FIT1008 Introduction to Computer Science
Practical Session 9

Semester 2, 2014

Objectives of this practical session

The aim of this prac is to gain experience in implementing, using and modifying linked lists. In order to do this, you will write a line-oriented text editor.

Testing

For this prac, you are required to write:

(1) a *function to test* each function you implement, and

(2) *at least two test cases* per function.

The cases need to show that your functions can handle both valid and invalid inputs.

File I/O

Important: Before doing this prac, you need to learn a bit about file I/O. A useful resource for you is:

- `http://docs.python.org/3.3/tutorial/inputoutput.html#reading-and-writing-files` from *The Python Tutorial*

Background

The editor `ed` was one of the first editors written for UNIX. In this prac we will use a linked list to implement a version of a line-oriented text editor based on `ed`. The text editor `ed` is very similar to the common UNIX text editors `vi` and `vim` (it is in fact their predecessor). To find out more about `ed`, you can read the man page (by typing: `man ed` into a linux or MacOS X terminal), or visit, for example:
`http://roguelife.org/~fujita/COOKIES/HISTORY/V6/ed.1.html`

Important: Our commands will be different from the `ed` commands.

To implement a simple line-oriented text editor, the idea is as follows. Suppose a file contains the following lines:

```
Yossarian decided
not to utter
another word.
```

where the string "Yossarian decided" is considered to be in line 0, "not to utter" is considered to be in line 1, etc. We want to store every line in the file in a data type that allows users to easily manipulate (delete/add/print) any line by simply providing the line number they want to modify. This means we should use a list data type (**as opposed to a stack or a queue**). Now, since we do not know how many lines there are in a file, and we want to efficiently add/delete elements of that list, we should use a linked data structure (to avoid re-sizing and shuffling). With the above input, the data structure would look like the diagram in Figure 1.

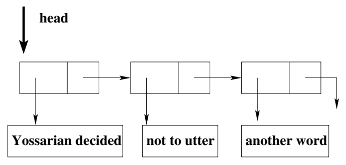

Figure 1: Example of a linked list of Strings

Knowing the position of each line allows users to manipulate (delete/add) lines without having to provide the entire string. It also allows us to locate them without having to compare the strings.

Task 1 [4 marks]

Write a Python program that implements a Linked List, and allows a user to perform the following commands on the Linked List using a menu:

insert num: which inserts a line of text in the Linked List before position *num*, and raises an exception if no *num* is given. You need to handle negative indices in the same way that Python does.

delete num: which deletes the line of text in the Linked List at position *num*, and deletes all the lines if no *num* is given. Also it raises an exception if the list is empty or the index is out of range.

print num: which prints the line at position *num*, and if no *num* is given prints all the lines.

quit: which quits the program.

Task 2 [2 marks]

Write a Python program that allows a user to perform the following commands using a menu:

read filename: which opens the file, *filename*, reads all the lines in from the file, put each line as a separate item into a linked list, and then closes the file.

write filename: which opens a file, *filename*, writes every item in the linked list into the file, and then closes the file.

print num: which prints the line at position *num*, and if no *num* is given prints all the lines.

delete num: which deletes the line of text in the Linked List at position *num*, and deletes all the lines if no *num* is given.

quit: which quits the program.

Important: All errors should be caught and a question mark, **?**, should be printed when an error occurs.
Tip: When you are testing your code, it is handy to have a source of reasonably large text files that you can use as test data. There is a huge repository of public domain text files at Project Gutenberg's websites. The Australian Project Gutenberg repository is at `http://gutenberg.net.au`. You are encouraged to download a couple of ebooks from here and use them to make sure your code can deal with large files. Be sure to download plain-text format, though – your program is not expected to deal with other formats.

Task 3 [4 marks]

(i) Modify the program you wrote in Task 2, so that instead of using a menu it now allows the user to type the following commands:

r filename which opens the file, *filename*, reads all the lines in from the file, puts each line as a separate item into a linked list, and then closes the file.

w filename which opens a file, *filename*, writes every item in the linked list into the file, and then closes the file.

p num which prints the line at position *num*, and if no *num* is given prints all the lines.

d num which deletes the line of text in the Linked List at position *num*, and deletes all the lines if no *num* is given.

q which quits the program.

h which lists the available commands.

(ii) Now include the following commands:

i num which inserts text, before the line at position *num*. The text is read in from the screen one line at a time until a user types a period on a line by itself. The program should handle negative values of *num* in the same way Python does for negative indices in lists.

a which is equivalent to *i num*, where *num* is the number of lines stored in the linked list.

Advanced Question [Bonus 2 marks]

Extend your program for Task 3, to include the following commands:

g word which prints out the line number and the line, for every line containing the string *word*.

s word new_word which replaces *word* in every line by *new_word*.

Hall of Fame

It is useful to be able to move around the text that you are editing. Due to histronical reasons[1] we will use the command *j* to move down one line, and *k* to move up one line. This means you will need to be able to traverse your Linked List forwards and backwards, and keep track of the current line. So, you will need to change your implementation to a Double Linked List, where each node points to the previous node as well as the next node.

Also, modify the other commands so that they update the current line, and print the current line number and line when a user types a period as a command.

Finally, modify commands *p* and *d* so you can have ranges of lines represented by *n:m*, e.g.,

d n:m

will delete all the lines numbered *n* to *m-1*.

[1] http://www.catonmat.net/blog/why-vim-uses-hjkl-as-arrow-keys/