CS4396 AI Project 1:

Auto Repair Expert System Report

By Yuvanesh Rajamani

**Problem description:**

Create an intelligent computer expert system for an auto repair application.  We will have to diagnose the problem with the vehicle and then recommend repairs for that problem. We will need to use the Backwards chaining and forward chaining methodologies we learned in class. The vehicle diagnosis should be found using backwards chaining with a minimum of 30 rules. The vehicles repairs should be determined using forward chaining.

**Domain:**

Our goal is to develop a program that can expertly diagnose common issues with vehicles and deliver a repair for that issue. Our program will use Backwards and Forward chaining to deliver accurate results.

**Methodologies:**

Backward Chaining:

Backwards chaining is the methodology we will use to determine the problem with our vehicle. This works by first getting a conclusion and working backwards to find the problem. This is implemented into our code by creating four data structures that contain the knowledge base(if-then rules), conclusion list, variable list, and clause variable list. First the program will ask for a conclusion and scan it in the conclusion list. Once found it will push the rule number onto the stack. Next it will instantiate any variables that are needed. It will look in the clause variable list for the variables used in the rules. Once the rule is instantiated you can pop the stack and then you have your problem.

Forwards Chaining:

Forwards chaining is the methodology we used to implement the repair portion of this program. Forward chaining is the opposite if backwards chaining in that it starts with the if side of the rules first. This is implemented using five data structures: the knowledge base(if-then), clause variable list, clause variable pointer, variable list, and conclusion variable queue. First the program will take the conclusion and place it into the queue with its value. The clause variable list is then searched for the variable. The rule number is then placed into the pointer. The rules are instantiated. Once the rule is true the then part is invoked and placed onto the back of the queue. Once there are no more if statements with the variable at the front if the queue, the variable is then removed from the queue.

**Decision Trees:**

Diagram

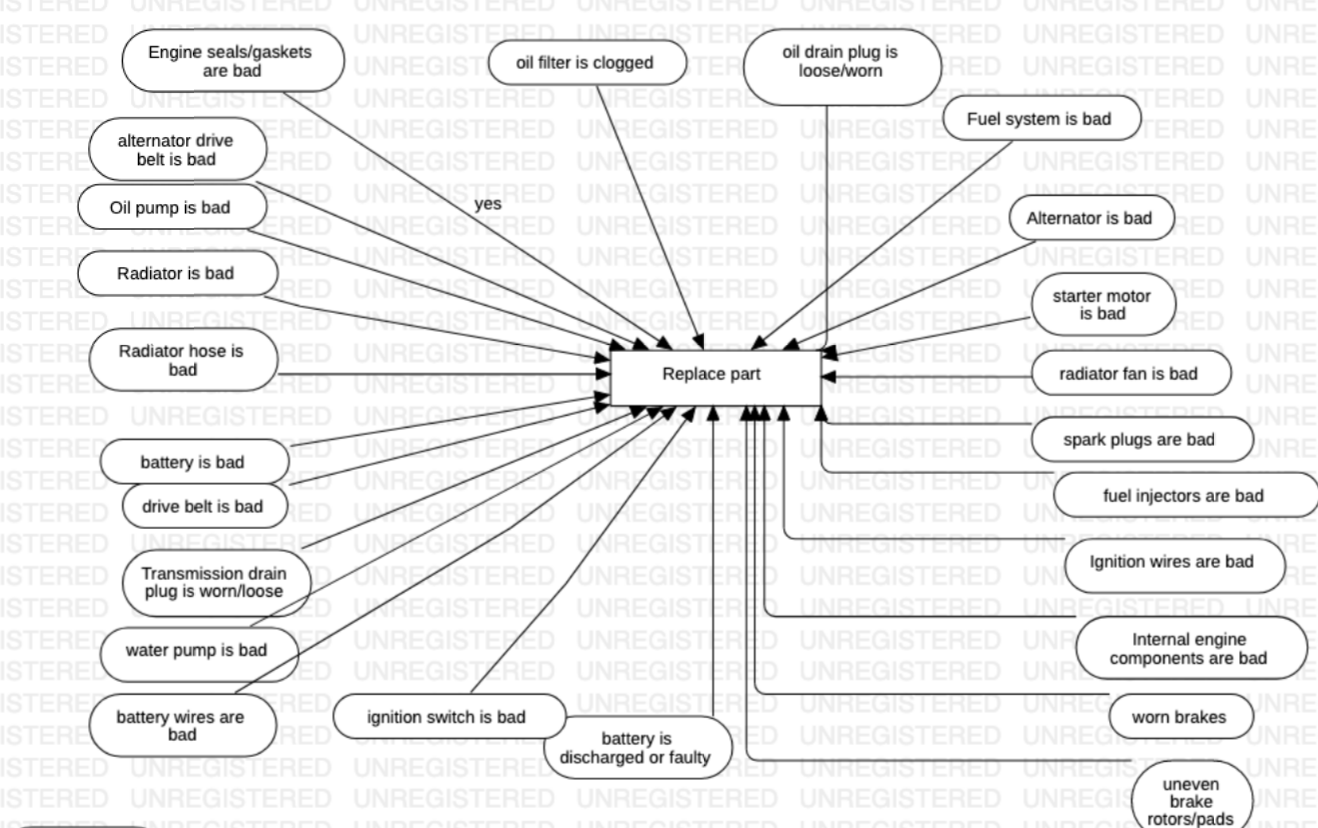
Description automatically generatedBackwards Chaining tree: Diagram

Description automatically generated Diagram

Description automatically generatedDiagram

Description automatically generated

Forwards Chaining tree:



**Rules:**

Backwards Chaining If-Then Rules:

|  |  |
| --- | --- |
|  | Rule |
| 10 | IF PROBLEM = NO |
|  | THEN PART = NONE |
| 20 | IF PROBLEM = YES AND OIL\_PRESSURE = YES AND OIL\_GRADE = YES AND OIL\_FILTER = YES |
|  | THEN PART = OIL PUMP |
| 30 | IF PROBLEM = YES AND OIL\_PRESSURE = YES AND OIL\_GRADE = YES AND OIL\_FILTER = NO |
|  | THEN PART = OIL FILTER |
| 40 | IF PROBLEM = YES AND OIL\_PRESSURE = YES AND OIL\_GRADE = NO |
|  | THEN PART = OIL |
| 50 | IF PROBLEM = YES AND LEAK = YES AND OIL\_LEAK = YES AND OIL\_DRAIN\_PLUG = YES |
|  | THEN PART = OIL DRAIN PLUG |
| 60 | IF PROBLEM = YES AND LEAK = YES AND OIL\_LEAK = YES AND ENGINE\_LEAK = YES |
|  | THEN PART = ENGINE GASKET |
| 70 | IF PROBLEM = YES AND LEAK = YES AND COOLANT\_LEAK = YES AND RADIATOR\_LEAK = YES |
|  | THEN PART = RADIATOR |
| 80 | IF PROBLEM = YES AND LEAK = YES AND COOLANT\_LEAK = YES AND RADIATOR\_HOSE = YES |
|  | THEN PART = RADIATOR HOSE |
| 90 | IF PROBLEM = YES AND LEAK = YES AND TRANS\_LEAK = YES AND TRANS\_DRAIN\_PLUG = YES |
|  | THEN PART = TRANSMISSION DRAIN PLUG |
| 100 | IF PROBLEM = YES AND ENGINE\_ROTATES = NO AND BATT\_TERMINALS = YES AND BATT\_VOLTAGE < 12 |
|  | THEN PART = BATTERY |
| 110 | IF PROBLEM = YES AND ENGINE\_ROTATES = NO AND BATT\_TERMINALS = YES AND BATT\_VOLTAGE > 12 AND START\_MOTOR = YES |
|  | THEN PART = IGNITION SWITCH |
| 120 | IF PROBLEM = YES AND ENGINE\_ROTATES = NO AND BATT\_TERMINALS = YES AND BATT\_VOLTAGE > 12 AND START\_MOTOR = NO |
|  | THEN PART = STARTER MOTOR |
| 130 | IF PROBLEM = YES AND ENGINE\_ROTATES = NO AND BATT\_TERMINALS = NO |
|  | THEN PART = BATTERY TERMINALS |
| 140 | IF PROBLEM = YES AND ENGINE\_ROTATES = YES AND FUEL = NO |
|  | THEN PART = FUEL\_LEVEL |
| 150 | IF PROBLEM = YES AND ENGINE\_ROTATES = YES AND FUEL = YES |
|  | THEN PART = FUEL SYSTEM |
| 160 | IF PROBLEM = YES AND ROUGH = YES AND VACUUM\_HOSE = YES |
|  | THEN PART = VACUUM HOSE |
| 170 | IF PROBLEM = YES AND ROUGH = YES AND SPARK\_PLUGS = YES |
|  | THEN PART = SPARK PLUGS |
| 180 | IF PROBLEM = YES AND ROUGH = YES AND BAD\_FUEL = YES |
|  | THEN PART = FUEL\_CONDITION |
| 190 | IF PROBLEM = YES AND ROUGH = YES AND FUEL\_INJECTORS = YES |
|  | THEN PART = FUEL INJECTORS |
| 200 | IF PROBLEM = YES AND ROUGH = YES AND IGNITION\_WIRES = YES |
|  | THEN PART = IGNITION WIRES |
| 210 | IF PROBLEM = YES AND ROUGH = YES AND KNOCKING = YES |
|  | THEN PART = ENGINE COMPONENTS |
| 220 | IF PROBLEM = YES AND BATT\_CHARGE = NO AND BATT\_TEST = NO |
|  | THEN PART = BATTERY |
| 230 | IF PROBLEM = YES AND BATT\_CHARGE = NO AND BATT\_TEST = YES AND ALTERNATOR = NO |
|  | THEN PART = ALTERNATOR |
| 240 | IF PROBLEM = YES AND BATT\_CHARGE = NO AND BATT\_TEST = YES AND ALTERNATOR = YES AND ALT\_DRIVE\_BELT = YES |
|  | THEN PART = ALTERNATOR DRIVE BELT |
| 250 | IF PROBLEM = YES AND BATT\_CHARGE = NO AND BATT\_TEST = YES AND ALTERNATOR = YES AND ALT\_DRIVE\_BELT = NO AND BATT\_WIRES = YES |
|  | THEN PART = BATTERY WIRES |
| 260 | IF PROBLEM = YES AND BATT\_CHARGE = NO AND BATT\_TEST = YES AND ALTERNATOR = YES AND ALT\_DRIVE\_BELT = NO AND BATT\_WIRES = NO |
|  | THEN PART = ELECTRICAL SYSTEM |
| 270 | IF PROBLEM = YES AND OVERHEAT = YES AND COOLANT = NO |
|  | THEN PART = COOLANT |
| 280 | IF PROBLEM = YES AND OVERHEAT = YES AND COOLANT = YES AND DRIVE\_BELT = YES |
|  | THEN PART = DRIVE BELT |
| 290 | IF PROBLEM = YES AND OVERHEAT = YES AND COOLANT = YES AND DRIVE\_BELT = NO AND RADIATOR = YES |
|  | THEN PART = RADIATOR |
| 300 | IF PROBLEM = YES AND OVERHEAT = YES AND COOLANT = YES AND DRIVE\_BELT = NO AND RADIATOR = NO AND RADIATOR\_FAN = NO |
|  | THEN PART = RADIATOR FAN |
| 310 | IF PROBLEM = YES AND OVERHEAT = YES AND COOLANT = YES AND DRIVE\_BELT = NO AND RADIATOR = NO AND RADIATOR\_FAN = YES |
|  | THEN PART = WATER PUMP |
| 320 | IF PROBLEM = YES AND FUEL\_USE = YES AND TIRE\_PRESSURE = NO |
|  | THEN PART = TIRE PRESSURE |
| 330 | IF PROBLEM = YES AND FUEL\_USE = YES AND TIRE\_PRESSURE = YES AND AIR\_FILTER = NO |
|  | THEN PART = AIR FILTER |
| 340 | IF PROBLEM = YES AND FUEL\_USE = YES AND TIRE\_PRESSURE = YES AND AIR\_FILTER = YES AND FUEL\_SMELL = YES |
|  | THEN PART = FUEL TANK |
| 350 | IF PROBLEM = YES AND FUEL\_USE = YES AND TIRE\_PRESSURE = YES AND AIR\_FILTER = YES AND FUEL\_SMELL = NO |
|  | THEN PART = FUEL MANAGEMENT SYSTEM |
| 360 | IF PROBLEM = YES AND CHECK\_BRAKE = YES AND WOBBLE = YES |
|  | THEN PART = BRAKE PARTS |
| 370 | IF PROBLEM = YES AND CHECK\_BRAKE = YES AND CHEMICAL\_SMELL = YES |
|  | THEN PART = BRAKE PARTS |
| 380 | IF PROBLEM = YES AND CHECK\_BRAKE = YES AND SPONGEY = YES |
|  | THEN PART = BRAKE LINES |
| 390 | IF PROBLEM = YES AND CHECK\_BRAKE = YES AND SQUEAK = YES |
|  | THEN PART = BRAKE PARTS |

