**CS5346**

**Project 1:**

**Expert Profession Recommender System**

**Submitted to: Dr. Moonis Ali**

**By:**

**Gentry Atkinson**

**Vishal Kumar M. G.**

**Table of Contents:**

1. Introduction 3
   1. Problem Statement
   2. Solution
2. Contributors 4
3. Analysis of the Problem 5
   1. Domain and Goal
   2. Problems with existing code
   3. Proposed Solution
4. Knowledge Base Design 7
   1. Decision Tree
   2. Rules Formation
5. Inference Engine Design 9
   1. Backward Chaining Tree
   2. Rules for Backward Chaining
   3. Forward Chaining Tree
   4. Rules for Forward Chaining
6. Methodology 21

1. Backward chaining

2. Data Structures used in Backward chaining.

3. Algorithm for Backward chaining.

4. Forward chaining.

5. Data Structures used in Forward chaining.

6. Algorithm used in Forward chaining.

1. Program Implementation 25
2. Sample Runs 94
   1. Sample 1
   2. Sample 2
   3. Sample 3
   4. Sample 4
   5. Sample 5
3. Analysis of Program Results 97
4. Comparison of Systems 98

**1 Introduction:**

**1.1 Problem Statement**

According to the National Center for Education Statistics, the current cost of attending a 4-year institution of higher education is currently more than $26,000 annually in the United States. With these high costs being placed of students it is imperative that we give students the tools to complete their educations as efficiently as possible and to realize the greatest result from their education after they’ve graduated.

Giving students expert guidance as early as possible in their educations will help them realize the greatest benefit from the cost of their education. But our nation collectively also invests a great amount in the education of our students on top of their contribution of tuition. We rely on students being properly placed in positions were they can excel. Properly pairing well educated students with high-demand jobs assures the greatest return to the community for the expense of educating a young adult.

The earlier a student can be given the expert guidance the better. Early guidance will allow a student to shape their education most effectively. But human guidance is often an expensive commodity. With college attendance rates steadily rising, undergraduate counselors find themselves advising larger groups of students without additional resources. It is therefore greatly advantageous to students, institutions, and society that an automated career recommender be developed.

**1.2 Solution**

Modern students have nearly continuous exposure to computer and network resources. By placing an automated recommender system at their disposal, students will be given access to to early and regular career guidance.

Any automated that is applied in this problem domain will need to be able to recommend a large variety of professions to a student based on his or her interests. Furthermore it would be beneficial if this system could recommend not only a profession but a specific area within a profession.

The focus of this project will be the implementation of a system which fits these criteria.

**2 Contributors:**

**2.1 Gentry Atkinson**

* Graphics and Report
* Rules Base
* Variable Lists
* Definition of Profession Conditions

**2.2 Vishal Kumar Mainka Ganeshbapu**

* Object Oriented code re-factoring
* Development of Data Structures
* Industry Research
* Definition of Area Conditions

**2.3 Outside Contributors**

This project would like to thank Dr. Moonis Ali of Texas State University for providing the example code that this project was built from.

We would also like to thank Dr. Robert Levine for his contribution of the core algorithms for Backward and Forward Chaining as Artificial Intelligence and Expert Systems: Comprehensive Guide (1990)

**3 Analysis of the Problem:**

**3.1: Domain and Goal**

* Develop Rules based on market research.
* Develop decision tree for both forward chaining and backward chaining.
* Converting the rules which we have proposed into the rules which are used as the knowledge base for both backward and forward chaining.
* Develop a program for both backward chaining and forward chaining and make it work individually.
* Improve code efficiency and reliability.
* Develop code which can be easily reused.
* Making sure both backward chaining object and Forward chaining object work in a single run.

**3.2: Problems with existing code**

* Existing code base has a lot of syntax errors.
* Global variable degrade code security and reliability.
* The use of GOTO statements produced code which was difficult to read and maintain.
* The use of c-strings rather than updated object oriented strings created inefficient and unreliable code.
* The code was written without the use of modern techniques which are intended to improve the readability and supportability of the code.
* The knowledge base was presented in a fashion that was difficult for human readers.
* Backward chaining and forward chaining was written in two separate executables rather than being built as objects which can be incorporated into larger projects.

**3.3: Proposed Solution**

* Eliminate all the syntax error.
* Modify the program to run without relying on global variables and GOTO statements.
* The GOTO statements will be converted into separate functions and made to work appropriately.
* Convert c-strings into string objects.
* We will change the code by implementing the OOPS concepts (Classes) to make the code more reusable and easier to understand.
* The knowledge base and Inference engine will be separated into separate functions which will improve the re-usability of the code.
* Constants will be used in the program to set easily changeable sizes for the variable arrays.
* Eliminate redundancy.
* Make both Backward chaining and Forward chaining program run in a single instance.

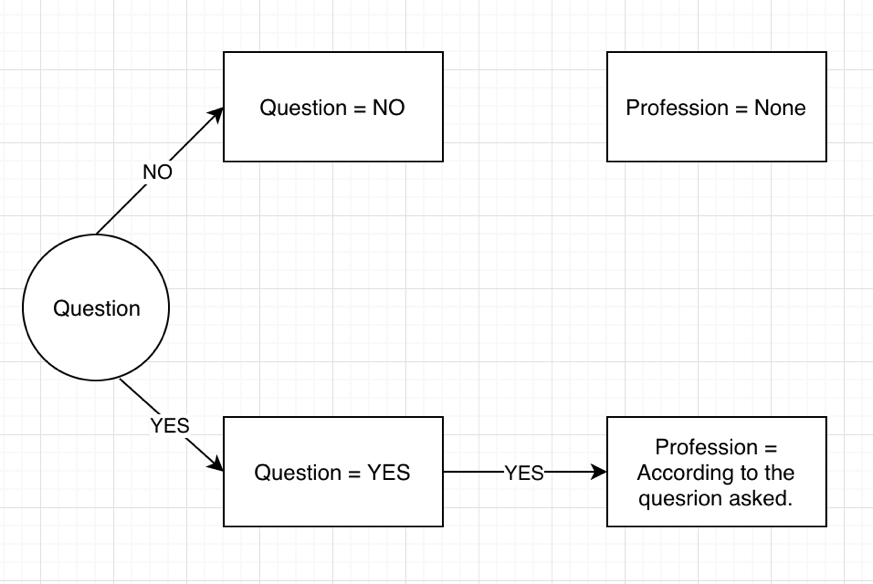
**4 Knowledge base Design:**

**4.1 Decision Tree:**

In the Decision Tree,

1. Rectangular boxes like these denote *conclusion* as well as *intermediate* nodes.
2. Circle shaped boxes denote *Decision* nodes.

Following is an extract from currently used decision tree:



This is beginning of the tree. The arrows mark the flow of program on action performed at the decision node. For better explanation here is how the tree is traversed:

1. If there ISN’T a Question, variable ‘Question’ will be set to ‘No’ and ‘Profession’ will be set to ‘None’.
2. If there IS a Question, variable ‘Question’ will be set to ‘YES’ and program will continue to look for the Profession.

We created 2 decision trees one for Backwards Chaining and other for Forward Chaining. Using these trees, we created rules for our Knowledge Base.

**4.2 Rules Formation**

As shown earlier we have a decision tree and we need to form a rule from it.

Our rules are in format such as:

IF condition 1 AND condition 2 THEN conclusion.

Example:

IF Question == YES AND Work == YES THEN Assign(Work = YES).

**5 Inference Engine Design:**

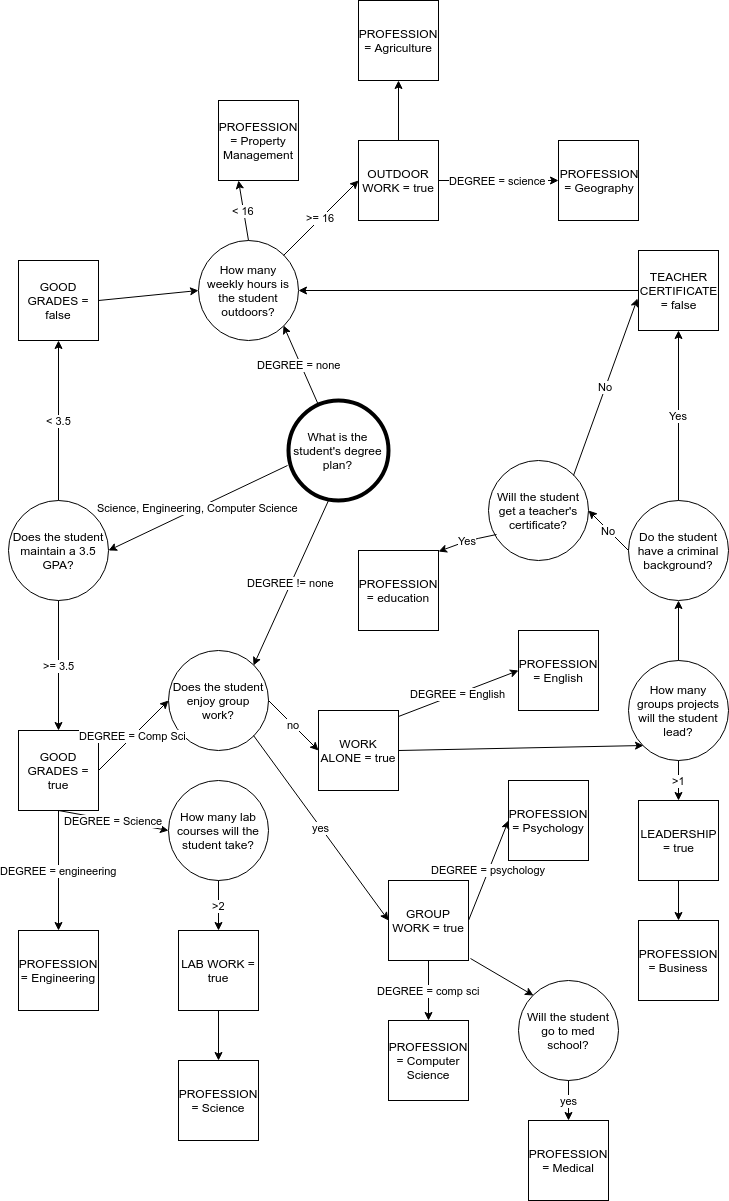
The Inference Engine is an integral part of this project. It is a reasoning system that evaluates the current state of the knowledge base, applies relevant rules and then sets the new knowledge into the knowledge base.

There are two parts of this inference engine:

1. Backward Chaining: We used this for identifying the Interest of the student.
2. Forward Chaining: We used this for providing Profession to given to the Student.

The Decision Tree for Backward chaining and Forward chaining are shown below:

**5.1 : Backward chaining Tree:**



**5.2 : Rules for Backward chaining:**

Rule 10: IF GPA >= 3.5 THEN GOOD GRADES = true

Rule 20: IF GPA < 3.5 THEN GOOD GRADES = false

Rule 30: IF COURSES WITH LABS > 2 THEN LABWORK = true

Rule 40: IF GROUP LEADER >= 1 THEN LEADERSHIP = true

Rule 50: IF DEGREE = engineering AND GOOD GRADES = true THEN PROFESSION = engineering.

Rule 60: IF DEGREE = science AND GOOD GRADES = true AND LABWORK = true THEN PROFESSION = science

Rule 70: IF DEGREE != none AND LEADERSHIP = true THEN PROFESSION = business

Rule 80: IF GROUP WORK = false THEN WORK ALONE = true

Rule 90: IF DEGREE != none AND MED SCHOOL = true AND GROUP WORK = true THEN PROFESSION = medical

Rule 100: IF DEGREE = english AND WORK ALONE = true THEN PROFESSION = English

Rule 110: IF HOURS OUTSIDE >= 16 THEN OUTDOOR WORK = true

Rule 120: IF DEGREE = science AND OUTDOOR WORK = true THEN PROFESSION = geography

Rule 130: IF DEGREE = psychology AND GROUP WORK = true THEN PROFESSION = psychology

Rule 140: IF DEGREE = none AND OUTDOOR WORK = true THEN PROFESSION = agriculture

Rule 150: IF MED SCHOOL = false and MED FIELD = true THEN MEDICAL CERTIFICATE = true

Rule 160: IF DEGREE != none AND MEDICAL CERTIFICATE = true AND GROUP WORK = true THEN PROFESSION = health care

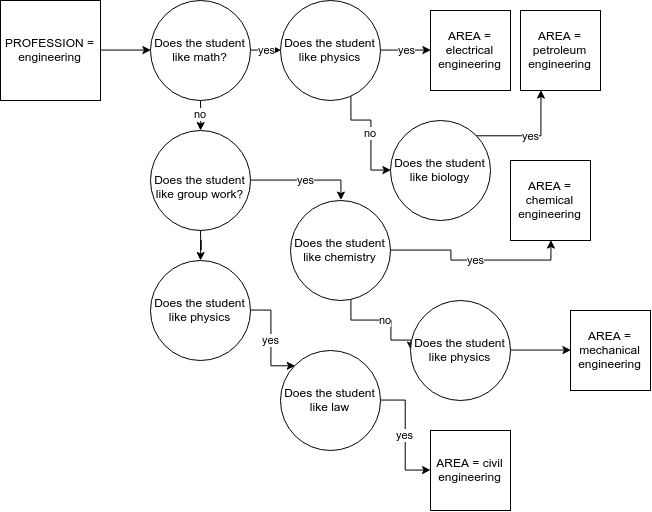
Rule 170: IF CRIMINAL BACKGROUND = true THEN TEACHER CERTIFICATE = false

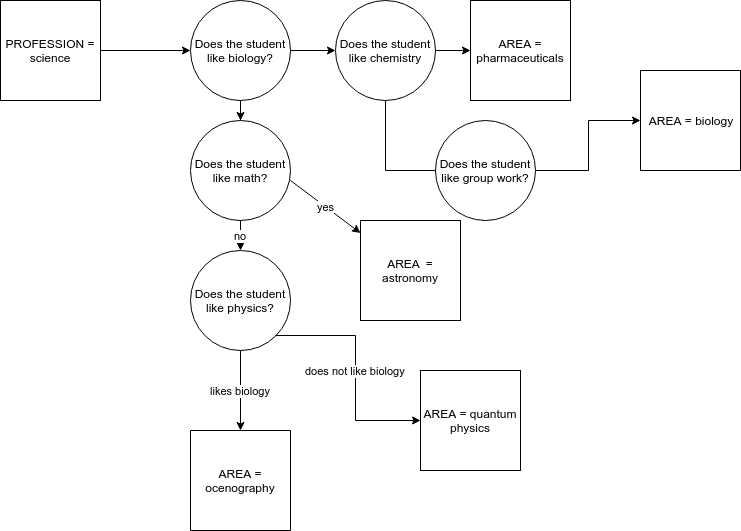
Rule 180: IF DEGREE != none AND TEACHER CERTIFICATE = true THEN PROFESSION = education

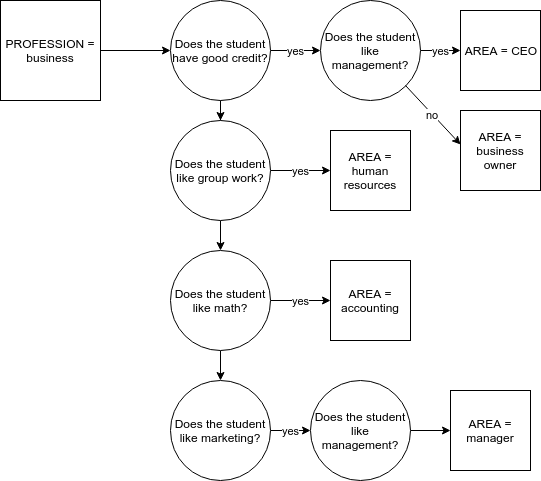
Rule 190: IF DEGREE = none AND OUTDOOR WORK = false THEN PROFESSION = property management.

