Assignment #1

Due date: Friday, 19 January 2024, 8:00 pm

- *Using `git`, get a copy of the assignment files from*
  `https://git.uwaterloo.ca/cs138-2024/skeleton/assgt1`
- *Submit your solutions using Marmoset as a single file under `a1qX.tar.gz` name for each question `X`. You can run the provided packaging script, `package.sh`, to generate this file and also use `submit_all.sh` to upload all of the questions into Marmoset.*
- *For each question `X`, there will be a directory named `a1qX`. Here, you will find a stub file named `a1qX.cc` in which you should implement your code; there will also be subdirectories for testing called `inputs` and `outputs-expected` (see below).*
- *We expect that your test cases provide 100% statement coverage for each `a1qX.cc` that you create.*
- *Providing complete statement coverage does not ensure the program's correctness; write enough unit tests to feel safe about your function's behaviour. Otherwise, you may lose marks if your code generates an incorrect result.*
- *For all programming questions below, write your solutions in the dialect of C++ used in class. You may use the following libraries and no others: `iostream`, `string`, `cassert`, `vector`.*
- *Use the C++ `string` class for character strings; do* not *use C-style `char*` variables (except for the `argv` parameter to `main`, which is unavoidable). The C++ `string` API has a lot of helpful features than can do some of the work for you.*
- *Feel free to define any sub-functions within the file to break things up and make your life easier. However, you are* ***not*** *allowed to add new files.*
- *Note that some questions are harder than others, and the harder ones will be worth more marks.*
- *Part one of assignment #2 builds on this work, so be sure you put in a good effort or you will have a lot of work to do.*

Directories

a1qX/ The directory for all files for question X, including your solution and the test inputs/outputs. Each question gets its own directory; feel free to reuse (copy) your tests across different questions.

a1qX/inputs/ The test input files for question X. For this assignment, each file in this directory will correspond to what you want to appear in the standard input stream during a single run of the program; each file will be named `testYY.in`, where `YY` is a two digit number. We will provide a couple of examples (plus the expected outputs, see below), but you should add new test cases of your own design here.

a1qX/outputs-expected/ The expected output files for question X. For each file `testYY.in` contained in the `inputs` directory above, there should be two files here — `testYY.out` and `testYY.err` — which correspond to the expected output that go to the standard output and error streams. Again, we will provide a few examples here, but you should also add your own pairs for each input file you provide.

scripts/ These are the scripts to compile, run tests, `valgrind`, `gcov`, etc. You will be running these scripts, but you will not have to change any files in this directory.

Scripts (in the scripts/ directory)

run_main.sh: Run code for a solution in interactive mode, so you can manually type in an input and see what happens.

run_tests.sh: Run all the tests for a question in batch mode. Examples:

- ./run_tests.sh 2 *run all the tests for q2*
- ./run_tests.sh 3 gcov *report the coverage metric for the q3 tests*
- ./run_tests.sh 4 valgrind *run valgrind on the q4 solution while running the tests*

submit_all.sh: Submits A1 to Marmoset. Uses the `package.sh` script to create a single tarball for each question with all of the `.cc` and test files.

others: There are a few other scripts. These are the main ones you need.

Assignment #1 questions

1. Write a program that reads in a positive integer `N` followed by several lines of text; note for all questions in this assignment you should use only the standard input stream `cin` for input. You should then print (to the standard output stream `cout`) each of the lines in the order in which they were read in, as long as the line is no more than `N` characters long. If any line exceeds `N` characters, print only the first `N` characters of that line. When you reach EOF, you are done reading (but don't forget to print the last line of text).

   Do *not* print `N` as part of your output. Use token-oriented input to read in `N` and line-oriented input to read in each line of text. In this question and the next, you should preserve (*i.e.,* not worry about) any existing whitespace in the input text that occurs after you've read in `N`.

   You may assume (without having to check) that the first token in the input stream is an integer and that it occurs on the first line of input, and that there is nothing else of interest on that line. However, you *should* check that `N` is positive; if not, print this error message on a line by itself (it must be exact, including spacing, but ignoring the leading spaces) to the standard error stream `cerr` (be sure to "flush" `cerr` with an `endl` after, here and elsewhere) and exit immediately.

   ```
   Error, line length must be positive.
   ```

   You should also use this error message for any following question that asks you to read in a positive integer `N` at the start. Assume that the text to be processed starts on the second line of input. Note that you may have to "eat" the newline character after `N` in the first line of input, which you can do with `cin.ignore()`, or a call to `getline` (and ignore whatever value is returned).

