Project 1: Expert Intelligent System for Cancer Diagnosis and Treatment

CS 4346: Artificial Intelligence with Dr. Moonis Ali

Group: Robert Jones, Brittany Hale, John Courtright

I. Problem Description

a. Explain Problem

Cancer is one of the leading causes of death worldwide. A timely, accurate, diagnosis is essential for successful treatment. Medical professionals usually rely on years of experience and a large variety of questions to diagnose different kinds of cancer. Different experiences, skill levels, and access to data has made diagnosis inconsistent and at times unreliable. This project proposes the development of an expert system that can assist in limiting these inconsistencies by giving access to the same knowledgebase for all users.

b. Domain Description

Though this is not a complete answer to cancer diagnostics, this project shows what an expert system can do when given a detailed knowledge base. The research that went into making the rules and knowledge base was conducted by 3 Computer Science students so the knowledge does not expand as far as it could if it were conducted by doctors with more experience. Nonetheless, the structure and current rule list can be easily modified or added to theoretically creating a knowledge base always expanding as all doctor's knowledge advances.

II. Methodology

a. Forward Chain

Forward chaining is a fact driven algorithm. Unlike backward chaining, we are given a starting spot then use questions to find our path to a conclusion. In our case we are given the diagnosis from our backward chaining algorithm and use it to identify our clause number if there is one. By converting the clause number to rule number we identify the paths of questions we will ask to accurately prescribe the patient with a treatment.

Clause Variable List (`clauseVL`): Stores predefined conditions in a structured 2D array.

Variable List (`variableList`): Keeps track of individual conditions and their user-input values.

Rule List (`ruleList`): Contains rules mapping conditions to recommended treatments.

Search and Update Functions: `search_cvl`, `clause_to_rule`, and `update_VL` work together to ask users questions and infer missing variables.

Algorithm

Initialization

At the start, we initialize the **Clause Variable List (clauseVL)**, **Variable List (variableList)**, and **Rule List (ruleList)**. We also set the **globalConclusionsCounter** to 0.

Processing Patient Diagnosis

The process method sets the initial diagnosis in variableList and globalConclusions. It then constructs a search criteria from the diagnosis given by the backward chain, and calls search_cvl to begin the reasoning process.

Searching the Clause Variable List

The search_cvl method iterates through clauseVL to find matching variables based on the search criteria. For each match found, it:

- Calls update_VL to update variableList accordingly.
- Calls clause_to_rule to map the clause to a rule.

Updating the Variable List

The update_VL method updates variableList based on user input for each variable found in clauseVL. It also updates globalConclusions and increments globalConclusionsCounter.

Mapping Clauses to Rules

The clause_to_rule method determines the rule index (RI) from the clause number and then calls validate_Ri to check if the rule conditions are met.

Validating Rule Index

The validate_Ri method verifies whether all conditions in a rule are satisfied by checking against variableList. If the rule is validated, it:

- Updates globalConclusions.
- Calls modifyTreatmentString.
- Increments globalConclusionsCounter.
- Calls printGlobalConclusions to display the results.

Modifying Treatment String

The modifyTreatmentString method takes the treatment from the most recent conclusion stored in globalConclusions and updates the treatment variable accordingly.

Printing Global Conclusions

The printGlobalConclusions method outputs all stored conclusions and then resets globalConclusions for the next cycle.

Retrieving the Final Treatment

The getTreatment method returns the final treatment string based on the conclusions reached during the process.

b. Backward Chain

Backward chaining is a goal-oriented reasoning method commonly used in expert systems and rule-based AI. Rather than starting with known facts and working forward, backward chaining begins with a goal and traces backward to determine if supporting evidence exists. In this program, the goal is drawn from a predefined conclusion list, which in our case consists of different cancer types the system can diagnose.

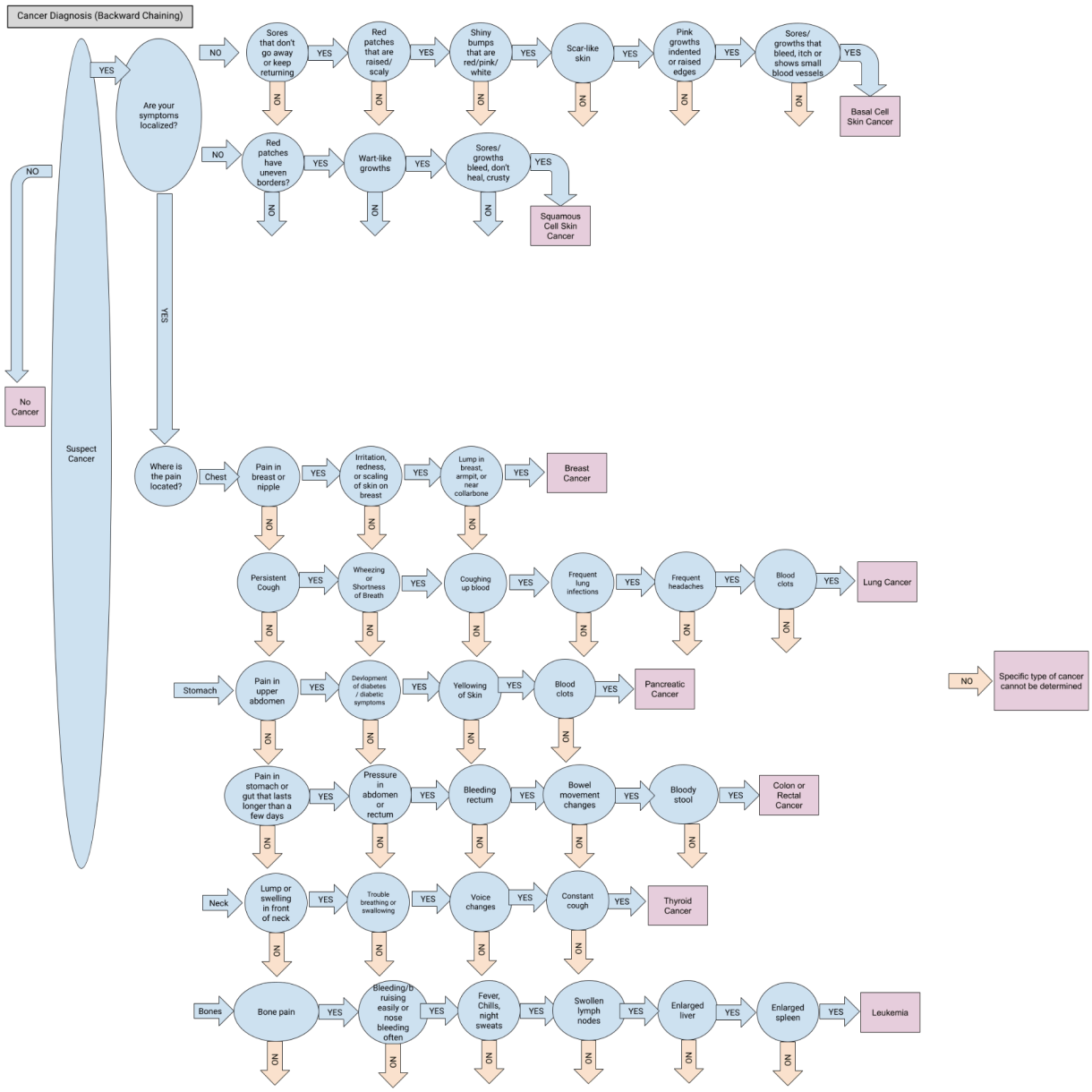
Backward Chaining Algorithm

1. The user selects a conclusion from the available conclusion list.
2. The system searches for the selected conclusion within the Conclusion List:
    - If the conclusion is not found, the user is notified and asked to select another.
    - If the conclusion is found, the corresponding rule number is pushed onto the Conclusion Stack along with the clause number.
3. The Clause Variable List is checked for IF conditions associated with the selected conclusion.
4. If any IF condition is itself a conclusion, it is added to the stack, and its conditions are evaluated recursively.
5. If any required variables have not been assigned values, the system prompts the user for input.

6. If the top statement on the stack has been assigned a value but does not meet the condition, it is removed, and the system looks for another matching conclusion. The process then returns to step 4.
7. If the top statement is satisfied, it is removed from the stack.
8. If additional conclusions remain in the stack, the clause number is incremented, and the process loops back to step 4.
9. If the stack is empty, the selected conclusion is confirmed as valid.

III. Rule Base

a. Backward Chain Decision Tree


Cancer Diagnosis (Backward Chaining)
Are your symptoms localized?
YES
NO
No Cancer
Suspect Cancer
Sores that don't go away or keep returning
Red patches that are raised/ scaly
Shiny bumps that are red/pink/ white
Scar-like skin
Pink growths indented or raised edges
Sores/ growths that bleed, itch or shows small blood vessels
Basal Cell Skin Cancer
Red patches have uneven borders?
Wart-like growths
Sores/ growths bleed, don't heal, crusty
Squamous Cell Skin Cancer
Where is the pain located?
Chest
Pain in breast or nipple
Irritation, redness, or scaling of skin on breast
Lump in breast, armpit, or near collarbone
Breast Cancer
Persistent Cough
Wheezing or Shortness of Breath
Coughing up blood
Frequent lung infections
Frequent headaches
Blood clots
Lung Cancer
Stomach
Pain in upper abdomen
Development of diabetes / diabetic symptoms
Yellowing of Skin
Blood clots
Pancreatic Cancer
Pain in stomach or gut that lasts longer than a few days
Pressure in abdomen or rectum
Bleeding rectum
Bowel movement changes
Bloody stool
Colon or Rectal Cancer
Neck
Lump or swelling in front of neck
Trouble breathing or swallowing
Voice changes
Constant cough
Thyroid Cancer
Bones
Bone pain
Bleeding/b ruising easily or nose bleeding often
Fever, Chills, night sweats
Swollen lymph nodes
Enlarged liver
Enlarged spleen
Leukemia
NO
Specific type of cancer cannot be determined

b. Backward Chain Rules

Rule **10**. If Suspect Cancer = NO, then Cancer = NO
Rule **20**. If Symptoms localized = YES, then Localized Cancers = YES
Rule **30**. If Symptoms localized = NO, then Skin Cancers = YES
Rule **40**. If Skin Cancers = YES
AND Sores = YES
AND Red Patches = YES
AND Warts = NO
AND New Scars = YES,
Then Basal Cell Skin Cancer = YES

