

# ECS 32B: Programming Assignment #2

Instructor: Aaron Kaloti

Summer Session #2 2020

## Contents

<b>1 Changelog</b>	<b>1</b>
<b>2 General Submission Details</b>	<b>1</b>
<b>3 Grading Breakdown</b>	<b>2</b>
<b>4 Gradescope Autograder Details</b>	<b>2</b>
4.1 Released Test Cases vs. Hidden Test Cases . . . . .	2
4.2 Released Test Cases' Inputs . . . . .	2
4.3 Changing Gradescope Active Submission . . . . .	2
<b>5 Ayayron™: An Editor That No One in Their Right Mind Would Ever Use</b>	<b>2</b>
5.1 Part #1 - Loading Keys: <code>load_keys()</code> . . . . .	3
5.2 Part #2 - Reading the Edited File's Contents . . . . .	4
5.3 Part #3 - Printing the Current File Contents: <code>print_current()</code> . . . . .	5
5.4 Part #4 - Saving the Current File's Contents: <code>save()</code> . . . . .	7
5.5 Part #5 - Moving the Cursor . . . . .	8
5.5.1 Part #5.1 - Leftward/Rightward Movement: <code>move_left()</code> and <code>move_right()</code> . . . . .	8
5.5.2 Part #5.2 - Upward/Downward Movement and Scrolling the Window: <code>move_up()</code> and <code>move_down()</code> . . . . .	11
5.6 Part #6 - Insertion: <code>insert()</code> . . . . .	19
5.7 Part #7 - Edit History . . . . .	24
5.7.1 Part #7.1 - Undoing an Insertion: <code>undo()</code> . . . . .	24
5.7.2 Part #7.2 - Redoing an Insertion: <code>redo()</code> . . . . .	27

## 1 Changelog

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

- v.1: Initial version.
- v.2: Made it clearer in part #3 the number of whitespace in certain portions (see red text).

## 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 Saturday, August 29. Gradescope will say 12:30 AM on Sunday, August 30, due to the “grace period” (as described in the syllabus). *Rely on the grace period for extra time at your own risk.*

Some students tend to email me very close to the deadline. This is a bad idea. There is no guarantee that I will check my email right before the deadline.

---

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

### 3 Grading Breakdown

The assignment is out of 160 points. Here is a *tentative* breakdown of the worth of each part.

- Part #1: 15
- Part #2: 20
- Part #3: 25
- Part #4: 15
- Part #5: 30
  - Part #5.1: 5
  - Part #5.2: 25
- Part #6: 20
- Part #7: 35
  - Part #7.1: 20
  - Part #7.2: 15

### 4 Gradescope Autograder Details

Once the autograder is released, I will update this document to include important details here. Note that you should be able to verify the correctness of your functions without depending on the autograder. I am aiming to work on the autograder around 08/22.

#### 4.1 Released Test Cases vs. Hidden Test Cases

You will not see the results of certain test cases until after the deadline; these are the hidden test cases, and Gradescope will not tell you whether you have gotten them correct or not until after the deadline. One purpose of this is to promote the idea that you should test if your own code works and not depend solely on the autograder to do it. Unfortunately, Gradescope will display a dash as your score until after the deadline, but you can still tell if you have passed all of the visible test cases if you see no test cases that are marked red for your submission.

#### 4.2 Released Test Cases' Inputs

TBA.

#### 4.3 Changing Gradescope Active Submission

Once the deadline occurs, whatever submission is your **active submission** will be the one that dictates your final score. By default, your active submission will be your latest submission. However, you can change which submission is your active submission, which I talk about a small video that you can find on Canvas [here](#). This video is from my ECS 32A course last summer session, so the autograder referenced therein is for a different assignment. Note that due to the hidden test cases, your score will show as a dash for all of the submissions, but you can still identify how many of the released test cases you got correct for each submission.

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

In this assignment, you will implement a text editor. The only actual editing that this editor will permit is insertion of a phrase on a given line (based on where the cursor is) and the trimming of trailing whitespace. Other helpful operations will include movement of the cursor, scrolling, and saving to a file.

Your entire submission will be in a file called `editor.py`. You can find a skeleton version of `editor.py` and a complete implementation of the `Stack` implementation from lecture in `stack.py` on Canvas. You can change the parameter names of the methods in `editor.py`, if you wish. You will only submit `editor.py` to the autograder; the autograder will use its own copy of `stack.py`, meaning that you are not allowed to modify this file. So far, the code should just be able to tell you when you've entered an invalid command and should allow you to quit.

```
1 >>> e = Editor('ignore', 'ignore')
2 >>> e.run()
3 Enter command: d
4
5 Enter command: a
```

```

6
7 Enter command: z
8 Invalid command.
9
10 Enter command: o
11 Invalid command.
12
13 Enter command: end
14 >>>

```

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 personally found part #5.2 to be the hardest part. In some autograder cases, I may try to reduce the dependency of certain parts on each other. For example, even if you cannot quite get part #5.2 fully working, it will be possible to get some points for parts #6 and #7.

Although you understandably may not understand every single detail of these specifications on your first read through, you may find it worthwhile to read the entire specifications before starting anyways.

I would recommend skimming the list of string and list methods [here](#) and [here](#). I am not prohibiting any of these on this assignment. I personally found the string methods `rstrip()` and `rjust()` to be useful.

Do not forget to close any file that you open, once you are done using it. Or just use the `with` operator.

You will find that you'll need to add members to the `Editor` class that are not explicitly stated in the directions; you can figure these out as you go.

## 5.1 Part #1 - Loading Keys: `load_keys()`

As you can see in `editor.py`, there are default values of each of the key members, from `self.down_key` to `self.redo_key`. Each such member contains the key that must be entered in order to do that operation; for example, to undo the last edit/insertion, the user would need to press the key whose value is in `self.undo_key`.

The last parameter (or, you could say, third explicit parameter), `settings_filename`, is the name of a settings file. This file is used in order to override the default keys. It is not always the case that one wants to override the default keys, which is why the `settings_filename` has a default value of `None`.

**Your task for this part** is to implement the `load_keys()` method. If properly formatted, the settings file should have eight lines, with one character per line. In order, the characters should be the following:

1. Down key.
2. Up key.
3. Left key.
4. Right key.
5. Insert key.
6. Save key.
7. Undo key.
8. Redo key.

You may assume that the file always has eight lines. The `load_keys()` method should **raise** a `ValueError` if any of the lines is empty or contains more than one character (ignoring the newline character).

This next detail is something that I hate to waste a paragraph on, but it is worth pointing out because it relates to potential bugs you may encounter. You may recall that each line read from the file will end with a newline character. Unfortunately, some editors (e.g. Sublime Text) do not automatically insert a newline character at the end of the last line of a text file. This is arguably a bug/ flaw in such editors<sup>1</sup>, and you should expect that any text file used by the autograder will have a newline character at the end of its last line. However, if you create your own text files and find that your code crashes on the last line, then it could be because of the lack of a newline character at the end of the last line, depending on which editor you used.