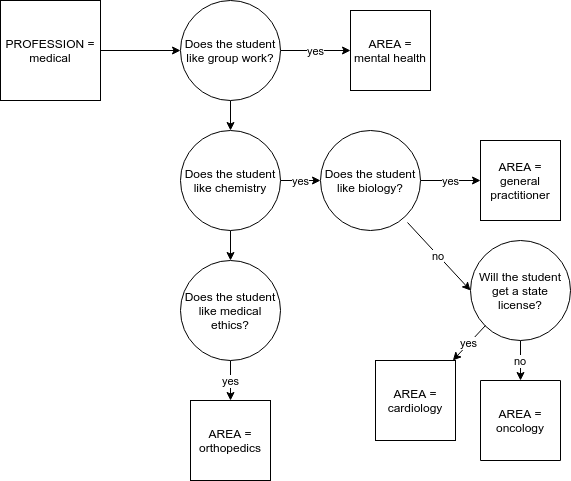
Rule 200: IF DEGREE = computer science AND GOOD GRADES = true AND GROUP WORK = true THEN PROFESSION = computer science

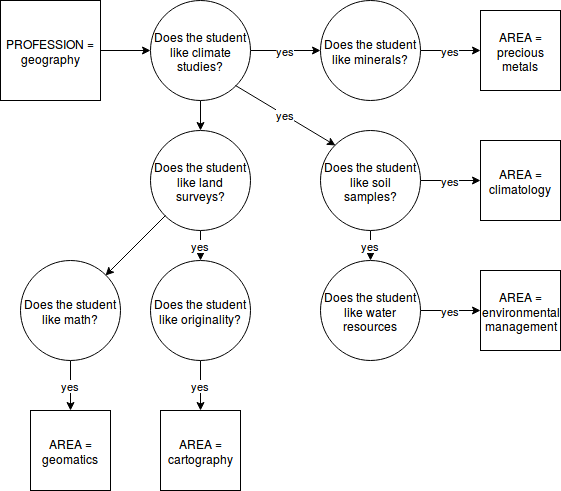
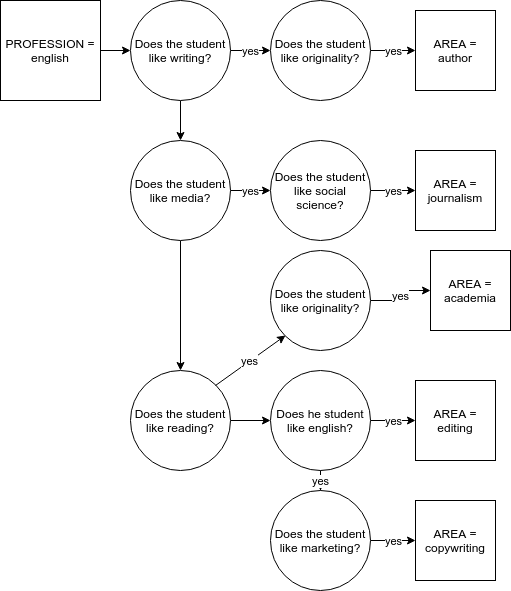
**5.3 : Forward chaining Trees:**







****

****

**5.4 : Rules for Forward chaining:**

Rule 10: IF PROFESSION = engineering LIKE MATH = yes, LIKE PHYSICS = yes THEN AREA = electrical engineering

Rule 20: IF PROFESSION = engineering AND LIKE PHYSICS = yes AND GROUP WORK = yes then AREA = mechanical

engineering

Rule 30: IF PROFESSION = engineering AND LIKE LAW = yes AND LIKE PHYSICS = yes the AREA = civil

Rule 40: IF PROFESSION = engineering AND LIKE MATH = yes AND LIKE BIOLOGY = yes THEN AREA = petroleum

engineering

Rule 50: IF PROFESSION = engineering AND LIKE CHEMISTRY = yes AND GROUP WORK = yes THEN AREA = chemical

engineering

Rule 60: IF PROFESSION = science AND LIKE BIOLOGY = yes AND LIKE CHEMISTRY= yes THEN AREA = pharmaceuticals

Rule 70: IF PROFESSION = science AND LIKE BIOLOGY = yes AND LIKE GROUP WORK = yes THEN AREA = biology

Rule 80: IF PROFESSION = science AND LIKE MATH =yes THEN AREA = astronomy

Rule 90: IF PROFESSION = science AND LIKE PHYSICS = yes THEN AREA = quantam physics

Rule 100: IF PROFESSION = science AND LIKE BIOLOGY = yes AND LIKE PHYSICS = yes THEN area = oceanography

Rule 110: IF PROFESSION = business AND GOOD CREDIT = yes THEN AREA = business owner

Rule 120: IF PROFESSION = business AND LIKE MARKETING = yes AND LIKE MANAGEMENT = yes THEN AREA = manager

Rule 130: IF PROFESSION = business AND LIKE MATH = yes THEN AREA = accounting

Rule 140: IF PROFESSION = business AND GROUP WORK = yes THEN AREA = human resources

Rule 150: IF PROFESSION = business AND GOOD CREDIT = yes, LIKE MANAGEMENT = yes THEN AREA = ceo

Rule 160: IF PROFESSION = medical AND GROUP WORK = yes THEN AREA = mental health

Rule 170: IF PROFESSION = medical LIKE CHEMISTRY =yes THEN AREA = oncology

Rule 180: IF PROFESSION = medical AND STATE LICENSURE = yes AND LIKE chemistry = yes THEN AREA = cardiology

Rule 190: IF PROFESSION = medical AND LIKE MEDICAL ETHICS = yes THEN AREA = orthopedics

Rule 200: IF PROFESSION = medical AND LIKE BIOLOGY = yes AND LIKE CHEMISTRY= yes THEN AREA = general

practitioner

Rule 210: IF PROFESSION = english AND LIKE WRITING = yes AND LIKE ORIGINALITY = yes THEN AREA = author

Rule 220: IF PROFESSION = english AND LIKE MEDIA = yes AND LIKE SOCIAL SCIENCE = yes THEN AREA = journalism

Rule 230: IF PROFESSION = english AND LIKE ENGLISH = yes AND LIKE READING = yes THEN AREA = editing

Rule 240: IF PROFESSION = english AND LIKE MARKETING = yes AND LIKE ENGLISH = yes THEN AREA = copy writing

Rule 250: IF PROFESSION = english AND LIKE READING = yes THEN LIKE ORIGINALITY = yes THEN AREA = academia

Rule 260: IF PROFESSION = geography AND LIKE CLIMATE = yes AND LIKE STONES = yes THEN AREA = precious metals

Rule 270: IF PROFESSION = geography AND LIKE LAND = yes AND LIKE ORIGINALITY = yes THEN AREA = cartography

Rule 280: IF PROFESSION = geography AND LIKE SOIL = yes AND LIKE CLIMATE = yes THEN AREA = climatology

Rule 290: IF PROFESSION = geography AND LIKE WATER RESOURCES = yes AND LIKE SOIL = yes THEN AREA =

environmental management

Rule 300: IF PROFESSION = geography AND LIKE MATH = yes THEN AREA = geomatics

Rule 310: IF PROFESSION = psychology AND LIKE SOCIAL SCIENCE = yes THEN AREA = mental therapy

Rule 320: IF PROFESSION = psychology AND GROUP WORK = yes AND LIKE ANALYTICAL SKILLS = yes THEN AREA =

psychological research

Rule 330: IF PROFESSION = psychology AND LIKE INTERNSHIPS = yes THEN AREA = industrial psychology

Rule 340: IF PROFESSION = psychology AND GROUP WORK = yes AND LIKE SOCIAL SCIENCE = yes THEN AREA =

Accessibility

Rule 350: IF PROFESSION = psychology AND LIKE COMMUNICATION = yes, LIKE ANALYTICAL SKILLS = yes THEN AREA

= clinical psychology

Rule 360: IF PROFESSION = agriculture AND GOOD CREDIT = yes AND LIKE MANAGEMENT = yes THEN AREA =

farm ownership

Rule 370: IF PROFESSION = agriculture AND GROUP WORK = yes AND LIKE MANAGEMENT = yes THEN AREA = ranch work

Rule 380: IF PROFESSION = agriculture AND LIKE MANAGEMENT = yes AND LIKE MARKETING = yes THEN AREA

= viniculture

Rule 390: IF PROFESSION = agriculture AND LIKE CHEMISTRY = yes THEN AREA = crop spraying

Rule 400: IF PROFESSION = agriculture AND LIKE WATER RESOURCES = yes THEN AREA = irrigation

Rule 410: IF PROFESSION = health care AND LIKE COMMUNICATION = yes THEN AREA = health services management

Rule 420: IF PROFESSION = health care AND LIKE MANAGEMENT = yes THEN AREA = health care administration

Rule 430: IF PROFESSION = health care AND LIKE BIOLOGY = yes THEN AREA = physicians assistant

Rule 440: IF PROFESSION = health care AND LIKE MANAGEMENT = yes AND GROUP WORK = yes THEN AREA

= public health

Rule 450: IF PROFESSION = health care AND LIKE ANATOMY = yes AND LIKE CHEMISTRY = yes THEN AREA = nursing

Rule 460: IF PROFESSION = education AND LIKE CHILDREN = yes THEN AREA = early childhood

Rule 470: IF PROFESSION = education AND LIKE READING = yes AND LIKE COMMUNICATIONS = yes THEN AREA = k-12

Rule 480: IF PROFESSION = education AND LIKE MATH = yes AND LIKE CHEMISTRY = yes THEN AREA = higher education

Rule 490: IF PROFESSION = education AND LIKE COMMUNICATIONS = yes, LIKE CHILDREN = yes THEN AREA =

teacher’s aid

Rule 500: IF PROFESSION = education AND LIKE SOCIAL SCIENCE = yes AND LIKE COMMUNICATION = yes THEN

AREA = counselor

Rule 510: IF PROFESSION = property management AND LIKE MANAGEMENT = yes THEN AREA = property manager

Rule 520: IF PROFESSION = property management AND GROUP WORK = yes THEN AREA = realty

Rule 530: IF PROFESSION = property management AND RELIABLE = yes THEN AREA = cleaning

Rule 540: IF PROFESSION = property management AND LIKE REPAIRS = yes THEN AREA = maintenance work

Rule 550: IF PROFESSION = property management AND LIKE COMMUNICATION = yes THEN AREA = painting

Rule 560: IF PROFESSION = cs AND LIKE MATH = yes AND SOCIAL SCIENCE = yes THEN AREA = ai

Rule 570: IF PROFESSION = cs AND LIKE MATH = yes AND LIKE ANALYTICAL SKILLS = yes THEN AREA = machine learning

Rule 580: IF PROFESSION = cs AND GROUP WORK = yes THEN AREA = software engineering

Rule 590: IF PROFESSION = cs AND LIKE MATH = yes AND LIKE MARKETING = yes THEN AREA = data science

Rule 600: IF PROFESSION = cs AND LIKE ORIGINALITY = yes THEN AREA = languages

**6 Methodology**:

**6.1 Backward chaining:**

Backward chaining (or backward reasoning) is an inference method described colloquially as working backward from the goal. It is used in automated theorem provers, inference engines, proof assistants, and other artificial intelligence applications

Backward chaining starts with a list of goals (or a hypothesis) and works backwards from the consequent to the antecedent to see if any data supports any of these consequents. An inference engine using backward chaining would search the inference rules until it finds one with a consequent (Then clause) that matches a desired goal. If the antecedent (If clause) of that rule is not known to be true, then it is added to the list of goals (for one's goal to be confirmed one must also provide data that confirms this new rule).

**6.2 Data Structures used in Backward Chaining:**

**Clause Variable List:**

Clause variable list stores all the variables used in IF part of the rules. These variables are stored in the array with four array slots allocated for each slot. If only one or two array slots are filled, the remaining slots are left blank. If all the clauses in the IF part of a rule are connected by the logical operator AND, all the variables in these clauses must be instantiated before the THEN part can be executed.

We can calculate the Clause variable number using the formula:

Clause\_Number=4\*((Rule\_Number/10)-1) + 1.

**Conclusion Stack:**

Conclusion stack is the most important data structure in implementing backward chaining. It tells which rule contains the conclusion that we are trying to reach and which clause number in the IF portion is currently examined for instantiated.

**Conclusion List:**

Conclusion list consists of the consequents (THEN part). A clause variable pointer keeps track of the current rule and currently executed clause in that rule. It consists of rule number, conclusion associated with that rule number and set of conditions which yields the conclusion. Conclusion list is complete when the THEN portion of each rule is placed in the same row as the rule number. If the IF part of a rule is true, we invoke the THEN part and instantiate the conclusion.

**Variable List:**

This Data structure contains two items: one is a variable name for each variable contained in the IF part of the knowledge base rules and the other item tells us whether or not the variable is instantiated. A variable only appears once in the list no matter how many condition clauses it appears. The instantiated column is always initially set to not instantiated (NI). It will be changed to instantiated (I) as each variable is set to a value.

**6.3 ALGORITHM FOR BACKWARD CHAINING:**

Step 1: Get the Conclusion for the problem from User.

Step 2: Search for first occurrence of the conclusion which is taken as input from the user in the conclusion list.

Step 3: If found place the rule on the conclusion stack using rule number and a [1] to represent the clause number. The clause number can be found by using the formula 4\*((rule\_number/10)-1)+1. If the conclusion is not found in the conclusion list, notify the user that an answer can’t be found.

Step 4: Instantiate IF clause of the statement.

Step 5: If any of the IF clause variables are not instantiated, ask the user for input.

Step 6: If one of the clauses is conclusion variable, place the conclusion variable’s rule on the top of the stack and go back to Step 4.

Step 7: If statement on top of the stack is not satisfied, remove the rule and search for next instance of the conclusion in the conclusion list

Step 8: If found go back to Step 4.

Step 9: If rule on the top of the stack is satisfied, remove it from the stack. If another conclusion variable is underneath, increment the clause number and for remaining clauses go back to step 4.

Step 10: If no other conclusion is underneath, we have solution for which user can conclude.

**6.4 FORWARD CHAINING:**

Forward chaining (or forward reasoning) is one of the two main methods of reasoning when using an inference engine. Forward chaining is a popular implementation strategy for expert systems, business and production rule systems. Forward chaining starts with the available data and uses inference rules to extract more data (from an end user, for example) until a goal is reached. An inference engine using forward chaining searches the inference rules until it finds one where the antecedent (If clause) is known to be true. When such a rule is found, the engine can conclude, or infer, the consequent (Then clause), resulting in the addition of new information to its data.

In our case, the Backward chaining process will generate a PROFESSION by considering all the inputs given by user and pass that PROFESSION to Forward chaining. Now the forward chaining will refer the information stored in the knowledge base and apply the forward chaining techniques and will provide the solution to the PROFESSION.

