A Natural Language Interface for MS-DOS using SWI-Prolog

Aaron Edwards, P12195501,
MSc Intelligent Systems with Robotics, De Montfort University

***Abstract*—Natural Language Processing (NLP) is the means by which machines can be made to interpret natural human language, for instance the English language. This report provides documentation for a project that was conducted to use SWI-Prolog to create an interface for MS-DOS, so that various simple operations can be carried out through the use of natural English language. The relevant literature is consulted in order to help form a better design, this design is then implemented and tested. An experiment is then carried out, to further assess the capabilities of the Prolog implementation. This experiment involves comparing this implementation against a sister system, created in C#. These systems are compared and contrasted prior to a conclusion being drawn as to the success of the Prolog NLP Interface for MS-DOS, and this project as a whole.**

***Index Terms*—Natural Language Processing (NLP), SWI-Prolog, Logical Programming, Human Computer Interaction (HCI), MS-DOS, C#, Object Orientated Programming (OO)**

I. INTRODUCTION

This report documents a project to design and implement an interface for MS-DOS using Natural Language Processing (NLP). This interface must be able to recognise natural english language, so that the user can issue commands to perform several simple tasks in the windows file system. Over the course of this report, the relevant background knowledge will be reviewed, and guided by the relevant available literature. This will be used to help produce a proposed design for the NLP interface. An online survey will be carried out to help tailor the design to work with the most commonly used phrasing of the commands to be interpreted. The proposed design will be described in detail, followed by the methodologies used in its creation. Next the details and results of an experiment on the effectiveness of the proposed system will be provided, followed by a critical review of these results. Finally a conclusion of this project as a whole will be given, with a view to reviewing how successfully this project has been carried out, along with the details of any possible improvements that could be made.

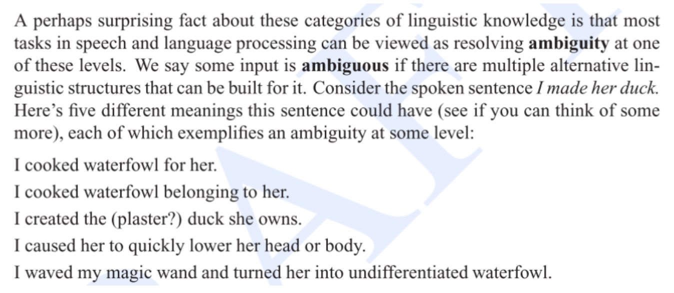
II. BACKGROUND

A. Prolog

Prolog is a programming language which falls under the “Declarative” category; specifically Prolog is a Logical programming language. This means that, as a language, Prolog is very good at logical inference and knowledge representation [1]. For these reasons Prolog is often used in situations where the encoding, storing and decoding of specific knowledge is important, such as Expert Systems [2], [3], [4]. While the studies of NLP and Expert Systems do not often intersect, Prolog’s ability to implicitly form and update a knowledge base, without forcing the programmer to specify what inferences should be gleaned, means that Prolog would be equally as capable of gleaning this knowledge directly from a user. This is one of the reasons why Prolog has always been a popular choice of technologies for use in Human Computer Interaction (HCI) [5], [6], [4], [2]. The combination of these strengths mean that Prolog is arguably the perfect language for use in NLP; the ability to model knowledge with facts, update that fact base by use of it’s rule base and Depth First Search technique, and the ability to provide a good HCI component makes a very good base for NLP applications, such as that to be solved as part of this project.

B. Natural Language Processing (NLP)

Natural Language Processing (NLP) is the science of creating technology that is able to interpret natural human language, and communicate with users in a way that is very intuitive for users [7], [8]. One of the main difficulties of this field (and of all aspect of Human Computer Interaction, HCI) is that humans are very subjective in how they communicate and perceive data [9]. The example provided in Figure 1 shows how even a simple sentence can have many different meanings, all based on context.


A perhaps surprising fact about these categories of linguistic knowledge is that most tasks in speech and language processing can be viewed as resolving **ambiguity** at one of these levels. We say some input is **ambiguous** if there are multiple alternative linguistic structures that can be built for it. Consider the spoken sentence *I made her duck.* Here's five different meanings this sentence could have (see if you can think of some more), each of which exemplifies an ambiguity at some level:

I cooked waterfowl for her.
I cooked waterfowl belonging to her.
I created the (plaster?) duck she owns.
I caused her to quickly lower her head or body.
I waved my magic wand and turned her into undifferentiated waterfowl.

Fig. 1. Ambiguity of Natural Language, [9]

NLP and contextual clues is one area in which most common Internet “chat-bots” fall short; they can be easily confused due to their inability to grasp contextual information. This often results in a very frustrating user experience, which is a polar opposite result to what NLP systems are designed to achieve. More recent works have been done to allow NLP solutions to utilise contextual models to help solve the ambiguity of natural human communication [8], [9]. This helps to give the NLP system a “strong sense of utilitarian, purpose driven conversation”[8]. For the purposes of this project, contextual information is fairly minimal; however it can be incorporated in order to give the user a more fluid experience. For example in the command “**Rename example1.txt to example2.txt, and then copy it to Desktop**” contextual information tells us that ‘it’ is likely to be the same file that we first must rename. There are a number of ways in which this particular case can be handled in order to provide the user with the best experience; firstly the command could be split into two separate commands when the system parses the word “then”. This would provide a very intuitive command system, as certain words in the English language are often used to describe an order of sequencing. However in some circumstances this may cause misinterpretations between the user and the NLP system. Secondly, in order to give the system a greater level of security, the file could be renamed as expected then have the user prompted for confirmation of copying the newly named ‘example2.txt’ to the directory ‘Desktop’. This reduces risk of misinterpretation, but also makes the system less user friendly and may cause users to feel as though they can’t trust the system to understand them, if it keeps having to confirm their commands.

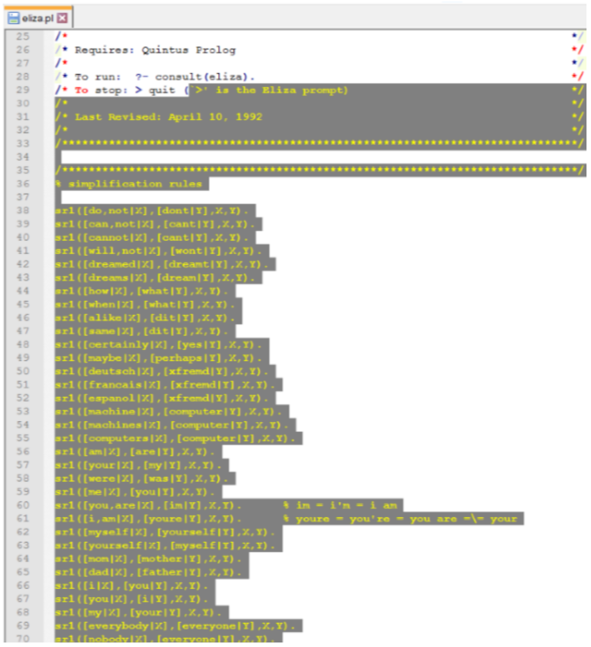
Another important aspect of NLP is spelling; any system that hopes to intuitively cater for human interaction has to model for the element of human error. In a text based NLP system, this means spelling errors. Spelling errors can be particularly difficult to deal with dynamically, as they can range from having letters in a word in the wrong order (e.g. “wrod” instead of “word”), alternatively at the other end of the spectrum there are cases when a user does not actually know how to spell a word, and is so far from the correct spelling that even sophisticated spell checking software cannot determine what the user is trying to say. There are several third party libraries available that are designed purely to aid in spell checking and correction [10], [11]. The use of these third party libraries was considered for use in this project, however these are both fairly heavy libraries which would reduce the efficiency of the resultant program significantly. This is because both of these libraries are designed for use with much larger software than will be produced for this project, and hence both of these libraries deal with a huge number of cases of misspelling, bad grammar, etc.

Another alternative to detecting spelling errors is an algorithm called the Levenshtein Distance [12]. This algorithm detects probable spelling errors by comparing two words, character by character. For every character that is not the same in each string, the Levenshtein Distance is incremented. The final output of this algorithm can be considered as a ”String Similarity” measure. This method is equally able to deal with words of differing lengths and different spellings, and the fact that the final output is a simple number means that this can simply be thresholded against a pre-defined ”limit of dissimilarity”, in which case it can be assumed that one word is not simply the other including a spelling mistake / typo. The Levenshtein Distance algorithm is fairly easy to implement in Prolog, due to the fact that the algorithm basically consists of iterating through a series of characters, comparing equality and simple arithmetic incrementation. Using Prolog’s inbuilt aptitude for Recursion and Backtracking, an implementation for this algorithm could be a very elegant and lightweight solution to the problem of Spelling Error Detection.

C. Eliza

One NLP application that has been used as a case study for this project is the Eliza program [13]. Eliza is an internet style chat-bot that was written using the Prolog programming language. One of the main techniques Eliza uses to give the impression of truly “understanding” what the user says to it is by swapping what Eliza considers to be equivalent words (see Figure 2). This reduces the number of possible input phrases that Eliza has to specifically be able to deal with. For example, Eliza considers “machine” and “computer” to mean exactly the same thing, contractions like “can’t” are also considered equal to their non-shortened versions (“cannot”,“can not”) [13]. While exchanging contracted words with their non-abbreviated equivalents (or visa versa) will not have any effect on the sentence, some words should not be swapped for one another even if they have very similar meanings. The key example of this in Eliza is the interchangeability of the words “machine” and “computer” [13]. There are relatively few circumstances where an equivalence between these words will preserve the user’s initial intent.