I personally found it easier to first check if any line in the file has the wrong length and then read the keys. This required two passes through the file, causing me to use the `seek()` method (with 0 as the argument) to tell the file object to start from the beginning of the file. Obviously, doing two passes is arguably inefficient, and you don't need to do that if you don't want to; you don't need to use `seek()`.

Below are examples (along with relevant text files) of what the result of `load_keys()` should be.

`settings1.txt`

---

<sup>1</sup>I say this because the POSIX standards define a text file as a file organized into lines, where each line ends in a newline character. This is as opposed to a binary file (which admittedly may seem like nonsense, because this definition implies that a text file that does not end in a newline character should be called a binary file even if it clearly only contains human-readable text, but that's beside the point).

```
1 d
2 u
3 l
4 r
5 i
6 s
7 z
8 y
```

bad\_settings1.txt

```
1 d
2 u
3 l
4 rr
5 i
6 s
7 z
8 y
```

bad\_settings2.txt

```
1 d
2 u
3 l
4
5 i
6 s
7 z
8 y
```

```
1 >>> e = Editor("ignore for now", "ignore this for now too") # default (no third argument)
2 >>> e.down_key
3 's'
4 >>> e.redo_key
5 'r'
6 >>> e = Editor("ignore for now", "ignore this for now too", "settings1.txt")
7 >>> e.down_key
8 'd'
9 >>> e.redo_key
10 'y'
11 >>> e.up_key
12 'u'
13 >>> e = Editor("ignore for now", "ignore this for now too", "bad_settings1.txt")
14 Traceback (most recent call last):
15 ...
16 ValueError: At least one line in settings is too long.
17 >>> e = Editor("ignore for now", "ignore this for now too", "bad_settings2.txt")
18 Traceback (most recent call last):
19 ...
20 ValueError: At least one line in settings is empty.
```

## 5.2 Part #2 - Reading the Edited File's Contents

The first explicit argument to the `Editor` initializer, called `infilename`, is the name of a file whose contents the editor will start with when editing. You need to read and store this file's contents. There are a few options that you have for storing the file's contents. The most intuitive (and probably best) option is a list of strings. Another option is a list of lists of characters. The primary benefit of a list of lists of characters is that you will not have to deal with the fact that strings are immutable in part #6, but other than that, a list of strings is perhaps preferable. If you go with the list of lists of characters, you may find the fact that `list(...)` can convert a string into a list of its characters to be useful, as shown below:

```
1 >>> list("abcd")
2 ['a', 'b', 'c', 'd']
```

Your initializer (or a helper function called by the initializer) should **raise** a `ValueError` in either of the following cases:

- At least one line in the file is longer than the window width (see part #3), which is 20 characters (not counting the newline character).
- The input file is empty.
- The input file has more than 30 lines.

No example output is shown, since part #3 indirectly does this. As stated in parts #3 and #4, the editor *trims* trailing whitespace (see part #3 for a description of what this means), so you may want to make use of the string method `rstrip()` in this part.

### 5.3 Part #3 - Printing the Current File Contents: `print_current()`

Whenever the edited file's contents are displayed, it will be within a window that is 20 characters wide and 10 characters tall. (These values 20 and 10 are stored in the `window_width` and `window_height` fields of the `Editor` class, as you can see at the start of the initializer.) Through this window, we see only a part of the edited file's contents at any given time. The file's contents will never be wider than the window, because the `Editor` initializer would have raised a `ValueError` and, as you'll see in part #6, the user will not be allowed to extend a line so that it's wider than the window<sup>2</sup>. Initially, the window should start at line 1; in part #5.2, you will allow the window to be scrolled down. For example, suppose that we have the following text file.

```
1 abcdef
2 ab
3
4 xyz
5 Hi there
6 Hi how are you
7 I am good
8 blah
9 blah blah blah blah
10 stream of
11 consciousness
12 don't fail me now
13 blah
14
15 that is all
```

Without the window being moved, `print_current()` would result in only the first 10 lines being printed, as shown below.

```
1      *
2      12345678901234567890
3 * 1 abcdef
4   2 ab
5   3
6   4 xyz
7   5 Hi there
8   6 Hi how are you
9   7 I am good
10  8 blah
11  9 blah blah blah blah
12 10 stream of
13 12345678901234567890
```

(Update: 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.)

Below is the above, but shown on the Python IDLE interpreter instead. This is what it should look like so far if you have completed this part. Note that the `run()` method already calls `print_current()` when appropriate. In fact, I would *highly recommend against* modifying `run()` unless you have a good reason to do so. You do not need to store the text files that you use in a folder called `text_files`; I did that for my own file organization.

```
1 >>> e = Editor('text_files/input3.txt', 'ignore for now')
2 >>> e.run()
3      *
4      12345678901234567890
5 * 1 blah
6   2 blah blah
7   3 blah blah blah
8   4
9   5 what is up
10  5
11  6
12  7
13  8
14  9
15 10
```

---

<sup>2</sup>As I said above, no one would want to use this editor.

```

16      12345678901234567890
17 Enter command:

```

As you can see, 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 part #5, you will allow the cursor to be moved, and in part #6, you will allow the user to make insertions starting at the current location of the editor. Before getting to those parts, you can always manually change the position of the cursor in the code in order to check if your `print_current()` implementation works regardless of the cursor's location, so as to minimize the chance that you have to modify `print_current()` once you are done with this part. Lastly, the line numbers of the file are shown. The max line number is 30 (since files that are longer than that are rejected by the `Editor` initializer). Since line numbers may be one digit or two digits, you need to be mindful of how many whitespaces to print before it; this is why I suggest the `rjust()` string method.

**Regarding trailing whitespaces:** "Trailing whitespace" refers to the occurrence of whitespace at the end of a line. When printing strings out, it is difficult to visually identify trailing whitespace. For example, for all you know, there could be 30 blank whitespaces after the asterisk printed in the first line of the above example output. **Do not worry about trailing whitespace in this assignment.** The autograder will ignore trailing whitespace when comparing the output of your code with mine. In some cases, part of the file itself may include trailing whitespace. For example, line #1 in the above example output could have four whitespaces after the `abcdef`, and line #3 could contain six whitespaces in it. You do not need to worry about trailing whitespace here either. **The editor will trim trailing whitespace**, meaning that all trailing whitespace will be removed, e.g. line #1 would have the four trailing whitespaces removed, and line #3 would be reduced to a completely empty line. This primarily matters for part #4.