**6.5 DATA STRUCTURE USED IN FORWARD CHAINING:**

**Clause Variable list:**

The clause variable list stores the antecedents (IF part).

**Conclusion Variable Queue:**

Variable on which we are working is placed on conclusion variable queue and are served on first come first serve basis.

**Clause Variable Pointer:**

It keeps track of clause within the rule being examined. It’s used to keep track of the rule and the clause within the rule being processed.

**Variable List:**

The variable list is used to know whether the variable is instantiated or not. When user enters some information for a variable, then it is instantiated, and the answer given by the user is stored.

**6.6 ALGORITHM FOR FORWARD CHAINING:**

Step 1: The condition is identified.

Step 2: The condition variable is placed on the conclusion variable queue and its value is marked on the variable list

Step 3: The clause variable list is searched for the variable whose name is the same as the one in the front of the queue.

Step 4: If found, the rule number and a 1 are placed into the clause variable pointer. If not found, go to step 7.

Step 5: Each variable in the IF clause of the rule that is not already instantiated is now instantiated. The variables are in the clause variable list. If all the clauses are true, the THEN part is invoked

Step 6: The instantiated THEN part of the variable is placed in the back of the conclusion variable queue.

Step 7: When there are no more IF statements containing the variable that is at the front of the conclusion variable queue, that variable is removed.

Step 8: If there are no more variables on the conclusion variable queue, end the session. If there are more variables, go to step 3.

**7 : Program Implementation:**

**7.1 Backward chaining header file (bc.h):**

class BC

{

private:

/\* Variables declared here \*/

static const int size = 21;

static const int clauseSize = 81;

std::string conclusionList[size], variableList[size], clauseVariableList[clauseSize];

std::string varble;

std::string goodGrades, degree;

std::string labwork, profession;

std::string outdoorWork, leadership;

std::string groupWork, medSchool, medCert;

std::string teachCert,medField,criminal,workAlone;

int coursesWithLabs, groupLeader, hoursOutside;

std::string buff;

bool done;

int instantiatedList[size];

int statementStack[size], clauseStack[size], statementNumber, f, statementActive, stackPointer;

float grade;

bool debug;

void B520();

void KeepProcessing();

void determine\_member\_concl\_list();

void push\_on\_stack();

void instantiate();

void initkbase(int);

void InBetweenFunction();

void popStack();

void B545();

public:

BC(bool debug = false);

void inferenceSection();

void printProfession();

std::string getProfession() {return profession;}

};

**7.2 Backward chaining .cpp file (bc.cpp)**

BC::BC(bool debug)

{

this->debug = debug;

// Stack space is 10 we initially place stack space at 10+1

stackPointer = size;

for (int i=1; i < size; i++)

{

//conclusionList[i] = "";

conclusionList[i].assign("");

//variableList[i] = "";

variableList[i].assign("");

instantiatedList[i]=0;

statementStack[i]=0;

clauseStack[i]=0;

}

for (int i=1; i < clauseSize + 1; i++)

//clauseVariableList[i] = "";

clauseVariableList[i].assign("");

// Initializing Conclusion List

/\*\*\* comment 305 \*\*\*\*\*/

//strcpy(conclt[1], "PO");

conclusionList[1].assign("GOODGRADES");

//strcpy(conclt[2], "QU");

conclusionList[2].assign("GOODGRADES");

//strcpy(conclt[3], "PO");

conclusionList[3].assign("LABWORK");

//strcpy(conclt[4], "PO");

conclusionList[4].assign("LEADERSHIP");

//strcpy(conclt[5], "PO");

conclusionList[5].assign("PROFESSION");

//strcpy(conclt[6], "PO");

conclusionList[6].assign("PROFESSION");

conclusionList[7].assign("PROFESSION");

conclusionList[8].assign("WORKALONE");

conclusionList[9].assign("PROFESSION");

conclusionList[10].assign("PROFESSION");

conclusionList[11].assign("OUTDOORWORK");

conclusionList[12].assign("PROFESSION");

conclusionList[13].assign("PROFESSION");

conclusionList[14].assign("PROFESSION");

conclusionList[15].assign("MEDICALCERTIFICATE");

conclusionList[16].assign("PROFESSION");

conclusionList[17].assign("TEACHINGCERTIFICATE");

conclusionList[18].assign("PROFESSION");

conclusionList[19].assign("PROFESSION");

conclusionList[20].assign("PROFESSION");

//printf("\*\*\* CONCLUSION LIST \*\*\*\n");

for (int i=1; i<21; i++) /\*printf("CONCLUSION %d %s\n", i,conclt[i])\*/

cout << "Conclusion: " << i << " " << conclusionList[i] << endl;

cout<<"HIT RETURN TO CONTINUE"<<endl;

getchar();

cout << "Variable List:" << endl;

/\*\*\*\* comment 367 \*\*\*\*\*/

//strcpy(varlt[1], "DE");

variableList[1].assign("DEGREE");

//strcpy(varlt[2], "DI");

variableList[2].assign("GOODGRADES");

//strcpy(varlt[3], "EX");

variableList[3].assign("LABWORK");

//strcpy(varlt[4], "GR");

variableList[4].assign("LEADERSHIP");

variableList[5].assign("MEDSCHOOL");

variableList[6].assign("GROUPWORK");

variableList[7].assign("WORKALONE");

variableList[8].assign("OUTDOORWORK");

variableList[9].assign("MEDICALCERTIFICATE");

variableList[10].assign("TEACHINGCERTIFICATE");

variableList[11].assign("GPA");

variableList[12].assign("PROFESSION");

variableList[13].assign("COURSESWITHLABS");

variableList[14].assign("GROUPLEADER");

variableList[15].assign("HOURSOUTSIDE");

variableList[16].assign("MEDFIELD");

variableList[17].assign("CRIMINALBACKGROUND");

for(int i=1; i<18; i++) /\*printf("VARIABLE %d %s\n", i, varlt[i])\*/

cout << "Variable: " << i << " " << variableList[i]<<endl;

cout<<"HIT RETURN TO CONTINUE"<<endl;

getchar();

// Initializing Clause Variable List

cout<<"\*\*\*\*Clause variable list\*\*\*\*"<<endl;

/\*\*\*\*\* comment 407 through 409 \*\*\*/

//strcpy(clvarlt[1], "DE");

clauseVariableList[1].assign("GPA");

//strcpy(clvarlt[5], "DE");

clauseVariableList[5].assign("GPA");

//strcpy(clvarlt[9], "DE");

clauseVariableList[9].assign("COURSESWITHLABS");

//strcpy(clvarlt[10], "DI");

clauseVariableList[13].assign("GROUPLEADER");

//strcpy(clvarlt[13], "QU");

clauseVariableList[17].assign("DEGREE");

clauseVariableList[18].assign("GOODGRADES");

//strcpy(clvarlt[14], "GR");

clauseVariableList[21].assign("DEGREE");

clauseVariableList[22].assign("GOODGRADES");

clauseVariableList[23].assign("LABWORK");

//strcpy(clvarlt[15], "EX");

clauseVariableList[25].assign("DEGREE");

clauseVariableList[26].assign("LEADERSHIP");

//strcpy(clvarlt[17], "QU");

clauseVariableList[29].assign("GROUPWORK");

//strcpy(clvarlt[18], "GR");

clauseVariableList[33].assign("DEGREE");

clauseVariableList[34].assign("MEDSCHOOL");

clauseVariableList[35].assign("GROUPWORK");

//strcpy(clvarlt[19], "EX");

clauseVariableList[37].assign("DEGREE");

clauseVariableList[38].assign("WORKALONE");

clauseVariableList[41].assign("HOURSOUTSIDE");

clauseVariableList[45].assign("DEGREE");

clauseVariableList[46].assign("OUTDOORWORK");

clauseVariableList[49].assign("DEGREE");

clauseVariableList[50].assign("GROUPWORK");

clauseVariableList[53].assign("DEGREE");

clauseVariableList[54].assign("OUTDOORWORK");

clauseVariableList[57].assign("MEDSCHOOL");

clauseVariableList[58].assign("MEDFIELD");

clauseVariableList[61].assign("DEGREE");

clauseVariableList[62].assign("MEDICALCERTIFICATE");

clauseVariableList[63].assign("GROUPWORK");

clauseVariableList[65].assign("CRIMINALBACKGROUND");

clauseVariableList[69].assign("DEGREE");

clauseVariableList[70].assign("TEACHINGCERTIFICATE");

clauseVariableList[73].assign("DEGREE");

clauseVariableList[74].assign("OUTDOORWORK");

clauseVariableList[77].assign("DEGREE");

clauseVariableList[78].assign("GOODGRADES");

clauseVariableList[79].assign("GROUPWORK");

//Printing Clause Variable List

for(int i=1; i<21; i++)

{

// printf("\*\* CLAUSE %d\n", i);

cout<<"\*\*\*CLAUSE\*\*\* "<<i << endl;

for(int j=1; j<5; j++)

{

int k = 4 \* (i-1) + j;

//printf("VARIABLE %d %s\n", j, clvarlt[k]); }

cout<<"VARIABLE: "<< j << " " << clauseVariableList[k]<<endl;

}

}

cout<<"HIT THE RETURN KEY TO CONTINUE..."<<endl;

cout<<endl;

getchar();

goodGrades.assign("");

degree.assign("");

labwork.assign("");

profession.assign("");

outdoorWork.assign("");

leadership.assign("");

groupWork.assign("");

medSchool.assign("");

medCert.assign("");

teachCert.assign("");

medField.assign("");

criminal.assign("");

workAlone.assign("");

coursesWithLabs = -1;

groupLeader = -1;

hoursOutside = -1;

grade = -1.0;

done = false;

return;

}

void BC::inferenceSection()

{

if (debug) cout << "inferenceSection() called" << endl;

//cout<<"\*\* ENTER CONCLUSION ? ";

//cin>>varble;

varble.assign("PROFESSION");

cout << "Conclusion is set to: " << varble << endl;

B520();

}

void BC::B520()

{

if (debug) cout << "B520 called" << endl;

if (debug) cout << "Variable in B520 is " << varble << endl;

if (done) return;

f=1;

determine\_member\_concl\_list();

if (statementNumber != 0)

{

/\* if sn = 0 then no conclusion of that name \*/

KeepProcessing();

if(statementNumber != 0)

{

InBetweenFunction();

popStack();

}

//thenPart();

}

}

void BC::KeepProcessing()

{

if (debug) cout << "KeepProcessing() called" << endl;

do

/\* push statement number (sn) and clause number=1 on

goal stack which is composed of the statement stack (statsk) and clause stack (clausk) \*/

{

push\_on\_stack();

B545();

} while((statementActive != 1) && (statementNumber !=0));

}

void BC::determine\_member\_concl\_list()

{

if (debug) cout << "determine\_member\_concl\_list() called for " << varble << endl;

if (done) return;

/\* routine to determine if a variable (varble) is a member of the

conclusion list (conclt). if yes return sn != 0.

if not a member sn=0;

\*/

/\* initially set to not a member \*/

statementNumber = 0;

/\* member of conclusion list to be searched is f \*/

if (debug) cout << "Value of f is currently " << f << endl;

int i = f;

//while((varble!=conclusionList[i]) && (i<20))

while(strcmp(varble.c\_str(), conclusionList[i].c\_str()) != 0 && i < 21)

i=i+1; /\* test for membership \*/

//if (varble == conclusionList[i])

if (strcmp(varble.c\_str(), conclusionList[i].c\_str()) == 0) {

statementNumber = i; /\* a member \*/

if (debug) cout << varble << " found at " << i << endl;

}

else

if (debug) cout << varble << " not found in conclussion list" << endl;

}

void BC::push\_on\_stack()

{

if (debug) cout << "Push\_on\_stack() called" << endl;

if (done) return;

stackPointer=stackPointer-1;

if (debug) cout << "Stack pointer at " << stackPointer << endl;

statementStack[stackPointer] = statementNumber;

if (debug){

cout << "Statement stack:\t";

for (int i = 1; i < size; i++) cout << statementStack[i] << "|";

cout << endl;

}

clauseStack[stackPointer] = 1;

if (debug){

cout << "Clause stack:\t\t";

for (int i = 1; i < size; i++) cout << clauseStack[i] << "|";

cout << endl;

}

}

void BC::instantiate()

{

if (debug) cout << "Instantiate called for " << varble << endl;

if (done) return;

int i=1;

/\* find variable in the list \*/

while ((strcmp(varble.c\_str(), variableList[i].c\_str()) != 0) && i<18)

i=i+1;

//if (varble == variableList[i] && instantiatedList[i] != 1)

if (/\*strcmp(varble.c\_str(), variableList[i].c\_str()) == 0) &&\*/i<18 && instantiatedList[i] != 1)

{

/\*found variable and not already instantiated \*/

/\*mark instantiated \*/

instantiatedList[i]=1;

/\* the designer of the knowledge base places the input statements to

instantiate the variables below in the case statement \*/

initkbase(i);

}

else

if (debug) cout << "Variable already has a value." << endl;

}

void BC::initkbase(int i)

{

if (debug) cout << "InitKBase() called for " << variableList[i] << " with index " << i << endl;

if (debug) cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;

switch (i)

{

/\*\*\*\*\* input statement \*\*\*\*\*/

/\* input statements for sample position knowledge

base \*/

/\*\*\*\*\* comment 1700 \*\*\*\*\*\*/

//case 1: printf("INPUT YES OR NO FOR DE-? ");

// gets(de);

case 1: cout << "What degree are you working towards?";

cin >> degree;

cout << endl;

break;

//case 2: printf("INPUT YES OR NO FOR DI-? ");

// gets(di);

case 2: cout << "What GPA do you maintain? ";

if (debug) cout << "Bad value in initKBase()" << endl;

cin >> grade;

if (grade >= 3.5)

goodGrades.assign("true");

else

goodGrades.assign("false");

break;

//case 3: printf("INPUT A REAL NUMBER FOR EX-? ");

// scanf("%f", &ex);

case 3: cout << "How many courses will you take that include a lab section?";

if (debug) cout << "Bad value in initKBase()" << endl;

cin >> coursesWithLabs;

break;

case 4:// printf("INPUT A REAL NUMBER FOR GR-? ");

// scanf("%f", &gr);

if (debug) cout << "Bad value in initKBase()" << endl;

cout << "How many group projects have you been the leader off?";

cin >> groupLeader;

if (groupLeader >= 1) leadership.assign("true");

else leadership.assign("false");

break;

case 5: cout << "Will you go to med school after college? ";

cin >> buff;

if (buff.compare("yes") == 0 || buff.compare("y") == 0)

medSchool.assign("true");

else medSchool.assign("false");

break;