Another area in which Eliza could be improved is in its ability to capture context. The lack of an implicit ‘memory’ about the subject being discussed with the user leads to each exchange with Eliza being treated as its own interaction, instead of a larger conversation. The ability to capture context in natural language is very difficult, as previously discussed, however Prolog’s innate ability to update its knowledge base with inferred knowledge would make this task slightly easier.



```
/*                                                                          */
/* Requires: Quintus Prolog                                                 */
/*                                                                          */
/* To run:  ?- consult(eliza).                                              */
/* To stop: > quit (`>' is the Eliza prompt)                                */
/*                                                                          */
/* Last Revised: April 10, 1992                                             */
/*                                                                          */
/****************************************************************************/

/****************************************************************************/
% simplification rules

sr1([do,not|X],[dont|Y],X,Y).
sr1([can,not|X],[cant|Y],X,Y).
sr1([cannot|X],[cant|Y],X,Y).
sr1([will,not|X],[wont|Y],X,Y).
sr1([dreamed|X],[dreamt|Y],X,Y).
sr1([dreams|X],[dream|Y],X,Y).
sr1([how|X],[what|Y],X,Y).
sr1([when|X],[what|Y],X,Y).
sr1([alike|X],[dit|Y],X,Y).
sr1([same|X],[dit|Y],X,Y).
sr1([certainly|X],[yes|Y],X,Y).
sr1([maybe|X],[perhaps|Y],X,Y).
sr1([deutsch|X],[xfremd|Y],X,Y).
sr1([francais|X],[xfremd|Y],X,Y).
sr1([espanol|X],[xfremd|Y],X,Y).
sr1([machine|X],[computer|Y],X,Y).
sr1([machines|X],[computer|Y],X,Y).
sr1([computers|X],[computer|Y],X,Y).
sr1([am|X],[are|Y],X,Y).
sr1([your|X],[my|Y],X,Y).
sr1([were|X],[was|Y],X,Y).
sr1([me|X],[you|Y],X,Y).
sr1([you,are|X],[im|Y],X,Y).      % im = i'm = i am
sr1([i,am|X],[youre|Y],X,Y).      % youre = you're = you are =\= your
sr1([myself|X],[yourself|Y],X,Y).
sr1([yourself|X],[myself|Y],X,Y).
sr1([mom|X],[mother|Y],X,Y).
sr1([dad|X],[father|Y],X,Y).
sr1([i|X],[you|Y],X,Y).
sr1([you|X],[i|Y],X,Y).
sr1([my|X],[your|Y],X,Y).
sr1([everybody|X],[everyone|Y],X,Y).
sr1([nobody|X],[everyone|Y],X,Y).
```

Fig. 2. Eliza's word equality engine, [13]

D. Survey

This survey will be ensure the anonymity of its participants, for the purposes of preserving their privacy and to ensure that no bias can be introduced into the results. The survey will be circulated through social media, using my own personal Facebook page.

A survey was conducted in order to get a good idea of how people would phrase commands given to a computer in natural English. The survey can be found at "**http://www.smartsurvey.co.uk/s/2HMZ0/**". The survey was published on the author's personal Facebook account, in order to easily get responses to these questions in a way that was not overly time intensive. Additionally, this distribution method helped to ensure that the results that were taken came from a good sample of the population, i.e. a mixture of people with differing levels of technical aptitude, age, gender, etc. This was done in an attempt to learn how the majority of people would communicate with computers, using natural English, without any bias that would be inherent from a person with programming or scripting experience.

III. Proposed Design

A. Design Overview

The first thing that was designed was the system architecture; both the problem to be solved and the way in which Prolog would naturally deal with it lend themselves to a Hierarchical Finite State Machine (HFSM). Due to the way in which Prolog works, each state can be considered to be one predicate (or in some cases a number of sub-predicates); the way in which predicates can call each other, and themselves recursively can be considered as the transitions between these states. Figure 3 shows how this system will be arranged, in terms of an HFSM.

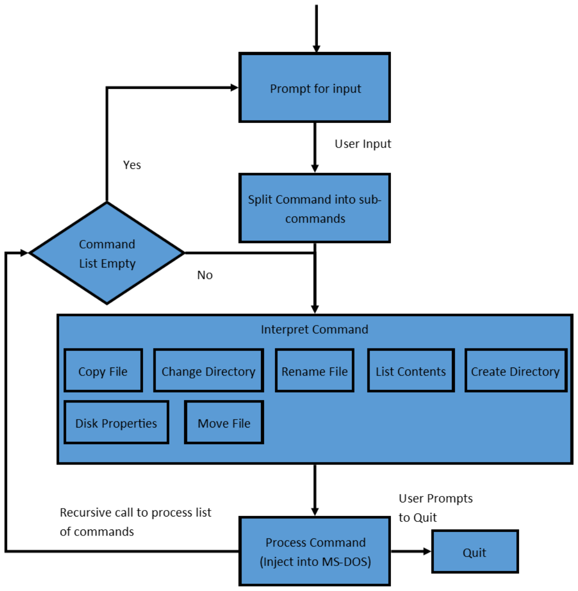


Fig. 3. Proposed System Design

The above Figure shows a Finite State Machine (FSM), which is a common control architecture for simple programmes. These allow discrete behaviours or actions to be enacted and transitioned between, sequentially [14]. While a logical programming language like Prolog does not explicitly support the implementation of FSM's, if the concept of a Predicate (which builds up the core of Prolog) is compared to the concept of a State, we find that they are fairly similar. This is one of the guiding principles that was used during the design stage of this project. This has helped to both formulate an overall design, and to reduce code redundancy and improve the readability of the produced code.

Firstly, when the program starts, a menu is provided to the user; this can be considered the first State within the FSM. This menu serves two purposes; firstly it will introduce the program to the user, so the user knows what is expected of them. Secondly it will let the user know that the program has started up successfully and is ready for their input.

The **et.pl** script is used to help structure the user's text as a list of Atoms, this list is then simplified, again using the predicates built into the **et.pl** script. This means that words such as "a", "the", etc. which are considered Stop Words, are removed. A list of Stop Words was downloaded for use in this project, although this list was modified to some extent to work better with this project's specific syntax and the kind of language that is expected from the user [15], [16]. The Stop Words formed the large majority of the **dictionary.pl** file, along with some conversions between words that can be considered equivalent. This simplified list of Atoms can now be considered a command.

The Levenshtein Distance algorithm is used to help detect probable spelling errors. Once this algorithm has detected an error, the system adds this spelling mistake to its knowledge base, so that these mistakes can be easily corrected in the next step of the program. This also helps the system to learn common spelling mistakes, so that the system has a better chance of correctly interpreting the user in the long term. The means by which spelling errors are corrected is the same as the method by which the user's input is "simplified" into a command.

This command is then parsed for the word "then", which implies multiple commands are linked together in a specific order. It should be noted that the simplification process converts equivalent words, such as "next" into "then", in order to make this stage more straight forward. Splitting the commands like this is one of the key methods by which this system attempts to capture contextual information from the user, and hence provide a better experience. Upon finding "then", the program will split the original phrase into a series of sub-commands. This is done through manipulating a list of lists, and through use of the append function (see the parse_command predicate in **NLP.pl**). Figure 4 shows both the inputs and the results of the parse_command predicate.

$$Input : [complete, command, A, then, command, B]$$

$$Result : [[complete, command, A], [command, B]]$$

Fig. 4. parse_command example

Once the system has generated this second order list of commands (list of lists) to complete it will iterate over this list, using recursion, processing each command one at a time until either the command "quit" is found, or the list of commands becomes empty. Once the command list is emptied, the program will wait for user input again, without another prompt. This has been omitted in order to avoid making the user feel pestered, which can be a common side effect of interacting with other chat bot style programmes.

B. Methodology

The first step in any project should always be to clarify the objectives and requirements of the system to be produced. To this end, the first task that was undertaken (after reading through the project specification document that was provided) was to draw up an online survey (see Appendix 7). This was a simple questionnaire designed to help highlight what kind of syntax and language people use when giving a computer commands, using natural language. This helped to refine the kind of words the program should search for, and what words can therefore be considered important, equivalent or irrelevant (Stop Words).

Following this process, the literature review was carried out, in order to learn from other researchers' experience and help ensure that a high quality program is produced. This process helped to refine the proposed design into something that works conceptually and is in line with the findings that other researchers have presented.

The implementation stage of this project largely went smoothly, although there was an issue with using the append function (as provided in **msdos.pl**). However that issue turned out to be cause by a misunderstanding of how this function worked; this function has an Arity of three (it has three parameters, denoted as append/3 in this case), the two lists to append and a final storage variable. Initially the storage variable and one of the lists to append were the same variable, in an attempt to re-use the same variable to reduce code complexity. As it turns out, this is an illegal call in SWI-Prolog. Once this became clear, the problem was rectified and the append function was used properly. One negative side effect of using append however, is that at it's conclusion the resultant list is backwards. This was easily solved with use of the **reverse/3** predicate.

During the implementation phase of this project, several changes were made to the code that was originally provided. One such change is the re-factoring of how the "translation" process works; originally the pre-processed input from the user was run through a series of rules, where all actions the program could perform were part of the same rule base. This was changed so that the proper command word (i.e. "copy" etc.) is detected and used to funnel the programme into the correct set of rules, specifically designed for rules of that type. This was done because, after some experimentation it was found that certain commands require a certain phrasing or syntax that is not very transferable to the other commands supported by this system. In order to ensure that the commands are properly interpreted, we must first ascertain which command we are dealing with. This method helps to reduce the frequency of missed variables or miscommunications between the user and the system.