Suppose that in the above example, the window instead was currently starting at line #4 and the cursor was at the sixth column and the eighth row. In that case, the output of `print_current()` would look like below. Again, it will not be possible to move the cursor or scroll the window until you've done part #5. However, you may want to experiment with moving the window artificially (by changing the relevant members that you hopefully will add to your `Editor` class) before moving on from this part.

```

1      *
2      12345678901234567890
3      4 xyz
4      5 Hi there
5      6 Hi how are you
6      7 I am good
7      8 blah
8      9 blah blah blah blah
9      10 stream of
10 * 11 consciousness
11 12 don't fail me now
12 13 blah
13 12345678901234567890

```

Below is an example (the file and the initial view) of what happens when the edited file lacks enough lines to fill the entire height of the window.

File:

```

1 blah blah
2 blah blah blah
3
4 what is up

```

Initial output of `print_current()`:

```

1      *
2      12345678901234567890
3 * 1 1 blah
4 2 2 blah blah
5 3 3 blah blah blah
6 4
7 5 5 what is up
8 6
9 7
10 8
11 9
12 10
13 12345678901234567890

```

## 5.4 Part #4 - Saving the Current File's Contents: `save()`

Implement the `save()` method so that it saves the contents of the currently edited file. The file that you should save to is the one whose name was given as the second explicit argument in the initializer of `Editor`. Its original contents should be erased. Note that the same file can be the one that was read and written to (in which case the first and second arguments to the `Editor` initializer would both be that file's name).

This method is already called by the `run()` method when the user enters the save key.

If you chose to store the edited file's contents in a list of lists of characters, then you may find the string method `join()` to be useful, as shown below.

```
1 >>> "".join(['a', 'b', 'c', 'd'])
2 'abcd'
```

Below is an example of how saving should behave.

Input file (first argument to initializer):

```
1 Line 1
2 Line 2
3 Line 3
4 Line 4
5 Line 5
6 Line 6
7 Line 7
8 Line 8
9 Line 9
10 Line 10
11 Line 11
12 Line 12
```

Interpreter interaction:

```
1 >>> e = Editor('text_files/input4.txt', 'text_files/output4.txt')
2 >>> e.run()
3      *
4      12345678901234567890
5 *   1 Line 1
6     2 Line 2
7     3 Line 3
8     4 Line 4
9     5 Line 5
10    6 Line 6
11    7 Line 7
12    8 Line 8
13    9 Line 9
14   10 Line 10
15    12345678901234567890
16 Enter command: e
17
18      *
19      12345678901234567890
20 *   1 Line 1
21    2 Line 2
22    3 Line 3
23    4 Line 4
24    5 Line 5
25    6 Line 6
26    7 Line 7
27    8 Line 8
28    9 Line 9
29   10 Line 10
30    12345678901234567890
31 Enter command: q
32 >>>
```

Resulting output file (second argument to initializer), which is exactly the same as the input file since no modifications were made:

```
1 Line 1
2 Line 2
3 Line 3
4 Line 4
5 Line 5
6 Line 6
7 Line 7
```

```

8 Line 8
9 Line 9
10 Line 10
11 Line 11
12 Line 12

```

As stated before, the editor removes trailing whitespace. Thus, if the initial file contains trailing whitespace, the output file will *not* contain it. I would deal with this by using the string method `rstrip()` while writing out each line to the output file.

Input file: (there are three whitespaces after “Line 3”)

```

1 Line 1
2 Line 2
3 Line 3
4 Line 4
5 Line 5
6 Line 6
7 Line 7
8 Line 8
9 Line 9
10 Line 10
11 Line 11
12 Line 12

```

Interpreter interaction:

```

1 [Basically the same as in the previous example.]

```

Resulting output file:

```

1 Line 1
2 Line 2
3 Line 3
4 Line 4
5 Line 5
6 Line 6
7 Line 7
8 Line 8
9 Line 9
10 Line 10
11 Line 11
12 Line 12

```

There is a file method called `writelines()` that may eliminate the need for a loop in this part. In my own implementation, I didn’t use `writelines()`, since I chose not to store the newline character at the end of each line.

## 5.5 Part #5 - Moving the Cursor

### 5.5.1 Part #5.1 - Leftward/Rightward Movement: `move_left()` and `move_right()`

The `move_left()` and `move_right()` methods should change the “x-coordinate” of the cursor, i.e. change which column the cursor is on. If moving the cursor would cause it to go out of bounds (i.e. to before the first column or after the 20th column), then the movement should be prevented.

Below are examples of how your code should behave once you have successfully implemented `move_left()` and `move_right()`. Note that the `run()` method already calls these methods when the appropriate directional commands are entered.

```

1 >>> e = Editor('text_files/input1.txt', 'ignore')
2 >>> e.run()
3
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 jkaljsou
16      12345678901234567890
17 Enter command: d

```



```

18      *
19      12345678901234567890
20 * 1 abcdef
21   2 ghi
22   3 jklm
23   4 jkj
24   5 jklasj
25   6 jklajs;lkfj
26   7 qiowuioj
27   8 lkjw;lkj
28   9 qwklejklj;c
29   10 jkaljsoiu
30      12345678901234567890
31 Enter command: d
32
33      *
34      12345678901234567890
35 * 1 abcdef
36   2 ghi
37   3 jklm
38   4 jkj
39   5 jklasj
40   6 jklajs;lkfj
41   7 qiowuioj
42   8 lkjw;lkj
43   9 qwklejklj;c
44   10 jkaljsoiu
45      12345678901234567890
46 Enter command: d
47
48      *
49      12345678901234567890
50 * 1 abcdef
51   2 ghi
52   3 jklm
53   4 jkj
54   5 jklasj
55   6 jklajs;lkfj
56   7 qiowuioj
57   8 lkjw;lkj
58   9 qwklejklj;c
59   10 jkaljsoiu
60      12345678901234567890
61 Enter command: d
62
63
64
65 [... omitting some parts in order to save electronic trees ...]
66
67
68
69      *
70      12345678901234567890
71 * 1 abcdef
72   2 ghi
73   3 jklm
74   4 jkj
75   5 jklasj
76   6 jklajs;lkfj
77   7 qiowuioj
78   8 lkjw;lkj
79   9 qwklejklj;c
80   10 jkaljsoiu
81      12345678901234567890
82 Enter command: d
83
84      *
85      12345678901234567890
86 * 1 abcdef
87   2 ghi
88   3 jklm
89   4 jkj
90   5 jklasj
91   6 jklajs;lkfj

