Programming Assignment: More Problem Solving with `vector`s and `string`s

Learning Outcomes

- Gain experience in functional decomposition and algorithm design
- Gain experience in using C++ standard library to solve practical problems

Task

A generalized checkerboard [of characters] is a rectangular grid with four parameters:

- the number of rows R
- the number of columns C
- the starting character $start$
- the cycle size $cycle$

The grid is a $R \times C$ matrix, where the element in row r and column c contains character $start + (r + c) \% cycle$. For example, here's a generalized checkerboard with $R = 5$, $C = 6$, $start =$ `'a'`, and $cycle = 4$:

$$
\begin{array}{c}
\mathrm{a\ b\ c\ d\ a\ b} \\
\mathrm{b\ c\ d\ a\ b\ c} \\
\mathrm{c\ d\ a\ b\ c\ d} \\
\mathrm{d\ a\ b\ c\ d\ a} \\
\mathrm{a\ b\ c\ d\ a\ b}
\end{array}
$$

Your task is to write the [entire] program that prints the checkerboard by reading five command-line parameters [with the program name, there will be six command-line parameters]. The first four parameters specify R, C, $start$, and $cycle$ [in that order] are used to create and fill the grid with values. The remaining parameter specifies an integral width W so that each grid element is printed as a $W \times W$ square.

Here are some examples:

```
$ ./board.out 5 6 a 4 1
abcdab
bcdabc
cdabcd
dabcda
abcdab
$ ./board.out 5 6 a 4 3
aaabbbcccdddaaabbb
aaabbbcccdddaaabbb
aaabbbcccdddaaabbb
bbbcccdddaaabbbccc
bbbcccdddaaabbbccc
bbbcccdddaaabbbccc
cccdddaaabbbcccddd
cccdddaaabbbcccddd
```

```
cccdddaaabbbcccddd
dddaaabbbcccdddaaa
dddaaabbbcccdddaaa
dddaaabbbcccdddaaa
aaabbbcccdddaaabbb
aaabbbcccdddaaabbb
aaabbbcccdddaaabbb
$
```

Implementation Details

Your interface should be declared in `pa.hpp`:

```
// cmdline_params includes all command-line parameters
// except first parameter [which is the program'name] ...
mystery_type create_board(std::vector<std::string> const& cmdline_params);
// width could be empty string if the required number of command-line
// parameters are not provided by client ...
void print_board(mystery_type const& board, std::string const& width);
```

and must be implemented in `pa.cpp`.

> ***Make sure to add compilation guards in your header file. Every public name in `pa.hpp` and `pa.cpp` should be scoped in namespace `hlp2` to avoid polluting the global namespace. To conserve space, example code will not be encapsulated in namespaces!!!***

Note that this assignment is an exercise in using the standard library to solve practical problems. Therefore, you should avoid explicit use of [raw] pointers and the use of operators `new`, or `new[]`, or `delete`, or `delete[]`. A brief review of the containers covered in lectures and previous assignments should quickly lead you to deciphering the exact nature of `mystery_type`.

As the names of the functions indicate, `create_board` fills the rectangular grid with appropriate values while `print_board` prints values in a rectangular grid to the standard output stream. This idea that a function will only do one thing and do that thing well is called single responsibility principle (SRP). Not adhering to the SRP may prevent your submission from generating incorrect results because the grader may only call one of the two functions or call them in arbitrary order.

Your program needs to be robust and should behave in the following manner:

- Your functions should print the same error message as mine on the standard error stream if too few command-line parameters are given to the program. Use the output files to determine the exact composition of the error message.
- Your functions must *silently exit* if any of the integral parameters are less than or equal to zero. The program should not crash nor print unnecessary diagnostic messages.
- Your functions must *silently exit* if the ASCII value of the start character plus the cycle size is greater than the largest correct ASCII value. The program should not crash nor print unnecessary diagnostic messages.

Submission Details

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

Header and source files

Submit `pa.hpp` and `pa.cpp`.

Compiling, executing, and testing

Use the many drivers and makefiles provided in HLP1 and HLP2 to design, implement, compile, execute, and test your interfaces and implementations. To test your implementation, output files are provided with the following format: text file `./input/???.txt` contains the command-line parameters while corresponding text file `./output/???-output.txt` contains the output generated by a correct implementation of the interface.

Your implementation's output for the input specified in `./input` must exactly match the concatenated files in `./output` . There are only two grades possible: $A+$ grade if your output matches correct concatenated output; otherwise F.

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 must exactly match correct output of the grader (you can see the inputs and outputs of the auto grader's tests). There are only two grades possible: $A+$ grade if your output matches correct output of auto grader; otherwise F.
    - 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.