

ECS 36C: Programming Assignment #3

Instructor: Aaron Kaloti

Winter Quarter 2022

Contents

1	Changelog	1
2	General Submission Details	2
3	Purpose of This Assignment	2
4	Reference Environment	2
5	Ayayron™: An Editor That No One in Their Right Mind Would Ever Use	2
5.1	Miscellaneous Remarks	2
5.2	Part #1 - Reading the Input File's Contents	3
5.3	Part #2 - The Main Loop of Your Program	3
5.4	Part #3 - Saving the File	4
5.5	Part #4 - Moving the Cursor Left/Right	5
5.6	Part #5 - Moving the Cursor Up/Down	7
5.7	Part #6 - Moving the Cursor Repeatedly	13
5.8	Part #7 - Repeating the Previous Command	15
5.9	Part #8 - Insertion	17
5.10	Part #9 - Edit History	23
5.11	Part #10 - Attempting to Quit with Unsaved Changes	27
5.12	Epilogue: Additional Features	30
6	Grading Breakdown	31
7	Autograder Details	31
7.1	Regarding Timeouts	31

1 Changelog

You should always refer to the latest version of this document.

- v.1: Initial version.
- v.2: To the end of the subsection “Miscellaneous Remarks”, I added information about what you can assume regarding use of whitespace in the commands.
- v.3:
 - Autograder details.
 - Added the compilation command (in the autograder section).
 - You may assume that the undoing of an insertion will never cause the cursor to be out of bounds. (That is, the autograder won't cause something like this.)
- v.4: Deadline pushed back to the night of Friday, 02/18.

*This content is protected and may not be shared, uploaded, or distributed.

2 General Submission Details

Partnering on this assignment is prohibited. If you have not already, you should read the section on academic misconduct in the syllabus.

This assignment is due the night of Friday, 02/18. Gradescope will say 12:30 AM on Saturday, 02/19, due to the “grace period” (as described in the syllabus). *Be careful about relying on the grace period for extra time; this could be risky.*

3 Purpose of This Assignment

- To show you why the [ECS 36C course description](#) says, “Extensive programming.”
- To give you experience with a project that takes *at least* 350-450 lines of code and that really tests your problem-solving skills and ability to handle a lot of details/specifications (like any real-world project would have).
- To give you a practical example of a reason to use a stack.

4 Reference Environment

The autograder, which is talked about at the end of this document, will compile and run your code in a Linux environment. That means that you should make sure your code compiles and behaves properly in a sufficiently similar environment. **The CSIF is one such environment.** Each student can remotely access the CSIF, and now that learning is back in person, the CSIF computers can be physically accessed in the Kemper basement. I talk more about the CSIF in the syllabus.

Do not assume that because your code compiles on an *insufficiently* similar environment (e.g. directly on your Mac laptop), it will compile on the Linux environment used by the CSIF.

You should avoid causes of undefined behavior in your code, such as uninitialized variables. If you have things like this in your code, then your code could end up generating the wrong answer when autograded, even if it generated the right answer when you tested it in a sufficiently similar environment.

5 Ayayron™: An Editor That No One in Their Right Mind Would Ever Use

Filename: `editor.cpp`

In this assignment, you will implement a pretty terrible text editor in a C++ file called `editor.cpp`. This is the only file that you will submit.

Although the parts below come in a certain order, you don’t have to do them in the exact order shown. Needless to say, some parts have to be done (or at the very least, *should* be done) before others; for example, it wouldn’t make much sense to implement the window scrolling before you implement the reading of the input file’s contents. I would encourage you to read this entire document before starting, because it is possible that some of the later parts could affect how you do the earlier parts. That is, you do not want to run into a situation in which, in a later part, you have to rewrite code you wrote in an earlier part.

5.1 Miscellaneous Remarks

Don’t forget that your output must match mine *exactly*. The only exception is trailing whitespace, which the autograder removes.

As with the cursor list assignment, it will still be possible to get credit even if you do not finish the entire assignment. Moreover, not all parts depend on the parts before it. What this means is that, for example, if you cannot get downward scrolling to completely work, you could still get some of the test cases for undo/redo correct. The order of the parts is not the only order in which you can do them.

You will be penalized if your program has any memory leaks or fails to close any file. However, these should be easy things to avoid if you use the typical C++ features/libraries (as opposed to the C ones) that follow RAII principles.

You do not need to worry about the user entering invalid inputs. For instance, later on, you will support the ability for the user to enter ‘r’ in order to trigger a redo operation. You do not need to worry about the user/autograder trying to trick your code with inputs like “ribbon” or “r o c k”. The autograder will not assess your program’s response to invalid inputs. However, in my own implementation, I made sure my program didn’t die/crash in the event of an invalid input, because I sometimes accidentally entered an invalid input while trying to test my code.

For any input that involves an initial character or word (e.g. `s 3, w 4, save output.txt`), you may assume that there will always be one space between the initial character/word and whatever comes after. For instance, you will never see the input `w 4`. This does not apply to spaces in what is inserted during an insertion operation. For instance, `i A B` is valid, even though there are five spaces between the ‘A’ and the ‘B’.

5.2 Part #1 - Reading the Input File's Contents

Your editor will be passed the name of the file to read as a command-line argument.

If your program is given too few or too many command-line arguments, then it should print out an appropriate message and return the appropriate exit code, as shown below. *Everything your program prints – including error messages like the below – should be printed to standard output, NOT standard error. You should not print to standard error at any point in this assignment.*

```
1 $ ./editor
2 Too few command-line arguments.
3 $ echo $?
4 1
5 $ ./editor a b
6 Too many command-line arguments.
7 $ echo $?
8 1
```

Your editor should read in the contents of the file. You will need to decide how best to store these contents, and that decision could be influenced by later parts. Since a file's contents are better read line-by-line rather than word-by-word, you may find `std::getline()` more useful than `>>` for reading from the file. Be careful about using *both* `std::getline()` and `>>` on the same stream, whether a file stream or the standard input stream (`std::cin`). As an example of what can go wrong if you use both of them: if you reach the end of a line with `>>` and then use `std::getline()` once, the latter will read an empty string, not the next line.

If any of the below occur, the editor should print an appropriate message and terminate, returning the appropriate exit code.

- The file cannot be opened. (Do not worry about the specific reason, e.g. the file does not exist, insufficient permissions, etc.)
- The file has more than 30 lines.
- At least one line in the file is more than 20 characters long. (The implicit newline character at the end of each line in the text file does not count.)

