Assignment 0: Hello 3D5A

This lab is not graded, but you should submit it via Submitty anyway just to make sure that everything is working.

Deadline: 26/09/22 5pm

Goals:

- Get up and running with Visual Studio Code or any editor of your choice.
- Get to grips with the basics of I/O in C
- Parse a CSV file so we can do some work with real data

Part 0 – Hello World

Use the file provided to set up Visual Studio Code, or any C editor of your choice, and run a "Hello World" application.

Listing 0: Sample Output

```
Hello, world!
```

No need to submit anything for this part.

Part 1 – Practicing input and output

This part is just to help you find your feet with Visual Studio Code and the C programming language. It also gives us a sense of where everyone stands when it comes to coding! Try to get through this nice and quick and move on to the problem in Part 2.
Remember the difference in how we handle input/output in C. Documentation with code samples can be found here:

http://www.cplusplus.com/reference/cstdio/printf/
http://www.cplusplus.com/reference/cstdio/scanf/

Write a program which prompts the user for their name and their age. Load these values into some suitable variables and then print the results to the terminal as shown below

Listing 1: Sample Output

```
Name >> Gary
Age >> 27
Hello Gary (27)
```

Submit your file for part 1 on Submitty.

Part 2 – A simple CSV parser

This being a module on data structures and algorithms, it would probably be a good thing if you could actually load some data. A popular format for sharing relatively simple data is Comma Separated Values (CSV). Excel and Libre Office Calc can both read and export in this format, so it's a good one to know.

In CSV, every line of a file describes a "record". Each field of the record is separated by a comma. For example, a CSV file containing student information might look something like the following

Listing 2: Sample CSV file with headers (you can usually throw away the first line)

```
Student Number,Forename,Surname,Course,Description
08493214,John,Doughboy,Engineering,"Hey guys!"
08452343 , Alice , Totesreal , Engineering ," Like ,  what?"
```

Spaces are allowed in fields, but commas that are not intended to escape the field should be placed in quotes (as in the Description field shown above).

Your task is to write a program that can read a CSV file **provided as an argument** and split it up into its constituent fields and records. If all goes well, you will be able to use your parser to load our datasets for the rest of the semester. Start with a simple implementation that prints one field per line and a double newline whenever the end of a record is reached. After that you can worry about storing the CSV data in some structs and use them to do something interesting.

Listing 3: Sample Output

```
08493214
John
Doughboy
Engineering
Hey guys!

08452343
Alice Totesreal
Engineering
Like, what?
```

One of the best way to implement this is with a function like the one given below. This function reads the file one field at a time and stores the field's value in the buffer. If the function encounters a newline or the end of the file, then it will return a 1 (meaning we have finished reading the record). Otherwise it returns 0.

```
int next_field( FILE *csv, char *buffer, int max_len );
```

I recommend solving this problem is simple stages:

1. Create a function like the one shown above. The function takes a FILE pointer, an array and the length of the array as input.

   Inside the function, read the file one letter at a time. (You might want to have a look at `http://www.cplusplus.com/reference/cstdio/fgetc/)`

2. If the letter is a comma, then the function should return 0. Otherwise it should store the letter in the buffer.

   Hint: remember how strings are stored in C!

3. If the character is a newline character ('\n'), or you have reached the end of the file, then you should return a 1 (this means that the field you have just read is the last field on the line).

   You can check when you have reached the end of the file using the `feof` function (documentation here: http://www.cplusplus.com/reference/cstdio/feof/)

4. Use this to implement the simple version of the parser, which prints one field per line and a double newline whenever the end of a record is reached.

You only need to submit 1 file for task 2 and 3 – but there are 2 tests to check each task separately.

Part 3 – Handle quote marks

Extend the function from Part 2 so that it can handle quoted fields. For this part, you should maintain a boolean which is true when you are inside quotes and false otherwise. If you encounter a comma but the boolean is true, then you keep reading from the file. Otherwise the function should stop reading and return. Note that the function will always return on a newline or end of file.

The quote character is basically an escape character. What this means is that whenever you encounter a quote, you should not store it in the array. Instead you should flip the value of the boolean and then immediately load the next value from the file stream and store it in the array.

This is actually a very simple function. If you find yourself writing 20 lines of code or more, then you might be overthinking the problem. If you have any questions or are struggling with the problem at all, just give one of your demonstrators a shout and they'll do their best to help you.
Submit (or update) your file for parts 2 and 3 to Submitty.

Part 4 – Wrap it all up!

Now extend your parser to store the data in a struct.
Once you have read the entire file "pokemon.csv", try to do some processing on the data. What is the average attack strength of a character?

Submit your file for part 4 to Submitty.

Extra bit

Kaggle is a website where people routinely share datasets in CSV format. There are tons of great stuff up there including film reviews, polling numbers, crime statistics and more. In fact, the CSV data we provided for this lab was obtained from Kaggle. Check it out and see how well your parser manages to read some actual data files.