Assignment 7
First version due Friday, March 14
Final version due Friday, April 11

A text editor is used to create and modify a text file. Emacs and pico and vi are examples of full screen editors: they display a full screen of information and let you move around the whole screen, making modifications wherever you wish. A line editor allows you to modify only one line, or a group of lines in the same way, though you can display a full screen of text. A line editor is not as easy to use as a screen editor, but it is much easier to write a line editor. For this assignment you will write a simple line editor.

The project is due in two parts. For the first part, finish the skeleton `text_editor.cc` I have provided, and the functions in `document.cc` that are required to make it run.

Your editor should recognize and properly execute the following commands, which are typed at the keyboard.

g ⟨*name*⟩	Get an existing file and load it into the editor. (The angle brackets are not typed as part of the command.)
n	Turn line numbering on if off and off if on (line numbering on means that the number of each line is displayed at the left when the t command is given). Line numbering starts at 1.
q	Leave the editor without saving the text.
x [*name*]	Leave the editor and save the text in a file. The brackets indicate that *name* is optional; if it is left out, the current file name is used. If a name is provided, it becomes the new current file name.
s [*name*]	Save the file under the current name if a new name is not provided, otherwise change the current name to *name* and save.
h	Print a screen summarizing the available editor commands.
a [N]	Start adding text after line N and before line N + 1. Pressing return starts a new line, typing a control-D signals the end of add mode. If N is zero or missing, add lines at the beginning of the file.
t [M [N]]	Display lines M through N on the screen. If M is given but not N, display only line M; if neither is given, display the whole document.
d M [N]	Delete lines M through N (just line M if N is not given) and move the remaining lines up to fill the gap.
m M [N] L	Move lines M through N so that they come immediately after line L (and they no longer appear at their original location). This command only makes sense for L less than M or L bigger than N. Close the gap left when the lines are moved. When N is missing, move just line M.
c M [N] L	Copy lines M through N so that they come immediately after line L (but they still appear at their original location). Note that this makes sense even for L between M and N. When N is missing, copy just line M.
f [M [N]] /*word*/	Display each line between M and N on which *word* appears. The slash characters are part of the command, not part of the string to be found.
r [M [N]] /*old*/*new*/	Replace each occurrence of string *old* with string. Optional parameters are as in the t command. *new* on each of lines M through N. The slash characters are part of the command, not part of the strings. Optional parameters are as in the t command.

Your finished program also should do as good a job as possible at catching errors so that the program doesn't crash. For example, typing a command that doesn't exist, or entering a number

for M or N or L that doesn't make sense, should be handled gracefully. This is very important for an editor because if the program crashes all unsaved changes are lost.

Copy the header file for the class, a starting point for the class functions, a skeleton main program, and my version of the program, from `~guichard/167/assignment7/` as usual. You must use the public section as is; you may modify the private section.

When reading from the keyboard, I strongly suggest you use the `getline` function for **all** input, **not** "`cin >>`". To read multiple lines, which you will need for the `a` command, use something like this:

```
while (getline(cin,nextline)) {
...
}
cin.clear();
```

When using `getline` to read from a file, the loop stops when the end of the file is reached. There is no end of file for `cin`, but you can make C++ think it has reached the end of the file by typing control-D to stop the loop. When you do this, cin is put into an error state, and further uses of getline will not succeed. To allow further keyboard input, the error state must be cleared, using the `clear` function.

Write your program so that it is possible to give an initial file to edit on the commandline:

```
text_editor myfile.txt
```

You will need to extract numerical values from a string, e.g., from the string "c 15 20 30" you will need to get the values 15, 20, and 30 as integers. While there are functions to do this, I think it is easier to do it directly. The code is something like this:

```
for (number = 0; i < s.length() && '0' <= s[i] && s[i] <= '9'; i++) {
  number = number*10 + s[i] - '0';
}
```

If `i` is initially the index of the first digit, this will put the correct value in `number`, and the value of `i` will be either the length of `s` or the postion of the first non-digit character. If you expect another integer in the string, you can skip space characters and then do this again.