/\*\*\*\*\* comment 1715 \*\*\*\*/

case 6: cout << "Do you like working in groups? ";

cin >> buff;

if (buff.compare("yes") == 0 || buff.compare("y") == 0)

groupWork.assign("true");

else groupWork.assign("false");

break;

case 7: if (debug) cout << "Bad value in initKBase()" << endl;

cout << "Do you like working in groups? ";

cin >> buff;

if (buff.compare("yes") == 0 || buff.compare("y") == 0)

workAlone.assign("false");

else workAlone.assign("true");

break;

case 8: if (debug) cout << "Bad value in initKBase()" << endl;

cout << "How many hours do you spend outside every week? ";

cin >> hoursOutside;

if (hoursOutside > 15)

outdoorWork.assign("true");

else

outdoorWork.assign("false");

break;

case 9: if (debug) cout << "Bad value in initKBase()" << endl;

cout << "Will you get a medical certification? ";

cin >> buff;

if (buff.compare("yes") == 0 || buff.compare("y") == 0)

medCert.assign("true");

else medCert.assign("false");

break;

case 10: if (debug) cout << "Bad value in initKBase()" << endl;

cout<< "Will you get a Teaching certification? ";

cin>>buff;

if(buff.compare("yes")==0||buff.compare("y")==0)

teachCert.assign("true");

else

teachCert.assign("false");

break;

case 11: cout<<"What is your GPA?";

cin >> grade;

if (debug) cout << "Grade is " << grade << endl;

break;

case 12 : if (debug) cout << "Bad value in initKBase()" << endl;

cout<< "What do you want to do?";

cin>>profession;

break;

case 13: cout << "How many courses will you take that include a lab section?";

cin >> coursesWithLabs;

break;

case 14: cout<< "How many projects have you been a group leader for?";

cin>>groupLeader;

break;

case 15: cout<<"How many hours are you outside per week? " ;

cin>>hoursOutside;

break;

case 16: cout<<"Are you interested in practicing health care?";

cin>>buff;

if(buff.compare("yes")==0||buff.compare("y")==0)

medField.assign("true");

else

medField.assign("false");

break;

case 17 : cout<<"Do you have a criminal background?";

cin>>buff;

if(buff.compare("yes")==0||buff.compare("y")==0)

criminal.assign("true");

else

criminal.assign("false");

break;

default: cout<<"This is not a valid input to initKBase()";

break;

} //end of switch

}

void BC::B545()

{

int i;

if (done) return;

if (debug) cout << "B545() called" << endl;

if (debug) cout << "Look at clause variable: " << (statementStack[stackPointer] -1) \*4 + clauseStack[stackPointer] << endl;

do

{

/\* calculate clause location in clause-variable list \*/

//B545:

i= (statementStack[stackPointer] -1) \*4 + clauseStack[stackPointer];

if (debug)

if (i == 69){

cout << "Should be looking for degree." << endl;

cout << "But I'm looking for: " << clauseVariableList[i] << endl;

}

/\* clause variable \*/

//varble = clauseVariableList[i];

varble.assign(clauseVariableList[i]);

if(strcmp(varble.c\_str(), "") != 0)

{

/\*is this clause variable a conclusion? \*/

f = 1;

determine\_member\_concl\_list();

if(statementNumber != 0)

B520(); /\* it is a conclusion push it \*/

/\* check instantiation of this clause \*/

instantiate();

clauseStack[stackPointer] = clauseStack[stackPointer] + 1;

}

} while(strcmp(varble.c\_str(), "") != 0);

statementNumber = statementStack[stackPointer];

statementActive = 0;

if (debug){

cout << "Values evaluated at B545: " << endl;

cout << "Degree: " << degree << endl;

cout << "Good Grades: " << goodGrades << endl;

cout << "Lab Work: " << labwork << endl;

cout << "Leadership: " << leadership << endl;

cout << "Group Work: " << groupWork << endl;

cout << "Work Alone: " << workAlone << endl;

cout << "Courses with Labs: " << coursesWithLabs << endl;

cout << "Statement Number: " << statementNumber << endl;

cout << "Med Field: " << medField << endl;

cout << "Med Cert: " << medCert << endl;

cout << "Hours Outside: " << hoursOutside << endl;

cout << "Outdoor Work: " << outdoorWork << endl;

cout << "Criminal Background: " << criminal << endl;

cout << "Teaching Certificate: " << teachCert << endl;

//getchar();

}

switch (statementNumber) { //Left hand side of Rules

/\* if part of statement 1 \*/

/\*\*\*\*\*\* comment 1500 \*\*\*\*/

case 1: //if(strcmp(degree.c\_str(), "NO") == 0) s = 1;

if (grade >= 3.5) {

statementActive = 1;

if (debug) cout << "Rule 1 satisfied" << endl;

}

break;

/\* if part of statement 2 \*/

/\*\*\*\*\* comment 1510 \*\*\*\*\*\*/

case 2: //if(strcmp(degree.c\_str(), "YES") == 0) s = 1;

if (grade < 3.5) {

statementActive = 1;

if (debug) cout << "Rule 2 satisfied" << endl;

}

break;

/\* if part of statement 3 \*/

case 3: //if((strcmp(degree.c\_str(), "YES") == 0) &&

// (strcmp(labwork.c\_str(), "YES") == 0)) s =1;

if (coursesWithLabs > 2) {

statementActive = 1;

if (debug) cout << "Rule 3 satisfied positively" << endl;

}

else {

statementActive = 1;

if (debug) cout << "Rule 3 satisfied negatively" << endl;

}

break;

/\* if part of statement 4 \*/

/\*\*\*\*\*\*\*\* comment 1560 \*\*\*\*\*\*/

case 4: //if((strcmp(goodGrades.c\_str(), "YES") == 0) &&

// (grade<3.5) && (grade >= 2)) s = 1;

if (groupLeader >= 1){

statementActive = 1;

if (debug) cout << "Rule 4 satisfied positively" << endl;

}

else {

statementActive = 1;

if (debug) cout << "Rule 4 satisfied negatively" << endl;

}

break;

/\*\*\*\*\*\*\*\* comment 1570 \*\*\*\*\*\*\*\*/

/\* if part of statement 5 \*/

case 5: //if((strcmp(goodGrades.c\_str(), "YES") == 0) &&

// (grade<3) && (grade<2)) s = 1;

if (strcmp(degree.c\_str(), "engineering") == 0 && strcmp(goodGrades.c\_str(), "true") == 0){

statementActive = 1;

if (debug) cout << "Rule 5 satisfied" << endl;

}

break;

/\* if part of statement 6 \*/

case 6: //if((strcmp(goodGrades.c\_str(), "YES") == 0) &&

// (grade >=3.5)) s = 1;

if (strcmp(degree.c\_str(), "science") == 0 && strcmp(goodGrades.c\_str(), "true") == 0 && strcmp(labwork.c\_str(), "true") == 0){

statementActive = 1;

if (debug) cout << "Rule 6 satisfied" << endl;

}

break;

case 7: if(strcmp(degree.c\_str(), "none") != 0 && strcmp(leadership.c\_str(), "true") == 0){

statementActive = 1;

if (debug) cout << "Rule 7 satisfied" << endl;

}

break;

case 8: if (strcmp(groupWork.c\_str(), "false") == 0){

statementActive = 1;

if (debug) cout << "Rule 8 satisfied positively" << endl;

}

else {

statementActive = 1;

if (debug) cout << "Rule 8 satisfied negatively" << endl;

}

break;

case 9: if (strcmp(degree.c\_str(), "none") != 0 && strcmp(medSchool.c\_str(), "true") == 0 && strcmp(groupWork.c\_str(), "true") == 0) {

statementActive = 1;

if (debug) cout << "Rule 9 satisfied" << endl;

}

break;

case 10: if (strcmp(degree.c\_str(), "english") == 0 && strcmp(workAlone.c\_str(), "true") == 0) {

statementActive = 1;

if (debug) cout << "Rule 10 satisfied" << endl;

}

break;

case 11: if (hoursOutside >= 16) {

statementActive = 1;

if (debug) cout << "Rule 11 satisfied positively" << endl;

}

else {

statementActive = 1;

if (debug) cout << "Rule 11 satisfied negatively" << endl;

}

break;

case 12: if (strcmp(degree.c\_str(), "science") == 0 && strcmp(outdoorWork.c\_str(), "true") == 0) {

statementActive = 1;

if (debug) cout << "Rule 12 satisfied" << endl;

}

break;

case 13: if (strcmp(degree.c\_str(), "psychology") == 0 && strcmp(groupWork.c\_str(), "true") == 0) {

statementActive = 1;

if (debug) cout << "Rule 13 satisfied" << endl;

}

break;

case 14: if (strcmp(degree.c\_str(), "none") == 0 && strcmp(outdoorWork.c\_str(), "true") == 0) {

statementActive = 1;

if (debug) cout << "Rule 14 satisfied" << endl;

}

break;

case 15: if (strcmp(medSchool.c\_str(), "false") == 0 && strcmp(medField.c\_str(), "true") == 0) {

statementActive = 1;

if (debug) cout << "Rule 15 satisfied positively" << endl;

}

else {

statementActive = 1;

if (debug) cout << "Rule 15 satisfied negatively" << endl;

}

break;

case 16: if (strcmp(degree.c\_str(), "none") != 0 && strcmp(medCert.c\_str(), "true") == 0 && strcmp(groupWork.c\_str(), "true") == 0) {

statementActive = 1;

if (debug) cout << "Rule 16 satisfied" << endl;

}

break;

case 17: if (strcmp(criminal.c\_str(), "true") == 0) {

statementActive = 1;

if (debug) cout << "Rule 17 satisfied positively" << endl;

}

else {

statementActive = 1;

if (debug) cout << "Rule 17 satisfied negatively" << endl;

}

break;

case 18: if (strcmp(degree.c\_str(), "none") != 0 && strcmp(teachCert.c\_str(), "true") == 0) {

statementActive = 1;

if (debug) cout << "Rule 18 satisfied" << endl;

}

break;

case 19: if (strcmp(degree.c\_str(), "none") == 0 && strcmp(outdoorWork.c\_str(), "false") == 0) {

statementActive = 1;

if (debug) cout << "Rule 19 satisfied" << endl;

}

break;

case 20: if (strcmp(degree.c\_str(), "cs") == 0 && strcmp(goodGrades.c\_str(), "true") == 0 && strcmp(groupWork.c\_str(), "true") == 0) {

statementActive = 1;

if (debug) cout << "Rule 20 satisfied" << endl;

}

break;

default : if (debug) cout << "Unusable statement number." << endl;

/\*\*\*\*\*\*\*\*\* comment 1680 \*\*\*\*\*\*/

} //end of switch

//ifkbase();

if( statementActive != 1)

{

/\* failed..search rest of statements for same conclusion \*/

/\* get conclusion \*/

i = statementStack[stackPointer];

//varble = conclusionList[i];

varble.assign(conclusionList[i]);

/\* search for conclusion starting at the next statement number \*/

f = statementStack[stackPointer] + 1;

determine\_member\_concl\_list();

stackPointer = stackPointer+1;

}

}

void BC::InBetweenFunction()

{

if (debug) cout << "InBetweenFunction called" << endl;

if (debug) cout << "Left side of rule " << statementNumber << " satisfied." << endl;

switch (statementNumber)

{

/\* then part of statement 1 \*/

/\*\*\*\*\*\*\* comment 1500 \*\*\*\*\*\*\*/

case 1: //strcpy(po, "NO");

//printf("PO=NO\n");

goodGrades.assign("true");

if (debug) cout<<"Good Grades = true" << endl;

instantiatedList[2] = 1;

break;

/\* then part of statement 2 \*/

/\*\*\*\*\*\* comment 1510 \*\*\*\*\*\*/

case 2: //strcpy(qu, "YES");

//printf("QU=YES\n");

goodGrades.assign("false");

if (debug)cout<<"Good Grades = false" << endl;

instantiatedList[2] = 1;

break;

/\* then part of statement 3 \*/

case 3: if (coursesWithLabs > 2){

labwork.assign("true");

if (debug)cout<<"LAB WORK = true" << endl;

}

else {

labwork.assign("false");

if (debug)cout<<"LAB WORK = false" << endl;

}

instantiatedList[3] = 1;

break;

/\* then part of statement 4 \*/

/\*\*\*\*\*\*\*\* comment 1560 \*\*\*\*\*\*/

case 4: if (groupLeader >= 1){

leadership.assign("true");

if (debug)cout<<"leadership = true" << endl;

}

else {

leadership.assign("false");

if (debug)cout<<"leadership = false" << endl;

}

instantiatedList[4] = 1;

break;

/\* then part of statement 5 \*/

/\*\*\*\*\*\* comment 1570 \*\*\*\*\*/

case 5: profession.assign("engineering");

if (debug)cout<<"Profession = Engineering" << endl;

instantiatedList[12] = 1;

done = true;

break;

/\* then part of statement 6 \*/

case 6: profession.assign("science");

if (debug)cout<<"Profession = Science" << endl;

instantiatedList[12] = 1;

done = true;

break;

case 7: profession.assign("business");

if (debug)cout<<"Profession = Business";

instantiatedList[12] = 1;

done = true;

break;

case 8: if (strcmp(groupWork.c\_str(), "false") == 0){

workAlone.assign("true");

if (debug)cout<<"workAlone = true" << endl;

}

else {

workAlone.assign("false");

if (debug)cout<<"workAlone = false" << endl;

}

instantiatedList[7] = 1;

break;

case 9: profession.assign("medical");

if (debug)cout<<"Profession = Medical" << endl;

instantiatedList[12] = 1;

done = true;

break;

case 10 : profession.assign("english");

if (debug)cout<<"Profession = English" << endl;

instantiatedList[12] = 1;

done = true;

break;

case 11: if (hoursOutside >= 16) {

outdoorWork.assign("true");

if (debug)cout<<"outdoorWork = true" << endl;

}

else {

outdoorWork.assign("false");

if (debug)cout<<"outdoorWork = false" << endl;

}

instantiatedList[8] = 1;

break;

case 12 : profession.assign("geography");

if (debug)cout<<"Profession = geography";

done = true;

instantiatedList[12] = 1;

break;

case 13 : profession.assign("psychology");

if (debug)cout<<"Profession = psychology" << endl;

done = true;

instantiatedList[12] = 1;

break;

case 14 : profession.assign("agriculture");

if (debug)cout<<"Profession = agriculture" << endl;

instantiatedList[12] = 1;

done = true;

break;

case 15: if (strcmp(medField.c\_str(), "true") == 0 && strcmp(medSchool.c\_str(), "false") == 0){

medCert.assign("true");

if (debug)cout<<"medCert = true" << endl;

}

else {

medCert.assign("false");

if (debug)cout<<"medCert = false" << endl;

}

instantiatedList[9] = 1;

instantiatedList[16] = 1;

break;

case 16 : profession.assign("healthcare");

if (debug)cout<<"Profession = Health Care" << endl;

instantiatedList[12] = 1;

done = true;

break;

case 17: if (strcmp(criminal.c\_str(), "true") == 0) {

teachCert.assign("false");

if (debug)cout<<"Teaching certification = false" << endl;

}

else {

teachCert.assign("true");

if (debug)cout<<"Teaching certification = true" << endl;

}

instantiatedList[10] = 1;

instantiatedList[17] = 1;

break;

case 18: profession.assign("education");

if (debug)cout<<"Profession = education" << endl;

instantiatedList[12] = 1;

done = true;

break;

case 19: profession.assign("propertymanagement");

if (debug)cout<<"Profession = property management" << endl;

instantiatedList[12] = 1;

done = true;

break;

case 20: profession.assign("cs");

if (debug)cout<<"Profession = CS" << endl;

instantiatedList[12] = 1;

done = true;

break;

