University of Wollongong

School of Computer Science and Software Engineering

CSCI124/MCS9124 Applied Programming Spring 2014

Assignment 4 (6 marks)

Due 11:59pm Sunday October 19, 2014 (End of Week 11).

Background:

Task

This assignment requires you to write a program in C++ to keep track of the "top 10 words" in a file. The "top" ranking is defined by the frequency of occurrence, for example, in an English text, the three-letter word "the" often appears the most often and thus "the" would appear at the top of the list of most-frequently-occurring words.

The assignment will test a student's knowledge of design, data structures (linked lists and hash tables), text processing, pointers and dynamic memory

Introduction

Two fields that rely heavily on text processing analysis are (i) authorship attribution studies and (ii) searching biological DNA. Authorship attribution studies try to determine the author of an unknown work, for example, the debate as to whether Shakespeare or Christopher Marlowe (or someone else) wrote a certain play. In authorship attribution studies, word frequency lists are often part of an initial analysis: what are the different words that are used and how frequently do those words appear in a certain work?

As you are aware given the fanfare of the 50th anniversary of Watson and Crick's discovery of the structure of DNA as well as the (near) completion of the sequencing of the human genome, searching through the "text" of DNA is a hot new research field. Bioinformatics (the collection, storage, management, and analysis of the huge amounts of biologically-relevant data) and genomics (the analysis of DNA sequence) are two new fields that are heavily dependent on "text" analysis. Since DNA is made up of (only) a four-letter alphabet ('A', 'G', 'C', and 'T') representing the four chemical nucleotides, the human and other genomes fill files and files with nothing but ACGT. Determining the frequency of each eight-letter word (called a "motif" in DNA Land) within a genome is a common question in genomics: what are the different motifs that appear in a section of a genome and how frequently do those motifs appear?

In this assignment your job is to write a program which produces a summary of the top words/ motifs. This is achieved using pointers and data structures which you learnt about in class.

Specification

You are to write a program which when invoked asks the user to enter an input file.

```
Enter file name?
```

The input file can be of one or two types: (1) an English story (e.g., `moby-dick.txt`) or (2) a file of DNA (e.g. `ecoli.fna`). The two files can be differentiated in the following manner:

1. DNA files *always* begin with a header line, beginning with the character `>`, e.g.

```
>gi|16127994|ref|NC_0009| Escherichia coli K12, complete genome
  (the details of this header line are not important)
```

2. English text files do *not* have a header line (the story just begins) e.g.

```
MOBY DICK OR THE WHALE
by Herman Melville

CHAPTER 1  Loomings
Call me Ishmael.  Some years ago--never mind how long precisely--
having little or no money in my purse, and nothing particular
```

In the `datasets` file you will find a number of DNA files and a number of English text files. These files are of varying sizes and can be used to test your program.

Once you program receives a file name it is to do some computational work which is explained below in detail. Once complete the program is to produce some output.

The output from your program should consist of an (unsorted) list of frequency-word pairs. In addition to this the program should indicate the number of hash table collisions – this will vary depending on the program. Examine the output of the following two runs with sample data provided.

```
Enter file name? smallenglish.txt

1       able
1       elba
2       I
1       ere
1       was
1       saw
1       stop
1       pots
1       oh
1       ho
```

```
Enter file name? testdna2.fna

7       ACGTACGT
6       CGTACGTA
6       GTACGTAC
6       TACGTACG
```

Note: The set above will produce 24 collisions as they all yield the same hash value.

