The University of Melbourne
School of Computing and Information Systems
COMP10002 Foundations of Algorithms
Semester 1, 2019
Assignment 2
Due: 4pm Friday 31st May 2019

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 and implementation.

2 The Story...

In Assignment 1, we studied basic principals in text cleansing. In this assignment, we will write code to analyse cleansed text. In particular, we are interested in a type of analysis called *sentiment analysis*.

Sentiment analysis is a process to identify opinions in a given piece of text and to determine whether the writer's attitude towards a particular topic (or product, etc.) is positive, negative, or neutral. For example, "`Algorithms are fun!`" expresses a positive attitude; "`I hate raining!`" expresses a negative attitude; and "`I am going to an FOA lecture.`" is often neutral. Sentiment analysis is widely applied to customer reviews, polls, and online social network data (tweets) for various applications such as customer analysis and product recommendations.

There are various sentiment analysis algorithms. The core of those algorithms are linguistic rules (such as "double negatives means positive") and statistics (such as how positive is the attitude when "`fun`" is used). Sentiment dictionaries are often used to support those algorithms. See the following example:

```
Sentence:       Sentiment:
algorithms      0
are             0
fun             4
although        0
she             0
dislikes        -2
going           0
to              0
school          0
on              0
a               0
rainy           -1
day             0
```

Here, the numbers represent the *sentiment scores* (*polarity*) of the words. For example, "`fun`" expresses a strong positive attitude and has a sentiment score of 4; "`dislikes`" and "`rainy`" express negative attitudes with sentiment sores of -2 and -1, respectively; all other words are neutral and have sentiment scores of 0. Overall, the sentence has a sentiment score of $4 + (-2) + (-1) = 1$.[1]

[1] Sentiment score computation for a sentence in real systems is more complex. To simplify the assignment, we just sum up the sentiment scores of the words.

3 Your Task

In this assignment, you will implement a sentiment analysis algorithm based on a dictionary. *Note that you do not need to have any knowledge in linguistics to complete this assignment.*

You are given a sentiment dictionary (with **at least 1 and at most 100 unique words**) and a sentence to be processed in the following format.

```
#break
-1
$broke*broken*breaking*breaks
#dislike
-2
$disliked*disliked*disliking*dislikes
#enjoy
3
$enjoyed*enjoyed*enjoying*enjoys
#fun
4
$
#lose
-2
$lost*lost*losing*loses
#rainy
-1
$
%%%%%%%%%
algorithms are fun although she dislikes going to school on a rainy day
```

The input starts with a list of words *sorted alphabetically*. Each word occupies *exactly* three lines. Line 1 starts with a '`#`', which is followed by the word itself (e.g., "`break`"). There are **at least 1 and at most 20** lowercase English letters in each word. There are no uppercase letters, digits, or any special characters.

Line 2 contains the sentiment score of the word. For example,"`break`" and "`dislike`" have sentiment scores of -1 and -2, respectively. You do not need to worry about how these scores are obtained.[2] The sentiment score always exists for each word in the dictionary, and it is within the ranges of $[-5, -1]$ and $[1, 5]$.

Line 3 starts with a '`$`', which is followed by the *variation forms* of the word that are separated by asterisks ('`*`'). If a word is a verb (e.g., "`break`"), it will have **4 variation forms** in the following order, where each form contains **at least 1 and at most 23** lowercase English letters: (i) past tense form (e.g., "`broke`"); (ii) past participle form (e.g, "`broken`"); (iii) present participle form (e.g., "`breaking`"); and (iv) third-person present form (e.g., "`breaks`"). If a word is a non-verb (e.g., "`fun`" and "`rainy`"), its variation forms will be empty. You may assume that any two words in the dictionary will **not** share the same variation forms (e.g., the variation forms of "`break`" and "`dislike`" are all different). You may also assume that any word will **not** be a variation form of another word in the dictionary (e.g., "`break`" is not a variation form of any other word). (*Hint: You do not need to understand what these variation forms mean to complete this assignment; for up to Stage 4, you may simply store all the variation forms of a word in a single string; for Stage 5, you will need to separate the variation forms and store them in multiple strings.)*

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 **at least 1 and at most 23** lowercase English letters. **There will be no punctuation marks (such as ',' or '.'), digits, or other special characters**.

You will label the words of the sentence with their sentiment scores by consulting the sentiment dictionary.