Below are examples showing how your program should react in the above situations. Note that you do not need to store the input files in a folder called `example_input_files`; I did that as part of my own file organization.

```
1 $ ./editor nonexistent_file
2 Failed to open file: nonexistent_file
3 $ echo $?
4 2
5 $ wc -l example_input_files/too_many_lines.txt
6 31 example_input_files/too_many_lines.txt
7 $ ./editor example_input_files/too_many_lines.txt
8 File example_input_files/too_many_lines.txt has too many lines.
9 $ echo $?
10 2
11 $ cat example_input_files/wide_line.txt
12 ab
13 cd
14 123456789012345678901
15 ef
16 gh
17 $ ./editor example_input_files/wide_line.txt
18 File example_input_files/wide_line.txt has at least one too long line.
19 $ echo $?
20 2
```

The input file will never be empty.

5.3 Part #2 - The Main Loop of Your Program

The main loop of your program should do the following:

- Display the current buffer contents¹.
- Prompt the user for a command.
- React to the command.

¹I don't want to say "current file contents" because, if a modification/insertion is made and a save operation is not done, then the output file's contents will be out-of-sync with what the editor shows.

Whenever the edited file's contents are displayed, it will be within a window that is 20 characters wide and 10 characters tall. Through this window, we see only a part of the current buffer contents at any given time. Initially, the window should start at line #1. In a later part, you will allow the window to be scrolled up and down.

The only command that your program should support at this point is quitting by entering 'q'. (You can see this being entered after "Enter command: " is printed in the below.) This should result in the printing of a message (see below) and the termination of the program.

Below are two examples of how your program should behave.

```
1 $ wc -l example_input_files/input1.txt
2 27 example_input_files/input1.txt
3 $ ./editor example_input_files/input1.txt
4      *
5      12345678901234567890
6 * 1|abcdef
7   2|ghi
8   3|jklm
9   4|jkj
10  5|jklasj
11  6|jklajs;lkfj
12  7|qiowuioj
13  8|lkjw;lkj
14  9|qwklejklj;c
15 10|jkaljsoiu
16      12345678901234567890
17 Enter command: q
18
19 Goodbye!
20 $ wc -l example_input_files/small.txt
21 5 example_input_files/small.txt
22 $ ./editor example_input_files/small.txt
23      *
24      12345678901234567890
25 * 1|Line 1
26   2|Line 2
27   3|Line 3
28   4|Line 4
29   5|Line 5
30   6
31   7
32   8
33   9
34  10
35      12345678901234567890
36 Enter command: q
37
38 Goodbye!
39 $ echo $?
40 0
```

Just so it's clear, there are two whitespaces between the asterisk and the number 1, in the case of the asterisk on the left column. If the asterisk were on line number 10, then there would be one whitespace between the asterisk and the number 10. In the second example, the vertical bar is not printed after the line number for line #6 onwards because `small.txt` does not have more than five lines.

Line numbers, as well as rows containing 12345678901234567890, are printed above and below the window's contents, for convenience. (It is sometimes useful to know which column you are on in an editor, even if that editor is terrible.) You can also see two asterisks in the above output as well. Those asterisks mark the current location of the cursor. In later parts, you will allow the cursor to be moved and allow the user to make insertions starting at the current location of the editor.

5.4 Part #3 - Saving the File

If the user enters "save", you may assume it will be followed by a filename, and your program should save the current buffer contents to that file. You may assume that there will never be issues (e.g. permission issues) with opening/creating the file to write to.

Below is an example of how your program should behave.

```
1 $ ./editor example_input_files/small.txt
2      *
3      12345678901234567890
4 * 1|Line 1
5   2|Line 2
```

```

6      3|Line 3
7      4|Line 4
8      5|Line 5
9      6
10     7
11     8
12     9
13    10
14      12345678901234567890
15 Enter command: save aaron_is_cool.txt
16
17     *
18      12345678901234567890
19 *   1|Line 1
20     2|Line 2
21     3|Line 3
22     4|Line 4
23     5|Line 5
24     6
25     7
26     8
27     9
28    10
29      12345678901234567890
30 Enter command: q
31
32 Goodbye!
33 $ cat aaron_is_cool.txt
34 Line 1
35 Line 2
36 Line 3
37 Line 4
38 Line 5

```

5.5 Part #4 - Moving the Cursor Left/Right

If the user types ‘a’ or ‘d’, then – assuming it would not cause out-of-bounds cursor movement – the cursor should be moved left or right, respectively². Below are examples.

```

1 $ ./editor example_input_files/input2.txt
2     *
3      12345678901234567890
4 *   1|abcdef
5     2|ab
6     3|
7     4|xyz
8     5|Hi there
9     6|Hi how are you
10    7|I am good
11    8|how are you
12    9|blah
13   10|blah blah blah blah
14      12345678901234567890
15 Enter command: d
16
17     *
18      12345678901234567890
19 *   1|abcdef
20     2|ab
21     3|
22     4|xyz
23     5|Hi there
24     6|Hi how are you
25     7|I am good
26     8|how are you
27     9|blah
28   10|blah blah blah blah
29      12345678901234567890
30 Enter command: a
31
32     *

```

²We’re going with WASD for cursor movement here. Sorry if you prefer a DVORAK keyboard layout.

```

33      12345678901234567890
34 * 1|abcdef
35   2|ab
36   3|
37   4|xyz
38   5|Hi there
39   6|Hi how are you
40   7|I am good
41   8|how are you
42   9|blah
43  10|blah blah blah blah
44      12345678901234567890
45 Enter command: a
46
47      *
48      12345678901234567890
49 * 1|abcdef
50   2|ab
51   3|
52   4|xyz
53   5|Hi there
54   6|Hi how are you
55   7|I am good
56   8|how are you
57   9|blah
58  10|blah blah blah blah
59      12345678901234567890
60 Enter command: d
61
62      *
63      12345678901234567890
64 * 1|abcdef
65   2|ab
66   3|
67   4|xyz
68   5|Hi there
69   6|Hi how are you
70   7|I am good
71   8|how are you
72   9|blah
73  10|blah blah blah blah
74      12345678901234567890
75 Enter command: d
76
77      *
78      12345678901234567890
79 * 1|abcdef
80   2|ab
81   3|
82   4|xyz
83   5|Hi there
84   6|Hi how are you
85   7|I am good
86   8|how are you
87   9|blah
88  10|blah blah blah blah
89      12345678901234567890
90 Enter command: d
91
92
93 ... (I've omitted much of the output here) ...
94
95
96      *
97      12345678901234567890
98 * 1|abcdef
99   2|ab
100  3|
101  4|xyz
102  5|Hi there
103  6|Hi how are you
104  7|I am good
105  8|how are you
106  9|blah

