Anatomy of a README File

Some assignments in CS70 require that you submit a README file, and some do not. This handout outlines the information that should be included in a README file when one is required. Note, however, that much of this information is required for *every* assignment—even when there is no README file required, you should still provide enough documentation for someone else to thoroughly understand the algorithms you have used and their limitations. When no README file is required, you may submit a README file anyway, or include such documentation as comments in your source files.

A README file is a freestanding document that provides a detailed overview of your code. Its purpose is to document your program both for potential users and for fellow programmers who will be examining your code. Your README file must be plain Ascii text (not HTML, PDF, or MS-Word format) that is formatted for viewing on an ordinary terminal screen (i.e., no line longer than 80 characters). In your README you must include the following information:

- Your name & email address
- The assignment number
- The purpose of your program (i.e., the problem statement)

  Usually, this section will be a paraphrased version of the homework specifications. (Direct copying is okay if the specification is reasonably short (one or two paragraphs), but you *must* give proper attribution.) Be sure not to confuse the purpose of the *program* with the purpose of the *assignment.* This section should also describe what your program can do.
- The format of your program's input and output
- How your program achieves its results and its general structure, including the
  - Data structures you used
  - Algorithms you used (for the program as a whole and for any interesting components)
  - Name and purpose of each file provided
- Any special restrictions, bugs, or limitations of your program (size of input, behavior with malformed input, etc.)
- Any improvements that can be made

In addition, you should use your README file to inform us about any unusual characteristics or circumstances. You could, for example, draw attention to a particularly elegant solution to an implementation problem, or document known bugs. You can

also use your README file to describe compilation failures or errors, situations that prevented you from completing the assignment, and so forth. In general, fewer points will be deducted for failures documented in the README file than for failures we discover ourselves.

The "general structure" section should describe how the program works and how it is broken up into classes. You should document each class and highlight important/interesting functions and the algorithms they use. You may gloss over small details by not giving full type signatures for functions and not covering insignificant and obvious functions—the person reading your README file will also be able to read the .hpp files for exact interface details (which will be suitably commented). In essence, someone reading your README file should get an overall picture of your code and have enough of a sense of the details to be able to read any single source file and understand what is going on without looking at the other source files.

The "files" section may be omitted if the program is contained in a single file. Otherwise, it should briefly explain how the program is organized into files. Files that "go together" (such as .hpp and .cpp files) should be discussed together. The purpose of this section is to help the reader locate various classes and functions, so the descriptions can usually be quite brief.

Your documentation should not contain more than very brief fragments of code or pseudocode. Concentrate on explaining the main features of your algorithm in English, as if you were talking to someone working on another system in another programming language.

Sometimes, students ask whether some particular item should be included in the README file. The general rule is, "If you're not sure, put it in!" The graders do not normally deduct points for having excessive material in the README, but being too brief may cause you to lose points.

The length of your README file will depend on the complexity of the program it describes. Simple programs need only short explanations whereas complex programs require more detail. In general, README files for programs later in the semester will be longer than those for earlier programs.

To help you understand what a good README file looks like, an example README file is provided below. Your README file need not look exactly like this one provided that it meets the requirements.

An Example README File

```
Name:       Melissa O'Neill
Email:      oneill@cs.hmc.edu
Assignment: 1

Purpose of Program
------------------

The shuffle program reads a sequence of lines from the user, terminated by
an empty line, and then prints out those lines in random order.
```

```
I/O Format
----------

After presenting the prompt

     Enter lines to shuffle, ending with a blank line

lines of text are read from standard input until an empty line is read.
(In the context of this program, a "blank line" means an empty line -- a
line consisting of only spaces is not considered blank.) No special
formatting of the lines is required. If EOF is reached before a blank line
is read, the program considers this input to be the same as a final empty
line.

The program outputs each line in random order.

General Structure
-----------------

The program is implemented by two classes (LineShuffler and Random) and
a main driver function.

Overall Algorithm

    The main driver function reads lines from the input stream and inserts
    them into a lineshuffler object, and then, once all the input has been
    read, draws lines from the lineshuffler and outputs them until no
    lines are left.

Data Structures

    The program as a whole relies on a single lineshuffler object.

LineShuffler class

    This class handles storing lines and retrieving them in random order.
    It provides three important member functions for accessing the stored
    lines:

        isEmpty()
            returns true when there are no lines stored

        addLine(line)
            adds a line to the collection of lines stored

        removeLine()
            removes a random line from those stored

    Data Representation

        Internally, the lines are stored in a vector of strings. In
        addition the class stores a Random object from which it
        obtains its own supply of random numbers.
```

```
    Algorithms

        addLine adds to the end of the vector (relying on the vector class
        to do all the hard work). removeLine removes a random line from the
        vector by picking a random line and removing it from the middle of
        the array and then moving the other lines up (preserving their
        order). Note that removeLine's algorithm is inefficient. [A more
        efficient approach would be to swap the line in the middle with the
        last line in the vector and then remove that last line. This
        approach would not preserve the order of the vector, but that can
        be seen as a good thing because we are trying to shuffle the
        lines.]

Random class

    This class generates random numbers. It provides the following
    important operations:

        Random()
            construct a random-number generator, using the current time as
            the seed

        next()
            provide a random number in the range [0..MAX] (where MAX is a
            class defined constant (in this case 32768)

        next(max)
            provide a random number in the range [0..max]

    Data Representation

        Internally, the random class stores the state of the random number
        generator using a single integer value.

    Algorithms & Limitations

        The random number generation technique is used in this class is
        based on example code provided in the ISO C specification. It
        written to be as quick and simple as possible, rather than to
        provide particularly good random numbers.

Files

    The following files implement the above classes and functions:

    shuffle.cpp       -  contains main, the main driver function
    lineshuffler.hpp  -  declares the interface for the LineShuffler class
    lineshuffler.cpp  -  defines the implementation of the LineShuffler class
    random.hpp        -  declares the interface for the Random class
    random.cpp        -  defines the implementation of the Random class
```

```
Limitations and Possible Improvements
-------------------------------------

The program only operates correctly for up to 32,768 lines, due to the
range of the random-number generator. With a better random-number
generator, the number of lines this program can process would largely be
limited by available memory and user patience (given the inefficient
algorithm described above). Beyond those practical limits, the number of
lines is also limited by the range of the int type.

The quality of the shuffle also depends on the quality of the random
numbers generated by the Random class.
```