Assignment 5

- Monthly Weather Statistics

File:
- Create a file named ‘**assignment5.c**’ (you MUST name it assignment5.c).
- Submit your ‘assignment5.c’ in **Gradescope**.

Write a C program that reads a csv file and find the monthly weather statistics.

Requirements:
- The weather file is in comma-delimited csv format, and the first line is the header line.
- The weather file contains three columns: month, day and temperature.
- You find the average for each month.
- Your program should check if the month is from 1 to 12 and the day is from 1 to 31.
- If there are multiple rows for a specific month and a day, only use the first row for calculations.
  - Hint: you can maintain a 2D array (month x day) to keep track of which days have already been used for calculations.
- The average temperature is calculated by (total temperature) / (number of days calculated).
  - For example, if there are 10 days for January, the temperatures will be summed and then divided by 10 to find the average temperature.
- The statistics should be written in the output file named: (csv filename).txt, without the parenthesis.
  - For example, if the csv filename is ‘weather.csv’, the result will be recorded in ‘weather.csv.txt’.
- You can print out the content of the output file using the ‘cat’ command. The below is the usage of the command.
  - cat <filename>
- Follow the example interaction for the output format.

Example Interaction (<u>all interactions are command-line interactions</u>):

```
$ gcc -Wall assignment5.c
$ ./a.out  jan.csv
$
$
$ cat  jan.csv.txt
```

```
***** Jan *****
Measured days: 20
Average temperature: 64.00

***** Total *****
Measured months: 1
Measured days: 20
Average temperature: 64.00

$ ./a.out  random.csv
$ cat  random.csv.txt
***** Feb *****
Measured days: 20
Average temperature: 68.72

***** Jul *****
Measured days: 20
Average temperature: 86.53

***** Total *****
Measured months: 2
Measured days: 40
Average temperature: 77.62
```

```
$
$
$ ./a.out  random2.csv
$
$
$ cat  random2.csv.txt
***** Aug *****
Measured days: 10
Average temperature: 87.60

***** Dec *****
Measured days: 12
Average temperature: 62.79

***** Total *****
Measured months: 2
Measured days: 22
```

```
Average temperature: 74.07
```

```
$
$
$ ./a.out weather.csv
$
$
$ cat weather.csv.txt
***** Jan *****
Measured days: 31
Average temperature: 64.65

***** Feb *****
Measured days: 28
Average temperature: 70.27

***** Mar *****
Measured days: 31
Average temperature: 72.31

***** Apr *****
Measured days: 30
Average temperature: 76.95

***** May *****
Measured days: 31
Average temperature: 79.61

***** Jun *****
Measured days: 30
Average temperature: 83.23

***** Jul *****
Measured days: 31
Average temperature: 86.48

***** Aug *****
Measured days: 31
Average temperature: 86.89

***** Sep *****
```

```
Measured days: 30
Average temperature: 83.78

***** Oct *****
Measured days: 31
Average temperature: 77.55

***** Nov *****
Measured days: 30
Average temperature: 70.43

***** Dec *****
Measured days: 31
Average temperature: 64.92

***** Total *****
Measured months: 12
Measured days: 365
Average temperature: 76.45
```

Grading scale:

- Total grade: 100
- A program that does not compile will result in zero.
- Runtime error
    - If your program gets 0 from the Gradescope Autograder, the TAs will download your program and run it manually. However, you will receive 50% of each test case only when the output is correct.
- Compilation warning: 5 points deduction
- Comments (refer to Chapter 2.3 from the textbook)
    - There is deduction for not putting student name, NetID, and description in the first line of the program.
    - Your program should have comments before program blocks.
- Indentation
    - 4 spaces recommended
    - Missing or incorrect indentation will get deduction.
    - Refer to the page 169 for ‘interest.c’ or page 173 for ‘deal.c’ in the textbook.

Rubric:

Comments	15 points
Indentation	15 points
Compilation warning	5 points deduction
Runtime error	5 points deduction
Test cases (Gradescope autograder)	60 points
Total	100 points

Gradescope hints:

- If your program gets a **timeout error**, it may have **an infinite loop** in one of the test cases. Pay attention to any loop statements (do…while, while and for loops).
- You can submit source code in Gradescope as many times as necessary until you achieve the desired grade.
- Gradescope matches each letter exactly. If you are missing a punctuation (space, period, comma or new line) or having a case error, Gradescope will mark your answer as incorrect. Pay attention to details in your results.
- A maximum of 70 points can be obtained from the Gradescope autograder.
- The last grade you received will be used for grading. You can access previous submissions by clicking 'Submission History' → 'Activate' from the 'Autograder Results.'
- After the deadline, the TAs will grade your last submission using the rubric.

Programming Style Guidelines:

The major purpose of programming style guidelines is to make programs easy to read and understand. Good programming style helps make it possible for a person knowledgeable in the application area to quickly read a program and understand how it works.

- Your program should begin with a comment that briefly summarizes what it does. This comment should also include your name and NetID.
- In most cases, a function should have a brief comment above its definition describing what it does. Other than that, comments should be written in front of a programming block in order for a reader to understand what is happening.

- Variable names and function names should be sufficiently descriptive that a knowledgeable reader can easily understand what the variable means and what the function does. If this is not possible, comments should be added to make the meaning clear.
- Use consistent indentation to emphasize block structure.
- Full line comments inside function bodies should conform to the indentation of the code where they appear.
- Macro definitions (#define) should be used for defining symbolic names for numeric constants. For example: #define PI 3.141592
- Use names of moderate length for variables. Most names should be between 2 and 20 characters long.
- Use underscores to make compound names easier to read: tot_vol and total_volumn are clearer than totalvolumn.