University of Toronto
APS105H1F — Computer Fundamentals
Lab 8: Pointers, Dynamic Allocation & Linked Lists

Introduction

In this lab we will solve the same problem as in Lab 6, but using different programming techniques. In particular, you will get experience with:

- dynamic allocation of individual variables,
- dynamic allocation of arrays,
- deleting previously allocated arrays and variables,
- defining and using `struct` datatypes,
- declaring and using pointer variables,
- inserting elements into a linked list,
- printing a linked list,
- finding elements in a linked list, and
- deleting memory allocated by a linked list.

The linked-list approach to the word-list problem is far more efficient. For example, in the complete text of H. G. Wells' "The War of the Worlds", although there are 60,207 words, only 6,911 of them are unique. Since the average word length is much less than 25 (actually, it is about 4.4 characters), the memory requirements for your program will be much less than for the Lab 6 approach.
You should attempt to complete the assignment *before* your scheduled lab period, as many of you will find it will take more time than will be available in the lab. Please keep in mind that the marking process will begin approximately sixty minutes before the end of the lab.

Part I—Reading the Words

Write a program named `lab8` to read text from a file, storing each *word* in a linked list. You may assume no word will be longer than 25 characters. Use a `struct` to define a data type to store the information for each node, as follows

```
struct WordListNode
{
  char         *word  ;
  int           count ;
  WordListNode *next  ;
} ;
```

As you read each word, remove the punctuation and convert to lower case as in Lab 6 (you may use your own code from Lab 6, or use the `void cleanWord(char s[])` function from the posted solutions). Once the punctuation has been removed, dynamically allocate a C-string array *just large enough* to store the word, and store the pointer to this array in the `word` field of a `WordListNode` variable when inserting it into the list.
The name of the file containing the words will be passed to your program using a command-line parameter, as in Lab 6.

Part II—Sorting

You will not need a special method to sort. Instead, when you add each newly-read word to the list, insert it in an appropriate place in the list such that the list remains sorted at all times. Each time you add a word, make sure to initialize its count to one.

Part III—Removing Repeated Words

You will not need a special method to remove repeated words if your list insertion function only adds words if they do not already appear in the list. However, you should increment the count of the word if it already appears in the list.

Part IV—Output

Once the words have been read into a list of unique words, you are to output the list with one word per line, followed by the number of times the word appeared in the file, in parenthesis. For example,

```
the (175)
```

This should be followed by the number of unique words and also the total number of words read from the file.

Part V—Searching

Implement a function to do sequential search on the list of words. After outputting the list of unique words and their counts, your program should repeatedly prompt the user for words to search for, until the user enters a word that starts with a non-letter. You will need to make the words entered lowercase and remove any leading/trailing punctuation, but you can re-use the functions you wrote to do this earlier.
If a search word is found, print out the word and its count. If a search word is not found, print the word in double quotes followed by `NOT FOUND`.
When the user is done searching, you **must delete any allocated memory** before your program exits.

Part VI—Debugging

If you wish to use DDD to help debug your program, you will find it necessary to type

```
set args <fileName>
```

where `<fileName>` is replaced with the name of the file you are reading your words from, into the command area at the bottom of the DDD window. This will cause DDD to pass the filename via the `argv[]` parameter in your `main` function when it runs your program.

Part VII—Marking

You must be able to compile the program and demonstrate it for the TA who marks your lab.

Submission

Before your lab is over you **must** submit your files using the unix `submitaps105f` command. For example, the command `submitaps105f 8 lab8.c` may be used to submit your file. The manual page for this command may be read by executing the unix `man submit` command.
The command `submitaps105f -l 8` may be used to confirm that you have successfully submitted your files.