# Assignment: STL and Data Processing

***If your submission generates the correct output by using one or more things in the [Restricted List](#) (yes - I admit the list is lengthy but rules are rules and are not meant to be broken), then don't submit to the auto-grader!!! What happens if you decide to ignore this warning? A human grader will regrade your submission to zero points. Second, subverting assessments by passing something off as something else is a violation of the academic integrity policy and appropriate remedies as cited in that policy will be enforced.***

## Learning Outcomes

This assignment will provide you with the knowledge and practice required to develop and implement software involving:

- Practice use of STL templates and algorithms.
- Practice working with STL containers.
- Practice use of callable objects and predicates in processing data within containers.

## Requirements

Your task is to implement the *solution.h* file that defines 5 missing functions used by the driver:

- `print_file_names(const file_records& map);`
- `print_non_empty_files(const file_records& map);`
- `print_empty_files(const file_records& map);`
- `get_parameters(file_records& map);`
- `remove_empty(file_records& map);`

There are a couple of challenges in solving this puzzle. First, you have to figure out the purpose, parameters and the return data types of each function based on the driver code. Second, you must observe a lengthy list of restrictions.

Study the templates available in the <functional>, <algorithm>, and <utility> headers before implementing any code for this assignment. You may find the following STL templates useful:

- `std::begin()`, `std::end()`, `std::size()`
- `std::back_insert_iterator`, `std::back_inserter()` std::inserter()
- `std::bind()`
- `std::copy()`, `std::copy_if()`
- `std::count()`, `std::count_if()`
- `std::reference_wrapper`, `std::ref()`, `std::cref()`
- `std::move()`, `std::forward()`, `std::forward_as_tuple()`
- `std::transform()`
- `std::tuple`, `std::make_tuple()`

## Restrictions

***You are not allowed to:***

- Include any headers.
- Define any other functions than the ones mentioned above.
- Define any new complex types or templates; type aliases are permitted.

- Use lambda expressions.
- Use following operators:
  - `.` (member access),
  - `->` (member access via a pointer),
  - `*` (dereference).
- Use explicit iteration (`for`, `while`, `do while`) or selection (`if`, `switch`, `?:`) statements or operators. You can use functions or function templates (such as `std::for_each`) to iterate over a collection of elements.
- Use `std::cout`, `std::cerr` or any other functions that perform printing of text to the console; you have to use the provided function to do it.
- Use keyword `auto`.

## Submission Details

Please read the following details carefully and adhere to all requirements to avoid unnecessary deductions. **Valgrind** and documentation using **Doxygen** is required.

### Submission file(s)

You're required to submit `solution.hpp`. Without any comments you can expect the file to be around 60 lines.

### Compiling, linking, and testing

To compile and test your code, simply execute the following command:

```
g++ -Wall -Werror -Wextra -Wconversion -pedantic -std=c++17 -o main main.cpp
./main > actual-output.txt
diff actual-output.txt expected-output.txt --strip-trailing-cr
```

Make sure that the resulting file `actual-output.txt` matches the provided file `expected-output.txt`.

### Automatic evaluation

1. In the course web page, click on the appropriate submission page to submit `solution.hpp`.
2. Please read the following rubrics to maximize your grade:
   - Your submission will receive an $F$ 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 provide a proportional grade based on how many incorrect results were generated by your submission. $A+$ grade if output of function matches correct output of auto grader.
   - A maximum of $D$ grade if Valgrind detects even a single memory leak or error. A teaching assistance will check you submission for such errors.
   - A deduction of one letter grade for missing file-level documentation in `solution.hpp`. A deduction of one letter grade for each missing function definition documentation block in `solution.hpp`. Your submission must have **one** file-level documentation block and function-level documentation blocks for ***every function you're defining***. 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 $E$.