The University of Melbourne
School of Computing and Information Systems
COMP10002 Foundations of Algorithms
Semester 1, 2019
Assignment 1
**Due: 4pm Friday 3rd May 2019**

## 1 Learning Outcomes

In this assignment, you will demonstrate your understanding of arrays, pointers, input processing, and functions. You will also extend your skills in terms of code reading, program design, testing, and debugging.

## 2 The Story...

Social networks such as Facebook and Twitter have become part of our daily lives. They generate large volumes of contents such as user posts, which describe people's activities, thoughts, feelings, and more. Various individuals and organisations are harvesting and analysing social network data for business, research, and political decision support. For example, the current generation of trading robots (trading software) are reading tweets to make trading decisions.[1,2]

While social network contents contain rich information, to make use of such contents is non-trivial. Particularly, such contents are often noisy and do not follow lexical and grammatical rules rigorously (for example, "`Just had 1st foa lecture, algrthms r awesome!!! #FOA`"). They cannot be handled directly by traditional text analytical tools that were developed based on well written documents such Wall Street Journal articles. The first step towards utilising social network contents is to clean up such noisy data. In this assignment, you will learn basic principles in text data cleansing and write a program to cleanse a list of Twitter user posts, a.k.a. *tweets*.

## 3 Your Task

You will be given an input that consists of multiple lines. Each line contains a tweet with **at least 1 and at most 280 characters (at least 1 of the characters is an English letter or a digit)**. There are **at least 1 and at most 100** lines. A sample input with ten tweets is shown below.

```
Just*had*1st*foa*lecture,*algrthms*r*awesome!!!*#FOA
^_^*assignment1*is*fun!
Ultimate*Answer*to*Life,*Universe,*Everything*.*.*.*#42
handball,*handball,*handball
Knowledge*Is*Power,*France*Is*Bacon.
Love*this*CUTE*cat!*https://anonymisedURL
***A*sample*tweet***
#YET*:)*#Another*#sample!&(&(!*Tweet
@#&**@comp10002***#2019**#assignment1*^_^***is*@FUN***)*)
Handball*Handball
```

Note that we have replaced all whitespaces (' ') with asterisks ('*') to simplify the debugging and testing process. There will *not* be any whitespaces in the input data.

[1] `https://www.cnbc.com/id/100666302`
[2] `https://slate.com/business/2015/04/bot-makes-2-4-million-reading-the-web-meet-the-guy-it-cost-a-fortune.html`

You will be given a skeleton code file named `assmt1.c` for this assignment on the LMS. The skeleton code file contains a `main` function that has been completed already. There are a few other functions which are incomplete. You need to add code into them for the following tasks (Stage 5 is for a challenge and is optional). **Note that you should *not* change the `main` function, but you are free to modify any other parts of the skeleton code (including adding more functions).**

## 3.1 Stage 1: Reading the First Tweet (Up to 3 Marks)

Your first task is to understand the skeleton code. Note the use of the type `tweet_t` in the skeleton code, which is essentially a `char` type array. Each `tweet_t` variable stores the content of a tweet.

You need to add code to the `stage_one` function to call the `read_one_tweet` function to read the first tweet. Here, the `read_one_tweet` function is already given to you. The `stage_one` function then calls the `tweet_tolower` function to convert all uppercase letters in the first tweet into lowercase letters. The `tweet_tolower` function is empty and you need to add code to complete it as well. Converting all words into their lowercase forms helps word frequency counting in real systems, for example, "foa" and "FOA" both refer to our subject.

The output for this stage given the above sample input should be (where "`mac:`" is the command prompt).

```
mac: ./assmt1 < test0.txt
Stage 1
==========
just*had*1st*foa*lecture,*algrthms*r*awesome!!!*#foa
```

As this example illustrates, the best way to get data into your program is to edit it in a text file (with a "`.txt`" extension, `jEdit` can do this), and then execute your program from the command line, feeding the data in via input redirection (using `<`). In the program, we will still use the standard input functions such as `getchar()` to read the data fed in from the text file. Our auto-testing system will feed input data into your submissions in this way as well. To simplify the assessment, your program should not print anything except for the data requested to be output (as shown in the output example).

You should plan carefully, rather than just leaping in and starting to edit the skeleton code. Then, before moving through the rest of the stages, you should test your program thoroughly to ensure its correctness.

## 3.2 Stage 2: Removing Non-alphanumeric Characters (Up to 8 Marks)

Now add code to the `stage_two` function to loop through the input tweets. For each tweet, your program should (i) convert uppercase letters into lowercase letters, and (ii) cleanse the tweet by keeping only English letters, digits, and asterisks ('`*`')[3] while removing all other characters. On the same sample input data, the additional output for this stage should be:

```
Stage 2
==========
just*had*1st*foa*lecture*algrthms*r*awesome*foa
*assignment1*is*fun
ultimate*answer*to*life*universe*everything****42
handball*handball*handball
knowledge*is*power*france*is*bacon
love*this*cute*cat*httpsanonymisedurl
***a*sample*tweet***
yet**another*sample*tweet
**comp10002***2019**assignment1****is*fun****
handball*handball
```

*Hint:* You may call the `tweet_tolower` function written in Stage 1 to convert a tweet into lowercase and write another function to cleanse the tweet. Alternatively, you may write a function to do both in a single pass over the tweet. The second approach is faster in practice, but we allow both approaches for the purpose of the assignment.