Forwards Chaining If-Then Rules:

|  |  |
| --- | --- |
|  | Rule |
| 10 | IF OIL\_PUMP = BAD |
|  | THEN REPLACE = YES |
| 20 | IF OIL\_FILTER = BAD |
|  | THEN REPLACE = YES |
| 30 | IF OIL = BAD |
|  | THEN REPLACE = YES |
| 40 | IF OIL\_DRAIN\_PLUG = BAD |
|  | THEN REPLACE = YES |
| 50 | IF ENGINE\_GASKET = BAD |
|  | THEN REPLACE = YES |
| 60 | IF RADIATOR = BAD |
|  | THEN REPLACE = YES |
| 70 | IF RADIATOR\_HOSE = BAD |
|  | THEN REPLACE = YES |
| 80 | IF TRANSMISSION\_DRAIN\_PLUG = BAD |
|  | THEN REPLACE = YES |
| 90 | IF BATTERY = BAD |
|  | THEN REPLACE = YES |
| 100 | IF IGNITION\_SWITCH = BAD |
|  | THEN REPLACE = YES |
| 110 | IF STARTER\_MOTOR = BAD |
|  | THEN REPLACE = YES |
| 120 | IF BATTERY\_TERMINALS = BAD |
|  | THEN RECONNECT = YES |
| 130 | IF FUEL\_LEVEL = EMPTY |
|  | THEN REPLACE = YES |
| 140 | IF FUEL\_CONDITION = BAD |
|  | THEN REPLACE = YES |
| 150 | IF FUEL\_SYSTEM = BAD |
|  | THEN REPLACE = YES |
| 160 | IF VACUUM\_HOSE = BAD |
|  | THEN REPLACE = YES |
| 170 | IF SPARK\_PLUGS = BAD |
|  | THEN REPLACE = YES |
| 180 | IF FUEL\_INJECTORS = BAD |
|  | THEN REPLACE = YES |
| 190 | IF IGNITION\_WIRES = BAD |
|  | THEN REPLACE = YES |
| 200 | IF ENGINE\_COMPONENTS = BAD |
|  | THEN REPLACE = YES |
| 210 | IF BATTERY = BAD |
|  | THEN REPLACE = YES |
| 220 | IF ALTERNATOR = BAD |
|  | THEN REPLACE = YES |
| 230 | IF ALTERNATOR\_BELT = BAD |
|  | THEN REPLACE = YES |
| 240 | IF BATTERY\_WIRES = BAD |
|  | THEN REPLACE = YES |
| 250 | IF ELECTRICAL\_SYSTEM = BAD |
|  | THEN REPLACE = YES |
| 260 | IF COOLANT = BAD |
|  | THEN REPLACE = YES |
| 270 | IF DRIVE\_BELT = BAD |
|  | THEN REPLACE = YES |
| 280 | IF RADIATOR = BAD |
|  | THEN REPLACE = YES |
| 290 | IF RADIATOR\_FAN = BAD |
|  | THEN REPLACE = YES |
| 300 | IF WATER\_PUMP = BAD |
|  | THEN REPLACE = YES |
| 310 | IF TIRE\_PRESSURE = BAD |
|  | THEN REPLACE = YES |
| 320 | IF AIR\_FILTER = BAD |
|  | THEN REPLACE = YES |
| 330 | IF FUEL\_TANK = BAD |
|  | THEN REPLACE = YES |
| 340 | IF FUEL\_MANAGEMENT\_SYSTEM = BAD |
|  | THEN REPLACE = YES |
| 350 | IF BRAKE\_PARTS = BAD |
|  | THEN REPLACE = YES |
| 360 | IF BRAKE\_PARTS = BAD |
|  | THEN REPLACE = YES |
| 370 | IF BRAKE\_LINES = BAD |
|  | THEN REPLACE = YES |
| 380 | IF BRAKE\_PARTS = BAD |
|  | THEN REPLACE = YES |

**Source Code Implementation:**

We built our program by using the erroneous c code provided by Dr.Moonis Ali. We translated it to c++ and removed any GOTO statements that were presented in them. We then made the code more in line with SOLID principles by breaking up responsibilities in the code. We then implemented our unique data structures into the code. Lastly we made header files for the two separate programs and made then run together in main.

**Source Code:**

Main.cpp:

#include "bc.h"  
#include "fc.h"  
#include <iostream>  
  
using namespace std;  
  
int main(){  
 string problem;  
 string solution;  
  
 cout << "Starting backwards chaining code....\n" << endl;  
  
 bc backwards;  
 problem.assign(backwards.start());  
  
 cout << "Done with backwards chaining, your problem is : \n" << problem << endl;

if (problem !="NO ISSUE"){

cout << "Starting forwards chaining code....\n" << endl ;

fc forwards(problem);

solution.assign(forwards.start());

cout << "To fix you need to " <<solution<<" "<< problem << endl;

;

}

else{

cout << "Your car is fine "<< endl;

}

}

Bc.h:

#include <string>  
  
using namespace std;  
  
class bc{  
private :  
 */\*\*\*\*\*backward chaining \*\*\*\*\*\*\*/* char concList[40][5]; */\* conclusion list \*/* char varList[45][17]; */\* variable list \*/* char clVarList[1676][17]; */\* clause variable list \*/* char varble[45];  
 char PART[4],PROBLEM[4],OIL\_PRESSURE[4],OIL\_FILTER[4],OIL\_GRADE[4],LEAK[4],OIL\_LEAK[4],COOLANT\_LEAK[4],  
 TRANS\_LEAK[4],OIL\_DRAIN\_PLUG[4],ENGINE\_LEAK[4],RADIATOR\_LEAK[4],RADIATOR\_HOSE[4],TRANS\_DRAIN\_PLUG[4],  
 ENGINE\_ROTATES[4],BATT\_TERMINALS[4],FUEL[4],START\_MOTOR[4],ROUGH[4],VACUUM\_HOSE[4],SPARK\_PLUGS[4],  
 BAD\_FUEL[4],FUEL\_INJECTORS[4],IGNITION\_WIRES[4],KNOCKING[4],BATT\_CHARGE[4],BATT\_TEST[4],ALTERNATOR[4],  
 ALT\_DRIVE\_BELT[4],BATT\_WIRES[4],OVERHEAT[4],COOLANT[4],DRIVE\_BELT[4],RADIATOR[4],RADIATOR\_FAN[4],FUEL\_USE[4],  
 TIRE\_PRESSURE[4],AIR\_FILTER[4],FUEL\_SMELL[4],CHECK\_BRAKE[4],WOBBLE[4],CHEMICAL\_SMELL[4],SPONGEY[4],  
 SQUEAK[4], end[4];  
 char buff[128];  
  
 int instList[45]; */\* instantiated list \*/* int stateStk[45]; */\* statement stack \*/* int clausStk[45] */\* clause stack \*/*, stateNum, flag, i, j, s, k, sp */\*stack pointer \*/*;  
 float BATT\_VOLTAGE;  
 void determine\_member\_concl\_list();  
 void push\_on\_stack();  
 void instantiate();  
 void chkClauses();  
 void bcLoop();  
 void popStack();  
 char yourproblem();  
  
public :  
 char \* start();  
 bc(){}  
};

Fc.h:

#include <string>  
#include <cstring>  
#include <iostream>  
using namespace std;  
  
class fc{  
private :  
 char cndVar[40][40];  
 char varList[45][17], */\* variable list\*/* clVarList[1445][17]; */\* clause var list \*/* char c[30] */\* condition variable \*/*, v[3]; */\*variable \*/* char OIL\_PUMP[4],OIL\_FILTER[4],OIL[4],OIL\_DRAIN\_PLUG[4],ENGINE\_GASKET[4],RADIATOR[4],RADIATOR\_HOSE[4],  
 TRANSMISSION\_DRAIN\_PLUG[4],BATTERY[4],IGNITION\_SWITCH[4],STARTER\_MOTOR[4],BATTERY\_TERMINALS[4],FUEL\_LEVEL[4],  
 FUEL\_CONDITION[4],FUEL\_SYSTEM[4],VACUUM\_HOSE[4],SPARK\_PLUGS[4],FUEL\_INJECTORS[4],IGNITION\_WIRES[4],ENGINE\_COMPONENTS[4]  
 ,ALTERNATOR[4],ALTERNATOR\_BELT[4],BATTERY\_WIRES[4],ELECTRICAL\_SYSTEM[4],COOLANT[4],DRIVE\_BELT[4],RADIATOR\_FAN[4],  
 WATER\_PUMP[4],TIRE\_PRESSURE[4],AIR\_FILTER[4],FUEL\_TANK[4],FUEL\_MANAGEMENT\_SYSTEM[4],BRAKE\_PARTS[4],BRAKE\_LINES[4],  
 REPLACE[4],RECONNECT[4];  
 int instList[17]; */\* instantiated list\*/* int flag, i, j, k, s, fp */\* front pointer \*/*;  
 int bp */\* back pointer \*/*, sn; */\* statement number \*/* int cn; */\* clause number \*/* char end[4];  
  
 void searchClauses();  
 void check\_instantiation();  
 void instantiate();  
 void fcLoop();  
  
public:  
 fc(string);  
 char \* start();  
};

Bc.cpp:

#include<string>  
#include<cstdio>  
#include<cstring>  
#include<iostream>  
  
#include "bc.h"  
  
using namespace std;  
  
char \* bc::start() {  
 */\*\*\*\*\* initialization section \*\*\*\*\*\*/* sp = 45;  
 for (i = 1; i < 45; i++) {  
 strcpy(concList[i], "");  
 strcpy(varList[i], "");  
 instList[i] = 0;  
 stateStk[i] = 0;  
 clausStk[i] = 0;  
 }  
 for (i = 1; i < 1677; i++) strcpy(clVarList[i], "");  
  
 */\*populate conclusion list\*/* for (int i = 1; i < 40; i++) {  
 strcpy(concList[i], "Part");  
 }  
  
 printf("\*\*\* CONCLUSION LIST \*\*\*\n");  
 for (i = 1; i < 40; i++) printf("CONCLUSION %d %s\n", i, concList[i]);  
 printf("HIT RETURN TO CONTINUE");  
 gets(buff);  
  
