract data. If not, break out of the loop and declare process failure.
* If more lines to process, repeat the loop. Otherwise, exit the loop.

Finally, End function is run to properly close the process.

The Pseudo-code

Highest Maintainability

Since any points of failure can be located in the flowchart and the input file. cause of failure can be identified very quickly.

In my case, after a long time in production, the transaction line failed to be identified. It turned out the line format had been changed. I just modified the regular expression and the transaction function. The rest of the setup was not affected. In 20 minutes, problem was solved.

It Can Be Turned into a Platform That Processes All Text Files

The base program can actually be expanded to handle different text files by holding many input setups. File names can indicate which setup should be used for processing. This makes the base program the central platform for processing all text files.