```

```

107 10|blah blah blah blah
108 12345678901234567890
109 Enter command: d
110
111 *
112 12345678901234567890
113 * 1|abcdef
114 2|ab
115 3|
116 4|xyz
117 5|Hi there
118 6|Hi how are you
119 7|I am good
120 8|how are you
121 9|blah
122 10|blah blah blah blah
123 12345678901234567890
124 Enter command: d
125
126 *
127 12345678901234567890
128 * 1|abcdef
129 2|ab
130 3|
131 4|xyz
132 5|Hi there
133 6|Hi how are you
134 7|I am good
135 8|how are you
136 9|blah
137 10|blah blah blah blah
138 12345678901234567890
139 Enter command: d
140
141 *
142 12345678901234567890
143 * 1|abcdef
144 2|ab
145 3|
146 4|xyz
147 5|Hi there
148 6|Hi how are you
149 7|I am good
150 8|how are you
151 9|blah
152 10|blah blah blah blah
153 12345678901234567890
154 Enter command: q
155
156 Goodbye!

```

5.6 Part #5 - Moving the Cursor Up/Down

Attempts at up and down cursor movement should be triggered by ‘w’ and ‘s’, respectively. The cursor cannot be moved to the nonexistent line #0, and it cannot be moved past the last line of the current buffer contents.

For the rest of this part’s directions, we talk about when *scrolling* occurs. Scrolling refers to the movement of the window that lets us see into the current buffer contents. The following rules dictate how scrolling up and down should work. I based these rules off of how mouse scrolling works in Sublime Text.

- **Scrolling up: Scrolling up can only occur when the user tries to move the cursor up while the cursor is at the top of the window.** Moreover, if the cursor is already at the first line, then scrolling cannot occur; otherwise, scrolling should proceed, with the cursor staying at the top of the window.
- **Scrolling down.** Let’s break this down into three scenarios:
 1. If the cursor is *not* at the last line but *is* at the bottom of the window, then scroll down, keeping the cursor at the bottom of the window (which means that the cursor advances one line).
 2. If the cursor is at the last line but *not* at the top of the window, then scroll down but keep the cursor on the same line of the current buffer contents.

3. If the cursor is at the last line of the file *and* at the top of the window, then scrolling down should be prohibited. In other words, although the user can scroll past the end of the file, they cannot cause the last line of the file to completely disappear.

Below are examples of how your program should behave after you have completed this part.

```
1 $ ./editor example_input_files/lines.txt
2      *
3      12345678901234567890
4 *   1|Line 1
5     2|Line 2
6     3|Line 3
7     4|Line 4
8     5|Line 5
9     6|Line 6
10    7|Line 7
11    8|Line 8
12    9|Line 9
13   10|Line 10
14      12345678901234567890
15 Enter command: s
16
17      *
18      12345678901234567890
19      1|Line 1
20 *   2|Line 2
21     3|Line 3
22     4|Line 4
23     5|Line 5
24     6|Line 6
25     7|Line 7
26     8|Line 8
27     9|Line 9
28    10|Line 10
29      12345678901234567890
30 Enter command: w
31
32      *
33      12345678901234567890
34 *   1|Line 1
35     2|Line 2
36     3|Line 3
37     4|Line 4
38     5|Line 5
39     6|Line 6
40     7|Line 7
41     8|Line 8
42     9|Line 9
43    10|Line 10
44      12345678901234567890
45 Enter command: w
46
47      *
48      12345678901234567890
49 *   1|Line 1
50     2|Line 2
51     3|Line 3
52     4|Line 4
53     5|Line 5
54     6|Line 6
55     7|Line 7
56     8|Line 8
57     9|Line 9
58    10|Line 10
59      12345678901234567890
60 Enter command: s
61
62      *
63      12345678901234567890
64      1|Line 1
65 *   2|Line 2
66     3|Line 3
67     4|Line 4
68     5|Line 5
```



```

69      6|Line 6
70      7|Line 7
71      8|Line 8
72      9|Line 9
73     10|Line 10
74      12345678901234567890
75 Enter command: s
76
77      *
78      12345678901234567890
79      1|Line 1
80      2|Line 2
81 *    3|Line 3
82      4|Line 4
83      5|Line 5
84      6|Line 6
85      7|Line 7
86      8|Line 8
87      9|Line 9
88     10|Line 10
89      12345678901234567890
90 Enter command: s
91
92
93 ... (I've omitted much of the output here) ...
94
95
96      *
97      12345678901234567890
98      1|Line 1
99      2|Line 2
100     3|Line 3
101     4|Line 4
102     5|Line 5
103     6|Line 6
104     7|Line 7
105 *    8|Line 8
106     9|Line 9
107    10|Line 10
108     12345678901234567890
109 Enter command: s
110
111      *
112      12345678901234567890
113      1|Line 1
114      2|Line 2
115      3|Line 3
116      4|Line 4
117      5|Line 5
118      6|Line 6
119      7|Line 7
120      8|Line 8
121 *    9|Line 9
122    10|Line 10
123     12345678901234567890
124 Enter command: s
125
126      *
127      12345678901234567890
128      1|Line 1
129      2|Line 2
130      3|Line 3
131      4|Line 4
132      5|Line 5
133      6|Line 6
134      7|Line 7
135      8|Line 8
136      9|Line 9
137 *   10|Line 10
138     12345678901234567890
139 Enter command: s
140
141      *
142     12345678901234567890

```

```

143 2|Line 2
144 3|Line 3
145 4|Line 4
146 5|Line 5
147 6|Line 6
148 7|Line 7
149 8|Line 8
150 9|Line 9
151 10|Line 10
152 * 11|Line 11
153 12345678901234567890
154 Enter command: s
155
156 *
157 12345678901234567890
158 3|Line 3
159 4|Line 4
160 5|Line 5
161 6|Line 6
162 7|Line 7
163 8|Line 8
164 9|Line 9
165 10|Line 10
166 11|Line 11
167 * 12|Line 12
168 12345678901234567890
169 Enter command: s
170
171 *
172 12345678901234567890
173 4|Line 4
174 5|Line 5
175 6|Line 6
176 7|Line 7
177 8|Line 8
178 9|Line 9
179 10|Line 10
180 11|Line 11
181 * 12|Line 12
182 13
183 12345678901234567890
184 Enter command: s
185
186 *
187 12345678901234567890
188 5|Line 5
189 6|Line 6
190 7|Line 7
191 8|Line 8
192 9|Line 9
193 10|Line 10
194 11|Line 11
195 * 12|Line 12
196 13
197 14
198 12345678901234567890
199 Enter command: s
200
201 *
202 12345678901234567890
203 6|Line 6
204 7|Line 7
205 8|Line 8
206 9|Line 9
207 10|Line 10
208 11|Line 11
209 * 12|Line 12
210 13
211 14
212 15
213 12345678901234567890
214 Enter command: s
215
216