[3] These are usually the informative contents of a tweet.

### 3.3 Stage 3: Removing Extra Asterisks and Finding the Longest Tweet (Up to 12 Marks)

Add code to the `stage_three` function to further cleanse the tweets by removing any leading, trailing, and consecutive asterisks. After this cleanse, for every tweet, there should *not* be asterisks at the start or the end. Also, there should be just one asterisk between any two adjacent words in a tweet. For example, after this stage, a tweet "`**comp10002***2019**assignment1****is*fun****`" should become "`comp10002*2019*assignment1*is*fun`".

Additionally, this stage should also output the total number of tweets, the longest tweet (that is, the tweet with the maximum *length*), and its length. Here, the length refers to the number of characters in a tweet after cleansed by the stages above. *In the case of ties, the tweet that appears the earliest in the input should be output.* The output for this stage given the sample input above is as follows.

```
Stage 3
==========
just*had*1st*foa*lecture*algrthms*r*awesome*foa
assignment1*is*fun
ultimate*answer*to*life*universe*everything*42
handball*handball*handball
knowledge*is*power*france*is*bacon
love*this*cute*cat*httpsanonymisedurl
a*sample*tweet
yet*another*sample*tweet
comp10002*2019*assignment1*is*fun
handball*handball
Total: 10
Longest: just*had*1st*foa*lecture*algrthms*r*awesome*foa
Length: 47
```

*Hint:* Study the `getword` function (Figure 7.13 in the textbook, link to source code available in lecture slides `lec06.pdf`) to learn how to skip leading and trailing asterisks.

### 3.4 Stage 4: Finding the Non-contained Tweets (Up to 15 Marks)

There are over 500 million tweets being posted daily. We may not need all the tweets and may remove some of them to save storage space. In particular, the tweets that are fully contained in another tweet do not offer extra information. Such tweets may be removed.

Now modify the `stage_four` function to loop through the tweets processed by the stages above, find the tweets that are **not** fully contained in any other tweet, and print out such tweets. You may assume that no two tweets can both contain each other. Your program should also print out the number of non-contained tweets. The additional output for this stage given the sample input above is as follows.

```
Stage 4
==========
just*had*1st*foa*lecture*algrthms*r*awesome*foa
ultimate*answer*to*life*universe*everything*42
handball*handball*handball
knowledge*is*power*france*is*bacon
love*this*cute*cat*httpsanonymisedurl
a*sample*tweet
yet*another*sample*tweet
comp10002*2019*assignment1*is*fun
Non-contained: 8
```

Here, "`handball*handball`" is fully contained in "`handball*handball*handball`"; "`assignment1*is*fun`" is fully contained in "`comp10002*2019*assignment1*is*fun`". These two tweets are not printed.

Note that you are **not** allowed to used any function from the `string.h` library for this stage. See next page for hints of this stage.

*Hint:* First, write a new function `is_contained` that takes two tweets `source_tweet` and `target_tweet` as the input, loops through the two tweets character by character (a two-layer nested loop), and returns 1 if `source_tweet` is contained in `target_tweet`, and 0 otherwise. Then, for every tweet, you may loop through the rest of the tweets to test whether this tweet is contained in another tweet by calling the `is_contained` function. It takes another two-layer nested loop to test all the tweets against each other. You do not need to use the `KMP` algorithm for this stage.

### 3.5 Stage 5: Sorting the Tweets (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).**

Add code to the `stage_five` function to sort and output the tweets (*including the contained ones*) in the ascending order of their lengths (number of characters after cleansing). When there is a tie, the tweets are further sorted alphabetically (that is, in dictionary order). You may assume that there will not be two tweets that are exactly the same. The additional output given the sample input above is as follows.

```
Stage 5
==========
a*sample*tweet
handball*handball
assignment1*is*fun
yet*another*sample*tweet
handball*handball*handball
comp10002*2019*assignment1*is*fun
knowledge*is*power*france*is*bacon
love*this*cute*cat*httpsanonymisedurl
ultimate*answer*to*life*universe*everything*42
just*had*1st*foa*lecture*algrthms*r*awesome*foa   /* print a newline ('\n') at the end */
```

You may use any sorting algorithm, library function, or sample code from the textbook. Make sure to attribute the source if you use any sample code.

## 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 in one file named `assmt1.c` for assessment**; detailed instructions on how to submit will be posted on the LMS once submissions are opened. Submission will NOT be done via the LMS; instead you will need to log in to a Unix server and submit your files to a software system known as `submit`. 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 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: ./assmt1 < test0.txt    /* Here '<' feeds the data from test0.txt into assmt1 */
```

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

```
gcc -Wall -std=c99 -o assmt1 assmt1.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 3rd May 2019** will lose penalty marks at the rate of two marks per day or part day late. Late submissions after 4pm Monday 6th May 2019 will **not** be accepted. Students seeking extensions for medical or other "outside my control" reasons should **email the subject cooridnator 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!*

©2019 The University of Melbourne
Prepared by Yi Han and Jianzhong Qi