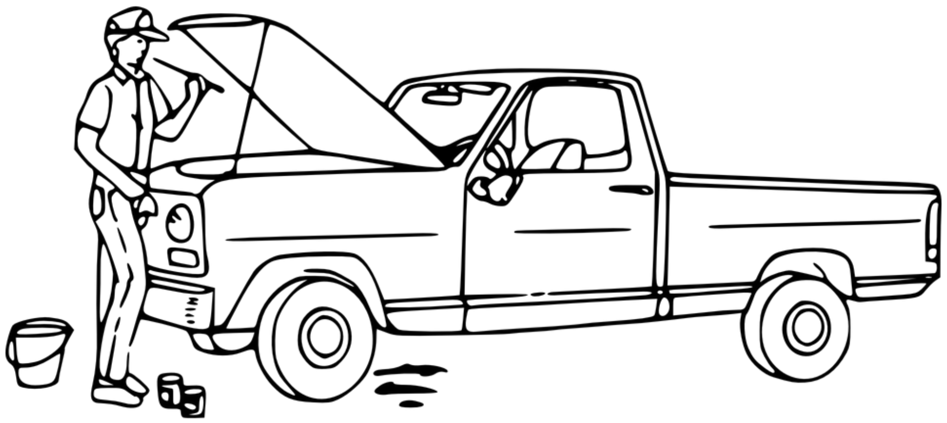


Diagnosing an Automobile Repair

An Expert system used for Car Failure Detection and Identification

Author: Carter Kindred

Class: CS 4346 (Intro to Artificial Intelligence)

Instructor: Dr. Moonis Ali

Date: October 7, 2019

Table of Contents

1. Introduction

1.1 Problem Statement

There are many issues that can come with one's own personal automobiles. It's important to understand the main issues of the automobiles so it can be repaired with The most effective way possible. Addressing only the correct problems is essential for all parties because if the incorrect problems are addressed through lack of knowledge or other reasons, there can cause lots of financial issues addressed. To reach the correct conclusions and diagnosing the car correctly.

1.2 Solution

Most of the issues for automobiles can be narrowed down to the location of the area that is having problems from outside observations. Such as a steaming hood, there is most likely a hood related problem for the automobile. The purpose of this project is to develop an intelligent expert system that can take a user's input relate it to a problem that is going on from the user's perspective and apply feedback using a knowledge base with backward and forward chaining. From this knowledge base, it should provide the user the correct information to correct such problems occurring from there automobile.

2. Contributions

2.1. Team Members: Kelby Webster, Abel Shinabery, and Carter Kindred

2.1.1. Kelby Webster

Forward Chaining Decision Tree

· Dependency Diagram

· Reference List

2.1.2.Abel Shinabery

· Coding of forward and backward chaining

2.1.3. Carter Kindred

· Forward Chaining function and implementation

· Backward Chaining Decision Tree

· Test and debug

2.1.4 All members

Rule format for Knowledge Base rules

Test and debug of entire application

Research for Knowledge Base rules

3. Analysis of problem

3.1 Domain

The domain for this system is all the unsure clients who have an issue with there personal automobiles and are unsure on how to fix it. Our goal as a team for this project is to provide proper solutions for clients seeking to fix their vehicle.

3.2 Problem with Existing code

The problems with the existing code provided us by the instructor is that it compiles and works but it is not and optimal solution for the given problem. Our group found issues with both reusability and readability. The readability problem was that the code was very hard to read and our group had lots of difficulting going through the code and understanding it. For the reusability aspect, could be improved by making changes to the source code, because many of the variable names were hard-coded.

3.3 Proposal for the Problem

Because of the problems with the original code and its lack of adaptability. We decided to make a more succinct, clear and useable changes for our expert system. For our expert system we have both forward and backward chaining in our program with there own text files and knowledge bases to be used. With the creation of our inference engine we used classes to extract the needed rules from our various knowledge bases and applied them appropriately. This in turns makes our expert system better and more efficient which is a vast improvement from the previous given code.

4. Knowledge Base Design

4.1 Decision trees

The forward chaining decision tree starts with the clause variable info list, from the clausevarinfo.h file. With the user's given input, it creates an IF statement based upon the various areas of the car that can potentially have an issue using matching from the keywords within the CarRepair.txt file. The areas that are specified are starting the car, under the hood, brakes, tires and outside, inside. Once the general area is known, it asks additional specific questions to help find the correct solution for the user.

For backward chaining, the problem begins with a query that the user enters stating what the problem with the automobile is, it does this by implementing the rulevarinfo.h file. Then the conclusion is matched to rule that contains a keyword in the CarDiagnoiss.txt file. When the conclusion is found, additional questions are asked by the user to find exactly if there is a problem, either yes or no and the reasons why there is an issue happening with the car.

4.2 Rules

The goal with the rules and strategy of our system is to help determine what repair and actions to take with forward chaining and what the specific issue with the car with backwards chaining. We backwards chain from the fault in the car and forward chain to an appropriate repair.

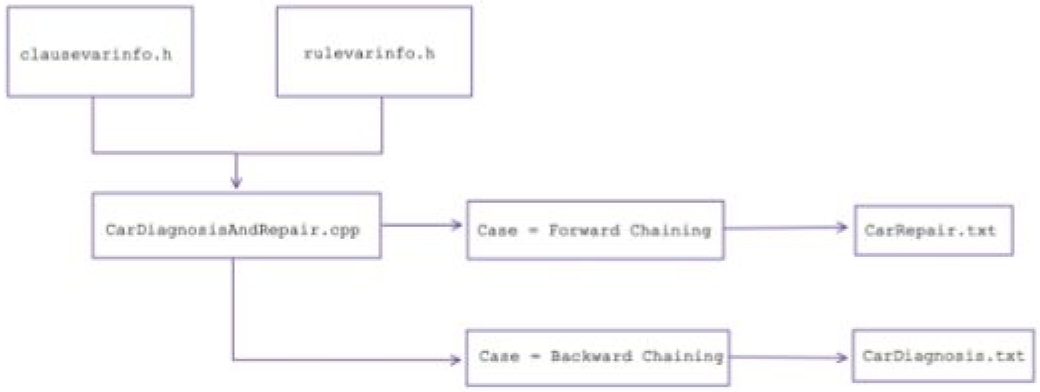
5.1 Inference Engine

An in-depth look of the internal structure of the number of key functions within our driver program which were developed

6. Program Implementation

6.1 Dependency Diagram

This diagram shows a visual representation our our system. The rules and clause variables are loaded first into the main driver file. The system then detects whether it is backward or forward chaining, and then proceeds to go the the correct knowledge base and parses the correct files. After it is done parsing, the driver function executes and returns the data, giving the correct instantiated rule its own set/notes, then gives the user the returned final repair decision.



6.2 Backward Chaining

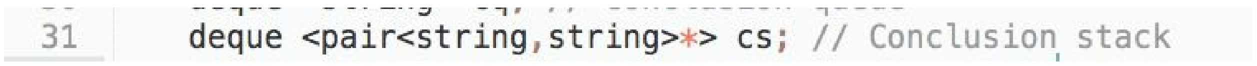
6.2.1 Backward Chaining Algorithm Description

Backward chaining is goal-driven which for our expert system is whether or not there is a problem in the specific location of the car. It first starts off with the effect or consequence, the THEN class and then backwards chains until it finds the causes of said consequence. It considers all the possibles paths that can lead to the conclusion. You then map out the conditions that are the various pathways. The path ways represent the variable list and clause variable list. Then what happens is known as recursive chaining. Which essentially looks for all consequences which are reached during the procedure.

6.6.2 Data Structures

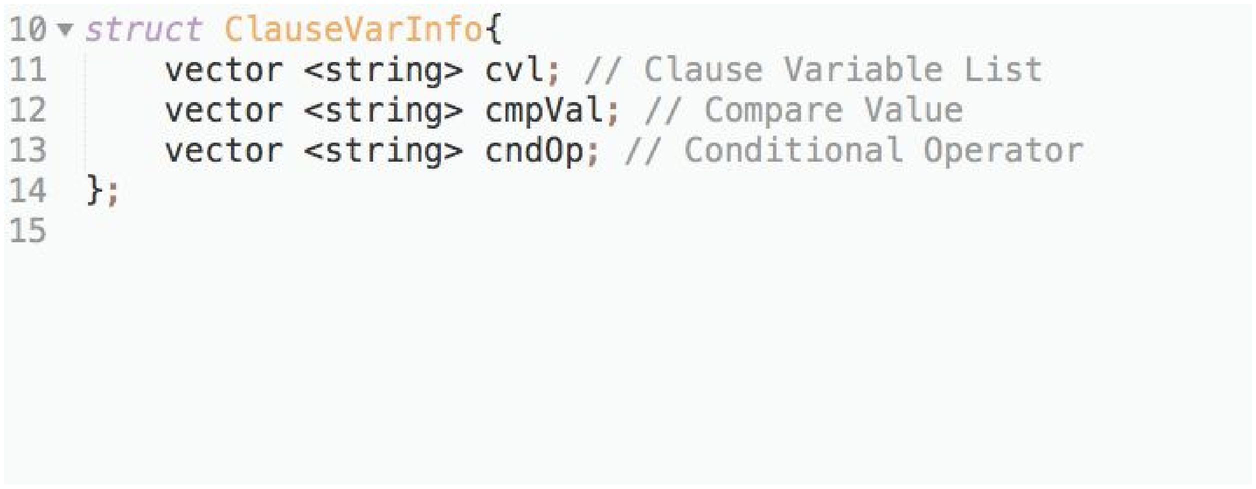
In our code we kept the same 4 main data structures, that were mentioned in class.

Conclusion stack



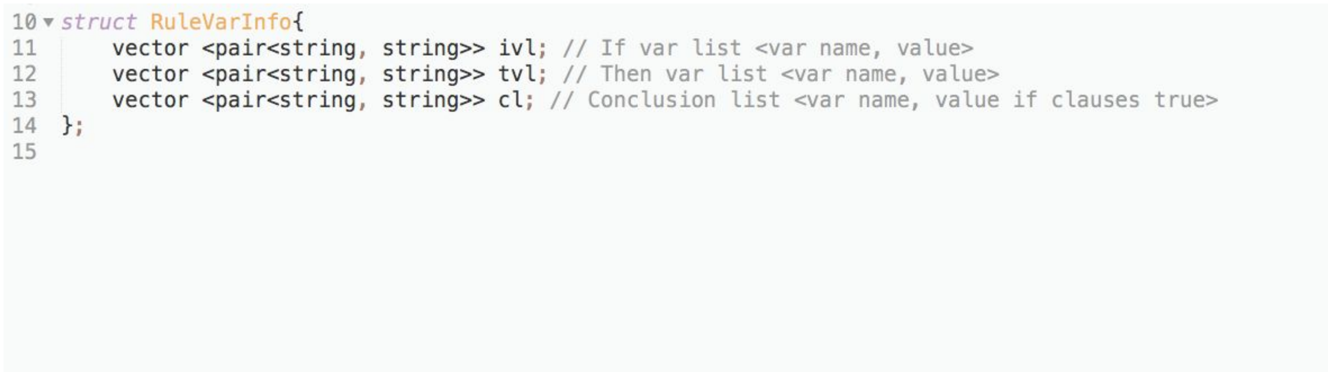
```
    deque <pair<string,string>*> cs; // Conclusion stack
```

Clause Variable List



```cpp
struct ClauseVarInfo{
    vector <string> cvl; // Clause Variable List
    vector <string> cmpVal; // Compare Value
    vector <string> cndOp; // Conditional Operator
};
```

Variable List & Conclusion List



```cpp
struct RuleVarInfo{
    vector <pair<string, string>> ivl; // If var list <var name, value>
    vector <pair<string, string>> tvl; // Then var list <var name, value>
    vector <pair<string, string>> cl; // Conclusion list <var name, value if clauses true>
};
```

6.3 Forward Chaining

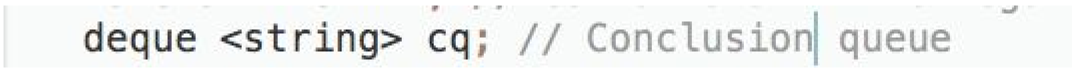
6.3.1 Forward Chaining Algorithm Description

Forward chaining is data-driven. There are specific conditions and want to figure out the conclusion that will be revealed from the conditions being instantiated. This process is done using the knowledge base logic. When you add a fact a process is triggered and the system goes through the rules and it sees if you can generate any new info pertaining to the automobile from them.

6.3.2 Data Structures

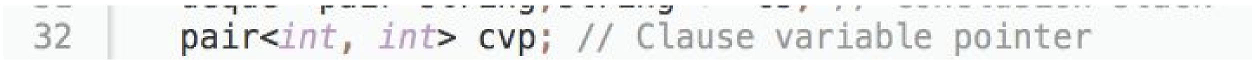
Uses the same 4 data structures that were mentioned in class.

Conclusion Variable Queue



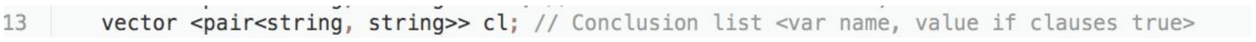
```
deque <string> cq; // Conclusion queue
```

Clause Variable Pointer



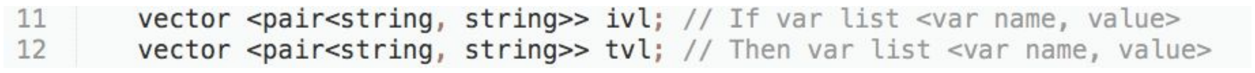
```
pair<int, int> cvp; // Clause variable pointer
```

Clause Variable List



```
vector <pair<string, string>> cl; // Conclusion list <var name, value if clauses true>
```

Variable List



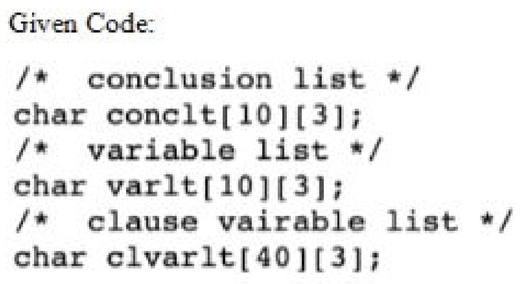
```
vector <pair<string, string>> ivl; // If var list <var name, value>
vector <pair<string, string>> tvl; // Then var list <var name, value>
```

7. Analysis of Programs - Instructor-provided Code Vs. Student Code

The goal of the project was to optimise and improve upon the given code that was originally provided to us for the project. This program was given to us in two separate files, one for forward chaining and one for backward chaining. The code given to us had a knowledge base that was its own file, hard-coded variables and several switch statements. These aspects had many issues coding wise. The following are the main aspects that we emphasized improvements for our expert system.

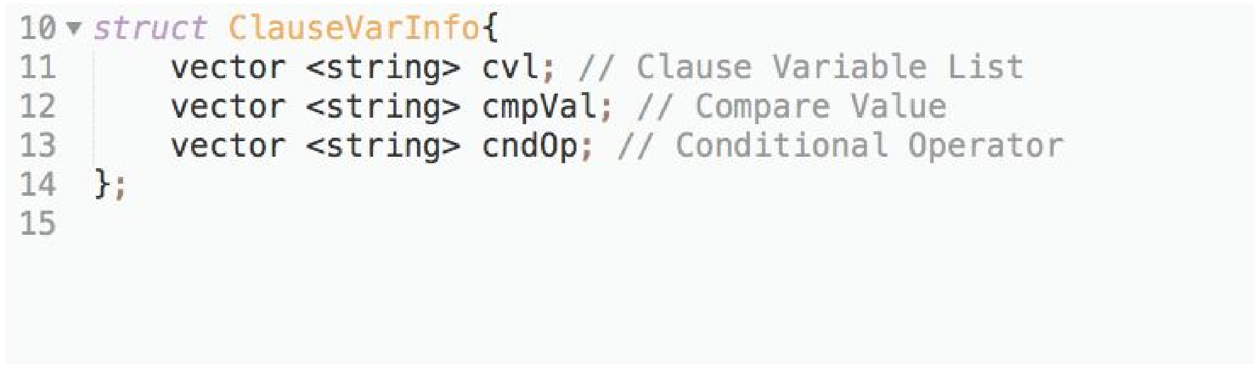
7.1 Readability

Our team worked extensively to help make the code more readable and simple, as opposed to the complicated unreadable original program. The code snippets are below



Given Code:

```
/*  conclusion list */
char conclt[10][3];
/*  variable list */
char varlt[10][3];
/*  clause vairable list */
char clvarlt[40][3];
```



```
struct ClauseVarInfo{
    vector <string> cvl; // Clause Variable List
    vector <string> cmpVal; // Compare Value
    vector <string> cndOp; // Conditional Operator
};
```

With the comparison between the two code examples, you can see that our example uses vectors instead of character arrays. Which is a lot easier to read and maintain. It is especially useful to use vectors because it avoids all the issues that come with using arrays, thus allowing it to run smoothly without any problems.