```

217 ... (I've omitted much of the output here) ...

```
218
219
220      *
221      12345678901234567890
222      9|Line 9
223      10|Line 10
224      11|Line 11
225 * 12|Line 12
226      13
227      14
228      15
229      16
230      17
231      18
232      12345678901234567890
233 Enter command: s
```

```
234
235      *
236      12345678901234567890
237      10|Line 10
238      11|Line 11
239 * 12|Line 12
240      13
241      14
242      15
243      16
244      17
245      18
246      19
247      12345678901234567890
248 Enter command: s
```

```
249
250      *
251      12345678901234567890
252      11|Line 11
253 * 12|Line 12
254      13
255      14
256      15
257      16
258      17
259      18
260      19
261      20
262      12345678901234567890
263 Enter command: s
```

```
264
265      *
266      12345678901234567890
267 * 12|Line 12
268      13
269      14
270      15
271      16
272      17
273      18
274      19
275      20
276      21
277      12345678901234567890
278 Enter command: s
```

```
279
280      *
281      12345678901234567890
282 * 12|Line 12
283      13
284      14
285      15
286      16
287      17
288      18
289      19
290      20
```

```

291     21
292     12345678901234567890
293 Enter command: w
294
295     *
296     12345678901234567890
297 * 11|Line 11
298 12|Line 12
299 13
300 14
301 15
302 16
303 17
304 18
305 19
306 20
307     12345678901234567890
308 Enter command: w
309
310     *
311     12345678901234567890
312 * 10|Line 10
313 11|Line 11
314 12|Line 12
315 13
316 14
317 15
318 16
319 17
320 18
321 19
322     12345678901234567890
323 Enter command: s
324
325     *
326     12345678901234567890
327 10|Line 10
328 * 11|Line 11
329 12|Line 12
330 13
331 14
332 15
333 16
334 17
335 18
336 19
337     12345678901234567890
338 Enter command: s
339
340     *
341     12345678901234567890
342 10|Line 10
343 11|Line 11
344 * 12|Line 12
345 13
346 14
347 15
348 16
349 17
350 18
351 19
352     12345678901234567890
353 Enter command: s
354
355     *
356     12345678901234567890
357 11|Line 11
358 * 12|Line 12
359 13
360 14
361 15
362 16
363 17
364 18

```

```

365     19
366     20
367     12345678901234567890
368 Enter command: s
369
370     *
371     12345678901234567890
372 * 12|Line 12
373     13
374     14
375     15
376     16
377     17
378     18
379     19
380     20
381     21
382     12345678901234567890
383 Enter command: w
384
385     *
386     12345678901234567890
387 * 11|Line 11
388 12|Line 12
389     13
390     14
391     15
392     16
393     17
394     18
395     19
396     20
397     12345678901234567890
398 Enter command: w
399
400     *
401     12345678901234567890
402 * 10|Line 10
403 11|Line 11
404 12|Line 12
405     13
406     14
407     15
408     16
409     17
410     18
411     19
412     12345678901234567890
413 Enter command: w
414
415     *
416     12345678901234567890
417 * 9|Line 9
418 10|Line 10
419 11|Line 11
420 12|Line 12
421     13
422     14
423     15
424     16
425     17
426     18
427     12345678901234567890
428 Enter command: q
429
430 Goodbye!

```

5.7 Part #6 - Moving the Cursor Repeatedly

Moving the cursor one spot at a time is a nightmare, so in this part, you will allow an integer to be given after any of ‘a’, ‘d’, ‘w’, or ‘s’ to indicate the number of times to perform that cursor movement. For instance “a 5” is the same as entering ‘a’ five times. Below are examples.

```

1 $ ./editor example_input_files/small.txt
2      *
3      12345678901234567890
4 * 1|Line 1
5   2|Line 2
6   3|Line 3
7   4|Line 4
8   5|Line 5
9   6
10  7
11  8
12  9
13  10
14      12345678901234567890
15 Enter command: d 3
16
17      *
18      12345678901234567890
19 * 1|Line 1
20   2|Line 2
21   3|Line 3
22   4|Line 4
23   5|Line 5
24   6
25   7
26   8
27   9
28   10
29      12345678901234567890
30 Enter command: a 2
31
32      *
33      12345678901234567890
34 * 1|Line 1
35   2|Line 2
36   3|Line 3
37   4|Line 4
38   5|Line 5
39   6
40   7
41   8
42   9
43   10
44      12345678901234567890
45 Enter command: s 3
46
47      *
48      12345678901234567890
49      1|Line 1
50      2|Line 2
51      3|Line 3
52 * 4|Line 4
53   5|Line 5
54   6
55   7
56   8
57   9
58   10
59      12345678901234567890
60 Enter command: w 1
61
62      *
63      12345678901234567890
64      1|Line 1
65      2|Line 2
66 * 3|Line 3
67   4|Line 4
68   5|Line 5
69   6
70   7
71   8
72   9
73   10
74      12345678901234567890

```

```

75 Enter command: s 4
76
77      *
78      12345678901234567890
79      3|Line 3
80      4|Line 4
81 * 5|Line 5
82      6
83      7
84      8
85      9
86      10
87      11
88      12
89      12345678901234567890
90 Enter command: w 100
91
92      *
93      12345678901234567890
94 * 1|Line 1
95      2|Line 2
96      3|Line 3
97      4|Line 4
98      5|Line 5
99      6
100     7
101     8
102     9
103     10
104     12345678901234567890
105 Enter command: d 80
106
107      *
108      12345678901234567890
109 * 1|Line 1
110      2|Line 2
111      3|Line 3
112      4|Line 4
113      5|Line 5
114      6
115      7
116      8
117      9
118      10
119      12345678901234567890
120 Enter command: q
121
122 Goodbye!

```

5.8 Part #7 - Repeating the Previous Command

If the user enters nothing, then your program should run the last command that the user entered. If this occurs for the first command (i.e. if there is no previous command), then an appropriate message should be printed, as shown below.