Rule **50**. If Skin Cancers = YES
AND Sores = YES
AND Red Patches = YES
AND Warts = YES,
Then Squamous Cell Skin Cancer = YES

Rule **60**. If Localized Cancers = YES, then CHECK PAIN

Rule **70**. If Pain = Chest
AND Pain in Breast or Nipple = YES
AND Red or Irritated or Scaly skin on Breast = YES
AND Lumps in Breast or Armpit or Collarbone = YES,
then Breast Cancer = YES

Rule **80**. If Pain = Chest
AND Cough or Breathing Issues = YES
AND Bloody Cough = YES
AND Lung Infections = YES
AND Blood Clots = YES
AND Frequent Headaches = YES,
then Lung Cancer = YES

Rule **90**. If Pain = Stomach
AND Diabetes = YES
AND Yellow Skin = YES
AND Blood Clots = YES,
then Pancreatic Cancer = YES

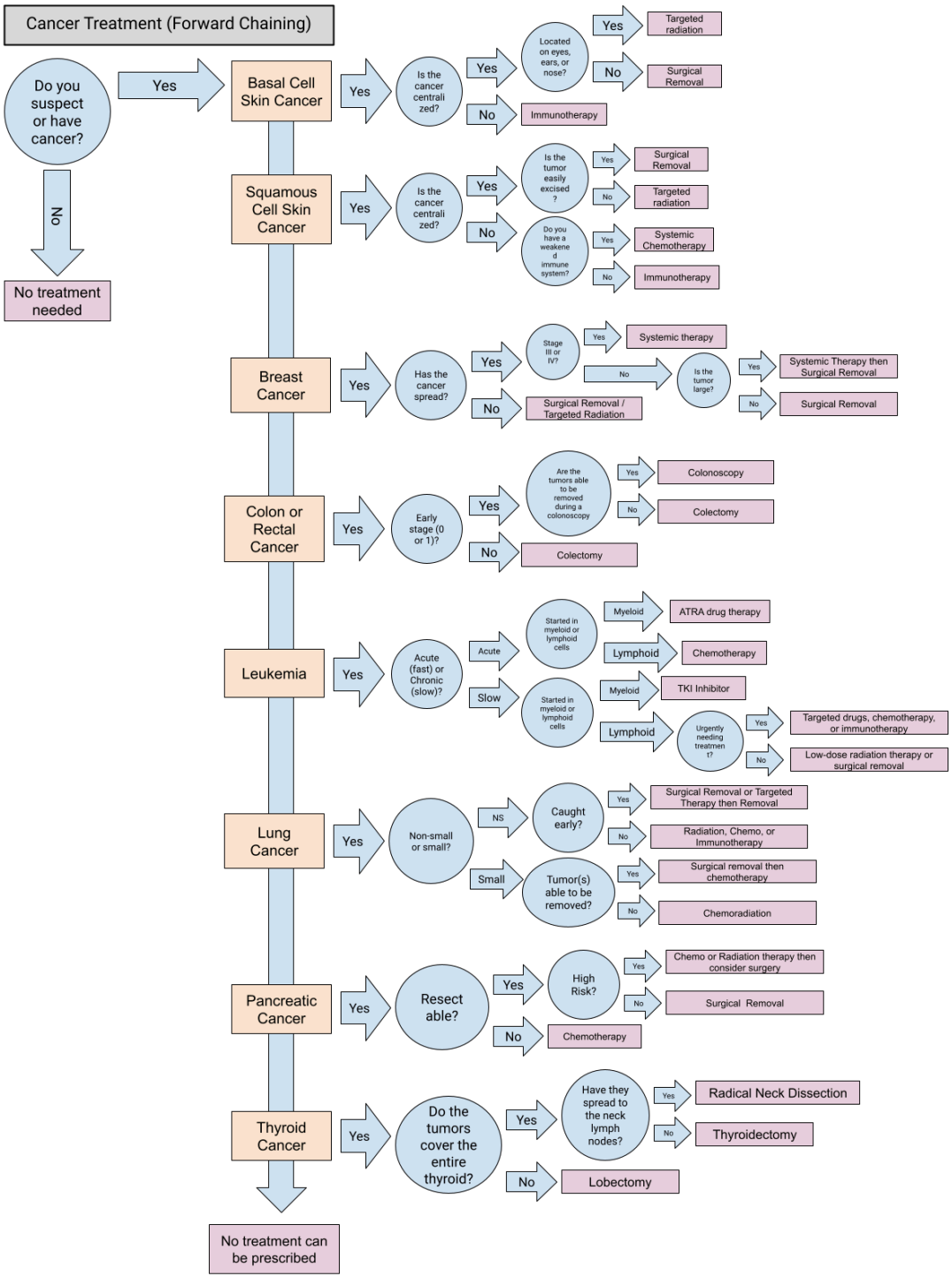
Rule **100**. If Pain = Stomach

AND Pressure in Abdomen = YES
AND Bowel Changes = YES
AND Blood from Rectum = YES,
then Colon or Rectal Cancer = YES

Rule **110**. If Pain = Neck
AND Neck Swelling or Lump = YES
AND Breathing or Swallowing Issues = YES
AND Voice Changes = YES
AND Constant Cough = YES,
then Thyroid Cancer = YES

Rule **120**. If Pain = Bones
AND Bleeding or Bruising Easily = YES
AND Fever or Chills or Night Sweats = YES
AND Enlarged Liver or Spleen = YES
AND Swollen Lymph Nodes = YES,
then Leukemia = YES

c. Forward Chain Decision Tree



d. Forward Chain Rules

Rule 10. If Cancer = NO, then Treatment = None

Rule 20. If Basal = YES AND
Centralized = YES AND
Excisable = YES
Then Treatment = Targeted Radiation

Rule 30. If Basal = YES AND
Centralized = YES AND
Excisable = NO
Then Treatment = Surgical Removal

Rule 40. If Basal = YES AND
Centralized = NO
Then Treatment = Immunotherapy

Rule 50. If Squamous = YES AND
Centralized = YES AND
Excisable = YES
Then Treatment = Surgical Removal

Rule 60. If Squamous = YES AND
Centralized = YES AND
Excisable = NO
Then Treatment = Targeted Radiation

Rule 70. If Squamous = YES AND
Centralized = NO AND
Weak = YES
Then Treatment = Chemotherapy

Rule 80. If Squamous = YES AND
Centralized = NO AND
Weak = NO
Then Treatment = Immunotherapy

Rule 90. If Breast = YES AND
Centralized = NO AND
Late = YES

Then Treatment = Systemic Therapy

Rule 100. If Breast = YES AND
Centralized = NO AND
Late = NO AND
Large = YES
Then Treatment = Systemic Therapy then Surgical Removal

Rule 110. If Breast = YES AND
Centralized = NO AND
Late = NO AND
Large = NO
Then Treatment = Surgical Removal

Rule 120. If Breast = YES AND
Centralized = YES AND
Excisable = YES
Then Treatment = Surgical Removal

Rule 130. If Breast = YES AND
Centralized = YES AND
Excisable = NO
Then Treatment = Targeted Radiation

Rule 140. If Colon = YES AND
Early = YES AND
Excisable = YES
Then Treatment = Colonoscopy

Rule 150. If Colon = YES AND
Early = YES AND
Excisable = NO
Then Treatment = Colectomy

Rule 160. If Colon = YES AND
Early = NO
Then Treatment = Colectomy

Rule 170. If Leukemia = YES AND

Acute = YES AND
Myeloid = YES
Then Treatment = ATRA Drug Therapy

Rule 180. If Leukemia = YES AND
Acute = YES AND
Lymphoid = YES
Then Treatment = Chemotherapy

Rule 190. If Leukemia = YES AND
Chronic = YES AND
Myeloid = YES
Then Treatment = TKI Inhibitor

Rule 200. If Leukemia = YES AND
Chronic = YES AND
Lymphoid = YES AND
Urgent = YES
Then Treatment = Targeted Drugs, Chemotherapy, or Immunotherapy

Rule 210. If Leukemia = YES AND
Chronic = YES AND
Lymphoid = NO AND
Urgent = NO
Then Treatment = Low-Dose Radiation or Surgical Removal

Rule 220. If Lung = YES AND
Small = NO AND
Early = YES
Then Treatment = Surgical Removal or Targeted Therapy then Removal

Rule 230. If Lung = YES AND
Small = NO AND
Early = NO
Then Treatment = Radiation, Chemo, or Immunotherapy

Rule 240. If Lung = YES AND
Small = YES AND
Excisable = YES

Then Treatment = Surgical Removal then Chemotherapy

Rule 250. If Lung = YES AND
Small = YES AND
Excisable = NO
Then Treatment = Chemoradiation

Rule 260. If Pancreatic = YES AND
Excisable = YES AND
Risk = YES
Then Treatment = Chemo or Radiation Therapy then consider Surgery

Rule 270. If Pancreatic = YES AND
Excisable = YES AND
Risk = NO
Then Treatment = Surgical Removal

Rule 280. If Pancreatic = YES AND
Excisable = NO
Then Treatment = Chemotherapy

Rule 290. If Thyroid = YES AND
Covered = YES AND
Centralized = NO
Then Treatment = Radical Neck Dissection

Rule 300. If Thyroid = YES AND
Covered = YES AND
Centralized = YES
Then Treatment = Thyroidectomy

Rule 310. If Thyroid = YES AND
Covered = YES
Then Treatment = Lobectomy

Rule 320. If Basal = NO AND
Squamous = NO AND
Breast = NO AND
Colon = NO AND

Leukemia = NO AND
Lung = NO AND
Pancreatic = NO AND
Thyroid = NO
Then Treatment = None

IV. Program Implementation

a. Implementation Style (class-based, .h structure)

For this program we decided it would best to split Dr.Allis program into header files. As for implementation of the given algorithms, we each decided an object-oriented design wold allow us the most flexibility and still strength. We each, showcased a vast implementation of C++ knowledge with the freedom given in the Project.