One issue that is still present is that the **msdos.pl** pass_to_os function does not seem to support multiple commands being issued sequentially. This results in secondary commands (commands supplied by the user following the word "then" or equivalent) not appearing in the command console, until the first console window is closed by the user. Any secondary commands are completed as expected, again following the same pattern that subsequent commands are only processed after the preceding command window is closed. Having thoroughly debugged this issue using the in-built **guitracer** supplied with SWI-Prolog [17], it can be verified that this problem stems from within the **win_exec/2** predicate. Upon finding that this issue was caused by the in-built predicate, the first thing that was done was some research into how this function works, and whether there are any alternatives that could be used more successfully. The documentation on the SWI-Prolog website does not go into details of how **win_exec/2** works, and there are no alternative methods that can be used in its place for this project [18]. As a result of this, the issue of not being able to inject multiple sequential commands will simply be regarded as an area of future improvement, to be investigated further at a later date.

IV. Testing and Experimentation

A. Use Case Testing

Once the Prolog NLP system was completed, the system went through a fairly vigorous testing cycle. This was done through a series of use case tests (see Appendix B), inspired by the results of the survey that was created. These test cases can be seen as documentation of the fairly thorough testing cycle that has been carried out on the completed NLP program, written in SWI-Prolog. Not every possible permutation of inputs that the system is able to deal with has been included in the test log, primarily due to an effort to help simplify this report. However the testing was carried out with Equivalence Partitioning and Boundary Value Analysis in mind [19], [20]. These testing methodologies state that the range of inputs can be split into categories and subcategories, and that each partition has equal weighting. i.e. All inputs that use the word "display" can be considered one such category. If one test that asserts a certain aspect of this category passes, the functionality for this part of the program must be working. The benefit of utilising these test methodologies is that it reduces the complexity of the testing cycle, while maintaining robustness.

As shown in Appendix B, all tests passed with the exception of the ability to supply sequential commands. This functionality works, as the system is able to populate the necessary list structures and then recursively send commands. However as previously stated, once these commands reach the **win_exec/2** function, the functionality breaks. This testing cycle has verified that the Prolog NLP program that has been created is able to cater for a wide range of user inputs, and can handle unknown commands sufficiently. It is able to allow for simple spelling mistakes, and is able to "learn" these mistakes, so that user inputs may have a long term effect on the system, helping it to further improve.

B. Experiment Details

In this section of the report, an experiment is to be devised to help assess the capabilities of the system that has been created. This will primarily be done by attempting to create an identical (as far as possible) program using C#. This language was chosen due to the facts that, at the time of creation the author was most competent with C++ and similar Object Orientated languages (like C#), C# also makes use of the .NET framework. This is significant because .NET allows for more complex String manipulation and evaluation, which will ensure fewer implementation issues arise from attempting to re-create this program. C# also allows for aesthetically appealing and intuitive User Interfaces to be created very quickly. This is a very important aspect of any system designed to improve HCI.

It should be noted, that as the purpose of this experiment is to investigate the ability of another language (C#) to solve NLP problems, the MS-DOS that is created will not be injected into the console, but simply printed verbatim onto the application's console window. This decision has been made to help reduce development time, and to ensure the focus for this experiment remains on the NLP aspects of this assignment.

C. Experiment Results

During the implementation of the C# NLP, several things became apparent; firstly the ability to structure a program in an OO fashion (i.e. into classes and functions), makes the code far more readable, it also makes maintaining the use of meaningful variable / function names far easier.

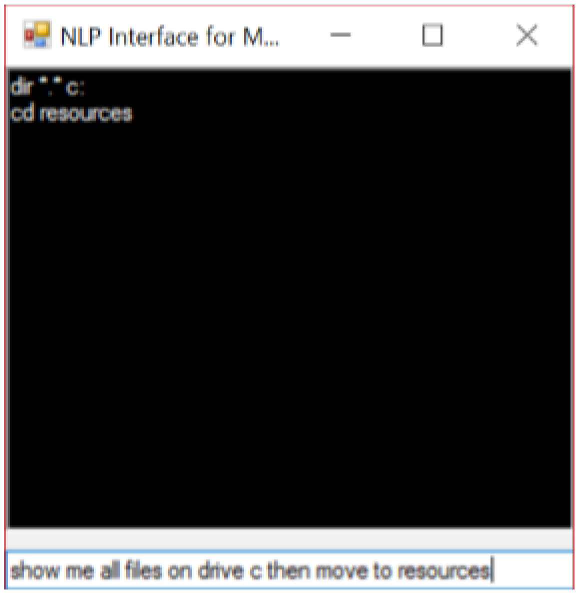
Secondly Prolog's lack of support for changing the value of a variable makes programming in Prolog far more convoluted, as once a variable is instantiated, it is difficult to alter that value. This leads to difficult to read code and often more complex solutions than are strictly necessary.

However Prolog's Depth First Search and Backtracking capabilities can provide very elegant solutions to problems that in C#, required the use of multi-dimensional iterative loops and multiple safety checks. Prolog was also far more effective at handling variables inside of its list of Atoms; in the "translation" section of each program Prolog was simply able to include a variable inside a list, and have it instantiated at run time (see Figure 5). C# however does not handle variables in the same way, as the list itself is a variable. This means that the list containing the command words must be iterated over, translating what words the system can interpret, and assuming those it does not are variables to be injected into MS-DOS. This is obviously a risky approach, as there is no way to validate this assumption, and it puts an amount of control in the users hands that is widely considered unwise.

$$translateMove([move, X], ['cmd\ /k\ cd\ ', X]).$$

Fig. 5. Example of Prolog Handling Variables inside a list of Atoms

The program that was created in C# does have one feature that could not be implemented in Prolog; the ability to give the system multiple commands in one phrase, and have them processed sequentially by the system, as shown in Figure 6. This feature allows for much more natural interaction with the system, as it enables the user to describe a series of actions to be taken, so that some higher goal can be achieved. For example if a user wanted to create a new directory and copy some files into it, the act of creating the directory is not their ultimate goal. This means that a system which requires both actions to be taken individually, with the same level of feedback being given for both actions, may seem cumbersome.



```
dir *.* c:
cd resources
```

```
show me all files on drive c then move to resources
```

Fig. 6. Example of Sequential Commands, in C#

D. Analysis

This section will provide a fairly high level comparison of the two programs that have been created for this project; the points discussed here will be fairly abstract, but logical extensions of the discussion exhibited in the previous sections of this report.

1) Benefits of Prolog:

- Variable Handling inside a list of atoms is easy and adaptable
- Backtracking & Depth First Search provides elegant solutions
- Recursive functions are less verbose and more elegant
- Prolog compiler makes recursion more efficient (by converting to implicit loops)
- Ability to edit knowledge base at run time gives scope for machine learning
- Rule base allows for greater scope to generalise commands

2) Drawbacks of Prolog:

- Program/Predicate structures make writing neat code more difficult
- Lack of support for changing variable values (easily) makes for messy code
- Lack of OO structure makes program design more conceptually difficult
- **win_exec/2** function unable to handle sequential commands

3) Benefits of C#:

- OO structure makes program design and structure very neat
- .NET framework makes string manipulation easy
- Able to implement sequential commands easily

4) Drawbacks of C#:

- Variable handling inside a list object is difficult and leads to insecurities
- Program code is more verbose
- Knowledge base is not handled very neatly (for example stop words are implemented as a list of strings that is populated at Parser object creation)
- No scope for machine learning, without a prohibitively large overhead for development time

It is difficult to say, were I given this project again in industry, whether Prolog or C# would be more likely to be used. This is primarily because the two languages are so different. The pro and con lists previously shown highlight these differences and help to give an idea where each language might be more useful, compared to the other. Two additional factors that were not mentioned in these lists is that C# is more widely known as a language, and also has more support for third party libraries. This would make using C# the more logical choice for use in industry, as although it lacks some of the elegance of Prolog, the language itself is more likely to be known within the team / company I would be working within. However, if the impact of introducing a potentially known technology into the work environment were to be removed, Prolog would be the language of choice. This is because it was found to be the more elegant solution, along with the other benefits previously mentioned.

V. Conclusion

In conclusion, the implementation for both the Prolog and C# variants of the proposed NLP system can be considered largely successful, but perhaps in different ways. The Prolog implementation is more robust and includes the scope for the system to "learn" user's spelling mistakes. The main failing of this system is its inability to deal with sequential commands. However that cannot necessarily be a failing of this project specifically, as this feature may have been considered out of scope when **win_exec/2** was created for SWI-Prolog. The C# implementation however does not handle variables such as file/directory names as well, but it does allow for sequential commands to be given, and the code itself is far more scalable and maintainable, thanks to the OO program structure.

The Prolog program can be considered more robust as its rule base allows for a greater amount of variation in the commands given to it, compared with the C# variant. Were this project to be conducted again, the implementation of a more advanced spelling correction system would also be investigated. The Levenshtein distance algorithm worked well to detect simple spelling mistakes / typos, however there are a relatively wide range of circumstances where this system could be "fooled" into replacing what should be considered a Stop Word, with a Command Word, and thus causing confusion in the translation stage of the program. This fault was evident in both the Prolog and C# variants of the program, and therefore was more likely to be a flaw in the algorithm, rather than a language specific fault or programming error.

The creation of the C# program helped to clarify how powerful Prolog can be, thanks to a few of the more powerful aspects of the language; namely its use of recursion, backtracking and the knowledge base represented by both facts and rules. However the C# implementation also helped to emphasise how important things like program structure, meaningful variable names and other things that can be considered "best programming practice" can aid in the production of a successful system. If Prolog were able to support more refined programming practices, it would likely be one of the most widely used languages for any sort of Expert or Knowledge Based system.

References

[1] "SWI-Prolog Manual - http://www.swi-prolog.org/pldoc/doc_for?object=manual," 2016.

[2] A. Nega, "Localized hybrid reasoning system for TB disease diagnosis," in *AFRICON 2015*, pp. 1–5, IEEE, sep 2015.

[3] C.-C. Yang, J.-D. Lin, M.-C. Ho, and Y.-C. Zhan, "The research on developing the expert system of road pavement maintaining knowledge condensed revised version," in *2016 Eighth International Conference on Advanced Computational Intelligence (ICACI)*, pp. 432–436, IEEE, feb 2016.

