Homework 1 - Extracting Data from a CSV file

Michael McAlpin
Instructor - COP3502 - CS-1
Summer 2017
EECS-UCF
michael.mcalpin@ucf.edu

June 1, 2017
Due on June 11, 2017 by 11:59 pm

Abstract

This assignment is based on a class of problem solved in enterprise computing; extraction, transformation, and loading. This is often referred to as ETL. The inputs will be data extracted from a leading aviation industry data and consulting firm, GCR. (See GCR.com for additional data.) The data is in a well known format where each data element is separated from the previous and following data elements by using a comma. It should be noted that this method of data manipulation is extremely common. The explicit order of the data fields and the desired outputs are defined in the "Specifications".

1 Objectives

The objectives of this assignment are to demonstrate proficiency in file I/O, data structures, and data transformation using C language resources. Specifically, you will read in data from a text file, use that data to populate a data structure, and print that data to *STDOUT* by accessing the newly populated structure.

1.1 Extraction

The first part of ETL is extraction. The filename of a text file will be passed to your program via the command line. The data contained in that file is to be read into memory (*i.e.*, extracted). Your program will be compiled and run on Eustis using the following commands:

```
gcc -o etl hw1etl.c
./etl inputFile
```

It is entirely possible that the input file either does not exist or is not where it is supposed to be. In such an event, your program should print an error message to *STDERR* that indicates which file is missing, then your program should **exit safely**. Use the following format for your error message (*fileName* should display the actual name of the missing file):

```
etl ERROR: File “fileName” not found.
```

The input file is in CSV (comma separated values) format where each line contains the data for one airport and the fields are as printed below. Note that these fields vary in size and content. Some fields may even be empty. Also note that the data for some of the fields are a *melange* of types. Specifically, the FAA Site Number and both latitude and longitude contain numbers, punctuation, and text.

For this assignment, treat all input data as character data.

Table 1: Airports Data Fields

Field Title	**Description**	**Size**
FAA Site Number	Contains leading digits followed by a decimal point and short text	Leading digits followed by a decimal point and zero to two digits and a letter
Loc ID	The airport's short name, i.e. MCO for Orlando	4 characters
Airport Name	The airport's full name, i.e. Orlando International	$\leq$ 50 characters
Associated City	The nearest city	$\leq$ 50 characters
State	State	2 characters
Region	FAA Region	3 characters
ADO	Airline Dispatch Office	3 characters
Use	Public or Private	2 characters
Latitude	DD-MM-ss.ssssDirection	Degrees, minutes, seconds. Direction is either N,S,E or W. Treated as a string (for now).
Longitude	See Latitude above.	ditto
Airport Ownership	Public or Private	2 characters
Part 139	FAA Regulation	No data
NPIAS Service Level	National Plan Integrated Airport Systems Descriptor	$\leq$ 50 characters
NPIAS Hub Type	Intentionally left blank	n/a
Airport Control Tower	Y/N	one character
Fuel	Fuel types available	up to 6 characters
Other Services	Collections of tag indicating INSTRuction, etc.	12 characters
Based Aircraft Total	Number of aircraft (may be blank)	Integer number
Total Operations	Takeoffs/Landings/etc (may be blank)	Integer number

1.2 Transformation

The second part of ETL is transformation. A list of comma separated values is convenient for text files, but it is far less convenient in memory. Once the data for a single airport has been read into a buffer, you will need to parse the buffer based on the commas between the data fields. The parsed data will then be used to populate a structure of the type `struct airPdata` (*i.e.* the data has been transformed from CSV to a data structure). The format of `airPdata`, shown below, will be defined in `airPdata.h`. Note that the `airPdata` structure uses the same names as the input file's *Field Names* (See Table 1 on page 2), though not all of the *Field Names* are used.

```
typedef struct airPdata{
  char *siteNumber; //FAA Site Number
  char *LocID;   //Airport's ``Short Name'', \textit{e.g.} MCO
  char *fieldName; //Airport Name
  char *city;    //Associated City
  char *state;   //State
  char *latitude; //Latitude
  char *longitude; //Longitude
  char controlTower;//Control Tower, this is a single character (Y/N)
} airPdata;
```

Remember, some of these fields will be of differing lengths for each airport. When you allocate memory structure's fields, you can assume that no entry will be longer than 50 characters (plus 1 character for the terminating NULL).