b. Source Code

main.cpp

```
// Title Comments smiley face
#include "backwardChain.h"
#include "forwardChain.h"
#include <iostream>
using namespace std;
int main()
{
   cout << "This program will help diagnose cancer and provide a treatment plan." << endl;
   // BACKWARD CHAINING
   BackwardChain backwardChaining; // CONSTRUCTOR
   // Start Backward Chain process & bring its diagnosis result to main
   string diagnosis = backwardChaining.startBackwardChain();
   // Prepare Diagnosis string for Forward Chaining treatment plan
   if (diagnosis == "Basal Skin Cancer")
   {
      diagnosis = "Basal";
   }
   if (diagnosis == "Squamous Skin Cancer")
   {
      diagnosis = "Squamous";
   }
   if (diagnosis == "Breast Cancer")
   {
      diagnosis = "Breast";
   }
```

```
    if (diagnosis == "Lung Cancer")
    {
        diagnosis = "Lung";
    }
    if (diagnosis == "Pancreatic Cancer")
    {
        diagnosis = "Pancreatic";
    }
    if (diagnosis == "Colon or Rectal Cancer")
    {
        diagnosis = "Colon";
    }
    if (diagnosis == "Thyroid Cancer")
    {
        diagnosis = "Thyroid";
    }
    // diagnosis == "Leukemia" would not actually change diagnosis string
    // FORWARD CHAINING
    Treatment treatmentObject; // CONSTRUCTOR
    // Start Forward Chaining
    treatmentObject.process(diagnosis);
    string patientTreatment = treatmentObject.getTreatment();
    // Print Diagnosis and Treatment
    cout << "Diagnosis: " << diagnosis << endl;
    cout << "Treatment: " << patientTreatment << endl;
    cout << endl;
    return 0;
}
```

backwardChain.h

```
// Class definition of BackwardChain
// Authors: Brittany Hale, John Courtright
#include <string>
#include <stack>
using namespace std;
class BackwardChain
{
private:
    // Data Members
    string conclt[14];  // Conclusion List
    string varlt[30];   // Variable List
    string clvarlt[73]; // Clause Variable List
    char varInt[30];    // Holds values of variables
    /* Stack stuff by Britt */
    stack<int> ruleStack;   // Stack for rule #s
    stack<int> clauseStack; // Stack for clause #s, stores slause #s for each rule at once.
    string var;       // Stored Variable
    string diagnosis; // Final Diagnosis
    string conclusion;
    int Ri = 0, Ci = 0; // Clause and Rule Num
    int conclusionPosition = 0;
    int finalConclusionPos = 0;
```

```
    int localized = 0;
    int suscancer = 0;
    // Private Function Prototypes
    bool search_conclusion_list(const string &var);
    int rule_to_clause(int Ri);
    void update_var_list(int Ci);
    void validate_Ri(int Ri, string &conclusion);
    void Process(const string &var);
    string getQuestion(int varNum);
public:
    // Constructor
    BackwardChain();
    // Public Function Prototypes
    string startBackwardChain(); // Entry point in main
    void printDiagnosis();     // Prints final diagnosis
};
```