Below are examples.

```

1 $ ./editor example_input_files/small.txt
2      *
3      12345678901234567890
4 * 1|Line 1
5      2|Line 2
6      3|Line 3
7      4|Line 4
8      5|Line 5
9      6
10     7
11     8
12     9
13     10
14     12345678901234567890
15 Enter command: d 2
16
17      *

```

```

18      12345678901234567890
19 *   1|Line 1
20     2|Line 2
21     3|Line 3
22     4|Line 4
23     5|Line 5
24     6
25     7
26     8
27     9
28    10
29      12345678901234567890
30 Enter command:
31
32      *
33      12345678901234567890
34 *   1|Line 1
35     2|Line 2
36     3|Line 3
37     4|Line 4
38     5|Line 5
39     6
40     7
41     8
42     9
43    10
44      12345678901234567890
45 Enter command: s
46
47      *
48      12345678901234567890
49     1|Line 1
50 *   2|Line 2
51     3|Line 3
52     4|Line 4
53     5|Line 5
54     6
55     7
56     8
57     9
58    10
59      12345678901234567890
60 Enter command:
61
62      *
63      12345678901234567890
64     1|Line 1
65     2|Line 2
66 *   3|Line 3
67     4|Line 4
68     5|Line 5
69     6
70     7
71     8
72     9
73    10
74      12345678901234567890
75 Enter command: q
76
77 Goodbye!
78 $ ./editor example_input_files/small.txt
79      *
80      12345678901234567890
81 *   1|Line 1
82     2|Line 2
83     3|Line 3
84     4|Line 4
85     5|Line 5
86     6
87     7
88     8
89     9
90    10
91      12345678901234567890

```



```

92 Enter command:
93
94 No previous command.
95
96      *
97      12345678901234567890
98 * 1|Line 1
99   2|Line 2
100  3|Line 3
101  4|Line 4
102  5|Line 5
103   6
104   7
105   8
106   9
107  10
108      12345678901234567890
109 Enter command: q
110
111 Goodbye!

```

5.9 Part #8 - Insertion

To insert, the user can enter ‘i’ followed by a space and the string to insert (which may consist of multiple words/whitespaces). Insertion should begin at the cursor’s position. Below is an example of a simple insertion scenario involving saving. After this example, I talk about more complicated scenarios.

```

1 $ ./editor example_input_files/input4.txt
2      *
3      12345678901234567890
4 * 1|blah blah blah
5   2|blah blah blah
6   3|blah blah blah
7   4|blah blah blah
8   5|blah blah blah
9   6|blah blah blah
10   7
11   8
12   9
13  10
14      12345678901234567890
15 Enter command: i ABCD
16
17      *
18      12345678901234567890
19 * 1|ABCD blah blah
20   2|blah blah blah
21   3|blah blah blah
22   4|blah blah blah
23   5|blah blah blah
24   6|blah blah blah
25   7
26   8
27   9
28  10
29      12345678901234567890
30 Enter command: d
31
32      *
33      12345678901234567890
34 * 1|ABCD blah blah
35   2|blah blah blah
36   3|blah blah blah
37   4|blah blah blah
38   5|blah blah blah
39   6|blah blah blah
40   7
41   8
42   9
43  10
44      12345678901234567890
45 Enter command: s 4

```

```

46
47      *
48      12345678901234567890
49      1|ABCD  blah  blah
50      2|blah  blah  blah
51      3|blah  blah  blah
52      4|blah  blah  blah
53      * 5|blah  blah  blah
54      6|blah  blah  blah
55      7
56      8
57      9
58      10
59      12345678901234567890
60 Enter command: i AA $$ BB
61
62      *
63      12345678901234567890
64      1|ABCD  blah  blah
65      2|blah  blah  blah
66      3|blah  blah  blah
67      4|blah  blah  blah
68      * 5|bAA  $$ BB  blah
69      6|blah  blah  blah
70      7
71      8
72      9
73      10
74      12345678901234567890
75 Enter command: save output
76
77      *
78      12345678901234567890
79      1|ABCD  blah  blah
80      2|blah  blah  blah
81      3|blah  blah  blah
82      4|blah  blah  blah
83      * 5|bAA  $$ BB  blah
84      6|blah  blah  blah
85      7
86      8
87      9
88      10
89      12345678901234567890
90 Enter command: q
91
92 Goodbye!
93 $ cat output
94 ABCD  blah  blah
95 blah  blah  blah
96 blah  blah  blah
97 blah  blah  blah
98 bAA  $$ BB  blah
99 blah  blah  blah
100 $ diff example_input_files/input4.txt output
101 0a1
102 > ABCD  blah  blah
103 4,5c5
104 < blah  blah  blah
105 < blah  blah  blah
106 ---
107 > bAA  $$ BB  blah

```

As you've seen, not every line in the file will be 20 characters wide, but the cursor can reach the rightmost column regardless. If the user attempts to insert past the end of a line in a file, then your editor should extend that line. Below is an example.

```

1 $ wc -m example_input_files/tiny.txt
2 4 example_input_files/tiny.txt
3 $ cat example_input_files/tiny.txt
4 ABC
5 $ ./editor example_input_files/tiny.txt
6      *
7      12345678901234567890

```

```

8 * 1|ABC
9 2
10 3
11 4
12 5
13 6
14 7
15 8
16 9
17 10
18 12345678901234567890
19 Enter command: d 5
20
21 *
22 12345678901234567890
23 * 1|ABC
24 2
25 3
26 4
27 5
28 6
29 7
30 8
31 9
32 10
33 12345678901234567890
34 Enter command: i past the end
35
36 *
37 12345678901234567890
38 * 1|ABC past the end
39 2
40 3
41 4
42 5
43 6
44 7
45 8
46 9
47 10
48 12345678901234567890
49 Enter command: save less_tiny.txt
50
51 *
52 12345678901234567890
53 * 1|ABC past the end
54 2
55 3
56 4
57 5
58 6
59 7
60 8
61 9
62 10
63 12345678901234567890
64 Enter command: q
65
66 Goodbye!
67 $ cat less_tiny.txt
68 ABC past the end

```

If the insertion would go past column #20, then insertion should continue on the next line. If the next line does not already exist, then you should add a new line and continue from there. If a new line cannot be added (because it would be line #31), then insertion should not continue. Below are examples.

```

1 ./editor example_input_files/input1.txt
2 *
3 12345678901234567890
4 * 1|abcdef
5 2|ghi
6 3|jklm
7 4|jkj
8 5|jklasj