 */\*populate variable list\*/* strcpy(varList[1], "PROBLEM");  
 strcpy(varList[2], "OIL\_PRESSURE");  
 strcpy(varList[3], "OIL\_FILTER");  
 strcpy(varList[4], "OIL\_GRADE");  
 strcpy(varList[5], "LEAK");  
 strcpy(varList[6], "OIL\_LEAK");  
 strcpy(varList[7], "COOLANT\_LEAK");  
 strcpy(varList[8], "TRANS\_LEAK");  
 strcpy(varList[9], "OIL\_DRAIN\_PLUG");  
 strcpy(varList[10], "ENGINE\_LEAK");  
 strcpy(varList[11], "RADIATOR\_LEAK");  
 strcpy(varList[12], "RADIATOR\_HOSE");  
 strcpy(varList[13], "TRANS\_DRAIN\_PLUG");  
 strcpy(varList[14], "ENGINE\_ROTATES");  
 strcpy(varList[15], "BATT\_TERMINALS");  
 strcpy(varList[16], "FUEL");  
 strcpy(varList[17], "BATT\_VOLTAGE");  
 strcpy(varList[18], "START\_MOTOR");  
 strcpy(varList[19], "ROUGH");  
 strcpy(varList[20], "VACUUM\_HOSE");  
 strcpy(varList[21], "SPARK\_PLUGS");  
 strcpy(varList[22], "BAD\_FUEL");  
 strcpy(varList[23], "FUEL\_INJECTORS");  
 strcpy(varList[24], "IGNITION\_SWITCH");  
 strcpy(varList[25], "KNOCKING");  
 strcpy(varList[26], "BATT\_CHARGE");  
 strcpy(varList[27], "BATT\_TEST");  
 strcpy(varList[28], "ALTERNATOR");  
 strcpy(varList[29], "ALT\_DRIVE\_BELT");  
 strcpy(varList[30], "BATT\_WIRES");  
 strcpy(varList[31], "OVERHEAT");  
 strcpy(varList[32], "COOLANT");  
 strcpy(varList[33], "DRIVE\_BELT");  
 strcpy(varList[34], "RADIATOR");  
 strcpy(varList[35], "RADIATOR\_FAN");  
 strcpy(varList[36], "FUEL\_USE");  
 strcpy(varList[37], "TIRE\_PRESSURE");  
 strcpy(varList[38], "AIR\_FILTER");  
 strcpy(varList[39], "FUEL\_SMELL");  
 strcpy(varList[40], "CHECK\_BRAKE");  
 strcpy(varList[41], "WOBBLE");  
 strcpy(varList[42], "CHEMICAL\_SMELL");  
 strcpy(varList[43], "SPONGEY");  
 strcpy(varList[44], "SQUEAK");  
  
 printf("\*\*\* VARIABLE LIST \*\n");  
 for (i = 1; i < 45; i++) printf("VARIABLE %d %s\n", i, varList[i]);  
 printf("HIT RETURN KEY TO CONTINUE");  
 gets(buff);  
  
 */\*populate clause variable list\*/* strcpy(clVarList[1], "PROBLEM");  
 strcpy(clVarList[45], "PROBLEM");  
 strcpy(clVarList[46], "OIL\_PRESSURE");  
 strcpy(clVarList[47], "OIL\_GRADE");  
 strcpy(clVarList[48], "OIL\_FILTER");  
 strcpy(clVarList[89], "PROBLEM");  
 strcpy(clVarList[90], "OIL\_PRESSURE");  
 strcpy(clVarList[91], "OIL\_GRADE");  
 strcpy(clVarList[92], "OIL\_FILTER");  
 strcpy(clVarList[133], "PROBLEM");  
 strcpy(clVarList[134], "OIL\_PRESSURE");  
 strcpy(clVarList[135], "OIL\_GRADE");  
 strcpy(clVarList[177], "PROBLEM");  
 strcpy(clVarList[178], "LEAK");  
 strcpy(clVarList[179], "OIL\_LEAK");  
 strcpy(clVarList[180], "OIL\_DRAIN\_PLUG");  
 strcpy(clVarList[221], "PROBLEM");  
 strcpy(clVarList[222], "LEAK");  
 strcpy(clVarList[223], "OIL\_LEAK");  
 strcpy(clVarList[224], "ENGINE\_LEAK");  
 strcpy(clVarList[265], "PROBLEM");  
 strcpy(clVarList[266], "LEAK");  
 strcpy(clVarList[267], "COOLANT\_LEAK");  
 strcpy(clVarList[268], "RADIATOR\_LEAK");  
 strcpy(clVarList[309], "PROBLEM");  
 strcpy(clVarList[310], "LEAK");  
 strcpy(clVarList[311], "COOLANT\_LEAK");  
 strcpy(clVarList[312], "RADIATOR\_HOSE");  
 strcpy(clVarList[353], "PROBLEM");  
 strcpy(clVarList[354], "LEAK");  
 strcpy(clVarList[355], "TRANS\_LEAK");  
 strcpy(clVarList[356], "TRANS\_DRAIN\_PLUG");  
 strcpy(clVarList[397], "PROBLEM");  
 strcpy(clVarList[398], "ENGINE\_ROTATES");  
 strcpy(clVarList[399], "BATT\_TERMINALS");  
 strcpy(clVarList[400], "BATT\_VOLTAGE");  
 strcpy(clVarList[441], "PROBLEM");  
 strcpy(clVarList[442], "ENGINE\_ROTATES");  
 strcpy(clVarList[443], "BATT\_TERMINALS");  
 strcpy(clVarList[444], "BATT\_VOLTAGE");  
 strcpy(clVarList[445], "START\_MOTOR");  
 strcpy(clVarList[485], "PROBLEM");  
 strcpy(clVarList[486], "ENGINE\_ROTATES");  
 strcpy(clVarList[487], "BATT\_TERMINALS");  
 strcpy(clVarList[488], "BATT\_VOLTAGE");  
 strcpy(clVarList[489], "START\_MOTOR");  
 strcpy(clVarList[529], "PROBLEM");  
 strcpy(clVarList[530], "ENGINE\_ROTATES");  
 strcpy(clVarList[531], "BATT\_TERMINALS");  
 strcpy(clVarList[573], "PROBLEM");  
 strcpy(clVarList[574], "ENGINE\_ROTATES");  
 strcpy(clVarList[575], "FUEL");  
 strcpy(clVarList[617], "PROBLEM");  
 strcpy(clVarList[618], "ENGINE\_ROTATES");  
 strcpy(clVarList[619], "FUEL");  
 strcpy(clVarList[661], "PROBLEM");  
 strcpy(clVarList[662], "ROUGH");  
 strcpy(clVarList[663], "VACUUM\_HOSE");  
 strcpy(clVarList[705], "PROBLEM");  
 strcpy(clVarList[706], "ROUGH");  
 strcpy(clVarList[707], "SPARK\_PLUGS");  
 strcpy(clVarList[749], "PROBLEM");  
 strcpy(clVarList[750], "ROUGH");  
 strcpy(clVarList[751], "BAD\_FUEL");  
 strcpy(clVarList[793], "PROBLEM");  
 strcpy(clVarList[794], "ROUGH");  
 strcpy(clVarList[795], "FUEL\_INJECTORS");  
 strcpy(clVarList[837], "PROBLEM");  
 strcpy(clVarList[838], "ROUGH");  
 strcpy(clVarList[839], "IGNITION\_WIRES");  
 strcpy(clVarList[881], "PROBLEM");  
 strcpy(clVarList[882], "ROUGH");  
 strcpy(clVarList[883], "KNOCKING");  
 strcpy(clVarList[925], "PROBLEM");  
 strcpy(clVarList[926], "BATT\_CHARGE");  
 strcpy(clVarList[927], "BATT\_TEST");  
 strcpy(clVarList[969], "PROBLEM");  
 strcpy(clVarList[970], "BATT\_CHARGE");  
 strcpy(clVarList[971], "BATT\_TEST");  
 strcpy(clVarList[972], "ALTERNATOR");  
 strcpy(clVarList[1013], "PROBLEM");  
 strcpy(clVarList[1014], "BATT\_CHARGE");  
 strcpy(clVarList[1015], "BATT\_TEST");  
 strcpy(clVarList[1016], "ALTERNATOR");  
 strcpy(clVarList[1017], "ALT\_DRIVE\_BELT");  
 strcpy(clVarList[1057], "PROBLEM");  
 strcpy(clVarList[1058], "BATT\_CHARGE");  
 strcpy(clVarList[1059], "BATT\_TEST");  
 strcpy(clVarList[1060], "ALTERNATOR");  
 strcpy(clVarList[1061], "ALT\_DRIVE\_BELT");  
 strcpy(clVarList[1062], "BATT\_WIRES");  
 strcpy(clVarList[1101], "PROBLEM");  
 strcpy(clVarList[1102], "BATT\_CHARGE");  
 strcpy(clVarList[1103], "BATT\_TEST");  
 strcpy(clVarList[1104], "ALTERNATOR");  
 strcpy(clVarList[1105], "ALT\_DRIVE\_BELT");  
 strcpy(clVarList[1106], "BATT\_WIRES");  
 strcpy(clVarList[1145], "PROBLEM");  
 strcpy(clVarList[1146], "OVERHEAT");  
 strcpy(clVarList[1147], "COOLANT");  
 strcpy(clVarList[1189], "PROBLEM");  
 strcpy(clVarList[1190], "OVERHEAT");  
 strcpy(clVarList[1191], "COOLANT");  
 strcpy(clVarList[1192], "DRIVE\_BELT");  
 strcpy(clVarList[1233], "PROBLEM");  
 strcpy(clVarList[1234], "OVERHEAT");  
 strcpy(clVarList[1235], "COOLANT");  
 strcpy(clVarList[1236], "DRIVE\_BELT");  
 strcpy(clVarList[1237], "RADIATOR");  
 strcpy(clVarList[1277], "PROBLEM");  
 strcpy(clVarList[1278], "OVERHEAT");  
 strcpy(clVarList[1279], "COOLANT");  
 strcpy(clVarList[1280], "DRIVE\_BELT");  
 strcpy(clVarList[1281], "RADIATOR");  
 strcpy(clVarList[1282], "RADIATOR\_FAN");  
 strcpy(clVarList[1321], "PROBLEM");  
 strcpy(clVarList[1322], "OVERHEAT");  
 strcpy(clVarList[1323], "COOLANT");  
 strcpy(clVarList[1324], "DRIVE\_BELT");  
 strcpy(clVarList[1325], "RADIATOR");  
 strcpy(clVarList[1326], "RADIATOR\_FAN");  
 strcpy(clVarList[1365], "PROBLEM");  
 strcpy(clVarList[1366], "FUEL\_USE");  
 strcpy(clVarList[1364], "TIRE\_PRESSURE");  
 strcpy(clVarList[1409], "PROBLEM");  
 strcpy(clVarList[1410], "FUEL\_USE");  
 strcpy(clVarList[1411], "TIRE\_PRESSURE");  
 strcpy(clVarList[1412], "AIR\_FILTER");  
 strcpy(clVarList[1453], "PROBLEM");  
 strcpy(clVarList[1454], "FUEL\_USE");  
 strcpy(clVarList[1455], "TIRE\_PRESSURE");  
 strcpy(clVarList[1456], "AIR\_FILTER");  
 strcpy(clVarList[1457], "FUEL\_SMELL");  
 strcpy(clVarList[1497], "PROBLEM");  
 strcpy(clVarList[1498], "FUEL\_USE");  
 strcpy(clVarList[1499], "TIRE\_PRESSURE");  
 strcpy(clVarList[1500], "AIR\_FILTER");  
 strcpy(clVarList[1501], "FUEL\_SMELL");  
 strcpy(clVarList[1541], "PROBLEM");  
 strcpy(clVarList[1542], "CHECK\_BRAKE");  
 strcpy(clVarList[1543], "WOBBLE");  
 strcpy(clVarList[1585], "PROBLEM");  
 strcpy(clVarList[1586], "CHECK\_BRAKE");  
 strcpy(clVarList[1587], "CHEMICAL\_SMELL");  
 strcpy(clVarList[1629], "PROBLEM");  
 strcpy(clVarList[1630], "CHECK\_BRAKE");  
 strcpy(clVarList[1631], "SPONGEY");  
 strcpy(clVarList[1673], "PROBLEM");  
 strcpy(clVarList[1674], "CHECK\_BRAKE");  
 strcpy(clVarList[1675], "SQUEAK");  
  
 printf("\*\*\* CLAUSE VARIABLE LIST \*\*\*\n");  
 for (i = 1; i < 40; i++) {  
 printf("\*\* CLAUSE %d\n", i);  
 for (j = 1; j < 45; j++) {  
 k = 44 \* (i - 1) + j;  
 printf("VARIABLE %d %s\n", j, clVarList[k]);  
 }  
 }  
  
