# Homework 4 - Data extraction, conversion, sorting, and build a CSV file output

Michael McAlpin
Instructor - COP3502 - CS-1
Fall 2016
EECS-UCF
michael.mcalpin@ucf.edu

November 23, 2016

**Abstract**

As discussed in *Homework 3* many ETL (extraction, transformation, and loading) problems parse data files wherein the data fields is separated by commas and data is converted from one format to another. This assignment is a continuation of that process - with an additional two steps. The first step is to sort the airports alphabetically. The second step is to sort the airports geographically and plan a route from south to north, all the while flying over land.

This assignment requires the data extraction, degree conversion, data formatting, sorting by two different criteria, and output the sorted results. The file inputs are defined in the *Inputs*. Similarly, the sort criteria are defined in *Processes* and the outputs are defined in *Outputs*.

## 1 Objectives

The objectives of this assignment are to demonstrate proficiency in file I/O, data structures, data transformation, sorting techniques, and file output using C language resources.

## 2 Inputs

There are two basic inputs, the input file name, passed via the command line, and the input file data sort order defined below.

### 2.1 Command Line arguments

The input file name will be input as follows:

- `hw4Sort filename.ext sortParameter`
- In the event that the input file is not available or there is an error finding the file, an appropriate error message shall be displayed. Use the example below for guidance.
- `hw4Sort ERROR: File "bogusFilename" not found.`
- In the event that the sortParameter is not available, an appropriate error message shall be displayed. Use the example below for guidance.
- `hw4Sort ERROR: sortParameter invalid or not found.`
- It would be appropriate to display the valid sort parameters in the error message. The *valid* sort parameters are **a** for *alphabetical sort* or **n** for *North Bound Exit*. The sort parameters can be entered in either upper or lower case.

## 2.2 Input File fields

The CSV input file contains the following fields. Please note these fields may vary in size, content, and validity of the data. Also note that some of the data formats are a *melange* of types. Specifically, note that both latitude and longitude contain numbers, punctuation, and text. Likewise, the FAA Site number contains digits, letters, and punctuation. (*This assignment will treat all input data as character data. Additional conversions are specified later in Processes.*)

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 | ~30 characters |
| Associated City | The nearest city | ~25 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.MASDirection | Degrees, minutes, seconds, milliarc-seconds followed by either N or S. |
| Longitude | DD-MM-SS.MASDirection | Degrees, minutes, seconds, milliarc-seconds followed by either E or W. |
| Airport Ownership | Public or Private | 2 characters |
| Part 139 | FAA Regulation | No data |
| NPIAS Service Level | National Plan Integrated Airport Systems Descriptor | ~10 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 |

# 3 Processes

The primary goal is to provide programmatic access to the data from the input CSV file. This must be accomplished using standard C file IO techniques. Also note that it is vital to utilize the *stuct airPdata* for all data retrieval/extraction and conversion. Likewise, use of the *stuct airPdata* is required for the file output.

## 3.1 Reading the input

There are several approaches to read the input. Perhaps the most important consideration is reading the line in for each airport. Please note that there is one line per airport. Also note, that once the line is read into the input buffer it might be advantageous to parse the input buffer based on the *comma* delimiter.

There are several approaches possible. Make sure to test on *Eustis* as line termination characters/behaviors vary amongst operating systems.

*Make sure that the output file is formatted with decimal degrees.*

## 3.2 Processing the data structure

The data conversions for this assignment, specified below, require a certain degree of parsing and calculation. Initially reading the input is to your advantage to deal with all data elements as *character data*. And then process the *latitude* and *longitude*, hereinafter referred to as *degrees*. The *degrees* are expressed as *sexagesimal (base 60)* numbers. Therefore use the functions created in *Homework 3* to establish *valid* latitudes and longitudes.

Please note that there are some airports whose *Loc ID* begin with **numerical digits**. There are also quite a few that contain two trailing digits. Typically these are helipads. For the purposes of this assignment those *airports* can be ignored or discarded from the input. *Careful review of these airports will reveal they typically start with the string FL or X and are followed by 2 digits.*

Once the input data has been appropriately extracted and converted, save it in memory in a *singly linked list*. This will facilitate further processing.