1.3 Loading

Finally, the third part of ETL is loading. With the data now in an `airPdata` structure it can be easily accessed by functions and/or other programs (*i.e.*, loaded). For this assignment, you will use pass the `airPdata` structure to a function (`PrintData(airPdata airport)`) that will print the data to *STDOUT* (aka the console). Before calling `PrintData` for the first time, make sure you print a header line that names each column. Specifically, use the following two lines of code:

```
printf("%-12s %-11s %-42s %-34s %-3s %-15s %-16s Tower\n",
   "FAA Site", "Short Name", "Airport Name", "City", "ST",
   "Latitude", "Longitude");

printf("%-12s %-11s %-42s %-34s %-3s %-15s %-16s =====\n",
   "========", "==========", "============", "====", "==",
   "========", "=========");
```

The "-" preceding each of the format specifiers left-justifies the printed values, while the numbers indicate the width of the printed field. Your data should be left-justified as well and should use widths that are identical to those in the header line. It is your choice whether you want to populate one `airPdata` structure and then print it, or to populate an array of `airPdata` structures and then print each of them. If you choose to populate and print one at a time, be sure to free any allocated memory before reallocating for the same variable. If you choose to read in all of the airports before printing them, you will have an easier time modifying your HW1 code when it comes time for HW3 and will only need to free the memory when you are done. Again, the choice is yours. An example of what the output should look like is shown on the next page.

FAA Site	Short Name	Airport Name	City	ST	Latitude	Longitude	Tower
03406.20*H	2FD7	AIR ORLANDO	ORLANDO	FL	28-26-08.0210N	081-28-23.2590W	N
03406.31*H	3FD5	ARNOLD PALMER HOSPITAL	ORLANDO	FL	28-31-21.0090N	081-22-49.2520W	N
03406.36*H	2FL5	BROOKSVILLE INTL AIRWAYS- INC	ORLANDO	FL	28-25-26.0000N	081-27-35.0000W	N
03406.24*H	FD99	DR P PHILLIPS HOSPITAL	ORLANDO	FL	28-25-43.0220N	081-28-38.2590W	N
03408.*A	ORL	EXECUTIVE	ORLANDO	FL	28-32-43.7000N	081-19-58.5000W	Y
03406.11*H	37FA	FLORIDA HOSPITAL	ORLANDO	FL	28-34-32.0020N	081-22-06.2490W	N
03406.22*H	FD36	FLORIDA HOSPITAL EAST ORLANDO	ORLANDO	FL	28-32-26.7000N	081-16-51.0000W	N
03406.40*H	FL76	HELI-PARTNERS I-DRIVE	ORLANDO	FL	27-23-04.0000N	081-29-07.0000W	N
03406.39*H	97FD	HELICOPTERS INTL	ORLANDO	FL	28-27-51.8300N	081-27-35.8800W	N
03407.2*A	ISM	KISSIMMEE GATEWAY	ORLANDO	FL	28-17-23.3000N	081-26-13.5000W	Y
03406.*C	91FL	LAKE CONWAY NORTH	ORLANDO	FL	28-28-45.0140N	081-22-03.2510W	N
03406.33*C	89FL	LAKE HIAWASSEE	ORLANDO	FL	28-31-45.0100N	081-28-51.2600W	N
03407.15*A	54FD	LM-ETS	ORLANDO	FL	28-22-03.0000N	081-04-34.0000W	N
03407.09*H	82FD	LOCKHEED MARTIN	ORLANDO	FL	28-26-48.4900N	081-27-03.6900W	N
03406.18*H	32FL	MEYER	ORLANDO	FL	28-30-05.0120N	081-26-39.2560W	N
03408.4*H	27FA	ORANGE COUNTY SHERIFF’S OFFICE	ORLANDO	FL	28-30-27.0110N	081-24-48.2540W	N
03407.*A	MCO	ORLANDO INTL	ORLANDO	FL	28-25-45.8000N	081-18-32.4000W	Y
03406.21*H	FD28	ORLANDO RGNL MEDICAL CENTER	ORLANDO	FL	28-31-31.0090N	081-22-37.2510W	N
03407.1*A	SFB	ORLANDO SANFORD INTL	ORLANDO	FL	28-46-37.1000N	081-14-05.7000W	Y
03406.29*H	7FA5	PREMIUM	ORLANDO	FL	28-23-21.0000N	081-29-19.0000W	N
03406.113*H	26FA	PRINCETON HOSPITAL	ORLANDO	FL	28-34-06.0040N	081-26-02.2550W	N
03406.14*A	01FA	RYBOLT RANCH	ORLANDO	FL	28-35-21.9970N	081-08-39.2290W	N
03406.38*C	12FL	TIMBERLACHEN	ORLANDO	FL	28-35-34.0000N	081-24-14.0000W	N
03406.34*H	0FL7	WKMG-TV	ORLANDO	FL	28-35-38.7000N	081-25-11.6000W	N
03406.3*H	13FD	YELVINGTON	ORLANDO	FL	28-31-07.0090N	081-22-59.2520W	N