/\*\*\*\*\*\* comment 1680 \*\*\*\*\*\*\*\*/

} //end of switch

}

void BC::popStack()

{

/\* pop the stack \*/

if (debug) cout << "PopStack() called" << endl;

statementStack[stackPointer] = 0;

clauseStack[stackPointer] = 0;

stackPointer = stackPointer+1;

if (debug) cout << "StackPointer is now " << stackPointer << endl;

if(stackPointer >= size)

{

// Finished

if (debug){

cout<<"\*\*\* SUCCESS \*\*\*"<<endl;

getchar();

}

return;

}

else

{

/\* stack is not empty \*/

/\* get next clause then continue \*/

clauseStack[stackPointer] = clauseStack[stackPointer]+1;

B545();

if((statementActive != 1) && (statementNumber !=0))

{

KeepProcessing();

}

else if(statementNumber != 0)

{

InBetweenFunction();

popStack();

}

}

return;

}

void BC::printProfession(){

cout << profession << endl;

}

**7.3 Forward Chaining header file. (FC.h)**

class FC

{

private:

int flag;

bool debug, done;

bool inst = false;

static const int size = 61;

static const int clauseSize = 241;

std::string conditionVariableQueue[size], variableList[size],clauseVariableList[clauseSize],clause,varble;

int instantiatedList[size];/\* instantiated list\*/

std::string fedint,interest,stock,dollar,fedmon;

std::string likePhysics, likeMath, groupWork, likeBiology;

std::string likeChemistry, likeMarketing, likeManagement;

std::string goodCredit, likeMedicalEthics, stateLicensure;

std::string likeWriting, likeOriginality, likeMedia;

std::string likeSocialScience, likeEnglish;

std::string likeReading, likeClimate, likeStones;

std::string likeLand, likeSoil, likeWaterResources;

std::string likeAnalyticalSkills, likeInternship, likeAnatomy;

std::string likeChildren, reliable, likeRepairs;

std::string profession, area, likeLaw, likeCommunication;

int f, k, statementActive, frontPointer /\* front pointer \*/;

int backPointer /\* back pointer \*/, statementNumber; /\* statement number \*/

int clauseNumber; /\* clause number \*/

bool isYes(std::string);

void ifcondtions();

void Result();

void takingInput(int);

void search();

void check\_instantiation();

void instantiate();

void inference();

void gotoF();

public:

FC(std::string, bool);

void start();

std::string getArea() {return area;}

std::string getProfession() {return profession;}

void printProfession();

void printArea();

};

**7.4 Forward Chaining .cpp file (FC.cpp)**

void FC::start()

{

inference();

gotoF();

}

void FC::check\_instantiation()

{

if (done) return;

if (debug) cout << "Check\_instantiation called for " << varble << endl;

int i=1;

/\* find variable in the variable list \*/

//while ((varble != variableList[i]) && (i < size))

while((varble.compare(variableList[i]) != 0) && i < size)

i = i+1;

/\* check if already instantiated \*/

if (instantiatedList[i] != 1)

{

if (debug) cout << "This is a new variable." << endl;

/\* mark instantiated \*/

instantiatedList[i] = 1;

/\* the designer of this knowledge base places the input

statements to instantiate the variables in this case

statement \*/

takingInput(i);

}

else

if (debug) cout << "I already have a value for this variable." << endl;

}

void FC::instantiate()

{

if(done) return;

if (debug) cout << "instantiate called for " << varble << endl;

int i=1;

/\* find varialbe in the varialbe list (varlt) \*/

//while ((varble != variableList[i]) && (i < size))

while (varble.compare(variableList[i]) != 0 && i < size)

i=i+1;

/\* instantiate it \*/

instantiatedList[i] = 1;

i = 1;

/\* determine if (v) is or already has been on the queue (cndvar) \*/

//while ((varble != conditionVariableQueue[i]) && (i < size))

while (varble.compare(conditionVariableQueue[i]) != 0 && i < size)

i=i+1;

/\* variable has not been on the queue. Store it in the back of the queue \*/

//if (varble != conditionVariableQueue[i])

if (varble.compare(conditionVariableQueue[i]) != 0)

{

conditionVariableQueue[backPointer] = varble;

backPointer=backPointer+1;

}

}

void FC::inference()

{

if (debug) cout << "inference called" << endl;

frontPointer=1;

backPointer=1;

/\*\*\*\*\*\* INFERENCE SECTION \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

//printf("ENTER CONDITION VARIABLE? ");

//cin>>clause;

cout << "Your condition variable is Profession." << endl;

cout << "Your current profession is: " << profession << endl;

clause.assign("PROFESSION");

/\* place condition variable c on condition var queue cndvar \*/

//conditionVariableQueue[backPointer] = clause;

conditionVariableQueue[backPointer].assign(clause);

/\* move backpointer (bp) to back \*/

backPointer = backPointer + 1;

/\* set the condition variable pointer consisting of the

statement number (sn) and the clause number (cn) \*/

statementNumber = 1;

clauseNumber = 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 (clvarlt) \*/

/\* start at the beginning \*/

f=1;

}

void FC::gotoF()

{

if(done) return;

int i;

if (debug) cout << "goToF called" << endl;

search();

/\* point to first clause in statement \*/

clauseNumber=1;

if (statementNumber != 0)/\* more statements \*/

{

/\* locate the clause \*/

i = 4 \* (statementNumber-1) + clauseNumber;

/\* clause variable \*/

//varble = clauseVariableList[i];

varble.assign(clauseVariableList[i]);

/\* are there any more clauses for this statement \*/

//while (varble != "")

while (varble.compare("") != 0)

/\* more clauses \*/

{

/\* check instantiation of this clause \*/

check\_instantiation();

clauseNumber = clauseNumber+1;

/\* check next clause \*/

i = 4 \* (statementNumber-1) + clauseNumber;

//varble = clauseVariableList[i];

varble.assign(clauseVariableList[i]);

}

/\* no more clauses - check IF part of statement \*/

statementActive = 0;

ifcondtions();

/\* see if the THEN part should be inovked, i.e., s=1 \*/

if (statementActive != 1)

{

f = statementNumber + 1;

gotoF();

}

Result();

if(inst)

{

instantiate();

inst = false;

}

f = statementNumber + 1;

gotoF();

}

/\* no more clauses in the clause variable list (clvarlt)

containing the variable in front of the queue (cndvar(fp))

then remove front variable (cndvar(fp)) and replace it by

the next variable (cndvar(fp+1)). If no more variables are

at the front of the queue, stop. \*/

/\* next queue variable \*/

frontPointer=frontPointer+1;

if (frontPointer < backPointer)

{

/\* check out the condition variable \*/

f = 1;

gotoF();

}

/\* no more conclusion variables on queue \*/

if (debug)cout<<"\*\*\* Success \*\*\*" << endl;

//exit(1);

return;

}

void FC::search()

{

if (debug) cout << "search called" << endl;

if (debug) cout << "current front of queue is " << conditionVariableQueue[frontPointer] << endl;

flag = 0;

statementNumber = f;

while ((flag == 0) && (statementNumber <= (clauseSize-1)/4))

{

clauseNumber=1;

k = (statementNumber-1)\*4+clauseNumber;

//while ((clauseVariableList[k] != conditionVariableQueue[frontPointer]) && (clauseNumber < 4))

while(clauseVariableList[k].compare(conditionVariableQueue[frontPointer]) != 0 && clauseNumber < 4)

{

clauseNumber = clauseNumber+1;

k = (statementNumber-1)\*4+clauseNumber;

}

//if (clauseVariableList[k] == conditionVariableQueue[frontPointer])

if (clauseVariableList[k].compare(conditionVariableQueue[frontPointer]) == 0)

flag = 1;

if (flag == 0)

statementNumber = statementNumber+1;

}

if (flag == 0)

statementNumber=0;

if (debug) cout << "SN after search is " << statementNumber << endl;

}

FC::FC(string profession, bool debug)

{

/\*\*\*\*\*\*\*\* INITIALIZATION SECTION \*\*\*\*\*\*\*\*\*\*\*/

for (int i=1;i < clauseSize; i++)

//clauseVariableList[i] = "";

clauseVariableList[i].assign("");

for (int i=1;i < size; i++)

{

//conditionVariableQueue[i] = "";

conditionVariableQueue[i].assign("");

//variableList[i] = "";

variableList[i].assign("");

instantiatedList[i] = 0;

}

/\* enter variables which are in the IF part, 1 at a time in

the exact order that they occur. Up to 3 variables per

IF statement. Do not duplicate any variable names. Any

name is used only once. If no more variables left, just

hit return key \*/

/\*\*\*\*\*\* comment 367 \*\*\*\*\*\*\*\*\*\*\*\*\*/

//variableList[1] = "IN";

//variableList[2] = "DO";

//variableList[3] = "FT";

//variableList[4] = "FM";

variableList[1].assign("LIKEPHYSICS");

variableList[2].assign("LIKEMATH");

variableList[3].assign("GROUPWORK");

variableList[4].assign("LIKEBIOLOGY");

variableList[5].assign("LIKECHEMISTRY");

variableList[6].assign("LIKEMARKETING");

variableList[7].assign("LIKEMANAGEMENT");

variableList[8].assign("GOODCREDIT");

variableList[9].assign("LIKEMEDICALETHICS");

variableList[10].assign("STATELICENSURE");

variableList[11].assign("LIKEWRITING");

variableList[12].assign("LIKEORIGINALITY");

variableList[13].assign("LIKEMEDIA");

variableList[14].assign("LIKESOCIALSCIENCE");

variableList[15].assign("LIKEENGLISH");

variableList[16].assign("LIKEREADING");

variableList[17].assign("LIKECLIMATE");

variableList[18].assign("LIKESTONES");

variableList[19].assign("LIKELAND");

variableList[20].assign("LIKESOIL");

variableList[21].assign("LIKEWATERRESOURCES");

variableList[22].assign("LIKEANALYTICALSKILLS");

variableList[23].assign("LIKEINTERNSHIP");

variableList[24].assign("LIKEANATOMY");

variableList[25].assign("LIKECHILDREN");

variableList[26].assign("RELIABLE");

variableList[27].assign("LIKEREPAIRS");

variableList[28].assign("PROFESSION");

variableList[29].assign("AREA");

variableList[30].assign("LIKELAW");

variableList[31].assign("LIKECOMMUNICATION");

cout<<"\*\*\* VARIABLE LIST \*\*\*"<<endl;

for (int i=1;i < size; i++) {

if (variableList[i].compare("") == 0) break;

cout<<"VARIABLE "<<i<<" "<<variableList[i]<<endl;

}

cout<<"HIT RETURN TO CONTINUE";

getchar();

/\* enter variables as they appear in the IF clauses, Up to 3

variables per IF statement. If no more variables left, just

hit return key \*/

/\*\*\*\*\*\* comment 407, 408 \*\*\*\*\*\*\*\*\*\*\*\*\*/

//clauseVariableList[1] = "IN";

//clauseVariableList[5] = "IN";

//clauseVariableList[9] = "DO";

//clauseVariableList[13] = "DO";

//clauseVariableList[17] = "FT";

//clauseVariableList[18] = "FM";

clauseVariableList[1].assign("PROFESSION");

clauseVariableList[2].assign("LIKEMATH");

clauseVariableList[3].assign("LIKEPHYSICS");

clauseVariableList[5].assign("PROFESSION");

clauseVariableList[6].assign("LIKEPHYSICS");

clauseVariableList[7].assign("GROUPWORK");

clauseVariableList[9].assign("PROFESSION");

clauseVariableList[10].assign("LIKELAW");

clauseVariableList[11].assign("LIKEPHYSICS");

clauseVariableList[13].assign("PROFESSION");

clauseVariableList[14].assign("LIKEMATH");

clauseVariableList[15].assign("LIKEBIOLOGY");

clauseVariableList[17].assign("PROFESSION");

clauseVariableList[18].assign("LIKECHEMISTRY");

clauseVariableList[19].assign("GROUPWORK");

clauseVariableList[21].assign("PROFESSION");

clauseVariableList[22].assign("LIKEBIOLOGY");

clauseVariableList[23].assign("LIKECHEMISTRY");

clauseVariableList[25].assign("PROFESSION");

clauseVariableList[26].assign("LIKEBIOLOGY");

clauseVariableList[27].assign("GROUPWORK");

clauseVariableList[29].assign("PROFESSION");

clauseVariableList[30].assign("LIKEMATH");

clauseVariableList[33].assign("PROFESSION");

clauseVariableList[34].assign("LIKEPHYSICS");

clauseVariableList[37].assign("PROFESSION");

clauseVariableList[38].assign("LIKEBIOLOGY");

clauseVariableList[39].assign("LIKEPHYSICS");

clauseVariableList[41].assign("PROFESSION");

clauseVariableList[42].assign("GOODCREDIT");

clauseVariableList[45].assign("PROFESSION");

clauseVariableList[46].assign("LIKEMARKETING");

clauseVariableList[47].assign("LIKEMANAGEMENT");

clauseVariableList[49].assign("PROFESSION");

clauseVariableList[50].assign("LIKEMATH");

clauseVariableList[53].assign("PROFESSION");

clauseVariableList[54].assign("GROUPWORK");

clauseVariableList[57].assign("PROFESSION");

clauseVariableList[58].assign("GOODCREDIT");

clauseVariableList[59].assign("LIKEMANAGEMENT");

clauseVariableList[61].assign("PROFESSION");

clauseVariableList[62].assign("GROUPWORK");

clauseVariableList[65].assign("PROFESSION");

clauseVariableList[66].assign("LIKECHEMISTRY");

clauseVariableList[69].assign("PROFESSION");

clauseVariableList[70].assign("STATELICENSURE");

clauseVariableList[71].assign("LIKECHEMISTRY");

clauseVariableList[73].assign("PROFESSION");

clauseVariableList[74].assign("LIKEMEDICALETHICS");

clauseVariableList[77].assign("PROFESSION");

clauseVariableList[78].assign("LIKEBIOLOGY");

clauseVariableList[79].assign("LIKECHEMISTRY");

clauseVariableList[81].assign("PROFESSION");

clauseVariableList[82].assign("LIKEWRITING");

clauseVariableList[83].assign("LIKEORIGINALITY");

clauseVariableList[85].assign("PROFESSION");

clauseVariableList[86].assign("LIKEMEDIA");

clauseVariableList[87].assign("LIKESOCIALSCIENCE");

clauseVariableList[89].assign("PROFESSION");

clauseVariableList[90].assign("LIKEENGLISH");

clauseVariableList[91].assign("LIKEREADING");

clauseVariableList[93].assign("PROFESSION");

clauseVariableList[94].assign("LIKEMARKETING");

clauseVariableList[95].assign("LIKEENGLISH");

clauseVariableList[97].assign("PROFESSION");

clauseVariableList[98].assign("LIKEREADING");

clauseVariableList[99].assign("LIKEORIGINALITY");

clauseVariableList[101].assign("PROFESSION");

clauseVariableList[102].assign("LIKECLIMATE");

clauseVariableList[103].assign("LIKESTONES");

clauseVariableList[105].assign("PROFESSION");

clauseVariableList[106].assign("LIKELAND");

clauseVariableList[107].assign("LIKEORIGINALITY");

clauseVariableList[109].assign("PROFESSION");

clauseVariableList[110].assign("LIKESOIL");

clauseVariableList[111].assign("LIKECLIMATE");