 */\*\*\*\*\*\* inference section \*\*\*\*\*/* printf("\*\* ENTER CONCLUSION : ");  
 gets(varble);  
 */\* get conclusion statement number (stateNum) from the conclusion list  
 (concList) \*/  
 /\* first statement starts search \*/* bcLoop();  
 return end;  
}  
  
void bc::determine\_member\_concl\_list() {  
*/\* routine to determine if a variable (varble) is a member of the  
 conclusion list (concList). if yes return stateNum != 0.  
 if not a member stateNum=0;  
\*/  
 /\* initially set to not a member \*/* stateNum = 0;  
 */\* member of conclusion list to be searched is flag \*/* i = flag;  
 while((strcmp(varble, concList[i]) != 0) && (i<40))  
 */\* test for membership \*/* {  
 i=i+1;  
 }  
  
 if (strcmp(varble, concList[i]) == 0) stateNum = i; */\* a member \*/*}  
  
void bc::push\_on\_stack()  
*/\* routine to push statement number (stateNum) and a clause number of 1 onto the  
conclusion stack which consists of the statement stack (stateStk) and the  
clause stack (clausStk)..to push decrement stack pointer (sp) \*/*{  
 sp=sp-1;  
 stateStk[sp] = stateNum;  
 clausStk[sp] = 1;  
}  
  
void bc::instantiate()  
*/\* routine to instantiate a variable (varble) if it isn't already. the  
instantiate indication (instList) is a 0 if not, a 1 if it is. the  
variable list (varList) contains the variable (varble). \*/*{  
 i=1;  
 */\* find variable in the list \*/* while((strcmp(varble, varList[i]) != 0) && (i<45)) i=i+1;  
 if((strcmp(varble, varList[i]) == 0) && (instList[i] != 1))  
 */\*found variable and not already instantiated \*/* {  
 instList[i]=1; */\*mark instantiated \*/  
 /\* the designer of the  
 knowledge base places the input statements to  
 instantiate the variables below in the case statement \*/* switch (i)  
 {  
 */\*\*\*\*\* input statement \*\*\*\*\*/  
 /\* input statements for sample position knowledge base \*/* case 1: printf("Is there a problem with the car? INPUT YES OR NO");  
 gets(PROBLEM);  
 break;  
 case 2: printf("Is the oil pressure low? INPUT YES OR NO");  
 gets(OIL\_PRESSURE);  
 break;  
 case 3: printf("Is the oil filter new? INPUT YES OR NO");  
 gets(OIL\_FILTER);  
 break;  
 case 4: printf("Is the oil grade correct? INPUT YES OR NO");  
 gets(OIL\_GRADE);  
 break;  
 case 5: printf("Do you see a leak? INPUT YES OR NO");  
 gets(LEAK);  
 break;  
 case 6: printf("Is it an oil leak? INPUT YES OR NO");  
 gets(OIL\_LEAK);  
 break;  
 case 7: printf("Is it a coolant leak? INPUT YES OR NO");  
 gets(COOLANT\_LEAK);  
 break;  
 case 8: printf("Is it a transmission fluid leak? INPUT YES OR NO");  
 gets(TRANS\_LEAK);  
 break;  
 case 9: printf("Is the leak coming from the oil drain plug? INPUT YES OR NO");  
 gets(OIL\_DRAIN\_PLUG);  
 break;  
 case 10: printf("Is the leak coming from a seam in the engine? INPUT YES OR NO");  
 gets(ENGINE\_LEAK);  
 break;  
 case 11: printf("Is the leak coming from the radiator? INPUT YES OR NO");  
 gets(RADIATOR\_LEAK);  
 break;  
 case 12: printf("Is the leak coming from a radiator hose? INPUT YES OR NO");  
 gets(RADIATOR\_HOSE);  
 break;  
 case 13: printf("Is the leak coming from the transmission drain plug? INPUT YES OR NO");  
 gets(TRANS\_DRAIN\_PLUG);  
 break;  
 case 14: printf("Does the engine rotate when starting? INPUT YES OR NO");  
 gets(ENGINE\_ROTATES);  
 break;  
 case 15: printf("Are the battery terminals connected? INPUT YES OR NO");  
 gets(BATT\_TERMINALS);  
 break;  
 case 16: printf("Is there fuel in the tank? INPUT YES OR NO");  
 gets(FUEL);  
 break;  
 case 17: printf("What is the battery voltage? INPUT A REAL NUMBER");  
 std::cin >> BATT\_VOLTAGE;  
 gets(buff);  
 break;  
 case 18: printf("Is the starter motor working? INPUT YES OR NO");  
 gets(START\_MOTOR);  
 break;  
 case 19: printf("Is the car running roughly? INPUT YES OR NO");  
 gets(ROUGH);  
 break;  
 case 20: printf("Are the vacuum hoses worn? INPUT YES OR NO");  
 gets(VACUUM\_HOSE);  
 break;  
 case 21: printf("Do the spark plugs looked burned or corroded? INPUT YES OR NO");  
 gets(SPARK\_PLUGS);  
 break;  
 case 22: printf("Is there water in the fuel? INPUT YES OR NO");  
 gets(BAD\_FUEL);  
 break;  
 case 23: printf(" Do the injectors look clogged or corroded? INPUT YES OR NO");  
 gets(FUEL\_INJECTORS);  
 break;  
 case 24: printf("Do the ignition wired look worn or cracked? INPUT YES OR NO");  
 gets(IGNITION\_WIRES);  
 break;  
 case 25: printf("Is there a knocking noise coming from inside the engine? INPUT YES OR NO");  
 gets(KNOCKING);  
 break;  
 case 26: printf("Does the battery hold a charge? INPUT YES OR NO");  
 gets(BATT\_CHARGE);  
 break;  
 case 27: printf("Does the battery test ok? INPUT YES OR NO");  
 gets(BATT\_TEST);  
 break;  
 case 28: printf("Does the alternator test ok? INPUT YES OR NO");  
 gets(ALTERNATOR);  
 break;  
 case 29: printf("Does the alternator drive belt look worn? INPUT YES OR NO");  
 gets(ALT\_DRIVE\_BELT);  
 break;  
 case 30: printf("Are the battery wires worn or cracked? INPUT YES OR NO");  
 gets(BATT\_WIRES);  
 break;  
 case 31: printf("Does the car overheat? INPUT YES OR NO");  
 gets(OVERHEAT);  
 break;  
 case 32: printf("Is there coolant? INPUT YES OR NO");  
 gets(COOLANT);  
 break;  
 case 33: printf("Is the drive belt worn or stretched? INPUT YES OR NO");  
 gets(DRIVE\_BELT);  
 break;  
 case 34: printf("Does the radiator look dirty or corroded? INPUT YES OR NO");  
 gets(RADIATOR);  
 break;  
 case 35: printf("Does the radiator fan work? INPUT YES OR NO");  
 gets(RADIATOR\_FAN);  
 break;  
 case 36: printf("Is there excessive fuel consumption? INPUT YES OR NO");  
 gets(FUEL\_USE);  
 break;  
 case 37: printf("Is the tire pressure correct? INPUT YES OR NO");  
 gets(TIRE\_PRESSURE);  
 break;  
 case 38: printf("Does the air filter look clean? INPUT YES OR NO");  
 gets(AIR\_FILTER);  
 break;  
 case 39: printf("Can you smell fuel under the car?INPUT YES OR NO");  
 gets(FUEL\_SMELL);  
 break;  
 case 40: printf("Is the check brake light on? INPUT YES OR NO");  
 gets(CHECK\_BRAKE);  
 break;  
 case 41: printf("Is there a wobble or vibration when braking? INPUT YES OR NO");  
 gets(WOBBLE);  
 break;  
 case 42: printf("Is there a chemical burning smell when braking? INPUT YES OR NO");  
 gets(CHEMICAL\_SMELL);  
 break;  
 case 43: printf("Is braking spongey or soft? INPUT YES OR NO");  
 gets(SPONGEY);  
 break;  
 case 44: printf("Is there a squeaking noise when braking?INPUT YES OR NO");  
 gets(SQUEAK);  
 break;  
  
  
  
  
 */\*\*\*\*\* comment 1715 \*\*\*\*/* }  
 */\* end of inputs statements for sample position knowledge  
 base \*/* }  
}  
  
void bc::bcLoop(){  
 flag=1;  
 determine\_member\_concl\_list();  
  
 if (stateNum != 0) { */\* if stateNum = 0 then no conclusion of that name \*/  
 /\* push statement number (stateNum) and clause number=1 on goal  
 stack which is composed of the statement stack (stateStk)  
 and clause stack (clausStk) \*/* do  
 {  
 push\_on\_stack();  
 do  
 {  
 */\* calculate clause location in clause-variable  
 list \*/* chkClauses();  
 } while(strcmp(varble, "") != 0); */\*do-while\*/  
 /\*no more clauses check if part of statement \*/* stateNum = stateStk[sp];  
 s = 0;  
 */\*\*\*\* if then statements \*\*\*\*/  
 /\* sample if parts of if then statements from  
 the position knowledge base \*/* switch (stateNum) {  
 */\* if part of statement 1 \*/  
 /\*\*\*\*\*\* comment 1500 \*\*\*\*/* case 1: if(strcmp(PROBLEM, "NO") == 0) s = 1;  
 break;  
 */\* if part of statement 2 \*/  
 /\*\*\*\*\* comment 1510 \*\*\*\*\*\*/* case 2: if((strcmp(PROBLEM, "YES") == 0) &&  
 (strcmp(OIL\_PRESSURE, "YES") == 0)&&  
 (strcmp(OIL\_GRADE, "YES") == 0)&&  
 (strcmp(OIL\_FILTER, "YES") == 0)) s = 1;  
 break;  
 */\* if part of statement 3 \*/* case 3:  
 if ((strcmp(PROBLEM, "YES") == 0) &&  
 (strcmp(OIL\_PRESSURE, "YES") == 0) &&  
 (strcmp(OIL\_GRADE, "YES") == 0) &&  
 (strcmp(OIL\_FILTER, "NO") == 0)) s = 1;  
 break;  
 */\* if part of statement 4 \*/  
 /\*\*\*\*\*\*\*\* comment 1560 \*\*\*\*\*\*/* case 4: if((strcmp(PROBLEM, "YES") == 0) &&  
 (strcmp(OIL\_PRESSURE, "YES") == 0) &&  
 (strcmp(OIL\_GRADE, "NO") == 0)) s = 1;  
 break;  
 */\*\*\*\*\*\*\*\* comment 1570 \*\*\*\*\*\*\*\*/  
 /\* if part of statement 5 \*/* case 5: if((strcmp(PROBLEM, "YES") == 0) &&  
 (strcmp(LEAK, "YES") == 0) &&  
 (strcmp(OIL\_LEAK, "YES") == 0) &&  
 (strcmp(OIL\_DRAIN\_PLUG, "YES") == 0)) s = 1;  
 break;  
 */\* if part of statement 6 \*/* case 6: if((strcmp(PROBLEM, "YES") == 0) &&  
 (strcmp(LEAK, "YES") == 0) &&  
 (strcmp(OIL\_LEAK, "YES") == 0) &&  
 (strcmp(ENGINE\_LEAK, "YES") == 0)) s = 1;  
 break;  
 case 7: if((strcmp(PROBLEM, "YES") == 0) &&  
 (strcmp(LEAK, "YES") == 0) &&  
 (strcmp(COOLANT\_LEAK, "YES") == 0) &&  
 (strcmp(RADIATOR\_LEAK, "YES") == 0)) s = 1;  
 break;  
  
 case 8:  
 if ((strcmp(PROBLEM, "YES") == 0) &&  
 (strcmp(LEAK, "YES") == 0) &&  
 (strcmp(COOLANT\_LEAK, "YES") == 0) &&  
 (strcmp(RADIATOR\_HOSE, "YES") == 0)) s = 1;  
 break;  
  
 case 9: if((strcmp(PROBLEM, "YES") == 0) &&  
 (strcmp(LEAK, "YES") == 0) &&  
 (strcmp(TRANS\_LEAK, "YES") == 0) &&  
 (strcmp(TRANS\_DRAIN\_PLUG, "YES") == 0)) s = 1;  
 break;  
  
 case 10: if((strcmp(PROBLEM, "YES") == 0) &&  
 (strcmp(ENGINE\_ROTATES, "NO") == 0) &&  
 (strcmp(BATT\_TERMINALS, "YES") == 0) &&  
 (BATT\_VOLTAGE<12)) s = 1;  
 break;  
  
 case 11: if((strcmp(PROBLEM, "YES") == 0) &&  
 (strcmp(ENGINE\_ROTATES, "NO") == 0) &&  
 (strcmp(BATT\_TERMINALS, "YES") == 0) &&  
 (BATT\_VOLTAGE >= 12.0) &&  
 (strcmp(START\_MOTOR, "YES") == 0) )s = 1;  
 break;  
 case 12: if((strcmp(PROBLEM, "YES") == 0) &&  
 (strcmp(ENGINE\_ROTATES, "NO") == 0) &&  
 (strcmp(BATT\_TERMINALS, "YES") == 0) &&  
 (BATT\_VOLTAGE>=12)&&  
 (strcmp(START\_MOTOR, "NO") == 0) )s = 1;  
  