```

```

9      6|jklajs;lkfj
10     7|qiowuioj
11     8|lkjw;lkj
12     9|qwklejklj;c
13     10|jkaljsoiu
14         12345678901234567890
15 Enter command: d 17
16
17         *
18         12345678901234567890
19 * 1|abcdef
20   2|ghi
21   3|jklm
22   4|jkj
23   5|jklasj
24   6|jklajs;lkfj
25   7|qiowuioj
26   8|lkjw;lkj
27   9|qwklejklj;c
28   10|jkaljsoiu
29       12345678901234567890
30 Enter command: s 3
31
32         *
33         12345678901234567890
34   1|abcdef
35   2|ghi
36   3|jklm
37 * 4|jkj
38   5|jklasj
39   6|jklajs;lkfj
40   7|qiowuioj
41   8|lkjw;lkj
42   9|qwklejklj;c
43   10|jkaljsoiu
44       12345678901234567890
45 Enter command: i ABCDEFGHIJKLMNOPQRSTUVWXYZ
46
47         *
48         12345678901234567890
49   1|abcdef
50   2|ghi
51   3|jklm
52 * 4|jkj                ABC
53   5|DEFGHIJKLMNOPQRSTUWV
54   6|XYZajs;lkfj
55   7|qiowuioj
56   8|lkjw;lkj
57   9|qwklejklj;c
58   10|jkaljsoiu
59       12345678901234567890
60 Enter command: s 100
61
62         *
63         12345678901234567890
64 * 27|keljweiourpiu
65   28
66   29
67   30
68   31
69   32
70   33
71   34
72   35
73   36
74       12345678901234567890
75 Enter command: i add a new line
76
77         *
78         12345678901234567890
79 * 27|keljweiourpiu      add
80   28| a new line
81   29
82   30

```

```

83 31
84 32
85 33
86 34
87 35
88 36
89 12345678901234567890
90 Enter command: s
91
92 *
93 12345678901234567890
94 27|keljweiourpiu add
95 * 28| a new line
96 29
97 30
98 31
99 32
100 33
101 34
102 35
103 36
104 12345678901234567890
105 Enter command: d 2
106
107 *
108 12345678901234567890
109 27|keljweiourpiu add
110 * 28| a new line
111 29
112 30
113 31
114 32
115 33
116 34
117 35
118 36
119 12345678901234567890
120 Enter command: i hi
121
122 *
123 12345678901234567890
124 27|keljweiourpiu add
125 * 28| a new line h
126 29|i
127 30
128 31
129 32
130 33
131 34
132 35
133 36
134 12345678901234567890
135 Enter command: s
136
137 *
138 12345678901234567890
139 27|keljweiourpiu add
140 28| a new line h
141 * 29|i
142 30
143 31
144 32
145 33
146 34
147 35
148 36
149 12345678901234567890
150 Enter command: i hi
151
152 *
153 12345678901234567890
154 27|keljweiourpiu add
155 28| a new line h
156 * 29|i h

```

```

157 30|i
158 31
159 32
160 33
161 34
162 35
163 36
164 12345678901234567890
165 Enter command: s
166
167 *
168 12345678901234567890
169 27|keljweieurpiu add
170 28| a new line h
171 29|i h
172 * 30|i
173 31
174 32
175 33
176 34
177 35
178 36
179 12345678901234567890
180 Enter command: a 2
181
182 *
183 12345678901234567890
184 27|keljweieurpiu add
185 28| a new line h
186 29|i h
187 * 30|i
188 31
189 32
190 33
191 34
192 35
193 36
194 12345678901234567890
195 Enter command: i Writing this whole thing will not work because this horrible editor does not let you go
past line 30.
196
197 *
198 12345678901234567890
199 27|keljweieurpiu add
200 28| a new line h
201 29|i h
202 * 30|i Wri
203 31
204 32
205 33
206 34
207 35
208 36
209 12345678901234567890
210 Enter command: save output.txt
211
212 *
213 12345678901234567890
214 27|keljweieurpiu add
215 28| a new line h
216 29|i h
217 * 30|i Wri
218 31
219 32
220 33
221 34
222 35
223 36
224 12345678901234567890
225 Enter command: q
226
227 Goodbye!
228 $ diff example_input_files/input1.txt output.txt
229 4,6c4,6

```

```

230 < jkj
231 < jklasj
232 < jklajs;lkfj
233 ---
234 > jkj                ABC
235 > DEFGHIJKLMNOPQRSTUW
236 > XYZajs;lkfj
237 27c27,30
238 < keljweiourpiu
239 ---
240 > keljweiourpiu      add
241 > a new line         h
242 > i                  h
243 > i                  Wri
244 $ wc -l example_input_files/input1.txt
245 27 example_input_files/input1.txt
246 $ wc -l output.txt
247 30 output.txt

```

5.10 Part #9 - Edit History

In this part, you will modify the editor so that *insertions* (no other operations) can be undone or redone by entering ‘u’ or ‘r’, respectively. This requires that your program maintain undo and redo histories.

When I say “new insertion” in this paragraph, I am referring to an insertion that is not caused by an undo operation or a redo operation. The undo operation should undo the last insertion done, whether that insertion was a “new insertion” or the result of a redo operation. The redo operation should redo the last insertion that was undone. If the user performs a “new insertion”, then the redo history should be erased.

Note that undo/redo operations do not care where the cursor or window are and do not change where the cursor or window are.

You are required to use a stack (or two stacks) to do this part. You should use `std::stack` from the STL.

v.3 Update: You may assume that the autograder will never do an undo operation that causes the cursor to go out of bounds (by undoing the insertion/creation of the line that the cursor happens to be on).

Below are examples of how your program should behave. Notice the messages that should be printed if there are no actions to undo or redo.

```

1 $ ./editor example_input_files/small.txt
2      *
3      12345678901234567890
4 *  1|Line 1
5    2|Line 2
6    3|Line 3
7    4|Line 4
8    5|Line 5
9    6
10   7
11   8
12   9
13  10
14      12345678901234567890
15 Enter command: i Hello there
16
17      *
18      12345678901234567890
19 *  1|Hello there
20    2|Line 2
21    3|Line 3
22    4|Line 4
23    5|Line 5
24    6
25    7
26    8
27    9
28   10
29      12345678901234567890
30 Enter command: u
31
32      *
33      12345678901234567890
34 *  1|Line 1
35    2|Line 2

```

```

36      3|Line 3
37      4|Line 4
38      5|Line 5
39      6
40      7
41      8
42      9
43      10
44      12345678901234567890
45 Enter command: u
46
47 Cannot undo.
48
49      *
50      12345678901234567890
51 * 1|Line 1
52   2|Line 2
53   3|Line 3
54   4|Line 4
55   5|Line 5
56   6
57   7
58   8
59   9
60  10
61      12345678901234567890
62 Enter command: r
63
64      *
65      12345678901234567890
66 * 1|Hello there
67   2|Line 2
68   3|Line 3
69   4|Line 4
70   5|Line 5
71   6
72   7
73   8
74   9
75  10
76      12345678901234567890
77 Enter command: r
78
79 Cannot redo.
80
81      *
82      12345678901234567890
83 * 1|Hello there
84   2|Line 2
85   3|Line 3
86   4|Line 4
87   5|Line 5
88   6
89   7
90   8
91   9
92  10
93      12345678901234567890
94 Enter command: i ABC
95
96      *
97      12345678901234567890
98 * 1|ABClo there
99   2|Line 2
100  3|Line 3
101  4|Line 4
102  5|Line 5
103  6
104  7
105  8
106  9
107 10
108      12345678901234567890
109 Enter command: s

