CA341

*Comparative Programming Languages*

***Student:*** *Nigel Guven*

***ID:*** *14493422*

Formal Logic Programming v Functional Programming

Analysis Report

**Due Date:** *10am 11th December*

1. **Assignment Objective**

Implement the following program in a ***logic programming language*** and a ***functional programming language*** wherein the program returns the ***longest common prefix*** of a ***list of strings*** of ***variable length***. For example, given the list

[“interview”, “interrupt”,”integrate”, ”intermediate”]

the longest common prefix is “inte”.

1. **Table of Contents**
2. **Logic Paradigm 4**

**1.1 Overview 4**

**1.2 Deductions 4**

**1.3 Negations 5**

**1.4 Non-Procedural Programming 5**

1. **Functional Paradigm 6**

**2.1 Overview 6**

**2.2 Higher-Order Functions 6**

**2.3 Immutable Values 7**

**2.4 Lazy Evaluation 7**

**2.5 Referential Transparency 7**

1. **Prolog – Logic Style Language 8**

**3.1 Implementation Overview 8**

**3.2 Logical Code 9**

**3.3 Interpretation 9**

1. **Haskell – Functional Style Language 11**

**4.1 Implementation Overview 11**

**4.2 Functional Code 12**

**4.3 Interpretation 12**

1. **Analytical Comparison 13**
2. **References 14**
3. **Logic Paradigm**

**1.1 Overview**

Logic Programming is a type of programming paradigm which is primarily based upon formal logic. It is based upon predicate calculus where the program is based upon a set of facts which can be seen as either statements or relationships which are of course true. There are then axioms and clauses which act like functions and can either be true or false. Logical Languages are high-level which require translation into machine language before they can be executed. There are three main concepts in logical programming: deduction; negations; non-procedural programming. Examples of logic languages are Prolog, F-Logic and QL.

**1.2 Deductions**

Computation is made by logical deduction. Data is expressed by statements which are either true or false. First-Order Logic is the construct for the design of the language which allows for deduction to occur. Given the statement ‘Charles Babbage invented the first mechanical computer’, through deduction we can create the clauses, ‘there exists X such that X is Charles Babbage and X invented the first mechanical computer’. Deductions are axioms that may infer interpretative statements upon a supplied set of facts.

**1.3 Negations**

Negation by Failure is an inference rule in logic programming. It is used to derive a statement given that all possible clauses have failed. NAF is primarily used to achieve completeness in a logical language where for every axiom in a program, the axiom may produce a demonstrable value. This is also known as being semantically valid.

**1.4 Non-Procedural Programming**

Logic Programming Languages incorporate the traits of the non-procedural paradigm. This paradigm specifies what needs to be computed but not how it should be done. The programmer must specify the set of items involved in computation, the relationships that some of them may have and the constraints which must hold to some Boolean value for the problem to be solved. The compiler then decides how to satisfy the constraints that are made. Another name for this paradigm is the declarative paradigm.

1. **Functional Paradigm**

**2.1 Overview**

Functional Programming is a paradigm which treats the computation of functions as mathematical expressions. The order of evaluation is dependent upon how the expression is declared. The programmer must understand what is to be computed rather than how it should be computed. Functional programs do not contain any variables; ergo data is immutable and circumvented of changing states. The expressions in this paradigm also have no side-effects which output anything other than the actual value.

The removal of variables in functional programming comes with its own problems. Iterative constructs like loops are non-existent which means the only way to implement iteration is through recursive functions. Examples of functional programming are Haskell, Wolfram and Lisp.

**2.2 Higher Order Functions**

Higher-Order Functions take other functions as arguments or can return a function as its output. They are essential in functional programming languages because they allow for more useful concepts like recursion and induction to take place with the program.

**2.3 Immutable Values**

Immutable values are common in functional programming. They do not allow an object to be modified from its created state. This makes the data more secure. In functional programming, you would compose functions to obtain a value that is required. The value cannot be saved and changed later. Rather than focussing on designing the data, there is an incentive for examining what a function can do.

**2.4 Lazy Evaluation**

Functional languages evaluate by needs. They do not have constructs for calling by value/reference. If the function argument is evaluated the value is stored for some purpose. Often referred to as lazy evaluation, it avoids the need to repeat evaluations which can reduce the running time of a program. This strategy for evaluation only fails when combined with imperative features such as exception handling and I/O because the order in which they operate becomes indeterminate.