 case 13:  
 if ((strcmp(PROBLEM, "YES") == 0) &&  
 (strcmp(ENGINE\_ROTATES, "NO") == 0) &&  
 (strcmp(BATT\_TERMINALS, "NO") == 0)) s = 1;  
 break;  
  
 case 14: if((strcmp(PROBLEM, "YES") == 0) &&  
 (strcmp(ENGINE\_ROTATES, "YES") == 0) &&  
 (strcmp(FUEL, "NO") == 0)) s = 1;  
 break;  
  
 case 15: if((strcmp(PROBLEM, "YES") == 0) &&  
 (strcmp(ENGINE\_ROTATES, "YES") == 0) &&  
 (strcmp(FUEL, "YES") == 0)) s = 1;  
 break;  
  
 case 16: if((strcmp(PROBLEM, "YES") == 0) &&  
 (strcmp(ROUGH, "YES") == 0) &&  
 (strcmp(VACUUM\_HOSE, "YES") == 0)) s = 1;  
 break;  
 case 17: if((strcmp(PROBLEM, "YES") == 0) &&  
 (strcmp(ROUGH, "YES") == 0) &&  
 (strcmp(SPARK\_PLUGS, "YES") == 0)) s = 1;  
 break;  
  
 case 18:  
 if((strcmp(PROBLEM, "YES") == 0) &&  
 (strcmp(ROUGH, "YES") == 0) &&  
 (strcmp(BAD\_FUEL, "YES") == 0)) s = 1;  
 break;  
  
 case 19: if((strcmp(PROBLEM, "YES") == 0) &&  
 (strcmp(ROUGH, "YES") == 0) &&  
 (strcmp(FUEL\_INJECTORS, "YES") == 0)) s = 1;  
 break;  
  
 case 20: if((strcmp(PROBLEM, "YES") == 0) &&  
 (strcmp(ROUGH, "YES") == 0) &&  
 (strcmp(IGNITION\_WIRES, "YES") == 0)) s = 1;  
 break;  
  
 case 21: if((strcmp(PROBLEM, "YES") == 0) &&  
 (strcmp(ROUGH, "YES") == 0) &&  
 (strcmp(KNOCKING, "YES") == 0)) s = 1;  
 break;  
 case 22: if((strcmp(PROBLEM, "YES") == 0) &&  
 (strcmp(BATT\_CHARGE, "NO") == 0) &&  
 (strcmp(BATT\_TEST, "NO") == 0)) s = 1;  
 break;  
  
 case 23:  
 if((strcmp(PROBLEM, "YES") == 0) &&  
 (strcmp(BATT\_CHARGE, "NO") == 0) &&  
 (strcmp(BATT\_TEST, "YES") == 0) &&  
 (strcmp(ALTERNATOR, "NO") == 0)) s = 1;  
 break;  
  
 case 24: if((strcmp(PROBLEM, "YES") == 0) &&  
 (strcmp(BATT\_CHARGE, "NO") == 0) &&  
 (strcmp(BATT\_TEST, "YES") == 0) &&  
 (strcmp(ALTERNATOR, "YES") == 0)&&  
 (strcmp(ALT\_DRIVE\_BELT, "YES") == 0)) s = 1;  
 break;  
  
 case 25: if((strcmp(PROBLEM, "YES") == 0) &&  
 (strcmp(BATT\_CHARGE, "NO") == 0) &&  
 (strcmp(BATT\_TEST, "YES") == 0) &&  
 (strcmp(ALTERNATOR, "YES") == 0)&&  
 (strcmp(ALT\_DRIVE\_BELT, "YES") == 0)&&  
 (strcmp(BATT\_WIRES, "YES") == 0)) s = 1;  
 break;  
  
 case 26: if((strcmp(PROBLEM, "YES") == 0) &&  
 (strcmp(BATT\_CHARGE, "NO") == 0) &&  
 (strcmp(BATT\_TEST, "YES") == 0) &&  
 (strcmp(ALTERNATOR, "YES") == 0)&&  
 (strcmp(ALT\_DRIVE\_BELT, "YES") == 0)&&  
 (strcmp(BATT\_WIRES, "NO") == 0)) s = 1;  
 break;  
 case 27: if((strcmp(PROBLEM, "YES") == 0) &&  
 (strcmp(OVERHEAT, "YES") == 0) &&  
 (strcmp(COOLANT, "NO") == 0)) s = 1;  
 break;  
  
 case 28: if((strcmp(PROBLEM, "YES") == 0) &&  
 (strcmp(OVERHEAT, "YES") == 0) &&  
 (strcmp(OVERHEAT, "YES") == 0) &&  
 (strcmp(DRIVE\_BELT, "YES") == 0)) s = 1;  
 break;  
  
 case 29: if((strcmp(PROBLEM, "YES") == 0) &&  
 (strcmp(OVERHEAT, "YES") == 0) &&  
 (strcmp(COOLANT, "YES") == 0)&&  
 (strcmp(DRIVE\_BELT, "NO") == 0)&&  
 (strcmp(RADIATOR, "YES") == 0)) s = 1;  
 break;  
 case 30: if((strcmp(PROBLEM, "YES") == 0) &&  
 (strcmp(OVERHEAT, "YES") == 0) &&  
 (strcmp(COOLANT, "YES") == 0)&&  
 (strcmp(DRIVE\_BELT, "NO") == 0)&&  
 (strcmp(RADIATOR, "NO") == 0)&&  
 (strcmp(RADIATOR\_FAN, "NO") == 0)) s = 1;  
 break;  
  
 case 31:  
 if((strcmp(PROBLEM, "YES") == 0) &&  
 (strcmp(OVERHEAT, "YES") == 0) &&  
 (strcmp(COOLANT, "YES") == 0)&&  
 (strcmp(DRIVE\_BELT, "NO") == 0)&&  
 (strcmp(RADIATOR, "NO") == 0)&&  
 (strcmp(RADIATOR\_FAN, "YES") == 0)) s = 1;  
 break;  
  
 case 32: if((strcmp(PROBLEM, "YES") == 0) &&  
 (strcmp(FUEL\_USE, "YES") == 0) &&  
 (strcmp(TIRE\_PRESSURE, "NO") == 0)) s = 1;  
 break;  
  
 case 33: if((strcmp(PROBLEM, "YES") == 0) &&  
 (strcmp(FUEL\_USE, "YES") == 0) &&  
 (strcmp(TIRE\_PRESSURE, "YES") == 0) &&  
 (strcmp(AIR\_FILTER, "NO") == 0)) s = 1;  
 break;  
  
 case 34: if((strcmp(PROBLEM, "YES") == 0) &&  
 (strcmp(FUEL\_USE, "YES") == 0) &&  
 (strcmp(TIRE\_PRESSURE, "YES") == 0) &&  
 (strcmp(AIR\_FILTER, "YES") == 0)&&  
 (strcmp(FUEL\_SMELL, "YES") == 0)) s = 1;  
 break;  
 case 35: if((strcmp(PROBLEM, "YES") == 0) &&  
 (strcmp(FUEL\_USE, "YES") == 0) &&  
 (strcmp(TIRE\_PRESSURE, "YES") == 0) &&  
 (strcmp(AIR\_FILTER, "YES") == 0)&&  
 (strcmp(FUEL\_SMELL, "NO") == 0)) s = 1;  
 break;  
  
 case 36: if((strcmp(PROBLEM, "YES") == 0) &&  
 (strcmp(CHECK\_BRAKE, "YES") == 0) &&  
 (strcmp(WOBBLE, "YES") == 0)) s = 1;  
 break;  
 case 37: if((strcmp(PROBLEM, "YES") == 0) &&  
 (strcmp(CHECK\_BRAKE, "YES") == 0) &&  
 (strcmp(CHEMICAL\_SMELL, "YES") == 0)) s = 1;  
 break;  
  
 case 38:if((strcmp(PROBLEM, "YES") == 0) &&  
 (strcmp(CHECK\_BRAKE, "YES") == 0) &&  
 (strcmp(SPONGEY, "YES") == 0)) s = 1;  
 break;  
 case 39: if((strcmp(PROBLEM, "YES") == 0) &&  
 (strcmp(CHECK\_BRAKE, "YES") == 0) &&  
 (strcmp(SQUEAK, "YES") == 0)) s = 1;  
 break;  
  
 }  
 */\* see if the then part should be invoked \*/* if( s != 1) {  
 */\* failed..search rest of statements for  
 same conclusion \*/  
 /\* get conclusion \*/* i = stateStk[sp];  
 strcpy(varble, concList[i]);  
 */\* search for conclusion starting at the  
 next statement number \*/* flag = stateStk[sp] + 1;  
 determine\_member\_concl\_list();  
 sp = sp+1;  
 }  
 */\* pop old conclusion and put on new one \*/* } while((s != 1) && (stateNum !=0)); */\* outer do-while loop \*/* if(stateNum != 0){  
 */\* if part true invoke then part \*/  
 /\* then part of if-then statements from the  
 position knowledge base \*/* switch (stateNum) {  
 */\* then part of statement 1 \*/  
 /\*\*\*\*\*\*\* comment 1500 \*\*\*\*\*\*\*/* case 1: strcpy(PROBLEM, "NO");  
 printf("NO ISSUE\n");  
 strcpy(end, "NO ISSUE");  
 break;  
 */\* then part of statement 2 \*/  
 /\*\*\*\*\*\* comment 1510 \*\*\*\*\*\*/* case 2: strcpy(PART, "YES");  
 printf("PART = OIL PUMP\n");  
 strcpy(end, "OIL\_PUMP");  
 break;  
 */\* then part of statement 3 \*/* case 3: strcpy(PART, "YES");  
 printf("PART = OIL FILTER\n");  
 strcpy(end, "OIL\_FILTER");  
 break;  
 */\* then part of statement 4 \*/  
 /\*\*\*\*\*\*\*\* comment 1560 \*\*\*\*\*\*/* case 4: strcpy(PART, "YES");  
 printf("OIL\n");  
 strcpy(end, "OIL");  
 break;  
 */\* then part of statement 5 \*/  
 /\*\*\*\*\*\* comment 1570 \*\*\*\*\*/* case 5: strcpy(PART, "YES");  
 printf("OIL DRAIN PLUG\n");  
 strcpy(end, "OIL\_DRAIN\_PLUG");  
 break;  
  
 case 6: strcpy(PART, "YES");  
 printf("ENGINE GASKET\n");  
 strcpy(end, "ENGINE\_GASKET");  
 break;  
 case 7: strcpy(PART, "YES");  
 printf("RADIATOR\n");  
 strcpy(end,"RADIATOR");  
 break;  
  
 case 8: strcpy(PART, "YES");  
 printf("RADIATOR HOSE\n");  
 strcpy(end,"RADIATOR\_HOSE");  
 break;  
  
 case 9: strcpy(PART, "YES");  
 printf("TRANSMISSION DRAIN PLUG\n");  
 strcpy(end,"TRANSMISSION\_DRAIN\_PLUG");  
 break;  
  
 case 10: strcpy(PART, "YES");  
 printf("BATTERY\n");  
 strcpy(end,"BATTERY");  
 break;  
  
 case 11: strcpy(PART, "YES");  
 printf("IGNITION SWITCH\n");  
 strcpy(end,"IGNITION\_SWITCH");  
 break;  
 case 12: strcpy(PART, "YES");  
 printf("STARTER MOTOR\n");  
 strcpy(end,"STARTER\_MOTOR");  
 break;  
  
 case 13: strcpy(PART, "YES");  
 printf("BATTERY TERMINALS\n");  
 strcpy(end,"BATTERY\_TERMINALS");  
 break;  
  
 case 14: strcpy(PART, "YES");  
 printf("FUEL LEVEL\n");  
 strcpy(end,"FUEL\_LEVEL");  
 break;  
  
 case 15: strcpy(PART, "YES");  
 printf("FUEL SYSTEM\n");  
 strcpy(end,"FUEL\_SYSTEM");  
 break;  
  
 case 16: strcpy(PART, "YES");  
 printf("VACUUM HOSE\n");  
 strcpy(end,"VACUUM\_HOSE");  
 break;  
 case 17: strcpy(PART, "YES");  
 printf("SPARK PLUGS\n");  
 strcpy(end,"SPARK\_PLUGS");  
 break;  
  