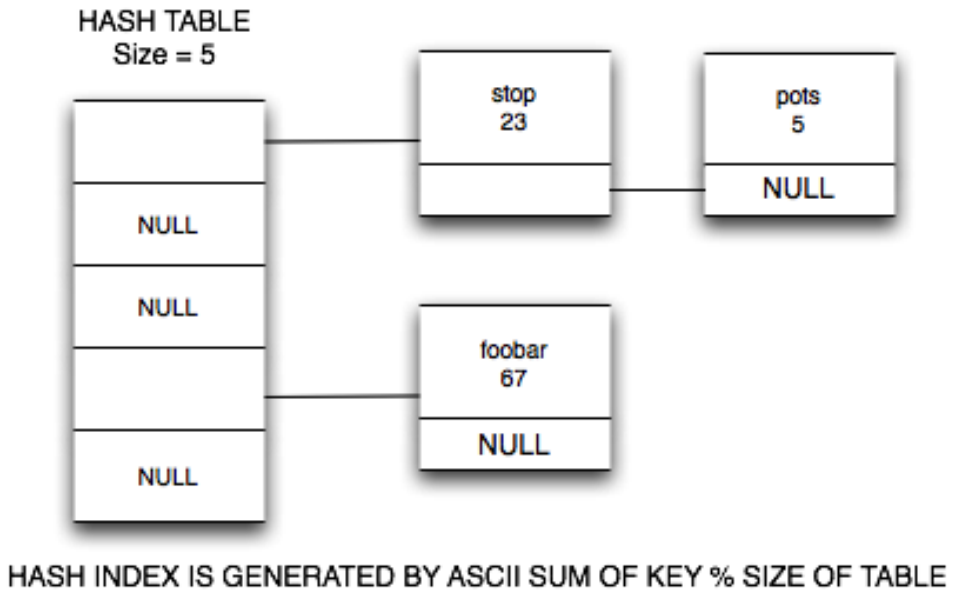
It should be noted your program only needs to output the top 10 words. If there are less then 10 then words in the collection then the program should handle this. *In addition to this your program should print out the number of collisions experienced when populating the hash table. A collision is identified when a hash table element is already populated with a list.*

Now that you understand what the program is to do – the question that needs to be asked is how can this be achieved.

In order to keep track of words and their associated frequencies, you should implement a Hash Table that uses the current word/ motif as the key. The simplest function to generate the hash value for the key would be the (summation of each ASCII value making up the key) modulus (the size of the hash table).

It is beyond the scope of this assignment to worry about a "good" hashing function. We can expect thousands of unique words in a large story. As well, there are 4^8 or 65,536 potential 8-letter motifs. The size of your hash table is up to you. The truth is, a poor hashing function and table size will actually "smoke test" your program in a very thorough fashion. *Note: I am not subtracting points if you have many collisions – infact you MUST have collisions. A good value for the hash table size would be 100 elements.*

Since you *will* have collisions (that is, words such as "pots" and "stop" will have the same total value when you sum their ASCII letter values), you should implement a dynamically allocated linked list at each cell in your hash table as shown below. It should be noted that the linked list associated with each cell will grow during execution.



In order to produce the desired output, you will have to walk the hash table – this means examine each element one by one. This is important when producing the list of the top 10 words.

Program Structure

It is assumed that you will use the following files to manage your source code:

`constants.h`	all constants (except those used within your hash table or list)
`list.h`	declaration of your linked list functions
`list.cpp`	definitions of your linked list functions
`hash.h`	declaration of your hash table functions
`hash.cpp`	definitions of your hash table functions
`main.cpp`	your main() routine and associated functions you write to run the program.

Your linkedlist and hashtable should be implemented as classes with appropriate constructors and destructors. You should try to make the list and the hashtable as generic as you can i.e. its not tightly coupled to the data being stored…

Tips/ Notes

(1) In this assignment you should ignore case. That is, you really should treat "That" and "that" as the same word, but since we have not required you to convert everything to lowercase, "That" and "that" will appear as two separate words; that is ok.

(2) Notice that we are ignoring punctuation marks. We could take them out, but for now, you can assume that "whale" and "whale." (notice the period after the 'e' in the second one) are different words; that is ok.

(3) You *must* skip the newlines when reading DNA. That is, if you read a newline (`'\n'`) character, you should throw that character away and read in another character. Said differently, your motifs should only have valid A, C, G, T letters in them. You should never have a motif such as: `"ACGTACG\n".`

(4) Take special care with your linked list. It is your responsibility to ensure the construction and destruction of the list. You are to ensure memory is not leaked.
(5) You will find lots of example in the lectures as to how linked lists work and how hash tables work. Use this code as a base to the assignment.
(6) You will have to make a number of design decisions on your own in this assignment e.g. max size of a word. Think about the problems presented very carefully prior to implementation.

Submission Procedure

To submit the assignment enter the following command.

```
submit –u USERNAME –c CSCI124 –a ass4 constants.h list.h list.cpp hash.h
hash.cpp main.cpp
```

Please note assignments that are late, will attract a 1 mark penalty each day late. Assignments will receive a zero mark if they are submitted more then 3 days late.