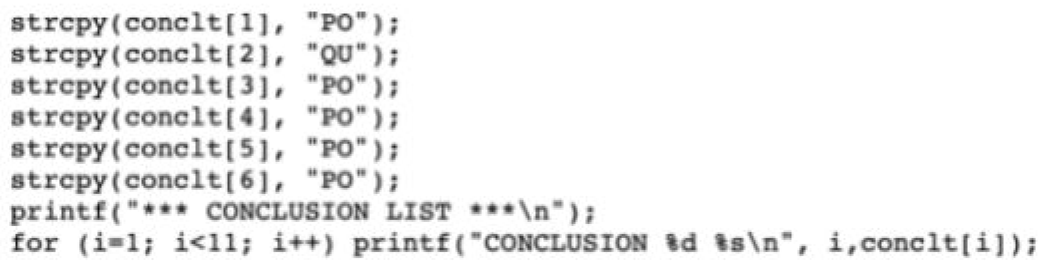
7.2 Maintainability

In our expert system we used a modular approach, which is the process of subdividing a program into separate sub-programs. So it enhances the ability to test and correct any issues we have with our code seamlessly with minimal issues. We also used object oriented design to help with the maintainability in such that that oop offers encapsulation which helps with the safety and access of data. The use of the clause and rule objects are what simplified the program

7.3 Reusability

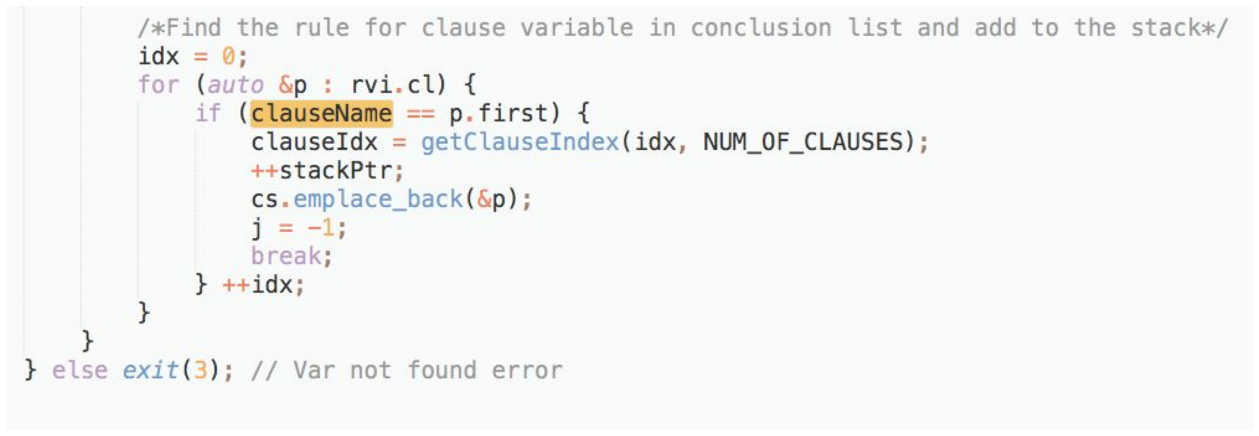
The original code used all hard-coded variables, which had predefined sizes in the system. The difference with our code is that our code has individual textfiles for both the forward and backward chaining and their respective knowledge bases. Examine the code snippets before:

Given code:



```
strcpy(conclt[1], "PO");
strcpy(conclt[2], "QU");
strcpy(conclt[3], "PO");
strcpy(conclt[4], "PO");
strcpy(conclt[5], "PO");
strcpy(conclt[6], "PO");
printf("*** CONCLUSION LIST ***\n");
for (i=1; i<11; i++) printf("CONCLUSION %d %s\n", i,conclt[i]);
```

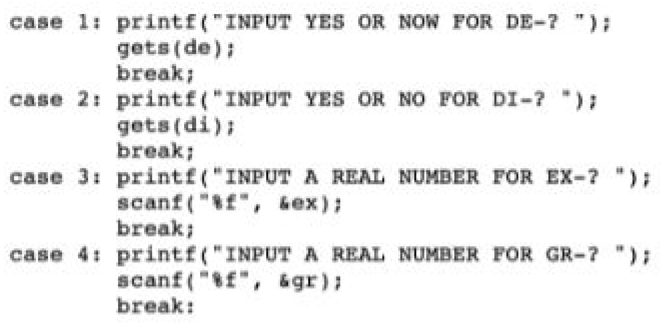
Our code:



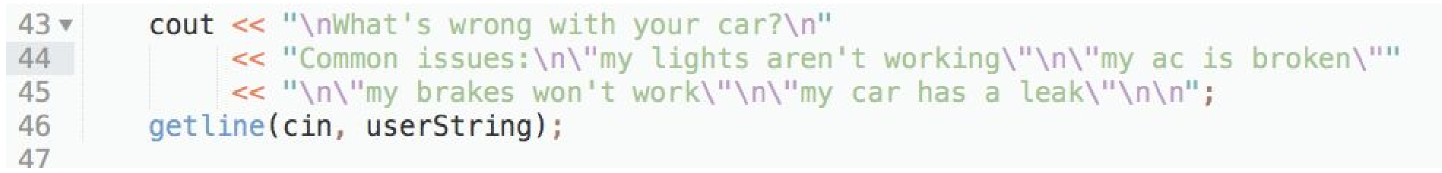
```
        /*Find the rule for clause variable in conclusion list and add to the stack*/
        idx = 0;
        for (auto &p : rvi.cl) {
            if (clauseName == p.first) {
                clauseIdx = getClauseIndex(idx, NUM_OF_CLAUSES);
                ++stackPtr;
                cs.emplace_back(&p);
                j = -1;
                break;
            } ++idx;
        }
    }
} else exit(3); // Var not found error
```

7.4 Usability

In the original code, there was a user interface that was very unclear, vague when the user was prompted to give info. With ours the user is guided simply and efficiently while always providing flexible phrasing and long phrasing, that will be understand the problem easier and reach the conclusion faster. Given code below, our original example and our code.



```
case 1: printf("INPUT YES OR NOW FOR DE-? ");
        gets(de);
        break;
case 2: printf("INPUT YES OR NO FOR DI-? ");
        gets(di);
        break;
case 3: printf("INPUT A REAL NUMBER FOR EX-? ");
        scanf("%f", &ex);
        break;
case 4: printf("INPUT A REAL NUMBER FOR GR-? ");
        scanf("%f", &gr);
        break:
```



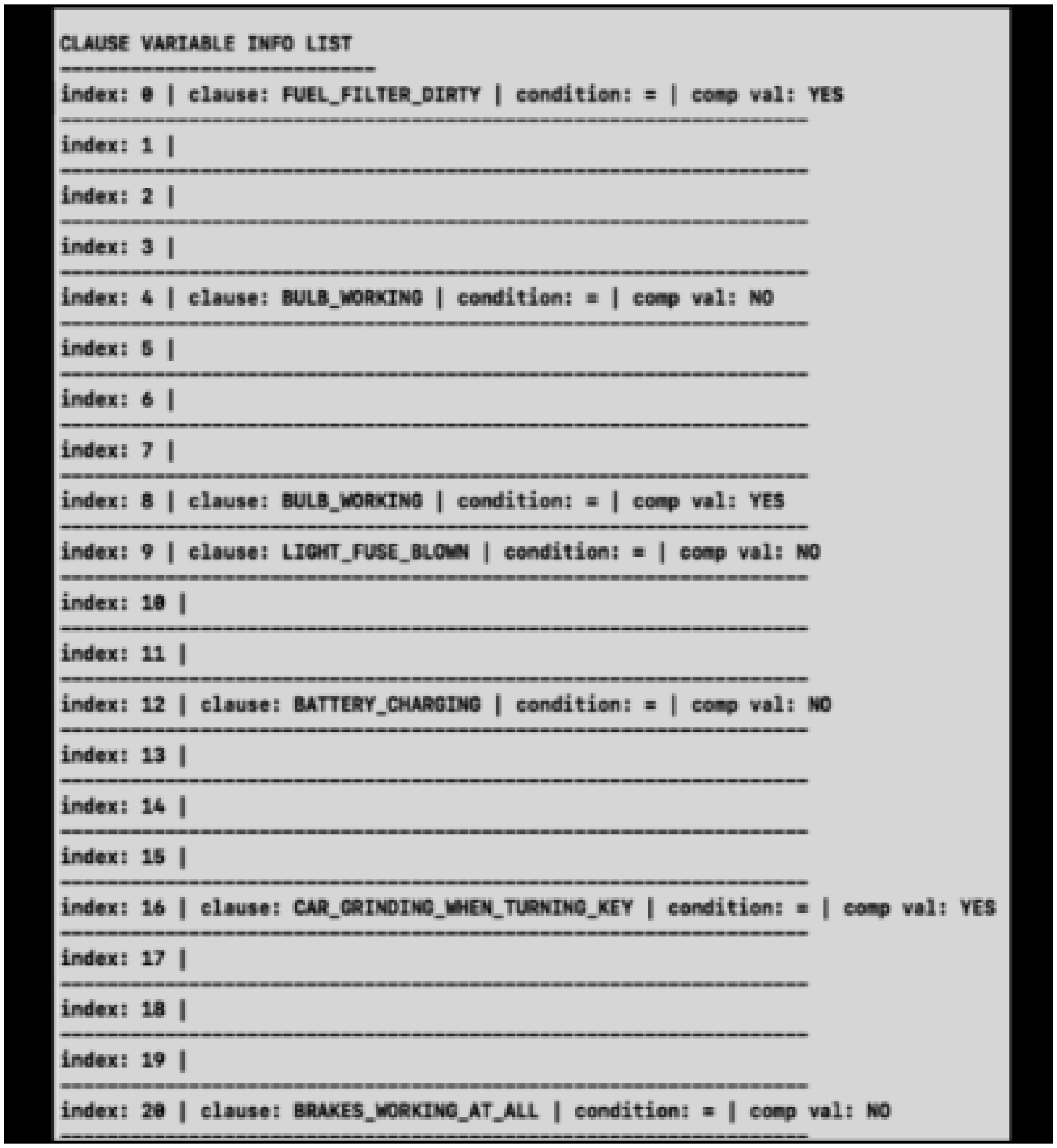
```
    cout << "\nWhat's wrong with your car?\n"
         << "Common issues:\n\"my lights aren't working\"\n\"my ac is broken\""
         << "\n\"my brakes won't work\"\n\"my car has a leak\"\n\n";
    getline(cin, userString);
```

7.5 Efficiency

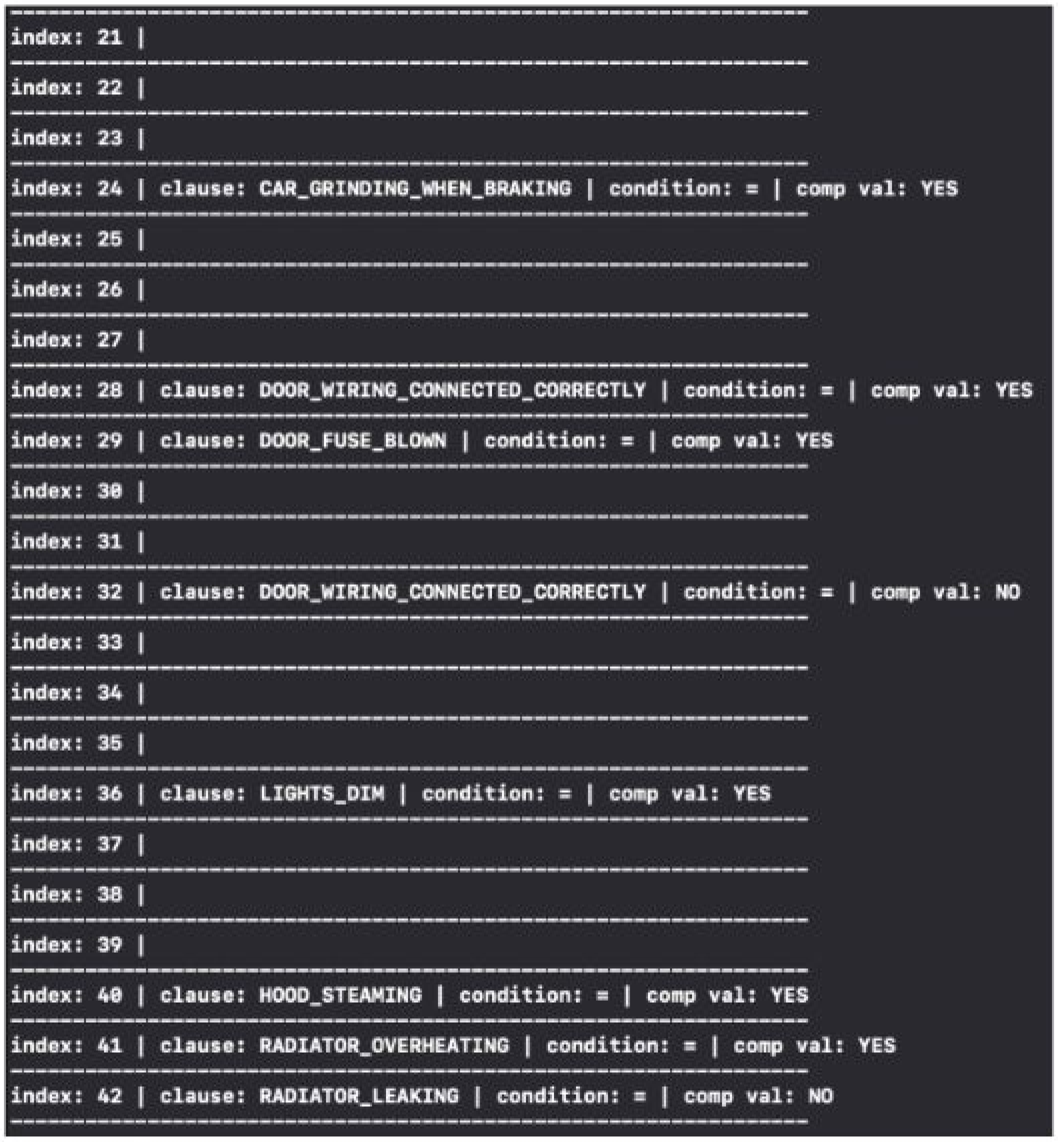
Overall, the new programs the new space and time complexity is worse. But with the benefits that come with the knowledge base being seperated, the comparison between the efficient between the two codes are complete difference, with our program being vastly superior on the efficiency side point. The data structures are reused throughout with the program, making access to the variables with backward and forward chaining extremely easy to use and test/debug with this process. Forward chaining does have access tot the variables instantiated during the backward chaining, thus reducing the information the system must store and apply to the users specific questions.

8. Sample Run of Application

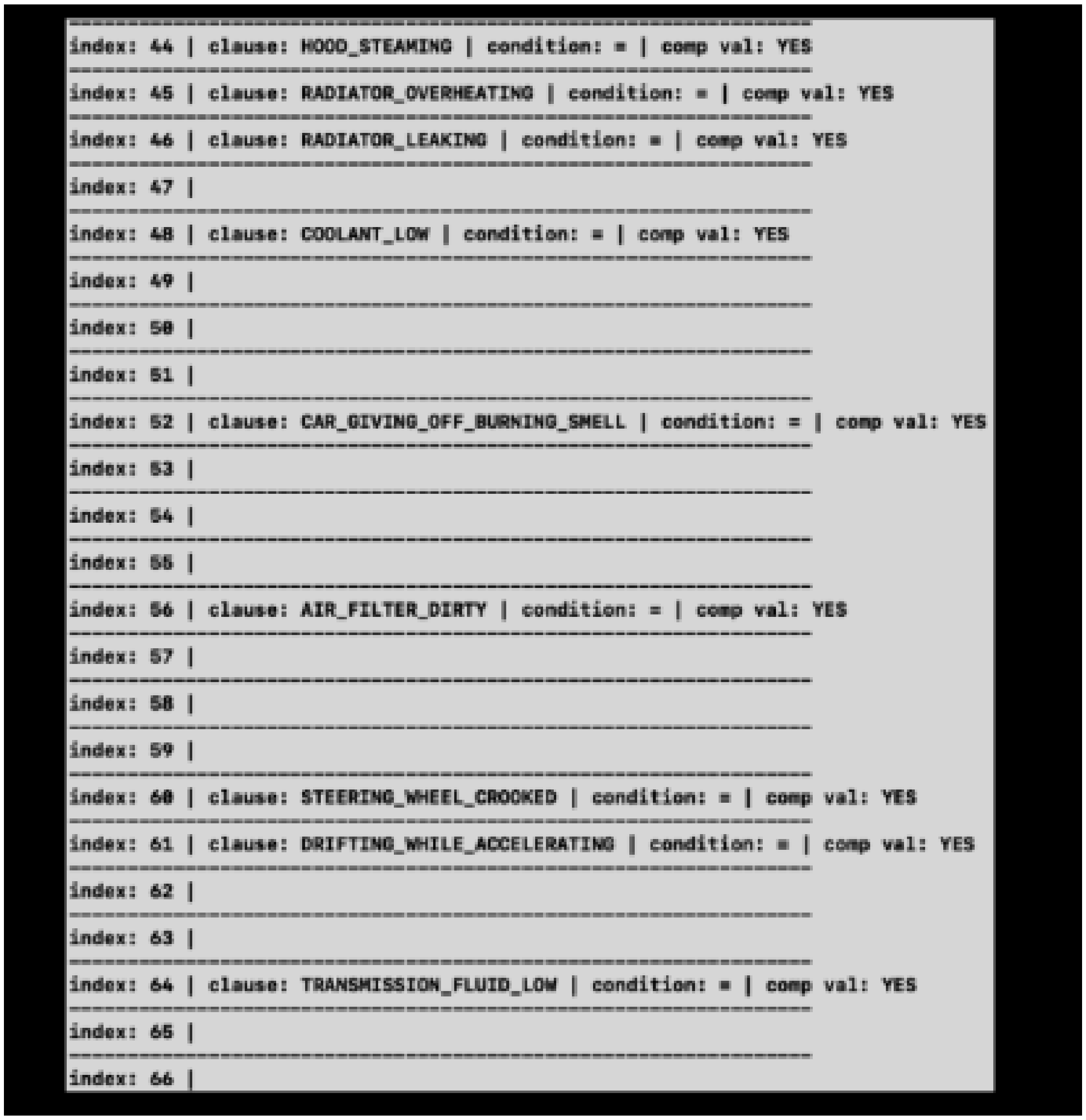
This is the sample run of our following expert system, at the start, all the specific rules will be added. The rule set is then displayed. The order of the display in this order: Clause Variable Info, “If” Var List, “Then” Var List, Conclusion List, and user prompt.



