

# Lab: Transitioning From C to C++ [Part 2]

## Learning Outcomes

- Gain experience with references,
- Gain experience with C++ dynamic memory allocation/deallocation operators
- Gain experience in defining simple user-defined types in C++
- Gain experience with file input/output techniques
- Gain experience with C++ standard library type `std::string`
- Gain experience with formatting output
- Gain experience with namespaces
- Practice overall problem-solving skills, as well as general design of a program
- Reading, understanding, and interpreting C++ code written by others

## Task

The objective of this lab is to develop a program containing structures, dynamically allocate array of structures, read data stored in a text file in functions into the dynamically allocated array, process the data contained in the array, and write relevant formatted data to an output text file only using facilities provided by C++ and the C++ standard library.

The program will print a report about all tsunami events recorded in a text file. In addition, your program must report the largest maximum wave height, the average wave height of all the tsunami events, 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.

Begin the task by studying the format of input data in one of the text files `tsunamis?.txt`. You'll notice that 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.

Define a type `Tsunami` in header file `q.hpp` to encapsulate heterogeneous information relevant to a tsunami event:

```

1  #ifndef Q_HPP
2  #define Q_HPP
3
4  #include <string> // to use C++ standard library std::string type
5  #include <fstream> // to use C++ file I/O interface
6  // other C++ [not C] standard library headers
7
8  namespace hlp2 {
9
10     struct Tsunami {
11         // data members
12     };
13
14 } // end namespace hlp2
15 #endif
16
```

The data members encapsulated in `Tsunami` must include the month, day, and year of the tsunami's occurrence [represented by `int`s]; its maximum wave height [represented by a `double`]; the number of fatalities [represented by an `int`]; and the geographical location [represented by a `std::string`].

In addition to type `Tsunami`, the header file `q.hpp` must declare the following interface:

```

1 namespace hlp2 {
2
3 // define Tsunami structure here ...
4 Tsunami* read_tsunami_data(std::string const& file_name, int& max_cnt);
5 void      print_tsunami_data(Tsunami const *arr,
6                               int size, std::string const& file_name);
7
8 } // end namespace hlp2

```

A brief summary of the interface follows:

- Function `read_tsunami_data` must do the following:
  - determine the number of tsunamis recorded in the input file specified by parameter `file_name` and then write this count to reference parameter `max_cnt`
  - dynamically allocate an array [where each element is of type `Tsunami`] with `max_cnt` number of elements
  - read each tsunami event from input file and store relevant data to the corresponding element of the dynamically allocated array
  - return a pointer to the first element of the dynamically allocated array
  - if the input file doesn't exist, the function should return a `nullptr`.
- Function `print_tsunami_data` prints to an output file stream reports about the tsunami events in array specified by parameter `arr`. Output text files `output?.txt` provide examples of the pretty table that your submission must generate. First, the function must open an output file stream that connects your program to the file name specified by parameter `file_name`. If the output file stream cannot be created in a good state, the function should do nothing and just return. Next, the function should print a nicely formatted list of the information recorded in each tsunami event as specified by the output file and in the following order and format:
  - month (formatted with 2 digits, left padded with zeroes),
  - day (formatted with 2 digits, left padded with zeros),
  - year (formatted with width of 4 digits),
  - number of fatalities (right-aligned),
  - maximum wave height (right-aligned with precision of exactly 2 digits), and
  - the location of the tsunami (left-aligned).

Next, the function must print certain summary statistics including

- the largest maximum wave height of the tsunami events recorded in the array
  - average maximum wave height of all the tsunamis
  - a formatted table of the list of the maximum wave heights and locations of those tsunamis with higher maximum wave heights than the average maximum wave height of all the tsunamis.
- Carefully examine the input and output files to ensure your output matches mine.

## Implementation Details

Begin the solution by looking at the input and corresponding output files to understand what needs to be done. Pencil an algorithm that takes the input and transforms it to the corresponding output.

After devising an overall algorithm, you're ready to the C++ implementation of the algorithm. The first step would be to be able to read each line in the input file [because there are as many tsunamis as the number of lines of text in the input file]. Research function `getline` from the C++ standard library. Once you know how many lines there are in the input file, you know how many tsunamis there are. Dynamically allocate memory for the appropriate number of `Tsunami` objects. Set the next character to read from the file back to the top of the file. Then, read each line, process the information in that line, and write the corresponding information to the output file.

Writing tabular information to the output file with proper alignment so that your output is exactly similar to my output will be the next challenge. You should look at the [manipulators](#) presented in `<iomanip>`: `left`, `right`, `setw`, `fixed`, `setprecision`, `setfill`, and so on.

