DALHOUSIE UNIVERSITY
Inspiring Minds

Fall 2021 (Sep7-Dec7)
Faculty of Computer Science

CSCI 4152/6509 — Natural Language Processing

Assignment 1

Due: *Tuesday, Sep 28, 2021 by midnight*
Worth: 90 marks ($90 = 22 + 19 + 19 + 15 + 15$)
Instructor: Vlado Keselj, vlado@cs.dal.ca vlado@dnlp.ca

Assignment Instructions:

The submission process for Assignment 1 is based on the `submit-nlp` command on `timberlea` as discussed in the lab, or in the equvalent way by using the course web site, where you need to follow 'Login' and then the 'File Submission' menu option.

Important: You must make sure that your course files on timberlea are **not** readable by other users. For example, if you keep your files in the directory `csci6509` or `csci4152` you can check its permission using the command:

```
ls -ld csci6509
```

or

```
ls -ld csci4152
```

and the output must start with `drwx------`. If it does not, for example if it starts with `drwxr-xr-x` or similar, then the permissions should fixed using the command:

```
chmod 700 csci6509
```

or

```
chmod 700 csci4152
```

1) (22 marks) Complete the Lab 1 as instructed. In particular, you will need to properly:

a) (4 marks) Submit the file 'hello.pl' as instructed.

b) (4 marks) Submit the file 'lab1-example2.pl' as instructed.

c) (4 marks) Submit the file 'lab1-example5.pl' as instructed.

d) (5 marks) Submit the file 'lab1-task1.pl' as instructed.

e) (5 marks) Submit the file 'lab1-task2.pl' as instructed.

All examples must compile and run correctly. For example, if a syntax error is introduced to an example program by your typing mistake or by introducing incorrect characters through copying and pasting from a PDF file, the solution will not be accepted. The lab instructions state that the programs should be tested before being submitted. If some code is given to be used, you should type it rather than copy-paste it, unless specified differently. This gives you also a better opportunity to learn the programming language and illustrated concepts.

2) (19 marks) Submit your answer to this question as a plain-text file called `a1q2.txt` using the `submit-nlp` command, or the course web site. Clearly separate your answers into parts a), b), and c).

A plain-text file is preferred, but if you want to include a figure you can instead submit a file named `a1q2.pdf`, `a1q2.jpg`, or `a1q2.png` in the format specified by the file name extension (pdf, jpg, or png). Drawing a figure by hand is okay if it is clear and readable.

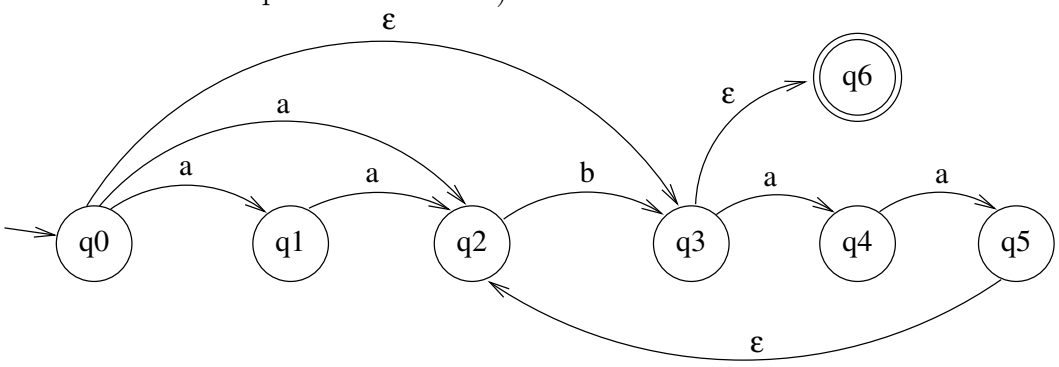
a) (7 marks) List the levels of NLP, with one-sentence description for each of them.

b) (12 marks) Consider an application on your phone or computer, which you can ask a question using your voice and it tries to give you an answer. Apple Siri is an example of such application. Choose three levels of NLP and briefly describe processing done at those levels that could be used in such application.

3) (19 marks) Submit answer to this question in a plain-text file named `a1q3.txt` using the `submit-nlp` command, or you can submit it as a pdf, jpg, or png file.

Clearly separate a) and b) parts in the solution.

Consider an NFA (Non-deterministic Finite Automaton) described by the following graph (note that NFA includes epsilon transitions):



a) (3 marks) Give three examples of words accepted by this NFA.

b) (3 marks) Briefly described in plain English what language is accepted by this NFA; i.e., what words are accepted by this NFA.

c) (3 marks) Write a regular expression that is equivalent to this NFA. (You do not have to use starting and ending anchors.)

d) (10 marks) Translate this NFA into a DFA using the process discussed in class. Submit your solution as the table and you **must** follow the algorithm described in class. You can submit your table either as part of a pdf or image file, or as a part of text file. (The filename must be as specified at the beginning of the assignment.)

If you use the plain-text format, then follow the table format as shown below as an example:

```
State      |   a    |    b   |   <-- (JUST AN EXAMPLE)
-----------+--------+--------+
S: q0q1    |  q0q1  |   q1   |
-----------+--------+--------+
    q1     |  q0q2  |  q0q2  |
-----------+--------+--------+
F: q0q2    |  q0q1  |   q1   |
-----------+--------+--------+
```

Explanation: Use characters minus, vertical line, and plus to draw the table. The columns correspond to input characters. The DFA states are set of NFA states shows as sequences of states in a sorted order by index (for example, use `q0q1` rather than `q1q0`). use labels `S:` and `F:` to denote start and finish states. If an NFA state is empty set, then use the word '`empty`' to denote it.