 case 18: strcpy(PART, "YES");  
 printf("FUEL CONDITION\n");  
 strcpy(end,"FUEL\_CONDITION");  
 break;  
  
 case 19: strcpy(PART, "YES");  
 printf("FUEL INJECTORS\n");  
 strcpy(end,"FUEL\_INJECTORS");  
 break;  
  
 case 20: strcpy(PART, "YES");  
 printf("IGNITION WIRES\n");  
 strcpy(end,"IGNITION\_WIRES");  
 break;  
  
 case 21: strcpy(PART, "YES");  
 printf("ENGINE COMPONENTS\n");  
 strcpy(end,"ENGINE\_COMPONENTS");  
 break;  
 case 22: strcpy(PART, "YES");  
 printf("BATTERY\n");  
 strcpy(end,"BATTERY");  
 break;  
  
 case 23: strcpy(PART, "YES");  
 printf("ALTERNATOR\n");  
 strcpy(end,"ALTERNATOR");  
 break;  
  
 case 24: strcpy(PART, "YES");  
 printf("ALTERNATOR DRIVE BELT\n");  
 strcpy(end,"ALTERNATOR\_DRIVE\_BELT");  
 break;  
  
 case 25: strcpy(PART, "YES");  
 printf("BATTERY WIRES\n");  
 strcpy(end,"BATTERY\_WIRES");  
 break;  
  
 case 26: strcpy(PART, "YES");  
 printf("ELECTRICAL SYSTEM\n");  
 strcpy(end,"ELECTRICAL\_SYSTEM");  
 break;  
 case 27: strcpy(PART, "YES");  
 printf("COOLANT\n");  
 strcpy(end,"COOLANT");  
 break;  
  
 case 28: strcpy(PART, "YES");  
 printf("DRIVE BELT\n");  
 strcpy(end,"DRIVE\_BELT");  
 break;  
  
 case 29: strcpy(PART, "YES");  
 printf("RADIATOR\n");  
 strcpy(end,"RADIATOR");  
 break;  
  
 case 30: strcpy(PART, "YES");  
 printf("RADIATOR FAN\n");  
 strcpy(end,"RADIATOR\_FAN");  
 break;  
  
 case 31: strcpy(PART, "YES");  
 printf("WATTER PUMP\n");  
 strcpy(end,"WATTER\_PUMP");  
 break;  
 case 32: strcpy(PART, "YES");  
 printf("TIRE PRESSURE\n");  
 strcpy(end,"TIRE\_PRESSURE");  
 break;  
 case 33: strcpy(PART, "YES");  
 printf("AIR FILTER\n");  
 strcpy(end,"AIR\_FILTER");  
 break;  
 case 34: strcpy(PART, "YES");  
 printf("FUEL TANK\n");  
 strcpy(end,"FUEL\_TANK");  
 break;  
 case 35: strcpy(PART, "YES");  
 printf("FUEL MANAGEMENT SYSTEM\n");  
 strcpy(end,"FUEL\_MANAGEMENT\_SYSTEM");  
 break;  
 case 36: strcpy(PART, "YES");  
 printf("BRAKE PARTS\n");  
 strcpy(end,"BRAKE\_PARTS");  
 break;  
 case 37: strcpy(PART, "YES");  
 printf("BRAKE PARTS\n");  
 strcpy(end,"BRAKE\_PARTS");  
 break;  
 case 38: strcpy(PART, "YES");  
 printf("BRAKE LINES\n");  
 strcpy(end,"BRAKE\_LINES");  
 break;  
 case 39: strcpy(PART, "YES");  
 printf("BRAKE PARTS\n");  
 strcpy(end,"BRAKE\_PARTS");  
 break;  
 }  
  
 popStack();  
 }  
 }  
}  
  
void bc::popStack()  
{  
 sp=sp+1;  
 if(sp >= 45) */\* finished \*/* printf("\*\*\* SUCCESS\n");  
 else {  
 */\* stack is not empty \*/  
 /\* get next clause then continue \*/* clausStk[sp] = clausStk[sp]+1;  
 chkClauses();  
 }  
}  
  
void bc::chkClauses()  
{  
 i = (stateStk[sp] - 1) \* 44 + clausStk[sp];  
 */\* clause variable \*/* strcpy(varble, clVarList[i]);  
 if (strcmp(varble, "") != 0) {  
 */\*is this clause variable a conclusion? \*/* flag = 1;  
 determine\_member\_concl\_list();  
 if (stateNum != 0)  
 */\* it is a conclusion push it \*/* bcLoop();  
 */\* check instantiation of this clause \*/* instantiate();  
 clausStk[sp] = clausStk[sp] + 1;  
 }  
}

Fc.cpp:

#include<string>  
#include<cstdio>  
#include<cstring>  
#include<iostream>  
  
#include "fc.h"  
  
using namespace std;  
  
fc::fc(string problem) {  
 strcpy(cndVar[1], problem.c\_str());  
}  
  
  
char \* fc::start() {  
 */\*\*\*\*\*\*\*\* INITIALIZATION SECTION \*\*\*\*\*\*\*\*\*\*\*/* fp = 1;  
 bp = 1;  
  
 for (i = 1; i < 1676; i++)  
 strcpy(clVarList[i], "");  
 for (i = 2; i < 40; i++)  
 strcpy(cndVar[i], "");  
 for (i = 1; i < 11; i++) {  
 instList[i] = 0;  
 }  
 for (i = 1; i < 45; i++)  
 strcpy(varList[i], "");  
 for (i = 1; i < 17; i++) {  
 instList[i] = 0;  
 }  
  
 strcpy(varList[1], "OIL\_PUMP");  
 strcpy(varList[2], "OIL\_FILTER");  
 strcpy(varList[3], "OIL");  
 strcpy(varList[4], "OIL\_DRAIN\_PLUG");  
 strcpy(varList[5], "ENGINE\_GASKET");  
 strcpy(varList[6], "RADIATOR");  
 strcpy(varList[7], "RADIATOR\_HOSE");  
 strcpy(varList[8], "TRANSMISSION\_DRAIN\_PLUG");  
 strcpy(varList[9], "BATTERY");  
 strcpy(varList[10], "IGNITION\_SWITCH");  
 strcpy(varList[11], "STARTER\_MOTOR");  
 strcpy(varList[12], "BATTERY\_TERMINALS");  
 strcpy(varList[13], "FUEL\_LEVEL");  
 strcpy(varList[14], "FUEL\_CONDITION");  
 strcpy(varList[15], "FUEL\_SYSTEM");  
 strcpy(varList[16], "VACUUM\_HOSE");  
 strcpy(varList[17], "SPARK\_PLUGS");  
 strcpy(varList[18], "FUEL\_INJECTORS");  
 strcpy(varList[19], "IGNITION\_WIRES");  
 strcpy(varList[20], "ENGINE\_COMPONENTS");  
 strcpy(varList[21], "ALTERNATOR");  
 strcpy(varList[22], "ALTERNATOR\_BELT");  
 strcpy(varList[23], "BATTERY\_WIRES");  
 strcpy(varList[24], "ELECTRICAL\_SYSTEM");  
 strcpy(varList[25], "COOLANT");  
 strcpy(varList[26], "DRIVE\_BELT");  
 strcpy(varList[27], "RADIATOR\_FAN");  
 strcpy(varList[28], "WATER\_PUMP");  
 strcpy(varList[29], "TIRE\_PRESSURE");  
 strcpy(varList[30], "AIR\_FILTER");  
 strcpy(varList[31], "FUEL\_TANK");  
 strcpy(varList[32], "FUEL\_MANAGEMENT\_SYSTEM");  
 strcpy(varList[33], "BRAKE\_PARTS");  
 strcpy(varList[34], "BRAKE\_LINES");  
 strcpy(varList[35], "REPLACE");  
 strcpy(varList[36], "RECONNECT");  
  
 printf("\*\*\* VARIABLE LIST \*\*\*\n");  
 for (i = 1; i < 37; i++)  
 printf("ENTER VARIABLE %d %s\n", i, varList[i]);  
 printf("HIT RETURN TO CONTINUE");  
 getchar();  
  
 printf("\*\*\* CLAUSE-VARIABLE LIST \*\*\*\n");  
 strcpy(clVarList[1], "OIL\_PUMP");  
 strcpy(clVarList[39], "OIL\_FILTER");  
 strcpy(clVarList[77], "OIL");  
 strcpy(clVarList[115], "OIL\_DRAIN\_PLUG");  
 strcpy(clVarList[153], "ENGINE\_GASKET");  
 strcpy(clVarList[191], "RADIATOR");  
 strcpy(clVarList[229], "RADIATOR\_HOSE");  
 strcpy(clVarList[267], "TRANSMISSION\_DRAIN\_PLUG");  
 strcpy(clVarList[305], "BATTERY");  
 strcpy(clVarList[343], "IGNITION\_SWITCH");  
 strcpy(clVarList[381], "STARTER\_MOTOR");  
 strcpy(clVarList[419], "BATTERY\_TERMINALS");  
 strcpy(clVarList[457], "FUEL\_LEVEL");  
 strcpy(clVarList[495], "FUEL\_CONDITION");  
 strcpy(clVarList[533], "FUEL\_SYSTEM");  
 strcpy(clVarList[571], "VACUUM\_HOSE");  
 strcpy(clVarList[609], "SPARK\_PLUGS");  
 strcpy(clVarList[647], "FUEL\_INJECTORS");  
 strcpy(clVarList[685], "IGNITION\_WIRES");  
 strcpy(clVarList[723], "ENGINE\_COMPONENTS");  
 strcpy(clVarList[761], "BATTERY");  
 strcpy(clVarList[799], "ALTERNATOR");  
 strcpy(clVarList[837], "ALTERNATOR\_BELT");  
 strcpy(clVarList[875], "BATTERY\_WIRES");  
 strcpy(clVarList[913], "ELECTRICAL\_SYSTEM");  
 strcpy(clVarList[951], "COOLANT");  
 strcpy(clVarList[989], "DRIVE\_BELT");  
 strcpy(clVarList[1027], "RADIATOR");  
 strcpy(clVarList[1065], "RADIATOR\_FAN");  
 strcpy(clVarList[1103], "WATER\_PUMP");  
 strcpy(clVarList[1141], "TIRE\_PRESSURE");  
 strcpy(clVarList[1179], "AIR\_FILTER");  
 strcpy(clVarList[1217], "FUEL\_TANK");  
 strcpy(clVarList[1255], "FUEL\_MANAGEMENT\_SYSTEM");  
 strcpy(clVarList[1293], "BRAKE\_PARTS");  
 strcpy(clVarList[1331], "BRAKE\_PARTS");  
 strcpy(clVarList[1369], "BRAKE\_LINES");  
 strcpy(clVarList[1407], "BRAKE\_PARTS");  
  
 for (i = 1; i < 38; i++) {  
 printf("\*\* CLAUSE %d\n", i);  
 for (j = 1; j < 38; j++) {  
 k = 38 \* (i - 1) + j;  
 printf("VARIABLE %d %s\n", j, clVarList[k]);  
 }  
 }  
  
 */\*\*\*\*\*\* INFERENCE SECTION \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
 /\* move backpointer (bp) to back \*/* bp = bp + 1;  
 */\* set the condition variable pointer consisting of the  
 statement number (sn) and the clause number (cn) \*/* sn = 1;  
 cn = 1;  
 */\* find the next statement number containing the condition variable  
 which is in front of the queue (cndVar), this statement number  
 is located in the clause variable list (clVarList) \*/  
 /\* start at the beginning \*/* flag = 1;  
 fcLoop();  
 return end;  
}  
  
void fc::fcLoop(){  
 bool exit = false;  
 while(! exit) {  
 searchClauses(); *//return with a statement number* cn = 1; */\* point to first clause in statement \*/* if (sn != 0)  
 {  
 i = 38 \* (sn - 1) + cn; */\* locate the clause \*/* strcpy(v, clVarList[i]); */\* clause variable \*/* while (strcmp(v, "") != 0) */\* more clauses \*/* {  
 */\* check instantiation of this clause \*/* check\_instantiation();  
 cn = cn + 1;  
 */\* check next clause \*/* i = 38 \* (sn - 1) + cn;  
 strcpy(v, clVarList[i]);  
 }  
  
