Data Structures Practical
Week 3

Tom Friedetzky

1. Consider the following code:

```
 1:  Input:  integer L, integer H
 2:  Output: ?
 3:  integer x = L
 4:  integer p = 1
 5:  while (x <= H) do
 6:      if (x is odd) then
 7:          p = p * x
 8:      endif
 9:      x = x + 1
10:  endwhile
11:  print "result is ", p
```

   (a) What does this algorithm do, that is, what is being computed in variable `p`?

   (b) Rewrite this code to use (i) the `do-while` construct, and (ii) the `for-endfor` construct.

2. Consider the "*string matching*" problem. Suppose some string T consisting of n many characters is stored in some array `T[1...n]`, and some shorter string S of length $m < n$ is stored in some other array `S[1...m]`. We assume that each character of each string is stored in a memory cell on its own, and that consecutive characters in any given string are stored in consecutive memory cells. Write code that prints out all positions k at which we find exact copies of S in T. For example, if

```
T = "Go Canucks Go!"
S = "Go"
```

   then n would 14 and m would be 2, and your code should output the numbers 1 and 12, as these are the (only) positions in T where we can find an exact copy of S. If S doesn't occur in T at all then your code should print a corresponding message. You may refer to individual characters within S or T by indexing them explicitly, i.e., you may use $T[i]$ or $S[j]$ to refer to the i-th character of T and the j-th character of S, respectively. This is the level of detail that I'm expecting: any comparisons should be on individual characters.

3. Recall that a natural number k is called prime if it can be divided without remainder only by one and itself (e.g., 17 is prime, 18 isn't). Write a simple algorithm in pseudo-code that tests whether a given number, i.e., a number that is given as input, is prime. You may use a condition like "*a divides b*".