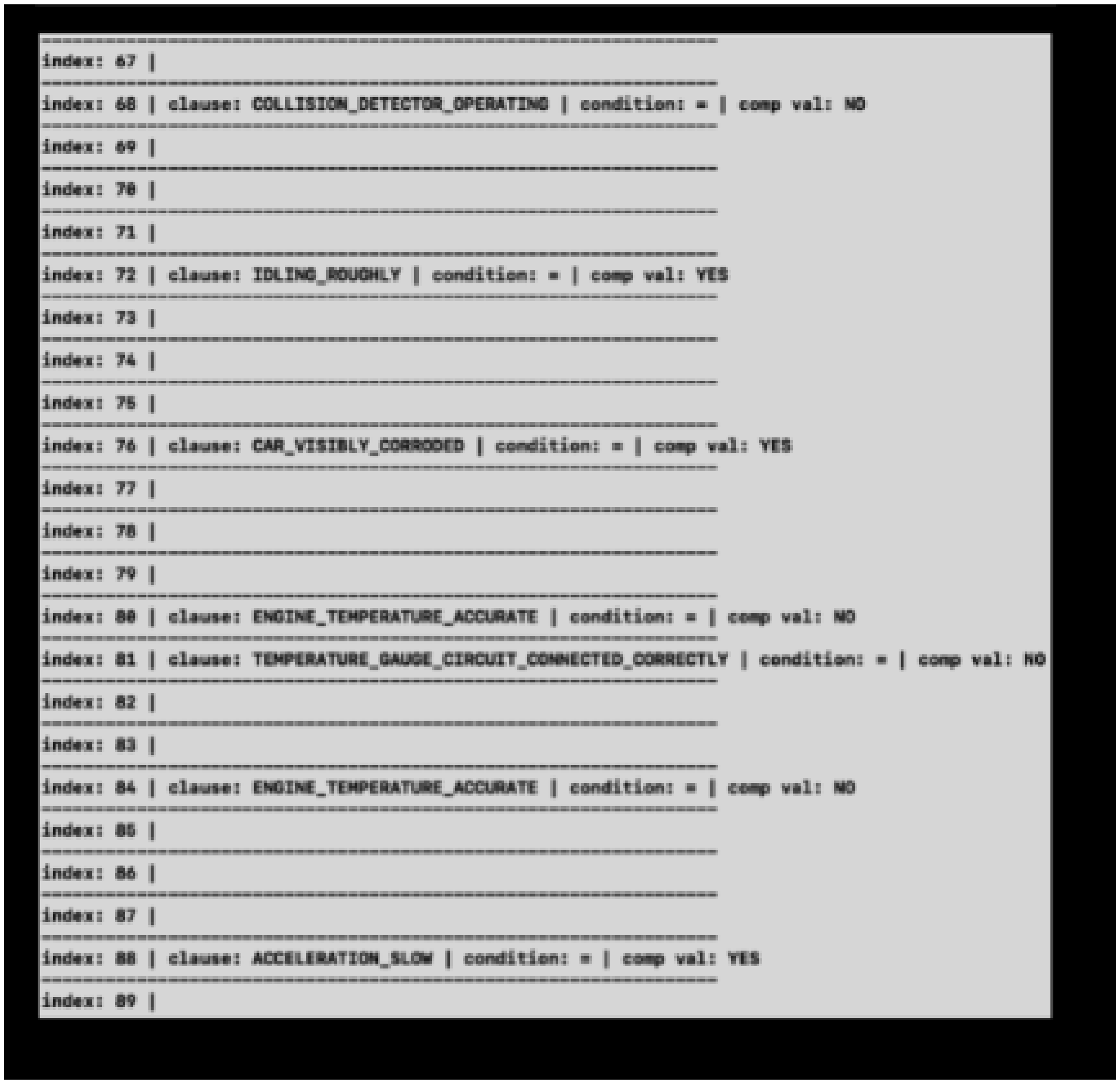
```
CLAUSE VARIABLE INFO LIST
---------------------------
index: 0 | clause: FUEL_FILTER_DIRTY | condition: = | comp val: YES
----------------------------------------------------------------
index: 1 |
----------------------------------------------------------------
index: 2 |
----------------------------------------------------------------
index: 3 |
----------------------------------------------------------------
index: 4 | clause: BULB_WORKING | condition: = | comp val: NO
----------------------------------------------------------------
index: 5 |
----------------------------------------------------------------
index: 6 |
----------------------------------------------------------------
index: 7 |
----------------------------------------------------------------
index: 8 | clause: BULB_WORKING | condition: = | comp val: YES
----------------------------------------------------------------
index: 9 | clause: LIGHT_FUSE_BLOWN | condition: = | comp val: NO
----------------------------------------------------------------
index: 10 |
----------------------------------------------------------------
index: 11 |
----------------------------------------------------------------
index: 12 | clause: BATTERY_CHARGING | condition: = | comp val: NO
----------------------------------------------------------------
index: 13 |
----------------------------------------------------------------
index: 14 |
----------------------------------------------------------------
index: 15 |
----------------------------------------------------------------
index: 16 | clause: CAR_GRINDING_WHEN_TURNING_KEY | condition: = | comp val: YES
----------------------------------------------------------------
index: 17 |
----------------------------------------------------------------
index: 18 |
----------------------------------------------------------------
index: 19 |
----------------------------------------------------------------
index: 20 | clause: BRAKES_WORKING_AT_ALL | condition: = | comp val: NO
----------------------------------------------------------------
```



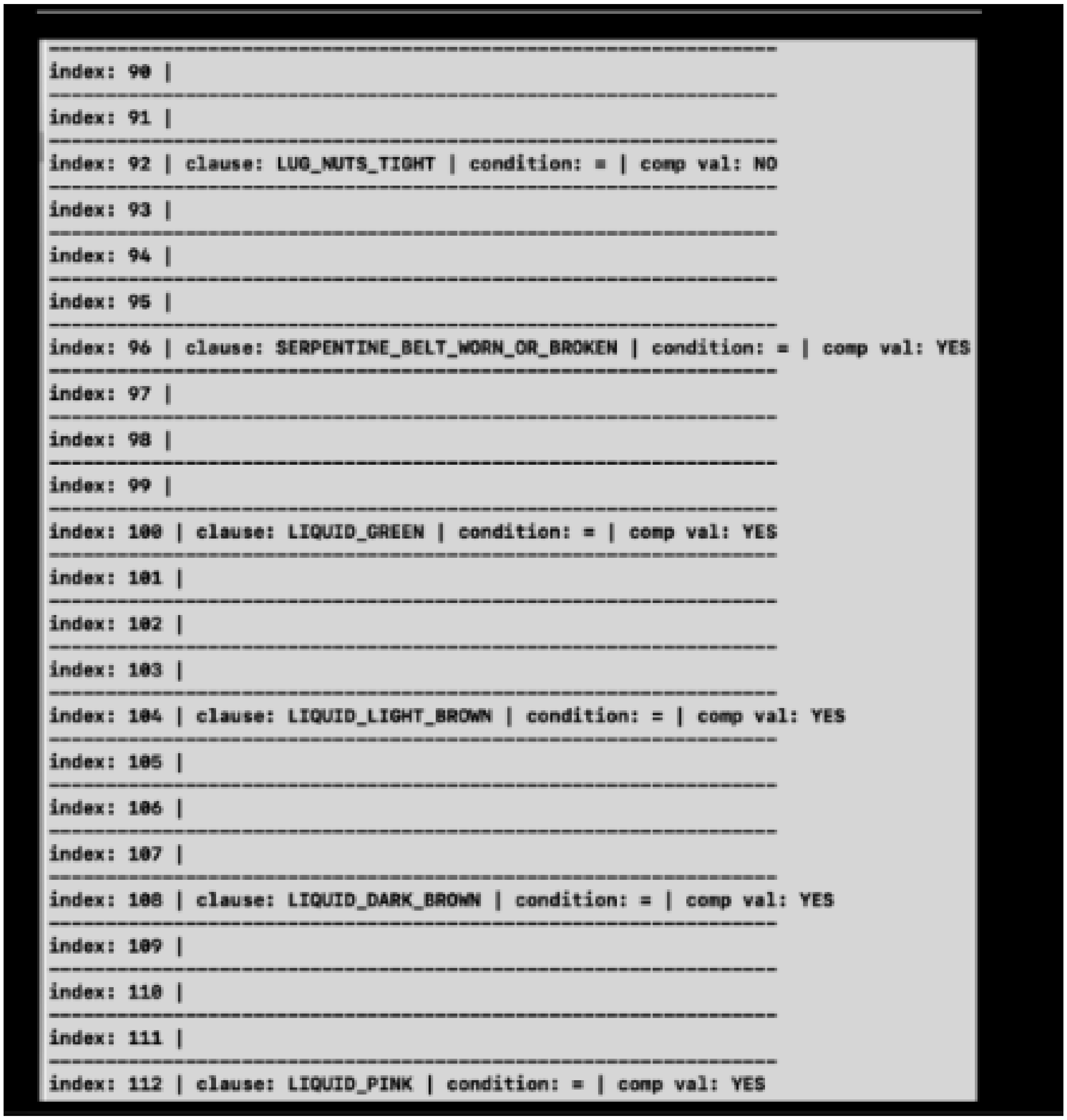
```
----------------------------------------------------------------
index: 21 |
----------------------------------------------------------------
index: 22 |
----------------------------------------------------------------
index: 23 |
----------------------------------------------------------------
index: 24 | clause: CAR_GRINDING_WHEN_BRAKING | condition: = | comp val: YES
----------------------------------------------------------------
index: 25 |
----------------------------------------------------------------
index: 26 |
----------------------------------------------------------------
index: 27 |
----------------------------------------------------------------
index: 28 | clause: DOOR_WIRING_CONNECTED_CORRECTLY | condition: = | comp val: YES
----------------------------------------------------------------
index: 29 | clause: DOOR_FUSE_BLOWN | condition: = | comp val: YES
----------------------------------------------------------------
index: 30 |
----------------------------------------------------------------
index: 31 |
----------------------------------------------------------------
index: 32 | clause: DOOR_WIRING_CONNECTED_CORRECTLY | condition: = | comp val: NO
----------------------------------------------------------------
index: 33 |
----------------------------------------------------------------
index: 34 |
----------------------------------------------------------------
index: 35 |
----------------------------------------------------------------
index: 36 | clause: LIGHTS_DIM | condition: = | comp val: YES
----------------------------------------------------------------
index: 37 |
----------------------------------------------------------------
index: 38 |
----------------------------------------------------------------
index: 39 |
----------------------------------------------------------------
index: 40 | clause: HOOD_STEAMING | condition: = | comp val: YES
----------------------------------------------------------------
index: 41 | clause: RADIATOR_OVERHEATING | condition: = | comp val: YES
----------------------------------------------------------------
index: 42 | clause: RADIATOR_LEAKING | condition: = | comp val: NO
----------------------------------------------------------------
```



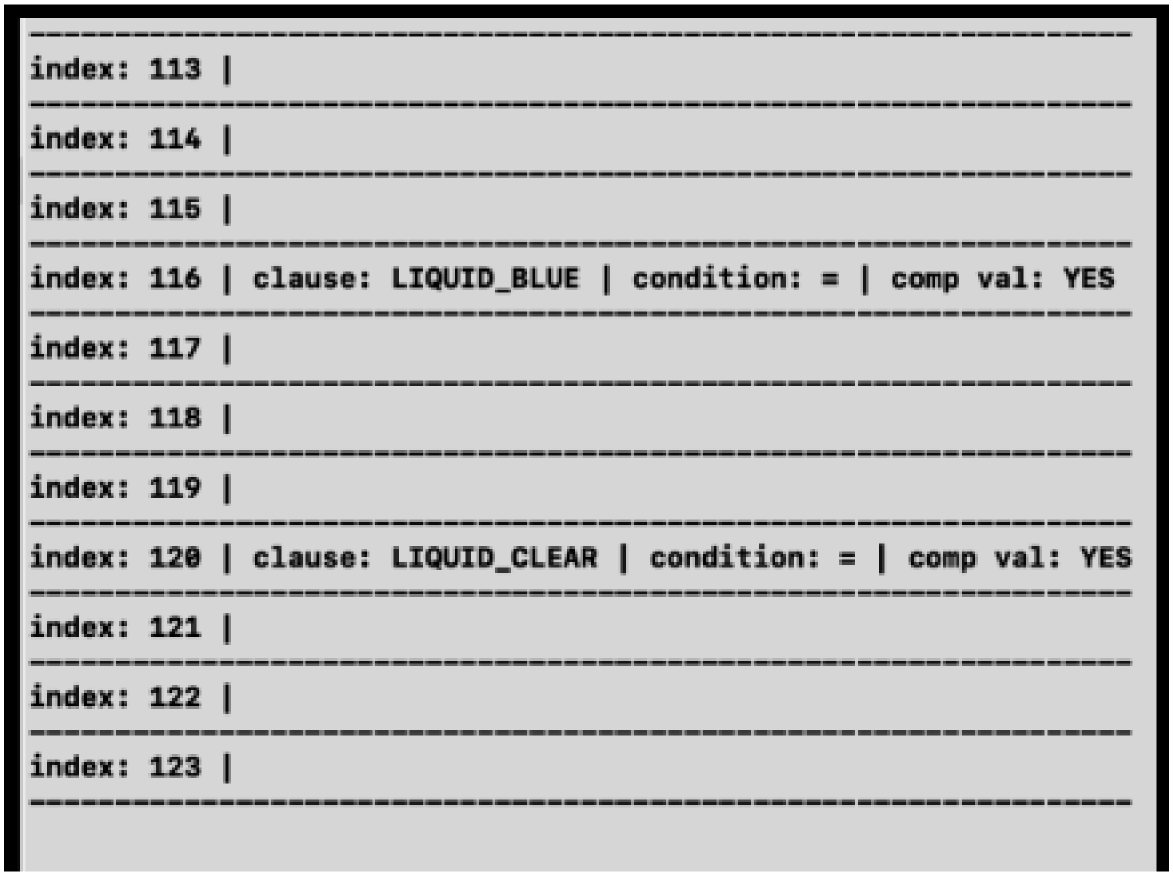
```
----------------------------------------------------------------
index: 44 | clause: HOOD_STEAMING | condition: = | comp val: YES
----------------------------------------------------------------
index: 45 | clause: RADIATOR_OVERHEATING | condition: = | comp val: YES
----------------------------------------------------------------
index: 46 | clause: RADIATOR_LEAKING | condition: = | comp val: YES
----------------------------------------------------------------
index: 47 |
----------------------------------------------------------------
index: 48 | clause: COOLANT_LOW | condition: = | comp val: YES
----------------------------------------------------------------
index: 49 |
----------------------------------------------------------------
index: 50 |
----------------------------------------------------------------
index: 51 |
----------------------------------------------------------------
index: 52 | clause: CAR_GIVING_OFF_BURNING_SMELL | condition: = | comp val: YES
----------------------------------------------------------------
index: 53 |
----------------------------------------------------------------
index: 54 |
----------------------------------------------------------------
index: 55 |
----------------------------------------------------------------
index: 56 | clause: AIR_FILTER_DIRTY | condition: = | comp val: YES
----------------------------------------------------------------
index: 57 |
----------------------------------------------------------------
index: 58 |
----------------------------------------------------------------
index: 59 |
----------------------------------------------------------------
index: 60 | clause: STEERING_WHEEL_CROOKED | condition: = | comp val: YES
----------------------------------------------------------------
index: 61 | clause: DRIFTING_WHILE_ACCELERATING | condition: = | comp val: YES
----------------------------------------------------------------
index: 62 |
----------------------------------------------------------------
index: 63 |
----------------------------------------------------------------
index: 64 | clause: TRANSMISSION_FLUID_LOW | condition: = | comp val: YES
----------------------------------------------------------------
index: 65 |
----------------------------------------------------------------
index: 66 |
```



