Author: Jeremy Keys

Date: 9/12/17

Problem statement:

The problem asks us to convert a file of strings, each separated by a new line, into a valid CSV file format which follows the following rules:

"You need to follow standard CSV file rules:
"1) if a line has a comma, then the entire line needs to be double quoted

"2) if a line has a double quote, the double quote needs to be duplicated

"3) it the line starts with white space, or ends in white space, then the entire line needs to be quoted" (https://nmsu.instructure.com/courses/1092489/assignments/5478880)

Methodology

source-converter.py :

```
#definitely need regex

import re
import sys

#constants
DEBUG = True
END_RECORD = "$$$$$"
START_RECORD = "%%%%%"

def debug(logMsg):
    if DEBUG:
        print(logMsg)

#check the user submitted a file name
if len(sys.argv) != 2:
    print("Please enter a source file name")
    sys.exit()

#put the file name in a string for readability
file_name = sys.argv[1]

#just for a little bit of house keeping/debugging, we will keep a running
list of every line that's been modified
modified_lines = []

#https://stackoverflow.com/questions/3277503/how-do-i-read-a-file-line-by-
line-into-a-list
with open(file_name) as f:
    #create a list of lines, stripped of the newline
    content = [line.rstrip('\n') for line in f]
```

```
    debug("printing the unmodified content:")
    debug(content)

    #open the file which will be output; we don't want to append
    out_file = open("out.txt", "w")

    #check that the source file contains a valid header, and err if there
isn't
    start_record = content.pop(0)
    if(start_record != START_RECORD):
        print("This is not a valid source file.")
        sys.exit()

    #define some variables
    i = 0
    prevLineWasEndRecord = False
    result_line = ""

    #read each line, modify if necessary, and write to a .csv file
    for line in content:
        i += 1
        debug("line #" + str(i) + ": " + line)

        #copy the line to the reference which will hold the outputted line
        result_line = line

        ########################### do the line modification
###########################


        #then check for double quotes, so they can be duplicated
        result_line = re.sub(r'"', r'""', result_line);

        #first check for whitespace at start of line
        #then check for commas; only bother if ws wasn't detected (thus else
if)     `\s` matches any ws char
        if re.match(r'\s', line) or re.search(r'.*\s$', line):
#`$` matches the end
            #add quotes around the line in the CSV
            result_line = r'"' + result_line + r'"'
        elif (re.search(r',' , result_line)):
            #add quotes around the line in the CSV
            result_line = r'"' + result_line + r'"'

        debug("modified line #" + str(i) + ": " + result_line)

        ########################### handle edge cases (meta-lines, etc)
###########################


        #if an end record is found, we print a newline and continue to the
next line
        if (line == END_RECORD):
            debug("current line is an end record; remove the previously
stored line, and replace w/ line w/o comma at end")
            #pop the string we had just appended
            poppedString = modified_lines.pop()
```

```
            debug("the string that was just popped: " + poppedString)
            #an replace it with the same string, minus the comma separator
            modified_lines.append(poppedString.rstrip(','))
            #and append a new line
            modified_lines.append("\n")

            #we have just encountered an END record
            prevLineWasEndRecord = True
            continue

        #if another START record after the first is found, the file
        #is only valid if it is immediately preceded by an END record
        elif (line == START_RECORD):
            if( not prevLineWasEndRecord ):
                print("\n\nThis is not a valid input file. \n\n")
                sys.exit()

                debug("%%%%% (start record) line found")
                #only reason for start record is to verify the file is valid,
so just continue to next line
                continue
            #if the file is still valid, we reset the flag and continue to
the next line
            else:
                prevLineWasEndRecord = False

        #if neither record was found, reset the flag
        else:
            prevLineWasEndRecord = False
            # out_file.write(result_line + ",")
            result_line = result_line + ","
            modified_lines.append(result_line)

    #end for loop, go to next line

out_file.writelines(modified_lines)
f.close()

debug("printing the modified content:")
debug(modified_lines)

print("The CSV file has been successfully created.")
```