```



```

110
111      *
112      12345678901234567890
113      1|ABClo there
114 *    2|Line 2
115      3|Line 3
116      4|Line 4
117      5|Line 5
118      6
119      7
120      8
121      9
122      10
123      12345678901234567890
124 Enter command: i DEF
125
126      *
127      12345678901234567890
128      1|ABClo there
129 *    2|DEFe 2
130      3|Line 3
131      4|Line 4
132      5|Line 5
133      6
134      7
135      8
136      9
137      10
138      12345678901234567890
139 Enter command: s
140
141      *
142      12345678901234567890
143      1|ABClo there
144      2|DEFe 2
145 *    3|Line 3
146      4|Line 4
147      5|Line 5
148      6
149      7
150      8
151      9
152      10
153      12345678901234567890
154 Enter command: i GHI
155
156      *
157      12345678901234567890
158      1|ABClo there
159      2|DEFe 2
160 *    3|GHIE 3
161      4|Line 4
162      5|Line 5
163      6
164      7
165      8
166      9
167      10
168      12345678901234567890
169 Enter command: r
170
171 Cannot redo.
172
173      *
174      12345678901234567890
175      1|ABClo there
176      2|DEFe 2
177 *    3|GHIE 3
178      4|Line 4
179      5|Line 5
180      6
181      7
182      8
183      9

```

```

184 10
185 12345678901234567890
186 Enter command: u
187
188 *
189 12345678901234567890
190 1|ABClo there
191 2|DEFe 2
192 * 3|Line 3
193 4|Line 4
194 5|Line 5
195 6
196 7
197 8
198 9
199 10
200 12345678901234567890
201 Enter command: u
202
203 *
204 12345678901234567890
205 1|ABClo there
206 2|Line 2
207 * 3|Line 3
208 4|Line 4
209 5|Line 5
210 6
211 7
212 8
213 9
214 10
215 12345678901234567890
216 Enter command: u
217
218 *
219 12345678901234567890
220 1|Hello there
221 2|Line 2
222 * 3|Line 3
223 4|Line 4
224 5|Line 5
225 6
226 7
227 8
228 9
229 10
230 12345678901234567890
231 Enter command: r
232
233 *
234 12345678901234567890
235 1|ABClo there
236 2|Line 2
237 * 3|Line 3
238 4|Line 4
239 5|Line 5
240 6
241 7
242 8
243 9
244 10
245 12345678901234567890
246 Enter command: i XYZ
247
248 *
249 12345678901234567890
250 1|ABClo there
251 2|Line 2
252 * 3|XYZe 3
253 4|Line 4
254 5|Line 5
255 6
256 7
257 8

```

```

258     9
259    10
260      12345678901234567890
261 Enter command: r
262
263 Cannot redo.
264
265     *
266      12345678901234567890
267 1|ABClo there
268 2|Line 2
269 * 3|XYZe 3
270 4|Line 4
271 5|Line 5
272 6
273 7
274 8
275 9
276 10
277      12345678901234567890
278 Enter command: u
279
280     *
281      12345678901234567890
282 1|ABClo there
283 2|Line 2
284 * 3|Line 3
285 4|Line 4
286 5|Line 5
287 6
288 7
289 8
290 9
291 10
292      12345678901234567890
293 Enter command: ... (I've omitted the rest of the output.) ...

```

5.11 Part #10 - Attempting to Quit with Unsaved Changes

If the user attempts to quit but has not yet saved the current state of the buffer contents to a file, then your program should inform the user of this and ask the user if he/she still wishes to quit, like what real editors (e.g. Sublime Text). Below are examples of the simple case. After these examples, I talk about why this is more complicated than it seems.

```

1 $ ./editor example_input_files/small.txt
2     *
3      12345678901234567890
4 * 1|Line 1
5 2|Line 2
6 3|Line 3
7 4|Line 4
8 5|Line 5
9 6
10 7
11 8
12 9
13 10
14      12345678901234567890
15 Enter command: i DEF
16
17     *
18      12345678901234567890
19 * 1|DEF 1
20 2|Line 2
21 3|Line 3
22 4|Line 4
23 5|Line 5
24 6
25 7
26 8
27 9
28 10
29      12345678901234567890

```

```

30 Enter command: q
31
32 You have unsaved changes.
33 Are you sure you want to quit (y or n)?
34 n
35
36      *
37      12345678901234567890
38 *  1|DEFe 1
39    2|Line 2
40    3|Line 3
41    4|Line 4
42    5|Line 5
43    6
44    7
45    8
46    9
47   10
48      12345678901234567890
49 Enter command: save blah
50
51      *
52      12345678901234567890
53 *  1|DEFe 1
54    2|Line 2
55    3|Line 3
56    4|Line 4
57    5|Line 5
58    6
59    7
60    8
61    9
62   10
63      12345678901234567890
64 Enter command: q
65
66 Goodbye!
67 $ cat blah
68 DEFe 1
69 Line 2
70 Line 3
71 Line 4
72 Line 5

```

Once we consider the edit history, things become more complicated. On editors like Sublime Text, if – after a save operation – you do a modification and then perform an undo operation (or if you perform an undo operation followed by a redo operation), the editor will recognize that you are back to the state you last saved and not think that you have unsaved changes. However, if – again, after a save operation – you were to insert “DEF” to replace “ABC” and then insert “ABC” to replace the “DEF” you just inserted, these will be perceived as unsaved changes, even though the current state is identical to the last state saved. Your editor should emulate all of this behavior. Below are examples.

```

1 $ ./editor example_input_files/small.txt
2      *
3      12345678901234567890
4 *  1|Line 1
5    2|Line 2
6    3|Line 3
7    4|Line 4
8    5|Line 5
9    6
10   7
11   8
12   9
13  10
14      12345678901234567890
15 Enter command: i ABC
16
17      *
18      12345678901234567890
19 *  1|ABCe 1
20    2|Line 2
21    3|Line 3
22    4|Line 4