[4] Yangdong Zhang, Zhen Fan, Zenguan Zhang, and Senlin Zhang, "Development of the Silk Relics Expert System based on SWI-Prolog," in *2015 Chinese Automation Congress (CAC)*, pp. 831–834, IEEE, nov 2015.

[5] F. Siasar djahantighi, M. Norouzifard, S. Davarpanah, and M. Shenassa, "Using natural language processing in order to create SQL queries," in *2008 International Conference on Computer and Communication Engineering*, pp. 600–604, IEEE, may 2008.

[6] J. Wielemaker, M. Hildebrand, and J. Van Ossenbruggen, "Using Prolog as the fundament for applications on the semantic web," 2007.

[7] A. Copestack, "Natural Language Processing," *Natural Language Processing*, pp. 2003–2004, 2004.

[8] V. Hung, "Context and NLP," in *Context in Computing*, pp. 143–154, New York, NY: Springer New York, 2014.

[9] D. S. Jurafsky and J. H. Martin, "Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition," *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*, 2000.

[10] "APE Library - https://github.com/Attempto/APE/blob/master/lexicon/spellcheck.pl."

[11] "string-similarity - https://www.npmjs.com/package/string-similarity."

[12] "Levenshtein Distance - http://people.cs.pitt.edu/~kirk/cs1501/Pruhs/Fall2006/Assignments/editdistance/Levenshtein%20Distance.htm."

[13] V. Patel and D. Elizondo, "Eliza."

[14] "Finite State Machines - https://embedded.eecs.berkeley.edu/research/hsc/class/ee249/lectures/l4-FSM-CFSM.pdf."

[15] "List of English Stop Words - XPO6 — XPO6 - http://xpo6.com/list-of-english-stop-words/."

[16] "Dropping common terms: stop words - http://nlp.stanford.edu/IR-book/html/htmledition/dropping-common-terms-stop-words-1.html."

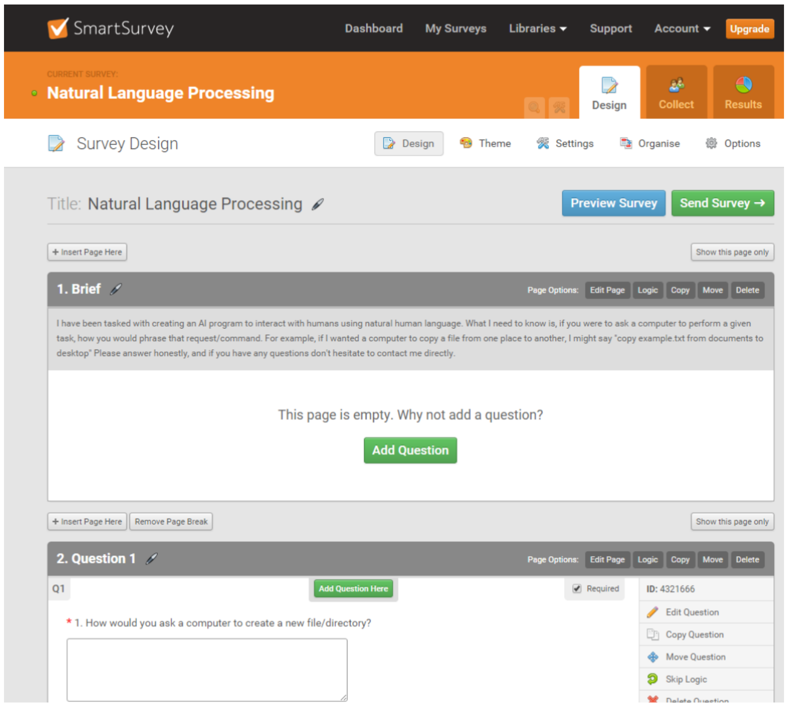
[17] "SWI-Prolog – guitracer/0 - http://www.swi-prolog.org/pldoc/man?predicate=guitracer/0."

[18] "win_exec/2 - http://www.swi-prolog.org/pldoc/doc_for?object=win_exec/2."

[19] S. Reid, "An empirical analysis of equivalence partitioning, boundary value analysis and random testing," in *Proceedings Fourth International Software Metrics Symposium*, pp. 64–73, IEEE Comput. Soc.

[20] "What is Boundary value analysis and Equivalence partitioning? - http://www.softwaretestinghelp.com/what-is-boundary-value-analysis-and-equivalence-partitioning/."

Appendix A
Survey



Title: Natural Language Processing

1. Brief

I have been tasked with creating an AI program to interact with humans using natural human language. What I need to know is, if you were to ask a computer to perform a given task, how you would phrase that request/command. For example, if I wanted a computer to copy a file from one place to another, I might say "copy example.txt from documents to desktop" Please answer honestly, and if you have any questions don't hesitate to contact me directly.

This page is empty. Why not add a question?

2. Question 1

* 1. How would you ask a computer to create a new file/directory?

Fig. 7. Screenshot of online survey that was created

Appendix B
Test Cases

Test Case	**Description**	**Input**	**Expected Result**	**Pass / Fail**
		Simple System Tests		
1	Displaying Directory/Drive Contents	*“show me the contents of this folder”*	All files/directories in this location shown	PASS
2	Displaying Directory/Drive Contents	*“display everything in this directory”*	All files/directories in this location shown	PASS
3	Displaying Directory/Drive Contents	*“list all* [Supported File Type] *files in this directory”*	All files of specified type in this location shown	PASS
4	Navigating File System	*“go to* [Directory] ”	Directory location changes to correct destination	PASS
5	Navigating File System	*“move up one layer”*	Directory location changes to correct destination	PASS
6	Navigating File System	*“change location to drive* [Drive] ”	Directory location changes to correct destination	PASS
7	Copying Files	*“copy all* [File Type] *files to* [Directory] ”	All files of specified type are copied to destination	PASS
8	Copying Files	*“copy* [File Name] *to drive* [Drive]”	Target file copied to destination	PASS
9	Creating Directories	*“create a new directory called* [Directory Name]”	Directory created with correct location and name	PASS
10	Transferring/Moving Files	*“move all* [File Type] *files to* [Directory Name]”	All files of specified type are moved to destination	PASS
11	Transferring/Moving Files	*“transfer* [File Name] *to* [Directory Name]”	Target file moved to destination	PASS
12	Type (shows contents/properties of file/directory/drive)	*“open* [File Name]”	Contents of file shown	PASS
13	Type (shows contents/properties of file/directory/drive)	*“type* [Drive Name]”	Contents/Properties of Drive shown	PASS
14	Display Properties of a Drive	*“what are the properties of* [Drive Name]”	Label/Serial Number shown	PASS
15	Display Properties of a Drive	*“what are the details of drive* [Drive Name]”	Label/Serial Number shown	PASS
16	Rename a file/directory	*“change the name of* [File/Directory Name] *to* [Name]”	Name of target is changed	PASS
17	Rename a file/directory	*“take* [File/Directory Name] *and set its name to* [Name]”	Name of target is changed	PASS
18	Unknown Command	*“utter gibberish”*	user prompted to rephrase command	PASS
		Spelling Correction Tests		
1	Missing One Letter	copy , *“cop”*	copy	PASS
2	Missing Three Letters	copy , *“c”*	No change made	PASS
3	One Extra Letters	copy , *“copyy”*	copy	PASS
4	Three Extra Letters	copy , *“copyyyy”*	No change made	PASS
5	One Incorrect Letters	copy , *“kopy”*	copy	PASS
6	Three Incorrect Letters	copy , *“kipe”*	No change made	PASS
		Sequential Commands Tests		
1	Move to a directory, then display contents	*“move to* [Directory Name] *then show me its contents”*	Contents of destination are shown	FAIL

Appendix C
Prolog Code

A. *NLP.pl*

```
%Self Starting Program, First thing to be done is consult the various supporting Prolog Scripts
:- consult(et),consult(msdos),consult(dictionary),consult(spellchecker),consult(levenshtein).

intro:-
write('Welcome! I am a program designed to help you navigate your Windows File System.'),nl,
write('Firstly if you want to quit, simply tell me so!'),nl,
write('I can be used to View Files inside a given Directory/Folder,'),nl,
write('I can Copy Files, Rename Files, Create new Folders/Directories, Transfer files to different
locations, show you the contents of files, I can give you the Properties of a given storage device,
and Move between Directories/Folders.'),nl,
take_input,nl.

%use et to take input from user, tokenise it into a list of words
take_input:-
tokenize_line(user,Commands),
tokens_words(Commands,Atom), %produce list of commands
simplify(Atom,SimplifiedAtoms), %remove redundant words, pass to dictionary.pl for simlification
spellChecker(SimplifiedAtoms), %run levenshtein spell checking, update knowledge base with results
correctSpelling(SimplifiedAtoms,CorrectedAtoms),
parse_command(CorrectedAtoms,[[]]). %parse command(s) into a list of instructions

%recursive stop condition
parse_command([],[]).

%list to parse is empty, but commands can be processed
parse_command([],CommandList):-
reverse(CommandList,[],NewCommandList),
translate_commands(NewCommandList).

%head of list is the word we are using as a delimiter
parse_command([then|Tail],[H|Rest]):-
parse_command(Tail,[[],H|Rest]).

%if head of list is not delimiter
parse_command([X|Tail],[H|Rest]):-
append(H,[X],NewCommandList),
parse_command(Tail,[NewCommandList|Rest]).

translate_commands([[]]). %recursive stop condition

translate_commands([H|Tail]):-
translate(H,_),!, %pass to msdos.pl for translation into msdos and sending the command to the OS
translate_commands(Tail).

%self starter
%:- intro,nl.
```