```

```

92 7 qiowuioj
93 8 lkjw;lkj
94 9 qwklejklj;c
95 10 jkaljsoiu
96 12345678901234567890
97 Enter command: d
98
99 *
100 12345678901234567890
101 * 1 abcdef
102 2 ghi
103 3 jklm
104 4 jkj
105 5 jklasj
106 6 jklajs;lkfj
107 7 qiowuioj
108 8 lkjw;lkj
109 9 qwklejklj;c
110 10 jkaljsoiu
111 12345678901234567890
112 Enter command: d
113
114 *
115 12345678901234567890
116 * 1 abcdef
117 2 ghi
118 3 jklm
119 4 jkj
120 5 jklasj
121 6 jklajs;lkfj
122 7 qiowuioj
123 8 lkjw;lkj
124 9 qwklejklj;c
125 10 jkaljsoiu
126 12345678901234567890
127 Enter command: d
128
129 *
130 12345678901234567890
131 * 1 abcdef
132 2 ghi
133 3 jklm
134 4 jkj
135 5 jklasj
136 6 jklajs;lkfj
137 7 qiowuioj
138 8 lkjw;lkj
139 9 qwklejklj;c
140 10 jkaljsoiu
141 12345678901234567890
142 Enter command: d
143
144 *
145 12345678901234567890
146 * 1 abcdef
147 2 ghi
148 3 jklm
149 4 jkj
150 5 jklasj
151 6 jklajs;lkfj
152 7 qiowuioj
153 8 lkjw;lkj
154 9 qwklejklj;c
155 10 jkaljsoiu
156 12345678901234567890
157 Enter command: a
158
159 *
160 12345678901234567890
161 * 1 abcdef
162 2 ghi
163 3 jklm
164 4 jkj
165 5 jklasj

```

```

166 6 jklajs;lkfj
167 7 qiowuioj
168 8 lkjw;lkj
169 9 qwklejklj;c
170 10 jkaljsou
171 12345678901234567890
172 Enter command: q
173 >>>

```

### 5.5.2 Part #5.2 - Upward/Downward Movement and Scrolling the Window: `move_up()` and `move_down()`

As I said previously, I personally found this part to be the hardest (although that might be because I had to come up with all of the below rules by myself), and there will be test cases in parts #6 and #7 that won't involve upward/downward cursor movement, meaning that you can still get those test cases correct even if you do not finish this part.

When no scrolling occurs, up/down movement is simple: move the cursor up or down. When scrolling occurs, however, then the rules are more complicated. The following rules dictate how scrolling up and down should work. I based these rules off of 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 of the file, then scrolling cannot occur; otherwise, scrolling should proceed, with the cursor staying at the top of the window.
- **Scrolling down. Scrolling down can only occur when the user tries to move the cursor down while the cursor is at the *last line of the file*.** This difference means that you can view past the end of the file when scrolling down (although you cannot edit past the end of the file), but you can't view before the start of the file when scrolling up. Here are four scenarios.
  1. If the cursor is *not* at the last line of the file but *is* at the bottom of the window, then scroll down (without moving the cursor).
  2. If the cursor is at the last line of the file but *not* at the top of the window, then scroll down but move the cursor *up*. This will simulate the effect of scrolling down while keeping the cursor at the same position relative to the file's 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. (Recall from part #2 that the `Editor` initializer rejects empty files, so you need not worry about that case.)

Below is an example of how your code should behave after you have completed this part.

Input file:

```

1 Line 1
2 Line 2
3 Line 3
4 Line 4
5 Line 5
6 Line 6
7 Line 7
8 Line 8
9 Line 9
10 Line 10
11 Line 11
12 Line 12

```

Interpreter interaction:

```

1 >>> e = Editor('text_files/lines.txt', 'ignore')
2 >>> e.run()
3
4 12345678901234567890
5 * 1 Line 1
6   2 Line 2
7   3 Line 3
8   4 Line 4
9   5 Line 5
10  6 Line 6
11  7 Line 7
12  8 Line 8
13  9 Line 9
14 10 Line 10

```

```

15      12345678901234567890
16 Enter command: w
17
18      *
19      12345678901234567890
20 * 1 Line 1
21   2 Line 2
22   3 Line 3
23   4 Line 4
24   5 Line 5
25   6 Line 6
26   7 Line 7
27   8 Line 8
28   9 Line 9
29  10 Line 10
30      12345678901234567890
31 Enter command: d
32
33      *
34      12345678901234567890
35 * 1 Line 1
36   2 Line 2
37   3 Line 3
38   4 Line 4
39   5 Line 5
40   6 Line 6
41   7 Line 7
42   8 Line 8
43   9 Line 9
44  10 Line 10
45      12345678901234567890
46 Enter command: w
47
48      *
49      12345678901234567890
50 * 1 Line 1
51   2 Line 2
52   3 Line 3
53   4 Line 4
54   5 Line 5
55   6 Line 6
56   7 Line 7
57   8 Line 8
58   9 Line 9
59  10 Line 10
60      12345678901234567890
61 Enter command: s
62
63      *
64      12345678901234567890
65      1 Line 1
66 * 2 Line 2
67   3 Line 3
68   4 Line 4
69   5 Line 5
70   6 Line 6
71   7 Line 7
72   8 Line 8
73   9 Line 9
74  10 Line 10
75      12345678901234567890
76 Enter command: w
77
78      *
79      12345678901234567890
80 * 1 Line 1
81   2 Line 2
82   3 Line 3
83   4 Line 4
84   5 Line 5
85   6 Line 6
86   7 Line 7
87   8 Line 8
88   9 Line 9

```

```

89 10 Line 10
90 12345678901234567890
91 Enter command: s
92
93 *
94 12345678901234567890
95 1 Line 1
96 * 2 Line 2
97 3 Line 3
98 4 Line 4
99 5 Line 5
100 6 Line 6
101 7 Line 7
102 8 Line 8
103 9 Line 9
104 10 Line 10
105 12345678901234567890
106 Enter command: s
107
108 *
109 12345678901234567890
110 1 Line 1
111 2 Line 2
112 * 3 Line 3
113 4 Line 4
114 5 Line 5
115 6 Line 6
116 7 Line 7
117 8 Line 8
118 9 Line 9
119 10 Line 10
120 12345678901234567890
121 Enter command: s
122
123
124
125 [... omitting some parts in order to save electronic trees ...]
126
127
128
129 *
130 12345678901234567890
131 1 Line 1
132 2 Line 2
133 3 Line 3
134 4 Line 4
135 5 Line 5
136 6 Line 6
137 7 Line 7
138 * 8 Line 8
139 9 Line 9
140 10 Line 10
141 12345678901234567890
142 Enter command: s
143
144 *
145 12345678901234567890
146 1 Line 1
147 2 Line 2
148 3 Line 3
149 4 Line 4
150 5 Line 5
151 6 Line 6
152 7 Line 7
153 8 Line 8
154 * 9 Line 9
155 10 Line 10
156 12345678901234567890
157 Enter command: s
158
159 *
160 12345678901234567890
161 1 Line 1
162 2 Line 2

