MIDDLE EAST TECHNICAL UNIVERSITY, NORTHERN CYPRUS CAMPUS
CNG315 Algorithms

Assignment 4: Pattern Detection in DNA Sequences

This assignment aims to help you practice **string processing** and **string matching.** Your main task in this assignment is to develop a simple application for pattern detection in DNA sequences using **C programming language**.

Overview

DNA, which stands for deoxyribonucleic acid, is the genetic material found in humans and nearly all living organisms (see Figure 1). Almost every cell in the human body contains the same DNA. The information stored in DNA is represented by a code consisting of four chemical bases: Adenine (A), Guanine (G), Cytosine (C), and Thymine (T).

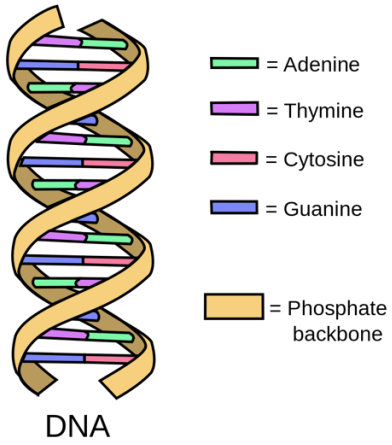

Figure 1: DNA[1]

In this assignment, you will need to develop a simple application that reads the DNA sequences in a given text file, asks the user to enter the patterns, and then lists how many times each pattern appears in how many sequences.

A **sample run** is provided below:

```
Enter the file path containing the DNA sequences: DNAsequences.txt
Enter the number of patterns: 4
Enter the pattern: TGGAT
Enter the pattern: GATG
Enter the pattern: TACCACAAA
Enter the pattern: GTAG

GATG is detected 10 times in 7 DNA sequences
GTAG is detected 5 times in 5 DNA sequences
TGGAT is detected 1 time in 1 DNA sequence
TACCACAAA is detected 0 time in 0 DNA sequence
```

Note: Please note that these patterns are sorted based on the number times they appear (GATG 10 times, GTAG 5 times, TGGAT 1 time, and TACCACAAA 0 time).

[1] https://acer.disl.org/news/2016/08/17/tool-talk-gene-sequencing/

Implementation Requirements

This application will have **a linked list for DNA sequences**. Each node in a linked list will keep one DNA sequence, so it should be created based on the following data structure. When you create your linked list, do not use a dummy node.

```
struct node{
   char sequence[81];
   struct node *next;
};
```

This application will also have **an array of data structures for patterns**. Each pattern will be represented with following data structure. The size of the array will be determined based on the user input (please see the sample run above)

```
struct pattern{
    char p[81];
    int times; //The number of times the pattern appears
    int sequences; //The number of sequences the pattern appears.
};
```

In this assignment, you are given a C file called "template4.c" attached. This will be the template code which you need to strictly follow. Inside the "template4.c" file, all function prototypes are declared. You will basically need to fill in the blanks and write the code for the following functions:

In addition to the main function, you will also need to implement the following functions and call them appropriately in the main function.

- **readSequences**: This function takes the file path, reads the sequences in the file where each line shows a different DNA sequence (maximum length can be 80), creates a linked list and stores these sequences in the linked list. It will then return the linked list of DNA sequences.
- **searchPatterns:** This function takes the linked of DNA sequences and the array of patterns, and then uses the Rabin-Karp Algorithm where d=4 (since there are 4 four chemical bases) and q=11 to calculate how many times each pattern appears and how many sequences includes each pattern and store these statistics in the array of patterns.

  The general pseudo-code of the **Rabin-Karp Algorithm** is given below.

```
RK(T, P, d, q)
      n := length[T];
      m := length[P];
      h := d^(m-1) mod q;
      p := 0;
      t_0 := 0;
      for i := 1 to m do
            p := (dp + P[i]) mod q;
            t_0 := (dt_0 + T[i]) mod q
      for s := 0 to n – m do
            if p = t_s then
                  if P[1..m] = T[s+1..s+m] then
                        print "pattern occurs with shift s"
            if s < n-m then
                  t_s+1 := (d(t_s – T[s+1]h) + T[s+m+1]) mod q
```

- **sortPatterns:** This function takes the array of patterns and sort them according to the number of times they appear by using the **Insertion Sort** in descending order.
- **printPatterns:** This function takes the array of patterns and prints them in the format as shown in the sample run.

Incremental and Modular Development

You are expected to follow an incremental development approach. Each time you complete a function or module, you are expected to test it to make sure that it works properly. You are also expected to follow modular programming which means that you are expected to divide your program into a set ofmeaningful functions.

Professionalism and Ethics

You are expected to complete the assignments on your own. Sharing your work with others, uploading the assignment to online websites to seek solutions, and/or presenting someone else's work as your own work will be considered as cheating.

Rules

- You need to write your program by using **C programming**.
- You are not allowed to change any code in the template, and you should abide by the prototypes and structures given. Changing anything in the template may result in an immediate o (zero). The places where you need to write your code are marked with the comment: "WRITE YOUR CODE HERE".
- **You need strictly follow the specifications given in this document.**
- You need to name your file with your student id, e.g., 1234567.c, and submit it to ODTUCLASS.
- It is suggested to use helper functions to support modular development.
- You need to ensure usage of comments to clarify your codes.

Grading Policy

The assignment will be graded based on the following scheme:

Grading Item	**Mark (out of 100)**
Main Function & Appropriate Function calls in the Entire Program	20
readSequences	10
searchPatterns	40
sortPatterns	20
printPatterns	10

Please note that if your function does not produce an expected output, **you will not receive any mark from that function**. You need to ensure your code runs and is testable. **If you submit a code which does not compile, you will automatically get zero.** Please note that code quality, modularity, efficiency, maintainability, and appropriate comments will be part of the grading. Therefore, **providing an expected output does not guarantee a full mark.**