Summary

You need to implement an inference engine for propositional logic in software based on the Truth Table (TT) checking, and Backward Chaining (BC) and Forward Chaining (FC) algorithms. Your inference engine will take as arguments a Horn-form Knowledge Base **KB** and a query **q** which is a proposition symbol and determine whether **q** can be entailed from **KB**. You will also need to write a report about how your program works with different knowledge bases and queries.

Implementation

You are encouraged to implement your software using Python

TT, FC and BC Inference Engine

The Truth Table Checking (*TT*) algorithm works with all types of knowledge bases. The Forward Chaining (*FC*) and Backward Chaining (*BC*) algorithms work with Horn-form knowledge bases. (See the description in Artificial Intelligence – A Modern Approach – the 3rd edition 2010, page 256-259.)

Given a knowledge base **KB** in Horn form (with TELL) and a query **q** which is a proposition symbol (with ASK), your program needs to answer whether or not **q** is entailed from **KB** using one of the three algorithms *TT*, or *FC*, or *BC*.

File Format: The problems are stored in simple text files consisting of both the knowledge base and the query:

- The knowledge base follows the keyword TELL and consists of Horn clauses separated by semicolons.
- The query follows the keyword ASK and consists of a proposition symbol.

For example, the following could be the content of one of the test files (`test1.txt`):

```
TELL
p2=> p3; p3 => p1; c => e; b&e => f; f&g => h; p2&p1&p3 =>d; p1&p3 => c; a; b;
p2;

ASK
d
```

Command Line Operation

program needs to operate from a DOS command-line interface to support batch testing. A DOS command-line interface can be brought up in Windows 7/8/10 by typing **cmd** into the search box at the ***Start*** button. However, please ensure that your program works on Windows 10 because we will be testing your program on Windows 10. This can be accomplished with a simple DOS .bat (batch) file if needed. Below are the three different arguments formats you need to support. Note the unique argument count for each.

`C:\Assignments> iengine <filename> <method>`
where **`iengine`** is your .exe file or a .bat (batch) file that calls your program with the parameters, **`filename`** is for the text file consisting of the problem, and **`method`** can be either `TT` (for Truth Table checking), or `FC` (for Forward Chaining), or `BC` (for Backward Chaining) to specify the algorithm.

(if you program in Python, we can accept Python scripts and execute your scripts using the following command from the CLI:

```
C:\Assignments> python iengine.py <filename> <method> )
```

For instance:
```
> iengine test1.txt FC
```

Standard output is an answer of the form YES or NO, depending on whether the ASK(ed) query **q** follows from the TELL(ed) knowledge base **KB**. When the method is *TT* and the answer is YES, it should be followed by a colon (:) and the number of models of **KB**. When the method is *FC* or *BC* and the answer is YES, it should be followed by a colon (:) and the list of propositional symbols entailed from **KB** that has been found during the execution of the specified algorithm.
For example, running **`iengine`** on the example `test1.txt` file with **`method`** TT should produce the following output:

```
> YES: 3
```

On the other hand, when I run my implementation of **`iengine`** with **`method`** FC on the example `test1.txt` it produces the following output:

```
> YES: a, b, p2, p3, p1, c, e, f, d
```
Note that your implementation might produce a different output and it would still be correct, depending on how you store and order the sentences in the knowledge base. I'll check that carefully to ensure that you won't be disadvantaged if your results don't look exactly the same as our results.

And running **`iengine`** with **`method`** BC on the example test file above should produce the following output:

```
> YES: p2, p3, p1, d
```

One potential research idea is to allow your program to deal with general knowledge bases that don't have to be a Horn knowledge base. Then you can implement your Truth Table algorithm to deal with the general knowledge bases and also implement a generic theorem prover such as a resolution-based one.

If you want to implement inference engine for general knowledge bases, please use the following syntax for different logical connectives:

~ for negation (☔)
& for conjunction (☂)
|| for disjunction (☂)
=> for implication (🦟)
<=> for biconditional (☂)