clauseVariableList[113].assign("PROFESSION");

clauseVariableList[114].assign("LIKEWATERESOURCES");

clauseVariableList[115].assign("LIKESOIL");

clauseVariableList[117].assign("PROFESSION");

clauseVariableList[118].assign("LIKEMATH");

clauseVariableList[121].assign("PROFESSION");

clauseVariableList[122].assign("LIKESOCIALSCIENCE");

clauseVariableList[125].assign("PROFESSION");

clauseVariableList[126].assign("GROUPWORK");

clauseVariableList[127].assign("LIKEANALYTICALSKILLS");

clauseVariableList[129].assign("PROFESSION");

clauseVariableList[130].assign("LIKEINTERNSHIP");

clauseVariableList[133].assign("PROFESSION");

clauseVariableList[134].assign("GROUPWORK");

clauseVariableList[135].assign("LIKESOCIALSCIENCE");

clauseVariableList[137].assign("PROFESSION");

clauseVariableList[138].assign("LIKECOMMUNICATION");

clauseVariableList[139].assign("LIKEANALYTICALSKILLS");

clauseVariableList[141].assign("PROFESSION");

clauseVariableList[142].assign("GOODCREDIT");

clauseVariableList[143].assign("LIKEMANAGEMENT");

clauseVariableList[145].assign("PROFESSION");

clauseVariableList[146].assign("GROUPWORK");

clauseVariableList[147].assign("LIKEMANAGEMENT");

clauseVariableList[149].assign("PROFESSION");

clauseVariableList[150].assign("LIKEMANAGEMENT");

clauseVariableList[151].assign("LIKEMARKETING");

clauseVariableList[153].assign("PROFESSION");

clauseVariableList[154].assign("LIKECHEMISTRY");

clauseVariableList[157].assign("PROFESSION");

clauseVariableList[158].assign("LIKEWATERRESOURCES");

clauseVariableList[161].assign("PROFESSION");

clauseVariableList[162].assign("LIKECOMMUNICATION");

clauseVariableList[165].assign("PROFESSION");

clauseVariableList[166].assign("LIKEMANAGEMENT");

clauseVariableList[169].assign("PROFESSION");

clauseVariableList[170].assign("LIKEIOLOGY");

clauseVariableList[173].assign("PROFESSION");

clauseVariableList[174].assign("LIKEMANAGEMENT");

clauseVariableList[175].assign("GROUPWORK");

clauseVariableList[177].assign("PROFESSION");

clauseVariableList[178].assign("LIKEANATOMY");

clauseVariableList[179].assign("LIKECHEMISTRY");

clauseVariableList[181].assign("PROFESSION");

clauseVariableList[182].assign("LIKECHILDREN");

clauseVariableList[185].assign("PROFESSION");

clauseVariableList[186].assign("LIKEREADING");

clauseVariableList[187].assign("LIKECOMMUNICATION");

clauseVariableList[189].assign("PROFESSION");

clauseVariableList[190].assign("LIKEMATH");

clauseVariableList[191].assign("LIKECHEMISTRY");

clauseVariableList[193].assign("PROFESSION");

clauseVariableList[194].assign("LIKECOMMUNICATION");

clauseVariableList[195].assign("LIKECHILDREN");

clauseVariableList[197].assign("PROFESSION");

clauseVariableList[198].assign("LIKESOCIALSCIENCE");

clauseVariableList[199].assign("LIKECOMMUNICATION");

clauseVariableList[201].assign("PROFESSION");

clauseVariableList[202].assign("LIKEMANAGEMENT");

clauseVariableList[205].assign("PROFESSION");

clauseVariableList[206].assign("GROUPWORK");

clauseVariableList[209].assign("PROFESSION");

clauseVariableList[210].assign("RELIABLE");

clauseVariableList[213].assign("PROFESSION");

clauseVariableList[214].assign("LIKEREPAIRS");

clauseVariableList[217].assign("PROFESSION");

clauseVariableList[218].assign("LIKECOMMUNICATION");

clauseVariableList[221].assign("PROFESSION");

clauseVariableList[222].assign("LIKEMATH");

clauseVariableList[223].assign("LIKESOCIALSCIENCE");

clauseVariableList[225].assign("PROFESSION");

clauseVariableList[226].assign("LIKEMATH");

clauseVariableList[227].assign("LIKEANALYTICALSKILLS");

clauseVariableList[229].assign("PROFESSION");

clauseVariableList[230].assign("GROUPWORK");

clauseVariableList[233].assign("PROFESSION");

clauseVariableList[234].assign("LIKEMATH");

clauseVariableList[235].assign("LIKEMARKETING");

clauseVariableList[237].assign("PROFESSION");

clauseVariableList[238].assign("LIKEORIGINALITY");

printf("\*\*\* CLAUSE-VARIABLE LIST \*\*\*\n");

for (int i = 1; i <= (clauseSize-1)/4 ; i++)

{

printf("\*\* CLAUSE %d\n", i);

for (int j = 1; j < 5; j++)

{

k = 4 \* (i - 1) + j;

cout<<"VARIABLE "<<j<< " "<<clauseVariableList[k]<<endl;

}

if (i==61)

{

cout<<"HIT RETURN TO CONTINUE";

getchar();

}

}

this->debug = debug;

this->profession.assign(profession);

done = false;

instantiatedList[28] = 1;

likePhysics.assign("");

likeMath.assign("");

groupWork.assign("");

likeBiology.assign("");

likeChemistry.assign("");

likeMarketing.assign("");

likeManagement.assign("");

goodCredit.assign("");

likeMedicalEthics.assign("");

stateLicensure.assign("");

likeWriting.assign("");

likeOriginality.assign("");

likeMedia.assign("");

likeSocialScience.assign("");

likeEnglish.assign("");

likeReading.assign("");

likeClimate.assign("");

likeStones.assign("");

likeLand.assign("");

likeSoil.assign("");

likeWaterResources.assign("");

likeAnalyticalSkills.assign("");

likeInternship.assign("");

likeAnatomy.assign("");

likeChildren.assign("");

reliable.assign("");

likeRepairs.assign("");

area.assign("");

likeLaw.assign("");

likeCommunication.assign("");

}

void FC::ifcondtions()

{

if (debug){

cout << "ifconditions called for SN: " << statementNumber << endl;

cout << "current values for ifs: " << endl;

cout << "Profession: " << profession << endl;

cout << "Like Math: " << likeMath << endl;

cout << "Like Physics: " << likePhysics << endl;

}

/\* sample IF-THEN statements from the position knowledge base \*/

//switch(statementNumber)

//{

/\* statement 1 \*/

/\*\*\*\*\* comment 1500 \*\*\*\*\*/

// case 1:

// if (interest == "FALL")

// statementActive=1;

// break;

// /\* statement 2 \*/

// /\*\*\*\*\* comment 1510 \*\*\*\*\*/

// case 2:

// if (interest == "RISE")

// statementActive=1;

// break;

// /\* statement 3 \*/

// /\*\*\*\*\* comment 1540 \*\*\*\*\*/

// case 3:

// if (dollar == "FALL")

// statementActive=1;

// break;

// /\* statement 4 \*/

// /\*\*\*\*\* comment 1550 \*\*\*\*\*/

// case 4:

// if (dollar == "RISE")

// statementActive=1;

// break;

// /\* statement 5 \*/

// case 5:

// if ((fedint == "FALL") && (fedmon == "ADD"))

// statementActive=1;

// break;

// /\*\*\*\*\* comment 1610 \*\*\*\*\*/

//}

switch(statementNumber){

case 1:

if (profession.compare("engineering") == 0 && isYes(likeMath) && isYes(likePhysics)){

statementActive = 1;

if (debug) cout << "Rule 1 satisfied." << endl;

}

break;

case 2:

if(profession.compare("engineering") == 0 && isYes(likePhysics) && isYes(groupWork)){

statementActive = 1;

if(debug) cout << "Rule 2 satisfied" << endl;

}

break;

case 3:

if(profession.compare("engineering") == 0 && isYes(likeLaw) && isYes(likePhysics)){

statementActive = 1;

if(debug) cout << "Rule 3 satisfied" << endl;

}

break;

case 4:

if(profession.compare("engineering") == 0 && isYes(likeMath) && isYes(likeBiology)){

statementActive = 1;

if(debug) cout << "Rule 4 satisfied" << endl;

}

break;

case 5:

if(profession.compare("engineering") == 0 && isYes(likeChemistry) && isYes(groupWork)){

statementActive = 1;

if(debug) cout << "Rule 5 satisfied" << endl;

}

break;

case 6:

if(profession.compare("science") == 0 && isYes(likeBiology) && isYes(likeChemistry)){

statementActive = 1;

if(debug) cout << "Rule 6 satisfied" << endl;

}

break;

case 7:

if(profession.compare("science") == 0 && isYes(likeBiology) && isYes(groupWork)){

statementActive = 1;

if(debug) cout << "Rule 7 satisfied" << endl;

}

break;

case 8:

if(profession.compare("science") == 0 && isYes(likeMath)){

statementActive = 1;

if(debug) cout << "Rule 8 satisfied" << endl;

}

break;

case 9:

if(profession.compare("science") == 0 && isYes(likePhysics)){

statementActive = 1;

if(debug) cout << "Rule 9 satisfied" << endl;

}

break;

case 10:

if(profession.compare("science") == 0 && isYes(likeBiology) && isYes(likePhysics)){

statementActive = 1;

if(debug) cout << "Rule 10 satisfied" << endl;

}

break;

case 11:

if(profession.compare("business") == 0 && isYes(goodCredit)){

statementActive = 1;

if(debug) cout << "Rule 11 satisfied" << endl;

}

break;

case 12:

if(profession.compare("business") == 0 && isYes(likeMarketing) && isYes(likeManagement)){

statementActive = 1;

if(debug) cout << "Rule 12 satisfied" << endl;

}

break;

case 13:

if(profession.compare("business") == 0 && isYes(likeMath)){

statementActive = 1;

if(debug) cout << "Rule 13 satisfied" << endl;

}

break;

case 14:

if(profession.compare("business") == 0 && isYes(groupWork)){

statementActive = 1;

if(debug) cout << "Rule 14 satisfied" << endl;

}

break;

case 15:

if(profession.compare("business") == 0 && isYes(goodCredit) && isYes(likeManagement)){

statementActive = 1;

if(debug) cout << "Rule 15 satisfied" << endl;

}

break;

case 16:

if(profession.compare("medical") == 0 && isYes(groupWork)){

statementActive = 1;

if(debug) cout << "Rule 16 satisfied" << endl;

}

break;

case 17:

if(profession.compare("medical") == 0 && isYes(likeChemistry)){

statementActive = 1;

if(debug) cout << "Rule 17 satisfied" << endl;

}

break;

case 18:

if(profession.compare("medical") == 0 && isYes(stateLicensure) && isYes(likeChemistry)){

statementActive = 1;

if(debug) cout << "Rule 18 satisfied" << endl;

}

break;

case 19:

if(profession.compare("medical") == 0 && isYes(likeMedicalEthics)){

statementActive = 1;

if(debug) cout << "Rule 19 satisfied" << endl;

}

break;

case 20:

if(profession.compare("medical") == 0 && isYes(likeBiology) && likeChemistry.compare("yes") == 0){

statementActive = 1;

if(debug) cout << "Rule 20 satisfied" << endl;

}

break;

case 21:

if(profession.compare("english") == 0 && isYes(likeWriting) && isYes(likeOriginality)){

statementActive = 1;

if(debug) cout << "Rule 21 satisfied" << endl;

}

break;

case 22:

if(profession.compare("engllish") == 0 && isYes(likeMedia) && isYes(likeSocialScience)){

statementActive = 1;

if(debug) cout << "Rule 22 satisfied" << endl;

}

break;

case 23:

if(profession.compare("english") == 0 && isYes(likeEnglish) && isYes(likeReading)){

statementActive = 1;

if(debug) cout << "Rule 23 satisfied" << endl;

}

break;

case 24:

if(profession.compare("english") == 0 && isYes(likeMarketing) && isYes(likeEnglish)){

statementActive = 1;

if(debug) cout << "Rule 24 satisfied" << endl;

}

break;

case 25:

if(profession.compare("english") == 0 && isYes(likeReading) && isYes(likeOriginality)){

statementActive = 1;

if(debug) cout << "Rule 25 satisfied" << endl;

}

break;

case 26:

if(profession.compare("geography") == 0 && isYes(likeClimate) && isYes(likeStones)){

statementActive = 1;

if(debug) cout << "Rule 26 satisfied" << endl;

}

break;

case 27:

if(profession.compare("geography") == 0 && isYes(likeLand) && isYes(likeOriginality)){

statementActive = 1;

if(debug) cout << "Rule 27 satisfied" << endl;

}

break;

case 28:

if(profession.compare("geography") == 0 && isYes(likeSoil) && isYes(likeClimate)){

statementActive = 1;

if(debug) cout << "Rule 28 satisfied" << endl;

}

break;

case 29:

if(profession.compare("geography") == 0 && isYes(likeWaterResources) && isYes(likeSoil)){

statementActive = 1;

if(debug) cout << "Rule 29 satisfied" << endl;

}

break;

case 30:

if(profession.compare("geography") == 0 && isYes(likeMath)){

statementActive = 1;

if(debug) cout << "Rule 30 satisfied" << endl;

}

break;

case 31:

if(profession.compare("psychology") == 0 && isYes(likeSocialScience)){

statementActive = 1;

if(debug) cout << "Rule 31 satisfied" << endl;

}

break;

case 32:

if(profession.compare("psychology") == 0 && isYes(groupWork) && isYes(likeAnalyticalSkills)){

statementActive = 1;

if(debug) cout << "Rule 32 satisfied" << endl;

}

break;

case 33:

if(profession.compare("psychology") == 0 && isYes(likeInternship)){

statementActive = 1;

if(debug) cout << "Rule 34 satisfied" << endl;

}

break;

case 34:

if(profession.compare("psychology") == 0 && isYes(groupWork) && isYes(likeSocialScience)){

statementActive = 1;

if(debug) cout << "Rule 34 satisfied" << endl;

}

break;

case 35:

if(profession.compare("psychology") == 0 && isYes(likeCommunication) && isYes(likeAnalyticalSkills)){

statementActive = 1;

if(debug) cout << "Rule 35 satisfied" << endl;

}

break;

case 36:

if(profession.compare("agriculture") == 0 && isYes(goodCredit) && isYes(likeManagement)){

statementActive = 1;

if(debug) cout << "Rule 36 satisfied" << endl;

}

break;

case 37:

if(profession.compare("agriculture") == 0 && isYes(groupWork) && isYes(likeManagement)){

statementActive = 1;

if(debug) cout << "Rule 37 satisfied" << endl;

}

break;

case 38:

if(profession.compare("agricuture") == 0 && isYes(likeManagement) && isYes(likeMarketing)){

statementActive = 1;

if(debug) cout << "Rule 38 satisfied" << endl;

}

break;

case 39:

if(profession.compare("agriculture") == 0 && isYes(likeChemistry)){

statementActive = 1;

if(debug) cout << "Rule 39 satisfied" << endl;

}

break;

case 40:

if(profession.compare("agriculture") == 0 && isYes(likeWaterResources)){

statementActive = 1;

if(debug) cout << "Rule 40 satisfied" << endl;