backwardChain.cpp

```
// Implementation of the Backward Chain Class
// Authors: Brittany Hale, John Courtright
#include "backwardChain.h"
#include <stack>
#include <cstring>
#include <iostream>
#include <limits>
using namespace std;
// Constructor ~ Implemented by John
BackwardChain::BackwardChain()
{
  // Initialize Lists
  for (int i = 0; i < 13; i++)
    conclt[i] = "";
  for (int i = 0; i < 31; i++)
    varlt[i] = "";
  for (int i = 0; i < 72; i++)
    clvarlt[i] = "";
  for (int i = 0; i < 31; i++)
    varInt[i] = 'X';
  // Populate Conclusion List
  conclt[1] = "Cancer";
  conclt[2] = "LocalCancer";
  conclt[3] = "Skin Cancer";
  conclt[4] = "Basal Skin Cancer";
  conclt[5] = "Squamous Skin Cancer";
  conclt[6] = "PainCheck";
  conclt[7] = "Breast Cancer";
  conclt[8] = "Lung Cancer";
  conclt[9] = "Pancreatic Cancer";
  conclt[10] = "Colon or Rectal Cancer";
  conclt[11] = "Thyroid Cancer";
  conclt[12] = "Leukemia";
  // Populate Variable List
```

```
varlt[1] = "Cancer";
varlt[2] = "LocalCancer";
varlt[3] = "Sores";
varlt[4] = "Patches";
varlt[5] = "Warts";
varlt[6] = "NScars";
varlt[7] = "Pain";
varlt[8] = "PBreast";
varlt[9] = "BreastSkin";
varlt[10] = "BreastLump";
varlt[11] = "Cough";
varlt[12] = "BloodyCough";
varlt[13] = "LungInfec";
varlt[14] = "Clots";
varlt[15] = "FreqHeadache";
varlt[16] = "Diabetes";
varlt[17] = "YellowSkin";
varlt[18] = "AbPressure";
varlt[19] = "BowelChange";
varlt[20] = "BloodRectum";
varlt[21] = "NeckLump";
varlt[22] = "BreathSwallowIssue";
varlt[23] = "VoiceChanges";
varlt[24] = "ConstCough";
varlt[25] = "BleedBruiseEasily";
varlt[26] = "FeverChillsNightSweats";
varlt[27] = "EnlargeLiverSpleen";
varlt[28] = "SwolLymph";
// Clause Variable List (updated by John)
/* enter variables as they appear in the if clauses. a maximum
of 3 variables per if statement. if no more variables hit return key. */
clvarlt[1] = "Cancer";
clvarlt[7] = "LocalCancer";
clvarlt[13] = "LocalCancer";
clvarlt[19] = "SkinCancer";
clvarlt[20] = "Sores";
clvarlt[21] = "Patches";
clvarlt[22] = "Warts";
clvarlt[23] = "NScars";
clvarlt[25] = "SkinCancer";
clvarlt[26] = "Sores";
clvarlt[27] = "Patches";
clvarlt[28] = "Warts";
clvarlt[31] = "LocalCancer";
clvarlt[37] = "Pain";
clvarlt[38] = "PBreast";
clvarlt[39] = "BreastSkin";
clvarlt[40] = "BreastLump";
clvarlt[43] = "Pain";
clvarlt[44] = "Cough";
clvarlt[45] = "BloodyCough";
clvarlt[46] = "LungInfec";
```

```
 clvarlt[47] = "Clots";
 clvarlt[48] = "FreqHeadache";
 clvarlt[49] = "Pain";
 clvarlt[50] = "Diabetes";
 clvarlt[51] = "YellowSkin";
 clvarlt[52] = "Clots";
 clvarlt[55] = "Pain";
 clvarlt[56] = "AbPressure";
 clvarlt[57] = "BowelChange";
 clvarlt[58] = "BloodRectum";
 clvarlt[61] = "Pain";
 clvarlt[62] = "NeckLump";
 clvarlt[63] = "BreathSwallowIssue";
 clvarlt[64] = "VoiceChanges";
 clvarlt[65] = "ConstCough";
 clvarlt[67] = "Pain";
 clvarlt[68] = "BleedBruiseEasily";
 clvarlt[69] = "FeverChillsNightSweats";
 clvarlt[70] = "EnlargeLiverSpleen";
 clvarlt[71] = "SwolLymph";
 // Initialize Variables
 int Ri = 0;
 int Ci = 0;
 int conclusionPosition = 0;
 string var = "";
 //Backward Chaining System Initialized.
 return;
}
// Searches for a matching variable in the conclusion list
bool BackwardChain::search_conclusion_list(const string &var)
{
 /*
   This function finds the matching variable in the
   conclusion list & the corresponding rule number, Ri.
 */
 //Searching for var in Conclusion list...
 int conclusionPosition = 1;
 int i = 1;
 while (i <= 12 && strcmp(var.c_str(), conclt[i].c_str()) != 0) // Iterate up to and including the last element
 {
  // If the var (Stores conclusion from user) and current conclusion from conclt are different (not 0)
  i++; // Increment till we get to equal strings or not found
 }
 // If we found the var in the conclt (strcmp = 0)
 if (i <= 12 && strcmp(var.c_str(), conclt[i].c_str()) == 0) // Check if within bounds and strings are equal
 {
  conclusionPosition = i;
  finalConclusionPos = i;
  this->Ri = i * 10; // Assign rule number from index
  //cout << "Conclusion '" << var << "' found at position " << conclusionPosition << "." << endl;
  return true;
 }
```

```
  else
  {
    // No conclusion found in list
    //cout << "Conclusion '" << var << "' not found." << endl;
    return false;
  }
}
// Converts Rule # Ri to Clause # Ci
// Updated by Brittany Hale
int BackwardChain::rule_to_clause(int Ri)
{
  /*
    Rule #s are sequenced 10,20,30,40,50,...
    Each rule has 6 slots in the Clause Variable list.
    Formula:
  */
  //cout << "Ri is " << Ri << endl;
  int Ci = 6 * (Ri / 10 - 1) + 1;
  //cout << "Ci is " << Ci << endl;
  return Ci;
}
// void BackwardChain::update_clause_stack(int Ci) // For clause stack set for each rule
// {
//   for(i = 0; i < 6; i++){ // Ci to Ci + 5
//     if(clvarlt[Ci + i] != ""){ // If spot in Conclusion Var List is not empty
//       clauseStack.push(Ci + i); // Push Ci num into Clause Stack
//     }
//   }
// }
void BackwardChain::update_var_list(int Ci)
{
  // Push variables from Ci to Ci + 5 onto clauseStack
  for (int i = Ci; i < Ci + 6 && i < 73; i++)
  { // Ensure within bounds
    if (clvarlt[i] != "")
      clauseStack.push(i);
  }
  // Process each variable in the stack
  while (!clauseStack.empty())
  {
    int currentCi = clauseStack.top(); // Get top variable index
    clauseStack.pop();                 // Remove from stack
    string currentVar = clvarlt[currentCi]; // Get the variable name
    // Step 1: Check if current variable is also a conclusion
    if (search_conclusion_list(currentVar))
    {
      if (currentVar == "CANCER")
        suscancer++;
      if (currentVar == "LOCALC")
        localized++;
      if (localized > 2 || suscancer > 2)
      {
```

```
        Process(currentVar); // Recursively process this variable
      }
    }
    // Step 2: Check if the variable is already instantiated
    int varIndex = 0;
    for (int i = 1; i < 29; i++)
    {
     if (varlt[i] == currentVar)
     {
      varIndex = i;
      break;
     }
    }
    // If the variable is not instantiated, ask the user
    if (varInt[varIndex] == 'X')
    {                          // Assuming 'X' means uninitialized
     cout << getQuestion(varIndex); // Call helper function for prompts
     char userInput;
     cin >> userInput;
     cout << endl;
     // Validate input
     while (userInput != 'Y' && userInput != 'N' &&
          userInput != 'C' && userInput != 'S' && userInput != 'B')
     {
      cout << "Invalid input. Choose a valid character: ";
      cin >> userInput;
     }
     varInt[varIndex] = userInput; // Store the response
     //cout << "You set " << currentVar << " to: " << userInput << ".\n";
    }
  }
}
string BackwardChain::getQuestion(int varNum)
{
  switch (varNum)
  {
  case 1:
    return "Do you suspect cancer? (Y/N): ";
  case 2:
    return "Are the symptoms localized? (Y/N): ";
  case 3:
    return "Do you have sores? (Y/N): ";
  case 4:
    return "Do you have red patches? (Y/N): ";
  case 5:
    return "Do you have warts? (Y/N): ";
  case 6:
    return "Do you have new scars? (Y/N): ";
  case 7:
    return "Where are you experiencing pain? (C = Chest, S = Stomach, N = Neck, B = Bones): ";
  case 8:
    return "Do you have pain in the breast or nipple? (Y/N): ";
```

```
  case 9:
    return "Do you have red, irritated, or scaly skin on the breast? (Y/N): ";
  case 10:
    return "Do you have lumps in the breast, armpit, or collarbone? (Y/N): ";
  case 11:
    return "Do you have a cough or breathing issues? (Y/N): ";
  case 12:
    return "Have you coughed up blood? (Y/N): ";
  case 13:
    return "Have you had lung infections? (Y/N): ";
  case 14:
    return "Do you have blood clots? (Y/N): ";
  case 15:
    return "Do you experience frequent headaches? (Y/N): ";
  case 16:
    return "Do you have diabetes or experience diabetic symptoms? (Y/N): ";
  case 17:
    return "Do you have yellow skin? (Y/N): ";
  case 18:
    return "Is there pressure in your abdomen? (Y/N): ";
  case 19:
    return "Have you noticed bowel changes? (Y/N): ";
  case 20:
    return "Have you experienced blood from the rectum? (Y/N): ";
  case 21:
    return "Do you have neck swelling or a lump? (Y/N): ";
  case 22:
    return "Do you have breathing or swallowing issues? (Y/N): ";
  case 23:
    return "Has your voice changed? (Y/N): ";
  case 24:
    return "Do you have a constant cough? (Y/N): ";
  case 25:
    return "Do you bleed or bruise easily? (Y/N): ";
  case 26:
    return "Do you experience fever, chills, or night sweats? (Y/N): ";
  case 27:
    return "Do you have an enlarged liver or spleen? (Y/N): ";
  case 28:
    return "Do you have swollen lympth nodes? (Y/N): ";
  default:
    return "Enter 'Y' ";
  }
}
// Validates if a rule's 'if' conditions are met
void BackwardChain::validate_Ri(int Ri, string &conclusion)
{
  /*
    This function checks if the values of variables in the ‘if’ clauses of rule Ri
    match the values in the variable list & derived global variable list.
    If they do, it assigns the rule's conclusion to the 'conclusion' variable.
    Otherwise, it does nothing and returns.
```

```
*/
int conclusionMet = 0;
//cout << Ri << endl;
switch (Ri)
{
case 10: // Rule 10
 if (varInt[1] == 'N')
 {
  //cout << "Conclusion: Cancer = NO\n";
  conclusionMet = 1;
 }
 break;
case 20: // Rule 20
 if (varInt[2] == 'Y')
 {
  //cout << "Conclusion: Localized Cancers = YES\n";
  conclusionMet = 1;
 }
 break;
case 30: // Rule 30
 if (varInt[2] == 'N')
 {
  //cout << "Conclusion: Skin Cancers = YES\n";
  conclusionMet = 1;
 }
 break;
case 40: // Rule 40
 if (varInt[3] == 'Y' && varInt[4] == 'Y' && varInt[5] == 'N' && varInt[6] == 'Y')
 {
  //cout << "Conclusion: Basal Cell Skin Cancer = YES\n";
  conclusionMet = 1;
 }
 break;
case 50: // Rule 50
 if (varInt[3] == 'Y' && varInt[4] == 'Y' && varInt[5] == 'Y')
 {
  //cout << "Conclusion: Squamous Cell Skin Cancer = YES\n";
  conclusionMet = 1;
 }
 break;
case 60: // Rule 60
 if (varInt[2] == 'Y')
 {
  //cout << "Conclusion: CHECK PAIN\n";
  conclusionMet = 1;
 }
 break;
case 70: // Rule 70
 if (varInt[7] == 'C' && varInt[8] == 'Y' && varInt[9] == 'Y' && varInt[10] == 'Y')
 {
  //cout << "Conclusion: Breast Cancer = YES\n";
  conclusionMet = 1;
```

```
    }
    break;
  case 80: // Rule 80
    if (varInt[7] == 'C' && varInt[11] == 'Y' && varInt[12] == 'Y' && varInt[13] == 'Y' && varInt[14] == 'Y' && varInt[15] ==
'Y')
    {
      //cout << "Conclusion: Lung Cancer = YES\n";
      conclusionMet = 1;
    }
    break;
  case 90: // Rule 90
    if (varInt[7] == 'S' && varInt[16] == 'Y' && varInt[17] == 'Y' && varInt[14] == 'Y')
    {
      //cout << "Conclusion: Pancreatic Cancer = YES\n";
      conclusionMet = 1;
    }
    break;
  case 100: // Rule 100
    if (varInt[7] == 'S' && varInt[18] == 'Y' && varInt[19] == 'Y' && varInt[20] == 'Y')
    {
      //cout << "Conclusion: Colon or Rectal Cancer = YES\n";
      conclusionMet = 1;
    }
    break;
  case 110: // Rule 110
    if (varInt[7] == 'N' && varInt[21] == 'Y' && varInt[22] == 'Y' && varInt[23] == 'Y' && varInt[24] == 'Y')
    {
      //cout << "Conclusion: Thyroid Cancer = YES\n";
      conclusionMet = 1;
    }
    break;
  case 120: // Rule 120
    if (varInt[7] == 'B' && varInt[25] == 'Y' && varInt[26] == 'Y' && varInt[27] == 'Y' && varInt[28] == 'Y')
    {
      //cout << "Conclusion: Leukemia = YES\n";
      conclusionMet = 1;
    }
    break;
  default:
    cout << "Invalid rule number.\n";
  }
  if (conclusionMet == 1)
  {
    //Conclusion met successfully.
    diagnosis = conclt[finalConclusionPos];
  }
  else
  {
    //No conclusion met for the given rule.
    return;
  }
}
```

```
// Processes backward chaining recursively
void BackwardChain::Process(const string &var)
{
 /*
  Runs a loop that:
  1. Calls search_conclusion_list(variable) to find matching variable & rule Ri.
  2. Calls rule_to_clause(Ri) to convert Ri to clause Ci.
  3. Calls update_var_list(Ci) to instantiate variables (may trigger recursion).
  4. Calls validate_Ri(Ri, conclusion) to verify rule satisfaction.
  5. Saves conclusion in the derived global variable list.
  6. If values don't match, continues to the next conclusion.
 */
 search_conclusion_list(var); // Converts conclusion pos to rule index (Ri)
 Ci = rule_to_clause(Ri);
 update_var_list(Ci);
 validate_Ri(Ri, conclusion);
}
// Public Member Function that Starts Process
string BackwardChain::startBackwardChain()
{
 //cout << "Backward Chain Diagnosis Begin..." << endl;
 cout << "Here are the possible conclusions: " << endl;
 for (int i = 1; i < 13; i++)
 {
  if (i == 1 || i == 2 || i == 3 || i == 6) continue; // Skip these conclusions, not intuitievly useful
  cout << "CONCLUSION " << i << " " << conclt[i] << "\n";
 }
 bool valid = false;
 int userInput = 0;
 // Validate before calling Process
 while (!valid)
 {
  //cout << "Select a conclusion from the list by inputting the corresponding number: ";
  cout << "Please select a cancer type you want to check for by inputting the corresponding number: ";
  cin >> userInput;
  if (cin.fail())
  {
   cout << "Invalid input. Please enter a valid number from the list.\n";
   cin.clear();                                  // Clear error flags
   cin.ignore(numeric_limits<streamsize>::max(), '\n'); // Discard invalid input
   continue;
  }
  if (userInput >= 1 && userInput <= 12)
  {
   valid = true;
  }
  else
  {
   cout << "Invalid choice. Please enter a number between 1 and 12.\n";
   cin.clear(); // Just in case extra characters are left in the buffer
   cin.ignore(numeric_limits<streamsize>::max(), '\n');
  }
```

```
 }
 finalConclusionPos = userInput;
 string choice = conclt[userInput];
 cout << "You selected: " << choice << endl << endl;
 // Proceed with the backward chaining process using userInput
 Process(choice);
 printDiagnosis();
 cout << endl;
 return diagnosis;
}
// Public Member Function that Retreives Diagnosis Variable
// Implemented by John
void BackwardChain::printDiagnosis()
{
 if (diagnosis == "")
 {
  cout << "No diagnosis has been determined." << endl;
  return;
 }
 cout << "Your final diagnosis is " << diagnosis << "." << endl;
 return;
}
```

forwardChain.h

```
// Implementation of the Forward Chain Class
// Author: Robert Jones
#include <string>
#include <iostream>
using namespace std;
class Treatment
{
   private:
      string clauseVL[271][9];
      string variableList[23][2];
      string ruleList[31][9];
      string treatmentResult;
      int globalConclusionsCounter;
      string globalConclusions[31];
      string treatment;
   public:
      //Constructor
      Treatment();
      void initializeCVL();
      void initializeVarList();
      void initializeRuleList();
      void search_cvl(string var);
      void clause_to_rule (int var);
      void update_VL(int var);
      void validate_Ri(int var);
      void process(string var);
      void printGlobalConclusions();
      void modifyTreatmentString();
```

```
    string getTreatment();
};
```

