

CSE341

MIDTERM PROJECT

SUBMISSION REPORT

Problem Solution

First of all, with the read-file function, I put all the characters in the file into a list and returned. Then I started to apply preprocess operations for input from the lineage. First I concatenated the strings by calling the function merge-string-variables (like legs, horse). Then I implemented a function (convert-string-to-list function) that converts the parentheses format received from the user into a list. Here I used the stack logic, values are pushed into the list until the closed parentheses come. If closed parentheses appear, pop up until the open parenthesis appears from the list, and each value is thrown to another list. After the open parenthesis comes, the other list is pushed back to the original list. Thus, the bracket format in the input file is converted into a list without breaking. Then evaluate-algorithm function is run to run queries. In this function, each process is navigated one by one. If it is fact or rule, these transactions are sent to the fact-list and rule-list lists. If it is query, it calls evaluate-query function. And every result from evaluate-query is append to each-result list. Thus, the answer from each query is kept in one place. And by using this each-result, it is written to the file. If we need to explain evaluate-query function that runs queries, first of all it is checked whether this query fact exists. A fact that matches the query is searched by calling find-match-facts function. If yes, evaluate-query returns true. If not, the list of the matching rules is obtained by calling the find-match-rules function. And if this list is not empty, each corresponding rule is sent to evaluate-query-for-rule function together with the query input. If a result returned from here is true, evaluate-query function returns true. evaluate-query-for-rule runs the functions in the body part of the rule according to the query. Here, instead of the parameters that need to be changed (for example, typing "horse" instead of "X"), the parameters in the query are added. Of course, these are only for parameters with capital letters. Only those parameters change according to the query. After the parameters are changed, all the functions in the body part of the rule are sent one by one to evaluate-query function. Thus, recursion is provided. Every result from body is thrown into a list. If one of these results is "nil", evaluate-query-for-rule function returns "nil" (false). If all the obtained results are "true", evaluate-query-for-rule function returns "true". And so the program works by providing recursion. The sample inputs and outputs of the program are as follows.

input.txt

```
≡ input.txt
1  (
2      ( ("legs" ("X" 2)) ( ("mammal" ("X")) ("arms" ("X" 2)) ) )
3      ( ("legs" ("X" 4)) ( ("mammal" ("X")) ("arms" ("X" 0)) ) )
4      ( ("mammal" ("horse")) ( ) )
5      ( ("arms" ("horse" 0)) ( ) )
6      ( ( ) ("legs" ("horse" 4)) ) |
7  )
```

output.txt

```
≡ output.txt
1  true
2  |
```

More complex test

input.txt

```
≡ input.txt
1  ∨ (
2      ( ("recursion" ("X")) ( ("can" ("X" 4)) ) )
3      ( ("can" ("X" 4)) ( ("legs" ("X" 4)) ) )
4      ( ("legs" ("X" 2)) ( ("mammal" ("X")) ("arms" ("X" 2)) ) )
5      ( ("legs" ("X" 4)) ( ("mammal" ("X")) ("arms" ("X" 0)) ) )
6      ( ("mammal" ("horse")) ( ) )
7      ( ("arms" ("horse" 0)) ( ) )
8      ( ( ) ("recursion" ("horse")) )
9      ( ( ) ("legs" ("horse" 2)) )
10     ( ( ) ("mammal" ("horse")) )
11 )
```

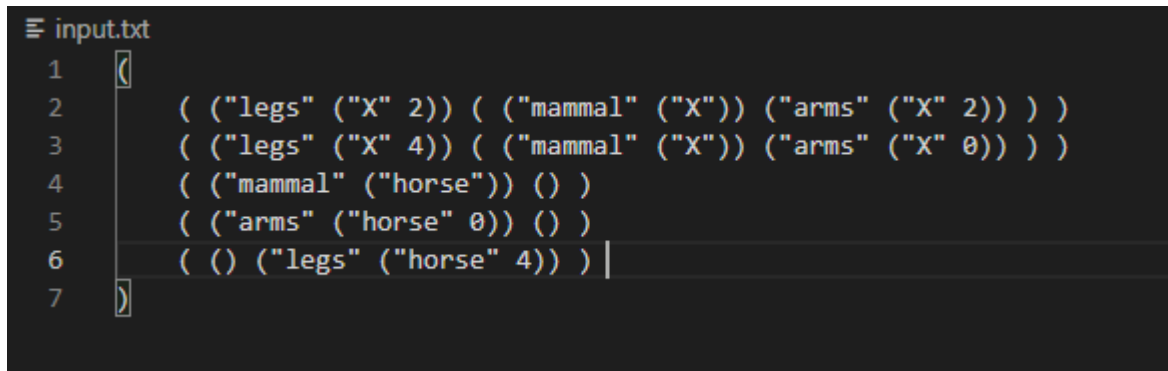
output.txt

```
≡ output.txt
1  true
2  false
3  true
4
```

In this report, I will explain the parts of the midterm project that I applied and what I did not apply, and how I did the project. First of all, the program I made only returns true false. It evaluates the incoming query according to the rules and facts and returns true or false to the "output.txt" file.

Results can be obtained from more than one query in the program.

The format of the "input.txt" file that should be given to the program must be correct. In the input given as an example in pdf, only quotation marks have been changed. Quotes must be ". Example" input.txt "is as follows.



```
≡ input.txt
1  (
2      ( ("legs" ("X" 2)) ( ("mammal" ("X")) ("arms" ("X" 2)) ) )
3      ( ("legs" ("X" 4)) ( ("mammal" ("X")) ("arms" ("X" 0)) ) )
4      ( ("mammal" ("horse")) ( ) )
5      ( ("arms" ("horse" 0)) ( ) )
6      ( ( ) ("legs" ("horse" 4)) ) |
7  )
```

The layout of spaces or new lines is not a problem. They are handled during the program. Only the order of the parentheses and the rule, fact and query must be in the correct format. Only the order of the parentheses and the rule, fact and query must be in the correct format. There is a high probability that the program will run incorrectly in a faulty input format.

If the first character in the string is a digit, this string is considered a number. Please set the inputs like this. For example "2abc" is considered as a digit.

There is NO handle operation for object names (such as "legs" "mammal"). It is not checked whether the initials are capital or small. Therefore, please be careful when using capital letters. This control is provided only for parameters (such as "X"). Its reason is due to the working logic of the program.