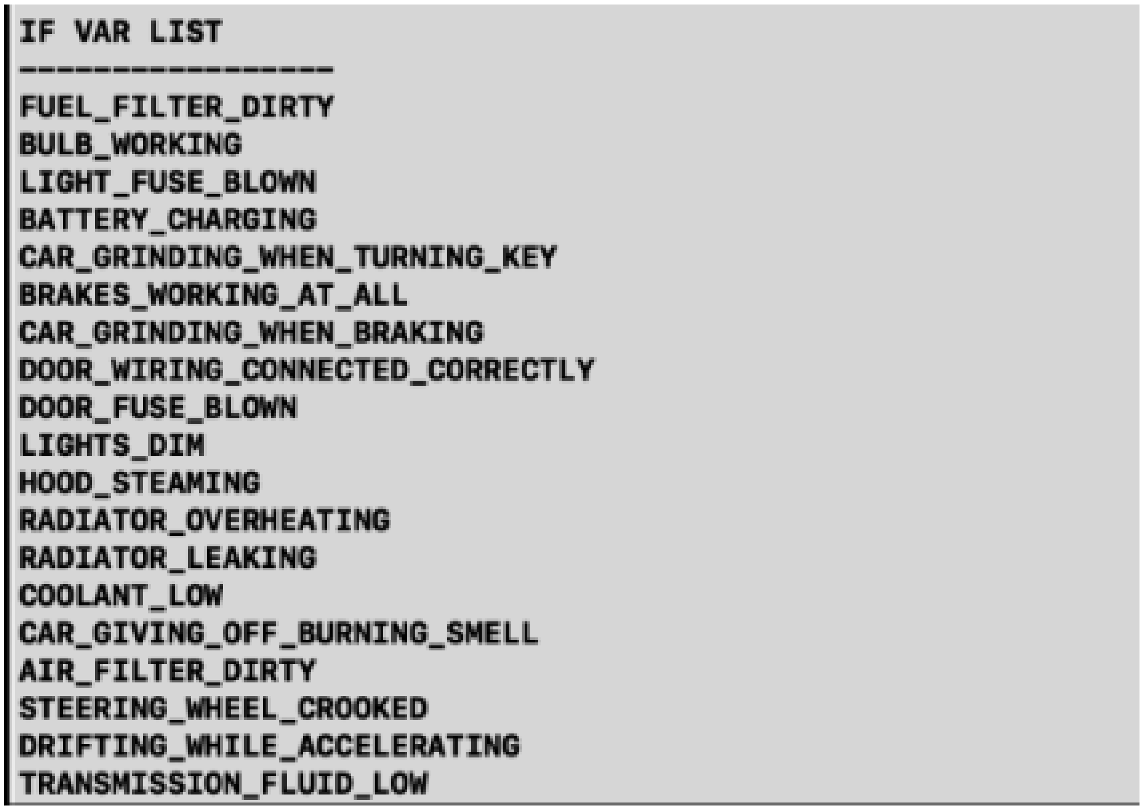
```
---------------------------------------------------------------------
index: 67 |
---------------------------------------------------------------------
index: 68 | clause: COLLISION_DETECTOR_OPERATING | condition: = | comp val: NO
---------------------------------------------------------------------
index: 69 |
---------------------------------------------------------------------
index: 70 |
---------------------------------------------------------------------
index: 71 |
---------------------------------------------------------------------
index: 72 | clause: IDLING_ROUGHLY | condition: = | comp val: YES
---------------------------------------------------------------------
index: 73 |
---------------------------------------------------------------------
index: 74 |
---------------------------------------------------------------------
index: 75 |
---------------------------------------------------------------------
index: 76 | clause: CAR_VISIBLY_CORRODED | condition: = | comp val: YES
---------------------------------------------------------------------
index: 77 |
---------------------------------------------------------------------
index: 78 |
---------------------------------------------------------------------
index: 79 |
---------------------------------------------------------------------
index: 80 | clause: ENGINE_TEMPERATURE_ACCURATE | condition: = | comp val: NO
---------------------------------------------------------------------
index: 81 | clause: TEMPERATURE_GAUGE_CIRCUIT_CONNECTED_CORRECTLY | condition: = | comp val: NO
---------------------------------------------------------------------
index: 82 |
---------------------------------------------------------------------
index: 83 |
---------------------------------------------------------------------
index: 84 | clause: ENGINE_TEMPERATURE_ACCURATE | condition: = | comp val: NO
---------------------------------------------------------------------
index: 85 |
---------------------------------------------------------------------
index: 86 |
---------------------------------------------------------------------
index: 87 |
---------------------------------------------------------------------
index: 88 | clause: ACCELERATION_SLOW | condition: = | comp val: YES
---------------------------------------------------------------------
index: 89 |
```



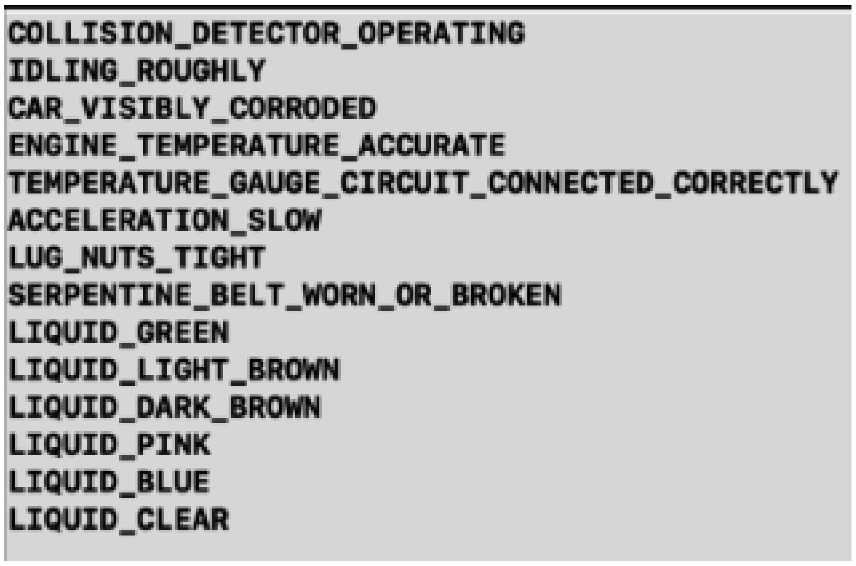
```
---------------------------------------------------------------
index: 90 |
---------------------------------------------------------------
index: 91 |
---------------------------------------------------------------
index: 92 | clause: LUG_NUTS_TIGHT | condition: = | comp val: NO
---------------------------------------------------------------
index: 93 |
---------------------------------------------------------------
index: 94 |
---------------------------------------------------------------
index: 95 |
---------------------------------------------------------------
index: 96 | clause: SERPENTINE_BELT_WORN_OR_BROKEN | condition: = | comp val: YES
---------------------------------------------------------------
index: 97 |
---------------------------------------------------------------
index: 98 |
---------------------------------------------------------------
index: 99 |
---------------------------------------------------------------
index: 100 | clause: LIQUID_GREEN | condition: = | comp val: YES
---------------------------------------------------------------
index: 101 |
---------------------------------------------------------------
index: 102 |
---------------------------------------------------------------
index: 103 |
---------------------------------------------------------------
index: 104 | clause: LIQUID_LIGHT_BROWN | condition: = | comp val: YES
---------------------------------------------------------------
index: 105 |
---------------------------------------------------------------
index: 106 |
---------------------------------------------------------------
index: 107 |
---------------------------------------------------------------
index: 108 | clause: LIQUID_DARK_BROWN | condition: = | comp val: YES
---------------------------------------------------------------
index: 109 |
---------------------------------------------------------------
index: 110 |
---------------------------------------------------------------
index: 111 |
---------------------------------------------------------------
index: 112 | clause: LIQUID_PINK | condition: = | comp val: YES
```



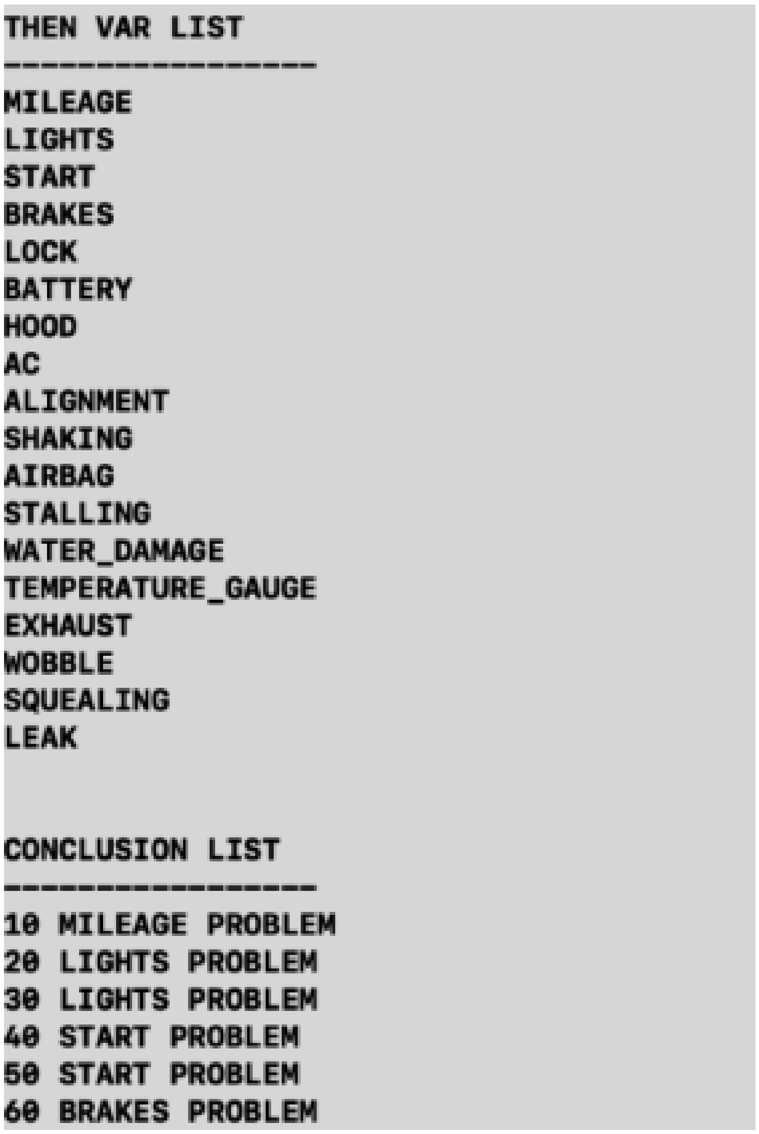
```
---------------------------------------------------------------
index: 113 |
---------------------------------------------------------------
index: 114 |
---------------------------------------------------------------
index: 115 |
---------------------------------------------------------------
index: 116 | clause: LIQUID_BLUE | condition: = | comp val: YES
---------------------------------------------------------------
index: 117 |
---------------------------------------------------------------
index: 118 |
---------------------------------------------------------------
index: 119 |
---------------------------------------------------------------
index: 120 | clause: LIQUID_CLEAR | condition: = | comp val: YES
---------------------------------------------------------------
index: 121 |
---------------------------------------------------------------
index: 122 |
---------------------------------------------------------------
index: 123 |
---------------------------------------------------------------
```



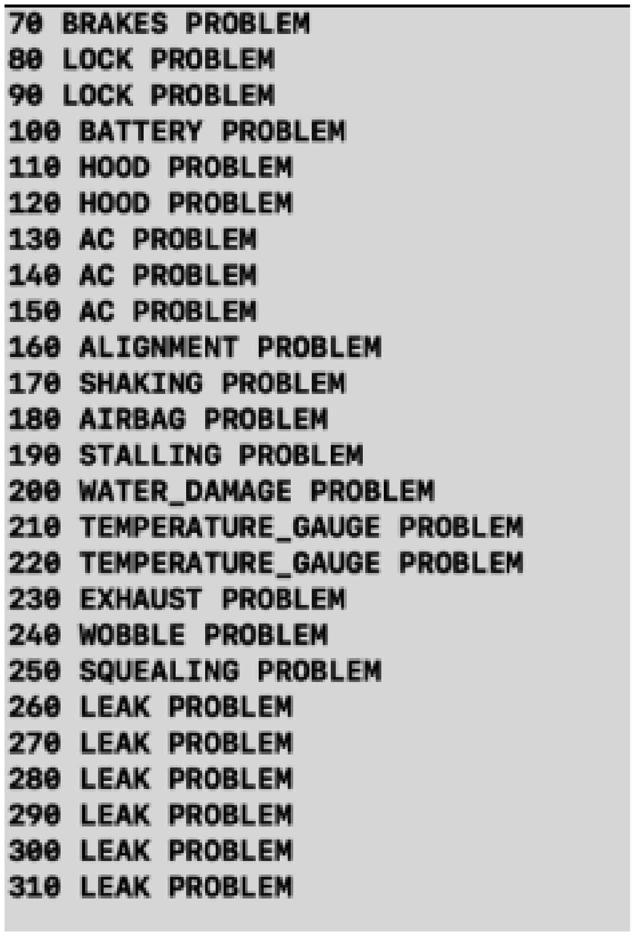
```
IF VAR LIST
-----------------
FUEL_FILTER_DIRTY
BULB_WORKING
LIGHT_FUSE_BLOWN
BATTERY_CHARGING
CAR_GRINDING_WHEN_TURNING_KEY
BRAKES_WORKING_AT_ALL
CAR_GRINDING_WHEN_BRAKING
DOOR_WIRING_CONNECTED_CORRECTLY
DOOR_FUSE_BLOWN
LIGHTS_DIM
HOOD_STEAMING
RADIATOR_OVERHEATING
RADIATOR_LEAKING
COOLANT_LOW
CAR_GIVING_OFF_BURNING_SMELL
AIR_FILTER_DIRTY
STEERING_WHEEL_CROOKED
DRIFTING_WHILE_ACCELERATING
TRANSMISSION_FLUID_LOW
```



```
COLLISION_DETECTOR_OPERATING
IDLING_ROUGHLY
CAR_VISIBLY_CORRODED
ENGINE_TEMPERATURE_ACCURATE
TEMPERATURE_GAUGE_CIRCUIT_CONNECTED_CORRECTLY
ACCELERATION_SLOW
LUG_NUTS_TIGHT
SERPENTINE_BELT_WORN_OR_BROKEN
LIQUID_GREEN
LIQUID_LIGHT_BROWN
LIQUID_DARK_BROWN
LIQUID_PINK
LIQUID_BLUE
LIQUID_CLEAR
```

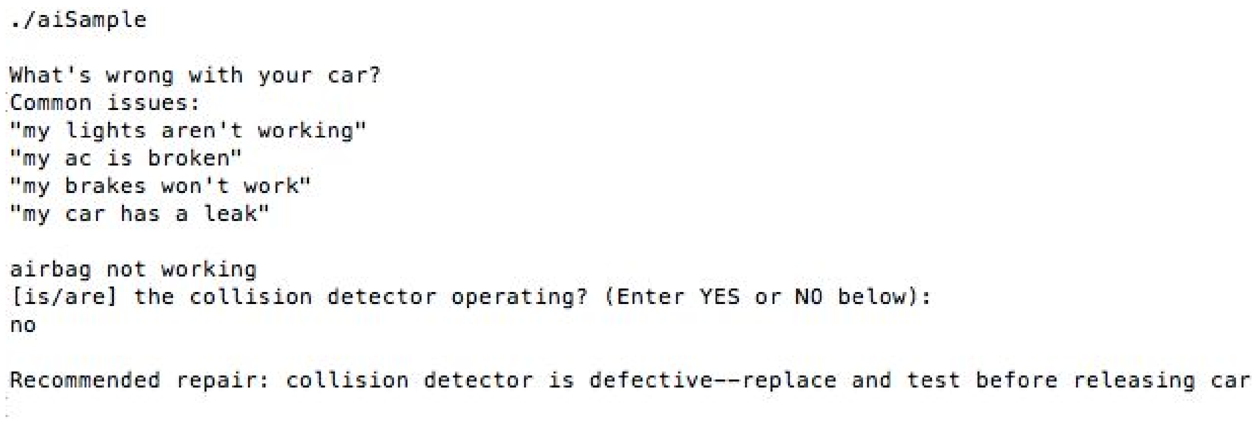


```
THEN VAR LIST
-----------------
MILEAGE
LIGHTS
START
BRAKES
LOCK
BATTERY
HOOD
AC
ALIGNMENT
SHAKING
AIRBAG
STALLING
WATER_DAMAGE
TEMPERATURE_GAUGE
EXHAUST
WOBBLE
SQUEALING
LEAK


CONCLUSION LIST
-----------------
10 MILEAGE PROBLEM
20 LIGHTS PROBLEM
30 LIGHTS PROBLEM
40 START PROBLEM
50 START PROBLEM
60 BRAKES PROBLEM
```



```
70 BRAKES PROBLEM
80 LOCK PROBLEM
90 LOCK PROBLEM
100 BATTERY PROBLEM
110 HOOD PROBLEM
120 HOOD PROBLEM
130 AC PROBLEM
140 AC PROBLEM
150 AC PROBLEM
160 ALIGNMENT PROBLEM
170 SHAKING PROBLEM
180 AIRBAG PROBLEM
190 STALLING PROBLEM
200 WATER_DAMAGE PROBLEM
210 TEMPERATURE_GAUGE PROBLEM
220 TEMPERATURE_GAUGE PROBLEM
230 EXHAUST PROBLEM
240 WOBBLE PROBLEM
250 SQUEALING PROBLEM
260 LEAK PROBLEM
270 LEAK PROBLEM
280 LEAK PROBLEM
290 LEAK PROBLEM
300 LEAK PROBLEM
310 LEAK PROBLEM
```

