CS 132, Winter 2022
Programming Project #1: Mad Libs (20 points)
Due: Thursday, January 13, 2022, 11:59 PM

This assignment focuses on strings and file I/O. Turn in a files `lib132.cpp`, `madlibs.cpp` and `mymadlib.txt`.

"Mad Libs" are short stories that have blanks called *placeholders* to be filled in. In the non-computerized version of the game, one person asks a second person to fill in each of the placeholders without the second person knowing the overall story. (For example, "`Tell me a noun: ___`") Once all placeholders are filled in, the second person is shown the resulting story, often with humorous results.

In this assignment you will present a menu to the user with three options: create a new mad lib, view a previously created mad lib, or quit. The user will then type a command. Your program should respond to the following three commands: `create`, `view`, or `quit`, case-insensitively. If any other command is typed, the user should be re-prompted.

When creating a new mad lib, your program should prompt the user for input and output file names. Then the program should read the input file and prompt the user to fill in any placeholders that are found without showing the user the rest of the story. As the user fills in each placeholder, the program should write the resulting text to the output file. The user can later view the mad lib that was created or quit the program. The log below shows one sample execution:

```
Welcome to the game of Mad Libs.
I will ask you to provide several words
and phrases to fill in a mad lib story.
The result will be written to an output file.

Create, view or quit? create
Input file name: oops.txt
File not found. Try again: TORZON.txt
File not found. Try again: tarzan.txt
Output file name: out1.txt

Please type an adjective: silly
Please type a plural noun: apples
Please type a noun: frisbee
Please type an adjective: hungry
Please type a place: Tacoma, WA
Please type a plural noun: bees
Please type a noun: umbrella
Please type a funny noise: burp
Please type an adjective: shiny
Please type a noun: jelly donut
Please type an adjective: beautiful
Please type a plural noun: spoons
Please type a person's name: Keanu Reeves
Your mad-lib story has been created!

Create, view or quit? X
Create, view or quit? I don't understand.
Create, view or quit? view
Input file name: OUT00I.txt
File not found. Try again: i forget the file name
File not found. Try again: out1.txt

One of the most silly characters in fiction is named
"Tarzan of the apples ." Tarzan was raised by a/an
frisbee and lives in the hungry jungle in the
heart of darkest Tacoma, WA . He spends most of his time
eating bees and swinging from tree to umbrella .
Whenever he gets angry, he beats on his chest and says,
" burp !" This is his war cry. Tarzan always dresses in
shiny shorts made from the skin of a/an jelly donut
and his best friend is a/an beautiful chimpanzee named
Cheetah. He is supposed to be able to speak to elephants and
spoons . In the movies, Tarzan is played by Keanu Reeves .

Create, view or quit?  QUIT
```

Notice that if an input file is not found, either for creating a mad lib or viewing an existing one, the user is re-prompted. No re-prompting occurs for the output file. If it does not already exist, it is created. If it already exists, overwrite its con-

tents. (These are the default behaviors of `ofstream`.) You may assume that the output file is not the same file as the input file.

Input Data Files:

Download the example input files from the class web site and save them in the same folder as your program. Mad lib input files are mostly just plain text, but they may also contain placeholders. Placeholders are represented as input tokens that begin and end with < and > . For example, the file `tarzan.txt` used in the previous log contains the following text. Placeholders are colored for emphasis:

```
One of the most <adjective> characters in fiction is named
"Tarzan of the <plural noun>." Tarzan was raised by an old
<noun> and lives in the <adjective> jungle in the
heart of darkest <place>. He spends most of his time
eating <plural noun> and swinging from tree to <noun>.
Whenever he gets angry, he beats on his chest and says,
"<funny noise>!" This is his war cry. Tarzan always dresses in
<adjective> shorts made from the skin of his <noun>
and his best friend is the <adjective> chimpanzee named
Cheetah. He is supposed to be able to speak to elephants and
<plural noun>. In the movies, Tarzan is played by <person's name>.
```

Contents of input file `tarzan.txt`

Your program should break the input into lines so that you can look for all its placeholders. Each placeholder token in the document causes the user to be prompted. The user's response to the prompt is inserted into the eventual output rather than the placeholder itself. You should accept whatever response the user gives, even a multi-word answer or blank answer. Your output Mad Lib story must retain the original spacing and placement of line breaks from the input story file. If the input file contains no placeholders, just output its contents.

When prompting the user to fill in a placeholder, give a different prompt depending on whether the placeholder begins with a vowel (a, e, i, o, or u, case-insensitively). If so, prompt for a response using "an". If not, use "a". For example, **<noun>** leads to "Please type **a** noun" while **<adjective>** leads to "Please type **an** adjective".

