The University of Melbourne
School of Computing and Information Systems
COMP10002 Foundations of Algorithms
Semester 1, 2017
Assignment 2
Due: 4pm Wednesday 24th May 2017

1 Learning Outcomes

In this assignment you will demonstrate your understanding of dynamic memory allocation, linked data structures, and search algorithms. You will further extend your skills in program design, testing, and debugging.

2 The Story...

Text analysis and understanding is becoming prevalent in our daily lives. For example, intelligent personal assistant apps such as Apple Siri and Google Now process user requests by first converting voice to text with a *speech recognition* algorithm, and then analysing the text and finding answers to the requests. Finally, the answers found are read out by a *text to speech* (TTS) algorithm.

To help computers understand a sentence, there are a few standard processing steps. In this assignment, we are interested in a processing step called *named-entity recognition* (NER). NER is a process to label the words in text that refer to real-world objects with proper names, such as persons, locations, organisations, movies, etc.

See the following example.

```
Sentence:          NER label:
hugh               FIRST_NAME
jackman            LAST_NAME
is
retiring
the
wolverine
character
after
logan              MOVIE
```

There are various NER algorithms. The core of those algorithms are linguistic rules (such as "last name usually follows first name") and statistics (such as how often "`logan`" is used as a movie name instead of as a person name). Entity name dictionaries are often used to support those algorithms.

In this assignment, you will implement an NER algorithm based on a dictionary. For simplicity, the algorithm only labels person names. *Note that you do not need to have any knowledge in linguistics to complete this assignment.*

3 Your Task

You are given a person name dictionary (with **at least one and up to 100 unique names**) and a sentence to be processed in the following format.

```
#hewitt
5 95 0
#hugh
40 60 0
#jackman
0 100 0
#logan
40 0 60
#melbourne
5 5 90
#sydney
5 5 90
#zack
40 40 20
%%%%%%%%%%
hugh jackman is retiring the wolverine character after logan
```

The input starts with a list of names sorted alphabetically. Each name occupies two lines. Line 1 starts with a '`#`', which indicates the start of a name and is followed by the name itself (e.g., "`hugh`"). There are **up to 30** lower case English letters in each name. There are no upper case letters or any special characters.

Line 2 contains 3 integers. They indicate the probabilities of the name to be used as a first name, a last name, and a non-name word, respectively (for simplicity, we do not consider middle names). For example,"`hugh`" has a probability of 40% to be a first name, 60% to be a last name, and 0% to be a non-name word; "`logan`", on the other hand, has a probability of 40% to be a first name, 0% to be a last name, and 60% to be a non-name word. You do not need to worry about how these probabilities are obtained[1]. You just need to use them as they are given. The three integers always exist for each name; they sum up to 100; and the first two integers are not both 0. *Hint: You may use an array of three integers to store the probabilities.*

The line "`%%%%%%%%%%`" indicates the end of the dictionary and the start of the given sentence.

The sentence given will occupy one line, where the words are separated by a single space. The sentence will contain at least one word, but **there is no upper limit on the number of words or the number of characters in the sentence**. Each word may contain **up to 30** lowercase English letters. **There will be no punctuation marks (e.g., ',' or '.')**.

You will label the names in the sentence by consulting the name dictionary.

3.1 Stage 1 - Reading One Name (Up to 4 Marks)

Your first task is to design a "`struct`" to represent a name. You will then read in a name from the input data, and output it in the following format. *Hint: To print out a '`%`' character, you can use "`%%`" in the formatting string of the* `printf()` *function.*

```
========================Stage 1========================        /* 25 '='s each side */
Word 0: hewitt
Label probabilities: 5% 95% 0%
```

[1]These probabilities are usually obtained from a large corpus of texts manually labelled.

3.2 Stage 2 - Reading the Whole Dictionary (Up to 8 Marks)

Next, continue to finish reading the whole dictionary. You need to design a proper data structure to store the names read. An array will do the job nicely. When this stage is done, your program should output: the total number of names in the dictionary and the average number of characters per name (up to two digits after the decimal point, by using "`%.2f`" in the `printf()` function). The output of this stage based on the sample input is as follows.

```
========================Stage 2========================
Number of words: 7
Average number of characters per word: 5.86
```

3.3 Stage 3 - Reading the Sentence (Up to 12 Marks)

Your third task is to read in the given sentence, break it into words, store the words in a *linked data structure*, and output the words one at a line. The output in this stage for the example above should be:

```
========================Stage 3========================
hugh
jackman
is
retiring
the
wolverine
character
after
logan
```

Hint: You may use the "`getword()`" *function (Figure 7.13 in the textbook, link to source code available in lecture slides) to read in words in a sentence. You may modify the linked list implementation in* "`listops.c`" *(link to source code available in lecture slides) to store the words.* You are free to use other linked data structures (e.g., queue, stack, etc) if you wish. Make sure to attribute the use of external code properly.

If you are not confident with using linked data structures, you may use an array of strings to store the words, assuming a maximum of 50 words in the sentence. However, if you do so, the full mark of this stage will be reduced from 4 to 2.

