DCU School of Computing
Assignment Submission

Student Name:	Kyrylo Khaletskyy
Student Number:	15363521
Programme:	BSc. in Computer Applications
Project Title:	Comparing Functional and Logical Programming
Module code:	CA341
Lecturer:	David Sinclair
Due Date:	10/11/17

Declaration

I declare that this material, which I now submit for assessment, is entirely my own work and has not been taken from the work of others, save and to the extent that such work has been cited and acknowledged within the text of my work. I understand that plagiarism, collusion, and copying is a grave and serious offence in the university and accept the penalties that would be imposed should I engage in plagiarism, collusion, or copying. I have read and understood the Assignment Regulations set out in the module documentation. I have identified and included the source of all facts, ideas, opinions, viewpoints of others in the assignment references. Direct quotations from books, journal articles, internet sources, module text, or any other source whatsoever are acknowledged, and the source cited are identified in the assignment references.

I have not copied or paraphrased an extract of any length from any source without identifying the source and using quotation marks as appropriate. Any images, audio recordings, video or other materials have likewise been originated and produced by me or are fully acknowledged and identified.

This assignment, or any part of it, has not been previously submitted by me or any other person for assessment on this or any other course of study. I have read and understood the referencing guidelines found at http://www.library.dcu.ie/citing&refguide08.pdf and/or recommended in the assignment guidelines.

I understand that I may be required to discuss with the module lecturer/s the contents of this submission.

Signed: Kyrylo Khaletskyy

Date: 10/12/2017

Functional LCP

The functional language I implemented longest common prefix in is Haskell. Tests can be seen in the image below.

```
GHCi, version 7.10.3: http://www.haskell.org/ghc/  :? for help
Prelude> :load functional.hs
[1 of 1] Compiling Main             ( functional.hs, interpreted )
Ok, modules loaded: Main.
*Main> lCP ["interview", "interrupt", "integrate", "intermediate"]
"inte"
*Main> lCP ["hello", "hell", "he"]
"he"
*Main> lCP []
""
*Main> lCP ["interview"]
"interview"
*Main>
```

The Functional implementation begins by defining the lCP function to input a list of strings and output a single string like shown in the image below. After that the there are two checker functions, the first one checks if the initial list is empty and the second function checks if one of the words in the list is empty, in both cases the program will return an empty string, showing that there is no common prefix among the strings provided. The third function lCP takes the first character in the first word as the common prefix to check and mark the first word as accepted in this common prefix and pass it to the prefix function.

```
lCP :: [String] -> String
lCP [] = []
lCP ([]:xs) = []
lCP ((x:xs):xss) = prefix (xss) [x:xs] [x]
```

The second helper function prefix takes in two list strings and a string and outputs a string as seen in the picture below. In the first prefix function if the accepted words take another character from the first word in the accepted list then add it to the common prefix to check and accept the word but if the first word in not longer than the common prefix then it is equal to the longest common prefix. The second prefix function checks the current word and if it is accepted mark it as accepted and check the next word. But if one fails then the last common prefix is the longest common prefix, therefore ignore the last character.

```
prefix :: [String] -> [String]-> String -> String
prefix [] (x:lst) cp | length x > length cp = prefix lst [x] (take ((length cp)+1) x)
                     | otherwise = cp

prefix (x:xs) lst cp | length x >= length cp && (take (length cp) x) == cp = prefix xs (x:lst) cp
                     | otherwise = init cp
```

Logic LCP

The logic programming language I implemented longest common prefix in is Prolog. Tests can be seen in the image below.

```
?- lCP(Longest,["interview", "interrupt", "integrate", "intermediate"]).
Longest = inte.

?- lCP(Longest,["hello", "hell", "he"]).
Longest = he.

?- lCP(Longest,[""]).
Longest = ''.

?- lCP(Longest,["interview"]).
Longest = interview.

?-
```

The Logic implementation begins by defining the lCP predicate, this predicate takes in a List of strings and outputs a single String. The main predicate uses the findall inbuilt function to create a list of instantiations of the helper predicate (outlined later) which obtains a list of prefixes common across all string contained in the inputted list. The last function takes the final prefix which was added into the new list previously, this prefix is the longest, as the order of the prefixes goes up with every iteration. The toString predicate takes the longest prefix and passes it onto the toString method.

```
lCP(Str,List):-
    findall(Z,helper(Z, List),X),
    last(X,Longest),
    toString(Str,Longest).
```

The toString method takes in a list of atoms and converts it to a single String using the inbuilt atom_chars function, and returns it back to lCP.

```
toString(Str,Longest):-
    atom_chars(Str,Longest).
```

The helper predicate takes in a list of words and outputs a list of atoms, the inbuilt function prefix takes the prefix of each word and the inbuilt function maplist check if the prefix can be successfully applied to each word in the list, when the program reaches a point where the next character doesn't equal to the prefix of other words the predicate terminates and returns a list of successful prefixes back to the findall function mentioned above.

```
helper(Z, List) :-
    maplist(prefix(Z),List).
```

Comparison

Functional programming is a paradigm which is based on lambda calculus. Functional programming executes programs by evaluating expressions, it avoids mutable data, so any data inside the application cannot be changed and only copied somewhere else, we may only modify the copy of the original data. Some inputs are coming to a function without a side effect, without any modification a new output is generated.

Logical programming is a paradigm which is based on the predicate calculus, it uses logic expressions to evaluate programs, rules are written as logical clauses with a head and body. Logic programming languages specify a set of attributes that a solution must have, rather than a set of steps to obtain a solution. It describes the program in terms of the properties of the solution. Backtracking is used by logic programming languages to compute results. Logic programming uses:

1. Facts: parent(John, Mary)
2. Rules: child(X,Y) :- parent(Y,X).
3. Queries: parent(Y,Mary).

Although both programming styles are declarative they are very different. The biggest difference between both programming paradigms is that Functional Programming uses mathematical expressions to solve problems whereas Logic Programming uses logic expressions. The functional programming uses functions which returns a value, on the other hand logic programming uses predicates which give a True or False as a result rather than values.

In my functional implementation of longest common prefix the program recursively checks and accepts/rejects the character in each word and from here the new prefix is built recursively as each character in each word is checked each character which has been accepted is added to a list. Whereas the logic implementation builds a list of prefixes while a match is present, you pass through a condition which is either true or false, and as soon as the predicate gives a false output the prefix is handed over to a the toString method which formats the list of atoms as needed.

References

Maryrosecook. A practical introduction to functional programming [Online].
Available from:
https://maryrosecook.com/blog/post/a-practical-introduction-to-functional-programming

courses.cs.vt.edu. Programming Languages [Online].
Available from:
https://courses.cs.vt.edu/~cs3304/Spring02/lectures/lect08.pdf

CA341 David Sinclair Comparative Programming Languages [Online].
Available from:
http://www.computing.dcu.ie/~davids/courses/CA341/CA341.html

CA341 David Sinclair Prolog [Online].
Available from:
http://www.computing.dcu.ie/~davids/CA208_Prolog_2p.pdf