2. Extend Q1 by reading in a positive integer `N` plus a single character `command` on the first line of input. If `command` is `'f'`, then do as in Q1. If `command` is `'r'`, print the lines (well, the first `N` characters of each line) in reverse order (*i.e.,* print the last line read in first, then the second-to-last line read in, *etc.*). If `command` is `g`, print only the (first `N` characters of the) lines that contain the special string `fnord` (in the order in which they were read in); note `fnord` could appear as a substring of a token (*e.g.,* `fnordly`), and you want to catch that case also. If `command` is any other character, print this error message to `cerr`:

   ```
   Error, command is illegal.
   ```

   Use token-oriented input for `N` and `command`, and line-oriented input for the text that follows. Note that in this question, the command is assumed to be a single `char`, not a `string`.

3. You will use only token-oriented input for this question. Write a program that reads in a positive integer `N` followed by a sequence of tokens. You are to print the tokens to the standard output organized in lines of at most `N` characters, with a single space between each token (but no trailing spaces after the last character of the last token on a line), in the order in which they are read in.

   For example, for this input stream:

```
35 12345678901234567890123456789012345678901234567890 Who steals
 my purse steals trash; 'tis something, nothing;
'Twas mine,     'tis his, and has been slave to thousands; fnord
        But he that filches from me my good name
Robs me fnord of     that which not enriches him,
And    makes me poor indeed.
```

   the output stream should be this:

```
12345678901234567890123456789012345
Who steals my purse steals trash;
'tis something, nothing; 'Twas
mine, 'tis his, and has been slave
to thousands; fnord But he that
filches from me my good name Robs
me fnord of that which not enriches
him, And makes me poor indeed.
```

   If any token is more than `N` characters long, print only the first `N` characters and discard the rest. This approach is called *ragged right* justification, because the text lines up perfectly on the left, but is uneven on the right.

4. (This one is harder.) Extend Q3 so that each output line is both right and left justified. That is, the last character of each line of output should be in column `N`. To do this, you will have to insert extra spaces between the tokens, starting with the gap between the first two tokens in the line. Make the distribution of extra spaces as even as possible; if you have J gaps in the current line but you have K more spaces to distribute somehow (assuming $1 < K < J$), then add one more space to each of the first K gaps. Note that this is the only approach that makes you put extra spaces between tokens.

   For example, the input from Q3 should produce this output:

```
12345678901234567890123456789012345
Who  steals  my purse steals trash;
'tis   something,   nothing;  'Twas
mine,  'tis his, and has been slave
to  thousands;  fnord  But  he that
filches  from  me my good name Robs
me fnord of that which not enriches
him,  And  makes  me  poor  indeed.
```

   Special case: If no more than one token will "fit" onto a given output line (*e.g.,* two consecutive tokens of length `N/2`), then print that token starting in the first column, and add as many trailing spaces as necessary to fill `N` columns in total.

5. (This one is harder also.) Extend Q4 so that you read in a positive integer `N` followed by two character strings `c1` and `c2`, followed by a sequence of tokens. Each line of output should be padded (or truncated, if need be) to `N` characters.

   The meaning of `c1` is:
   - `rr` Ragged right processing (*i.e.,* align tokens as in Q3)
   - `j` Right and left justify (*i.e.,* align tokens as in Q4)
   - `rl` Ragged left processing (see below)
   - `c` Centre the lines (see below)

   The meaning of `c2` is:
   - `f` Print all of the lines you have built up in the order in which they were read in
   - `r` Print all of the lines you have built up in reverse order
   - `g` Print only lines that contain the string `fnord` anywhere, in the order in which they were read in

   If `c1` or `c2` are any other string value, print this message to `cerr`:

   ```
   Error, command is illegal.
   ```

   Ragged left is like ragged right, except that the right hand columns line up and the text is padded on the left, as below:

   ```
   12345678901234567890123456789012345
     Who steals my purse steals trash;
        'tis something, nothing; 'Twas
    mine, 'tis his, and has been slave
       to thousands; fnord But he that
     filches from me my good name Robs
   me fnord of that which not enriches
        him, And makes me poor indeed.
   ```

   Centred text is padded on both the right and left. If there are an odd number of spaces to pad, put the extra space in the front.

   ```
   12345678901234567890123456789012345
    Who steals my purse steals trash;
      'tis something, nothing; 'Twas
    mine, 'tis his, and has been slave
     to thousands; fnord But he that
    filches from me my good name Robs
   me fnord of that which not enriches
      him, And makes me poor indeed.
   ```

   Note that printing lines in reverse order does not change the lines themselves; so this is the correct output for the commands `"rl r"`.

   ```
        him, And makes me poor indeed.
   me fnord of that which not enriches
     filches from me my good name Robs
       to thousands; fnord But he that
    mine, 'tis his, and has been slave
        'tis something, nothing; 'Twas
     Who steals my purse steals trash;
   12345678901234567890123456789012345
   ```