}

break;

case 41:

if(profession.compare("healthcare") == 0 && isYes(likeCommunication)){

statementActive = 1;

if(debug) cout << "Rule 41 satisfied" << endl;

}

break;

case 42:

if(profession.compare("healthcare") == 0 && isYes(likeManagement)){

statementActive = 1;

if(debug) cout << "Rule 42 satisfied" << endl;

}

break;

case 43:

if(profession.compare("healthcare") == 0 && isYes(likeBiology)){

statementActive = 1;

if(debug) cout << "Rule 43 satisfied" << endl;

}

break;

case 44:

if(profession.compare("healthcare") == 0 && isYes(likeManagement) && isYes(groupWork)){

statementActive = 1;

if(debug) cout << "Rule 44 satisfied" << endl;

}

break;

case 45:

if(profession.compare("healthcare") == 0 && isYes(likeAnatomy) && isYes(likeChemistry)){

statementActive = 1;

if(debug) cout << "Rule 45 satisfied" << endl;

}

break;

case 46:

if(profession.compare("education") == 0 && isYes(likeChildren)){

statementActive = 1;

if(debug) cout << "Rule 46 satisfied" << endl;

}

break;

case 47:

if(profession.compare("education") == 0 && isYes(likeReading) && isYes(likeCommunication)){

statementActive = 1;

if(debug) cout << "Rule 47 satisfied" << endl;

}

break;

case 48:

if(profession.compare("education") == 0 && isYes(likeMath) && isYes(likeChemistry)){

statementActive = 1;

if(debug) cout << "Rule 48 satisfied" << endl;

}

break;

case 49:

if(profession.compare("education") == 0 && isYes(likeCommunication) && isYes(likeChildren)){

statementActive = 1;

if(debug) cout << "Rule 49 satisfied" << endl;

}

break;

case 50:

if(profession.compare("education") == 0 && isYes(likeSocialScience) && isYes(likeCommunication)){

statementActive = 1;

if(debug) cout << "Rule 50 satisfied" << endl;

}

break;

case 51:

if(profession.compare("propertymanagement") == 0 && isYes(likeManagement)){

statementActive = 1;

if(debug) cout << "Rule 51 satisfied" << endl;

}

break;

case 52:

if(profession.compare("propertymanagement") == 0 && isYes(groupWork)){

statementActive = 1;

if(debug) cout << "Rule 53 satisfied" << endl;

}

break;

case 53:

if(profession.compare("propertymanagement") == 0 && isYes(reliable)){

statementActive = 1;

if(debug) cout << "Rule 53 satisfied" << endl;

}

break;

case 54:

if(profession.compare("propertymanagement") == 0 && isYes(likeRepairs)){

statementActive = 1;

if(debug) cout << "Rule 54 satisfied" << endl;

}

break;

case 55:

if(profession.compare("propertymanagement") == 0 && isYes(likeCommunication)){

statementActive = 1;

if(debug) cout << "Rule 55 satisfied" << endl;

}

break;

case 56:

if(profession.compare("cs") == 0 && isYes(likeSocialScience)){

statementActive = 1;

if(debug) cout << "Rule 56 satisfied" << endl;

}

break;

case 57:

if(profession.compare("cs") == 0 && isYes(likeMath) && isYes(likeAnalyticalSkills)){

statementActive = 1;

if(debug) cout << "Rule 58 satisfied" << endl;

}

break;

case 58:

if(profession.compare("cs") == 0 && isYes(groupWork)){

statementActive = 1;

if(debug) cout << "Rule 58 satisfied" << endl;

}

break;

case 59:

if(profession.compare("cs") == 0 && isYes(likeMarketing)){

statementActive = 1;

if(debug) cout << "Rule 59 satisfied" << endl;

}

break;

case 60:

if(profession.compare("cs") == 0 && isYes(likeOriginality)){

statementActive = 1;

if(debug) cout << "Rule 60 satisfied" << endl;

}

break;

default:

if (debug) cout << "bad value passed to ifconditions()" << endl;

}//end of switch

}//end of ifconditions()

void FC::Result()

{

if (debug) cout << "result called for SN: " << statementNumber << endl;

if (statementNumber > 0 && statementNumber < size)

done = true;

/\* invoke THEN part \*/

switch (statementNumber)

{

// /\*\*\*\*\*\*\*\*\*\*\* comment 1500 \*\*\*\*\*\*\*\*\*\*\*/

// /\* put variable on the conclusion variable queue \*/

// case 1:

// stock = "RISE";

// cout<<"ST=RISE"<<endl;

// varble = "ST";

// inst = true;

// break;

// /\*\*\*\*\*\*\*\*\*\*\* comment 1510 \*\*\*\*\*\*\*\*\*\*\*/

// /\* put variable on the conclusion variable queue \*/

case 1:

area.assign("electricalengineering");

if (debug) cout << "Area is now electrical engineering" << endl;

varble.assign("AREA");

inst = true;

break;

// case 2:

// stock = "FALL";

// cout<<"ST=FALL"<<endl;

// varble = "ST";

// inst = true;

// break;

// /\*\*\*\*\*\*\*\*\*\*\* comment 1540 \*\*\*\*\*\*\*\*\*\*\*/

// /\* put variable on the conclusion variable queue \*/

case 2:

area.assign("mechanicalengineering");

if (debug) cout << "Area is now mechanical engineering"<< endl;

inst = true;

break;

// case 3:

// interest = "RISE";

// cout<<"IN=RISE"<<endl;

// varble = "IN";

// inst = true;

// break;

// /\*\*\*\*\*\*\*\*\*\*\* comment 1550 \*\*\*\*\*\*\*\*\*\*\*/

// /\* put variable on the conclusion variable queue \*/

case 3:

area.assign("civilengineering");

if (debug) cout << "Area is now civil engineering"<< endl;

inst = true;

break;

// case 4:

// interest = "FALL";

// cout<<"IN=FALL"<<endl;

// varble = "IN";

// inst = true;

// break;

// /\* put variable on the conclusion variable queue \*/

case 4:

area.assign("petroleumengineering");

if (debug) cout << "Area is now petroleum engineering"<< endl;

inst = true;

break;

// case 5:

// interest = "FALL";

// cout<<"IN=FALL"<<endl;

// varble = "IN";

// inst = true;

// break;

case 5:

area.assign("chemicalengineering");

if (debug) cout << "Area is now chemical engineering"<< endl;

inst = true;

break;

case 6:

area.assign("pharmaceuticals");

if (debug) cout << "Area is now pharmaceuticals"<< endl;

inst = true;

break;

case 7:

area.assign("biology");

if (debug) cout << "Area is now biology"<< endl;

inst = true;

break;

case 8:

area.assign("astronomy");

if (debug) cout << "Area is now astronomy"<< endl;

inst = true;

break;

case 9:

area.assign("quantumphysics");

if (debug) cout << "Area is now quantum physics"<< endl;

inst = true;

break;

case 10:

area.assign("oceanography");

if (debug) cout << "Area is now oceanography"<< endl;

inst = true;

break;

case 11:

area.assign("businessowner");

if (debug) cout << "Area is now business owner"<< endl;

inst = true;

break;

case 12:

area.assign("manager");

if (debug) cout << "Area is now manager"<< endl;

inst = true;

break;

case 13:

area.assign("accounting");

if (debug) cout << "Area is now accounting"<< endl;

inst = true;

break;

case 14:

area.assign("humanresources");

if (debug) cout << "Area is now human resources"<< endl;

inst = true;

break;

case 15:

area.assign("ceo");

if (debug) cout << "Area is now ceo"<< endl;

inst = true;

break;

case 16:

area.assign("mentalhealth");

if (debug) cout << "Area is now mental health"<< endl;

inst = true;

break;

case 17:

area.assign("oncology");

if (debug) cout << "Area is now oncology"<< endl;

inst = true;

break;

case 18:

area.assign("cardiology");

if (debug) cout << "Area is now cardiology"<< endl;

inst = true;

break;

case 19:

area.assign("orthopedics");

if (debug) cout << "Area is now orthopedics"<< endl;

inst = true;

break;

case 20:

area.assign("generalpractitioner");

if (debug) cout << "Area is now general practitioner"<< endl;

inst = true;

break;

case 21:

area.assign("author");

if (debug) cout << "Area is now author"<< endl;

inst = true;

break;

case 22:

area.assign("journalism");

if (debug) cout << "Area is now journalism"<< endl;

inst = true;

break;

case 23:

area.assign("editing");

if (debug) cout << "Area is now editing"<< endl;

inst = true;

break;

case 24:

area.assign("copywriting");

if (debug) cout << "Area is now copywriting"<< endl;

inst = true;

break;

case 25:

area.assign("academia");

if (debug) cout << "Area is now academia"<< endl;

inst = true;

break;

case 26:

area.assign("preciousmetals");

if (debug) cout << "Area is now precious metals"<< endl;

inst = true;

break;

case 27:

area.assign("cartography");

if (debug) cout << "Area is now cartography"<< endl;

inst = true;

break;

case 28:

area.assign("climatology");

if (debug) cout << "Area is now climatology"<< endl;

inst = true;

break;

case 29:

area.assign("environmentalmanagement");

if (debug) cout << "Area is now enironmental management"<< endl;

inst = true;

break;

case 30:

area.assign("geomatics");

if (debug) cout << "Area is now geomatics"<< endl;

inst = true;

break;

case 31:

area.assign("mentaltherapy");

if (debug) cout << "Area is now mental therapy"<< endl;

inst = true;

break;

case 32:

area.assign("psychologicalresearch");

if (debug) cout << "Area is now psychological research"<< endl;

inst = true;

break;

case 33:

area.assign("industrialpsychology");

if (debug) cout << "Area is now industrial psychology"<< endl;

inst = true;

break;

case 34:

area.assign("accesibility");

if (debug) cout << "Area is now accesibility"<< endl;

inst = true;

break;

case 35:

area.assign("clinicalpsychology");

if (debug) cout << "Area is now clinical psychology"<< endl;

inst = true;

break;

case 36:

area.assign("farmownership");

if (debug) cout << "Area is now farm ownership"<< endl;

inst = true;

break;

case 37:

area.assign("ranchwork");

if (debug) cout << "Area is now ranch work"<< endl;

inst = true;

break;

case 38:

area.assign("viniculture");

if (debug) cout << "Area is now viniculture"<< endl;

inst = true;

break;

case 39:

area.assign("cropspraying");

if (debug) cout << "Area is now crop spraying"<< endl;

inst = true;

break;

case 40:

area.assign("irrigation");

if (debug) cout << "Area is now irrigation"<< endl;

inst = true;

break;

case 41:

area.assign("healthservicesmanagement");

if (debug) cout << "Area is now health services management"<< endl;

inst = true;

break;

case 42:

area.assign("healthcareadministration");

if (debug) cout << "Area is now health care administration"<< endl;

inst = true;

break;

case 43:

area.assign("physiciansassistant");

if (debug) cout << "Area is now physicians assistant"<< endl;

inst = true;

break;

case 44:

area.assign("publichealth");

if (debug) cout << "Area is now public health"<< endl;

inst = true;

break;

case 45:

area.assign("nursing");

if (debug) cout << "Area is now nursing"<< endl;

inst = true;

break;

case 46:

area.assign("earlychildhood");

if (debug) cout << "Area is now early childhood"<< endl;

inst = true;

break;

case 47:

area.assign("kthrough12");

if (debug) cout << "Area is now k-12"<< endl;

inst = true;

break;

case 48:

area.assign("highereducation");

if (debug) cout << "Area is now higher education"<< endl;

inst = true;

break;

case 49:

area.assign("teachersaid");

if (debug) cout << "Area is now teachers aid"<< endl;

inst = true;

break;

case 50:

area.assign("counselor");

if (debug) cout << "Area is now counselor"<< endl;

inst = true;

break;

case 51:

area.assign("propertymanager");

if (debug) cout << "Area is now property manager"<< endl;

inst = true;

break;

case 52:

area.assign("realty");

if (debug) cout << "Area is now realty"<< endl;

inst = true;

break;

case 53:

area.assign("cleaning");

if (debug) cout << "Area is now cleaning"<< endl;

inst = true;

break;

case 54:

area.assign("maintenancework");

if (debug) cout << "Area is now maintenance work"<< endl;

inst = true;

break;

case 55:

area.assign("painting");

if (debug) cout << "Area is now painting"<< endl;

inst = true;

break;

case 56:

area.assign("ai");

if (debug) cout << "Area is now ai"<< endl;

inst = true;

break;

case 57:

area.assign("machinelearning");

if (debug) cout << "Area is now machine learning"<< endl;

inst = true;

break;

case 58:

area.assign("softwareengineering");

if (debug) cout << "Area is now software engineering"<< endl;

inst = true;

break;

case 59:

area.assign("datascience");

if (debug) cout << "Area is now data science"<< endl;

inst = true;

break;

case 60:

area.assign("languages");

if (debug) cout << "Area is now languages" << endl;

inst = true;

break;

default:

if(debug) cout << "Bad value passed to result()" << endl;

/\*\*\*\*\*\*\*\*\*\*\* comment 1610 \*\*\*\*\*\*\*\*\*\*\*/

} //end of switch

} // end of result func

void FC::takingInput(int i)

{

if (debug) cout << "takingInput called for variable: " << i << endl;

switch (i)

{

/\* input statements for sample position knowledge base \*/

case 1:

// cout<<"RISE OR FALL FOR IN? ";

// cin>>interest;

// break;

cout << "Do you enjoy working with physics: ";

cin >> likePhysics;

cout << endl;

break;

case 2:

// cout<<"RISE OR FALL FOR DO? ";

// cin>>dollar;

// break;

cout << "Do you enjoy doing math: ";

cin >> likeMath;

cout << endl;

break;

case 3:

// cout<<"RISE OR FALL FOR FT? ";

// cin>>fedint;

// break;

cout << "Do you work well in groups: ";

cin >> groupWork;

cout << endl;

break;

case 4:

// cout<<"ADD OR SUBTRACT FOR FM? ";

// cin>>fedmon;

