Problem #2: Patch Generation with Delta Encoding (Simplified Diff)

Delta encoding or delta compression is a form of representing data as differences between multiple data sets, or one set of data sequentially. Depending on the data sets, it can be optimal to delta encode for memory usage or file-size purposes. See more at Google: Delta Encoding.

Implement a program, simple_delta, in C (C99) or C++14 that will generate a "patch" representing a delta difference between two files. It should both compile and execute on a desired Linux flavor with an open-source toolchain, such as gcc, g++, etc. It should receive two command-line arguments, one for each file to compare. The goal is to generate a delta "patch" on stdout such that it could apply to an original file to match the contents of a new file.

Example:

```
$ cat original_file
abc
$ cat new_file
abcd
$ ./simple_delta original_file new_file
d
```

In this example, the character delta difference in hexadecimal for the two files is: 0x00000064. A delta difference between one character and a non-existent character, in this case 'd' with none, should result in a difference of the character. For simplicity, do not convert any delta values to ASCII or vice versa when generating differences. In this example, 0x00 should remain zero and not convert to ASCII zero.

Assumptions and program behavior:

- Assume ASCII text files as input
- Assume input text files of any size
- Assume file termination characters exist for input text files
- Assume single byte delta encoding for all differences will suffice
- Ignore any white space (e.g., comparing character sequences "abc" and "a b c" would be equivalent to comparing "abc" and "abc")
- The delta difference, if any, should output to the stdout stream
- Routines from standard libraries that will make life easier can be used. This does not include anything to assist specifically with delta encoding
- Notify the user of any errors (e.g., file does not exist) and exit gracefully

Please provide a Makefile or script to build your source and unit tests. Ensure your code is well documented and you have stated your assumptions in the comments. Program usage should print if incorrect arguments are provided.