```

```

23 5|Line 5
24 6
25 7
26 8
27 9
28 10
29 12345678901234567890
30 Enter command: save output
31
32 *
33 12345678901234567890
34 * 1|ABCe 1
35 2|Line 2
36 3|Line 3
37 4|Line 4
38 5|Line 5
39 6
40 7
41 8
42 9
43 10
44 12345678901234567890
45 Enter command: u
46
47 *
48 12345678901234567890
49 * 1|Line 1
50 2|Line 2
51 3|Line 3
52 4|Line 4
53 5|Line 5
54 6
55 7
56 8
57 9
58 10
59 12345678901234567890
60 Enter command: q
61
62 You have unsaved changes.
63 Are you sure you want to quit (y or n)?
64 n
65
66 *
67 12345678901234567890
68 * 1|Line 1
69 2|Line 2
70 3|Line 3
71 4|Line 4
72 5|Line 5
73 6
74 7
75 8
76 9
77 10
78 12345678901234567890
79 Enter command: r
80
81 *
82 12345678901234567890
83 * 1|ABCe 1
84 2|Line 2
85 3|Line 3
86 4|Line 4
87 5|Line 5
88 6
89 7
90 8
91 9
92 10
93 12345678901234567890
94 Enter command: q
95
96 Goodbye!

```

```

97 $ ./editor example_input_files/small.txt
98      *
99      12345678901234567890
100 *   1|Line 1
101     2|Line 2
102     3|Line 3
103     4|Line 4
104     5|Line 5
105     6
106     7
107     8
108     9
109    10
110      12345678901234567890
111 Enter command: i ABC
112
113      *
114      12345678901234567890
115 *   1|ABCe 1
116     2|Line 2
117     3|Line 3
118     4|Line 4
119     5|Line 5
120     6
121     7
122     8
123     9
124    10
125      12345678901234567890
126 Enter command: i Lin
127
128      *
129      12345678901234567890
130 *   1|Line 1
131     2|Line 2
132     3|Line 3
133     4|Line 4
134     5|Line 5
135     6
136     7
137     8
138     9
139    10
140      12345678901234567890
141 Enter command: q
142
143 You have unsaved changes.
144 Are you sure you want to quit (y or n)?
145 y
146
147 Goodbye!

```

5.12 Epilogue: Additional Features

Do not add anything from this subsection into what you end up submitting to Gradescope.

Below is a non-exhaustive list of features you could add to this editor *on your own time* (again, not for your submission to Gradescope), in order to make it more complex and, perhaps, more résumé-worthy.

- Other forms of modification: deletion, find-and-replace, etc.
- Use the curses library to make the editor take up the entire terminal in one area, so that you don't just have to keep printing out the file's contents periodically. The curses library also allows you to react to inputs without the user having to press the Enter key. It's admittedly pretty difficult to explain what the curses library can do for you, because it can do so much, so I would recommend you go through a tutorial on it³; it really opens up the possibilities when it comes to terminal-based applications and would make your editor more usable by allowing you to not have to press the Enter key all the time.
- *Requires ECS 50 knowledge of bits and bit operators*: Support files that are encrypted in a certain way, e.g. with run-length encoding or Huffman encoding.

³It's possible that a later assignment in this course has you use the curses library; I'm not sure yet. (The problem is it's impossible to autograde such an assignment.)

- The ability to edit multiple files simultaneously (like having multiple tabs in an editor).
- A clipboard for copy/pasting.
- Whatever features make your editor so good that you can use it to write your code in future courses :-)

6 Grading Breakdown

As stated in the updated syllabus, this assignment is worth 9% of your final grade. Below is the breakdown of the worth of the test cases for each part (for a total of 90 points):

- Cases for part #1: 5 points. (Cases #1 through #5)
- Cases for part #2: 10 points. (Cases #6 through #7)
- Case for part #3: 5 points. (Case #8)
- Cases for part #4: 5 points. (Cases #9 through #10)
- Cases for part #5: 17.5 points. (Cases #11 through #17)
- Cases for part #6: 5 points. (Cases #18 through #19)
- Cases for part #7: 5 points. (Cases #20 through #21)
- Cases for part #8: 15 points. (Cases #22 through #27)
- Cases for part #9: 15 points. (Cases #28 through #31)
- Cases for part #10: 7.5 points. (Cases #32 through #34)

7 Autograder Details

Once the autograder is set up on Gradescope, I will send a Canvas announcement and add additional details below.

If you haven't already, you should read what I say in the syllabus about the Gradescope autograder.

As mentioned above, the *only* file that you should submit to Gradescope is `editor.cpp`. Any other file that you submit will be ignored. The autograder will compile your code with the below command.

```
1 g++ -std=c++14 -Wall -Werror editor.cpp -o editor
```

Your output must match mine *exactly*.

The autograder will grade your submission out of 90 points.

Even if you do not finish a specific part, you might still be able to pass some of the test cases corresponding to later parts. For instance, even if you cannot get saving to a file to work, you could still pass test cases concerning scrolling up/down and undo/redo (except for the ones that happen to involve saving).

Each test case corresponds to a shell script that you can find in the `case_scripts` folder on Canvas. Scripts in this folder often get their standard input from files that you can find in the `stdin` folder on Canvas. Of course, if your understanding of Linux and shell scripting is not that great, it might limit your ability to understand what is going on in the shell scripts, which is why I encouraged you to make yourself more comfortable with Linux and shell scripting [in a Canvas announcement](#) sent towards the start of this quarter.

At least one of the hidden test cases is failed if your code has memory leaks.

7.1 Regarding Timeouts

As this is a bigger assignment with a lot more output for the autograder to evaluate, it is possible that the autograder can “give up” if there is too much output, which usually happens if your program did not end, whether due to an infinite loop or because it did not terminate when expected. When this happens, you could see a general message about the autograder timing out, or you could see messages like the below on specific test cases.

```
1 Test Failed: Command '['bash', './run.sh']' timed out after 30 seconds
```

```
1 Test Failed: [Errno 12] Cannot allocate memory
```

If you get a case-specific error message, then you should run the visible test case's input on your end (on the CSIF) and debug from there. If your program is behaving properly on the CSIF, then feel free to email me.

If you get a general autograder timeout message, then you should run all visible test cases' inputs on your end and debug from there. If your program is behaving properly on the CSIF, then feel free to email me.

If the autograder is taking a long time, it is probably not a good idea to just keep resubmitting the same code over and over and hope that something different somehow happens.

If you get the below message, then please email me with the URL of this Gradescope submission. In the meantime, you should run the visible test cases' inputs on your end and see if they all work.

1 The autograder failed to execute correctly. Please ensure that your submission is valid. Contact your course staff for help in debugging this issue. Make sure to include a link to this page so that they can help you most effectively.