**2.5 Referential Transparency**

When an expression is used we are only interested in what it can output as a value. Functional programs do not have assignment statements which allow for variables to change. An expression is declared to be referentially transparent if it can be replaced with the value it should output and not change the overall behaviour of the program. These expressions are known as pure functions and are a key concept of the functional paradigm.

1. **Prolog – Logic Style Language**



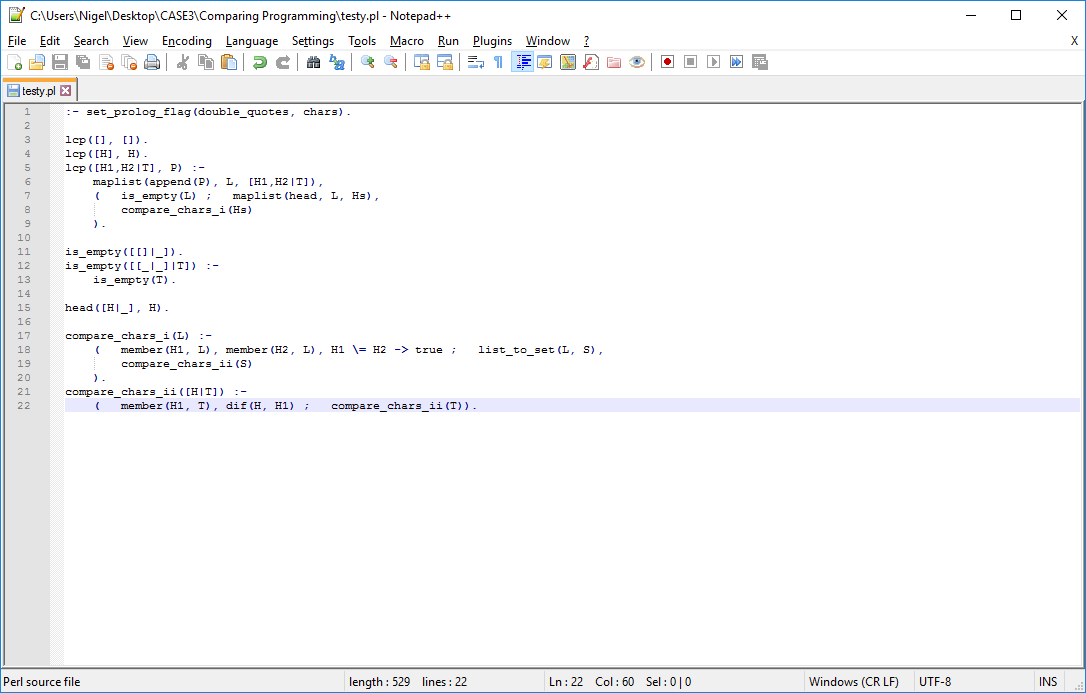
**3.1 Implementation Overview**

I will be using Prolog for the logic style of programming.

Prolog is a declarative formal-logic language which contains within a standard program, a set of facts and a list of expressions which create a relation upon the former. Queries can made in the Prolog shell which calls expressions and forms a result of either true or false.

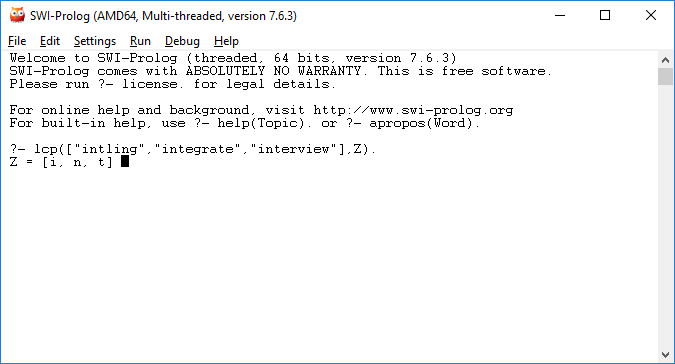
Prolog is very useful for parsing binary trees through the act of pattern matching which checks a sequence of items for the presence of some data. Prolog is strictly involved in non-numeric computation although it can read integers but is unable to perform mathematical equations due to its limitations as a purely logic language. Prolog is also utilised in artificial intelligence where the manipulation of symbols is performed.