 */\* no more clauses - check IF part of statement \*/* s = 0;  
 */\* sample IF-THEN statements from the position knowledge base \*/* switch (sn) {  
 case 1:  
 if (strcmp(OIL\_PUMP, "BAD") == 0) s = 1;  
 break;  
 case 2:  
 if (strcmp(OIL\_FILTER, "BAD") == 0) s = 1;  
 break;  
 case 3:  
 if (strcmp(OIL, "BAD") == 0) s = 1;  
 break;  
 case 4:  
 if (strcmp(OIL\_DRAIN\_PLUG, "BAD") == 0) s = 1;  
 break;  
 case 5:  
 if (strcmp(ENGINE\_GASKET, "BAD") == 0)s = 1;  
 break;  
 case 6:  
 if (strcmp(RADIATOR, "BAD") == 0) s = 1;  
 break;  
 case 7:  
 if (strcmp(RADIATOR\_HOSE, "BAD") == 0) s = 1;  
 break;  
 case 8:  
 if (strcmp(TRANSMISSION\_DRAIN\_PLUG, "BAD") == 0) s = 1;  
 break;  
 case 9:  
 if (strcmp(BATTERY, "BAD") == 0) s = 1;  
 break;  
 case 10:  
 if (strcmp(IGNITION\_SWITCH, "BAD") == 0) s = 1;  
 break;  
 case 11:  
 if (strcmp(STARTER\_MOTOR, "BAD") == 0) s = 1;  
 break;  
 case 12:  
 if (strcmp(BATTERY\_TERMINALS, "BAD") == 0) s = 1;  
 break;  
 case 13:  
 if (strcmp(FUEL\_LEVEL, "BAD") == 0) s = 1;  
 break;  
 case 14:  
 if (strcmp(FUEL\_CONDITION, "BAD") == 0) s = 1;  
 break;  
 case 15:  
 if (strcmp(FUEL\_SYSTEM, "BAD") == 0) s = 1;  
 break;  
 case 16:  
 if (strcmp(VACUUM\_HOSE, "BAD") == 0) s = 1;  
 break;  
 case 17:  
 if (strcmp(SPARK\_PLUGS, "BAD") == 0) s = 1;  
 break;  
 case 18:  
 if (strcmp(FUEL\_INJECTORS, "BAD") == 0) s = 1;  
 break;  
 case 19:  
 if (strcmp(IGNITION\_WIRES, "BAD") == 0) s = 1;  
 break;  
 case 20:  
 if (strcmp(ENGINE\_COMPONENTS, "BAD") == 0) s = 1;  
 break;  
 case 21:  
 if (strcmp(BATTERY, "BAD") == 0) s = 1;  
 break;  
 case 22:  
 if (strcmp(ALTERNATOR, "BAD") == 0) s = 1;  
 break;  
 case 23:  
 if (strcmp(ALTERNATOR\_BELT, "BAD") == 0) s = 1;  
 break;  
 case 24:  
 if (strcmp(BATTERY\_WIRES, "BAD") == 0) s = 1;  
 break;  
 case 25:  
 if (strcmp(ELECTRICAL\_SYSTEM, "BAD") == 0) s = 1;  
 break;  
 case 26:  
 if (strcmp(COOLANT, "BAD") == 0) s = 1;  
 break;  
 case 27:  
 if (strcmp(DRIVE\_BELT, "BAD") == 0) s = 1;  
 break;  
 case 28:  
 if (strcmp(RADIATOR\_FAN, "BAD") == 0) s = 1;  
 break;  
 case 29:  
 if (strcmp(WATER\_PUMP, "BAD") == 0) s = 1;  
 break;  
 case 30:  
 if (strcmp(TIRE\_PRESSURE, "BAD") == 0) s = 1;  
 break;  
 case 31:  
 if (strcmp(AIR\_FILTER, "BAD") == 0) s = 1;  
 break;  
 case 32:  
 if (strcmp(FUEL\_TANK, "BAD") == 0) s = 1;  
 break;  
 case 33:  
 if (strcmp(FUEL\_MANAGEMENT\_SYSTEM, "BAD") == 0) s = 1;  
 break;  
 case 34:  
 if (strcmp(BRAKE\_PARTS, "BAD") == 0) s = 1;  
 break;  
 case 35:  
 if (strcmp(BRAKE\_LINES, "BAD") == 0) s = 1;  
 break;  
 }  
  
  
 */\* see if the THEN part should be inovked, i.e., s=1 \*/* if (s != 1) {  
 flag = sn + 1;  
 fcLoop();  
 continue;  
 }  
 else {  
 exit = true;  
 }  
 */\* invoke THEN part \*/* switch (sn) {  
  
 */\* put variable on the conclusion variable queue \*/* case 1:  
 strcpy(REPLACE, "YES");  
 printf("Replace = YES\n");  
 strcpy(v, "Replace");  
 strcpy(end, "Replace");  
 instantiate();  
 break;  
 */\* put variable on the conclusion variable queue \*/* case 2:  
 strcpy(REPLACE, "YES");  
 printf("Replace = YES\n");  
 strcpy(v, "Replace");  
 strcpy(end, "Replace");  
 instantiate();  
 break;  
 */\* put variable on the conclusion variable queue \*/* case 3:  
 strcpy(REPLACE, "YES");  
 printf("Replace = YES\n");  
 strcpy(v, "Replace");  
 strcpy(end, "Replace");  
 instantiate();  
 break;  
 */\* put variable on the conclusion variable queue \*/* case 4:  
 strcpy(REPLACE, "YES");  
 printf("Replace = YES\n");  
 strcpy(v, "Replace");  
 strcpy(end, "Replace");  
 instantiate();  
 break;  
 */\* put variable on the conclusion variable queue \*/* case 5:  
 strcpy(REPLACE, "YES");  
 printf("Replace = YES\n");  
 strcpy(v, "Replace");  
 strcpy(end, "Replace");  
 instantiate();  
 break;  
 case 6:  
 strcpy(REPLACE, "YES");  
 printf("Replace = YES\n");  
 strcpy(v, "Replace");  
 strcpy(end, "Replace");  
 instantiate();  
 break;  
 case 7:  
 strcpy(REPLACE, "YES");  
 printf("Replace = YES\n");  
 strcpy(v, "Replace");  
 strcpy(end, "Replace");  
 instantiate();  
 break;  
 case 8:  
 strcpy(REPLACE, "YES");  
 printf("Replace = YES\n");  
 strcpy(v, "Replace");  
 strcpy(end, "Replace");  
 instantiate();  
 break;  
 case 9:  
 strcpy(REPLACE, "YES");  
 printf("Replace = YES\n");  
 strcpy(v, "Replace");  
 strcpy(end, "Replace");  
 instantiate();  
 break;  
 case 10:  
 strcpy(REPLACE, "YES");  
 printf("Replace = YES\n");  
 strcpy(v, "Replace");  
 strcpy(end, "Replace");  
 instantiate();  
 break;  
  
 case 11:  
 strcpy(REPLACE, "YES");  
 printf("Replace = YES\n");  
 strcpy(v, "Replace");  
 strcpy(end, "Replace");  
 instantiate();  
 break;  
 case 12:  
 strcpy(REPLACE, "YES");  
 printf("Replace = YES\n");  
 strcpy(v, "Replace");  
 strcpy(end, "Replace");  
 instantiate();  
 break;  
 case 13:  
 strcpy(REPLACE, "YES");  
 printf("Replace = YES\n");  
 strcpy(v, "Replace");  
 strcpy(end, "Replace");  
 instantiate();  
 break;  
  
 case 14:  
 strcpy(REPLACE, "YES");  
 printf("Replace = YES\n");  
 strcpy(v, "Replace");  
 strcpy(end, "Replace");  
 instantiate();  
 break;  
  
 case 15:  
 strcpy(REPLACE, "YES");  
 printf("Replace = YES\n");  
 strcpy(v, "Replace");  
 strcpy(end, "Replace");  
 instantiate();  
 break;  
  
 case 16:  
 strcpy(REPLACE, "YES");  
 printf("Replace = YES\n");  
 strcpy(v, "Replace");  
 strcpy(end, "Replace");  
 instantiate();  
 break;  
  
 case 17:  
 strcpy(REPLACE, "YES");  
 printf("Replace = YES\n");  
 strcpy(v, "Replace");  
 strcpy(end, "Replace");  
 instantiate();  
 break;  
 case 18:  
 strcpy(REPLACE, "YES");  
 printf("Replace = YES\n");  
 strcpy(v, "Replace");  
 strcpy(end, "Replace");  
 instantiate();  
 break;  
 case 19:  
 strcpy(REPLACE, "YES");  
 printf("Replace = YES\n");  
 strcpy(v, "Replace");  
 strcpy(end, "Replace");  
 instantiate();  
 break;  
  
 case 20:  
 strcpy(REPLACE, "YES");  
 printf("Replace = YES\n");  
 strcpy(v, "Replace");  
 strcpy(end, "Replace");  
 instantiate();  
 break;  
  
 case 21:  
 strcpy(REPLACE, "YES");  
 printf("Replace = YES\n");  
 strcpy(v, "Replace");  
 strcpy(end, "Replace");  
 instantiate();  
 break;  
  
 case 22:  
 strcpy(REPLACE, "YES");  
 printf("Replace = YES\n");  
 strcpy(v, "Replace");  
 strcpy(end, "Replace");  
 instantiate();  
 break;  
  
 case 23:  
 strcpy(REPLACE, "YES");  
 printf("Replace = YES\n");  
 strcpy(v, "Replace");  
 strcpy(end, "Replace");  
 instantiate();  
 break;  
 case 24:  
 strcpy(REPLACE, "YES");  
 printf("Replace = YES\n");  
 strcpy(v, "Replace");  
 strcpy(end, "Replace");  
 instantiate();  
 break;  
 case 25:  
 strcpy(REPLACE, "YES");  
 printf("Replace = YES\n");  
 strcpy(v, "Replace");  
 strcpy(end, "Replace");  
 instantiate();  
 break;  
  
 case 26:  
 strcpy(REPLACE, "YES");  
 printf("Replace = YES\n");  
 strcpy(v, "Replace");  
 strcpy(end, "Replace");  
 instantiate();  
 break;  
  
 case 27:  
 strcpy(REPLACE, "YES");  
 printf("Replace = YES\n");  
 strcpy(v, "Replace");  
 strcpy(end, "Replace");  
 instantiate();  
 break;  
  
 case 28:  
 strcpy(REPLACE, "YES");  
 printf("Replace = YES\n");  
 strcpy(v, "Replace");  
 strcpy(end, "Replace");  
 instantiate();  
 break;  
  
 case 29:  
 strcpy(REPLACE, "YES");  
 printf("Replace = YES\n");  
 strcpy(v, "Replace");  
 strcpy(end, "Replace");  
 instantiate();  
 break;  
 case 30:  
 strcpy(REPLACE, "YES");  
 printf("Replace = YES\n");  
 strcpy(v, "Replace");  
 strcpy(end, "Replace");  
 instantiate();  
 break;  
 case 31:  
 strcpy(REPLACE, "YES");  
 printf("Replace = YES\n");  
 strcpy(v, "Replace");  
 strcpy(end, "Replace");  
 instantiate();  
 break;  
  
 case 32:  
 strcpy(REPLACE, "YES");  
 printf("Replace = YES\n");  
 strcpy(v, "Replace");  
 strcpy(end, "Replace");  
 instantiate();  
 break;  
  
 case 33:  
 strcpy(REPLACE, "YES");  
 printf("Replace = YES\n");  
 strcpy(v, "Replace");  
 strcpy(end, "Replace");  
 instantiate();  
 break;  
  
 case 34:  
 strcpy(REPLACE, "YES");  
 printf("Replace = YES\n");  
 strcpy(v, "Replace");  
 strcpy(end, "Replace");  
 instantiate();  
 break;  
  
 case 35:  
 strcpy(REPLACE, "YES");  
 printf("Replace = YES\n");  
 strcpy(v, "Replace");  
 strcpy(end, "Replace");  
 instantiate();  
 break;  
 case 36:  
 strcpy(REPLACE, "YES");  
 printf("Replace = YES\n");  
 strcpy(v, "Replace");  
 strcpy(end, "Replace");  
 instantiate();  
 break;  
 case 37:  
 strcpy(REPLACE, "YES");  
 printf("Replace = YES\n");  
 strcpy(v, "Replace");  
 strcpy(end, "Replace");  
 instantiate();  
 break;  
  
 case 38:  
 strcpy(REPLACE, "YES");  
 printf("Replace = YES\n");  
 strcpy(v, "Replace");  
 strcpy(end, "Replace");  
 instantiate();  
 break;  
 }  
 }  
 fp = fp + 1;  
 if (fp < bp) {  
 */\* check out the condition variable \*/* flag = 1;  
 fcLoop();  
 continue;  
 }  
 return;  
 }  
}  
*/\* no more conclusion variables on queue \*/  
  