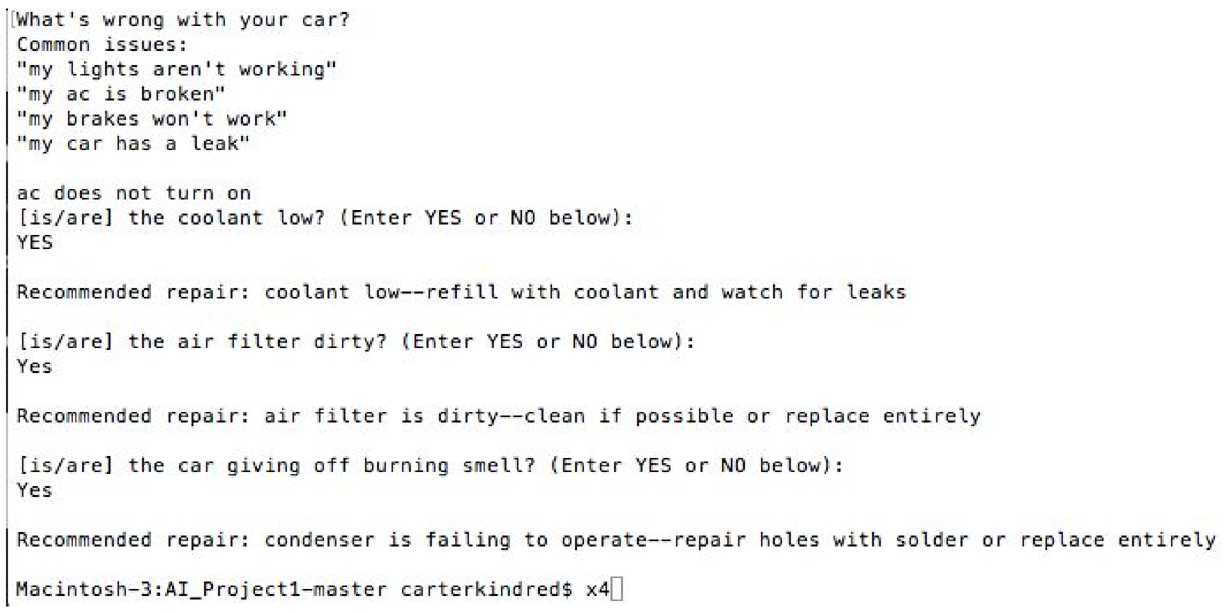
Example outputs(air bag not working input):



```
./aiSample

What's wrong with your car?
Common issues:
"my lights aren't working"
"my ac is broken"
"my brakes won't work"
"my car has a leak"

airbag not working
[is/are] the collision detector operating? (Enter YES or NO below):
no

Recommended repair: collision detector is defective--replace and test before releasing car
```

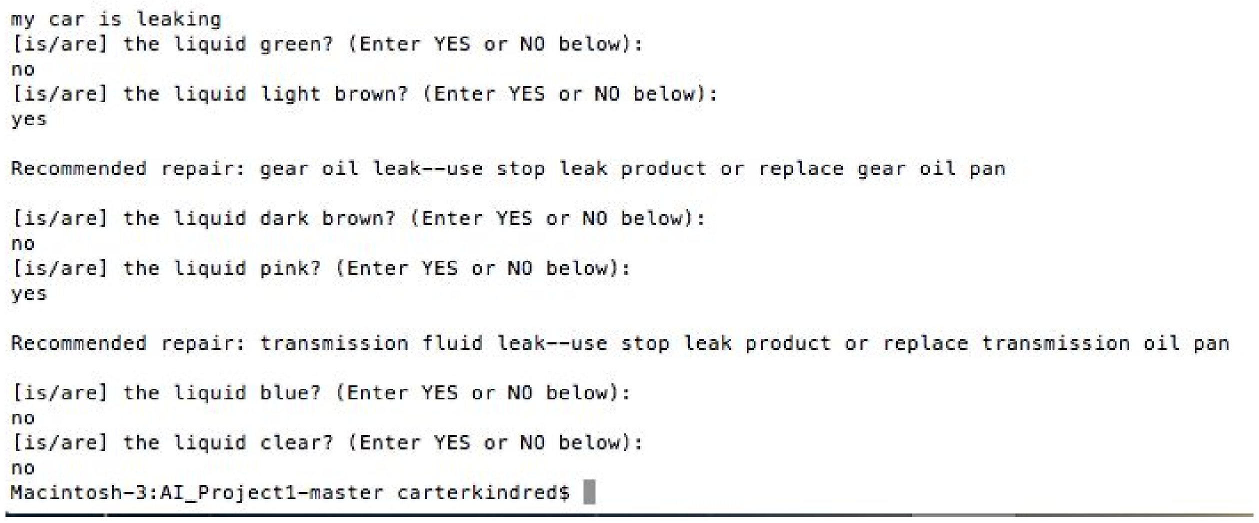
Example output(showing the AC does not turn on):

.



```
What's wrong with your car?
Common issues:
"my lights aren't working"
"my ac is broken"
"my brakes won't work"
"my car has a leak"

ac does not turn on
[is/are] the coolant low? (Enter YES or NO below):
YES

Recommended repair: coolant low--refill with coolant and watch for leaks

[is/are] the air filter dirty? (Enter YES or NO below):
Yes

Recommended repair: air filter is dirty--clean if possible or replace entirely

[is/are] the car giving off burning smell? (Enter YES or NO below):
Yes

Recommended repair: condenser is failing to operate--repair holes with solder or replace entirely

Macintosh-3:AI_Project1-master carterkindred$ x4
```

Example output(my car is leaking output):



```
my car is leaking
[is/are] the liquid green? (Enter YES or NO below):
no
[is/are] the liquid light brown? (Enter YES or NO below):
yes

Recommended repair: gear oil leak--use stop leak product or replace gear oil pan

[is/are] the liquid dark brown? (Enter YES or NO below):
no
[is/are] the liquid pink? (Enter YES or NO below):
yes

Recommended repair: transmission fluid leak--use stop leak product or replace transmission oil pan

[is/are] the liquid blue? (Enter YES or NO below):
no
[is/are] the liquid clear? (Enter YES or NO below):
no
Macintosh-3:AI_Project1-master carterkindred$
```

9. Conclusion

Overall in this project we as a group gained a lot of deep understanding and how to use backwards and forwards chaining, with everything ranging from the algorithms to the reasoning behind what they do and why. For me i personally learning about the forward chaining and its implementation and how the data-driveness of it is completely different as opposed to the goal driven backwards chaining. And our group feels that our design meets all the requirements and is very flexible that can allow for further optimality in the future.

10. References

[1] S. Russell and P. Norvig, Artificial Intelligence: A Modern Approach, 3rd ed. New York City, NY, USA: Prentice-Hall Pearson Education, 2009.

[2] AutoZone, “Troubleshooting,” AutoZone, Accessed September 9, 2019. [Online]. Available: https://www.autozone.com/repairinfo/common/repairInfoMain.jsp?leftNavPage=troubleShooting&targetPage=troubleShooting

[3] M. Rosenthal, “Diagnostic flowchart for a car that won’t start or stalls,” IfItJams, 2016. [Online]. Available: https://www.ifitjams.com/starting.htm

[4] M. Rosenthal, “Diagnosing car engine overheating and antifreeze leaks,” IfItJams, 2008. [Online]. Available: https://www.ifitjams.com/starting.htm

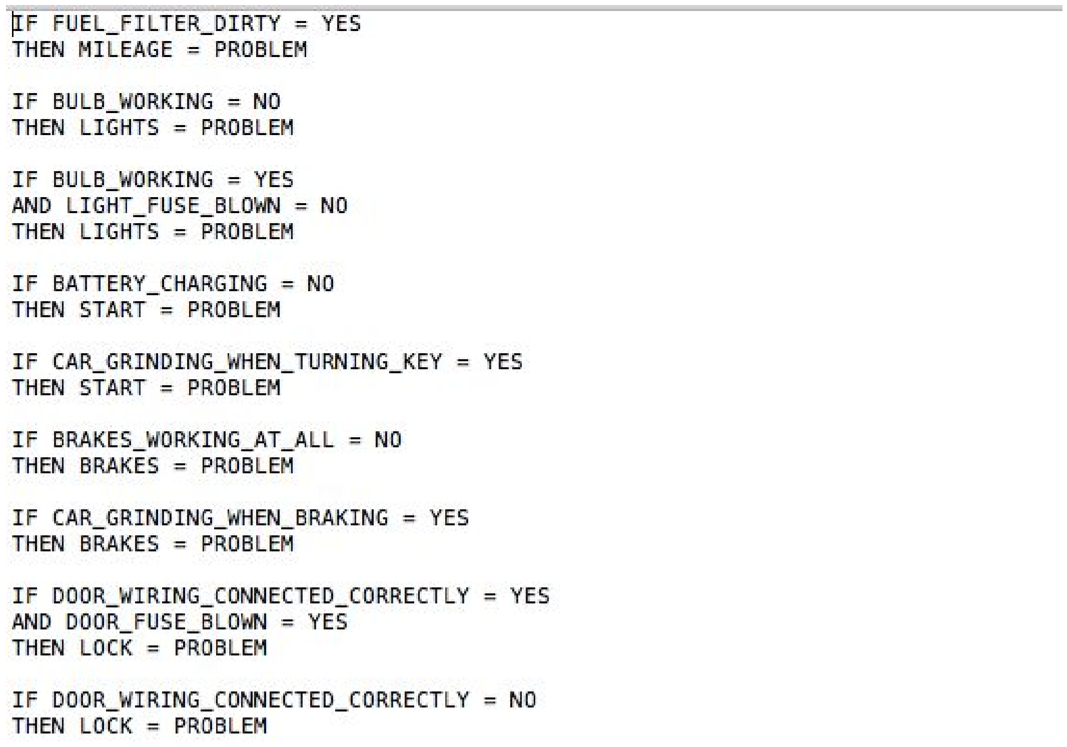
[5] M. Rosenthal, “Troubleshooting brake failure and brake fluid leaks,” IfItJams, 2015. [Online]. Available: https://www.ifitjams.com/starting.htm

11. Project Description

Create an intelligent computer expert system for an auto repair shop. After diagnosing the problem with either forward or backwards chaining, the system should return a correct repair based on the user input. Perform research using Web or any other source to collect knowledge about the symptoms, diagnoses as well as repair recommendations. The auto mechanic will feed the symptoms of the car. Your expert system will diagnose the problem with the car and will recommend repairs. After collecting knowledge, develop a decision tree. Then transform the decision tree into rules. The Diagnosis decision tree should be big enough to generate a minimum of thirty rules. The Repair decision tree should be big enough to generate a rule for each diagnosed problem. Rules should contain variables. Implement the expert system program, employing Backward Chaining and Forward Chaining methodologies. Programs based on these methodologies are provided on TRACS. In order to give you experience in rewriting a better source code, these programs, written in C, are intentionally written poorly and are inefficient and erroneous. Rewrite these programs in C++ by employing Software Engineering principles which prohibits ‘GO TO’ statements and discourage global variable. Separate Knowledge base and Inference Engine parts of each program and bring efficiency in functionality and output using your creativity. Efficiency methods include dynamic memory management, use of objects, and Hashing functions. Though you can totally rewrite the programs, they must be based on the methodologies used in these programs. Using any programs from any other source including web will be treated as plagiarism subject to severe punishment.

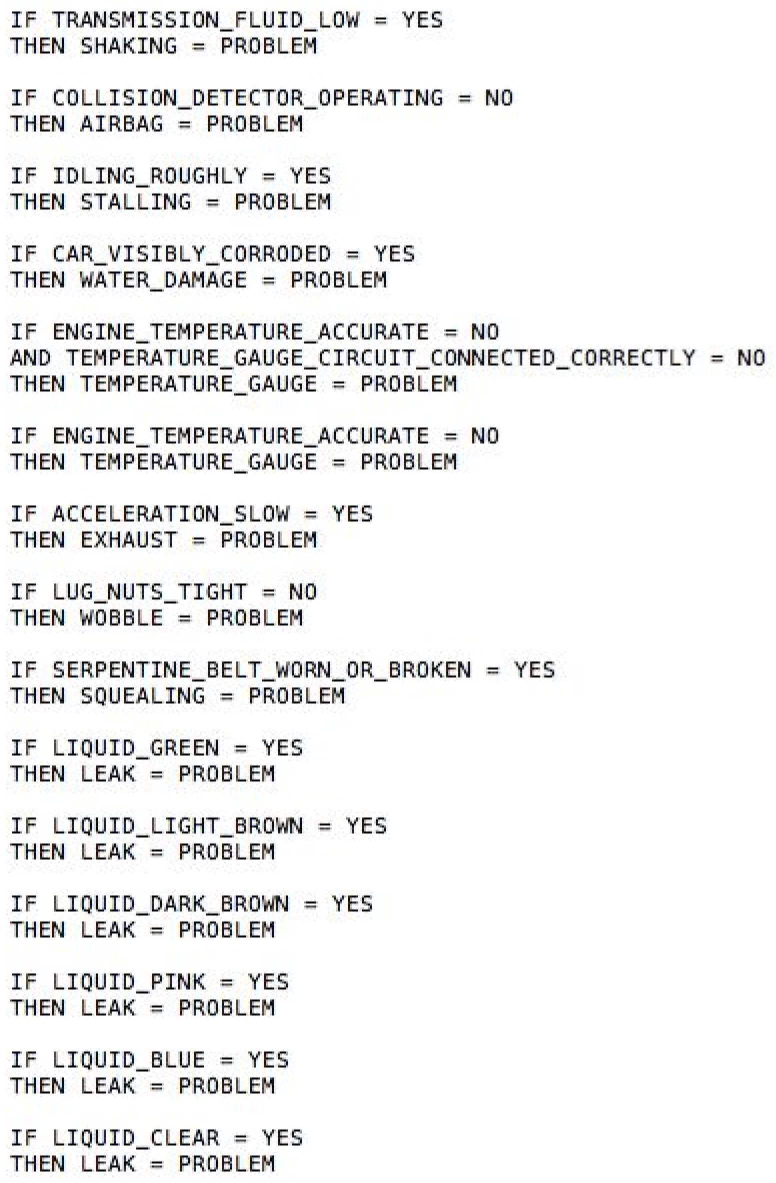
Develop a user-friendly interface, which receives input data from a user in restricted English format, uses keyword matching, and responds in a restricted English format.