SourceToCSV.java :

```
import java.io.*;
import java.util.*;

public class SourceToCSV {

    private static final boolean DEBUG = true;
    private static final String START_RECORD = "%%%%%";
    private static final String END_RECORD = "$$$$$";

    public static void main(String []argv) {
```

```
        //error checking
        if(argv.length != 1) {
            System.out.println("Please enter a source file name.");
            // throw new FileNotFoundException();
            return;
        }

        /*** Declare some variables ***/

        ArrayList<String> content = new ArrayList<String>();
        ArrayList<String> modified_lines = new ArrayList<String>();
        String fileName = argv[0].trim(); //remove the ws
        String result_line;
        boolean prevLineWasEndRecord = false;

        if(DEBUG) { System.err.println("fileName: " + fileName); }

        /*** try to open the file and read the start record  ***/

        //https://stackoverflow.com/questions/5868369/how-to-read-a-large-
text-file-line-by-line-using-java
        try (BufferedReader br = new BufferedReader(new
FileReader(fileName))) {
            PrintWriter outputWriter = new PrintWriter("out-java.txt");
            //read the first record of the file, to make sure it matches what
we expect
            String start_rec = br.readLine();
            if (!start_rec.equals(START_RECORD)) {
                System.out.println("This is not a valid source file.");
                throw new FileNotFoundException();
            }

            /*** Read every line into a list ***/

            String inputLine;
            while ((inputLine = br.readLine()) != null) {
               content.add(inputLine);
            }

            if(DEBUG) { System.err.println("all input lines have been
read."); }

            /*** Process all the lines in the list and populate a new one w/
them ***/

            for (String new_line : content) {
                //delete any newline at end of new_line
                result_line = new_line.replaceAll("[\n]", "");

                System.out.println("result_line after replacing newline: " +
result_line);

                //Every double quote must be replaced with two double quotes
                result_line = result_line.replaceAll("\"", "\"\"");

                System.out.println("result_line after quoting double quotes:
" + result_line);
```

```
                //https://stackoverflow.com/questions/19276151/check-for-
leading-trailing-whitespaces

                //if there exists WS at either the end or beginning of the
line, or if the line contains a comma, then we must quote the enter line: `"`
+ line +  `"`
                if(result_line.contains(",") ||
Character.isWhitespace(result_line.charAt(0)) ||
Character.isWhitespace(result_line.charAt(result_line.length()-1))) {
                    result_line = "\"" + result_line + "\"";
                    System.out.println("result_line after WS or comma
detected: " + result_line);

                }

                System.out.println("\nresult_line after all modifications: "
+ result_line + "\n\n");


                //if we are looking at an end record, we need to remove the
modified line from the list and delete its ending comma; also, we must add a
new line to the CSV file, so we add a 'character string' "\n" to the lines.
                if(new_line.equals(END_RECORD)) {                       //get
it
                    String poppedString =
modified_lines.get(modified_lines.size() - 1);
                    modified_lines.remove(poppedString);
//remove it
                    poppedString = poppedString.replaceFirst(",$", ""); //fix
it
                    modified_lines.add(poppedString);               //replace
it
                    modified_lines.add("\n");                       //add a
newline
                    prevLineWasEndRecord = true;                        //set
the flag
                } else if (new_line.equals(START_RECORD)) {
                    //if we have another start record, the file is valid
(iff) it is immediately preceded by an end record
                    if( !prevLineWasEndRecord ) {
                        System.out.println("Another start record was found
that is not matched with a preceding end record!");
                        throw new FileNotFoundException();
                    }

                } else /* just another record*/ {
                    //we need to append a comma to every record in order to
make it a valid CSV line;
                    result_line = result_line + ",";

                    System.out.println("\nFinal result_line: " + result_line
+ "\n\n");
                    //the above logic for encountering an END_RECORD fixes
the final record in a START...END block by removing its appended comma
                    prevLineWasEndRecord = false;   //reset the flag
                    modified_lines.add(result_line);
```

```
                }
            } //end while, go to the next line or we're finished with the
file

            /*** now write the modified CSV-consistent lines to the output
file ***/

            for ( String finished_line : modified_lines ) {
                System.out.println(finished_line);
                System.out.println(finished_line.toCharArray());
                outputWriter.write(finished_line.toCharArray());
            }

            /*** finish up (can't happen in finally) ***/

            br.close();
            outputWriter.close();

        } catch ( Exception e ) {
            System.err.println(e.getMessage());
            if(DEBUG) e.printStackTrace();
            System.err.println("The file did not exist or was invalid.");

        } finally {
            System.out.println("The CSV file has been successfully
created.");
        }
    }
}
```