You should think about authoring functions [these should not be part of `h1p2` namespace since they're private implementation details of `q.cpp`] such as:

1. Function `largest_max_ht` to return the largest maximum wave height in the tsunami events.
2. Function `avg_max_ht` to return the average maximum wave height of the tsunami events.

After writing the largest maximum wave height information and average maximum height information, the only thing left is to write the tabular information about tsunami events with maximum wave heights higher than the average maximum wave height.

You should write the solution using the following headers in the C++ standard library: `<string>`, `<iostream>`, `<fstream>`, and `<iomanip>`. It is ok to use additional headers from C++ standard library but it is not ok to use C headers - such headers are not necessary and the auto grader will not accept your submission.

Read this again: ***Your implementation can include headers `<string>`, `<iostream>`, `<fstream>`, and `<iomanip>` [plus any other C++ headers you like]. Addition of any other headers [even in comments] such as `<cstring>`, `<cstdlib>` and the use of functions not declared in these headers such as `std::malloc`, `std::strlen` or just `strlen` [even in comments] will prevent the auto grader from accepting your submission.***

### Header file `q.hpp`

You must provide a declaration of function `q` in namespace `h1p2` in header file `q.hpp`. Details are given above.

### Source file `q.cpp`

You must provide a definition of the necessary functions in namespace `h1p2` in source file `q.cpp`. It would look like this:

```

1 // include necessary C++ header files ...
2
3 // Important notes: The auto grade will not accept any functions declared in
4 // C standard library [even in comments]!!!
5 // You're warned!!!
6
7 namespace hlp2 {
8     // provide definitions of functions here ...
9 }

```

## Driver `q-driver.cpp`

The driver accepts two additional parameters in addition to the name of the program indicating the name of the input file to analyze and the name of the output file containing the analysis. You are given three files to test: `tsunamis1.txt`, `tsunamis2.txt`, and `tsunamis3.txt`, and corresponding correct output files `output?.txt`.

## Submission Details

Please read the following details carefully and adhere to all requirements to avoid unnecessary deductions.

### Header file

Submit `q.hpp`.

### Source file

Submit `q.cpp`.

## Compiling, executing, and testing

Download `q-driver.cpp`, and input and output files from the assignment web page. Create the executable program by compiling and linking directly on the command line:

```

1 $ g++ -std=c++17 -pedantic-errors -Wall -Wextra -Werror q.cpp q-driver.cpp -o
   q.out

```

In addition to the program's name, function `main` is authored to take two command-line parameters to specify the name of the input file and the name of the output file.

```

1 $ ./q.out tsunamis1.txt your-output1.txt

```

Compare your submission's output with the correct output in `output1.txt`:

```

1 $ diff -y --strip-trailing-cr --suppress-common-lines your-output1.txt
   output1.txt

```

If the `diff` command is not silent, your output is incorrect and must be debugged.

You'll have to repeat this process for the remaining input files.

## File-level and function-level documentation

Every source and header file *must* begin with a *file-level* documentation block. This module will use [Doxygen](#) to tag source and header files for generating html-based documentation. In addition, every function that you declare and define and submit for assessment must contain *function-level documentation*. This documentation should consist of a description of the function, the inputs, and return value.

## Submission and automatic evaluation

1. In the course web page, click on the appropriate submission page to submit the necessary files.
2. Please read the following rubrics to maximize your grade. Your submission will receive:
  - *F* grade if your submission doesn't compile with the full suite of `g++` options.
  - *F* grade if your submission doesn't link to create an executable.
  - Your implementation's output doesn't match correct output of the grader (you can see the inputs and outputs of the auto grader's tests). The auto grader will assign 50% of the grade based on the input and output files given to you. The remaining 50% of the grade will be awarded based on the additional tests implemented by the auto grader.
  - The auto grade will provide a proportional grade based on how many incorrect results were generated by your submission. *A+* grade if your output matches correct output of auto grader.
  - A deduction of one letter grade for each missing documentation block in your submissions. Each source file must have **one** file-level documentation block and a function-level documentation block for each defined function. A teaching assistant will physically read submitted source files to ensure that these documentation blocks are authored correctly. Each missing or incomplete or copy-pasted (with irrelevant information from some previous assessment) block will result in a deduction of a letter grade. For example, if the automatic grader gave your submission an *A+* grade and one documentation block is 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 two documentation blocks are missing, your grade will be later reduced from *C* to *F*.