```

```

163     3 Line 3
164     4 Line 4
165     5 Line 5
166     6 Line 6
167     7 Line 7
168     8 Line 8
169     9 Line 9
170 * 10 Line 10
171     12345678901234567890
172 Enter command: s
173
174     *
175     12345678901234567890
176     2 Line 2
177     3 Line 3
178     4 Line 4
179     5 Line 5
180     6 Line 6
181     7 Line 7
182     8 Line 8
183     9 Line 9
184     10 Line 10
185 * 11 Line 11
186     12345678901234567890
187 Enter command: s
188
189     *
190     12345678901234567890
191     3 Line 3
192     4 Line 4
193     5 Line 5
194     6 Line 6
195     7 Line 7
196     8 Line 8
197     9 Line 9
198     10 Line 10
199     11 Line 11
200 * 12 Line 12
201     12345678901234567890
202 Enter command: s
203
204     *
205     12345678901234567890
206     4 Line 4
207     5 Line 5
208     6 Line 6
209     7 Line 7
210     8 Line 8
211     9 Line 9
212     10 Line 10
213     11 Line 11
214 * 12 Line 12
215     13
216     12345678901234567890
217 Enter command: s
218
219     *
220     12345678901234567890
221     5 Line 5
222     6 Line 6
223     7 Line 7
224     8 Line 8
225     9 Line 9
226     10 Line 10
227     11 Line 11
228 * 12 Line 12
229     13
230     14
231     12345678901234567890
232 Enter command: s
233
234     *
235     12345678901234567890
236     6 Line 6

```

```

237 7 Line 7
238 8 Line 8
239 9 Line 9
240 10 Line 10
241 11 Line 11
242 * 12 Line 12
243 13
244 14
245 15
246 12345678901234567890
247 Enter command: s
248
249 *
250 12345678901234567890
251 7 Line 7
252 8 Line 8
253 9 Line 9
254 10 Line 10
255 11 Line 11
256 * 12 Line 12
257 13
258 14
259 15
260 16
261 12345678901234567890
262 Enter command: w
263
264 *
265 12345678901234567890
266 7 Line 7
267 8 Line 8
268 9 Line 9
269 10 Line 10
270 * 11 Line 11
271 12 Line 12
272 13
273 14
274 15
275 16
276 12345678901234567890
277 Enter command: w
278
279 *
280 12345678901234567890
281 7 Line 7
282 8 Line 8
283 9 Line 9
284 * 10 Line 10
285 11 Line 11
286 12 Line 12
287 13
288 14
289 15
290 16
291 12345678901234567890
292 Enter command: s
293
294 *
295 12345678901234567890
296 7 Line 7
297 8 Line 8
298 9 Line 9
299 10 Line 10
300 * 11 Line 11
301 12 Line 12
302 13
303 14
304 15
305 16
306 12345678901234567890
307 Enter command: s
308
309 *
310 12345678901234567890

```

```

311     7 Line 7
312     8 Line 8
313     9 Line 9
314    10 Line 10
315    11 Line 11
316 * 12 Line 12
317    13
318    14
319    15
320    16
321    12345678901234567890
322 Enter command: s
323
324     *
325    12345678901234567890
326     8 Line 8
327     9 Line 9
328    10 Line 10
329    11 Line 11
330 * 12 Line 12
331    13
332    14
333    15
334    16
335    17
336    12345678901234567890
337 Enter command: s
338
339     *
340    12345678901234567890
341     9 Line 9
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    12345678901234567890
352 Enter command: s
353
354     *
355    12345678901234567890
356    10 Line 10
357    11 Line 11
358 * 12 Line 12
359    13
360    14
361    15
362    16
363    17
364    18
365    19
366    12345678901234567890
367 Enter command: s
368
369     *
370    12345678901234567890
371    11 Line 11
372 * 12 Line 12
373    13
374    14
375    15
376    16
377    17
378    18
379    19
380    20
381    12345678901234567890
382 Enter command: s
383
384     *

```



```

385      12345678901234567890
386 * 12 Line 12
387      13
388      14
389      15
390      16
391      17
392      18
393      19
394      20
395      21
396      12345678901234567890
397 Enter command: w
398
399      *
400      12345678901234567890
401 * 11 Line 11
402      12 Line 12
403      13
404      14
405      15
406      16
407      17
408      18
409      19
410      20
411      12345678901234567890
412 Enter command: w
413
414      *
415      12345678901234567890
416 * 10 Line 10
417      11 Line 11
418      12 Line 12
419      13
420      14
421      15
422      16
423      17
424      18
425      19
426      12345678901234567890
427 Enter command: w
428
429      *
430      12345678901234567890
431 * 9 Line 9
432      10 Line 10
433      11 Line 11
434      12 Line 12
435      13
436      14
437      15
438      16
439      17
440      18
441      12345678901234567890
442 Enter command: w
443
444      *
445      12345678901234567890
446 * 8 Line 8
447      9 Line 9
448      10 Line 10
449      11 Line 11
450      12 Line 12
451      13
452      14
453      15
454      16
455      17
456      12345678901234567890
457 Enter command: s
458

```

```

459      *
460      12345678901234567890
461      8 Line 8
462 * 9 Line 9
463      10 Line 10
464      11 Line 11
465      12 Line 12
466      13
467      14
468      15
469      16
470      17
471      12345678901234567890
472 Enter command: s
473
474      *
475      12345678901234567890
476      8 Line 8
477      9 Line 9
478 * 10 Line 10
479      11 Line 11
480      12 Line 12
481      13
482      14
483      15
484      16
485      17
486      12345678901234567890
487 Enter command: w
488
489      *
490      12345678901234567890
491      8 Line 8
492 * 9 Line 9
493      10 Line 10
494      11 Line 11
495      12 Line 12
496      13
497      14
498      15
499      16
500      17
501      12345678901234567890
502 Enter command: w
503
504      *
505      12345678901234567890
506 * 8 Line 8
507      9 Line 9
508      10 Line 10
509      11 Line 11
510      12 Line 12
511      13
512      14
513      15
514      16
515      17
516      12345678901234567890
517 Enter command: w
518
519      *
520      12345678901234567890
521 * 7 Line 7
522      8 Line 8
523      9 Line 9
524      10 Line 10
525      11 Line 11
526      12 Line 12
527      13
528      14
529      15
530      16
531      12345678901234567890
532 Enter command: w

```

```

533
534      *
535      12345678901234567890
536 *   6 Line 6
537     7 Line 7
538     8 Line 8
539     9 Line 9
540    10 Line 10
541    11 Line 11
542    12 Line 12
543    13
544    14
545    15
546      12345678901234567890
547 Enter command: w
548
549 [... etc. ...]

```

## 5.6 Part #6 - Insertion: `insert()`

The `insert()` method should insert its argument at the appropriate part of the file contents, based on where the cursor is. See the below examples. (Note that this should not modify the output file at all yet; to be clear, if the user never enters the save command, then the output file is ignored.)

The `run()` method already detects if the user enters the insert command, and if the user does, the `run()` method parses whatever is after the insert command and sends it to the `insert()` method.

If the insertion is successful, then the `insert()` method should return `True`. If the insertion would extend past the end of the current line (the one that the cursor is on), then the function should return `False`; *you should avoid changing the file's currently depicted contents in this case*. Note that writes that span multiple lines are not supported in this editor, which also means that you cannot add lines. You *can*, however, extend an already-existing line (not past the width of the window, of course).

