Assignment - C++ Standard Lib: Data Containers

Topics and references

- Standard library containers.
- Linked lists.
- C++ strings.

Learning Outcomes

- Practice using linked lists in a real-world problem.
- Experience using templates in a useful application.
- Perform string manipulation in a meaningful way.

Task

1. Implement a class that performs a search in an input text file for a given search `string` and place in a `std::list` linked list the corresponding starting subscript index position for each time the search text is found.
2. The search is to be non-case-sensitive and would not be restricted to whole words, for example, if the input file contains `Hello world`, a search for `hello world` would have only `0` added to the linked list. A search for `lo` would result in just `3` in the list, and a search for `l` would cause the list to contain `2`, `3`, and `9` in ascending order. Lastly, a search for `hlp` must result in an empty list since `hlp` is not in the input file.
3. The given `print` function outputs the contents of the list to standard output as follows:

   For the search for `hlp` the output would be

   ```
   'hlp' not found
   ```

   For the search on `lo` the output would be

   ```
   'lo' found at 1 character position(s): 3
   ```

   And for the search for `l` the output would be

   ```
   'l' found at 3 character position(s): 2    3    9
   ```

   The above output formatting of the list contents `2`, `3`, and `9` uses the insertion operator that you are to overload.

Submission Details

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

Header file

There is no header to submit.

Source file

You are to implement in `q.cpp` the text searcher class `finder` that is defined in `q.h` and submit `q.cpp`. `finder` is defined as follows:

```
#ifndef Q_H
#define Q_H

#include <fstream>
#include <string>
#include <list>

namespace hlp2
{
class finder
{
public:

    using size_type = size_t;
    using container_type = std::list<size_type>;

    finder(std::string const&);

    void find(std::string const&);
    void print();

private:

    std::string search_space_filename;
    std::ifstream search_space_stream;
    std::string search_space_txt;
    std::string search_str;
    container_type pos_ls;
};
}
#endif
```

Compiling, executing, and testing

Download `qdriver.cpp`, `q.cpp`, `q.h`, `makefile`, `search-space.txt` that has the document to search, and `output2.txt` which has the expected output.

Run `make` with the default rule to bring program executable `q.out` up to date:

```
$ make
```

Directly test your implementation by running `make` with target `test`:

```
$ make test
```

If the `diff` command in the `test` rule is not silent, then one or more of your function definitions is incorrect and will require further work.

Documentation

This module uses Doxygen to tag source and header files for generating html-based documentation. The 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. In the course web page, click on the appropriate submission page to submit `q.cpp`.
2. Please read the following rubrics to maximize your grade. Your submission will receive:
    - `F` grade if your `q.cpp` doesn't compile with the full suite of `g++` options.
    - `F` grade if your `q.cpp` 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 deduction of one letter grade for each missing documentation block in `q.cpp`. Your submission `q.cpp` must have **one** file-level documentation block and **one** function-level documentation block. 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`.