forwardChain.cpp

```
// Implementation of the Forward Chain Class
// Author: Robert Jones
#include "forwardChain.h"
Treatment::Treatment()
{
   initializeCVL();
   initializeVarList();
   initializeRuleList();
   globalConclusionsCounter = 0;
}
// Initializes the clause variable list into a 2D array. Empty strings represent the spaces. If not empty, clause variables will be accessible with indexs [clauseIndex][0-7]
void Treatment::initializeCVL()
{
 clauseVL[1][0] = "Cancer=Basal"; clauseVL[1][1] = "Centralized=YES"; clauseVL[1][2]= "Excisable=YES"; clauseVL[1][3] = "";
      clauseVL[1][4] = ""; clauseVL[1][5] = ""; clauseVL[1][6] = ""; clauseVL[1][7] = ""; clauseVL[1][8] = "";
   clauseVL[10][0] = "Cancer=Basal"; clauseVL[10][1] = "Centralized=YES"; clauseVL[10][2]= "Excisable=NO"; clauseVL[10][3] = "";
      clauseVL[10][4] = ""; clauseVL[10][5] = ""; clauseVL[10][6] = ""; clauseVL[10][7] = ""; clauseVL[10][8] = "";
   clauseVL[19][0] = "Cancer=Basal"; clauseVL[19][1] = "Centralized=NO"; clauseVL[19][2] = ""; clauseVL[19][3] = ""; clauseVL[19][4] = ""; clauseVL[19][5] = "";
      clauseVL[19][6] = ""; clauseVL[19][7] = ""; clauseVL[19][8] = "";
   clauseVL[28][0] = "Cancer=Squamous"; clauseVL[28][1] = "Centralized=YES"; clauseVL[28][2] = "Excisable=YES"; clauseVL[28][3] = ""; clauseVL[28][4] = "";
      clauseVL[28][5] = ""; clauseVL[28][6] = ""; clauseVL[28][7] = ""; clauseVL[28][8] = "";
   clauseVL[37][0] = "Cancer=Squamous"; clauseVL[37][1] = "Centralized=YES"; clauseVL[37][2] = "Excisable=NO"; clauseVL[37][3] = ""; clauseVL[37][4] = "";
      clauseVL[37][5] = ""; clauseVL[37][6] = ""; clauseVL[37][7] = ""; clauseVL[37][8] = "";
   clauseVL[46][0] = "Cancer=Squamous"; clauseVL[46][1] = "Centralized=NO"; clauseVL[46][2] = "Weak=YES"; clauseVL[46][3] = ""; clauseVL[46][4] = "";
      clauseVL[46][5] = ""; clauseVL[46][6] = ""; clauseVL[46][7] = ""; clauseVL[46][8] = "";
   clauseVL[55][0] = "Cancer=Squamous"; clauseVL[55][1] = "Centralized=NO"; clauseVL[55][2] = "Weak=NO"; clauseVL[55][3] = ""; clauseVL[55][4] = "";
      clauseVL[55][5] = ""; clauseVL[55][6] = ""; clauseVL[55][7] = ""; clauseVL[55][8] = "";
   clauseVL[64][0] = "Cancer=Breast"; clauseVL[64][1] = "Centralized=NO"; clauseVL[64][2] = "Late=YES"; clauseVL[64][3] = ""; clauseVL[64][4] = "";
      clauseVL[64][5] = ""; clauseVL[64][6] = ""; clauseVL[64][7] = ""; clauseVL[64][8] = "";
   clauseVL[73][0]= "Cancer=Breast"; clauseVL[73][1] = "Centralized=NO"; clauseVL[73][2] = "Late=NO"; clauseVL[73][3] = "Large=YES"; clauseVL[73][4] = "";
      clauseVL[73][5] = ""; clauseVL[73][6] = ""; clauseVL[73][7] = ""; clauseVL[73][8] = "";
   clauseVL[82][0]= "Cancer=Breast"; clauseVL[82][1] = "Centralized=NO"; clauseVL[82][2] = "Late=NO"; clauseVL[82][3] = "Large=NO"; clauseVL[82][4] = "";
      clauseVL[82][5] = ""; clauseVL[82][6] = ""; clauseVL[82][7] = ""; clauseVL[82][8] = "";
   clauseVL[91][0]= "Cancer=Breast"; clauseVL[91][1] = "Centralized=YES"; clauseVL[91][2] = "Excisable=YES"; clauseVL[91][3] = ""; clauseVL[91][4] = "";
      clauseVL[91][5] = ""; clauseVL[91][6] = ""; clauseVL[91][7] = ""; clauseVL[91][8] = "";
```

```
   clauseVL[100][0]= "Cancer=Breast"; clauseVL[100][1] = "Centralized=YES"; clauseVL[100][2] = "Excisable=NO";
clauseVL[100][3] = ""; clauseVL[100][4] = "";
      clauseVL[100][5] = ""; clauseVL[100][6] = ""; clauseVL[100][7] = ""; clauseVL[100][8] = "";
   clauseVL[109][0] = "Cancer=Colon"; clauseVL[109][1] = "Early=YES"; clauseVL[109][2] = "Excisable=YES";
clauseVL[109][3] = ""; clauseVL[109][4] = "";
      clauseVL[109][5] = ""; clauseVL[109][6] = ""; clauseVL[109][7] = ""; clauseVL[109][8] = "";
   clauseVL[118][0] = "Cancer=Colon"; clauseVL[118][1] = "Early=YES"; clauseVL[118][2] = "Excisable=NO";
clauseVL[118][3] = ""; clauseVL[118][4] = "";
      clauseVL[118][5] = ""; clauseVL[118][6] = ""; clauseVL[118][7] = ""; clauseVL[118][8] = "";
   clauseVL[127][0] = "Cancer=Colon"; clauseVL[127][1] = "Early=NO"; clauseVL[127][2] = ""; clauseVL[127][3] = "";
clauseVL[127][4] = "";
      clauseVL[127][5] = ""; clauseVL[127][6] = ""; clauseVL[127][7] = ""; clauseVL[127][8] = "";
   clauseVL[136][0] = "Cancer=Leukemia"; clauseVL[136][1] = "Acute=YES"; clauseVL[136][2] = "Myeloid=YES";
clauseVL[136][3] = ""; clauseVL[136][4] = "";
      clauseVL[136][5] = ""; clauseVL[136][6] = ""; clauseVL[136][7] = ""; clauseVL[136][8] = "";
      //Myeloid=NO is the same as Lymphoid=YES
   clauseVL[145][0] = "Cancer=Leukemia"; clauseVL[145][1] = "Acute=YES"; clauseVL[145][2] = "Myeloid=NO";
clauseVL[145][3] = ""; clauseVL[145][4] = "";
      clauseVL[145][5] = ""; clauseVL[145][6] = ""; clauseVL[145][7] = ""; clauseVL[145][8] = "";
   clauseVL[154][0] = "Cancer=Leukemia"; clauseVL[154][1] = "Chronic=YES"; clauseVL[154][2] = "Myeloid=YES";
clauseVL[154][3] = ""; clauseVL[154][4] = "";
      clauseVL[154][5] = ""; clauseVL[154][6] = ""; clauseVL[154][7] = ""; clauseVL[154][8] = "";
   clauseVL[163][0] = "Cancer=Leukemia"; clauseVL[163][1] = "Chronic=YES"; clauseVL[163][2] = "Myeloid=NO";
clauseVL[163][3] = "Urgent=YES"; clauseVL[163][4] = "";
      clauseVL[163][5] = ""; clauseVL[163][6] = ""; clauseVL[163][7] = ""; clauseVL[163][8] = "";
   clauseVL[172][0] = "Cancer=Leukemia"; clauseVL[172][1] = "Chronic=YES"; clauseVL[172][2] ="Myeloid=NO";
clauseVL[172][3] = "Urgent=NO"; clauseVL[172][4] = "";
      clauseVL[172][5] = ""; clauseVL[172][6] = ""; clauseVL[172][7] = ""; clauseVL[172][8] = "";
   clauseVL[181][0] = "Cancer=Lung"; clauseVL[181][1] = "Small=NO"; clauseVL[181][2] = "Early=YES"; clauseVL[181][3]
= ""; clauseVL[181][4] = "";
      clauseVL[181][5] = ""; clauseVL[181][6] = ""; clauseVL[181][7] = ""; clauseVL[181][8] = "";
   clauseVL[190][0] = "Cancer=Lung"; clauseVL[190][1] = "Small=NO"; clauseVL[190][2] = "Early=NO"; clauseVL[190][3] =
""; clauseVL[190][4] = "";
      clauseVL[190][5] = ""; clauseVL[190][6] = ""; clauseVL[190][7] = ""; clauseVL[190][8] = "";
   clauseVL[199][0] = "Cancer=Lung"; clauseVL[199][1] = "Small=YES"; clauseVL[199][2] = "Excisable=YES";
clauseVL[199][3] = ""; clauseVL[199][4] = "";
      clauseVL[199][5] = ""; clauseVL[199][6] = ""; clauseVL[199][7] = ""; clauseVL[199][8] = "";
   clauseVL[208][0] = "Cancer=Lung"; clauseVL[208][1] = "Small=YES"; clauseVL[208][2] = "Excisable=NO";
clauseVL[208][3] = ""; clauseVL[208][4] = "";
      clauseVL[208][5] = ""; clauseVL[208][6] = ""; clauseVL[208][7] = ""; clauseVL[208][8] = "";
   clauseVL[217][0] = "Cancer=Pancreatic"; clauseVL[217][1] = "Excisable=YES"; clauseVL[217][2] = "Risk=YES";
clauseVL[217][3] = ""; clauseVL[217][4] = "";
      clauseVL[217][5] = ""; clauseVL[217][6] = ""; clauseVL[217][7] = ""; clauseVL[217][8] = "";
   clauseVL[226][0] = "Cancer=Pancreatic"; clauseVL[226][1] = "Excisable=YES"; clauseVL[226][2] = "Risk=NO";
clauseVL[226][3] = ""; clauseVL[226][4] = "";
      clauseVL[226][5] = ""; clauseVL[226][6] = ""; clauseVL[226][7] = ""; clauseVL[226][8] = "";
   clauseVL[235][0] = "Cancer=Pancreatic"; clauseVL[235][1] = "Excisable=NO"; clauseVL[235][2] = ""; clauseVL[235][3] =
""; clauseVL[235][4] = "";
      clauseVL[235][5] = ""; clauseVL[235][6] = ""; clauseVL[235][7] = ""; clauseVL[235][8] = "";
   clauseVL[244][0] = "Cancer=Thyroid"; clauseVL[244][1] = "Covered=YES"; clauseVL[244][2] = "Centralized=NO";
clauseVL[244][3] = ""; clauseVL[244][4] = "";
      clauseVL[244][5] = ""; clauseVL[244][6] = ""; clauseVL[244][7] = ""; clauseVL[244][8] = "";
```

```
   clauseVL[253][0] = "Cancer=Thyroid"; clauseVL[253][1] = "Covered=YES"; clauseVL[253][2] = "Centralized=YES";
clauseVL[253][3] = ""; clauseVL[253][4] = "";
      clauseVL[253][5] = ""; clauseVL[253][6] = ""; clauseVL[253][7] = ""; clauseVL[253][8] = "";
   clauseVL[262][0] = "Cancer=Thyroid"; clauseVL[262][1] = "Covered=NO"; clauseVL[262][2] = ""; clauseVL[262][3] = "";
clauseVL[262][4] = "";
      clauseVL[262][5] = ""; clauseVL[262][6] = ""; clauseVL[262][7] = ""; clauseVL[262][8] = "";
}
void Treatment::initializeVarList()
{
   variableList[0][0] = "Cancer"; variableList[0][1] = "";
   variableList[1][0] = "Basal"; variableList[1][1] = "";
   variableList[2][0] = "Centralized"; variableList[2][1] = "";
   variableList[3][0] = "Excisable"; variableList[3][1] = "";
   variableList[4][0] = "Squamous"; variableList[4][1] = "";
   variableList[5][0] = "Weak"; variableList[5][1] = "";
   variableList[6][0] = "Breast"; variableList[6][1] = "";
   variableList[7][0] = "Late"; variableList[7][1] = "";
   variableList[8][0] = "Large"; variableList[8][1] = "";
   variableList[9][0] = "Colon"; variableList[9][1] = "";
   variableList[10][0] = "Early"; variableList[10][1] = "";
   variableList[11][0] = "Leukemia"; variableList[11][1] = "";
   variableList[12][0] = "Acute"; variableList[12][1] = "";
   variableList[13][0] = "Myeloid"; variableList[13][1] = "";
   variableList[14][0] = "Lymphoid"; variableList[14][1] = "";
   variableList[15][0] = "Chronic"; variableList[15][1] = "";
   variableList[16][0] = "Urgent"; variableList[16][1] = "";
   variableList[17][0] = "Lung"; variableList[17][1] = "";
   variableList[18][0] = "Small"; variableList[18][1] = "";
   variableList[19][0] = "Pancreatic"; variableList[19][1] = "";
   variableList[20][0] = "Risk"; variableList[20][1] = "";
   variableList[21][0] = "Thyroid"; variableList[21][1] = "";
   variableList[22][0] = "Covered"; variableList[22][1] = "";
}
void Treatment::initializeRuleList()
{

   ruleList[1][0] = "Cancer=Basal"; ruleList[1][1] = "Centralized=YES"; ruleList[1][2]= "Excisable=YES"; ruleList[1][3] = "";
      ruleList[1][4] = ""; ruleList[1][5] = ""; ruleList[1][6] = ""; ruleList[1][7] = ""; ruleList[1][8] = "Treatment=Targeted
Radiation";
   ruleList[2][0] = "Cancer=Basal"; ruleList[2][1] = "Centralized=YES"; ruleList[2][2]= "Excisable=NO"; ruleList[2][3] = "";
      ruleList[2][4] = ""; ruleList[2][5] = ""; ruleList[2][6] = ""; ruleList[2][7] = ""; ruleList[2][8] = "Treatment=Surgical
Removal";
   ruleList[3][0] = "Cancer=Basal"; ruleList[3][1] = "Centralized=NO"; ruleList[3][2] = ""; ruleList[3][3] = ""; ruleList[3][4] =
""; ruleList[3][5] = "";
      ruleList[3][6] = ""; ruleList[3][7] = ""; ruleList[3][8] = "Treatment=Immunotherapy";
   ruleList[4][0] = "Cancer=Squamous"; ruleList[4][1] = "Centralized=YES"; ruleList[4][2] = "Excisable=YES"; ruleList[4][3] =
""; ruleList[4][4] = "";
      ruleList[4][5] = ""; ruleList[4][6] = ""; ruleList[4][7] = ""; ruleList[4][8] = "Treatment=Surgical Removal";