B. *msdos.pl*

```
translateMove([move,X],['cmd /k cd ',X]). %move directory
translateMove([move,up,one,directory],['cmd /k cd ../']). %move one directory up
translateMove([move,up,one,layer],['cmd /k cd ../']). %move one directory up
translateMove([move,back],['cmd /k cd ../']). %move one directory up
translateMove([move,drive,X],['cmd /k cd ',X,':']). %move to different drive

translateShow([show,drive,X],['cmd /k dir ',X,':']). %show all files
translateShow([show],['cmd /k dir']). %show current directory
translateShow([show,all,files,drive,Y],['cmd /k dir ',Y,':*.*']). %show all files on one drive
translateShow([show,all,X,files,drive,Y],['cmd /k dir ',Y,':*.',X]). %show certain type of file
translateShow([show,X,files,drive,Y],['cmd /k dir ',Y,':*.',X]). %show certain type of file
translateShow([show,all,files,directory,X],['cmd /k dir ','\\',X]).
translateShow([show,all,files,directory],['cmd /k dir ']).
translateShow([show,all,X,files,directory],['cmd /k dir *.',X]). %show certain type of file
translateShow([show,contents,directory],['cmd /k dir ']).
translateShow([show,all,contents,directory],['cmd /k dir ']).
```

```
translateCopy([copy,all,from,drive,X,drive,Y], ['cmd /k copy ',X,':*.* ',Y,':']). %copy all
translateCopy([copy,File,drive,X], ['cmd /k copy /-Y ',File,'.* ',X,':']).
translateCopy([copy,File,X], ['cmd /k copy /-Y ',File,'.* ',X]).
translateCopy([copy,all,files,directory,X], ['cmd /k copy /-Y ','*.* ',X]).
translateCopy([copy,all,files,X], ['cmd /k copy /-Y ','*.* ',X]).
translateCopy([copy,all,F,files,directory,X], ['cmd /k copy /-Y ','*.',F,' ',X]).
translateCopy([copy,all,F,files,X], ['cmd /k copy /-Y ','*.',F,' ',X]).
translateCopy([copy,all,files,drive,X], ['cmd /k copy /-Y ','*.* ',X,':']).

translateCreate([create,directory,X], ['cmd /k mkdir ',X]).
translateCreate([create,new,directory,X], ['cmd /k mkdir ',X]).
translateCreate([create,directory,X,drive,Y], ['cmd /k mkdir ',Y,':',X]).

translateTransfer([transfer,all,drive,X,drive,Y], ['cmd /k move ',X,':*.* ',Y,':']).
translateTransfer([transfer,File,drive,X], ['cmd /k move ',File,'.* ',X,':']).
translateTransfer([transfer,File,X], ['cmd /k move ',File,'.* ',X]).
translateTransfer([transfer,all,files,directory,X], ['cmd /k move ','*.* ',X]).
translateTransfer([transfer,all,files,drive,X], ['cmd /k move ','*.* ',X,':']).
translateTransfer([transfer,all,F,files,X], ['cmd /k move ','*.',F,' ',X]).
translateTransfer([transfer,all,F,files,directory,X], ['cmd /k move ','*.',F,' ',X]).
translateTransfer([transfer,all,F,files,drive,X], ['cmd /k move ','*.',F,' ',X,':']).

translateType([type,X], ['cmd /k type ',X]).
translateType([type,file,X], ['cmd /k type ',X]).
translateType([type,directory,,X], ['cmd /k type ',X]).
translateType([type,drive,X], ['cmd /k type ',X,':']).
translateType([type,X], ['cmd /k type ',X,':']).

translateVol([properties,X], ['cmd /k vol ',X,':']).
translateVol([properties,drive,X], ['cmd /k vol ',X,':']).
translateVol([properties,drive], ['cmd /k vol C:']).
translateVol([properties], ['cmd /k vol C:']).

translateRename([rename,X,Y],['cmd /k rename ',X,' ',Y]).
translateRename([X,rename,Y],['cmd /k rename ',X,' ',Y]).

translate(Input,Result) :- %capture phrase and split into which command is being called
captureCommandWord(Input,Result),
pass_to_os(Result),
take_input.

translate(_,[]) :-
write('I do not understand, try re-phrasing the command'), nl,
take_input.

%TYPE
captureCommandWord(Input,Result):-
member(type,Input),
translateType(Input,Result).

%PORPERTIES
captureCommandWord(Input,Result):-
member(properties,Input),
translateVol(Input,Result).

   %CREATE
captureCommandWord(Input,Result):-
member(create,Input),
translateCreate(Input,Result).

%TRANSFER (MOVE FILES)
captureCommandWord(Input,Result):-
member(transfer,Input),
translateTransfer(Input,Result).

   %COPY FILES
captureCommandWord(Input,Result):-
member(copy,Input),
translateCopy(Input,Result).

%QUIT
captureCommandWord(Input,_):-
member(quit,Input),
write('Quitting, goodbye'),
%relative file path will only work with full prolog installed, portable version will not find this file
%write to kb file, so spelling mistakes can be picked up again if needed
tell('./spellChecker.pl'),
listing(sp),
told.

%MOVE DIRECTORY (CD)
captureCommandWord(Input,Result):-
member(move,Input),
translateMove(Input,Result).

%SHOW (DIR)
captureCommandWord(Input,Result):-
member(show,Input),
translateShow(Input,Result).

%RENAME
captureCommandWord(Input,Result):-
member(rename,Input),
translateRename(Input,Result).

captureCommandWord([],_):-
write('No Commands have been given to me'),nl,
take_input. %commands passed are an empty list


process_commands :-
   repeat,
      write('Command -> '),
      tokenize_line(user,X),
      tokens_words(X,What),
      simplify(What,SimplifiedWords),
      translate(SimplifiedWords,Command),
      pass_to_os(Command),
      Command == [quit],
   !.

pass_to_os([])     :- !.

pass_to_os([quit]) :- !.

pass_to_os(Command) :-
   concat(Command,String),
   win_exec(String,show).


concat([H|T],Result) :-
   name(H,Hstring),
   concat(T,Tstring),
   append(Hstring,Tstring,Result).

concat([],[]).

append([H|T],L,[H|Rest]) :-
append(T,L,Rest).
append([],L,L).

%stop condition
output_commands([]).

%recursively output all commands in list
output_commands([Head|Tail]):-
write(Head),nl,
output_commands(Tail).

%recursive stop check, when empty list AND accumulator and result are the same
reverse([],Acc,Acc).

%take a list, add its head as the head of the accumulator and recurse
reverse([H|T],Acc,Result):-
reverse(T,[H|Acc],Result).
```

C. *spellChecker.pl*

```
:-dynamic sp/2.

correctSpelling(List,Result) :-
  sp(List,NewList),
  !,
  correctSpelling(NewList,Result).

correctSpelling([W|Words],[W|NewWords]) :-
  correctSpelling(Words,NewWords).

correctSpelling([],[]).

spellChecker([]).

spellChecker([Head|Tail]):-
spellCheck(Head),
spellChecker(Tail).

spellChecker([_|Tail]):-
spellChecker(Tail).

%---------PREDICATES TO CHECK FOR SPELLING
%---- ERRORS IN WORDS WE KNOW ABOUT------%
spellCheck(Atom):-
levenshtein(type,Atom).
spellCheck(Atom):-
levenshtein(properties,Atom).
spellCheck(Atom):-
levenshtein(create,Atom).
spellCheck(Atom):-
levenshtein(transfer,Atom).
spellCheck(Atom):-
levenshtein(copy,Atom).
spellCheck(Atom):-
levenshtein(quit,Atom).
spellCheck(Atom):-
levenshtein(show,Atom).
spellCheck(Atom):-
levenshtein(rename,Atom).
spellCheck(Atom):-
levenshtein(disk,Atom).
```

D. *levenshtein.pl*

```
%----LEVENSHTEIN DISTANCE SPELL CORRECTION----%
levenshtein(Word1,Word2):-
atom_chars(Word1,List1),
atom_chars(Word2,List2),
lev(List1,List2,0),
%add connection for this spelling mistake to kb
asserta(sp([Word2|X],[Word1|X])).

%lev does not unify,
%recurse using next word in phrase
levenshtein(Word1,[_|Phrase]):-
levenshtein(Word1,Phrase).

%no more words in phrase to check
levenshtein(_,[]).

%Both empty lists, recursive stop condition
lev([],[],Result):-
!,
Result < 3,
Result > 0.

lev([],Word2,Result):- %Empty list clause
length(Word2,L),
Diff is (Result + L),
lev([],[],Diff).

lev(Word1,[],Result):- %Empty list clause
length(Word1,L),
Diff is (Result + L),
lev([],[],Diff).

%Head of both lists are the same
lev([X|Tail1],[Y|Tail2],Result):-
X == Y,
lev(Tail1,Tail2,Result).

%Differing Heads of lists
lev([X|Tail1],[Y|Tail2],Result):-
X \== Y,
Diff is (Result + 1),
lev(Tail1,Tail2,Diff).
```

