Project

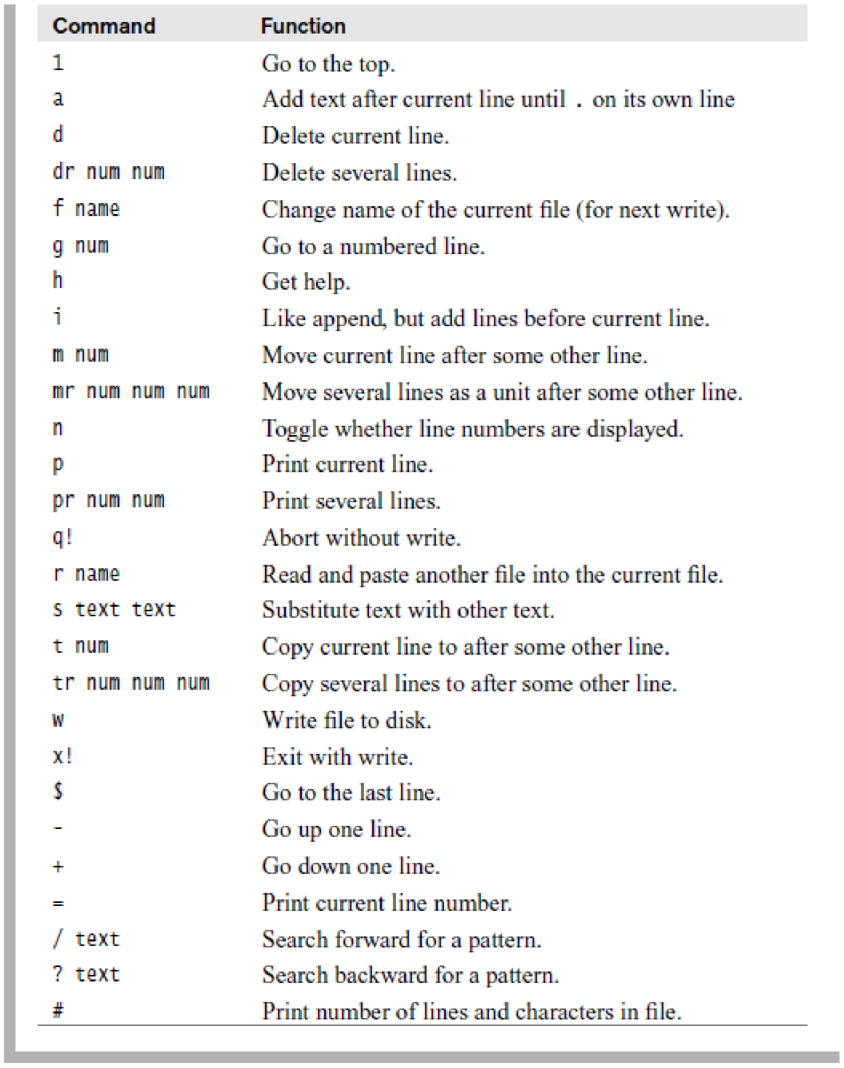
Exercise 1

Write a line-based text editor.

- The command syntax is similar to the Unix line editor ed.
- The internal copy of the file is maintained as a linked list of lines.
- To be able to go up and down in the file, you have to maintain a doubly linked list. Most commands are represented by a one-character string.
- Some are two characters and require an argument (or two).

figure 17.31

Commands for editor in Exercise 17.19



Command	Function
`1`	Go to the top.
`a`	Add text after current line until `.` on its own line
`d`	Delete current line.
`dr num num`	Delete several lines.
`f name`	Change name of the current file (for next write).
`g num`	Go to a numbered line.
`h`	Get help.
`i`	Like append, but add lines before current line.
`m num`	Move current line after some other line.
`mr num num num`	Move several lines as a unit after some other line.
`n`	Toggle whether line numbers are displayed.
`p`	Print current line.
`pr num num`	Print several lines.
`q!`	Abort without write.
`r name`	Read and paste another file into the current file.
`s text text`	Substitute text with other text.
`t num`	Copy current line to after some other line.
`tr num num num`	Copy several lines to after some other line.
`w`	Write file to disk.
`x!`	Exit with write.
`$`	Go to the last line.
`-`	Go up one line.
`+`	Go down one line.
`=`	Print current line number.
`/ text`	Search forward for a pattern.
`? text`	Search backward for a pattern.
`#`	Print number of lines and characters in file.

Exercise 2

An output-restricted double-ended queue supports insertions from both ends but accesses and deletions only from the front. Implement this data structure with a singly linked list.

Exercise 3

Implement a binary search tree to allow duplicates.

- Hint: Have each node stores data structures that are considered duplicates (using the first item in this structure) to control branching.

In addition to the existing operators of the tree, implement also the following:

- Return number of duplicates of an element
- Find and replace all duplicates of element A with element B
- Show all the trees together with the duplicates (pre-order, in-order, post-order, level-order
- Remove all existing duplicates (leaving only one copy) of an element
- Remove only one copy of the existing duplicates of an element
- Remove all existing duplicates (leaving only one copy) for all the duplicated elements in tree
- Print all elements that have duplicates together with the number of duplicates
- Print the number of all the duplicates in the tree
- Show only the nodes that have/do not have duplicates

Exercise 4

Suppose you manage a restaurant that has a promotion for loyal customers. Every month, you need to send an offer to a group of customers based on the amount they have spent at your restaurant. For example, in one month, customers who have spent 50 –100 Euro will receive a free drink coupon, and in another month, customers who have spent 100–500 Euro will receive a 50% discount coupon. The spending ranges may change every month, and you need to be able to identify the group of customers to whom you will send the offer each month. What type of data structure would you use to solve this problem? Implement the solution.

**One solution is to use linear probing closed hash to resolve collisions. The hash table key will be the amount a customer spent last month at your restaurant. The value will represent the customer, and the hash function should preserve the order of the keys. The algorithm to find the correct group of customers for an X–Y euro interval: perform a binary search on the keys in the hash table and continue until the smallest key ≥ X is found. Then, start linear iteration through the hash table according to the order of the keys. Examine each key and add the corresponding customer (or list of customers, in case two customers spent the same amount). When a key larger than Y is found, stop the iteration and return the list of found customers.*