   ruleList[5][0] = "Cancer=Squamous"; ruleList[5][1] = "Centralized=YES"; ruleList[5][2] = "Excisable=NO"; ruleList[5][3] =
""; ruleList[5][4] = "";
      ruleList[5][5] = ""; ruleList[5][6] = ""; ruleList[5][7] = ""; ruleList[5][8] = "Treatment=Targeted Radiation";
```

```
    ruleList[6][0] = "Cancer=Squamous"; ruleList[6][1] = "Centralized=NO"; ruleList[6][2] = "Weak=YES"; ruleList[6][3] = "";
ruleList[6][4] = "";
        ruleList[6][5] = ""; ruleList[6][6] = ""; ruleList[6][7] = ""; ruleList[6][8] = "Treatment=Chemotherapy";
    ruleList[7][0] = "Cancer=Squamous"; ruleList[7][1] = "Centralized=NO"; ruleList[7][2] = "Weak=NO"; ruleList[7][3] = "";
ruleList[7][4] = "";
        ruleList[7][5] = ""; ruleList[7][6] = ""; ruleList[7][7] = ""; ruleList[7][8] = "Treatment=Immunotherapy";
    ruleList[8][0] = "Cancer=Breast"; ruleList[8][1] = "Centralized=NO"; ruleList[8][2] = "Late=YES"; ruleList[8][3] = "";
ruleList[8][4] = "";
        ruleList[8][5] = ""; ruleList[8][6] = ""; ruleList[8][7] = ""; ruleList[8][8] = "Treatment=Systemic Therapy";
    ruleList[9][0]= "Cancer=Breast"; ruleList[9][1] = "Centralized=NO"; ruleList[9][2] = "Late=NO"; ruleList[9][3] =
"Large=YES"; ruleList[9][4] = "";
        ruleList[9][5] = ""; ruleList[9][6] = ""; ruleList[9][7] = ""; ruleList[9][8] = "Treatment=Systemic Therapy then Surgical
Removal";
    ruleList[10][0]= "Cancer=Breast"; ruleList[10][1] = "Centralized=NO"; ruleList[10][2] = "Late=NO"; ruleList[10][3] =
"Large=NO"; ruleList[10][4] = "";
        ruleList[10][5] = ""; ruleList[10][6] = ""; ruleList[10][7] = ""; ruleList[10][8] = "Treatment=Surgical Removal";
    ruleList[11][0]= "Cancer=Breast"; ruleList[11][1] = "Centralized=YES"; ruleList[11][2] = "Excisable=YES"; ruleList[11][3] =
""; ruleList[11][4] = "";
        ruleList[11][5] = ""; ruleList[11][6] = ""; ruleList[11][7] = ""; ruleList[11][8] = "Treatment=Surgical Removal";
    ruleList[12][0]= "Cancer=Breast"; ruleList[12][1] = "Centralized=YES"; ruleList[12][2] = "Excisable=NO"; ruleList[12][3] =
""; ruleList[12][4] = "";
        ruleList[12][5] = ""; ruleList[12][6] = ""; ruleList[12][7] = ""; ruleList[12][8] = "Treatment=Targeted Radiation";
    ruleList[13][0] = "Cancer=Colon"; ruleList[13][1] = "Early=YES"; ruleList[13][2] = "Excisable=YES"; ruleList[13][3] = "";
ruleList[13][4] = "";
        ruleList[13][5] = ""; ruleList[13][6] = ""; ruleList[13][7] = ""; ruleList[13][8] = "Treatment=Colonoscopy";
    ruleList[14][0] = "Cancer=Colon"; ruleList[14][1] = "Early=YES"; ruleList[14][2] = "Excisable=NO"; ruleList[14][3] = "";
ruleList[14][4] = "";
        ruleList[14][5] = ""; ruleList[14][6] = ""; ruleList[14][7] = ""; ruleList[14][8] = "Treatment=Colectomy";
    ruleList[15][0] = "Cancer=Colon"; ruleList[15][1] = "Early=NO"; ruleList[15][2] = ""; ruleList[15][3] = ""; ruleList[15][4] =
"";
        ruleList[15][5] = ""; ruleList[15][6] = ""; ruleList[15][7] = ""; ruleList[15][8] = "Treatment=Colectomy";
    ruleList[16][0] = "Cancer=Leukemia"; ruleList[16][1] = "Acute=YES"; ruleList[16][2] = "Myeloid=YES"; ruleList[16][3] =
""; ruleList[16][4] = "";
        ruleList[16][5] = ""; ruleList[16][6] = ""; ruleList[16][7] = ""; ruleList[16][8] = "Treatment=ATRA Drug Therapy";
        //Myeloid=NO is the same as Lymphoid=YES
    ruleList[17][0] = "Cancer=Leukemia"; ruleList[17][1] = "Acute=YES"; ruleList[17][2] = "Myeloid=NO"; ruleList[17][3] = "";
ruleList[17][4] = "";
        ruleList[17][5] = ""; ruleList[17][6] = ""; ruleList[17][7] = ""; ruleList[17][8] = "Treatment=Chemotherapy";
    ruleList[18][0] = "Cancer=Leukemia"; ruleList[18][1] = "Chronic=YES"; ruleList[18][2] = "Myeloid=YES"; ruleList[18][3] =
""; ruleList[18][4] = "";
        ruleList[18][5] = ""; ruleList[18][6] = ""; ruleList[18][7] = ""; ruleList[18][8] = "Treatment=TKI Inhibitor";
    ruleList[19][0] = "Cancer=Leukemia"; ruleList[19][1] = "Chronic=YES"; ruleList[19][2] = "Myeloid=NO"; ruleList[19][3] =
"Urgent=YES"; ruleList[19][4] = "";
        ruleList[19][5] = ""; ruleList[19][6] = ""; ruleList[19][7] = ""; ruleList[19][8] = "Treatment=Targeted Drugs, Chemoterapy,
or Immunotherapy";
    ruleList[20][0] = "Cancer=Leukemia"; ruleList[20][1] = "Chronic=YES"; ruleList[20][2] ="Myeloid=NO"; ruleList[20][3] =
"Urgent=NO"; ruleList[20][4] = "";
        ruleList[20][5] = ""; ruleList[20][6] = ""; ruleList[20][7] = ""; ruleList[20][8] = "Treatment=Low-Dose Radiation or
Surgical Removal";
    ruleList[21][0] = "Cancer=Lung"; ruleList[21][1] = "Small=NO"; ruleList[21][2] = "Early=YES"; ruleList[21][3] = "";
ruleList[21][4] = "";
```

```
    ruleList[21][5] = ""; ruleList[21][6] = ""; ruleList[21][7] = ""; ruleList[21][8] = "Treatment=Surgical Removal or Targeted Therapy then Removal";
  ruleList[22][0] = "Cancer=Lung"; ruleList[22][1] = "Small=NO"; ruleList[22][2] = "Early=NO"; ruleList[22][3] = ""; ruleList[22][4] = "";
    ruleList[22][5] = ""; ruleList[22][6] = ""; ruleList[22][7] = ""; ruleList[22][8] = "Treatment=Radiation, Chemo, or Immunotherapy";
  ruleList[23][0] = "Cancer=Lung"; ruleList[23][1] = "Small=YES"; ruleList[23][2] = "Excisable=YES"; ruleList[23][3] = ""; ruleList[23][4] = "";
    ruleList[23][5] = ""; ruleList[23][6] = ""; ruleList[23][7] = ""; ruleList[23][8] = "Treatment=Surgical Removal then Chemotherapy";
  ruleList[24][0] = "Cancer=Lung"; ruleList[24][1] = "Small=YES"; ruleList[24][2] = "Excisable=NO"; ruleList[24][3] = ""; ruleList[24][4] = "";
    ruleList[24][5] = ""; ruleList[24][6] = ""; ruleList[24][7] = ""; ruleList[24][8] = "Treatment=Chemoraditation";
  ruleList[25][0] = "Cancer=Pancreatic"; ruleList[25][1] = "Excisable=YES"; ruleList[25][2] = "Risk=YES"; ruleList[25][3] = ""; ruleList[25][4] = "";
    ruleList[25][5] = ""; ruleList[25][6] = ""; ruleList[25][7] = ""; ruleList[25][8] = "Treatment=Chemo or Raditation Therapy then consider Surgery";
  ruleList[26][0] = "Cancer=Pancreatic"; ruleList[26][1] = "Excisable=YES"; ruleList[26][2] = "Risk=NO"; ruleList[26][3] = ""; ruleList[26][4] = "";
    ruleList[26][5] = ""; ruleList[26][6] = ""; ruleList[26][7] = ""; ruleList[26][8] = "Treatment=Surgical Removal";
  ruleList[27][0] = "Cancer=Pancreatic"; ruleList[27][1] = "Excisable=NO"; ruleList[27][2] = ""; ruleList[27][3] = ""; ruleList[27][4] = "";
    ruleList[27][5] = ""; ruleList[27][6] = ""; ruleList[27][7] = ""; ruleList[27][8] = "Treatment=Chemotherapy";
  ruleList[28][0] = "Cancer=Thyroid"; ruleList[28][1] = "Covered=YES"; ruleList[28][2] = "Centralized=NO"; ruleList[28][3] = ""; ruleList[28][4] = "";
    ruleList[28][5] = ""; ruleList[28][6] = ""; ruleList[28][7] = ""; ruleList[28][8] = "Treatment=Radical Neck Dissection";
  ruleList[29][0] = "Cancer=Thyroid"; ruleList[29][1] = "Covered=YES"; ruleList[29][2] = "Centralized=YES"; ruleList[29][3] = ""; ruleList[29][4] = "";
    ruleList[29][5] = ""; ruleList[29][6] = ""; ruleList[29][7] = ""; ruleList[29][8] = "Treatment=Thyroidectomy";
  ruleList[30][0] = "Cancer=Thyroid"; ruleList[30][1] = "Covered=NO"; ruleList[30][2] = ""; ruleList[30][3] = ""; ruleList[30][4] = "";
    ruleList[30][5] = ""; ruleList[30][6] = ""; ruleList[30][7] = ""; ruleList[30][8] = "Treatment=Lobectomy";
  /*ruleList[31][0] = "Basal=NO"; ruleList[31][1] = "Squamous=NO"; ruleList[31][2] = "Breast=NO"; ruleList[31][3] = "Colon=NO"; ruleList[31][4] = "Leukemia=NO";
    ruleList[31][5] = "Lung=NO"; ruleList[31][6] = "Pancreatic=NO"; ruleList[31][7] = "Thyroid=NO"; ruleList[31][8] = "Treatment=None";*/
  }
void Treatment::search_cvl(string variable)
{
  for (int CI = 1; CI < 271; CI++)
  {
    if(clauseVL[CI][0]== variable)
    {
      update_VL(CI);
      clause_to_rule(CI);
    }
  }
  return;
}
void Treatment::clause_to_rule(int clauseNumber)
{
  int RI = ((clauseNumber/9)+1);
```

```
    validate_Ri(RI);
    return;
}
void Treatment::update_VL(int clauseNumber)
{
    string preModVar;
    string postModVar;
    for (int i = 1; i < 8; i++)
    {
        if (clauseVL[clauseNumber][i] != "")
        {
            preModVar = clauseVL[clauseNumber][i];
            std::size_t seperator = preModVar.find("=");
            postModVar = preModVar.substr(0, seperator);

            string userInput;
            if(postModVar == "Basal")
            {
                if(variableList[1][1] == "")
                {
                    do{
                        cout<< "Patient has Basal Cell Carcinoma? (YES/NO) ";
                        cin >> userInput;
                    }while(userInput != "YES" && userInput != "NO");
                    variableList[1][1] = userInput;
                    globalConclusions[globalConclusionsCounter]=(variableList[1][0] + "=" + variableList[1][1]);
                    globalConclusionsCounter++;
                }
            }
            if(postModVar == "Centralized")
            {
                if(variableList[2][1] == "")
                {
                    do{
                        cout<< "Is it Centralized? (YES/NO) ";
                        cin >> userInput;
                    }while(userInput != "YES" && userInput != "NO");
                    variableList[2][1] = userInput;
                    globalConclusions[globalConclusionsCounter]=(variableList[2][0] + "=" + variableList[2][1]);
                    globalConclusionsCounter++;
                }
            }
            if(postModVar == "Excisable")
            {
                if(variableList[3][1] == "")
                {
                    do{
                        cout<< "Is it Excisable? (YES/NO) ";
                        cin >> userInput;
                    }while(userInput != "YES" && userInput != "NO");
                    variableList[3][1] = userInput;
                    globalConclusions[globalConclusionsCounter]=(variableList[3][0] + "=" + variableList[3][1]);
```

```
            globalConclusionsCounter++;
        }
    }
    if(postModVar == "Squamous")
    {
        if(variableList[4][1] == "")
        {
            do{
                cout<< "Patient has squamous cell carcinoma? (YES/NO) ";
                cin >> userInput;
            }while(userInput != "YES" && userInput != "NO");
            variableList[4][1] = userInput;
            globalConclusions[globalConclusionsCounter]=(variableList[4][0] + "=" + variableList[4][1]);
            globalConclusionsCounter++;
        }
    }
    if(postModVar == "Weak")
    {
        if(variableList[5][1] == "")
        {
            do{
                cout<< "Is the patient experiencing weakness? (YES/NO) ";
                cin >> userInput;