*Hint:* You may want to read part #7 before starting this part, because parts #6 and #7 are linked such that you may find you have to go back and modify your `insert()` method in order to do part #7.

Below is an example of how your code should behave after finishing this part. This example does not involve cursor movement (i.e. part #5).

Input file:

```

1 Line 1
2 Line 2
3 Line 3
4 Line 4
5 Line 5
6 Line 6
7 Line 7
8 Line 8
9 Line 9
10 Line 10
11 Line 11
12 Line 12

```

Interpreter interaction:

```

1 >>> e = Editor('text_files/lines.txt', 'output.txt')
2 >>> e.run()
3      *
4      12345678901234567890
5 *   1 Line 1
6     2 Line 2
7     3 Line 3
8     4 Line 4
9     5 Line 5
10    6 Line 6
11    7 Line 7
12    8 Line 8
13    9 Line 9
14   10 Line 10
15      12345678901234567890
16 Enter command: i XYZ
17
18      *
19      12345678901234567890

```

```

20 * 1 XYZe 1
21 2 Line 2
22 3 Line 3
23 4 Line 4
24 5 Line 5
25 6 Line 6
26 7 Line 7
27 8 Line 8
28 9 Line 9
29 10 Line 10
30 12345678901234567890
31 Enter command: e
32
33 *
34 12345678901234567890
35 * 1 XYZe 1
36 2 Line 2
37 3 Line 3
38 4 Line 4
39 5 Line 5
40 6 Line 6
41 7 Line 7
42 8 Line 8
43 9 Line 9
44 10 Line 10
45 12345678901234567890
46 Enter command: q
47 >>>

```

Resulting output file:

```

1 XYZe 1
2 Line 2
3 Line 3
4 Line 4
5 Line 5
6 Line 6
7 Line 7
8 Line 8
9 Line 9
10 Line 10
11 Line 11
12 Line 12

```

Below is another example. This example *does* involve cursor movement.

Input file:

```

1 Line 1
2 Line 2
3 Line 3
4 Line 4
5 Line 5
6 Line 6
7 Line 7
8 Line 8
9 Line 9
10 Line 10
11 Line 11
12 Line 12

```

Interpreter interaction:

```

1 >>> e = Editor('text_files/lines.txt', 'output.txt')
2 >>> e.run()
3 *
4 12345678901234567890
5 * 1 Line 1
6 2 Line 2
7 3 Line 3
8 4 Line 4
9 5 Line 5
10 6 Line 6
11 7 Line 7
12 8 Line 8
13 9 Line 9

```

```

14 10 Line 10
15 12345678901234567890
16 Enter command: s
17
18 *
19 12345678901234567890
20 1 Line 1
21 * 2 Line 2
22 3 Line 3
23 4 Line 4
24 5 Line 5
25 6 Line 6
26 7 Line 7
27 8 Line 8
28 9 Line 9
29 10 Line 10
30 12345678901234567890
31 Enter command: d
32
33 *
34 12345678901234567890
35 1 Line 1
36 * 2 Line 2
37 3 Line 3
38 4 Line 4
39 5 Line 5
40 6 Line 6
41 7 Line 7
42 8 Line 8
43 9 Line 9
44 10 Line 10
45 12345678901234567890
46 Enter command: d
47
48 *
49 12345678901234567890
50 1 Line 1
51 * 2 Line 2
52 3 Line 3
53 4 Line 4
54 5 Line 5
55 6 Line 6
56 7 Line 7
57 8 Line 8
58 9 Line 9
59 10 Line 10
60 12345678901234567890
61 Enter command: i abracadabra
62
63 *
64 12345678901234567890
65 1 Line 1
66 * 2 Liabracadabra
67 3 Line 3
68 4 Line 4
69 5 Line 5
70 6 Line 6
71 7 Line 7
72 8 Line 8
73 9 Line 9
74 10 Line 10
75 12345678901234567890
76 Enter command: s
77
78 *
79 12345678901234567890
80 1 Line 1
81 2 Liabracadabra
82 * 3 Line 3
83 4 Line 4
84 5 Line 5
85 6 Line 6
86 7 Line 7
87 8 Line 8

```

```

88     9 Line 9
89    10 Line 10
90      12345678901234567890
91 Enter command: s
92
93      *
94      12345678901234567890
95     1 Line 1
96     2 Liabracadabra
97     3 Line 3
98 *   4 Line 4
99     5 Line 5
100    6 Line 6
101    7 Line 7
102    8 Line 8
103    9 Line 9
104   10 Line 10
105      12345678901234567890
106 Enter command: d
107
108      *
109      12345678901234567890
110     1 Line 1
111     2 Liabracadabra
112     3 Line 3
113 *   4 Line 4
114     5 Line 5
115     6 Line 6
116     7 Line 7
117     8 Line 8
118     9 Line 9
119    10 Line 10
120      12345678901234567890
121 Enter command: d
122
123      *
124      12345678901234567890
125     1 Line 1
126     2 Liabracadabra
127     3 Line 3
128 *   4 Line 4
129     5 Line 5
130     6 Line 6
131     7 Line 7
132     8 Line 8
133     9 Line 9
134    10 Line 10
135      12345678901234567890
136 Enter command: d
137
138      *
139      12345678901234567890
140     1 Line 1
141     2 Liabracadabra
142     3 Line 3
143 *   4 Line 4
144     5 Line 5
145     6 Line 6
146     7 Line 7
147     8 Line 8
148     9 Line 9
149    10 Line 10
150      12345678901234567890
151 Enter command: d
152
153      *
154      12345678901234567890
155     1 Line 1
156     2 Liabracadabra
157     3 Line 3
158 *   4 Line 4
159     5 Line 5
160     6 Line 6
161     7 Line 7

```

```

162      8 Line 8
163      9 Line 9
164     10 Line 10
165     12345678901234567890
166 Enter command: d
167
168      *
169     12345678901234567890
170      1 Line 1
171      2 Liabracadabra
172      3 Line 3
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     12345678901234567890
181 Enter command: d
182
183      *
184     12345678901234567890
185      1 Line 1
186      2 Liabracadabra
187      3 Line 3
188 *    4 Line 4
189      5 Line 5
190      6 Line 6
191      7 Line 7
192      8 Line 8
193      9 Line 9
194     10 Line 10
195     12345678901234567890
196 Enter command: i blah
197
198      *
199     12345678901234567890
200      1 Line 1
201      2 Liabracadabra
202      3 Line 3
203 *    4 Line 4   blah
204      5 Line 5
205      6 Line 6
206      7 Line 7
207      8 Line 8
208      9 Line 9
209     10 Line 10
210     12345678901234567890
211 Enter command: e
212
213      *
214     12345678901234567890
215      1 Line 1
216      2 Liabracadabra
217      3 Line 3
218 *    4 Line 4   blah
219      5 Line 5
220      6 Line 6
221      7 Line 7
222      8 Line 8
223      9 Line 9
224     10 Line 10
225     12345678901234567890
226 Enter command: q
227 >>>