You may assume that a placeholder will appear entirely on a single line; no placeholder will ever span across multiple lines. A given line could have < or > characters in it that are not part of placeholders, and these should be retained and included in the output. You may assume that a placeholder token will contain at least one character between its < and > (in other words, no file will contain the token <>).

"View mad lib" simply means to read the given text file and print its contents to the console. You do not need to do any processing on the file or any kind of testing on its contents; just output the file's entire contents to the console. The line breaks in the original file should be preserved in the output.

Your program's menu should work properly regardless of the order or number of times its commands are chosen. For example, the user should be able to run each command (such as `create` or `view`) many times if so desired. The user should also be able to run the program again later and choose the `view` option without first choosing the `create` option on that run. The user should be able to run the program and immediately quit with the `quit` option if so desired.

Creative Aspect (`mymadlib.txt`):

Along with your program, submit a file `mymadlib.txt` with a story of your own creation, in a format suitable for use as input to your program. Look at the sample input files on the web site as examples. For full credit, your story should be at least 8 lines long and contain at least five (5) placeholders.

Library File (`lib132.cpp`):

Throughout the quarter, put all functions that you wish were built into C++, as they are generally useful, into `lib132.cpp`. You may include code from lecture, lab, small problems and other code you write yourself. **You may not include code from the internet or that other people have written**. For example, you may want to include `toLowerCase` from your small problems and `equalsIgnoreCase` from lecture. You can use any of these functions in projects. `lib132.cpp` must adhere to the same style standards as the rest of your project.

Implementation Details:

Collections: You are **forbidden from using data structures** (e.g. array, vector, map, etc.) on this program. They are not needed, and one of the goals of this project is for you to practice manipulating strings and file streams.

I/O: Your program has a **console**-based user interface. Produce console output using **`cout`** and read console input using **`cin`** and **`getline`**.

You will also write code for reading **input files** and outputting to **output files**. Read a file using an **`ifstream`** object, along with functions such as **`getline`** to read lines from the file. Output to a file using an **`ofstream`** object. Make sure to **close** your file streams when done reading or writing.

A non-trivial part of the program involves **string manipulation**. You may want to look up members of the C++ string class such as **`find`**, **`length`**, **`substr`**, and so on.

You must use **C++** facilities (**`cout`**, **`ifstream`**, **`ofstream`**, **`string`**) instead of C equivalents (**`printf`**, **`fopen`**, **`char*`**).

Style Guidelines:

External correctness: Your output must match the sample outputs exactly. Use the **Output Comparison** tool linked from the project page of the course website to make sure of this.

Compiling: Your code should compile without any errors or warnings and should work with any C++ compiler and input files as given. You may not use any compiler-specific functions.

Style: Half of your grade will be based on the style of your code. There are many general C++ coding styles that you should follow. Use whitespace and indentation properly. Limit lines to 100 characters. Give meaningful names to functions/variables and follow C++'s naming standards. Localize variables. You may only use material covered in lecture. You must also follow the following guidelines:

Procedural decomposition: Your `main` function should represent a concise summary of the overall program. It is okay for `main` to contain some code such as console output statements to `cout`. But `main` should not perform too large a share of the overall work itself directly, such as reading the lines of the input file. Instead, it should make calls to other functions to help it achieve the overall goal.

Each function should perform a single clear and coherent task. No one function should do too large a share of the overall work. As a rough estimate, a function whose body (excluding the header and closing brace) has more than 30 lines is likely too large. You should avoid "chaining" long sequences of function calls together without coming back to `main`. Your functions should also be used to help you avoid **redundant code**. If you are performing identical or very similar commands repeatedly, factor out the common code and logic into a helper function, or otherwise remove the redundancy. Use parameters and returns to get data where you need it.

Variables and types: Use descriptive variable and function names. Use appropriate data types for each variable and parameter; for example, do not use a `double` if the variable is intended to hold an integer, and do not use an `int` if the variable is storing a true/false state that would be better suited to a `bool`. **Do not declare any global variables**; every variable in your program must be declared inside one of your functions and must exist in that scope.

Commenting: Your code should have adequate **commenting**. The top of your file should have a descriptive comment header with your name, the assignment name, date, course and a description of what your code does. Each function should have a comment header describing that function's behavior, any **parameters** it accepts and any values it **returns**, and any assumptions the function makes about how it will be used. For larger functions, you should also place a brief inline comment on any complex sections of code to explain what the code is doing.