

School of Computer Science and Software Engineering
Clayton Campus, Monash University
CSE1303 Part A
Summer Semester, 2002

Practical Session 4: Linked Lists



Aims of this Practical Session

1. To familiarise yourself with Linked Lists and how to implement them.
2. To practice allocating and deallocating dynamic memory.
3. To implement a basic text editor

Project: To write a simple version of **ed**, a line-oriented text editor.

Background:

The editor **ed** was one of the first editors written for UNIX. In this prac we will use a Linked List to implement our simple version of **ed**. **ed** is very similar to, and in fact is the predecessor of **vi** and **vim**.

To find out more about **ed**, either read the man page (by typing: `man ed` at a linux prompt), or the book “**Software Tools**”, by Brian Kernighan and P.J. Plauger.

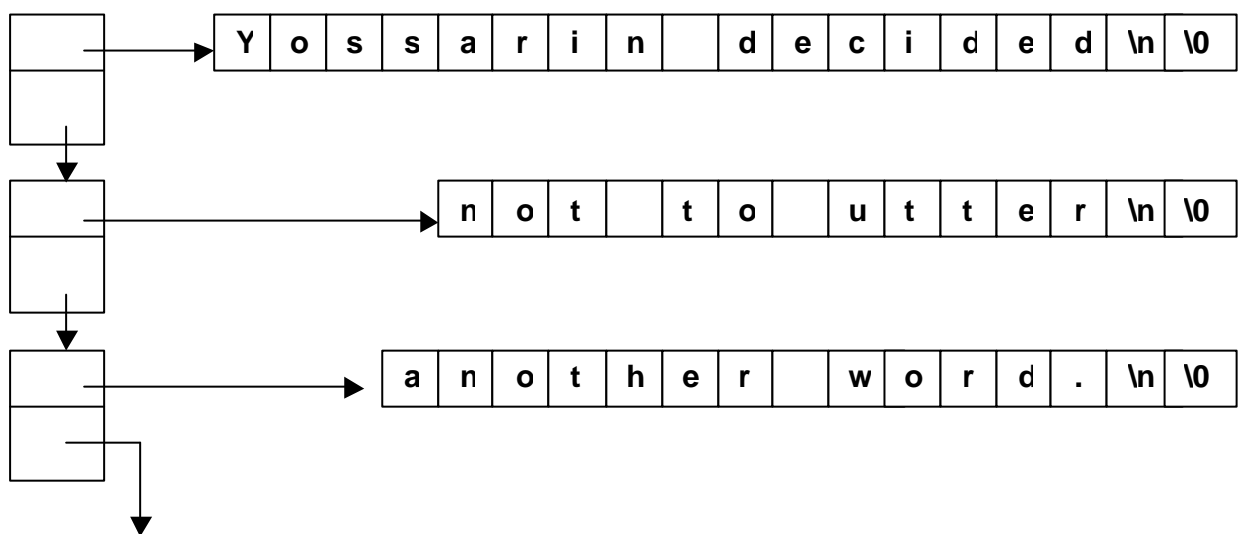
The courseware notes for the subject CSE1402 also discuss **vi/vim** in some depth, these are available at:
<http://www.csse.monash.edu.au/courseware/cse1402/>.

To implement a simple text editor, the idea is as follows;

Suppose a file contains the following lines:

**Yossarin decided
not to utter
another word.**

Then the addresses of copies of these lines would be stored in a Linked List, so that:



Knowing the starting address of each line allows you to manipulate the individual strings/lines (delete words/add words/substitute words). It also allows you to change the order of whole lines easily by manipulating the linked list. You can search for words or lines, you can duplicate lines, etc, which are all things that a text editor needs to be able to do.

Preparation (3 marks if completed before class)

Before this prac:

1. Modify the code for a Linked Lists given in lectures, so that instead of holding numbers of type **float** it holds **address** of a character string.
 - ⊕ **You must use the function `strdup` to make a copy of each character string in function `makeNode`. If your string library does not contain a copy of `strdup`, use the one given in the solutions for Tute 3.**
2. Write a C program which uses a Linked List to store character strings, and has a menu with the following options:
 - **Read** which asks the user for a character string, and puts the character string at the end of the Linked List.
 - **Print**: which asks a user for a number and prints the character string whose address is contained in that position in the Linked List.
 - **Quit** which allows the user to quit the menu and exit the program.
 - ⊕ **You may assume all character strings have at most 300 characters. However, character strings may contain blanks, tabs and a newline character, so you will need to use the function `fgets` to read in the character string.**

Question 1: (3 marks)

Write C program which uses a Linked List and has a menu with the following options:

- **Read** which asks the user for a file name. Opens the file. Reads all the lines in the file and stores them in the Linked List, one line per node. Then closes the file.
- **Write**: which asks a user for a file name. Opens the file. Prints all the lines stored in the Linked List into the file. Then closes the file.
- **Print** which asks a user for a number, **n**, and prints to **stdout** the line contained in position **n-1** in the Linked List.
- **Quit**: which allows the user to quit the menu and exit the program.

Question 2: (2 marks)

Rewrite the program you wrote in Question 1 so that it has the following properties:

- Do not ask the user for input.
- The program must be able to handle invalid input without exiting.
- Instead of a menu the user needs to type one of the following commands

sr file

Which opens **file**. Reads all the lines in **file** and stores them at the end of the Linked List, one line per node. Then closes **file**.

w file

Which opens **file** and prints all the lines stored in the Linked List **file**. Then closes **file**.

number p

Which prints to **stdout** the line in position **number-1**.

q

Which allows the user to exit the program.

Note: You should use `fgets` to read the command, then `sscanf` to convert to the actual commands, because of the 3rd command above and Question 3.

- ⊕ Your program **must not** prompt the user for a command.
- ⊕ For **any invalid input** your program **must not exit** but print a sensible error message. Note you will probably need to change the code you wrote for the Linked List.

Question 3: (2 marks)

Extend the program you wrote in Question 2 so that the user can type one of the following commands.

number d

Which deletes the line at position **number-1**.

number a

Which inserts, immediately after position **number-1** in the Linked List, any text subsequently written by the user. The text is inserted one line per node, until the user types at the start of line the character string “. \n”.

- ⊕ Do not store the line “. \n” in the Linked List.

Advanced Questions: (2 marks)

The commands, such as **\$r** and **number a**, can be implemented by doing repeated calls to the function **insertItem**. However, there are more efficient methods. Two possibilities are:

1. Keep track of the current position and the corresponding node, and to utilise this information when inserting items and deleting nodes.
2. Write and use the function

```
Node* insertNext(Node* nodePtr, char* s);
```

which inserts an address of a copy of a character string in a new node in the Link List after the node pointed to by **nodePtr**, and returns the address of the new node.

Questions:

- Extend the program so that the user can type one of the following commands.

n₁, n₂ d

Which deletes the lines from position **n₁-1** to position **n₂-1** inclusive.

n₁, n₂ p

Which prints the lines from position **n₁-1** to position **n₂-1** inclusive.

n₁, n₂ m n₃

Which moves the lines from position **n₁-1** to position **n₂-1** inclusive, to position **n₃** to position **n₃ + n₂ - n₁** inclusive.