CS 240: Programming in C
Fall 2020
Homework 2

Prof. Turkstra

Due September 16, 2020 9:00 PM

1 Goals

The purpose of this homework is to teach you file operation skills. You will open files, read data, process data, and write information to a new file.

2 Getting Started

1. Log in to one of the Purdue CS Linux systems. These include data.cs.purdue.edu and borgNN.cs.purdue.edu where NN=01,02, etc
2. Inside your cs240 directory, setup a hw2 directory by running the following commands...

   ```
   $ git clone ~cs240/repos/$USER/hw2
   $ cd hw2
   ```

3. This creates your hw2 directory, and also copies what is called a **Makefile** into it. This Makefile has a number of targets defined to make your life easier...

   When you are ready to try your work, you can run:

   ```
   $ make hw2_main
   ```

   to build hw2_main, or run:

   ```
   $ make hw2_test
   ```

   to build the test module. Or, just run:

   ```
   $ make
   ```

   to build both. For this assignment, the test module will produce a number of files in `test_data_files` folder under your directory. Running it many times will result in many files. You can run:

   ```
   $ make clean
   ```

   to clear out these temporary files.

   Note: any time you run make, your changes are automatically pushed to a git repository in the cs240 course account. As mentioned before, we **strongly** encourage you to always work on a Purdue CS Linux system. This mechanism cannot work otherwise.

3 The Big Idea

It is sometimes the case that one wishes to learn more about the auction price for sold houses. Common questions might include "how many houses did people sell last month?" or "what was the average selling price in a certain area?"

We have created input files that contain some basic information (described in Section 4.2) that can be used to answer questions similar to the above. Given these files, users will be able to formulate queries using specific constraints to extract information of interest from the raw data.

You will develop a simple set of database query functions that will:

- Calculate the total sold price for a specified month and year
- Calculate the maximum sold price for all sold houses
- Generate a list of all sold houses above a given threshold amount
- Generate a comparison on the total sold price in two different files for a specified month and year
- Calculate the average sold price for a specified area

Examine "hw2.h" carefully. It contains declarations for the global variables, constants, and the prototypes for the functions you will write.

Special Rules

- Just to make sure you know how to use `fopen()` and `fscanf()`, we won't allow you to use `access()`, `feof()`, or `ferror()` for this assignment. In other words, you should just use `fopen()`, `fscanf()`, and `fprintf()`.
- For this assignment, you will have to read a string name from a file into a fixed-size buffer. Just for this assignment, you may hard-code the value of the maximum length of that string into the format specifier for `fscanf()`. In practice, you should instead programmatically generate the format string, but since you have not yet been taught how, you may instead hard-code the value.
- When parsing files in this assignment, you should not create a large, fixed-size array into which you read all of the file's data before answering a particular question.

4 The Assignment

The necessary information about opening/reading from/writing to files is both in the class text and the lecture slides.

4.1 Functions you will write

`int get_month_sold(char *in_file, int month, int year);`

This function will open the datafile named *in_file*, read the records therein, and return the total sold price for the given month and year. If any errors occur, you should instead return an appropriate error code as described in Section 4.4.

`int get_max_sold(char *in_file);`

This function will open the datafile named *in_file*, read the records therein, and return the maximum sold price. If any errors occur, you should instead return an appropriate error code as described in Section 4.4.

`int get_auction_list(char *in_file, char *out_file, int cutoff);`

Given a cutoff value *cutoff*, this function will read *in_file* and, for each line, determine whether the sold price is above the provided cutoff. If it is, then the information of the data will be written to *out_file* as the following format.

For instance, if your input file looked like:

```
Jack,Tippecanoe,100000,09,01,2019
David,Warren,120000,13,05,2019
Jone,Clinton,80000,28,06,2019
Zoe,Warren,90000,16,07,2019
Penny,Clinton,120000,20,07,2019
```

And `get_auction_list()` was called with a cutoff value of 100000, the output would look like:

```
Name: David, Area: Warren, Price: 120000, 13/5/2019.
Name: Penny, Area: Clinton, Price: 120000, 20/7/2019.
```

Finally, this function should keep track of number of auctions above the threshold and return it upon completion. If any errors occur, you should instead return an appropriate error code as described in Section 4.4.

`int compare_price(char *in_file_1, char *in_file_2, char *out_file, int month, int year);`

For all data in *in_file_1* and *in_file_2*, this function will compare which file has the total higher sold price for the given month and year, and output it to *out_file*.