3.4 Stage 4 - Labelling the Names (Up to 15 Marks)

The next stage is to label the names. You will use the binary search algorithm (*Hint: You may use the* "`binary_search()`" *function, link to source code available in lecture slides, or* "`bsearch()`" *provided by* "`stdlib.h`") to look up each word from the sentence in the dictionary. If the word is not found, you simply output the word and a "`NOT_NAME`" label. If the word is found, you need to check probabilities of the word to be a first name and to be a last name. If either of the probability is not zero, you need to label the word as a first name and/or a last name accordingly. You do not need to consider the probability of the word to be a non-name word.

The output of your program in this stage for the example above should be:

```
========================Stage 4========================
hugh                            FIRST_NAME, LAST_NAME
jackman                         LAST_NAME
is                              NOT_NAME
retiring                        NOT_NAME
the                             NOT_NAME
wolverine                       NOT_NAME
character                       NOT_NAME
after                           NOT_NAME
logan                           FIRST_NAME
```

Note that if you use a sequential search instead of the binary search, the full mark of this stage will be reduced from 3 to 2.

3.5 Stage 5 - Refining the Labels (Up to 15 Marks)

This stage is for a challenge. *You do NOT need to complete this stage to obtain the full mark of the assignment.* Please do not start on this stage until your program through to Stage 4 is (in your opinion) perfect, and is submitted, and is verified, and is backed up.

Stage 4 sample output is still imperfect as compared with the labelling example shown in Page 1 of this document. There is ambiguity on whether "`hugh`" should be a first name or a last name, and "`logan`" has been mis-labelled as a first name. It is a real challenge to remove such ambiguity and wrong labels in real world systems. The state-of-the-art systems can now achieve around 90% accuracy in common NER tasks.

In this stage, you will make an attempt to refine the word labels of the given sentence. There is no specific algorithms or steps given in this stage. You need to work out your own algorithm that determines a single label for each word in the given sentence based on the probabilities given in the dictionary and some common sense (such as "two last names do not usually appear together").

The ideal output of your program in this stage for the example above should be:

```
========================Stage 5========================
hugh                            FIRST_NAME
jackman                         LAST_NAME
is                              NOT_NAME
retiring                        NOT_NAME
the                             NOT_NAME
wolverine                       NOT_NAME
character                       NOT_NAME
after                           NOT_NAME
logan                           NOT_NAME
```

There is one bonus mark for this stage. The bonus mark earned in this stage will compensate for the marks you lost in the earlier stages (assuming that you lost some, your total mark will not exceed 15).

There is no golden standard algorithm for this stage. Your marks for this stage will be determined by the accuracy of the labels. If you achieve an accuracy of 80% or more on labelling the words in the sentence that appear in the name dictionary, you will earn 1 bonus mark; if you achieve an accuracy of 50% or more and below 80% on labelling the words in the sentence that appear in the name dictionary, you will earn 0.5 bonus marks. No bonus marks will be awarded for an accuracy below 50%.

4 Submission and Assessment

This assignment is worth 15% of the final mark. A detailed marking scheme will be provided on the LMS.

You need to submit all your code (including any external code you used such as the functions in "`listops.c`") in one file named `assmt2.c` for assessment. The submission process is similar to that of Assignment 1. You will need to log in to a Unix server (dimefox.eng.unimelb.edu.au or nutmeg.eng.unimelb.edu.au) and submit your files using the following command:

```
submit comp10002 a2 assmt2.c
```

You can (and should) use submit both **early and often** - to get used to the way it works, and also to check that your program compiles correctly on our test system, which has some different characteristics to the lab machines. You should verify your submission using the following commands:

```
verify comp10002 a2 > receipt-a2.txt /* Here '>' writes the output into receipt-a2.txt */
more receipt-a2.txt
```

You will be given a sample test file `test0.txt` and the sample output `test0-output.txt`. You can test your code on your own machine with the following command and compare the output with `test0-output.txt`:

```
mac: ./assmt2 < test0.txt    /* Here '<' feeds the data from test0.txt into assmt2 */
```

Only the last submission made before the deadline will be marked. You may discuss your work with others, but what gets typed into your program must be individual work, **not** copied from anyone else. Do **not** give hard copy or soft copy of your work to anyone else; do **not** "lend" your memory stick to others; and do **not** ask others to give you their programs "just so that I can take a look and get some ideas, I won't copy, honest". The best way to help your friends in this regard is to say a very firm "no" when they ask for a copy of, or to see, your program, pointing out that your "no", and their acceptance of that decision, is the only thing that will preserve your friendship. *A sophisticated program that undertakes deep structural analysis of C code identifying regions of similarity will be run over all submissions in "compare every pair" mode.* See https://academichonesty.unimelb.edu.au for more information.

Deadline: Programs not submitted by **4pm Wednesday 24th May 2017** will lose penalty marks at the rate of 2 marks per day or part day late. Late submissions after 4pm Friday 26th May 2017 will **not** be accepted. Students seeking extensions for medical or other "outside my control" reasons should email the lecturer at jianzhong.qi@unimelb.edu.au. If you attend a GP or other health care professional as a result of illness, be sure to take a Health Professional Report form with you (get it from the Special Consideration section of the Student Portal), you will need this form to be filled out if your illness develops into something that later requires a Special Consideration application to be lodged. You should scan the HPR form and send it in connection with any non-Special Consideration assignment extension requests.

And remember, *Algorithms are fun!*