### 3.2.1 Latitude/Longitude Input

The *latitude* and *longitude* are both degrees, expressed as shown in the table below.

The conversion of the DDD-MM-SS.MASD string is shown in Table 2. The formula to convert a *sexagesimal* degree measurement to a digital degree measurement is shown below.

$$degrees^{decimal} = \pm DDD + MM/60 + SS.MAS/60^2$$

Table 2: Degrees

| Placeholder | Name | Value | Decimal |
|---|---|---|---|
| DD | Degrees | 180 | 0-180 |
| MM | Minutes | 0-59 | $\frac{value}{60}$ |
| SS.MAS | Seconds.MilliArcSeconds | 0-59.0-9999 | $\frac{value}{60^2}$ |
| D | Direction | N,S,E,W | See Table 3 |

Table 3: Direction

| Unit | Name | Decimal Sign |
|---|---|---|
| Latitude | N<br>S | +<br>- |
| Longitude | E<br>W | +<br>- |

Note that the $\pm$ is derived from the information in Table 3 above.

## 3.3 Functions

### 3.3.1 float sexag2decimal(char *degreeString);

**Description:** Convert the *sexagesimal* input string of *chars* to a **decimal** degree based on the formula in Tables 2 and 3.

**Special Cases:** If a NULL pointer is passed to this function, simply return **0.0**. Similarly, if the **DD-MM-SS.MASD** fields have invalid or out-of-range data, return **0.0**.

**Caveat:** Even though the *valid* range of Degrees is from 0 to 180, the data files for the Continental US and Florida are from 0 to 99. Make sure that the conversion can handle all valid cases correctly.

**Hint:** Take care to make sure the values for each numeric component are within their valid ranges. Refer to Table 2 for the ranges.

**Returns:** A floating point representation of the calculated *decimal degrees* or **0.0** in the special cases mentioned above.

### 3.3.2 void sortByLocID(lListAirPdata *airports);

**Description:** Sorts the airports *alphabetically* by the string named *Loc ID* with only **one airport per letter**. That is the first airport to begin with a *Loc ID* starting

with the letter **A** would be the *only* airport beginning with the letter A.
Increments the *seqNumber* as each airport is updated as a selected output.

**Special Cases:** Remember the helipads! In other words, it is acceptable to skip airports whose *Loc ID* begin with a number, or start with either **FL** or **X** followed by two digits.

**Caveat:** Since the sorting options are mutually exclusive, this function can destructively manipulate the input list to produce the desired results.

**Hint:** It might be faster to use the *Loc ID* as a hash value or perhaps as an index into an array, after all, there can only be one airport per letter of the alphabet.

**Returns:** Nothing. However the input data should be seriously modified by this process.

#### 3.3.3 void sortByLatitude(lListAirPdata *airports);

**Description:** Sorts the airports by *latitude* from **South** to *North*. Think of this as an *Escape from Key West to Georgia*.
Increments the *seqNumber* as each airport is updated as a selected output.

**Special Cases:** Remember the helipads! In other words, it is acceptable to skip airports whose *Loc ID* begin with a number, or start with either **FL** or **X** followed by two digits.

**Output:** Output the airports' data per the *output file specification* derived from walking thru the *AVL* tree until reaching the maximum latitude for the Florida border. *For the purposed of this exercise, assume 31 degrees North.*

**Caveat:** Since the sorting options are mutually exclusive, this function can destructively manipulate the input list to produce the desired results.

**Hint:** It might be faster to use the *Latitude* as a measurement criteria for building an *AVL* tree.

**Returns:** Nothing. However the input data should be seriously modified by this process.

# 4 Outputs

The outputs of the program will be populated `Struct airPdata` data. This data will be formatted so as to provide output as defined in the following sections.