If the total sold price in *in_file_1* is higher than the total sold price in *in_file_2* for the given month and year, then the output format should be (x is the total sold price difference):

```
<in_file_1> has more overall sold price by x than <in_file_2> in mm/yyyy.
```

If the total sold price in *in_file_2* is higher than the total sold price in *in_file_1* for the given month and year, then the output format should be (x is the total sold price difference):

```
<in_file_2> has more overall sold price by x than <in_file_1> in mm/yyyy.
```

If the total sold price in both files for the given month and year are the same, then the output format should be:

```
<in_file_1> and <in_file_2> have the same overall sold price in mm/yyyy.
```

For instance, if your first input file looks like:

```
Jack,Tippecanoe,100000,09,01,2019
David,Warren,120000,13,05,2019
Jone,Clinton,80000,28,06,2019
Zoe,Warren,90000,16,07,2019
Penny,Clinton,120000,20,07,2019
```

And your second input file looks like:

```
Jack,Tippecanoe,120000,09,01,2019
David,Warren,120000,13,05,2019
Jone,Clinton,30000,28,06,2019
Zoe,Warren,20000,16,07,2019
Penny,Clinton,130000,20,07,2019
```

And `compare_price(‘‘file1.txt’’,‘‘file2.txt’’, 7, 2019)` was called, the output would look like:

```
file1.txt has more overall sold price by 60000 than file2.txt in 7/2019.
```

Finally, this function should return the absolute difference of the total sold price in a given month and year between two files. If any errors occur, you should instead return an appropriate error code as described in Section 4.4.

`float get_area_average(char *in_file, char *area);`

This function will open the datafile named *in_file*, read the records therein, and return the average sold price for the given area. If any errors occur, you should instead return an appropriate error code as described in Section 4.4.

4.2 Input Files

Each input file will contain a list of person's house sold information. The days need not be in any particular order. The files will be formatted as follows:

```
name,area,price,dd,mm,yyyy
name,area,price,dd,mm,yyyy
name,area,price,dd,mm,yyyy
...
```

- `name (char buffer[MAX_NAME_LEN])` - The name of the person to which all the following data belongs. This field may contain any non-newline ASCII character.

- `area (char buffer[MAX_AREA_LEN])` - The area of the sold house. This field may contain any non-newline ASCII character.
- `price (int)` - The price of the sold house.
- `dd (int)` - The day of the record.
- `mm (int)` - The month of the record.
- `yyyy (int)` - The year of the record.

And example file looks like this:

```
Jack,Tippecanoe,100000,09,01,2019
David,Warren,120000,13,05,2019
Jone,Clinton,80000,28,06,2019
Zoe,Warren,90000,16,07,2019
Penny,Clinton,120000,20,07,2019
```

4.3 Header Files

We provide a header file, "hw2.h", for you. It contains prototypes for each of the functions that you will write as well as #definitions for the constants. You should not alter this file. We will replace it with the original when grading.

4.4 Error Codes

- `BAD_RECORD`: Indicates unexpected characters/fields/wrong data type in a record.
- `BAD_DATE`: Indicates that a date in a specified input file is invalid, or that a date argument passed as an input to a function is invalid. You may assume that any date with a month between 1 and 12, a day between 1 and 31 (both inclusive), and a non-negative year is valid.
- `BAD_CUTOFF`: Indicates that a cutoff argument passed as an input to a function is invalid. You may assume that a non-negative cutoff is valid.
- `FILE_READ_ERROR`: Indicates that file cannot be opened for reading.
- `FILE_WRITE_ERROR`: Indicates that file cannot be opened for writing.
- `NO_DATA_POINT`: Indicates that the function would operate on no data. This includes if an input file itself has no data, **or**, for functions that take in a month/year/cutoff/area specification, if a file does not have any data for the specified month/year/cutoff/area combination.

These Standard Rules Apply

- You may add any #includes you need to the top of your hw2.c file.
- You may not create any global variables other than those that are provided for you. Creation of additional global variables will impact your style grade.
- You should check for any failures and return an appropriate value.
- You should not assume any maximum size for the input files. The test program will generate files of arbitrary size.
- Do not look at anyone else's source code. Do not work with any other students.

Submission

To submit your program for grading, type:

```
$ make submit
```

In your hw2 directory. You can do this as often as you wish. We encourage you to submit your code as often as possible. Only your final submission will be graded.

5 Grading

The operation of your functions will be graded out of 100 points. The point breakdown will be determined by the test program.

The test program will be run many times when grading. It is your job to do the same. The lowest score will be your final grade.

This homework will also have a style grade based on 20 points, with 1 point deducted for each code standard violation found.

Your code must compile successfully using `-Wall -Werror -std=c99` to receive any credit. Code that does not compile will be assigned an automatic score of 0.

If your program crashes (e.g., segmentation fault) at any point during the testing process, your score will be also be an automatic 0.