11.2 Backward Chaining Rules (screenshots of CarDiagnosis.txt)



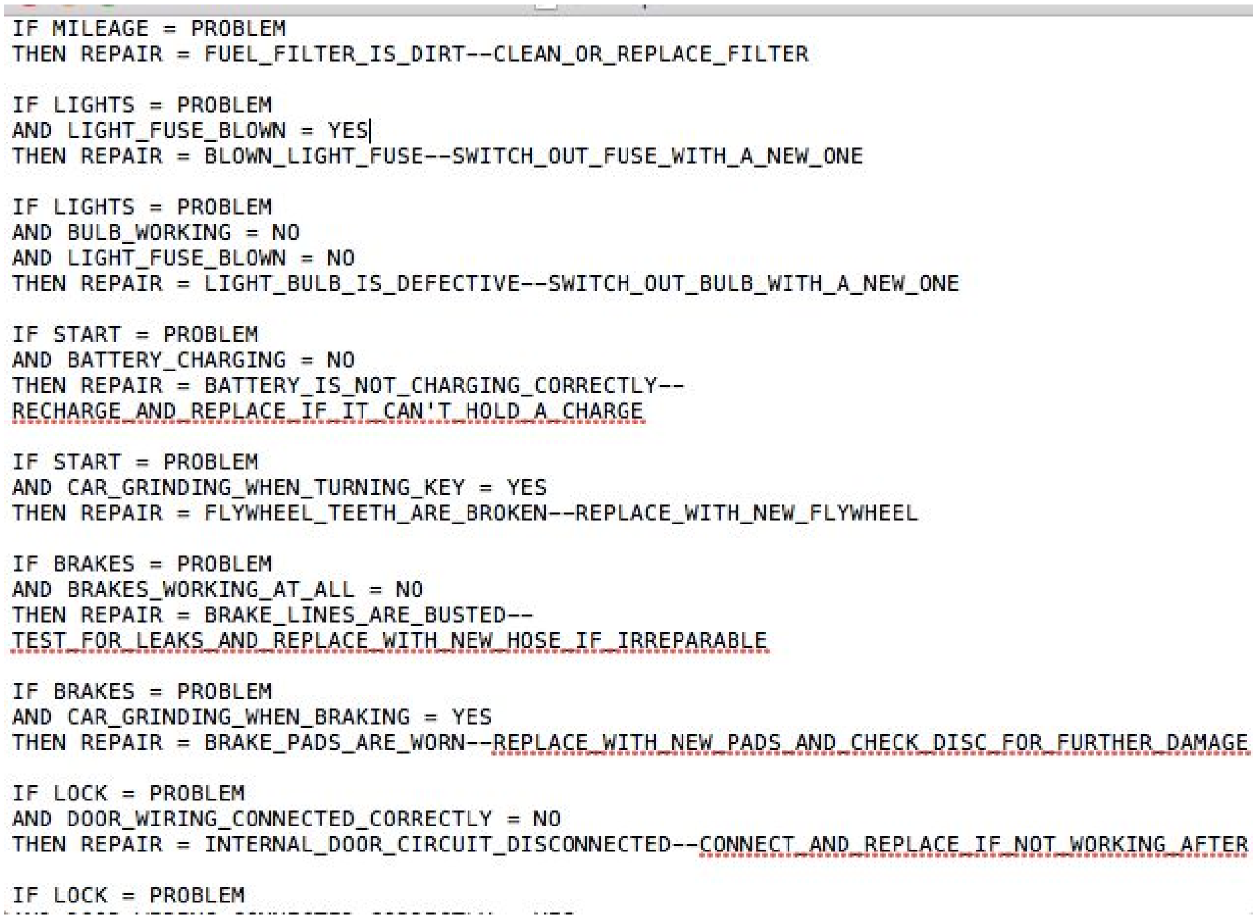
```
IF FUEL_FILTER_DIRTY = YES
THEN MILEAGE = PROBLEM

IF BULB_WORKING = NO
THEN LIGHTS = PROBLEM

IF BULB_WORKING = YES
AND LIGHT_FUSE_BLOWN = NO
THEN LIGHTS = PROBLEM

IF BATTERY_CHARGING = NO
THEN START = PROBLEM

IF CAR_GRINDING_WHEN_TURNING_KEY = YES
THEN START = PROBLEM

IF BRAKES_WORKING_AT_ALL = NO
THEN BRAKES = PROBLEM

IF CAR_GRINDING_WHEN_BRAKING = YES
THEN BRAKES = PROBLEM

IF DOOR_WIRING_CONNECTED_CORRECTLY = YES
AND DOOR_FUSE_BLOWN = YES
THEN LOCK = PROBLEM

IF DOOR_WIRING_CONNECTED_CORRECTLY = NO
THEN LOCK = PROBLEM
```

```
IF DOOR_WIRING_CONNECTED_CORRECTLY = NO
THEN LOCK = PROBLEM

IF LIGHTS_DIM = YES
THEN BATTERY = PROBLEM

IF HOOD_STEAMING = YES
AND RADIATOR_OVERHEATING = YES
AND RADIATOR_LEAKING = NO
THEN HOOD = PROBLEM

IF HOOD_STEAMING = YES
AND RADIATOR_OVERHEATING = YES
AND RADIATOR_LEAKING = YES
THEN HOOD = PROBLEM

IF COOLANT_LOW = YES
THEN AC = PROBLEM

IF CAR_GIVING_OFF_BURNING_SMELL = YES
THEN AC = PROBLEM

IF AIR_FILTER_DIRTY = YES
THEN AC = PROBLEM

IF STEERING_WHEEL_CROOKED = YES
AND DRIFTING_WHILE_ACCELERATING = YES
THEN ALIGNMENT = PROBLEM
```

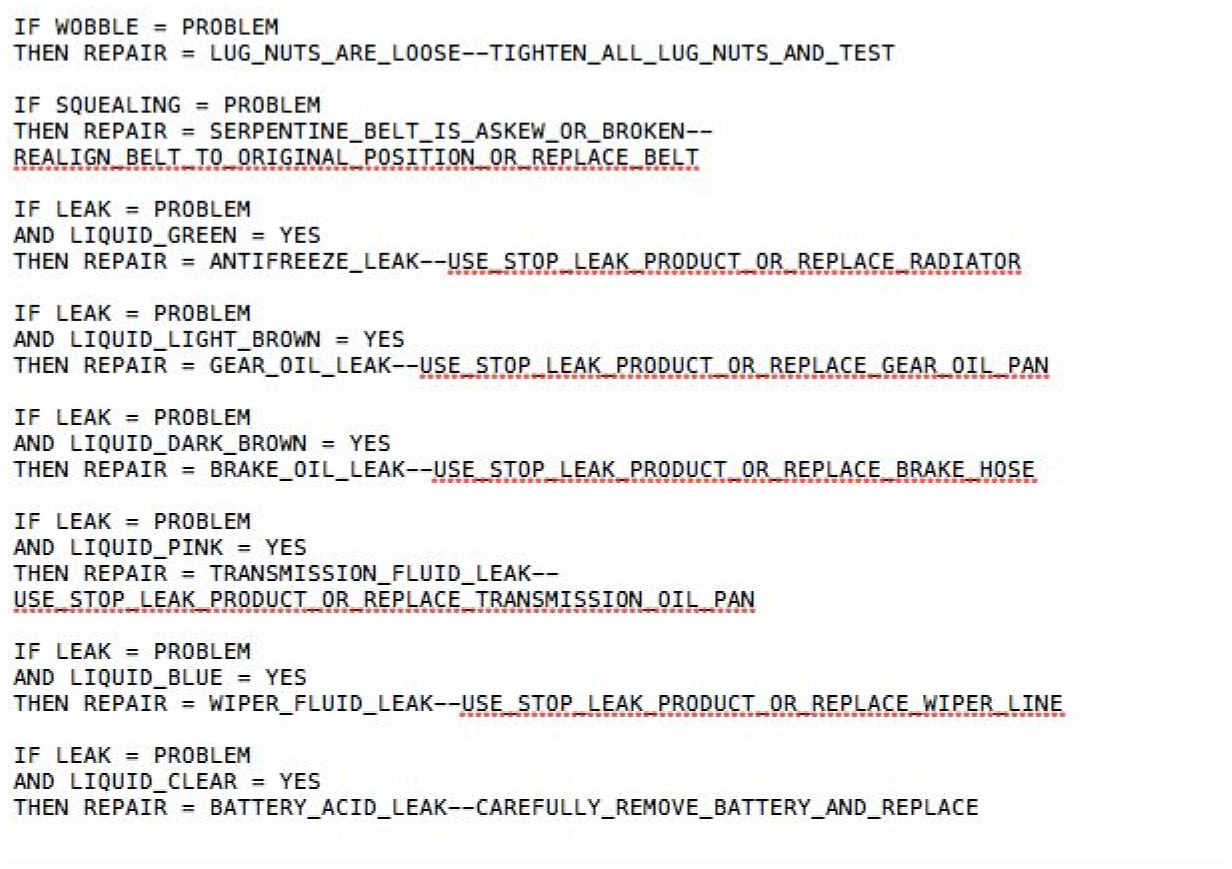


```
IF TRANSMISSION_FLUID_LOW = YES
THEN SHAKING = PROBLEM

IF COLLISION_DETECTOR_OPERATING = NO
THEN AIRBAG = PROBLEM

IF IDLING_ROUGHLY = YES
THEN STALLING = PROBLEM

IF CAR_VISIBLY_CORRODED = YES
THEN WATER_DAMAGE = PROBLEM

IF ENGINE_TEMPERATURE_ACCURATE = NO
AND TEMPERATURE_GAUGE_CIRCUIT_CONNECTED_CORRECTLY = NO
THEN TEMPERATURE_GAUGE = PROBLEM

IF ENGINE_TEMPERATURE_ACCURATE = NO
THEN TEMPERATURE_GAUGE = PROBLEM

IF ACCELERATION_SLOW = YES
THEN EXHAUST = PROBLEM

IF LUG_NUTS_TIGHT = NO
THEN WOBBLE = PROBLEM

IF SERPENTINE_BELT_WORN_OR_BROKEN = YES
THEN SQUEALING = PROBLEM

IF LIQUID_GREEN = YES
THEN LEAK = PROBLEM

IF LIQUID_LIGHT_BROWN = YES
THEN LEAK = PROBLEM

IF LIQUID_DARK_BROWN = YES
THEN LEAK = PROBLEM

IF LIQUID_PINK = YES
THEN LEAK = PROBLEM

IF LIQUID_BLUE = YES
THEN LEAK = PROBLEM

IF LIQUID_CLEAR = YES
THEN LEAK = PROBLEM
```

11.3 Forward Chaining Rules (screenshots with CarRepair.txt)

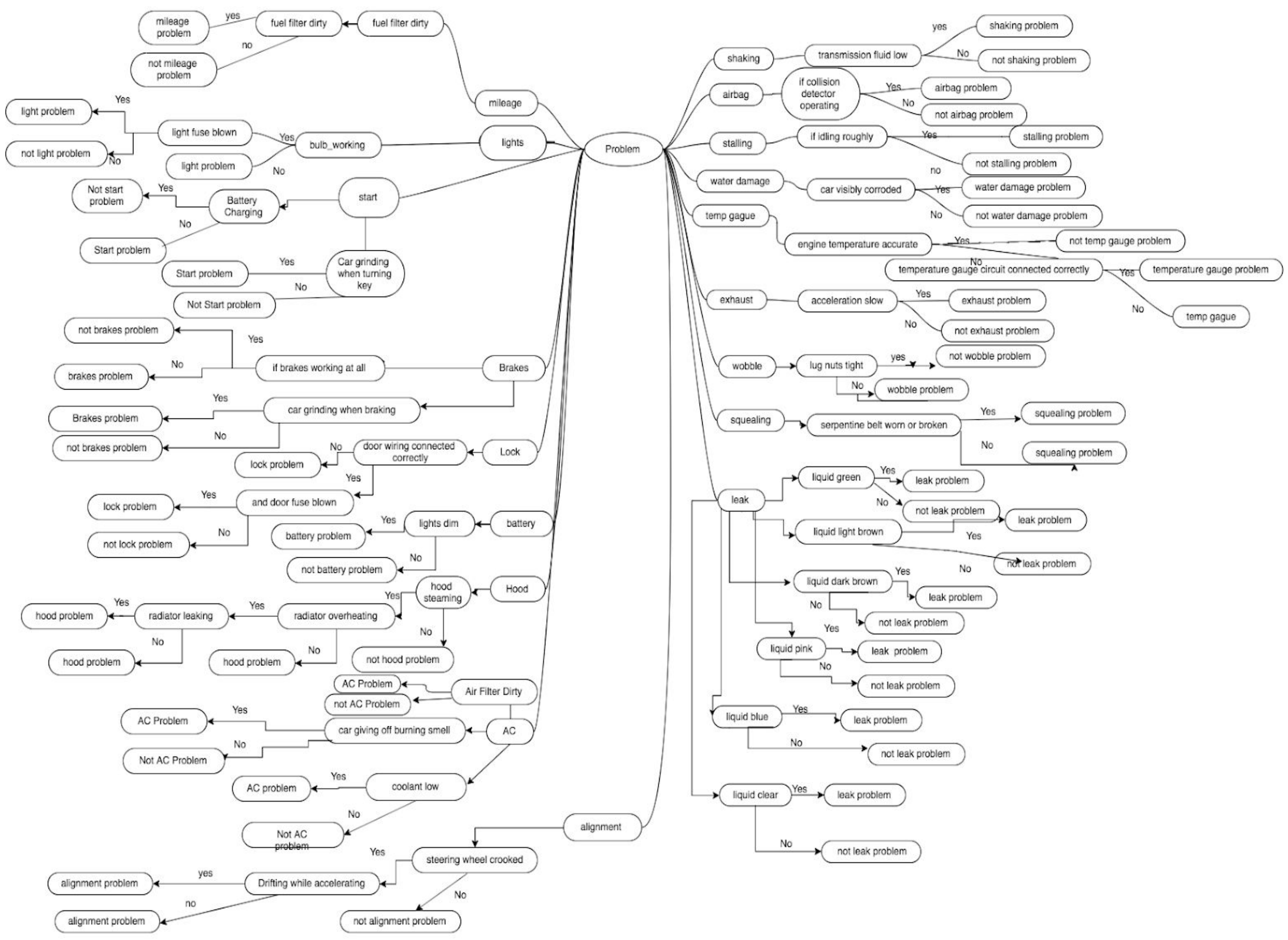


```
IF MILEAGE = PROBLEM
THEN REPAIR = FUEL_FILTER_IS_DIRT--CLEAN_OR_REPLACE_FILTER

IF LIGHTS = PROBLEM
AND LIGHT_FUSE_BLOWN = YES
THEN REPAIR = BLOWN_LIGHT_FUSE--SWITCH_OUT_FUSE_WITH_A_NEW_ONE

IF LIGHTS = PROBLEM
AND BULB_WORKING = NO
AND LIGHT_FUSE_BLOWN = NO
THEN REPAIR = LIGHT_BULB_IS_DEFECTIVE--SWITCH_OUT_BULB_WITH_A_NEW_ONE

IF START = PROBLEM
AND BATTERY_CHARGING = NO
THEN REPAIR = BATTERY_IS_NOT_CHARGING_CORRECTLY--
RECHARGE_AND_REPLACE_IF_IT_CAN'T_HOLD_A_CHARGE

IF START = PROBLEM
AND CAR_GRINDING_WHEN_TURNING_KEY = YES
THEN REPAIR = FLYWHEEL_TEETH_ARE_BROKEN--REPLACE_WITH_NEW_FLYWHEEL

IF BRAKES = PROBLEM
AND BRAKES_WORKING_AT_ALL = NO
THEN REPAIR = BRAKE_LINES_ARE_BUSTED--
TEST_FOR_LEAKS_AND_REPLACE_WITH_NEW_HOSE_IF_IRREPARABLE

IF BRAKES = PROBLEM
AND CAR_GRINDING_WHEN_BRAKING = YES
THEN REPAIR = BRAKE_PADS_ARE_WORN--REPLACE_WITH_NEW_PADS_AND_CHECK_DISC_FOR_FURTHER_DAMAGE

IF LOCK = PROBLEM
AND DOOR_WIRING_CONNECTED_CORRECTLY = NO
THEN REPAIR = INTERNAL_DOOR_CIRCUIT_DISCONNECTED--CONNECT_AND_REPLACE_IF_NOT_WORKING_AFTER

IF LOCK = PROBLEM
IF LOCK = PROBLEM
AND DOOR_WIRING_CONNECTED_CORRECTLY = YES
THEN REPAIR = BLOWN_DOOR_FUSE--SWITCH_OUT_FUSE_WITH_A_NEW_ONE

IF BATTERY = PROBLEM
AND LIGHTS_DIM = YES
THEN REPAIR = ALTERNATOR_IS_NOT_WORKING_CORRECTLY--
FLASH_WITH_POWER_SUPPLY_AND_REPLACE_IF_PROBLEM_PERSISTS

IF HOOD = PROBLEM
AND RADIATOR_OVERHEATING = YES
AND RADIATOR_LEAKING = NO
THEN REPAIR = RADIATOR_IS_NOT_COOLING_CORRECTLY--REPLACE_RADIATOR_ENTIRELY

IF HOOD = PROBLEM
AND RADIATOR_LEAKING = YES
THEN REPAIR = ANTIFREEZE_LEAK--USE_STOP_LEAK_PRODUCT_OR_REPLACE_RADIATOR

IF AC = PROBLEM
AND COOLANT_LOW = YES
THEN REPAIR = COOLANT_LOW--REFILL_WITH_COOLANT_AND_WATCH_FOR_LEAKS

IF AC = PROBLEM
AND AIR_FILTER_DIRTY = YES
THEN REPAIR = AIR_FILTER_IS_DIRTY--CLEAN_IF_POSSIBLE_OR_REPLACE_ENTIRELY

IF AC = PROBLEM
AND CAR_GIVING_OFF_BURNING_SMELL = YES
THEN REPAIR = CONDENSER_IS_FAILING_TO_OPERATE--
REPAIR_HOLES_WITH_SOLDER_OR_REPLACE_ENTIRELY

IF ALIGNMENT = PROBLEM
AND STEERING_WHEEL_CROOKED = YES
DRIFTING_WHILE_ACCELERATING = YES
THEN REPAIR = TIRES_ARE_OUT_OF_ALIGNMENT--REALIGN_TIRES_TO_PROPER_POSITION

IF SHAKING = PROBLEM
THEN REPAIR = TRANSMISSION_FLUID_IS_LOW--REFILL_FLUID_AND_CHECK_FOR_LEAKS

IF AIRBAG = PROBLEM
THEN REPAIR = COLLISION_DETECTOR_IS_DEFECTIVE--REPLACE_AND_TEST_BEFORE_RELEASING_CAR

IF STALLING = PROBLEM
THEN REPAIR = EGR_VALVE_IS_DEFECTIVE_OR_DIRTY--CLEAN_OR_REPLACE_IF_PROBLEM_PERSISTS

IF WATER_DAMAGE = PROBLEM
AND CAR_VISIBLY_CORRODED = YES
THEN REPAIR = THERE_ARE_HOLES_IN_THE_RUSTED_METAL--REPAIR_IF_POSSIBLE_OR_REPLACE_PART

IF TEMPERATURE_GAUGE = PROBLEM
AND TEMPERATURE_GAUGE_CIRCUIT_CONNECTED_CORRECTLY = NO
THEN REPAIR = COOLANT_IS_PROBABLY_LOW--REFILL_AND_REPLACE_GAUGE_IF_PROBLEM_NOT_FIXED

IF TEMPERATURE_GAUGE = PROBLEM
AND TEMPERATURE_GAUGE_CIRCUIT_CONNECTED_CORRECTLY = YES
THEN REPAIR = TEMP_GAUGE_CIRCUIT_IS_NOT_CONNECTED_CORRECTLY--RECONNECT_CIRCUIT_OR_REWIRE

IF EXHAUST = PROBLEM
THEN REPAIR = TURBOCHARGER_IS_NOT_WORKING_PROPERLY--
CLEAN_TURBOCHARGER_AND_REPLACE_AIR_FILTER
```