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

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

Your first task is to design a "`struct`" to represent a word in the dictionary. You will then read in a word from the input data, and output it in the following format.

```
========================Stage 1========================        /* 25 '='s each side */
First word: break
Sentiment score: -1
Forms: broke*broken*breaking*breaks
```

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 words read. An array will do the job nicely. When this stage is done, your program should output: the total number of words in the dictionary, the average number of characters per word, and the average sentiment score of the words (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: 6
Average number of characters: 4.83
Average sentiment score: 0.17
```

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========================
algorithms
are
fun
although
she
dislikes
going
to
school
on
a
rainy
day
```

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 (queues, stacks, etc) if you wish. Make sure to attribute the use of external code properly.

If you are not confident with linked data structures, you may use an array of strings to store the words, assuming a maximum of 20 words. If you do so, the full mark of this stage will be reduced from 4 to 2.

3.4 Stage 4 - Labelling and Calculating the Sentiment Score (Up to 15 Marks)

The next stage is to label the words and to calculate the sentiment score of the input sentence. 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 sentiment dictionary. If the word is not found, you simply output the word and a sentiment score of 0. If the word is found, you need to output its sentiment score accordingly. Your program should also sum up the sentiment scores of the words in the sentence and output the sum at the end of this stage.

See the next page for a sample output of this stage given the input example above.

```
========================Stage 4========================
algorithms              0
are                     0
fun                     4
although                0
she                     0
dislikes                0
going                   0
to                      0
school                  0
on                      0
a                       0
rainy                   -1
day                     0
Overall score:          3
```

Note that if sequential search is used instead of binary search, the full mark of this stage will be reduced from 3 to 2. Note also that in this stage, we are only matching with the words in the dictionary but *not* their variation forms. Thus, "`dislikes`" in the sentence has not found a match in the dictionary. The variation forms will be considered in the next stage.

3.5 Stage 5 - Handling the Variation Forms (Up to 15 Marks)

This stage is for a challenge. If you are successful in this stage, you can earn back 1 mark you lost in the earlier stages (assuming that you lost some, your total mark will not exceed 15).

In this stage, you will match the words in the given sentence with dictionary words and their variation forms. If a word in the sentence is matched by a dictionary word or its variation forms, you should output the matched dictionary word (*not* its variation form) and its sentiment score. If a word cannot be matched, you should output "`NOT_FOUND`" and a sentiment score of 0. You should also output the sum of the sentiment scores at the end. The output in this stage for the input example above should be:

```
========================Stage 5========================
algorithms              NOT_FOUND               0
are                     NOT_FOUND               0
fun                     fun                     4
although                NOT_FOUND               0
she                     NOT_FOUND               0
dislikes                dislike                 -2
going                   NOT_FOUND               0
to                      NOT_FOUND               0
school                  NOT_FOUND               0
on                      NOT_FOUND               0
a                       NOT_FOUND               0
rainy                   rainy                   -1
day                     NOT_FOUND               0
Overall score:                                  1       /* print a '\n' at the end */
```

If you use sequential search for this stage, you can only earn 0.5 bonus marks. To earn the full 1 bonus mark, **you need to design a search algorithm that can determine whether a word in the given sentence is in the dictionary (in original or variation forms) with $O(m \log(n_1 + n_2))$ or lower time complexity on average**, where n_1 is the number of dictionary words, n_2 is the number of variation forms, and m is the maximum length of a variation form. To achieve this, you might need supporting data structures. You may use any library functions to construct the supporting data structures, and the construction costs are excluded from the search time complexity.

At the end of your submission file, you need to add a comment that states the time complexity of your search algorithm, and explains why it has that time complexity. Note also that if you use binary search in both Stages 4 and 5, you should *not write more than one* binary search functions. Instead, you should use function pointers to customise the behaviour of the binary search function in different cases.

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. Only the last submission made before the deadline will be marked. You should verify your submission using the following commands:

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

Note that we are using the following command to compile your code on the submission server.

```
gcc -Wall -std=c99 -o assmt2 assmt2.c
```

The flag "`-std=c99`" enables the compiler to use a more modern standard of the C language – `C99`. To ensure that your submission works properly on the submission server, you should use this command to compile your code on your local machine as well.

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 Friday 31st May 2019** will lose penalty marks at the rate of 2 marks per day or part day late. Late submissions after 11:59pm Sunday 2nd June 2019 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!*