E. *dictionary.pl*

```
%ensure sr can be modified (added to)

simplify(List,Result) :-
  sr(List,NewList),
  !,
  simplify(NewList,Result).

simplify([W|Words],[W|NewWords]) :-
  simplify(Words,NewWords).

simplify([],[]).

%---------------Equivalent words---------------%
sr([display|X],[show|X]).
sr([list|X],[show|X]).
sr([describe|X],[properties|X]).
sr([details|X],[properties|X]).
sr([vol|X],[properties|X]).
sr([read|X],[type|X]).
sr([open|X],[type|X]).
sr([load|X],[type|X]).
sr([go|X],[move|X]).
sr([change,location|X],[move|X]).
sr([change,directory|X],[move|X]).
sr([move,all|X],[transfer,all|X]).
sr([relocate|X],[transfer|X]).
sr([replicate|X],[copy|X]).
sr([duplicate|X],[copy|X]).
sr([disk|X],[drive|X]).
sr([what,files|X],[files|X]).
sr([what|X],[files|X]).
sr([file|X],[files|X]).
sr([everything|X],[all|X]).
sr([folder|X],[directory|X]).
sr([make,new|X],[create|X]).
sr([every|X],[all|X]).
sr([next|X],[then|X]).
sr([before|X],[then|X]).
sr([name|T],[rename|T]).
sr([set,the,name,of,X,to,Y|T],[rename,X,Y|T]).
sr([change,the,name,of,X,to,Y|T],[rename,X,Y|T]).
sr([A,A|X],[A|X]). %remove duplicate words
sr([exit|X],[quit|X]).
sr([stop|X],[quit|X]).
%---------------file extensions---------------%
sr([text|X],[txt|X]).
sr([word,documents|X],[doc|X]).
sr([picture|X],[png|X]).
sr([latex,documents|X],[tex|X]).
sr([latex|X],[tex|X]).
%---------------redundant words---------------%
sr([the|X],X).
sr([is|X],X).
sr([are|X],X).
sr([there|X],X).
sr([any|X],X).
sr([please|X],X).
sr([a|X],X).
sr([able|X],X).
sr([about|X],X).
sr([across|X],X).
sr([after|X],X).
sr([almost|X],X).
sr([also|X],X).
sr([am|X],X).
sr([among|X],X).
sr([an|X],X).
sr([and|X],X).
sr([as|X],X).
sr([at|X],X).
sr([be|X],X).
sr([because|X],X).
sr([been|X],X).
sr([but|X],X).
sr([by|X],X).
sr([called|X],X).
sr([can|X],X).
sr([cannot|X],X).
sr([could|X],X).
sr([dear|X],X).
sr([did|X],X).
sr([do|X],X).
sr([does|X],X).
sr([either|X],X).
sr([else|X],X).
sr([ever|X],X).
sr([for|X],X).
sr([from|X],X).
sr([get|X],X).
sr([got|X],X).
sr([had|X],X).
sr([has|X],X).
sr([have|X],X).
sr([he|X],X).
sr([her|X],X).
sr([hers|X],X).
sr([him|X],X).
sr([his|X],X).
sr([how|X],X).
sr([however|X],X).
sr([i|X],X).
sr([if|X],X).
sr([in|X],X).
sr([into|X],X).
sr([it|X],X).
sr([its|X],X).
sr([just|X],X).
sr([least|X],X).
sr([let|X],X).
sr([like|X],X).
sr([likely|X],X).
sr([may|X],X).
sr([me|X],X).
sr([might|X],X).
sr([most|X],X).
sr([must|X],X).
sr([my|X],X).
sr([neither|X],X).
sr([no|X],X).
sr([nor|X],X).
sr([not|X],X).
sr([of|X],X).
sr([off|X],X).
sr([often|X],X).
sr([on|X],X).
sr([only|X],X).
sr([or|X],X).
sr([other|X],X).
sr([our|X],X).
sr([own|X],X).
sr([rather|X],X).
sr([said|X],X).
sr([say|X],X).
sr([says|X],X).
sr([she|X],X).
sr([should|X],X).
sr([since|X],X).
sr([so|X],X).
sr([some|X],X).
sr([take|X],X).
sr([than|X],X).
sr([this|X],X).
sr([that|X],X).
sr([their|X],X).
sr([them|X],X).
sr([these|X],X).
sr([they|X],X).
sr([tis|X],X).
sr([to|X],X).
sr([too|X],X).
sr([twas|X],X).
sr([us|X],X).
sr([wants|X],X).
sr([was|X],X).
sr([we|X],X).
sr([were|X],X).
sr([when|X],X).
sr([where|X],X).
sr([which|X],X).
sr([while|X],X).
sr([who|X],X).
sr([whom|X],X).
sr([why|X],X).
sr([will|X],X).
sr([with|X],X).
sr([would|X],X).
sr([yet|X],X).
sr([you|X],X).
sr([your|X],X).
```

F. et.pl

```
% et.pl - M. Covington      2003 February 12

% ET the Efficient Tokenizer

% Measured speed: On a 1-GHz Pentium III,
% about 6 seconds per megabyte of text tokenized,
% or about three lines of text per millisecond.

% New in this version: Special handling of apostrophes.
% The apostrophe is a whitespace character except in
% the sequence ’t which is treated as just t,
% making morphological analysis easier.

%%
%% User-callable routines
%%

% tokens_words(+Tokens,-Words)
%  From the output of the other routines, extracts just
%  the word tokens and converts them to atoms.

tokens_words([],[]).

tokens_words([w(Chars)|Tokens],[Atom|Atoms]) :-
   !,
   atom_chars(Atom,Chars),
   tokens_words(Tokens,Atoms).

tokens_words([_|Tokens],Atoms) :-
   % skip non-word tokens
   tokens_words(Tokens,Atoms).

% tokenize_file(+Filename,-Tokens)
%  Reads and entire file and tokenizes it.

tokenize_file(Filename,Tokens) :-
   open(Filename,read,Stream),
   tokenize_stream(Stream,Tokens),
   close(Stream).

% tokenize_stream(+Stream,-Tokens)
%  Reads an entire stream and tokenizes it.

tokenize_stream(Stream,[]) :-
   at_end_of_stream(Stream),
   !.

tokenize_stream(Stream,Tokens) :-
   tokenize_line_dl(Stream,Tokens/Tail),
   tokenize_stream(Stream,Tail).


% tokenize_line(+Stream,-Tokens)
%  Reads a line of input and returns a list of tokens.

tokenize_line(Stream,Tokens) :-
   tokenize_line_dl(Stream,Tokens/[]).


% tokenize_line_dl(+Stream,-Tokens/Tail)
%  Like tokenize_line, but uses a difference list.
%  This makes it easier to append the results of successive calls.

tokenize_line_dl(Stream,Tail/Tail) :-
   at_end_of_stream(Stream),                        % unnecessary test?
   !.

tokenize_line_dl(Stream,Dlist) :-
```

```
   get_char_and_type(Stream,Char,Type),
   tokenize_line_x(Type,Char,Stream,Dlist).

%%
%% Auxiliary predicates for tokenization
%%

% get_char_and_type(+Stream,-Char,-Type)
%  Reads a character, determines its type, and translates
%  it as specified in char_type_char.

get_char_and_type(Stream,Char,Type) :-
   get_char(Stream,C),
   char_type_char(C,Type,Char).


% tokenize_line_x(+Type,+Char,+Stream,-Tokens/Tail)
%  Tokenizes (the rest of) a line of input.
%  Type and Char describe the character that has been read ahead.

tokenize_line_x(eol,_,_,Tail/Tail) :-              % end of line mark; terminate
   !.

tokenize_line_x(whitespace,_,Stream,Dlist) :-      % whitespace, skip it
   !,
   tokenize_line_dl(Stream,Dlist).


% Word tokens and number tokens have to be completed,
% maintaining 1 character of read-ahead as this is done.
% NewChar and NewType are the character read ahead
% after completing the token.

tokenize_line_x(letter,Char,Stream,[w(T)|Tokens]/Tail) :-
   !,
   tokenize_letters(letter,Char,Stream,T,NewType,NewChar),
   tokenize_line_x(NewType,NewChar,Stream,Tokens/Tail).

tokenize_line_x(digit,Char,Stream,[n(T)|Tokens]/Tail) :-
   !,
   tokenize_digits(digit,Char,Stream,T,NewType,NewChar),
   tokenize_line_x(NewType,NewChar,Stream,Tokens/Tail).


% A period is handled like a digit if it is followed by a digit.
% This handles numbers that are written with the decimal point first.

tokenize_line_x(_, '.', Stream,Dlist) :-
   peek_char(Stream,P),
   char_type_char(P,digit,_),
   !,
   % Start over, classifying '.' as a digit
   tokenize_line_x(digit, '.', Stream,Dlist).


% Special characters and unidentified characters are easy:
% they stand by themselves, and the next token begins with
% the very next character.

tokenize_line_x(special,Char,Stream,[s(Char)|Tokens]/Tail) :-   % special char
   !,
   tokenize_line_dl(Stream,Tokens/Tail).

tokenize_line_x(_,Char,Stream,[other(Char)|Tokens]/Tail) :-     % unidentified char
   !,
   tokenize_line_dl(Stream,Tokens/Tail).

% tokenize_letters(+Type,+Char,+Stream,-Token,-NewChar,-NewType)
%   Completes a word token beginning with Char, which has
%   been read ahead and identified as type Type.
%   When the process ends, NewChar and NewType are the
%   character that was read ahead after the token.

tokenize_letters(letter,Char,Stream,[Char|Rest],NewType,NewChar) :-
   % It’s a letter, so process it, read another character ahead, and recurse.
   !,
   get_char_and_type(Stream,Char2,Type2),
   tokenize_letters(Type2,Char2,Stream,Rest,NewType,NewChar).

tokenize_letters(_,’’’’,Stream,Rest,NewType,NewChar) :-
   %
   % Absorb an apostrophe, but only when it precedes t.
   % This keeps words together like doesn’t, won’t.
   %
   peek_char(Stream,t),
   !,
   get_char(Stream,_),
   tokenize_letters(letter,t,Stream,Rest,NewType,NewChar).

tokenize_letters(Type,Char,_,[],Type,Char).
   % It’s not a letter, so don’t process it; pass it to the calling procedure.


% tokenize_digits(+Type,+Char,+Stream,-Token,-NewChar,-NewType)
%   Like tokenize_letters, but completes a number token instead.
%   Additional subtleties for commas and decimal points.

tokenize_digits(digit,Char,Stream,[Char|Rest],NewType,NewChar) :-
   % It’s a digit, so process it, read another character ahead, and recurse.
   !,
   get_char_and_type(Stream,Char2,Type2),
   tokenize_digits(Type2,Char2,Stream,Rest,NewType,NewChar).

tokenize_digits(_, ’,’, Stream,Rest,NewType,NewChar) :-
   peek_char(Stream,P),
   char_type_char(P,digit,Char2),
   !,
   % It’s a comma followed by a digit, so skip it and continue.
   get_char(Stream,_),
   tokenize_digits(digit,Char2,Stream,Rest,NewType,NewChar).

tokenize_digits(_, ’.’, Stream,[’.’|Rest],NewType,NewChar) :-
   peek_char(Stream,P),
   char_type_char(P,digit,Char2),
   !,
   % It’s a period followed by a digit, so include it and continue.
   get_char(Stream,_),
   tokenize_digits(digit,Char2,Stream,Rest,NewType,NewChar).

tokenize_digits(Type,Char,_,[],Type,Char).
   % It’s not any of those, so don’t process it;
   % pass it to the calling procedure.

%%
%% Character classification
%%

% char_type_char(+Char,-Type,-TranslatedChar)
%   Classifies all characters as letter, digit, special, etc.,
%   and also translates each character into the character that
%   will represent it, converting upper to lower case.

char_type_char(Char,Type,Tr) :-
   char_table(Char,Type,Tr),
   !.

char_type_char(Char,special,Char).
```

```
% End of line marks
char_table(end_of_file, eol, end_of_file).
char_table('\n',        eol, '\n'       ).

% Whitespace characters
char_table(' ',     whitespace,  ' ').     % blank
char_table('\t',    whitespace,  ' ').     % tab
char_table('\r',    whitespace,  ' ').     % return
char_table('''',    whitespace, '''').     % apostrophe does not translate to blank

% Letters
char_table(a,     letter,    a ).
char_table(b,     letter,    b ).
char_table(c,     letter,    c ).
char_table(d,     letter,    d ).
char_table(e,     letter,    e ).
char_table(f,     letter,    f ).
char_table(g,     letter,    g ).
char_table(h,     letter,    h ).
char_table(i,     letter,    i ).
char_table(j,     letter,    j ).
char_table(k,     letter,    k ).
char_table(l,     letter,    l ).
char_table(m,     letter,    m ).
char_table(n,     letter,    n ).
char_table(o,     letter,    o ).
char_table(p,     letter,    p ).
char_table(q,     letter,    q ).
char_table(r,     letter,    r ).
char_table(s,     letter,    s ).
char_table(t,     letter,    t ).
char_table(u,     letter,    u ).
char_table(v,     letter,    v ).
char_table(w,     letter,    w ).
char_table(x,     letter,    x ).
char_table(y,     letter,    y ).
char_table(z,     letter,    z ).
char_table('A',   letter,    a ).
char_table('B',   letter,    b ).
char_table('C',   letter,    c ).
char_table('D',   letter,    d ).
char_table('E',   letter,    e ).
char_table('F',   letter,    f ).
char_table('G',   letter,    g ).
char_table('H',   letter,    h ).
char_table('I',   letter,    i ).
char_table('J',   letter,    j ).
char_table('K',   letter,    k ).
char_table('L',   letter,    l ).
char_table('M',   letter,    m ).
char_table('N',   letter,    n ).
char_table('O',   letter,    o ).
char_table('P',   letter,    p ).
char_table('Q',   letter,    q ).
char_table('R',   letter,    r ).
char_table('S',   letter,    s ).
char_table('T',   letter,    t ).
char_table('U',   letter,    u ).
char_table('V',   letter,    v ).
char_table('W',   letter,    w ).
char_table('X',   letter,    x ).
char_table('Y',   letter,    y ).
char_table('Z',   letter,    z ).

% Digits
char_table('0',   digit,     '0' ).
char_table('1',   digit,     '1' ).
char_table('2',   digit,     '2' ).
char_table('3',   digit,     '3' ).
char_table('4',   digit,     '4' ).
char_table('5',   digit,     '5' ).
char_table('6',   digit,     '6' ).
char_table('7',   digit,     '7' ).
char_table('8',   digit,     '8' ).
char_table('9',   digit,     '9' ).

% Everything else is a special character.
```

