**CS 332/532 – 1G- Systems Programming**

**Lab 6**

### ****Objectives****

The objective of this lab is to introduce you to standard I/O libraries and I/O streams.

1. Introduction of standard I/O streams.
2. Write a program to read given comma separated file and use C structures to store and display the data.
3. Write a program to read given comma separated file, then sort given fields, and write sorted data into new file.

### ****Description****

#### **Example 1:**

*Open I/O Stream:*The standard I/O stream allows you to open a file in read, write, or append modes. This mode can be combined in a single open function call (see Figure 5.2 in Section 5.5 of the textbook for a complete list of options that can be specified). For example:

FILE \*fptr;   
fptr = fopen("listings.csv", "rw+");

Here file name is “listings.csv”, the file is open for reading and writing.

*Input Stream*: The standard I/O stream allows us to read from the open file. These functions allow us to read a file character by character – getchar(), line by line – fgets(), or with specific size – fread() (see Section 5.6 in textbook).

*Output Stream*: The standard I/O stream allow you to write to an open file. These functions allow you to write to a file character by character – putchar(), line by line – fputs(), or with specific size – fwrite() (see Section 5.6 in textbook).

Now let’s use these functions and write a program. We will use APIs available in Linux and C to develop different versions of this program as shown below:

1. uses getc fucntion to read one element at a time (we implement a function called getLine) - getline1.c
2. uses the getline function provided by the C library - getline2.c
3. uses getdelim function (similar to version 2, except that we use space as the delimiter not newline) - getline3.c
4. uses fgets function - getline4.c
5. uses fscanf function (note the difference between the first four versions and this version) - getline5.c
6. uses fprintf function to write output to a file and also uses fprintf instead of printf to write to standard output and standard error streams - getline6.c

You can use the corresponding man page to find out more about each of the functions used above. You can also extend the above examples to use putc, puts, putchar, fputc, and fputs functions to write the output.

#### **Example 2:**

We will now write a program to read a comma separated file (“listing.csv”) and use the C structures to store and display the data on the console. The program and the sample input file used are available here: listing.c and listings.csv

The given file has 13 different attributes and these attributes can be divided into three different datatypes: integer, character array, and float. And collectively we can create a C structure to represent these attributes and then create an array of such structures to store multiple entities.

1. Define a structure called listing with all attributes as individual members of the struct listing.

struct listing {   
 int id, host\_id, minimum\_nights, number\_of\_reviews, calculated\_host\_listings\_count, availability\_365;  
 char \*host\_name, \*neighbourhood\_group, \*neighbourhood, \*room\_type;   
 float latitude, longitude, price;   
};

2. Define a function which can help to parse each line in the file and return the above defined structure. For this task you need to learn the string tokenizer function (strtok) which is available in <string.h> header file. You can find out more about the strtok function by typing man strtok. You will notice that when you invoke the strtok function for the first time you provide the pointer to the character array and on subsequent invocations of strtok we use NULL as the argument.

struct listing getfields(char\* line){   
 struct listing item;   
  
 item.id = atoi(strtok(line, ","));   
 item.host\_id = atoi(strtok(NULL, ","));   
 item.host\_name = strdup(strtok(NULL, ","));   
 item.neighbourhood\_group = strdup(strtok(NULL, ","));   
 item.neighbourhood = strdup(strtok(NULL, ","));   
 item.latitude = atof(strtok(NULL, ","));   
 item.longitude = atof(strtok(NULL, ","));   
 item.room\_type = strdup(strtok(NULL, ","));   
 item.price = atof(strtok(NULL, ","));   
 item.minimum\_nights = atoi(strtok(NULL, ","));   
 item.number\_of\_reviews = atoi(strtok(NULL, ","));   
 item.calculated\_host\_listings\_count = atoi(strtok(NULL, ","));   
 item.availability\_365 = atoi(strtok(NULL, ","));   
  
 return item;   
}

***Note***: Read the man pages for the functions strtok, atoi, atof, strdup that are used in the above example.

3. Now use fopen function to open file in read only mode. Notice that the fopen function returns a pointer of FILE type (file pointer) unlike the open function that returns an integer value as the file descriptor.

FILE \*fptr = fopen("listings.csv", "r");

4. Then loop through till the end of file (use fgets function) and store all data in the array of structures.

count = 0;  
while (fgets(line, LINESIZE, fptr) != NULL){  
 list\_items[count++] = getfields(line);  
}

5. Now invoke the function to display the structure in a loop.

for (i=0; i<count; i++)  
 displayStruct(list\_items[i]);

6. Finally, use fclose function to close the file.

fclose(fptr);

### ****Homework****

Write a program to read the input file listings.csv and write two new functions that will sort the data based on the host\_name and price. You can perform the following steps in the C program:

1. Implement Step 1 to 5 as described in the Exercise above.
2. Now use any sorting algorithm to sort by the keys host\_name and price separately. Make sure when you sort on any given attribute that you rearrange the entire structure. For example, if you are sorting the whole list by host\_name then you need to change the order of structures based on the host\_name. Use the qsort function provided by the C library for sorting. Here is an example from the man page on how to use qsort function: funcptr2.c
3. Then open new files in write mode.
4. Then loop through all new structure and use fputs or similar write function to write the sorted structure to new files.
5. Close the files.
6. Test your program.
7. Create a file called README.md and include instructions for compiling and executing your program.

### ****Submission****

You are required to submit the lab solution to Canvas

**1. To upload solution to Canvas**: Upload the C source code, PDF version of C source code, and README.txt file to Canvas. Do not include any executable or object files.

Use the following command to create PDFs of your source code.(replace the <Source\_code\_File\_name> and <Output\_Source\_code\_File\_name> with your C source code file name):

$enscript <Source\_code\_File\_Name>.c -o - | ps2pdf - <Output\_Source\_code\_File\_Name>.pdf