

# 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? First, a script will do the hard work of detecting subversive submissions and 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

---

The following functions are used by the driver implemented in `main.cpp`:

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

Your task is to provide the definitions of these functions in `solution.hpp`. There are a couple of challenges in solving this puzzle. First, using the driver code, you've to figure out the purpose of each function, the meaning of the function's parameters, and the data type of the function's return value. Second, you must provide the definition of each function while observing a lengthy list of [restrictions](#).

Study the following templates [declared in [`<functional>`](#), [`<algorithm>`](#), and [`<utility>`](#) headers] before implementing any code for this assignment:

- `std::begin()`, `std::end()`, `std::size()`
- `std::back_insert_iterator`, `std::back_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.

## Submission file(s)

You will be submitting header file `solution.hpp`.

## Compiling, executing, and testing

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

```
1 | g++ -Wall -Werror -Wextra -Wconversion -pedantic -std=c++17 -o main main.cpp
2 | ./main > actual-output.txt
3 | 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`.

## Documentation

This module will use [Doxygen](#) to tag source and header files for generating html-based documentation. Every source and header file *must* begin with *file-level* documentation block. 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. Click on the submission page to submit the necessary file(s).
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 `g++` options [shown above].
  2. *F* grade if your submission doesn't link to create an executable.
  3. A maximum of *D* grade if Valgrind detects even a single memory leak or error. A human grader will check your submission for such errors.
  4. *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.
  5. 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*.