Assignment #2

Due dates:
Part 1: Monday, 30 January 2023, 7:59 pm
Part 2: Friday, 3 Feburary 2023, 7:59 pm

- *Assignment files are available in the usual place using* `git`.
- *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`, `fstream`, `string`, `vector`, `cassert`.
- *For part 2, you are to put all your work in a single C++ solution file named* `a2p2.cc`*; we've already created a stub for you with the signatures of the functions you need to create. Please don't create separate* `.cc` *files for each individual question.*
- *There are 4 Marmoset projects for this assignment. They are named* `a2p1t`, `a2p1c`, `a2p2t` *and* `a2p2t`.
  - `a2p1t` *and* `a2p2t` *are where you will be submitting your tests.*
  - `a2p1c` *and* `a2p2c` *are where you will be submitting your code (solutions).*

  Your tests will be graded on both ***coverage*** *and* ***correctness****. We highly encourage you to write your test cases* ***before*** *you write your solutions. Test coverage will be assessed by running your tests* ***on the staff code****. For full marks your tests only need 90% coverage on the staff solution, but 100% coverage is possible.*
- *The file* `README.md` *in* `git` *contains details regarding the provided scripts as well as how to submit to Marmoset.*
- ***Valgrind will not be assessed on this assignment.***
- *For each question, 90% of your grade is your code/solution and 10% is your tests. This means your final grade for this assignment will be:*

$$(\texttt{a2p1c\_grade} \times 0.9 + \texttt{a2p1t\_grade} \times 0.1) \times 0.5 + (\texttt{a2p2c\_grade} \times 0.9 + \texttt{a2p2t\_grade} \times 0.1) \times 0.5$$

Part I: Text Processing - 50%

We're going to re-organize the logic of Assignment #1 to make it more like a real command-line Unix program. Write a program that reads in a positive integer `N`, plus a character string `textFileName`, followed by a sequence of commands using token-oriented input from `cin`. `N` is the line length, as before. `textFileName` should be the name of a file in the current directory in which you are working that contains the text you are to process.

After you read in `N` and `textFileName` from `cin`, you should then process the text as in Q3–5 of Assignment #1 and store it in an appropriate data structure. Note that you will need to use *file-based* token-oriented input (as discussed in class) to read in the text.

Then you will read in a sequence of commands (which are string tokens) from `cin`. You should continue to process commands until either EOF is detected in `cin` or the `q` (for "quit") command is encountered. The commands are as follows:

`rr`	Switch justification mode to ragged right (the default)
`rl`	Switch justification mode to ragged left
`c`	Switch justification mode to centred
`j`	Switch justification mode to right and left justified
`f`	Change the print direction mode to forward (the default)
`r`	Change the print direction mode to reverse
`p`	Print all of the lines (using the current justification mode and print direction)
`k <k>`	Print the k^{th} line of text you have built up (if there is one; if not, print nothing)
`s <s>`	Print only lines that contain the specified string `<s>` anywhere
`q`	Quit gracefully

Justification mode `j` should cause *all* lines to printed as right and left justified, *including the final line.* Commands `p`, `k`, and `s` should print their line(s) according to the current justification scheme (which is ragged right by default) and print direction (forward, by default). In the above, `<k>` represents an actual integer and `<s>` represents an arbitrary character string; so if you notice that the command is `k` (or `s`) then read the next token into an integer (or string) variable. Valid values of `<k>` are between 0 and $M-1$, where M is the total number of lines you end up with. If the print direction is forward, then `k 0` prints the first line, and `k M-1` prints the last line; if the print direction mode is reverse, then `k 0` prints the *last* line and `k M-1` prints the first.

Note that in a single session, a user may print the lines, then change the print direction and justification mode and print them again, then print only the k^{th} line, then change direction and justification mode again, and print the results one more time. This means that you have to put some thought into just when you perform the justification, and what you store where. If `N` is less than one, then print this error message to `cerr` and quit:

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

If the specified file name can't be found, print this error message to `cerr` and quit:

```
Error, cannot open specified text file.
```

If `command` is any other string, print this message to `cerr` and quit:

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

Part II: Linked List - 50%

For each of the following questions, use this definition for `Node` (as presented in class):

```
struct Node{
    std::string val;
    Node* next;
};
```

1. Write a C++ function called `makeList` that takes a vector of strings and returns a pointer to a list of those strings in the order in which they were read in. Make sure the `next` pointer of the last element is `nullptr`. The signature of the function should look like this:[1]

   ```
   Node* makeList (const std::vector<std::string> v) { ... }
   ```

2. Write a C++ function called `list2string` that takes an existing list (i.e., a pointer to `Node`), and produce a string `s` of the `val` fields in order (`N->val` is before `N->next->val`), separated by exactly one space; refer to `a2p2Test.cc` for an example. You may assume that the last element's `next` pointer is `nullptr`. The signature of the function should look like this:

   ```
   std::string list2string (const Node* first) { ... }
   ```

3. Write a C++ function called `pair2sortedString` that takes two `Node` pointers and produce a string `s` of the `val` fields in lexicographical order (use the normal "<" operator to compare string values), separated by exactly one space. (The values of the `next` pointers aren't relevant for this function; refer to `a2p2Test.cc` for an example.) You should `assert` at the beginning of the function that neither pointer is `nullptr`. The signature of the function should look like this:

   ```
   std::string pair2sortedString (const Node* p1, const Node* p2) { ... }
   ```

4. Write a C++ function called `sortPair` that takes two `Node` pointers and returns a pointer to a list of the two elements sorted lexicographically. You should `assert` at the beginning of the function that neither pointer is `nullptr`. Make sure the `next` pointer of the last element is `nullptr`. (The *incoming* values of the `next` pointers aren't relevant for this function.) You should not create any new `Node`s, just reset pointers appropriately. The signature of the function should look like this:

   ```
   Node* sortPair (Node* p1, Node* p2) { ... }
   ```

[1] Note that `v` should really be a reference parameter, but we haven't covered that topic yet.

5. Write a C++ function called `makeSortedPairList` that takes two strings and returns a pointer to a lexicographically sorted list of the two strings. Make sure the `next` pointer of the last element is `nullptr`. The signature of the function should look like this:

```
Node* makeSortedPairList (const std::string s1, const std::string s2) { ... }
```

6. Write a C++ function called `append` that takes two pointers that each point to the first element of a list of `string`s, call them `p1` and `p2`, and returns a pointer to the list that results from appending `p2`'s list onto the end of `p1`'s list. For example, if the first list consists (in order) of the strings `alpha baker charlie` and the second list consists of `delta echo`, then the function should return a pointer to the list whose elements are (in order) `alpha baker charlie delta echo`. You should not create any new `Node`s, just reset pointers appropriately. The signature of the function should look like this:

```
Node* append (Node* p1, Node* p2) { ... }
```