1st Set of Lab Exercises

Part 1

Nucleotides are organic molecules that form the building blocks of nucleic acids (DNA) when three (3) or more are joined together to form a nucleic acid macromolecule. A nucleotide consists of (among other things) a nitrogenous base. The nitrogenous bases in DNA are adenine, guanine, thymine and cytosine and consequently we can symbolize the corresponding nucleotides that compose it with the initial letter of each nitrogenous base. Thus the nucleotide sequence in a DNA macromolecule can be represented as a sequence of the characters 'a', 'c', 'g' and 't', e.g.

```
c a a t g c t a g c a a t t g c t a g c
```

Each such sequence encodes the structure of a protein as follows: Three nucleotides of the DNA macromolecule specify an amino acid molecule. The sequence of amino acids specified by the consecutive triplets of DNA nucleotides corresponds to the macromolecule of a protein. Obviously, the specification of the protein depends on the point from which this process of matching triplets of nucleotides with amino acid molecules will begin. The precise finding of this point (if it exists) for the correct coding of a given protein is the subject of this exercise.

For the purposes of this exercise, we define a genetic sequence as a sequence of characters each of which can take the values 'a', 'c', 'g' or 't'. The coding for an amino acid is a genetic sequence of length 3. There are 25 amino acids, which we will refer to by the capital letters (of the English alphabet) from 'A' to 'Y'. A protein corresponds to a sequence of amino acids, that is, to a sequence of characters each of which can take the values 'A' to 'Y'.

Since an amino acid, as defined, corresponds to a genetic sequence of three characters (nucleotide codes) and since the possible characters at each of the three (3) positions of the sequence are four (4), it follows that in total there are 43 = 64 different encodings of amino acids, each of which corresponds to exactly one amino acid (the encoding is “well defined”), but an amino acid can have more than one encoding (the mapping function is not “1-1”).

Now, given a protein and a large genetic sequence that may contain the protein, the task is to determine the genetic coding of the amino acids in the protein. An example will illustrate the process we use to do this:

Suppose we know that the protein M S I Q H M R is located somewhere in the genetic sequence attgctagcaatgctagcaattgctagcaatt but we don't know where, and we don't know any coding for the amino acids. First, try aligning the first few characters:

```
a t t g c t a g c a a t g c t a g c a a t t g c t a g c a a t t

M       S       I       Q       H       M       R
```

This approach would work if a code for M was att, for S it was gct, etc. But this is not the case because gct cannot be a code for both S and H, as shown above. Having two different genetic

codes for an amino acid (for example, M is coded as att or agc in this case) is allowed, but each genetic code must represent only one amino acid.

The sequences do not align on the first attempt, so we try to align the protein sequence by the second character of the genetic sequence:

```
a t t g c t a g c a a t g c t a g c a a t t g c t a g c a a t t
  M     S     I     Q     H     M     R
```

Again, cta cannot encode two different amino acids (S and H), so this alignment cannot be accepted either. The same happens if we start from the third character of the genetic sequence. Let's finally see what happens if we start from the fourth character:

```
a t t g c t a g c a a t g c t a g c a a t t g c t a g c a a t t
      M     S     I     Q     H     M     R
```

In this case there are two problematic triples: agc corresponds to S and H and gct corresponds to M and Q. So this alignment is also rejected. Continuing to shift the protein sequence along the genetic sequence in this way, we will eventually have an alignment that could work:

```
a t t g c t a g c a a t g c t a g c a a t t g c t a g c a a t t
                    M     S     I     Q     H     M     R
```

This alignment shows that atg and agc are the genetic code for the amino acid M, cta is a code for S, gca for I, etc.

This is an artificial example to show how the method works. In real data, the sequences are much longer.

Start with the skeleton program provided with the exercise (file main.c). In this program, an array of characters from a specific genetic sequence is initialized. Extend the program to ask the user to enter a protein sequence from the keyboard. Assume that the protein sequence is at most 300 characters long. The program will read the protein and then try to find a valid alignment with the specific genetic sequence. If such an alignment exists, it will find it and print on the screen the genetic coding of the amino acids as well as the position in the genetic sequence table where the alignment found begins. If there is no valid alignment, the program will print a message that it was unable to find a valid genetic coding of the amino acids.

ATTENTION: Your program should perform all the necessary checks that will ensure its correct operation.

Part 2

The Fibonacci sequence is defined recursively for each non-negative integer n as follows:

$F(n) = F(n-1) + F(n-2)$, $n>1$ and $F(0)=0$, $F(1)=1$.

Based on this definition, the calculation of the value of the sequence for a given n can be implemented either with a recursive function, according to the above definition, or with a non-recursive function that starts the calculation from the smallest values of n and proceeds to the value for the given n, keeping at each step the last two values of the sequence that have been calculated so far.

DEVELOPMENT OF AN INITIAL PROGRAM

Develop a program that will implement both methods of calculating the Fibonacci sequence through two functions, one recursive and one non-recursive respectively.

The program will initially ask the user to enter a positive integer and then calculate the value of the Fibonacci sequence with this number as input by successively calling the two functions you implemented and printing the final result that each one gives, indicating which function gave each result.

FIRST EXTENSION

You are then asked to extend the program you created so that each time a function call starts, it prints an appropriate message of the type:

Computing fib(...)

where the input value will appear in place of the ellipsis. When the call is finished, it will print a message of the type:

fib(...) finished with result ...

In the case of a recursive function, the above messages should be displayed for each recursive call, allowing the user to see how the recursion works.

Change the function name (fib) in the above sample messages to match the names you have used for the two functions you implemented.

SECOND EXTENSION

Finally, extend the program that resulted from the first extension so that it calculates the number of additions that are performed during each call of a function. In the case of a recursive function,

the calculation of the number for each call will also add the number of operations performed by the individual recursive calls. What do you observe? Which of the two functions is more economical in terms of the number of additions performed?

DELIVERABLE

Your deliverable will be the program that resulted after the implementation of the second extension. This program will include, among other things, detailed comments on the observations you made comparing the number of operations performed during its execution (after implementing the second extension).

ATTENTION: Your program should perform all the necessary checks that will ensure its correct operation.