// break;

cout << "Do you enjoy studying biology: ";

cin >> likeBiology;

cout << endl;

break;

case 5:

cout << "Do you enjoy studying chemistry: ";

cin >> likeChemistry;

cout << endl;

break;

case 6:

cout << "Do you like marketing goods and services: ";

cin >> likeMarketing;

cout << endl;

break;

case 7:

cout << "Do you enjoy managing employees: ";

cin >> likeManagement;

cout << endl;

break;

case 8:

cout << "Do you maintain a good credit score: ";

cin >> goodCredit;

cout << endl;

break;

case 9:

cout << "Are you interested in medical ethics: ";

cin >> likeMedicalEthics;

cout << endl;

break;

case 10:

cout << "Will you get a state license to practice your job: ";

cin >> stateLicensure;

cout << endl;

break;

case 11:

cout << "Do you like writing: ";

cin >> likeWriting;

cout << endl;

break;

case 12:

cout << "Do you enjoy practicing creativity and originality: ";

cin >> likeOriginality;

cout << endl;

break;

case 13:

cout << "Do you follow the media closely: ";

cin >> likeMedia;

cout << endl;

break;

case 14:

cout << "Have you studied social sciences: ";

cin >> likeSocialScience;

cout << endl;

break;

case 15:

cout << "Do you enjoy studying grammar and syntax: ";

cin >> likeEnglish;

cout << endl;

break;

case 16:

cout << "Do you read much: ";

cin >> likeReading;

cout << endl;

break;

case 17:

cout << "Are you interested in studying the climate: ";

cin >> likeClimate;

cout << endl;

break;

case 18:

cout << "Would you want to work with minerals: ";

cin >> likeStones;

cout << endl;

break;

case 19:

cout << "Would you enjoy doing land surveys: ";

cin >> likeLand;

cout << endl;

break;

case 20:

cout << "Would you enjoy taking soil samples: ";

cin >> likeSoil;

cout << endl;

break;

case 21:

cout << "Do you have an interest in maintaining water resources: ";

cin >> likeWaterResources;

cout << endl;

break;

case 22:

cout << "Are you interested in data analysis: ";

cin >> likeAnalyticalSkills;

cout << endl;

break;

case 23:

cout << "Are you interested in doing an internship: ";

cin >> likeInternship;

cout << endl;

break;

case 24:

cout << "Would you like to study human anatomy: ";

cin >> likeAnatomy;

cout << endl;

break;

case 25:

cout << "Do you enjoy working with children: ";

cin >> likeChildren;

cout << endl;

break;

case 26:

cout << "Are you a reliable person: ";

cin >> reliable;

cout << endl;

break;

case 27:

cout << "Do you enjoy doing repairs: ";

cin >> likeRepairs;

cout << endl;

break;

case 28:

cout << "What have you chosen as your profession: ";

cin >> profession;

cout << endl;

break;

case 29:

cout << "What area are you interested in going into: ";

cin >> area;

cout << endl;

break;

case 30:

cout << "Would you like to study the law: ";

cin >> likeLaw;

cout << endl;

break;

case 31:

cout << "Are you interested in practicing human communication: ";

cin >> likeCommunication;

cout << endl;

break;

default:

if (debug) cout << "Bad value passed to takingput()" << endl;

} //end of switch

} //end of takinginput()

bool FC::isYes(string inString){

if (inString.compare("yes") == 0 || inString.compare("y") == 0)

return true;

else

return false;

}

void FC::printProfession(){

if(profession.compare("science") == 0)

cout << "Science" << endl;

else if (profession.compare("engineering")==0)

cout << "Engineering" << endl;

else if (profession.compare("business")==0)

cout << "Business" << endl;

else if (profession.compare("medical")==0)

cout << "Medical" << endl;

else if (profession.compare("english")==0)

cout << "English" << endl;

else if (profession.compare("geography")==0)

cout << "Geography" << endl;

else if (profession.compare("psychology")==0)

cout << "Psychology" << endl;

else if (profession.compare("healthcare")==0)

cout << "Health Care" << endl;

else if (profession.compare("agriculture")==0)

cout << "Agriculture" << endl;

else if (profession.compare("education")==0)

cout << "Education" << endl;

else if (profession.compare("propertymanagement")==0)

cout << "Property Management" << endl;

else if (profession.compare("cs")==0)

cout << "Computer Science" << endl;

else

cout << "No Profession" << endl;

return;

}

void FC::printArea(){

if(area.compare("electricalengineering") == 0)

cout << "Electrical Engineering" << endl;

else if (area.compare("mechanicalengineering")==0)

cout << "Mechanical Engineering" << endl;

else if (area.compare("civilengineering")==0)

cout << "Civil Engineering" << endl;

else if (area.compare("petroleumengineering")==0)

cout << "Petroleum Engineering" << endl;

else if (area.compare("chemicalengineering")==0)

cout << "Chemical Engineering" << endl;

else if (area.compare("pharmaceuticals")==0)

cout << "Pharmaceuticals" << endl;

else if (area.compare("biology")==0)

cout << "Biology" << endl;

else if (area.compare("astronomy")==0)

cout << "Astronomy" << endl;

else if (area.compare("quantumphysics")==0)

cout << "Quantam Physics" << endl;

else if (area.compare("oceanography")==0)

cout << "Oceanography" << endl;

else if (area.compare("businessowner")==0)

cout << "Owning a Small Business" << endl;

else if (area.compare("manager")==0)

cout << "Management" << endl;

else if (area.compare("accounting")==0)

cout << "Accounting" << endl;

else if (area.compare("humanresources")==0)

cout << "Human Resources" << endl;

else if (area.compare("ceo")==0)

cout << "finding a position as a CEO" << endl;

else if (area.compare("mentalhealth")==0)

cout << "Mental Health" << endl;

else if (area.compare("oncology")==0)

cout << "Oncology" << endl;

else if (area.compare("cardiology")==0)

cout << "Cardiology" << endl;

else if (area.compare("orthopedics")==0)

cout << "Orthopedics" << endl;

else if (area.compare("generalpractitioner")==0)

cout << "General Practice" << endl;

else if (area.compare("author")==0)

cout << "Authoring your own Novel" << endl;

else if (area.compare("journalism")==0)

cout << "Journalism" << endl;

else if (area.compare("editing")==0)

cout << "Editing" << endl;

else if (area.compare("copywriting")==0)

cout << "Copy Writing" << endl;

else if (area.compare("academia")==0)

cout << "Academia" << endl;

else if (area.compare("preciousmetals")==0)

cout << "prospecting for Precious Metals" << endl;

else if (area.compare("cartography")==0)

cout << "Cartography" << endl;

else if (area.compare("climatology")==0)

cout << "Climate Studies" << endl;

else if (area.compare("environmentalmanagement")==0)

cout << "Environmental Management" << endl;

else if (area.compare("geomatics")==0)

cout << "Geomatics" << endl;

else if (area.compare("mentaltherapy")==0)

cout << "Mental Therapy" << endl;

else if (area.compare("psychologicalresearch")==0)

cout << "Psychological Research" << endl;

else if (area.compare("industrialpsychology")==0)

cout << "Industrial Psychology" << endl;

else if (area.compare("accesibility")==0)

cout << "Accesibility" << endl;

else if (area.compare("clinicalpsychology")==0)

cout << "Clinical Psychology" << endl;

else if (area.compare("farmownership")==0)

cout << "Farm Ownership" << endl;

else if (area.compare("ranchwork")==0)

cout << "Working on a Ranch" << endl;

else if (area.compare("viniculture")==0)

cout << "Viniculture" << endl;

else if (area.compare("cropspraying")==0)

cout << "Crop Spraying" << endl;

else if (area.compare("irrigation")==0)

cout << "Irrigation" << endl;

else if (area.compare("healthservicesmanagement")==0)

cout << "Health Services Management" << endl;

else if (area.compare("healthcareadministration")==0)

cout << "Healthcare Administration" << endl;

else if (area.compare("physiciansassistant")==0)

cout << "a position as a Physician's Assistant" << endl;

else if (area.compare("publichealth")==0)

cout << "Public Health" << endl;

else if (area.compare("nursing")==0)

cout << "Nursing" << endl;

else if (area.compare("earlychildhood")==0)

cout << "Early Childhood Education" << endl;

else if (area.compare("kthrough12")==0)

cout << "K Through 12" << endl;

else if (area.compare("highereducation")==0)

cout << "Higher Education" << endl;

else if (area.compare("teachersaid")==0)

cout << "a position as a Teacher's Aid" << endl;

else if (area.compare("counselor")==0)

cout << "Counseling Students" << endl;

else if (area.compare("propertymanager")==0)

cout << "Managing Properties" << endl;

else if (area.compare("realty")==0)

cout << "Realty" << endl;

else if (area.compare("cleaning")==0)

cout << "Cleaning" << endl;

else if (area.compare("maintenancework")==0)

cout << "Maintenance Work" << endl;

else if (area.compare("painting")==0)

cout << "Painting" << endl;

else if (area.compare("ai")==0)

cout << "Artificial Intelligence" << endl;

else if (area.compare("machinelearning")==0)

cout << "Machine Learning" << endl;

else if (area.compare("softwareengineering")==0)

cout << "Software Engineering" << endl;

else if (area.compare("datascience")==0)

cout << "Data Science" << endl;

else if (area.compare("languages")==0)

cout << "Languages" << endl;

else

cout << area << endl;

return;

}

**7.5 Main.cpp**

int main () {

cout << "This recomender will help you choose a profession and an area..." << endl;

BC backChainer(false);

backChainer.inferenceSection();

FC forwardChainer (backChainer.getProfession(), false);

forwardChainer.start();

cout << "Congratulation! Your future profession will be ";

forwardChainer.printProfession();

cout << "With a focus in ";

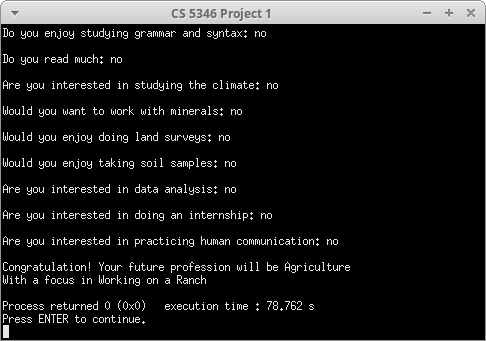
forwardChainer.printArea();

return 0;

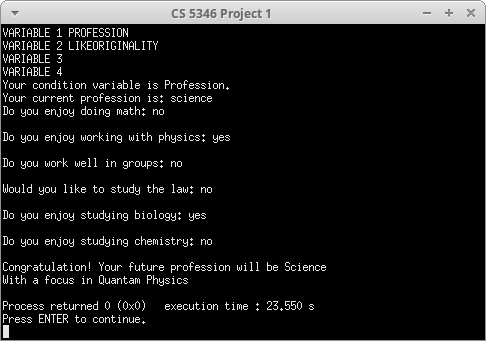
}

**8. Sample Output:**

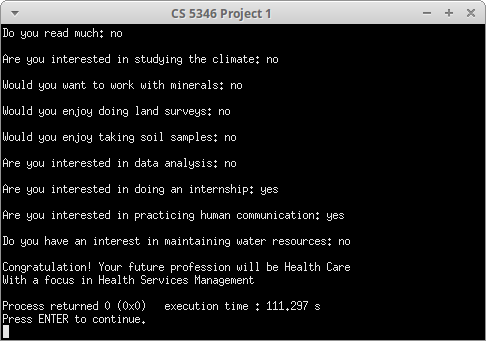
**Run 1:**

****

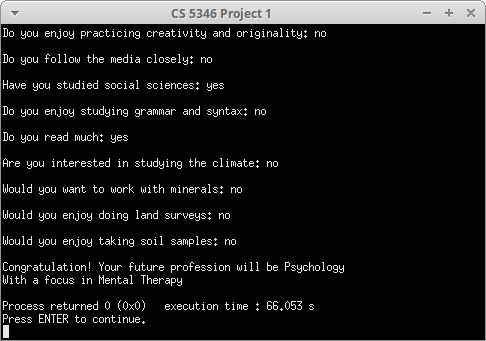
**Run 2:**

****

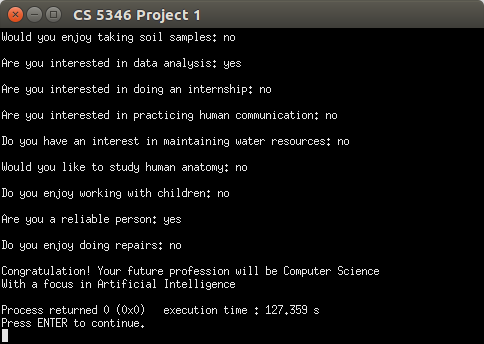
**Run 3:**

****

**Run 4:**

****

**Run 5:**



**9 Analysis of Results:**

After extensive testing the newly produced code has been found to reliably and consistently generate suggested Professions and Areas for a wide variety of students with noticeable lag or delay. The results which are generated by the code are consistent with the research that was performed prior to the development of the knowledge base.

It is our belief that this system will be effective in guiding young students towards the Profession and Area which is most in line with their desires and the intentions of the degree programs that they are studying. Furthermore the knowledge base has been developed in such away as to guide students into those careers which are most demanded by industry first. In this way the system ensures the greatest consistency between the students it is guiding and the society which it is guiding them towards.

The testing of the code generated by this project has demonstrated that consistent input will always generate a consistent result and the code is therefore free from randomness and inconsistency. Furthermore the code was developed in accordance with “best practices” as currently accepted by the academic community and can therefore be expected to be secure against error and common malicious activities.

The average execution time of one run of the program is 10 to 12 seconds with the vast majority of that time being spent waiting for responses from the users. This is more than fast enough time for the user to have a smooth and fulfilling experience. Furthermore, this execution is lightweight enough for many instances of this program to run on a single system without overly taxing the resources of a reasonably powerful system. This suggests that this program could be easily adapted to a web interface of a client-server architecture to better enable universities to serve a large student body with finite resources.

Future iterations of this project would shift the code base farther towards pure object orientation. We feel that this would allow for greater expandability of the project and allow for more efficient and dynamic allocation of memory. Building the rules themselves as objects rather than statically coding them into the forward and backward chaining objects would be a fuller realization of the original intent of the project.

**10 Comparison of Systems:**

**10.1 Comparison to Old System**

Compared to the previous code base, this project has added substantial usability, security, and maintainability. These advantages are largely accounted for by three factors. First, the adaptation of object orientation and modern coding standards. Second, the re-construction of the user interface to produce interactions that are more natural for the user. And third, the body of research that went into the construction of the knowledge base.

The prior system included a user interface which was counter-intuitive and difficult to follow. This project revised the set of questions and responses employed by the system. The new questions and responses not only reflect the addition of a new knowledge base but are also crafted to be closer to natural language and to be easier to follow for novice users. A degree of flexibility was also built into the question and answer sections to allow the users to answer the system in a more natural tone. We feel that this will make the system more easily accessible by a lay user.

The prior system relied on several techniques which are now considered to be incompatible with best practices. Specifically the use of global variables and GOTO statements created notable security risks by unnecessarily exposing values outside of the scope of their use. This can lead to unintended side affects which can cascade through the execution of the program and create unwanted and unpredictable results.

These outdated coding practices also created a program that would be very difficult to maintain by a team over any period of time. The short variable names, sparseness of comments, and “spaghetti” style coding was extremely difficult to follow and even harder to alter. Furthermore the marketability of the code was diminished by exposing all of the source rather than separating the source into a file separate from a header file which can be more easily read. The new system has closed all of these security gaps, divided the system into functions which are intuitive and easy to follow, and divided the code into libraries so that they can be distributed without exposing the source code as developed by this team.

**10.2 Comparison of Backward and Forward Chaining**

This system incorporates two varieties of inference engine. Forward and Backward Chaining are very similar but are different enough to warrant analysis. As explained earlier Backward Chaining focus on the conclusions to derive conclusions and Forward Chaining focuses and variables to derive conclusions.

From a user perspective the Backward Chaining system developed to recommend the Profession of the User is much more terse and reaches it’s conclusions using a smaller set of questions. This is partly because the total set of Professions is much smaller than the total set of Areas. This creates the impression that the Forward Chaining algorithm is better informed.

This team also found Forward Chaining to be the more robust algorithm. It was implemented more quickly and appears to be more fault tolerant. However, it does ask a greater number of questions so it produces a user experience which flows less naturally. Future implementations of this project will “short circuit” some answers so that fewer will be asked to reach the same conclusion.