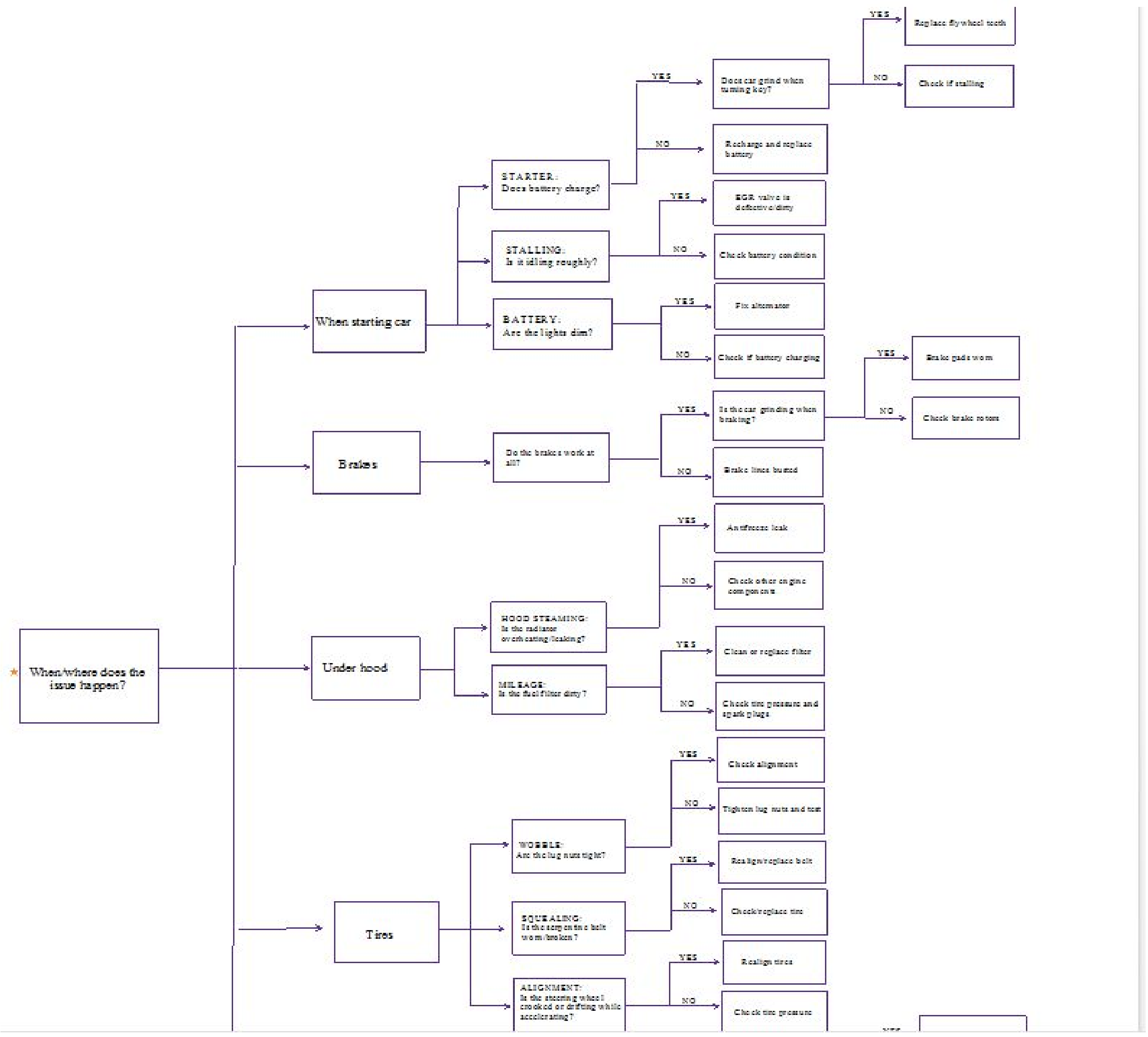


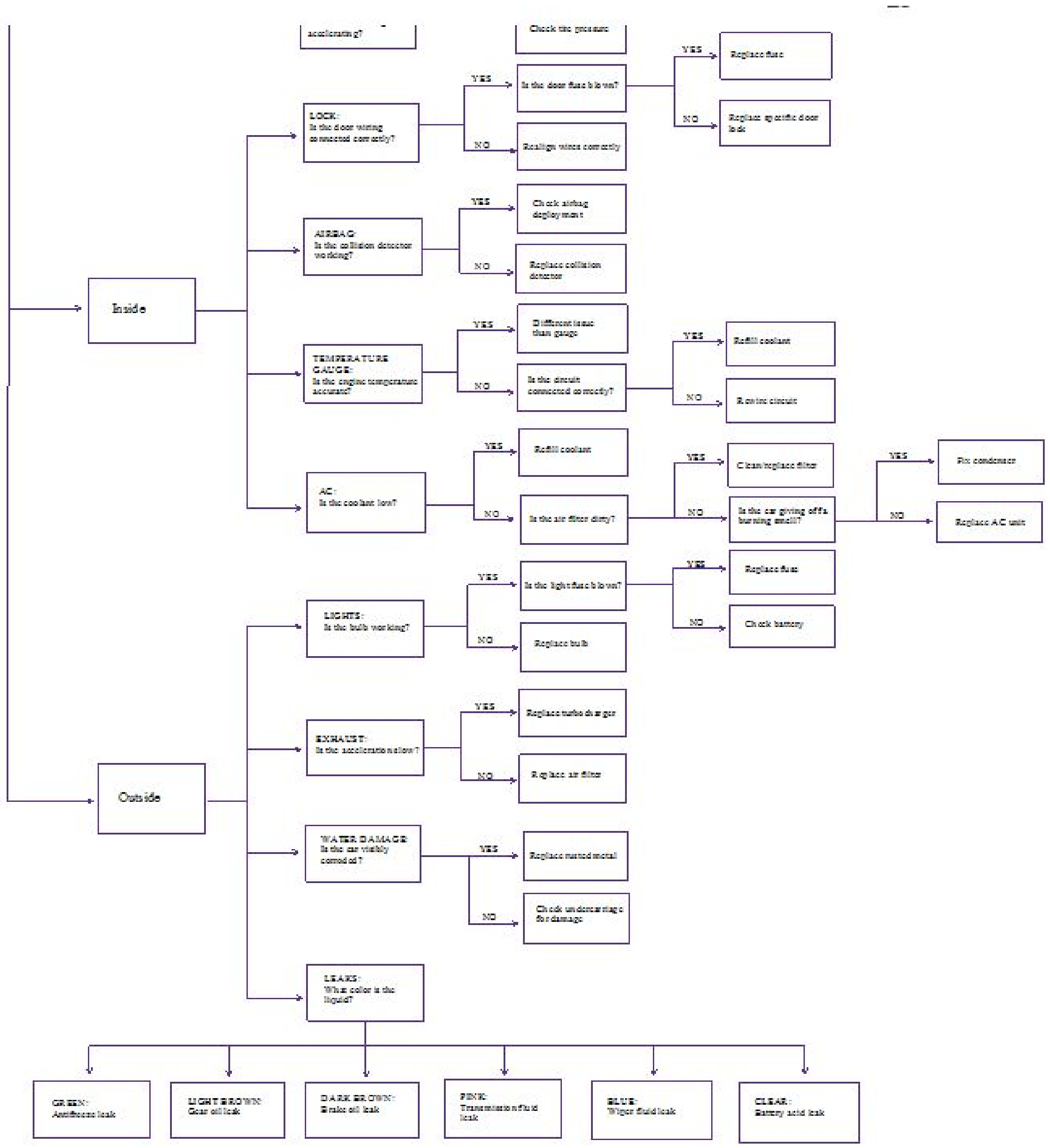
```
IF WOBBLE = PROBLEM
THEN REPAIR = LUG_NUTS_ARE_LOOSE--TIGHTEN_ALL_LUG_NUTS_AND_TEST

IF SQUEALING = PROBLEM
THEN REPAIR = SERPENTINE_BELT_IS_ASKEW_OR_BROKEN--
REALIGN_BELT_TO_ORIGINAL_POSITION_OR_REPLACE_BELT

IF LEAK = PROBLEM
AND LIQUID_GREEN = YES
THEN REPAIR = ANTIFREEZE_LEAK--USE_STOP_LEAK_PRODUCT_OR_REPLACE_RADIATOR

IF LEAK = PROBLEM
AND LIQUID_LIGHT_BROWN = YES
THEN REPAIR = GEAR_OIL_LEAK--USE_STOP_LEAK_PRODUCT_OR_REPLACE_GEAR_OIL_PAN

IF LEAK = PROBLEM
AND LIQUID_DARK_BROWN = YES
THEN REPAIR = BRAKE_OIL_LEAK--USE_STOP_LEAK_PRODUCT_OR_REPLACE_BRAKE_HOSE

IF LEAK = PROBLEM
AND LIQUID_PINK = YES
THEN REPAIR = TRANSMISSION_FLUID_LEAK--
USE_STOP_LEAK_PRODUCT_OR_REPLACE_TRANSMISSION_OIL_PAN

IF LEAK = PROBLEM
AND LIQUID_BLUE = YES
THEN REPAIR = WIPER_FLUID_LEAK--USE_STOP_LEAK_PRODUCT_OR_REPLACE_WIPER_LINE

IF LEAK = PROBLEM
AND LIQUID_CLEAR = YES
THEN REPAIR = BATTERY_ACID_LEAK--CAREFULLY_REMOVE_BATTERY_AND_REPLACE
```

11.4 Backward Chaining Decision Tree



11. 5 Forward Chaining Decision Tree


When/where does the issue happen?
When starting car
STARTER: Does battery charge?
YES
Does car grind when turning key?
YES
Replace flywheel teeth
NO
Check if stalling
NO
Recharge and replace battery
STALLING: Is it idling roughly?
YES
EGR valve is defective/dirty
NO
Check battery condition
BATTERY: Are the lights dim?
YES
Fix alternator
NO
Check if battery charging
Brakes
Do the brakes work at all?
YES
Is the car grinding when braking?
YES
Brake pads worn
NO
Check brake rotors
NO
Brake lines busted
Under hood
HOOD STEAMING: Is the radiator overheating/leaking?
YES
Antifreeze leak
NO
Check other engine components
MILEAGE: Is the fuel filter dirty?
YES
Clean or replace filter
NO
Check tire pressure and spark plugs
Tires
WOBBLE: Are the lug nuts tight?
YES
Check alignment
NO
Tighten lug nuts and test
SQUEALING: Is the serpentine belt worn/broken?
YES
Realign/replace belt
NO
Check/replace tire
ALIGNMENT: Is the steering wheel crooked or drifting while accelerating?
YES
Realign tires
NO
Check tire pressure