**3.2 Logical Code**



**3.3 Interpretation**

This implementation of the longest common prefix assignment is designed to search a set of lists of unknown length. The query format in the Prolog shell is:



The query returns Z which is some variable and contains a list of characters that display the longest common prefix of the list elements. Prolog generally outputs true or false statements so therefore the set\_prolog\_flag definition permits the query to output the list of characters.

The program searches the strings recursively. This is done by the main lcp function searching for the empty lists and returning an empty list. This is the base case. There are two following steps that are inductive. The former requires the prefix of only two items and the latter then further complicates this by looking for a set of lists of an infinite amount.

The maplist function is built into the prolog library and its returns true if some goal can be applied to the elements of a list which it takes as a parameter. The list\_to\_set function checks whether one list has the same elements as another.

There is a predicate function called compare\_chars which compares two characters and returns true until the characters do not match. If there are more lists to parse, then a second function checks the list of new atoms to find the comparable characters. It cecks if the head of the list is a member of the tail ,otherwise dif, another function only returns true if its parameters are different.

1. **Haskell – Functional Style Language**

****

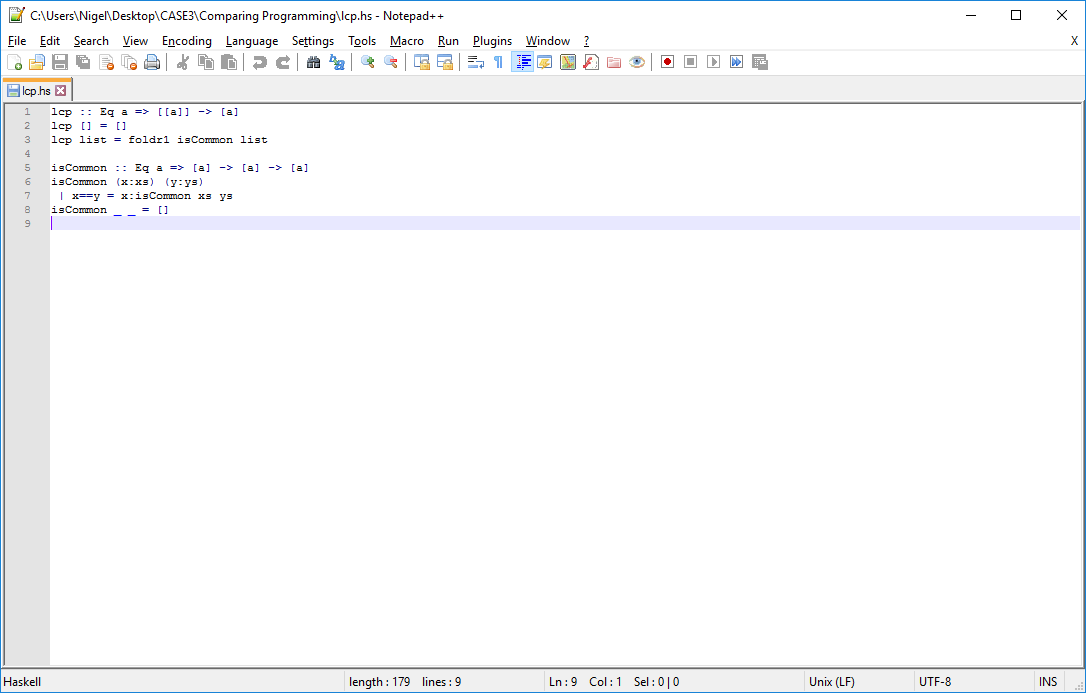
**4.1 Implementation Overview**

I will be using Haskell to comprise the functional programming aspect of the assignment.

Haskell is a purely functional language with no side-effects and a limited referential transparency. It does not evaluate an expression unless that expression is necessary to the overall outcome of the program. Functions calculate an expression and return a result.