Images out debugging output and files compared to provided output file

The provided output file (with which to compare our own) contains an extra space in one of the strings. I believe this is an error in the provided output file, not in the logic of my programs.

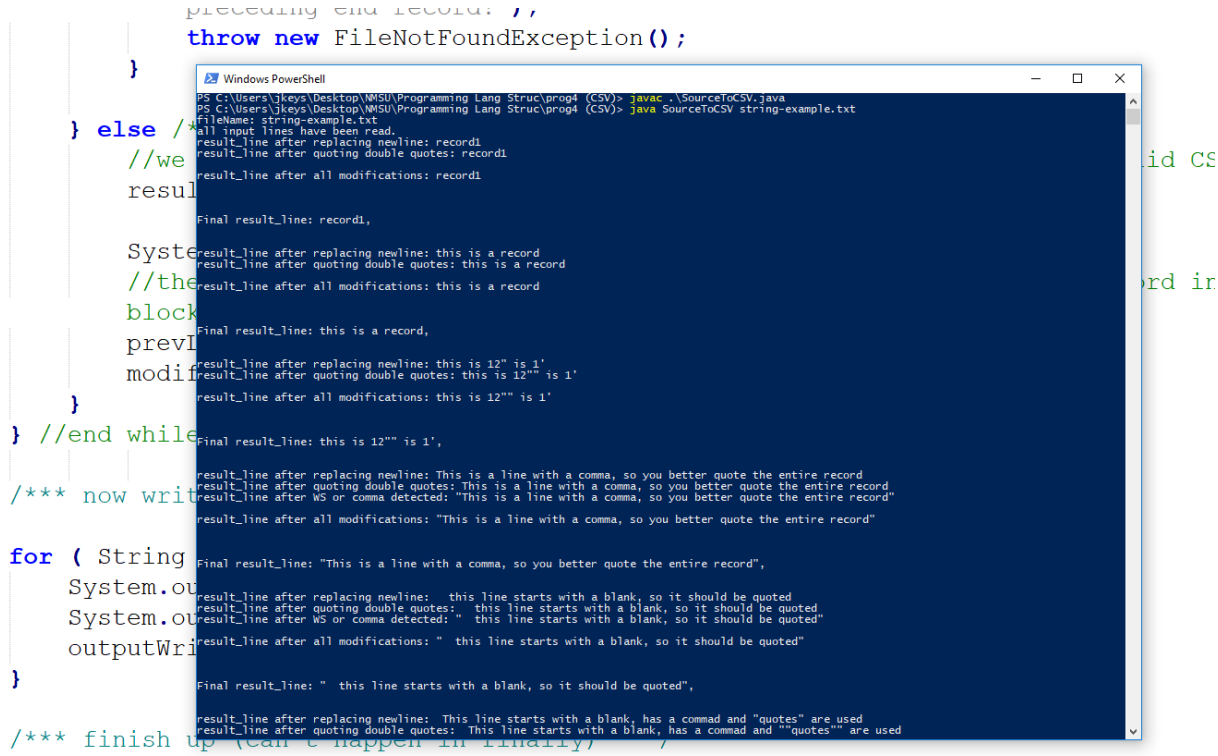

Figure 1 Sample output from the Java program

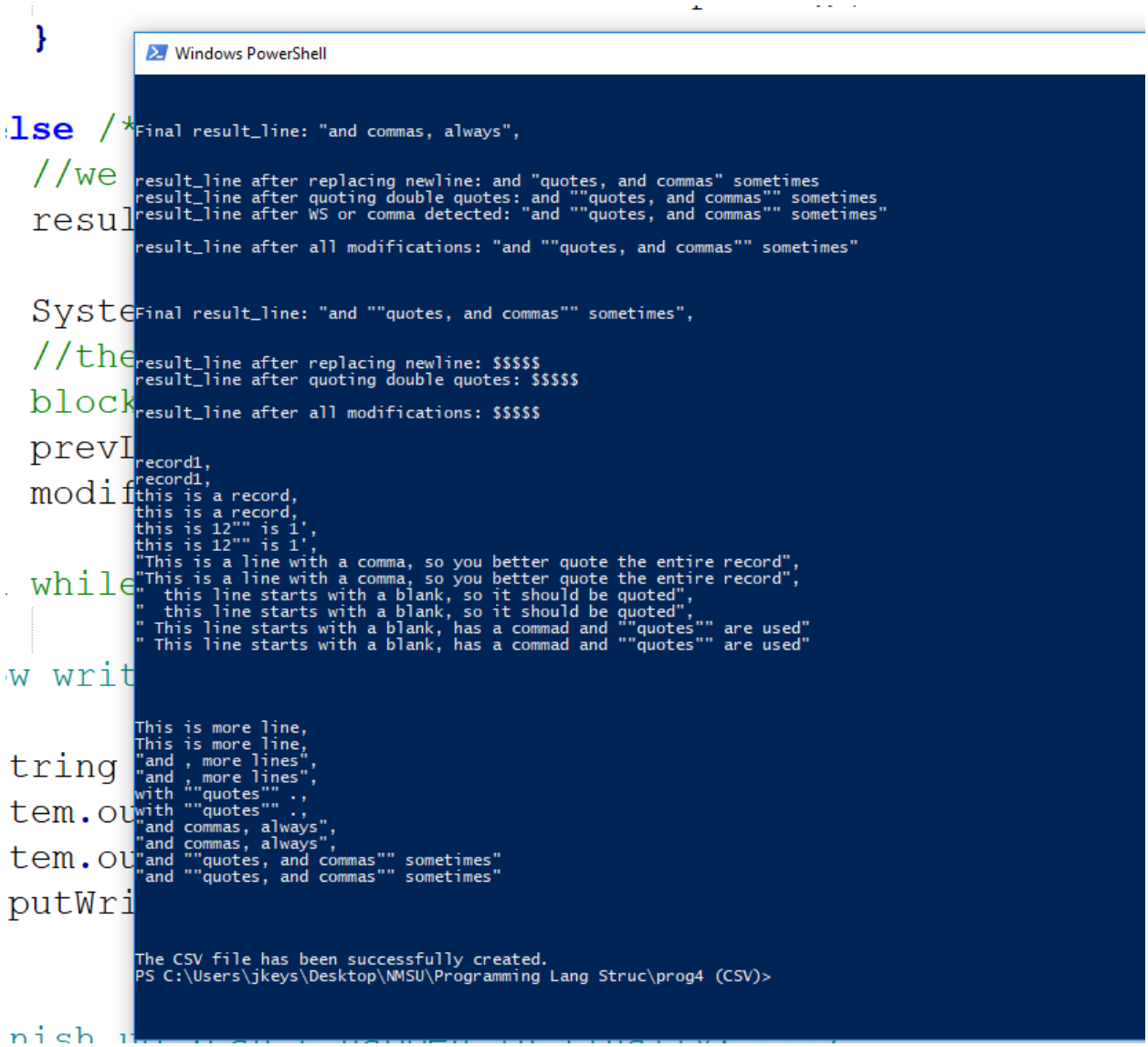

Figure 2 Final sample output of a run of the Java program, showing the modified "lines"

```
commas"" sometimes"
```

```
always","and ""quotes, and commas"" s
```