```

Resulting output file:

```

1 Line 1
2 Liabracadabra
3 Line 3
4 Line 4   blah
5 Line 5
6 Line 6

```

```
7 Line 7
8 Line 8
9 Line 9
10 Line 10
11 Line 11
12 Line 12
```

Below is an example where the insertion would extend past the end of the file. You do not need to print the error message, as the `run()` method already does this; your `insert()` method merely needs to return the correct boolean value.

```
1 ...
2
3      *
4      12345678901234567890
5 * 1 abcdef
6   2 ghi
7   3 jklm
8   4 jkj
9   5 jklasj
10  6 jklajs;lkfj
11  7 qiowuioj
12  8 lkjw;lkj
13  9 qwklejklj;c
14 10 jkaljsoiu
15      12345678901234567890
16 Enter command: i akjsdlk;fjasklfjlkja;jfl;kj
17
18 ERROR: Invalid write.
19
20      *
21      12345678901234567890
22 * 1 abcdef
23   2 ghi
24   3 jklm
25   4 jkj
26   5 jklasj
27   6 jklajs;lkfj
28   7 qiowuioj
29   8 lkjw;lkj
30   9 qwklejklj;c
31  10 jkaljsoiu
32      12345678901234567890
33 Enter command: q
34 >>>
```

## 5.7 Part #7 - Edit History

In this part, you will modify the editor so that insertions can be undone/redone. Note that insertion is the only operation that this applies to; operations like cursor movements and saving cannot be undone/redone.

Below are some implementation tips and/or restrictions that apply to both parts #7.1 and #7.2 but that may not make sense until you read the directions for those parts.

- As you can see in the Editor initializer as it was provided on Canvas, the `undo_history` and `redo_history` members are set to instances of `class Stack`. You are not allowed to change this. You must create a class used to represent an insertion operation. (If you want, you can have a class representing an undo-able insertion operation and a separate class representing a redo-able insertion operation.) This class / these classes should have enough information for you to undo or redo the insertion. Your stacks must only ever contain instances of this class / these classes. You are *not* allowed to store entire file buffers/contents in the stacks (i.e. do not store 2D lists in your stacks), as this would be unacceptably inefficient in terms of memory.
- My own implementations of `undo()` and `redo()` are about 10 lines each.

### 5.7.1 Part #7.1 - Undoing an Insertion: `undo()`

When the undo key is entered, the `run()` method calls the `undo()` method. Implement the `undo()` method so that calling it completely undoes the last insertion that was done. Your editor must preserve an undo history that allows the user to undo as many operations as they wish, until there are no more operations to undo.

To be clear, there are only two ways in which an operation can be *added* to the undo history:



1. An insertion is done (i.e. part #6).
2. An insertion is redone (i.e. part #7.2).

The function should return `False` if there are no commands to undo, in which case the `run()` method will already know to print the appropriate message. Otherwise, the function should return `True`. Note that an infinitely long<sup>3</sup> undo history is supported.

Below are examples of how undoing an insertion should behave.

```
1 >>> e = Editor('text_files/lines.txt', 'ignore')
2 >>> e.run()
3
4      *
5      12345678901234567890
6
7 * 1 Line 1
8   2 Line 2
9   3 Line 3
10  4 Line 4
11  5 Line 5
12  6 Line 6
13  7 Line 7
14  8 Line 8
15  9 Line 9
16 10 Line 10
17      12345678901234567890
18 Enter command: i abc
19
20      *
21      12345678901234567890
22
23 * 1 abce 1
24   2 Line 2
25   3 Line 3
26   4 Line 4
27   5 Line 5
28   6 Line 6
29   7 Line 7
30   8 Line 8
31   9 Line 9
32  10 Line 10
33      12345678901234567890
34 Enter command: u
35
36      *
37      12345678901234567890
38
39 * 1 Line 1
40   2 Line 2
41   3 Line 3
42   4 Line 4
43   5 Line 5
44   6 Line 6
45   7 Line 7
46   8 Line 8
47   9 Line 9
48  10 Line 10
49      12345678901234567890
50 Enter command: u
51
52 ERROR: No operation to undo.
53
54      *
55      12345678901234567890
56
57 * 1 Line 1
58   2 Line 2
59   3 Line 3
60   4 Line 4
61   5 Line 5
62   6 Line 6
63   7 Line 7
64   8 Line 8
65   9 Line 9
66  10 Line 10
67      12345678901234567890
68 Enter command: s
```

---

<sup>3</sup>Or, at least, as long as the amount of memory that Python will let you use.

```

64
65      *
66      12345678901234567890
67      1 Line 1
68 *    2 Line 2
69      3 Line 3
70      4 Line 4
71      5 Line 5
72      6 Line 6
73      7 Line 7
74      8 Line 8
75      9 Line 9
76      10 Line 10
77      12345678901234567890
78 Enter command: i abcd
79
80      *
81      12345678901234567890
82      1 Line 1
83 *    2 abcd 2
84      3 Line 3
85      4 Line 4
86      5 Line 5
87      6 Line 6
88      7 Line 7
89      8 Line 8
90      9 Line 9
91      10 Line 10
92      12345678901234567890
93 Enter command: i XYZ
94
95      *
96      12345678901234567890
97      1 Line 1
98 *    2 XYZd 2
99      3 Line 3
100     4 Line 4
101     5 Line 5
102     6 Line 6
103     7 Line 7
104     8 Line 8
105     9 Line 9
106     10 Line 10
107     12345678901234567890
108 Enter command: u
109
110     *
111     12345678901234567890
112     1 Line 1
113 *    2 abcd 2
114     3 Line 3
115     4 Line 4
116     5 Line 5
117     6 Line 6
118     7 Line 7
119     8 Line 8
120     9 Line 9
121     10 Line 10
122     12345678901234567890
123 Enter command: u
124
125     *
126     12345678901234567890
127     1 Line 1
128 *    2 Line 2
129     3 Line 3
130     4 Line 4
131     5 Line 5
132     6 Line 6
133     7 Line 7
134     8 Line 8
135     9 Line 9
136     10 Line 10
137     12345678901234567890

```

```
138 Enter command: q
139 >>>
```

### 5.7.2 Part #7.2 - Redoing an Insertion: redo()

Implement the `redo()` method so that it redoes (re-does?) the last operation that was added to the redo history. The only reason that an operation should be added to the redo history is if an operation was undone; any undone insertion should be added to the redo history. *If the user does a new insertion (i.e. as in part #6), then the entire redo history should be erased. This is similar to the behavior of typical editors.*

The function should return `False` if there are no commands to redo, in which case the `run()` method will already know to print the appropriate message. Otherwise, the function should return `True`. An infinitely long redo history is supported.