4) (15 marks) Write a Perl program named `a1q4.pl` and submit it using the `submit-nlp` command.

Sometimes we want to go through a large amount of text and search for any email addresses that appear in the text. Your task is to write the program which will search for lines containing an email address, and print then the email address, followed by a colon (`:`), followed by a space, and then the line containing that email.

In order to capture realistic email addresses, we will consider email address to be a string with the following conditions:

1. it starts with a letter,
2. after letter, it can be followed by an arbitrary sequence of letters, digits, minus sign (`-`), equal sign (`=`), period (`.`), plus sign (`+`), or underscore (`_`),
3. one at-sign (`@`),
4. after at-sign, there must be again a non-empty sequence of letters, digits, minus sign, equal sign, period, plus sign, or underscore, which **must** start with a letter or digit, **must** include at least one period, and the last character **must be a letter**.

For each line in which your program recognizes an email address, you should print that email address, a colon, a space, and the line itself. If a line contains more than one valid email, you must print only the first one, then colon, space, and the full line.

These requirements are far from perfect for the real task, but you must follow them exactly for this assignment. For example, if you read line input:

```
This is a line with email email@blah, or 123@Dal.ca, or a@b.
```

the program would not print the line because there is no valid email. On the other hand, with a few minor modifications, such as:

```
This is a line with email email@bla.h, or 123@Dal.ca, or a@b.
This is a line with email email@blah, or 12a3@Dal.ca, or a@b.
This is a line with email email@blah, or 123@Dal.ca, or a@b.X.
```

all three lines contain one valid email each: '`email@bla.h`', '`a3@Dal.ca`', and '`a@b.X`'.

You should use the Perl 'diamond' operator so that the program can read either standard input, or open files with filenames given as parameters. After reading each line, the program must decide whether to print output or not. For example, for the above input of three lines, which all contain valid email addresses, the output should look like this:

```
email@bla.h: This is a line with email email@bla.h, or 123@Dal.ca, or a@b.
a3@Dal.ca: This is a line with email email@blah, or 12a3@Dal.ca, or a@b.
a@b.X: This is a line with email email@blah, or 123@Dal.ca, or a@b.X.
```

If you are not familiar with the concepts of the *standard input* and *standard output*, you should look them up, particularly in the context of Linux or Unix operating systems. The standard input basically means that you can type the input using the keyboard after running the program from the command line, or redirect input from a file using a command such as '`./prog.pl < input.txt`'. The standard output means that the program prints to the screen, or that output can also be captured into a file using a command such as `./prog.pl < input.txt > output.txt`.

Important: Do not print any extra output other than specified! Sometimes students in order to make a more user-friendly interface print additional output such as prompts to users to enter input. This would be an error. You must follow precisely the specifications of the problem.

Your solution will be mainly marked on correctness. The markers will also look at the style. There are no significant requirements on the style other than reasonable indentation, and reasonable clarity of code. You can include comments, but they are not mandatory other than the header comment, which must identify file name, author, and the course. This is an example of such comment:

```
#!/usr/bin/perl
# File: a1q4.pl Author: Vlado Keselj
# Solution to question 4 of assignment 1, CSCI4152/6509 Fall 2021
```

5) (15 marks, programming) Write and submit a program written in Perl, Python, C, C++, or Java, named `a1q5.pl`, `a1q5.py`, `a1q5.c`, `a1q5.cc`, or `a1q5.java`, for the task given below.

The program is meant to give a basic statistic analysis of HTML tags in a file. We will consider HTML tag to be any string that starts with less-then sign (`<`) and ends with greater than sign (`>`). A tag can span more than one line. If outside of a tag we see the string `<!--` we will consider it to be the start of a HTML comment, which will end with the string `-->`. The part of the content inside a comment is ignored even if it appear to have some tags.

If a comment does not end; i.e., you cannot find ending `-->`, you should consider the end of the input to be the end of the comment.

If a tag does not end, you should not consider it to be a valid tag.

After a start of a tag, you should always look only for the end of a tag (`>`) even when the tag contains again strings such as '`<`' or '`<!--`'. The tags can also contains spaces and new lines.

At the end the program must print the number of tags, minimal tag length, maximal tag length, and average tag length exactly in this format:

```
Tag count: 3
Min length: 2
Max length: 16
Avg length: 8.00
```

There must be exactly one space after each colon (`:`), and the average length must be printed in exactly two decimal points rounded.

Important: Do not print any extra output other than specified! Sometimes students in order to make a more user-friendly interface print additional output such as prompts to users to enter input. This would be an error. You must follow precisely the specifications of the problem.

The starting '`<`' and ending '`>`' must be counted in the length, as well as any new-line characters in the input. The test files will be in the Unix-style format, so one special character '`\n`' is used at the end of each line. If there are no tags present in the input, just report 0 (zero) for all values.

For example, for the following input:

```
No tags here. This is a tag: <html>
We allow empty tags <>, or <multiline
tags>. <!-- a comment <ignored> -->
```

the program must print out:

```
Tag count: 3
Min length: 2
Max length: 16
Avg length: 8.00
```

Your solution will be mainly marked on correctness. The markers will also look at the style. There are no significant requirements on the style other than reasonable indentation, and reasonable clarity of code. You can include comments, but they are not mandatory other than the header comment, which must identify file name, author, and the course. This is an example of such comment in Perl:

```
#!/usr/bin/perl
# File: a1q5.pl Author: Vlado Keselj
# Solution to question 5 of assignment 1, CSCI4152/6509 Fall 2021
```