Exercise 1

In the file data.csv you find 100 lines with the format ID, "TITLE" from stackoverflow.

a) Read the file and print the longest title.
b) Read the file and print for each line the ID and the number word in the titles.
c) Read the file and create a new CSV file with the format ID, "WORD" for all words in the title.

Exercise 2

In this exercise, you must make a Reverse Polish Calculator. You can find a description on Wikipedia. You should use the description in the "Postfix algorithm" section as basis for your implementation of the calculator. Your implementation should accept expressions (strings) as input and compute the result of the expressions. So, given, for instance, the string "5 5 +" must return 10, "5 5 2 * +" must return 15, etc. The format of the input is a string with integers and operators separated by spaces (like in the above examples). Your implementation must at least support the following operations: addition, subtraction, multiplication, and division. You can add as many operations as you like, even functions like cos, sqrt, pow (^), etc.

The calculator must be tested with a number of unit tests, you must at least test this:

- Given an empty or null string it should return the result 0
- All implemented operations
- Complex expressions, i.e. expressions with two, three, and four different operators
- The input "5 1 2 + 4 * + 3 -" should give 14 as result

Try to find test for any important border cases, so the calculator is as stable as possible, and prepared for future changes/expansions.

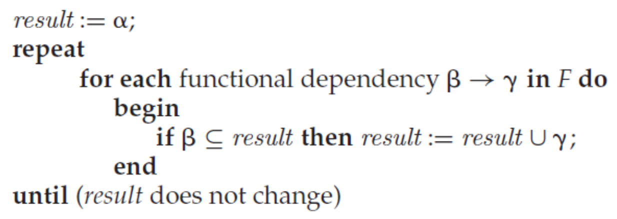
Optional: If you want your calculator to be able to compute expressions with infix notation, you could implement the Shunting-yard algorithm, which convert expressions in infix notation to reverse polish notation, e.g.

"3 + 4 * 2 / (1 - 5) ^ 2 ^ 3" => "3 4 2 * 1 5 − 2 3 ^ ^ / +"

and it will thus be possible to use your reverse polish calculator. If you choose to implement this too, remember unit testing.

Exercise 3

In the database book from section 1 you find the pseudo code for an attribute closure algorithm on page 341:



```
result := $\alpha$;
repeat
        for each functional dependency $\beta \rightarrow \gamma$ in $F$ do
            begin
                if $\beta \subseteq$ result then result := result $\cup \gamma$;
            end
until (result does not change)
```

Figure 8.8 An algorithm to compute α^+, the closure of α under F.

a) Create a class that can represent a functional dependency, i.e. a set of attributes that determent another set of attributes, e.g. ABC -> DE. The class should have two properties, LeftHandSide and RightHandSide, returning the left and right side of the functional dependency.
b) Create a program that can compute the closure given the starting attribute(s) and a set of functional dependencies represented as a list/array of the class created in 3.a.
c) Extend the functionality with the function FindKeys(relation, functionalDependencies), with a relation (i.e. a set of attributes) and a set of functional dependencies as arguments, that can find all candidate keys (minimal keys) for the given relation, e.g. FindKeys({A,B,C}, {A->B, B->C}) = A.