            }while(userInput != "YES" && userInput != "NO");
            variableList[5][1] = userInput;
            globalConclusions[globalConclusionsCounter]=(variableList[5][0] + "=" + variableList[5][1]);
            globalConclusionsCounter++;
        }
    }
    if(postModVar == "Breast")
    {
        if(variableList[6][1] == "")
        {
            do{
                cout<< "Patient has Breast cancer? (YES/NO) ";
                cin >> userInput;
            }while(userInput != "YES" && userInput != "NO");
            variableList[6][1] = userInput;
            globalConclusions[globalConclusionsCounter]=(variableList[6][0] + "=" + variableList[6][1]);
            globalConclusionsCounter++;
        }
    }
    if(postModVar == "Late")
    {
        if(variableList[7][1] == "")
        {
            do{
                cout<< "Is it Late stage? (YES/NO) ";
                cin >> userInput;
            }while(userInput != "YES" && userInput != "NO");
            variableList[7][1] = userInput;
            globalConclusions[globalConclusionsCounter]=(variableList[7][0] + "=" + variableList[7][1]);
```

```
            globalConclusionsCounter++;
        }
    }
    if(postModVar == "Large")
    {
        if(variableList[8][1] == "")
        {
            do{
                cout<< "Is it Large? (YES/NO) ";
                cin >> userInput;
            }while(userInput != "YES" && userInput != "NO");
            variableList[8][1] = userInput;
            globalConclusions[globalConclusionsCounter]=(variableList[8][0] + "=" + variableList[8][1]);
            globalConclusionsCounter++;
        }
    }
    if(postModVar == "Colon")
    {
        if(variableList[9][1] == "")
        {
            do{
                cout<< "Patient has Colon cancer? (YES/NO) ";
                cin >> userInput;
            }while(userInput != "YES" && userInput != "NO");
            variableList[9][1] = userInput;
            globalConclusions[globalConclusionsCounter]=(variableList[9][0] + "=" + variableList[9][1]);
            globalConclusionsCounter++;
        }
    }
    if(postModVar == "Early")
    {
        if(variableList[10][1] == "")
        {
            do{
                cout<< "Was the cancer caught early? (YES/NO) ";
                cin >> userInput;
            }while(userInput != "YES" && userInput != "NO");
            variableList[10][1] = userInput;
            globalConclusions[globalConclusionsCounter]=(variableList[10][0] + "=" + variableList[10][1]);
            globalConclusionsCounter++;
        }
    }
    if(postModVar == "Leukemia")
    {
        if(variableList[11][1] == "")
        {
            do{
                cout<< "Patient has Leukemia? (YES/NO) ";
                cin >> userInput;
            }while(userInput != "YES" && userInput != "NO");
            variableList[11][1] = userInput;
            globalConclusions[globalConclusionsCounter]=(variableList[11][0] + "=" + variableList[11][1]);
```

```
            globalConclusionsCounter++;
        }
    }
    if(postModVar == "Acute")
    {
        if(variableList[12][1] == "")
        {
            do{
                cout<< "Is it Acute Leukemia? (YES/NO) ";
                cin >> userInput;
            }while(userInput != "YES" && userInput != "NO");
            variableList[12][1] = userInput;
            globalConclusions[globalConclusionsCounter]=(variableList[12][0] + "=" + variableList[12][1]);
            globalConclusionsCounter++;
        }
    }
    if(postModVar == "Myeloid")
    {
        if(variableList[13][1] == "")
        {
            do{
                cout<< "Did it start in Myeloid cells? (YES/NO) ";
                cin >> userInput;
            }while(userInput != "YES" && userInput != "NO");
            variableList[13][1] = userInput;
            globalConclusions[globalConclusionsCounter]=(variableList[13][0] + "=" + variableList[13][1]);
            globalConclusionsCounter++;
        }
    }
    if(postModVar == "Lymphoid")
    {
        if(variableList[14][1] == "")
        {
            do{
                cout<< "Did it start in Lymphoid cells? (YES/NO) ";
                cin >> userInput;
            }while(userInput != "YES" && userInput != "NO");
            variableList[14][1] = userInput;
            globalConclusions[globalConclusionsCounter]=(variableList[14][0] + "=" + variableList[14][1]);
            globalConclusionsCounter++;
        }
    }
    if(postModVar == "Chronic")
    {
        if(variableList[15][1] == "")
        {
            do{
                cout<< "Is it Chronic Leukemia? (YES/NO) ";
                cin >> userInput;
            }while(userInput != "YES" && userInput != "NO");
            variableList[15][1] = userInput;
            globalConclusions[globalConclusionsCounter]=(variableList[15][0] + "=" + variableList[15][1]);
```

```
            globalConclusionsCounter++;
        }
    }
    if(postModVar == "Urgent")
    {
        if(variableList[16][1] == "")
        {
            do{
                cout<< "Is treatment needed urgently? (YES/NO) ";
                cin >> userInput;
            }while(userInput != "YES" && userInput != "NO");
            variableList[16][1] = userInput;
            globalConclusions[globalConclusionsCounter]=(variableList[16][0] + "=" + variableList[16][1]);
            globalConclusionsCounter++;
        }
    }
    if(postModVar == "Lung")
    {
        if(variableList[17][1] == "")
        {
            do{
                cout<< "Patient has lung cancer? (YES/NO) ";
                cin >> userInput;
            }while(userInput != "YES" && userInput != "NO");
            variableList[17][1] = userInput;
            globalConclusions[globalConclusionsCounter]=(variableList[17][0] + "=" + variableList[17][1]);
            globalConclusionsCounter++;
        }
    }
    if(postModVar == "Small")
    {
        if(variableList[18][1] == "")
        {
            do{
                cout<< "Are the tumors small? (YES/NO) ";
                cin >> userInput;
            }while(userInput != "YES" && userInput != "NO");
            variableList[18][1] = userInput;
            globalConclusions[globalConclusionsCounter]=(variableList[18][0] + "=" + variableList[18][1]);
            globalConclusionsCounter++;
        }
    }
    if(postModVar == "Pancreatic")
    {
        if(variableList[19][1] == "")
        {
            do{
                cout<< "Patient has pancreatic cancer? (YES/NO) ";
                cin >> userInput;
            }while(userInput != "YES" && userInput != "NO");
            variableList[19][1] = userInput;
            globalConclusions[globalConclusionsCounter]=(variableList[19][0] + "=" + variableList[19][1]);
```

```
            globalConclusionsCounter++;
        }
    }
    if(postModVar == "Risk")
    {
        if(variableList[20][1] == "")
        {
            do{
                cout<< "Is the patient high risk? (YES/NO) ";
                cin >> userInput;
            }while(userInput != "YES" && userInput != "NO");
            variableList[20][1] = userInput;
            globalConclusions[globalConclusionsCounter]=(variableList[20][0] + "=" + variableList[20][1]);
            globalConclusionsCounter++;
        }
    }
    if(postModVar == "Thyroid")
    {
        if(variableList[21][1] == "")
        {
            do{
                cout<< "Patient has Thyroid Cancer? (YES/NO) ";
                cin >> userInput;
            }while(userInput != "YES" && userInput != "NO");
            variableList[21][1] = userInput;
            globalConclusions[globalConclusionsCounter]=(variableList[21][0] + "=" + variableList[21][1]);
            globalConclusionsCounter++;
        }
    }
    if(postModVar == "Covered")
    {
        if(variableList[22][1] == "")
        {
            do{
                cout<< "Do the tumors cover the entire thyroid? (YES/NO) ";
                cin >> userInput;
            }while(userInput != "YES" && userInput != "NO");
            variableList[22][1] = userInput;
            globalConclusions[globalConclusionsCounter]=(variableList[22][0] + "=" + variableList[22][1]);
            globalConclusionsCounter++;
        }
    }
   }
}
void Treatment::validate_Ri(int ruleNumber)
{
   bool indexValidated = true;
   string variable;
   string value;
   for (int i=0; i <8; i++)
   {
```

```
        string ruleCondition = ruleList[ruleNumber][i];
        bool varPresent = false;
        if (ruleCondition != "")
        {
            std::size_t seperator = ruleCondition.find("=");
            variable = ruleCondition.substr(0, seperator);
            value = ruleCondition.substr(seperator + 1);
            for(int j=0; j<23; j++)
            {
                if(variableList[j][0] == variable)
                {
                    varPresent=true;
                    if (variableList[j][1] != value) {
                        indexValidated = false;
                        break;
                    }
                }
            }
            if (varPresent == false)
            {
                indexValidated = false;
                break;
            }
            if(indexValidated == false)
            {
                break;
            }
        }
    }
    if (indexValidated == true)
    {
        string conclusion = ruleList[ruleNumber][8];
        globalConclusions[globalConclusionsCounter] = conclusion;
        modifyTreatmentString();
        globalConclusionsCounter++;
        printGlobalConclusions();
        return;
    }
}
void Treatment::process(string patientDiagnosis)
{
    variableList[0][1] = patientDiagnosis;
    string searchCriteria = "Cancer=" + patientDiagnosis;
    globalConclusions[globalConclusionsCounter] = searchCriteria;
    globalConclusionsCounter++;
    search_cvl(searchCriteria);
    return;
}
void Treatment::printGlobalConclusions()
{
    cout << endl;
    for(int i = 0; i <= globalConclusionsCounter; i++)
```

```
    {
        if (globalConclusions[i] != "")
        {
            cout << globalConclusions[i] << endl;
            globalConclusions[i] = "";
        }
    }
    cout << endl;
    return;
}
void Treatment::modifyTreatmentString()
{
    string postModTreatment = globalConclusions[globalConclusionsCounter];
    std::size_t seperator = postModTreatment.find("=");
    treatment = postModTreatment.substr(seperator+1);
    return;
}
string Treatment::getTreatment()
{
    cout << endl;
    return treatment;
}
```