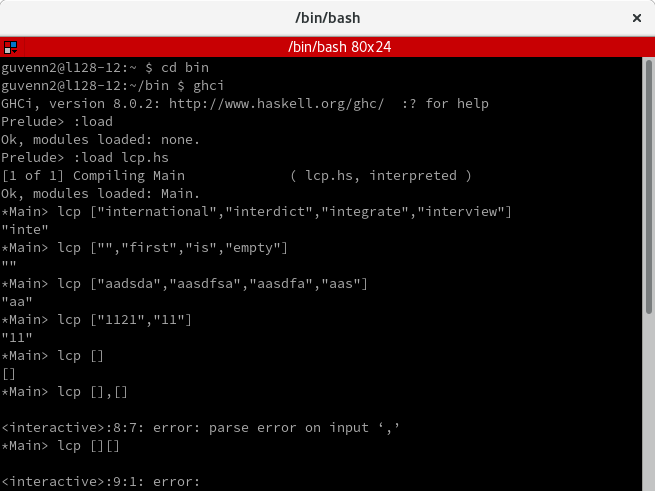
Haskell is statically typed which is where the compiler knows which code is an integer and which is a float. This allows for errors to be caught at compile-time. Haskell also has type-inference that it can deduce that a variable name is of some data type, so variables do not need to be declared for some data type. Haskell is often compared to imperative languages as it is a high-level language, usually shorter and neater which produces mostly bug-free programs.

**4.2 Functional Code**



**4.3 Interpretation**

The above section returns the longest common prefix from a list of strings in the following format:



The method utilises polymorphism through the polytype ‘a’. The first argument is the input of a variable amount of strings contained in an array. The second argument produces the result stored as an array but through the console is displayed as a string. This is because strings in Haskell are viewed as character arrays. The list of strings is passed through a second function which examines the list recursively and, using foldr1, which works backwards to compare the commonality of characters in each string of the list.

Once there are two matching string prefixes, the ‘lcp’ function then moves onto the next two strings from the tail of the list.

1. **Analytical Comparison**

One can plainly see that the Haskell program has fewer lines of code and therefore is more user-readable and presumably efficient for resources. Haskell uses functions to achieve its goal while Prolog uses predicates which do not necessarily return a value. The Prolog program required the writing of exceptions to allow for the output of non-boolean data. Haskell did not require any imports from its library. There were many expressions to be written and recursive properties to be implemented in the Prolog program which produced many bugs before being weaned out through testing.

I had written the program in Java before starting either task. I then translated the Java code as close as possible to the Haskell and Prolog program. It was much easier to translate it into Haskell. My Prolog program was also dependent upon the Prolog library and documentation.

In contrast, my Haskell program was more efficient in that the code was short and could be better understood. I had problems with using specific data types for arguments, so I instead chose to use the polytype which I had learned about in my CA320 module. I did not have to use the Haskell library which made it easier for me to debug the program.

In conclusion, I would much prefer to write this style of program in the functional language, Haskell. It is more efficient with handling string than the logic equivalent. Prolog was harder to implement with manipulating strings. Of course, an imperative or Object-Oriented language would be even more efficient than either.

1. **References**

<https://www.csee.umbc.edu/courses/331/fall12/notes/prolog/prologoverview.ppt.pdf>

<https://thebittheories.com/levels-of-programming-languages-b6a38a68c0f2>

<http://wiki.c2.com/?LogicProgramming>

<http://www.computing.dcu.ie/~davids/CA208_Prolog_2p.pdf>

<http://cs.union.edu/~striegnk/learn-prolog-now/html/node90.html>

<https://en.wikipedia.org/wiki/Functional_programming#Use_in_industry>

<http://rextester.com/l/haskell_online_compiler>

<https://hackage.haskell.org/package/base-4.10.1.0/docs/Data-Eq.html>

<https://medium.com/@cscalfani/so-you-want-to-be-a-functional-programmer-part-1-1f15e387e536>

<https://maryrosecook.com/blog/post/a-practical-introduction-to-functional-programming>

<http://hkupty.github.io/2016/Functional-Programming-Concepts-Idioms-and-Philosophy/>

<http://www.computing.dcu.ie/~hamilton/teaching/CA320/notes.html>

<https://bernardopires.com/2013/10/try-logic-programming-a-gentle-introduction-to-prolog/>

<http://learnyouahaskell.com/introduction>

<http://i3.cpcache.com/product/1157449058/swiprolog_owl_round_ornament.jpg?height=460&width=460&qv=90>

<http://dev.stephendiehl.com/hask/img/haskell_logo.svg>

<http://www.swi-prolog.org/pldoc/man?section=flags>

<http://www.swi-prolog.org/pldoc/doc_for?object=root>