APPENDIX D
C# CODE

A. *Form1.cs*

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace NLP
{
    public partial class Form1 : Form
    {
        private NLP nlp;
        public Form1()
        {
            InitializeComponent();
            nlp = new NLP();
        }

        private void commandEntryTextBox_KeyDown(object sender, KeyEventArgs e)
        {
            if(e.KeyData.Equals(Keys.Enter) && (commandEntryTextBox.Text.Length > 0))
            {
                List<string> commands = nlp.process(commandEntryTextBox.Text);
                foreach(string command in commands)
                {
                    consoleOutput.AppendText(command + "\n");
                }
            }
        }
    }
}
```

B. *NLP.cs*

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace NLP
{
    class NLP
    {
        Parser parser;
        SpellChecker spellchecker;
        public NLP()
        {
            parser = new Parser();
            spellchecker = new SpellChecker();
        }
        public List<string> process(string input)
        {
```

```
    //turn input into a list of strings (individual words)
    List<string> tokens = parser.tokenise(input);
    tokens = parser.simplify(tokens);
    tokens = spellchecker.spellCheck(tokens);
    //turn that list into a list of lists (commands)
    //translate
    List<List<string>> commands = parser.createSubCommands(tokens);
    List<string> msdos = new List<string>();
    foreach (List<string> c in commands)
    {
        string s = translate(c);
        msdos.Add(s);
    }
    return msdos;
}

private string translate(List<string> command)
{
    string result = "I do not understand, try re-phrasing the command";

    if (command.Contains("show"))
    {
        result = translateShow(command);
    }
    else if (command.Contains("properties"))
    {
        result = translateProperties(command);
    }
    else if (command.Contains("type"))
    {
        result = translateType(command);
    }
    else if (command.Contains("move"))
    {
        result = translateMove(command);
    }
    else if (command.Contains("transfer"))
    {
        result = translateTransfer(command);
    }
    else if (command.Contains("rename"))
    {
        result = translateRename(command);
    }
    else if (command.Contains("create"))
    {
        result = translateCreate(command);
    }
    else if (command.Contains("copy"))
    {
        result = translateCopy(command);
    }
    else if (command.Contains("quit"))
    {
        result = "\x3"; //inject CTRL+C
    }

    return result;
}

private string translateCopy(List<string> command)
{
    StringBuilder builder = new StringBuilder();
    for (int i = 0; i < command.Count(); i++)
    {
        if (command[i].Equals("copy"))
        {
            builder.Append("copy ");
        }
```

```
            else if (command[i].Equals("all") &&
                !command.Contains("text") &&
                !command.Contains("latex") &&
                !command.Contains("word"))
            {
                builder.Append("*.* ");
            }
            else if (command[i].Equals("text"))
            {
                builder.Append("*.txt ");
            }
            else if (command[i].Equals("latex"))
            {
                builder.Append("*.tex ");
            }
            else if (command[i].Equals("word"))
            {
                builder.Append("*.doc ");
            }
            else if (command[i].Equals("drive") && i < command.Count - 1)
            {
                builder.Append(command[i + 1] + ": ");
            }
            else if (command[i].Equals("directory") && i < command.Count - 1)
            {
                builder.Append(command[i + 1]);
            }
            else if (!command.Contains("drive") && !command.Contains("directory"))
            {
                builder.Append(command[i]);
            }
        }
        string result = builder.ToString();
        return result;
    }

    private string translateCreate(List<string> command)
    {
        StringBuilder builder = new StringBuilder();
        for (int i = 0; i < command.Count(); i++)
        {
            if (command[i].Equals("create"))
            {
                builder.Append("mkdir ");
            }
            else if (command[i].Equals("drive") && i < command.Count - 1)
            {
                builder.Append(command[i + 1] + ": ");
            }
            else if (command[i].Equals("directory") && i < command.Count - 1)
            {
                builder.Append(command[i + 1]);
            }
            else if (!command[i].Equals("new") &&
                !command.Contains("drive") &&
                !command.Contains("directory"))
            {
                builder.Append(command[i]);
            }
        }
        string result = builder.ToString();
        return result;
    }

    private string translateRename(List<string> command)
    {
        StringBuilder builder = new StringBuilder();
        for (int i = 0; i < command.Count(); i++)
        {
            builder.Append(command[i]);
        }
        string result = builder.ToString();
        return result;
    }

    private string translateTransfer(List<string> command)
    {
        StringBuilder builder = new StringBuilder();
        for (int i = 0; i < command.Count(); i++)
        {
            if (command[i].Equals("transfer"))
            {
                builder.Append("move ");
            }
            else if (command[i].Equals("all") &&
                !command.Contains("text") &&
                !command.Contains("latex") &&
                !command.Contains("word"))
            {
                builder.Append("*.* ");
            }
            else if (command[i].Equals("text"))
            {
                builder.Append("*.txt ");
            }
            else if (command[i].Equals("latex"))
            {
                builder.Append("*.tex ");
            }
            else if (command[i].Equals("word"))
            {
                builder.Append("*.doc ");
            }
            else if (command[i].Equals("drive") && i < command.Count - 1)
            {
                builder.Append(command[i + 1] + ": ");
            }
            else if (command[i].Equals("directory") && i < command.Count - 1)
            {
                builder.Append(command[i + 1]);
            }
            else if (!command.Contains("drive") && !command.Contains("directory"))
            {
                builder.Append(command[i]);
            }
        }
        string result = builder.ToString();
        return result;
    }

    private string translateMove(List<string> command)
    {
        StringBuilder builder = new StringBuilder();
        for (int i = 0; i < command.Count(); i++)
        {
            if (command[i].Equals("move"))
            {
                builder.Append("cd ");
            }
            else if (command[i].Equals("back"))
            {
                builder.Append("../ ");
            }
            else if (command[i].Equals("drive") && i < command.Count - 1)
            {
                builder.Append(command[i + 1] + ": ");
            }
            else if (command[i].Equals("directory") && i < command.Count - 1)
            {
                builder.Append(command[i + 1]);
            }
            else if (!command.Contains("drive") && !command.Contains("directory"))
            {
                builder.Append(command[i]);
            }
        }
        string result = builder.ToString();
        return result;
    }

    private string translateType(List<string> command)
    {
        StringBuilder builder = new StringBuilder();
        for (int i = 0; i < command.Count(); i++)
        {
            if (command[i].Equals("type"))
            {
                builder.Append("type ");
            }
            else if (command[i].Equals("drive") && i < command.Count - 1)
            {
                builder.Append(command[i + 1] + ": ");
            }
            else if (command[i].Equals("directory") && i < command.Count - 1)
            {
                builder.Append(command[i + 1]);
            }
            else if(!command.Contains("drive") && !command.Contains("directory"))
            {
                builder.Append(command[i]);
            }
        }
        string result = builder.ToString();
        return result;
    }

    private string translateProperties(List<string> command)
    {
        StringBuilder builder = new StringBuilder();
        for (int i = 0; i < command.Count(); i++)
        {
            if (command[i].Equals("properties"))
            {
                builder.Append("vol ");
            }
            if (command[i].Equals("drive") && i < command.Count - 1)
            {
                builder.Append(command[i + 1] + ": ");
            }
            else if (!command.Contains("drive"))
            {
                builder.Append("C:");
            }
        }
        string result = builder.ToString();
        return result;
    }

    private string translateShow(List<string> command)
    {
        StringBuilder builder = new StringBuilder();
        for (int i = 0; i < command.Count(); i++)
        {
            if (command[i].Equals("show"))
            {
                builder.Append("dir ");
            }
```

```
                else if (command[i].Equals("all") &&
                    !command.Contains("text") &&
                    !command.Contains("latex") &&
                    !command.Contains("word"))
                {
                    builder.Append("*.* ");
                }
                else if (command[i].Equals("text"))
                {
                    builder.Append("*.txt ");
                }
                else if (command[i].Equals("latex"))
                {
                    builder.Append("*.tex ");
                }
                else if (command[i].Equals("word"))
                {
                    builder.Append("*.doc ");
                }
                else if (command[i].Equals("drive") && i < command.Count - 1)
                {
                    builder.Append(command[i + 1] + ": ");
                }
                else if (command[i].Equals("directory") && i < command.Count - 1)
                {
                    builder.Append(command[i + 1]);
                }
            }
            string result = builder.ToString();
            return result;
        }
    }
}
```