Below are examples of how redoing an insertion should behave.

```
1 >>> e = Editor('text_files/lines.txt', 'ignore')
2 >>> e.run()
3
4      *
5      12345678901234567890
6
7 * 1 Line 1
8   2 Line 2
9   3 Line 3
10  4 Line 4
11  5 Line 5
12  6 Line 6
13  7 Line 7
14  8 Line 8
15  9 Line 9
16 10 Line 10
17      12345678901234567890
18 Enter command: i hi there friend
19
20      *
21      12345678901234567890
22
23 * 1 hi there friend
24   2 Line 2
25   3 Line 3
26   4 Line 4
27   5 Line 5
28   6 Line 6
29   7 Line 7
30   8 Line 8
31   9 Line 9
32  10 Line 10
33      12345678901234567890
34 Enter command: u
35
36      *
37      12345678901234567890
38
39 * 1 Line 1
40   2 Line 2
41   3 Line 3
42   4 Line 4
43   5 Line 5
44   6 Line 6
45   7 Line 7
46   8 Line 8
47   9 Line 9
48  10 Line 10
49      12345678901234567890
50 Enter command: r
51
52      *
53      12345678901234567890
54
55 * 1 hi there friend
56   2 Line 2
57   3 Line 3
58   4 Line 4
59   5 Line 5
60   6 Line 6
61   7 Line 7
62   8 Line 8
63   9 Line 9
```

```

59 10 Line 10
60 12345678901234567890
61 Enter command: r
62
63 ERROR: No operation to redo.
64
65 *
66 12345678901234567890
67 * 1 hi there friend
68 2 Line 2
69 3 Line 3
70 4 Line 4
71 5 Line 5
72 6 Line 6
73 7 Line 7
74 8 Line 8
75 9 Line 9
76 10 Line 10
77 12345678901234567890
78 Enter command: u
79
80 [... skipping many cursor movements ...]
81
82 *
83 12345678901234567890
84 5 Line 5
85 6 Line 6
86 7 Line 7
87 8 Line 8
88 9 Line 9
89 10 Line 10
90 11 Line 11
91 * 12 Line 12
92 13
93 14
94 12345678901234567890
95 Enter command: d
96
97 *
98 12345678901234567890
99 5 Line 5
100 6 Line 6
101 7 Line 7
102 8 Line 8
103 9 Line 9
104 10 Line 10
105 11 Line 11
106 * 12 Line 12
107 13
108 14
109 12345678901234567890
110 Enter command: i blah blah
111
112 *
113 12345678901234567890
114 5 Line 5
115 6 Line 6
116 7 Line 7
117 8 Line 8
118 9 Line 9
119 10 Line 10
120 11 Line 11
121 * 12 Linblah blah
122 13
123 14
124 12345678901234567890
125 Enter command: w
126
127 *
128 12345678901234567890
129 5 Line 5
130 6 Line 6
131 7 Line 7
132 8 Line 8

```

```

133     9 Line 9
134    10 Line 10
135 * 11 Line 11
136    12 Linblah blah
137     13
138     14
139    12345678901234567890
140 Enter command: w
141
142     *
143    12345678901234567890
144     5 Line 5
145     6 Line 6
146     7 Line 7
147     8 Line 8
148     9 Line 9
149 * 10 Line 10
150    11 Line 11
151    12 Linblah blah
152     13
153     14
154    12345678901234567890
155 Enter command: d
156
157     *
158    12345678901234567890
159     5 Line 5
160     6 Line 6
161     7 Line 7
162     8 Line 8
163     9 Line 9
164 * 10 Line 10
165    11 Line 11
166    12 Linblah blah
167     13
168     14
169    12345678901234567890
170 Enter command: i haha
171
172     *
173    12345678901234567890
174     5 Line 5
175     6 Line 6
176     7 Line 7
177     8 Line 8
178     9 Line 9
179 * 10 Linehaha
180    11 Line 11
181    12 Linblah blah
182     13
183     14
184    12345678901234567890
185 Enter command: w
186
187     *
188    12345678901234567890
189     5 Line 5
190     6 Line 6
191     7 Line 7
192     8 Line 8
193 * 9 Line 9
194    10 Linehaha
195    11 Line 11
196    12 Linblah blah
197     13
198     14
199    12345678901234567890
200 Enter command: i abcd
201
202     *
203    12345678901234567890
204     5 Line 5
205     6 Line 6
206     7 Line 7

```

```

207      8 Line 8
208 *    9 Lineabcd
209      10 Linehaha
210      11 Line 11
211      12 Linblah blah
212      13
213      14
214      12345678901234567890
215 Enter command: i XYZ
216
217      *
218      12345678901234567890
219      5 Line 5
220      6 Line 6
221      7 Line 7
222      8 Line 8
223 *    9 LineXYZd
224      10 Linehaha
225      11 Line 11
226      12 Linblah blah
227      13
228      14
229      12345678901234567890
230 Enter command: u
231
232      *
233      12345678901234567890
234      5 Line 5
235      6 Line 6
236      7 Line 7
237      8 Line 8
238 *    9 Lineabcd
239      10 Linehaha
240      11 Line 11
241      12 Linblah blah
242      13
243      14
244      12345678901234567890
245 Enter command: u
246
247      *
248      12345678901234567890
249      5 Line 5
250      6 Line 6
251      7 Line 7
252      8 Line 8
253 *    9 Line 9
254      10 Linehaha
255      11 Line 11
256      12 Linblah blah
257      13
258      14
259      12345678901234567890
260 Enter command: r
261
262      *
263      12345678901234567890
264      5 Line 5
265      6 Line 6
266      7 Line 7
267      8 Line 8
268 *    9 Lineabcd
269      10 Linehaha
270      11 Line 11
271      12 Linblah blah
272      13
273      14
274      12345678901234567890
275 Enter command: r
276
277      *
278      12345678901234567890
279      5 Line 5
280      6 Line 6

```

```

281 7 Line 7
282 8 Line 8
283 * 9 LineXYZd
284 10 Linehaha
285 11 Line 11
286 12 Linblah blah
287 13
288 14
289 12345678901234567890
290 Enter command: u
291
292 *
293 12345678901234567890
294 5 Line 5
295 6 Line 6
296 7 Line 7
297 8 Line 8
298 * 9 Lineabcd
299 10 Linehaha
300 11 Line 11
301 12 Linblah blah
302 13
303 14
304 12345678901234567890
305 Enter command: i :(
306
307 *
308 12345678901234567890
309 5 Line 5
310 6 Line 6
311 7 Line 7
312 8 Line 8
313 * 9 Line:(cd
314 10 Linehaha
315 11 Line 11
316 12 Linblah blah
317 13
318 14
319 12345678901234567890
320 Enter command: r
321
322 ERROR: No operation to redo.
323
324 *
325 12345678901234567890
326 5 Line 5
327 6 Line 6
328 7 Line 7
329 8 Line 8
330 * 9 Line:(cd
331 10 Linehaha
332 11 Line 11
333 12 Linblah blah
334 13
335 14
336 12345678901234567890
337 Enter command: q
338 >>>

```