Windows PowerShell

```
The CSV file has been successfully created.
PS C:\Users\jkeys\Desktop\NMSU\Programming Lang Struc\prog4 (CSV)> python3.6m.exe .\source-converter.py .\string-example.txt
The CSV file has been successfully created.
PS C:\Users\jkeys\Desktop\NMSU\Programming Lang Struc\prog4 (CSV)> python3.6m.exe .\source-converter.py .\string-example.txt
printing the unmodified content:
['%%%%%', 'record1', 'this is a record', 'this is 12" is 1\'', 'This is a line with a comma, so you better quote the entire record', '  this li
ne starts with a blank, so it should be quoted', ' This line starts with a blank, has a commad and "quotes" are used', '$$$$$', '%%%%%', 'This
is more line', 'and , more lines', 'with "quotes" .', 'and commas, always', 'and "quotes, and commas" sometimes', '$$$$$']
line #1: record1
modified line #1: record1
line #2: this is a record
modified line #2: this is a record
line #3: this is 12" is 1'
modified line #3: this is 12"" is 1'
line #4: This is a line with a comma, so you better quote the entire record
modified line #4: "This is a line with a comma, so you better quote the entire record"
line #5:   this line starts with a blank, so it should be quoted
modified line #5: "  this line starts with a blank, so it should be quoted"
line #6:  This line starts with a blank, has a commad and "quotes" are used
modified line #6: " This line starts with a blank, has a commad and ""quotes"" are used"
line #7: $$$$$
modified line #7: $$$$$
current line is an end record; remove the previously stored line, and replace w/ line w/o comma at end
the string that was just popped: " This line starts with a blank, has a commad and ""quotes"" are used",
line #8: %%%%%
modified line #8: %%%%%
line #9: This is more line
modified line #9: This is more line
line #10: and , more lines
modified line #10: "and , more lines"
line #11: with "quotes" .
modified line #11: with ""quotes"" .
line #12: and commas, always
modified line #12: "and commas, always"
line #13: and "quotes, and commas" sometimes
modified line #13: "and ""quotes, and commas"" sometimes"
line #14: $$$$$
modified line #14: $$$$$
current line is an end record; remove the previously stored line, and replace w/ line w/o comma at end
the string that was just popped: "and ""quotes, and commas"" sometimes",
printing the modified content:
['record1,', 'this is a record,', 'this is 12"" is 1\',', '"This is a line with a comma, so you better quote the entire record",', '"  this lin
e starts with a blank, so it should be quoted",', '" This line starts with a blank, has a commad and ""quotes"" are used"', '\n', 'This is more
line,', '"and , more lines",', 'with ""quotes"" .,', '"and commas, always",', '"and ""quotes, and commas"" sometimes"', '\n']
The CSV file has been successfully created.
PS C:\Users\jkeys\Desktop\NMSU\Programming Lang Struc\prog4 (CSV)>
```

Figure 3 Sample debugging output from the Python program.

```
record1,this is a record,this is 12"" is 1","This is a line with a
comma, so you better quote the entire record"," this line starts
with a blank, so it should be quoted"," This line starts with a
blank, has a commad and ""quotes"" are used"
This is more line,"and , more lines",with ""quotes"" .,"and commas,
always","and ""quotes, and commas"" sometimes"
```

```
1 record1,this is a record,this is 12"" is 1","This is a line with a
  comma, so you better quote the entire record","  this line starts
  with a blank, so it should be quoted"," This line starts with a
  blank, has a commad and ""quotes"" are used"
2 This is more line,"and , more lines",with ""quotes"" .,"and commas,
  always","and ""quotes, and commas"" sometimes"
3
```

Figure 4 Given output (left) compared to source-convert.py output (right)

```
record1,this is a record,this is 12"" is 1","This is a line with a
comma, so you better quote the entire record"," this line starts
with a blank, so it should be quoted"," This line starts with a
blank, has a commad and ""quotes"" are used"
This is more line,"and , more lines",with ""quotes"" .,"and commas,
always","and ""quotes, and commas"" sometimes"
```

```
1 record1,this is a record,this is 12"" is 1","This is a line with a
  comma, so you better quote the entire record","  this line starts
  with a blank, so it should be quoted"," This line starts with a
  blank, has a commad and ""quotes"" are used"
2 This is more line,"and , more lines",with ""quotes"" .,"and commas,
  always","and ""quotes, and commas"" sometimes"
3
```

Figure 5 Given output (left) compared to SourceToCSV.java output (right)