2 Required Functions

void printData(airPdata airport);

Description: Prints the data for a given airport, using the same format as the provided header line.

Input: A pointer to an `airPdata` structure.

Special Cases: If a NULL pointer is passed to this function, print an error message to *STDERR* and return from the function without printing anything to *STDOUT*.

Returns: Nothing

3 Testing

There are several possible approaches for parsing the input data. Regardless of the approach you use, make sure to test your code on Eustis **even if it works perfectly on your machine**. If your code does not compile on Eustis you will receive a 0 for the assignment. There will be four (4) files provided for testing your code, they are as follows.

Table 2: Test Files

Filename	Description
twolines.csv	Two lines of test data, where one line consists of lower case letters, one unique letter per field, the other line will consist of uppercase letters.
orlando5.csv	Five lines of Orlando airport data.
orlando.csv	All 26 of the Orlando airports.
florida.csv	All 877 of Florida's airports.

The expected output for these test cases will also be provided. To compare your output to the expected output you will first need to redirect *STDOUT* to a text file. Run your code with the following command (substitute the actual name of the input CSV file):

```
./etl inputFile > output.txt
```

The run the following command (substitute the actual name of the expected output file):

```
diff output.txt expectedOutputFile
```

If there are any differences the relevant lines will be displayed (note that even a single extra space will cause a difference to be detected). If nothing is displayed, then congratulations the outputs match! For each of the four (4) test cases, your code will need to output to *STDOUT* text that is identical to the corresponding *expectedOutputFile*. If your code crashes for a particular test case, you will not get credit for that case.

4 Grading

Scoring will be based on the following rubric:

Table 3: Grading Rubric

Deduction	Description
-100	Code does not compile on *Eustis*
-100	Code does not accept the input filename from the command line
- 15	Code does not show an error message and/or does not exit safely when there is a file I/O problem
- 20	crashed on twolines.csv, or output does not match
- 20	crashed on orlando5.csv, or output does not match
- 20	crashed on orlando.csv, or output does not match
- 20	crashed on florida.csv, or output does not match
- 5	Missing the academic honesty affirmation (See Submission Instructions)

5 Submission Instructions

The assignment shall be submitted via *WebCourses*. There should only be one file in the submission.

- The main source file named `hw1etl.c` (The submitted file should be all lowercase, but to capitalized version is printed here to avoid any spelling errors from misreading the filename: HW1ETL.C. Again, your filename should be all lowercase.)

The header file `airPdata.h` should not be submitted. The graders will already have it. Do not rely on a modified copy of `airPdata.h`, and do not hard code the `airPdata` structure in your main source file. Doing so will cause your code to not compile.

Include a comment at the top of your main source file that contains the following statement (substitute your name and NID) - "I [name] ([NID]) affirm that this program is entirely my own work and that I have neither developed my code together with any another person, nor copied any code from any other person, nor permitted my code to be copied or otherwise used by any other person, nor have I copied, modified, or otherwise used program code that I have found in any external source, including but not limited to, online sources. I acknowledge that any violation of the above terms will be treated as academic dishonesty."