```
/* CMPUT 201 Assignments                                                    */
/* Dues:                                                                    */
/* #1:    11:55pm, September 29, 2019;                                      */
/* #2:    11:55pm, November 3, 2019;                                        */
/* #3:    11:55pm, December 6, 2019                                         */
```

(Mandatory assignment cover-sheet; without it, your work will not be marked.)

Submitting student: Nawalage Dinuka Ravindu Cooray

Collaborating classmates: none

Other collaborators: none

References (excluding textbook and lecture slides): https://codeforwin.org/2018/02/c-program-append-data-file.html for file writing https://www.programiz.com/c-programming/c-file-input-output as well.

Regardless of the collaboration method allowed, you must always properly acknowledge the sources you used and people you worked with. Your professor reserves the right to give you an exam (oral, written, or both) to determine the degree that you participated in the making of the deliverable, and how well you understand what was submitted. For example, you may be asked to explain any solution that was submitted and why you choose to write it that way. This may impact the mark that you receive for the deliverable.

So, whenever you submit a deliverable, especially if you collaborate, you should be prepared for an individual inspection/walkthrough in which you explain what every line of assignment does and why you choose to write it that way.

```
/* CMPUT 201 Assignment #3                                          */
/* Due:    11:55pm, December 6, 2019;                               */
```

From Assignments #1 & #2, you have learnt what a sequence is, as well as a subsequence, a common subsequence, a longest common subsequence (LCS for short), a longest tandem subsequence (LTS for short), and a longest palindrome subsequence (LPS for short).

Let $S$ and $T$ be two given sequences. We can now similarly define a *longest common tandem subsequence* (LCTS for short) for $S$ and $T$ and a *longest common palindrome subsequence* (LCPS for short) for $S$ and $T$.

Assignment #3 is on computing an LCTS and an LCPS for any two given sequences of length up to $10,000$ over the digit alphabet $\Sigma = \{0, 1, 2, \ldots, 9\}$. Recall that your C programs for Assignments #1 & #2 are able to read in a sequence of length up to $10,000$, and are able to compute an LCS, an LTS, and an LPS.[1]

The following list contains the specifications for Assignment #3 (10 marks in total):

1. Write a multi-file multi-function C program with multiple functionalities (i.e. objectives).
   (a) Name the file `main.c` (C source file) to contain the definition of your `main()` function, which mainly displays the flow chart, *i.e.*, multiple flows of function calls according to command-line arguments;
   compose a corresponding `main.h` header file (C header file) to contain all the needed directives;
   (b) all codes for computing an LCS should be organized into two files called `lcs.c` and `lcs.h`;
   (c) all codes for computing an LTS and an LCTS should be organized into two files called `lts.c` and `lts.h`;
   (d) all codes for computing an LPS and an LCPS should be organized into two files called `lps.c` and `lps.h`;
   (e) all the other codes for utilities such as printing can be organized into two files called `utility.c` and `utility.h`.
   (f) Compose a `makefile` for compiling your program, that is, by running the `make` command, your program will be successfully compiled without any warning message.
2. Your program can be run with options through command-line arguments:

```
-g:    to generate an instance consisting of two sequences over the digit alphabet
-c:    to compute an LCS for the two input sequences
-t:    to compute an LTS for the input sequence
-p:    to compute an LPS for the input sequence
-ct:   to compute an LCTS for the two input sequences
-cp:   to compute an LCPS for the two input sequences
-i inputfilename:    to read in sequence(s) from file "inputfilename"
-o outputfilename:   to write all the results into the file "outputfilename"
```

The valid argument/option combinations are (`"a|b"` means `"a"` and/or `"b"`, and `"[a]"` means the argument `"a"` is or is not there):

```
-g [-o outputfilename]
-t|-p [-i inputfilename] [-o outputfilename]
-c|-ct|-cp [-t|-p] [-i inputfilename] [-o outputfilename]
```

[1] If you didn't do Assignment #1 and/or Assignment #2, you might consider purchasing a copy of each solution by paying 2 marks from your overall total.

The first is for random sequence generation; the second is to read in only one sequence and then computes; the last is to read in two sequences and then computes, and possibly to use the first sequence to compute an LTS and/or an LPS.

Note that the order of the options is irrelevant (just like our `gcc`) and the input sequences need to be validated (as done in Assignment #1, if invalid, reports it and terminates your program).

When invalid options are entered, such as `"-x"`, your program prints out all the above options with their objectives and terminates.

3. (a) For option `-g`, your program provides an interface to ask for the lengths of the two sequences:

```
Enter the lengths of the two sequences: 25 26
```

(The maximum length we will be testing for is 10,000.) Your program generates two random sequences of the two specified lengths, respectively, and writes them out to the `stdout` (separated by a newline character):

```
3110283025318292818318143
1311230172803163125184 1324
```

If the option `-o outputfilename` is supplied, then the two random sequences are written into and appended to the end of the file `"outputfilename"`.

(b) Options `-c`, `-ct`, `-cp` require two input sequences. If the option `-i inputfilename` is supplied, then the sequences are read from the instance file `"inputfilename"`;[2] otherwise, the following interface is provided for reading in two sequences:

```
Enter the first sequence: 3110283025318292818318143
Enter the second sequence: 1311230172803163125184 1324
```

Your program computes an LCS/LCTS/LCPS of the two sequences and writes it out to the `stdout` (the following `xxx` and `yyyy` should be replaced by the actual results):

```
# an LCS/LCTS/LCPS (length = xxx) for the two sequences is:
yyyy
```

If the option `-o outputfilename` is supplied, then the above is written into and appended to the end of the file `"outputfilename"`.

(c) Options `-t`, `-p` require only one input sequence. If the option `-i inputfilename` is supplied, then the sequence is read from the instance file `"inputfilename"`;[3] otherwise, the following interface is provided for reading in one sequence:

```
Enter the sequence: 3110283025318292818318143
```

Your program computes an LTS/LPS of the sequence and writes it out to the `stdout` (the following `zzz` and `wwww` should be replaced by the actual results):

```
# an LTS/LPS (length = zzz) for the sequence is:
wwww
```

If the option `-o outputfilename` is supplied, then the above is written into and appended to the end of the file `"outputfilename"`.

(d) Time your program: for each option among `-c`, `-t`, `-p`, `-ct` and `-cp`, report the average running time of your program on a sequence (or two sequences) of length 50, 100, 200, 500, 1000, 2000, 5000 and 10000, respectively, in seconds on our lab machines. The average should be taken over at least 10 runs.

[2] If the instance file contains more than two sequences, then only the first two sequences are read in; if the instance file contains less than two newline characters, then the second sequence is empty or both sequences are empty.

[3] If the instance file contains more than one sequence, then only the first sequence is read in; if the instance file is empty, then the sequence is empty too.

Include these results (an $8 \times 5$ table) as comments at the front of the C source file containing the `main()` function. For any entry you are not able to fill, place a `'-'` (dash) in it.

4. (Coding style requirement:) You should partition your source code into multiple files as indicated. A simple `makefile` is needed such that by typing `make` command, your program is successfully compiled without any warning message.

   Use space, tab, newline to create block structures in your code and place necessary comments when a block expands over 6 lines.

   //End of description of Assignment #3.