12. Source Code

```
#include <iostream>

#include <string>
```

```
#include <vector>

#include <deque>

#include <fstream>

#include <algorithm>

#include "clausevarinfo.h"

#include "rulevarinfo.h"

using namespace std;



bool recommendRepair(RuleVarInfo&, ClauseVarInfo&, string&, deque <string>&, pair
<int, int> &, const int&);

void diagnoseProblem(RuleVarInfo&, ClauseVarInfo&, string&, deque
<pair<string,string>*>, const int&);

inline pair<int, pair<int, string>> getVarInfo(RuleVarInfo&, string&);

inline int getClauseIndex(int&, const int&);

inline string searchInputForConclusion(string, RuleVarInfo&);

inline string splitString(string);

inline string toLower(string);

inline void toUpper(string&);

inline void print(ClauseVarInfo&, RuleVarInfo&);

inline void extractRules(vector <string>&, string);

inline void fillDataStructs(vector <string>&, ClauseVarInfo&, RuleVarInfo&, const int&);




int main() {
    const int NUM_OF_CLAUSES = 4; // Highest number of if clauses in rule base

    vector<string> ruleString; // Contains extracted data from text file rules

    ClauseVarInfo cvi; // Contains all info regarding clause variables

    RuleVarInfo rvi; // Contains all info regarding rules

    deque <string> cq; // Conclusion queue

    deque <pair<string,string>*> cs; // Conclusion stack

    pair<int, int> cvp; // Clause variable pointer


    extractRules(ruleString, "backward");

    fillDataStructs(ruleString, cvi, rvi, NUM_OF_CLAUSES);

    // print(cvi, rvi); // uncomment to print out contents of data structures


    string userString;

    string searchTerm;

    string termValue = "problem";

    toUpper(termValue);


    cout << "\nWhat's wrong with your car?\n"

        << "Common issues:\n\"my lights aren't working\"\n\"my ac is broken\""

        << "\n\"my brakes won't work\"\n\"my car has a leak\"\n\n";

    getline(cin, userString);


/* Search the input for clause variable */
```

```
        toUpper(userString);

        searchTerm = searchInputForConclusion(userString, rvi);

        toUpper(searchTerm);


        diagnoseProblem(rvi, cvi, searchTerm, cs, NUM_OF_CLAUSES);

    /* Empty string vector and fill with forward chaining rules */

        ruleString.clear();

        extractRules(ruleString, "forward");

        fillDataStructs(ruleString, cvi, rvi, NUM_OF_CLAUSES);


        // If no repair rule satisfied

        if(!recommendRepair(rvi, cvi, searchTerm, cq, cvp, NUM_OF_CLAUSES))

            cout << "\nSorry! No repair could be found in our database for the provided
information\n";







    }



    /* Uses the backward chaining AI method to ask for answers to IF clauses until a
conclusion has been reached

     *

     * rvi - Shared info of if and then variable names and their values
```

```
        * cvi - Shared info of clause variable names and their values

        * searchTerm - Term selected from user input to detect initial conclusion

        * cs - the conclusion stack

        * NUM_OF_CLAUSES - maximum number of claus variables for each rule

        *

        */

        void diagnoseProblem(RuleVarInfo &rvi, ClauseVarInfo &cvi, string &searchTerm, deque
<pair<string,string>*> cs, const int &NUM_OF_CLAUSES){

          int idx = 0; // Keeps track of index in range-based loops

          int clauseIdx = 0; // First clause related to current rule

          bool contains = false; // True - Element found ; False - Element not found


        /*Check if value from user is inside of a rule and add to stack if found*/

          for(auto &p : rvi.cl) {

            if (searchTerm == p.first) {

              clauseIdx = getClauseIndex(idx, NUM_OF_CLAUSES);

              cs.emplace_front(&p);

              contains = true;

              break;

            } ++idx;

          } if(!contains){

            cout << "I could not find this conclusion. Try again.";

            exit(1);

          }
          string clauseName = cvi.cvl[clauseIdx]; // Index for current rule clause variables in cvl

          int stackPtr = 0; // Index for current element in stack being processed


          idx = 0;

          pair<int, pair<int, string>> varInfo; // Contains data for current clause var in relation to
var lists

          int varIdx; // Index for var in the list it was found in

          bool instantiated;   // True - not instantiated; False - instantiated

          string list; // IF - in IF var list; THEN - in THEN var list

          string *varValue; // Points to value for given variable found


          // Clause comparison strings and values

          string clauseValue;

          string condition;

          string compareValue;

          string answer;

          string numMessage = "(Enter numeric value)"; // If value entered by user will be a
string

          string strMessage = "(YES/NO)"; // If value entered by user will be a float

          int currRule; // Current rule number


        /*Search lists for containing clause var and collect data when found*/

          while(stackPtr > -1) {
            for (int j = 0; j < NUM_OF_CLAUSES; ++j) {


                clauseName = cvi.cvl[clauseIdx + j];


                if (clauseName.empty() || j ==3) break; // No more clauses to process


                varInfo = getVarInfo(rvi, clauseName);

                varInfo.first == 0 ? list = "IF" : list = "THEN";

                !varInfo.second.second.empty() ? instantiated = true : instantiated = false;

                varIdx = varInfo.second.first;

                varValue = &rvi.ivl[varIdx].second;


        /*Test for list and NI status*/

                if (varInfo.first != -1) {

                    if (list == "IF" && !instantiated) {

                        cout << "[is/are] the " << toLower(splitString(clauseName)) << "? (Enter
YES or NO below): \n";

                        cin >> *varValue;

                        toUpper(*varValue);

                    } else if (list == "THEN" && !instantiated) {


                        /*Find the rule for clause variable in conclusion list and add to the stack*/

                        idx = 0;
                        for (auto &p : rvi.cl) {
                            if (clauseName == p.first) {
                                clauseIdx = getClauseIndex(idx, NUM_OF_CLAUSES);
                                ++stackPtr;
                                cs.emplace_back(&p);
                                j = -1;
                                break;
                            } ++idx;
                        }
                    }
                } else exit(3); // Var not found error
            }

        /*See if all clause conditions are met for current conclusion on stack*/
            for(int k=0;k<NUM_OF_CLAUSES;++k) {

                answer = "";
                clauseName = cvi.cvl[clauseIdx + k];

                if(clauseName.empty()) {
                    if(stackPtr < 0) break;
                    idx = 0;
                    for (auto &p : rvi.cl) {
                        if (cs[stackPtr]->first == p.first) break;
                    ++idx;
                } break;
            }



            varInfo = getVarInfo(rvi, clauseName);
            clauseValue = varInfo.second.second;
            condition = cvi.cndOp[clauseIdx + k];
            compareValue = cvi.cmpVal[clauseIdx + k];

            varInfo = getVarInfo(rvi, cs[stackPtr]->first);
            varIdx = varInfo.second.first;
            if(varInfo.first == 0) {
                varValue = &rvi.ivl[varIdx].second;
            } else
                varValue = &rvi.tvl[varIdx].second;



            if(clauseValue == compareValue){

        /*Set variable value to the value stored if all clauses are true*/
                if(k == 3 || cvi.cvl[clauseIdx + k + 1].empty()){
                    *varValue = cs[stackPtr]->second;
                    answer = *varValue;
```

```
                break;
            }
        } else
            break;
    }

    if(!answer.empty() && stackPtr<=0) {
        return;
    } else if(stackPtr<0){
        break;
    }

/*Go to next rule that has the same conclusion as the current rule. Replace on stack if
found.
 * If none, pop the stack*/
    currRule = ((clauseIdx/NUM_OF_CLAUSES)+1)*10;
    idx = 0;
    for (auto &p : rvi.cl) {
        if (cs[stackPtr]->first == p.first && ((idx+1)*10) > currRule) {
            clauseIdx = getClauseIndex(idx, NUM_OF_CLAUSES);
            cs[stackPtr] = &p;
            break;
        }
        ++idx;
```

```
            }

            if(idx != rvi.cl.size()) {

                continue;

            }



            cs.pop_back();

            --stackPtr;



            if(stackPtr > -1)

                clauseName = cs[stackPtr]->first;

        /*Reset clause index*/

            idx = 0;

            for (auto &p : rvi.cl) {

                if (clauseName == p.first && idx >= clauseIdx) {

                    clauseIdx = getClauseIndex(idx, NUM_OF_CLAUSES);

                    break;

                } ++idx;

            }

        }

    }
```

/* Uses the forward chaining AI method to decide what the effects of the values deduced in backward chaining will have

*

```
          * rvi - Shared info of if and then variable names and their values

          * cvi - Shared info of clause variable names and their values

          * searchTerm - Term selected from user input to detect initial conclusion

          * cq - the conclusion queue

          * cvp - the clause variable pointer

          * NUM_OF_CLAUSES - maximum number of claus variables for each rule

          *

          */

         bool recommendRepair(RuleVarInfo &rvi, ClauseVarInfo &cvi, string &searchTerm,
deque <string> &cq, pair<int, int> &cvp, const int &NUM_OF_CLAUSES){


             int idx = 0; // Keeps track of index in range-based loops

             bool contains = false; // True - Element found ; False - Element not found

             int currRule; // Current rule number

             int currClause = 0; // Current clause number

             int clauseIdx = 0; // Index of first clause for current rule

             bool repairFound = false; // return true if a repair was found, false is none was
deduced


         /* If term entered is a clause variable, add to queue and set pointer */

             for (auto &s : cvi.cvl) {

                 if (searchTerm == s) {

                     cq.push_back(s);

                     currRule = ((idx / NUM_OF_CLAUSES) + 1) * 10;
```

```
                clauseIdx = 4 * (currRule / 10 - 1);

                cvp.first = currRule;

                cvp.second = currClause;

                contains = true;

                break;

            }

            ++idx;

        }


        pair<int, pair<int, string>> varInfo; // Contains data for current clause var in relation to
var lists

        int varIdx; // Index for var in the list it was found in

        string *varValue; // Points to value for given variable found

        string termValue = "PROBLEM";


        if (!contains) {

            cout << "Could not find this clause in the rules. Try again.";

            exit(1); // Exit with not found error

        } else {

            varInfo = getVarInfo(rvi, searchTerm);

            varIdx = varInfo.second.first;

            varInfo.first == 0 ? varValue = &rvi.ivl[varIdx].second : varValue =
&rvi.tvl[varIdx].second;

            *varValue = termValue;
```

```
                }



                int testIdx = 0;

                while (!cq.empty()) {



            /* While there are still clauses to check */

                    while (cvp.second < NUM_OF_CLAUSES && cvi.cvl[clauseIdx + cvp.second] != "")
{



                        varInfo = getVarInfo(rvi, cvi.cvl[clauseIdx + cvp.second]);

                        varIdx = varInfo.second.first;

                        varInfo.first == 0 ? varValue = &rvi.ivl[varIdx].second : varValue =
&rvi.tvl[varIdx].second;


                        if (*varValue == "") {

                            cout << "[is/are] the " << toLower(splitString(cvi.cvl[clauseIdx + cvp.second]))
<< "? (Enter YES or NO below): " << "\n";

                            cin >> *varValue;

                            toUpper(*varValue);

                        }


                        ++cvp.second;
                    }

            /* All variables are instantiated for this conclusion, test for true or false*/
                    while (cvp.second > 0) {



                        if(clauseIdx != 0) {
                            testIdx = clauseIdx + (NUM_OF_CLAUSES - cvp.second - 1);
                        } else {
                            testIdx = 0;
                        }

                        /* If the clause variable is the first in the IF variable list */
                        if(cvi.cvl[testIdx] == "") {
                            testIdx = testIdx - (NUM_OF_CLAUSES - cvp.second - 1);
                        }

                        varInfo = getVarInfo(rvi, cvi.cvl[testIdx]);
                        varIdx = varInfo.second.first;

                        varInfo.first == 0 ? varValue = &rvi.ivl[varIdx].second : varValue =
&rvi.tvl[varIdx].second;

            /* If a clause condition is not met, move to next possible clause */
```

```
                    if (*varValue != cvi.cmpVal[testIdx]) {
                        break;
                    }

                    --cvp.second;
                }

                varInfo = getVarInfo(rvi, rvi.cl[(cvp.first / 10) - 1].first);
                varIdx = varInfo.second.first;
                varInfo.first == 0 ? varValue = &rvi.ivl[varIdx].second : varValue =
&rvi.tvl[varIdx].second;

        /* If all clauses are true then set the conclusion variable to the correct value */
                if (cvp.second == 0) {
                    repairFound = true;
                    *varValue = rvi.cl[(cvp.first / 10) - 1].second;

                    /* Split string with _ being the delimiter and make lowercase */
                    cout << "\nRecommended repair: " <<  toLower(splitString(*varValue)) << "\n\n";

        /* If conclusion deduced is also a clause variable*/
                    for (auto &s : cvi.cvl) {
                        if (*varValue == s) {
                            cq.push_back(*varValue);
                            break;
                        }
                    }

                }

        /* Check if another rule exists that uses this clause variable */
                idx = 0;
                contains = false;
                currClause = 0;
                for (auto &s : cvi.cvl) {
                    if (cq[0] == s && currRule < ((idx / NUM_OF_CLAUSES) + 1) * 10) {
                        currRule = ((idx / NUM_OF_CLAUSES) + 1) * 10;
                        clauseIdx = 4 * (currRule / 10 - 1);
                        cvp.first = currRule;
                        cvp.second = currClause;
                        contains = true;
                        break;
                    }
                    ++idx;
                }
                if(!contains){
                        cq.pop_front();

                    }




                }

                return repairFound;

        }




        /* Searches and returns the appropriate list info for a given list variable

         *

         * rvi - the rule variable info lists

         * s - string variable value to be searched

         *

         * Pair returned - First value indicates list returned from (0 for if, 1 for then)

         * Second value is a pair containing the index where the variable was found

         * and then the value for that variable

         *

         */

        inline pair<int, pair<int, string>> getVarInfo(RuleVarInfo& rvi, string& s){


                int i = 0;

                /*Check IF var list first*/

                for(auto &p : rvi.ivl){
                    if(s == p.first) {

                        return make_pair(0, make_pair(i, p.second));

                    }

                    ++i;

                }


                i = 0;

                /*Must be inside of THEN var list*/

                for(auto &p : rvi.tvl){

                    if(s == p.first) {

                        return make_pair(1, make_pair(i, p.second));

                    }

                    ++i;

                }


                return make_pair(-1, make_pair(i, "")); // Not inside either - Error

        }


        /* Returns the clause index for a given rule number

         *

         * i - index of rule

         * cn - maximum number of clauses

         *

         */
```

```
inline int getClauseIndex(int&i, const int&cn){
    int ruleNum = (i+1)*10;
    return cn * (ruleNum / 10 - 1 );
}

inline string searchInputForConclusion(string str, RuleVarInfo &rvi) {
    int idx = 0; // Index of first occurrence of search term
    string cvn; // Clause variable name

    /* If clause variable is found, extract the substring and return it*/
    for(auto &s : rvi.cl){
        if(s.first.empty()) {
            continue;
        }
        idx = str.find(s.first);
        if(idx != string::npos){
            cvn = str.substr(idx, s.first.length());
            break;
        }
    }

    return cvn;
}
```

```
/* Replaces all underscores ( _ ) with empty spaces
 *
 * s -  String to be manipulated
 *
 */
inline string splitString(string s) {
    int i = 0;
    for (auto &c : s) {
        if (c == 95)
            s.replace(i, 1, " ");
        ++i;
    }

    return s;

}



/* Changes all lowercase chars in a string to uppercase
 *
 * s -  String to be manipulated
 *
 */
inline void toUpper(string &s){
    for(auto & c : s)
        c = toupper(c);
}



/* Changes all uppercase chars in a string to lowercase
 *
 * s -  String to be manipulated
 *
 */
inline string toLower(string s){
    for(auto &c : s)
        c = tolower(c);

    return s;
}



/* Pull data from rule base file and put into a vector of strings
 *
 * rs -  Rule string with all rule info as strings
 *
 */
inline void extractRules(vector <string> &rs, string method){
```

```
ifstream f; // file

string w; // word

string fName; // file name


if(method == "backward")

    fName = "CarDiagnosis.txt";

else

    fName = "CarRepair.txt";




// Collect rules info from knowledge base

f.open(fName);

if (!f) {

    cout << "Unable to open file";

    exit(1);

}

while (f >> w) {

    if (w != "\n") {

        if(w.substr(0,7) == "RULE: "){

            rs.emplace_back(" ");

        }

        rs.emplace_back(w);

    }
```

```
        }

        f.close();

    }




    /* Fill all data structures with info extracted from the rule base file

     *

     * rs - Rule string with all rule info as strings

     * cvi - Clause variable info container

     * rvi - Rule variable info container

     * cn - Highest number of if clauses in rules

     *

     */

    inline void fillDataStructs(vector <string> &rs, ClauseVarInfo &cvi, RuleVarInfo &rvi,
const int &cn) {

        int clsCtr = 0; // Current clause

        int ruleCtr = 1; // Current rule

        bool contains; // True - Element found ; False - Element not found

        string clauseName; // Name of clause

        string clauseOp; // Conditional operator for clause

        string clauseCompare; // Comparison value for clause


        for (int i = 0; i < rs.size(); ++i) {
```

```
contains = false;

clauseName = rs[i+1];

clauseOp = rs[i+2];

clauseCompare = rs[i+3];


/*Process IF statement variables*/

if (rs[i] == "IF" || rs[i] == "AND") {

    cvi.cvl.emplace_back(clauseName);

    cvi.cndOp.emplace_back(clauseOp);

    cvi.cmpVal.emplace_back(clauseCompare);

    ++clsCtr;

    /*If current var not inside of IF var list, then add new item to it*/

    for (auto &p : rvi.ivl) {

        if (p.first == clauseName) {

            contains = true;

            break;

        }

    }

    if(!contains){

        rvi.ivl.emplace_back(make_pair(rs[i + 1], ""));

    }

}

/*Process THEN statement variables*/

if (rs[i] == "THEN") {
        /*Place empty entries for clause slots not used*/
        for (int j = 0; j < cn - clsCtr; ++j) {
            cvi.cvl.emplace_back("");
            cvi.cndOp.emplace_back("");
            cvi.cmpVal.emplace_back("");
        }
        clsCtr = 0;
        /*Stores rule conclusion name and the value to be set to if conditions are true*/
        rvi.cl.emplace_back(make_pair(clauseName, clauseCompare));
        ++ruleCtr;
        /*If current var not inside of THEN var list, then add new item to it*/
        for (auto &p : rvi.tvl) {
            if (p.first == rs[i + 1]) {
                contains = true;
                break;
            }
        }
        if(!contains){
            rvi.tvl.emplace_back(make_pair(rs[i + 1], ""));
        }
    }
}
/*Remove any IF clause vars that are also THEN vars from the IF var list*/
for(int j=0;j<rvi.ivl.size();++j){
        for(int k=0;k<rvi.tvl.size();++k) {
            if (rvi.ivl[j].first == rvi.tvl[k].first){
                rvi.ivl.erase(rvi.ivl.begin() + j);
                break;
            }
        }
    }
}



/* Print all info for data structures
 *
 * cvi - Clause variable info container
 * rvi - Rule variable info container
 *
 */
inline void print(ClauseVarInfo &cvi, RuleVarInfo &rvi) {
    cout << endl << endl;
    cout << "CLAUSE VARIABLE INFO LIST" << endl << "--------------------------" << endl;
    for (int i = 0; i < cvi.cvl.size(); ++i) {
        if (cvi.cvl[i] == "") {
            cout << "index: " << i << " |";
        } else {
            cout << "index: " << i;
            cout << " | clause: " << cvi.cvl[i];
            cout << " | condition: " << cvi.cndOp[i];
            cout << " | comp val: " << cvi.cmpVal[i];
        }
        cout << endl << "--------------------------------------------------------------" << endl;
    }

    cout << "\n\nIF VAR LIST\n-----------------\n";
    for(auto &p : rvi.ivl){
        cout << p.first << " " << p.second << " \n";
    }
    cout << "\n\nTHEN VAR LIST\n-----------------\n";
    for(auto &p : rvi.tvl){
        cout << p.first << " " << p.second << " \n";
    }
    cout << "\n\nCONCLUSION LIST\n-----------------\n";
    for(int j=0;j<rvi.cl.size();++j){
        cout << ((j+1)*10) << " " << rvi.cl[j].first << " " << rvi.cl[j].second << " \n";
    }

}
```

Clausevarinfo.h:

```
#include <string>

#include <vector>


using namespace std;


struct ClauseVarInfo{

    vector <string> cvl; // Clause Variable List

    vector <string> cmpVal; // Compare Value

    vector <string> cndOp; // Conditional Operator

};
```

Rulevarinfo.h:

```
#include <vector>

#include <string>


using namespace std;


struct RuleVarInfo{
    vector <pair<string, string>> ivl; // If var list <var name, value>

    vector <pair<string, string>> tvl; // Then var list <var name, value>

    vector <pair<string, string>> cl; // Conclusion list <var name, value if clauses true>

};
```