## 4.1 Data Structures

### 4.1.1 struct airPdata

The structure `struct airPdata` is described below. Please note the correlation with the data file's *Field Names* refer to Table 1 on page 3 for more information.

```
typedef struct airPdata{
  int  seqNumber; //The process output's sequence number
  char *LocID;    //Airport's ``Short Name'', ie MCO
  char *fieldName; //Airport Name
  char *city;     //Associated City
  float longitude; //Longitude
  float latitude; //Latitude
} airPdata;
```

### 4.1.2 linked list of airPdata

```
typedef struct lListAirPdata{
  airPdata *curAirPdataP;    //Pointer to the Airport Data
  lListAirPdata *nextAirPdataList; // pointer to next
} lListAirPdata;
```

## 4.2 File output

The file output for this assignment is *stdout*, aka the console. Make sure there is a headline that names each column. For example:

```
seqNumber,code,name,city,lat,lon
1,DAB,DAYTONA BEACH INTL,DAYTONA BEACH,29.1797,-81.0581
2,FLL,FORT LAUDERDALE/HOLLYWOOD INTL,FORT LAUDERDALE,26.0717,-80.1494
3,GNV,GAINESVILLE RGNL,GAINESVILLE,29.6900,-82.2717
4,JAX,JACKSONVILLE INTL,JACKSONVILLE,30.4939,-81.6878
5,EYW,KEY WEST INTL,KEY WEST,24.5561,-81.7594
6,LAL,LAKELAND LINDER RGNL,LAKELAND,27.9889,-82.0183
7,MLB,MELBOURNE INTL,MELBOURNE,28.1025,-80.6450
8,MIA,MIAMI INTL,MIAMI,25.7953,-80.2900
9,APF,NAPLES MUNI,NAPLES,26.1522,-81.7756
10,SGJ,NORTHEAST FLORIDA RGNL,ST AUGUSTINE,29.9592,-81.3397
11,ECP,NORTHWEST FLORIDA BEACHES INTL,PANAMA CITY,30.3581,-85.7956
12,OCF,OCALA INTL-JIM TAYLOR FIELD,OCALA,29.1717,-82.2239
13,MCO,ORLANDO INTL,ORLANDO,28.4292,-81.3089
```

Things to note:

- The sample output above is an **example only**.
- Digital degrees are expressed as floating point numbers of varying digits of precision. This is an artifact of *Javascript*. In this exercise 4 digits to the right of the decimal point is sufficient.

- The first line of the file identifies the field names. This is a material fact and will adversely impact the output of the data in the webpage. *Capitalization and spelling matter - and must match the first line above.*
- The text shown above has been converted to uppercase as a piece of information to help debugging. String case conversion is not required for this exercise.

Once the output has been verified, redirect the *stdout* to a file named **myTestAirports.csv**. Make sure that all code, inputs, and outputs are built, tested, and the output file is exported from *Eustis*.

## 4.3 Testing

There is one input file provided for program testing. It is described below. The program's output will be to *stdout*. Redirect the output to the file named *myAirports.csv*.

Table 4: Test Files

| Filename | Description |
|---|---|
| FL-ALL-airports.csv | A list of the *875* Florida airports, wherein all the data is formatted as defined in the Input Specification. |

# 5 Grading

Scoring will be based on the following rubric:

Table 5: Grading Rubric

| Percentage | Description |
|---|---|
| -100 | Cannot compile on *Eustis* |
| - 50 | Cannot accept input filename as command line argument |
| - 50 | Cannot accept *sortType* as command line argument |
| - 30 | Cannot read input file |
| - 15 | Cannot output *Loc ID* sorted airport data to *stdout* |
| - 15 | Does not **exclude helipads** from input data |
| - 15 | Cannot output *Latitude* sorted airport data to *stdout* |
| - 15 | Does not convert *latitude* and *longitude* to decimal degrees |
| - 10 | Does not catch errors in *degrees* fields. |

# 6 Submission Instructions

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

- The main source file named `hw4Sort.c`
- A comment in the source file containing the following statement -"Your statement that the program is entirely your own work and that you have neither developed your code together with any another person, nor copied program code from any other person, nor permitted your code to be copied or otherwise used by any other person, nor have you copied, modified, or otherwise used program code that you have found in any external source, including but not limited to, online sources"