C. Parser.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace NLP
{
    class Parser
    {
        private List<string> stopWords;
        private string[,] equivalentWords;
        public Parser()
        {
            populateStopWords();
            populateEquivalentWords();
        }
        public List<string> tokenise(string input)
        {
            input = input.ToLower(); //cast to lower case
            List<string> output = new List<string>();
            StringBuilder builder = new StringBuilder();
            foreach (char c in input)
            {
                //take only desired characters
                if (char.IsLetterOrDigit(c) || (c == ' ') || (c == '\\') || (c == ':') || (c == '.'))
                {
                    builder.Append(c);
                }
            }
            //deliminate by whitespace, split into list of tokens
            output = ((builder.ToString()).Split(' ')).ToList();
            return output;
        }

        public List<string> simplify(List<string> atoms)
        {
            for (int i = 0; i < atoms.Count(); i++)
            {
                if (stopWords.Contains(atoms[i]))
                {
                    atoms.Remove(atoms[i]);
                }
                //divide by two as each array contains two elements
                for (int j = 0; j < equivalentWords.Length / 2; j++)
                {
                    if (atoms[i].Equals(equivalentWords[j, 0]))
                    {
                        atoms[i] = equivalentWords[j, 1]; //swap for equivalent partner
                    }
                }
            }
            return atoms;
        }

        public List<List<string>> createSubCommands(List<string> input)
        {
            List<List<string>> commands = new List<List<string>>();
            commands.Add(new List<string>()); //ensure at least one command
            int counter = 0;
            foreach (string s in input)
            {
                if (s.Equals("then"))
                {
                    commands.Add(new List<string>());
                    counter++;
                }
                else
                {
                    commands.ElementAt(counter).Add(s);
                }
            }
            return commands;
        }

        private void populateStopWords()
        {
            stopWords = new List<string>();
            stopWords.Add("the");
            stopWords.Add("is");
            stopWords.Add("are");
            stopWords.Add("there");
            stopWords.Add("any");
            stopWords.Add("please");
            stopWords.Add("a");
            stopWords.Add("able");
            stopWords.Add("about");
            stopWords.Add("across");
            stopWords.Add("after");
            stopWords.Add("almost");
            stopWords.Add("also");
            stopWords.Add("am");
            stopWords.Add("among");
            stopWords.Add("an");
            stopWords.Add("and");
            stopWords.Add("as");
            stopWords.Add("at");
            stopWords.Add("be");
            stopWords.Add("because");
            stopWords.Add("been");
            stopWords.Add("but");
            stopWords.Add("by");
```

```
stopWords.Add("called");
stopWords.Add("can");
stopWords.Add("cannot");
stopWords.Add("could");
stopWords.Add("dear");
stopWords.Add("did");
stopWords.Add("do");
stopWords.Add("does");
stopWords.Add("either");
stopWords.Add("else");
stopWords.Add("ever");
stopWords.Add("for");
stopWords.Add("from");
stopWords.Add("get");
stopWords.Add("got");
stopWords.Add("had");
stopWords.Add("has");
stopWords.Add("have");
stopWords.Add("he");
stopWords.Add("her");
stopWords.Add("hers");
stopWords.Add("him");
stopWords.Add("his");
stopWords.Add("how");
stopWords.Add("however");
stopWords.Add("i");
stopWords.Add("if");
stopWords.Add("in");
stopWords.Add("into");
stopWords.Add("it");
stopWords.Add("its");
stopWords.Add("just");
stopWords.Add("least");
stopWords.Add("let");
stopWords.Add("like");
stopWords.Add("likely");
stopWords.Add("may");
stopWords.Add("me");
stopWords.Add("might");
stopWords.Add("most");
stopWords.Add("must");
stopWords.Add("my");
stopWords.Add("neither");
stopWords.Add("no");
stopWords.Add("nor");
stopWords.Add("not");
stopWords.Add("of");
stopWords.Add("off");
stopWords.Add("often");
stopWords.Add("on");
stopWords.Add("only");
stopWords.Add("or");
stopWords.Add("other");
stopWords.Add("our");
stopWords.Add("own");
stopWords.Add("rather");
stopWords.Add("said");
stopWords.Add("say");
stopWords.Add("says");
stopWords.Add("she");
stopWords.Add("should");
stopWords.Add("since");
stopWords.Add("so");
stopWords.Add("some");
stopWords.Add("take");
stopWords.Add("than");
stopWords.Add("that");
stopWords.Add("this");
stopWords.Add("their");
```

```
            stopWords.Add("them");
            stopWords.Add("these");
            stopWords.Add("they");
            stopWords.Add("tis");
            stopWords.Add("to");
            stopWords.Add("too");
            stopWords.Add("twas");
            stopWords.Add("us");
            stopWords.Add("wants");
            stopWords.Add("was");
            stopWords.Add("we");
            stopWords.Add("were");
            stopWords.Add("when");
            stopWords.Add("where");
            stopWords.Add("which");
            stopWords.Add("while");
            stopWords.Add("who");
            stopWords.Add("whom");
            stopWords.Add("why");
            stopWords.Add("will");
            stopWords.Add("with");
            stopWords.Add("would");
            stopWords.Add("yet");
            stopWords.Add("you");
            stopWords.Add("your");
        }

        private void populateEquivalentWords()
        {
            equivalentWords = new string[24, 2] { {"display","show"}, { "list", "show" },
            { "describe", "properties" },{ "details", "properties" },{"vol","properties"},
            {"read","type"},{"open","type"},{"load","type"},{"go","move"},
            {"relocate","transfer"},{"disk","drive"},{"what","files"},{"file","files"},
            {"everything","all"},{"folder","directory"},{"make","create"},{"every","all"},
            {"next","then"},{"before","then"},{"name","rename"},{"exit","quit"},
            {"stop","quit"}, {"replicate","copy" }, {"duplicate","copy"}};
        }
    }
}
```

D. SpellChecker.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace NLP
{
    class SpellChecker
    {
        List<string> knownWords;
        public SpellChecker()
        {
            knownWords = new List<string>();
            populateKnownWords();
        }

        public List<string> spellCheck(List<string> atoms)
        {

            for (int s = 0; s < atoms.Count; s++)
            {
                for (int k = 0; k < knownWords.Count; k++)
                {
                    atoms[s] = levenshteinDistaince(knownWords[k], atoms[s]);
                    if(atoms[s].Equals(knownWords[k]))
                    {
                        break;
                    }
                }
            }
            return atoms;
        }

        private string levenshteinDistaince(string known, string atom)
        {
            int distance = 0;
            int minLength = Math.Min(known.Length, atom.Length);
            for (int i = 0; i < minLength; i++)
            {
                if (!(known.ElementAt(i).Equals(atom.ElementAt(i))))
                {
                    distance++;
                }
            }
            distance += Math.Abs(known.Length - atom.Length);
            if (distance < 3)
            {
                return known;
            }
            else
            {
                return atom;
            }
        }

        private void populateKnownWords()
        {
            knownWords.Add("type");
            knownWords.Add("properties");
            knownWords.Add("create");
            knownWords.Add("transfer");
            knownWords.Add("copy");
            knownWords.Add("quit");
            knownWords.Add("show");
            knownWords.Add("rename");
            knownWords.Add("disk");
        }
    }
}
```