Cormac Duggan
Student ID: 17100348

In this paper I will explain the design choices and analyze the differences between the Functional design paradigm and its implementation of a method to find the longest common prefix of a list of words, and the Logical design paradigm and its implementation of the same method to find the longest common prefix of a list of words.

Beginning with the Functional implementation I decided to use Haskell to create the program as it is what I am most familiar with. The basis for the program is a definition that takes in a set of strings and will recurse through them checking for the largest prefix iterating through the entire set. This is done using the foldl command which iterates through a list 2 items at a time (item 1 and 2 then item 2 and 3 then item 3 and 4 etc). These two items are sent to the second function which determines the longest prefix between the two then sends that prefix back to be checked against the next item in the list. Once the list has been checked in its entirety the original prefix function will return the longest prefix string.

In the Logical implementation of the prefix problem I chose to use prolog to create the same program to find the longest prefix of a given set of strings. The Logical version of this program uses the same essential system to find the longest common prefix as the Functional system but is implemented using predicate logic instead of functional. In this version the prefix function takes in a list of strings along with a variable to contain the common prefix of the two. It then takes the first two items of the list and finds the longest common substring and recurs for every item after the first item in the list. Each time the function recurs the Ans variable is set to the possible substrings available between the two words that were checked. Once the last word is checked the recursion returns up all the values that were saved in the Ans variable. If there is a substring that matches all the way up the recursion it will return true for the prefix predicate and will print the Ans variable for the prefix that was identified in the list of strings. The very first line in the prefix.pl file sets a prolog flag that tells it to compare anything in double quotes as characters instead of as atoms. The code I wrote originally made sense to me but didnt work so I looked on the SWI-Prolog docs in the Environmental Control section and found that this would properly compare characters of the strings in the prefix1 predicate.

The essential difference between the two implementations is the Functional paradigm system of using functions versus the Logical paradigm system of using predicates and predicate logic. In the Functional system an input data type is given to a function which may or may not call upon other functions, but the entirety of the code is self contained inside of functions, and the returned value is also a data type. In the Logical approach the prefix predicate is passed a list of strings and an empty variable. The predicate will return either a true or false value and what values can work for the provided empty variable in the predicate call. What determines the value in the variable and if the predicate returns true or false can be assigned to multiple different things, in this case a separate predicate and a recursive form of the first predicate are used to determine what values can be used to return true for the prefix predicate. Simply put, everything is reliant on parts of the input being or not being equal to each other. The equality or inequality is what determines the valid or lack of valid answers in the end.

Personally I would prefer to write code using the Functional paradigm over the Logical paradigm any day. Predicate logic seems very hard to wrap my head around on

anything more than a small scale application. The functionality of the two and the efficiency of both implementations seem to be pretty close to the same so I would say it really comes down to preference and what makes sense to the person that is programming when choosing between the two.