Assignment: Problem-Solving with Structures and Files

Learning Outcomes

- Structures, structures in functions, and arrays of structures
- Command-line parameters
- File input/output
- Standard C library functions

Task

The objective of this assignment is to develop a program containing structures, structures in functions, and arrays of structures using appropriate functions from the C standard library. The program will print a report giving the maximum wave height for the tsunamis recorded in a data file. In addition to the maximum wave height, the report must include the average wave height and the location of all tsunamis with a wave height higher than the average height. All heights in the data file are measured in meters.

Implementation Details

Data structure and functions

Declare data structure `struct Tsunami` to encapsulate heterogeneous information relevant to a tsunami event. The data members encapsulated in `struct Tsunami` must include the tsunami's date of occurrence [including month, day, and year], its maximum wave height, number of human fatalities caused by the tsunami, and geographical location. The month, day, year, and number of fatalities are represented as `int`s, the maximum wave height is represented as a double-precision floating-point number, and the location is represented by a null-terminated character string with a maximum of 80 characters [not including the null character].

Declare the following functions to solve the problem:

```c
int    read_data(char const *file_name, struct Tsunami *arr, int max_cnt);
void   print_info(struct Tsunami const *arr, int size, FILE *out_stm);
double get_max_height(struct Tsunami const *arr, int size);
double get_average_height(struct Tsunami const *arr, int size);
void   print_height(struct Tsunami const *arr, int size,
                    double height, FILE *out_stm);
```

A brief summary of each of the five functions follows:

1. Function `read_data` copies a maximum of `max_cnt` number of tsunami events recorded in text file specified by parameter `file_name` into an array whose base address is specified by parameter `arr`. Your implementation must open the text file, check if the file exists, and then copy the information recorded for each event into a variable of type `struct Tsunami`. The function should return the number of tsunami events stored in the array. Each *line* of the text file contains information for each tsunami event in the following order: integer

values representing month, day, year, fatalities, a double-precision floating-point value representing maximum wave height and a sequence of characters (that may contain whitespace characters) representing the tsunami's location. Study the format of input data in text files `tsunamis?.txt` that can be downloaded from the assignment web page.

2. Function `print_info` prints to output stream `out_stm` a nicely formatted list of the information recorded in each tsunami event in the following order and format:
   - month [formatted with 2 digits or more, left padded with zeroes],
   - day [formatted with 2 digits or more, left padded with zeroes],
   - year [formatted with width of at least 4 digits],
   - number of fatalities [formatted with width of at least 6 digits],
   - maximum wave height [formatted with width of at least 5 digits and precision of 2 digits], and
   - the location of the tsunami [printed as a character string].

   Output text files `output?.txt` provide examples of the pretty table that your submission must generate.

3. Function `get_max_height` returns the maximum wave height in array specified by base address `arr` with `size` number of tsunami events.

4. Function `get_average_height` returns the average wave height in array specified by base address `arr` with `size` number of tsunami events.

5. Function `print_height` prints to output stream `out_stm` the maximum wave heights and locations of those tsunamis with higher maximum wave heights than the wave height specified by parameter `height`. Parameters `arr` and `size` represent the base address of an array of tsunami events and the number of events to evaluate, respectively. The maximum wave height must be formatted with width of at least 5 digits and precision of 2 digits.

Strategy

As usual, you'll declare structure type `struct Tsunami` and the five functions in `q.h`. Include `<stdio.h>` to access type `FILE *`. Define the five functions declared in `q.h` in source file `q.c`. You can use any functions declared in the C standard library to implement the necessary functions. To successfully compile `q.c`, don't forget to include C standard library headers [containing declarations of those library functions you're using in the definitions of the five functions].

Download the driver source file `qdriver.c`, a *makefile*, input files containing tsunami events, and corresponding (correct) output files. Use the strategy of providing *stub* functions to establish and verify linkage between source files `qdriver.c` and `q.c`, and implementing and verifying the correct behavior of one function at a time. Build executable `q.out` using *makefile*:

```
$ make
```

and use command-line parameters to specify the input and output file names to the program:

```
$ ./q.out tsunamis1.txt myoutput1.txt
```

Compare the output from your implementation with the correct implementation in `output1.txt` using `diff`:

```
$ diff -y --strip-trailing-cr --suppress-common-lines myoutput1.txt output1.txt
```

If you're confident about structures, arrays of structures, passing structures to functions, file I/O, reading data from a text file to an array of structures, and can implement the required functions, proceed with the assignment. Otherwise, review this material using the handout on structures [available on the course web page].

File-level and Function-level documentation

Every source and header file you submit *must* contain file-level documentation blocks whose purpose is to provide human readers [yourself and other programmers] useful information about the purpose of this source file at some later point of time

Every function that you declare in a header file [and define in a corresponding source file] must contain a function-level documentation block.

> ***Don't copy and paste documentation blocks from previous assignments. Annoyed graders will definitely subtract grades to the full extent specified in the rubrics below when they detect such copy-and-paste scenarios.***

Submission and automatic evaluation

1. In the course web page, click on the submission page to submit `q.h` and `q.c`.
2. Read the following rubrics to maximize your grade. Your submission will receive:
    1. F grade if your submission doesn't compile with the full suite of `gcc` options [shown above].
    2. F grade if your submission doesn't link to create an executable.
    3. $A+$ grade if the submission's output matches the correct output. Otherwise, a proportional grade is assigned based on how many incorrect results were generated by your submission.
    4. A deduction of one letter grade for each missing documentation block. Every submitted file must have one file-level documentation block. Every function that you declare in a header file must provide a function-level documentation block. A teaching assistant will physically read submitted source files to ensure that these documentation blocks are authored correctly. Each missing block will result in a deduction of a letter grade. For example, if the automatic grader gave your submission an $A+$ grade and the three documentation blocks are missing, your grade will be later reduced from $A+$ to $B+$. Another example: if the automatic grade gave your submission a C grade and the three documentation blocks are missing, your grade will be later reduced from C to F.