//==========================================================================  
/\* Routine to instantiate a variable (v) if it isn't already.  
The instantiate indication (instList) is a 0 if not, a 1 if it is.  
The vriable list (varList) contains the variable (v) \*/*void fc::check\_instantiation()  
{  
 i=1;  
  
 */\* find variable in the variable list \*/* while ((strcmp(v, varList[i]) != 0) && (i <= 37)) i = i+1;  
  
 */\* check if already instantiated \*/* if (instList[i] != 1)  
 {  
 instList[i] = 1; */\* mark instantiated \*/* switch (i)  
 {  
 */\* input statements for sample position knowledge base \*/* case 1:  
 printf("Is the Oil pump GOOD OR BAD?");  
 std::cin >> OIL\_PUMP;  
 break;  
 case 2:  
 printf("Is the Oil filter GOOD OR BAD?");  
 std::cin >> OIL\_FILTER;  
 break;  
 case 3:  
 printf("Is the Oil GOOD OR BAD?");  
 std::cin >> OIL;  
 break;  
 case 4:  
 printf("Is the Oil drain plug GOOD OR BAD?");  
 std::cin >> OIL\_DRAIN\_PLUG;  
 break;  
 case 5:  
 printf("Is the engine gasket GOOD OR BAD?");  
 std::cin >> ENGINE\_GASKET;  
 break;  
 case 6:  
 printf("Is the radiator GOOD OR BAD?");  
 std::cin >> RADIATOR;  
 break;  
 case 7:  
 printf("Is the radiator hose GOOD OR BAD?");  
 std::cin >> RADIATOR\_HOSE;  
 break;  
 case 8:  
 printf("Is the transmission drain plug GOOD OR BAD?");  
 std::cin >> TRANSMISSION\_DRAIN\_PLUG;  
 break;  
 case 9:  
 printf("Is the battery GOOD OR BAD?");  
 std::cin >> BATTERY;  
 break;  
 case 10:  
 printf("Is the ignition switch GOOD OR BAD?");  
 std::cin >> IGNITION\_SWITCH;  
 break;  
 case 11:  
 printf("Is the starter motor GOOD OR BAD?");  
 std::cin >> STARTER\_MOTOR;  
 break;  
 case 12:  
 printf("Is the battery terminals GOOD OR BAD?");  
 std::cin >> BATTERY\_TERMINALS;  
 break;  
 case 13:  
 printf("Is the fuel level GOOD OR BAD?");  
 std::cin >> FUEL\_LEVEL;  
 break;  
 case 14:  
 printf("Is the fuel condition GOOD OR BAD?");  
 std::cin >> FUEL\_CONDITION;  
 break;  
 case 15:  
 printf("Is the fuel system GOOD OR BAD?");  
 std::cin >> FUEL\_SYSTEM;  
 break;  
 case 16:  
 printf("Is the vacuum hose GOOD OR BAD?");  
 std::cin >> VACUUM\_HOSE;  
 break;  
 case 17:  
 printf("Is the spark plugs GOOD OR BAD?");  
 std::cin >> SPARK\_PLUGS;  
 break;  
 case 18:  
 printf("Is the fuel injectors GOOD OR BAD?");  
 std::cin >> FUEL\_INJECTORS;  
 break;  
 case 19:  
 printf("Is the ignition wires GOOD OR BAD?");  
 std::cin >> IGNITION\_WIRES;  
 break;  
 case 20:  
 printf("Is the engine components GOOD OR BAD?");  
 std::cin >> ENGINE\_COMPONENTS;  
 break;  
 case 21:  
 printf("Is the battery GOOD OR BAD?");  
 std::cin >> BATTERY;  
 break;  
 case 22:  
 printf("Is the alternator GOOD OR BAD?");  
 std::cin >> ALTERNATOR;  
 break;  
 case 23:  
 printf("Is the alternator belt GOOD OR BAD?");  
 std::cin >> ALTERNATOR\_BELT;  
 break;  
 case 24:  
 printf("Is the battery wires GOOD OR BAD?");  
 std::cin >> BATTERY\_WIRES;  
 break;  
 case 25:  
 printf("Is the electrical system GOOD OR BAD?");  
 std::cin >> ELECTRICAL\_SYSTEM;  
 break;  
 case 26:  
 printf("Is the coolant GOOD OR BAD?");  
 std::cin >> COOLANT;  
 break;  
 case 27:  
 printf("Is the drive belt GOOD OR BAD?");  
 std::cin >> DRIVE\_BELT;  
 break;  
 case 28:  
 printf("Is the radiator GOOD OR BAD?");  
 std::cin >> RADIATOR;  
 break;  
 case 29:  
 printf("Is the radiator fan GOOD OR BAD?");  
 std::cin >> RADIATOR\_FAN;  
 break;  
 case 30:  
 printf("Is the water pump GOOD OR BAD?");  
 std::cin >> WATER\_PUMP;  
 break;  
 case 31:  
 printf("Is the tire pressure GOOD OR BAD?");  
 std::cin >> TIRE\_PRESSURE;  
 break;  
 case 32:  
 printf("Is the air filter GOOD OR BAD?");  
 std::cin >> AIR\_FILTER;  
 break;  
 case 33:  
 printf("Is the fuel tank GOOD OR BAD?");  
 std::cin >> FUEL\_TANK;  
 break;  
 case 34:  
 printf("Is the fuel management system GOOD OR BAD?");  
 std::cin >> FUEL\_MANAGEMENT\_SYSTEM;  
 break;  
 case 35:  
 printf("Is the brake parts GOOD OR BAD?");  
 std::cin >> BRAKE\_PARTS;  
 break;  
 case 36:  
 printf("Is the brake parts GOOD OR BAD?");  
 std::cin >> BRAKE\_PARTS;  
 break;  
 case 37:  
 printf("Is the brake lines GOOD OR BAD?");  
 std::cin >> BRAKE\_LINES;  
 break;  
 case 38:  
 printf("Is the brake parts GOOD OR BAD?");  
 std::cin >> BRAKE\_PARTS;  
 break;  
  
 }  
 }  
 */\* end of input statements for the position knowledge base \*/*}  
  
*//==========================================================================  
/\* Search clause variable list for a varialbe (clVarList) equal to the  
one in front of the conclusion queue (cndVar). Return the statement  
number (sn). If there is no match, i.e., sn=0, the first statement  
for the space is flag. \*/*void fc::searchClauses()  
{  
 sn = flag;  
  
 while ((flag == 1) && (sn <= 38))  
 {  
 cn=1;  
 k = (sn-1)\*38 + cn;  
 *//cout << sn << " " << cndVar[fp] << endl;* while ((strcmp(clVarList[k], cndVar[fp]) != 0) && (cn < 37))  
 {  
 cn = cn+1;  
 k = (sn-1)\*38 + cn;  
 }  
 if (strcmp(clVarList[k], cndVar[fp]) == 0)flag = 0;  
 if (flag == 1) sn = sn+1;  
 }  
  
 if (flag == 1) sn=0;  
}  
  
*//==========================================================================  
/\* Routine to instantiate a variable (v) and then place it on the  
back of the queu (cndVar[bp]), if it is not already there. \*/*void fc::instantiate()  
{  
 i=1;  
 */\* find variable in the variable list (varList) \*/* while ((strcmp(v, varList[i]) != 0) && (i <= 38)) i=i+1;  
  
 */\* instantiate it \*/* instList[i] = 1;  
 i = 1;  
  
 */\* determine if (v) is or already has been on the queue (cndVar) \*/* while (((strcmp(v, cndVar[i]) != 0)) && (i <= 38)) i=i+1;  
  
 */\* variable has not been on the queue. Store it in the back of the queue \*/* if (strcmp(v, cndVar[i]) != 0)  
 {  
 strcpy(cndVar[bp], v);  
 bp=bp+1;  
 }  
}

**Program Run:**

Test Case 1:

\*\* ENTER CONCLUSION : **Part**

Is there a problem with the car? INPUT YES OR NO**NO**

NO ISSUE

\*\*\* SUCCESS

Done with backwards chaining, your problem is:

NO ISSUE

Your car is fine

Test Case 2:

\*\* ENTER CONCLUSION : **Part**

Is there a problem with the car? INPUT YES OR NO**YES**

Is the oil pressure low? INPUT YES OR NO**NO**

Is the oil grade correct? INPUT YES OR NO**YES**

Is the oil filter new? INPUT YES OR NO**NO**

Do you see a leak? INPUT YES OR NO**YES**

Is it an oil leak? INPUT YES OR NO**YES**

Is the leak coming from the oil drain plug? INPUT YES OR NO**NO**

Is the leak coming from a seam in the engine? INPUT YES OR NO**YES**

ENGINE GASKET

\*\*\* SUCCESS

Done with backwards chaining, your problem is :

ENGINE\_GASKET

Starting forwards chaining code....

Is the engine gasket GOOD OR BAD?**BAD**

Replace = YES

To fix you need to Replace ENGINE\_GASKET

Test Case 3:

\*\* ENTER CONCLUSION : **Part**

Is there a problem with the car? INPUT YES OR NO**YES**

Is the oil pressure low? INPUT YES OR NO**NO**

Is the oil grade correct? INPUT YES OR NO**YES**

Is the oil filter new? INPUT YES OR NO**YES**

Do you see a leak? INPUT YES OR NO**NO**

Is it an oil leak? INPUT YES OR NO**NO**

Is the leak coming from the oil drain plug? INPUT YES OR NO**NO**

Is the leak coming from a seam in the engine? INPUT YES OR NO**NO**

Is it a coolant leak? INPUT YES OR NO**NO**

Is the leak coming from the radiator? INPUT YES OR NO**NO**

Is the leak coming from a radiator hose? INPUT YES OR NO**NO**

Is it a transmission fluid leak? INPUT YES OR NO**NO**

Is the leak coming from the transmission drain plug? INPUT YES OR NO**NO**

Does the engine rotate when starting? INPUT YES OR NO**YES**

Are the battery terminals connected? INPUT YES OR NONO

What is the battery voltage? INPUT A REAL NUMBER**11**

Is the starter motor working? INPUT YES OR NO**NO**

Is there fuel in the tank? INPUT YES OR NO**YES**

FUEL SYSTEM

\*\*\* SUCCESS

Done with backwards chaining, your problem is :

FUEL\_SYSTEM

Starting forwards chaining code....

Is the fuel system GOOD OR BAD?**BAD**

Replace = YES

To fix you need to Replace FUEL\_SYSTEM

**Analysis of Program Run:**

The effects of our features allow us to run both of the algorithms efficiently together. The backwards chaining program gets the user input on what the problem is to find the part and the forward chaining program will get that information to find a repair. The modifications to the program makes this possible.

**Analysis of Results:**

The results above show the implementations of our features. The programs starts the backwards chaining program and gets a conclusions from the user. The program will ask the user for any uninstantiated variables then get you a result which is the part that is problem with the vehicle. The program will then start the forward chaining program and ask the is whether the part from the backwards chaining program is good or not. The program will then suggest a repair for that. The modifications implemented allows the program to run this efficiently.

**Conclusions:**

I learned a lot about forward chaining and backward chaining through this project. It’s very complex and can take a while to understand but it works well on these kinds of problems. After thoroughly understanding these methodologies was I able to implement them into the programs.

**Team Member Contributions:**

Yuvanesh Rajamani:

* Translating Backwards code into cpp
* Fixing Erroneous code
* Implemented data structures into backwards code
* Translating forwards code into cpp
* Implemented data structures into forwards code
* Debugging

Robert Maldonado:

* Translating Backwards code into cpp
* Translating forwards code into cpp
* Fixing Erroneous code
* Implemented data structures into forwards code
* Implemented headers
* Debugging

John Edwards:

* Created diagnosis tree and repair tree
* developed data structures
* Implemented data structures into forwards code
* Fixing Erroneous code
* Debugging

**References:**

* Dr.Moonis Ali
* Old car manuel