V. Sample Runs

a. Sample 1

```
This program will help diagnose cancer and provide a treatment plan.
Here are the possible conclusions:
CONCLUSION 4 Basal Skin Cancer
CONCLUSION 5 Squamous Skin Cancer
CONCLUSION 7 Breast Cancer
CONCLUSION 8 Lung Cancer
CONCLUSION 9 Pancreatic Cancer
CONCLUSION 10 Colon or Rectal Cancer
CONCLUSION 11 Thyroid Cancer
CONCLUSION 12 Leukemia
Please select a cancer type you want to check for by inputting the corresponding
number: 11
You selected: Thyroid Cancer

Do you have a constant cough? (Y/N): Y

Has your voice changed? (Y/N): Y

Do you have breathing or swallowing issues? (Y/N): Y

Do you have neck swelling or a lump? (Y/N): Y

Where are you experiencing pain? (C = Chest, S = Stomach, N = Neck, B = Bones): N

Your final diagnosis is Thyroid Cancer.

Do the tumors cover the entire thyroid? (YES/NO) YES
Is it Centralized? (YES/NO) NO

Cancer=Thyroid
Covered=YES
Centralized=NO
Treatment=Radical Neck Dissection


Diagnosis: Thyroid
Treatment: Radical Neck Dissection

Program ended with exit code: 0
```

b. Sample 2

```
This program will help diagnose cancer and provide a treatment plan.
Here are the possible conclusions:
CONCLUSION 4 Basal Skin Cancer
CONCLUSION 5 Squamous Skin Cancer
CONCLUSION 7 Breast Cancer
CONCLUSION 8 Lung Cancer
CONCLUSION 9 Pancreatic Cancer
CONCLUSION 10 Colon or Rectal Cancer
CONCLUSION 11 Thyroid Cancer
CONCLUSION 12 Leukemia
Please select a cancer type you want to check for by inputting the corresponding
number: 8
You selected: Lung Cancer

Do you experience frequent headaches? (Y/N): Y

Do you have blood clots? (Y/N): Y

Have you had lung infections? (Y/N): Y

Have you coughed up blood? (Y/N): Y

Do you have a cough or breathing issues? (Y/N): Y

Where are you experiencing pain? (C = Chest, S = Stomach, N = Neck, B = Bones): C

Your final diagnosis is Lung Cancer.

Are the tumors small? (YES/NO) NO
Was the cancer caught early? (YES/NO) YES

Cancer=Lung
Small=NO
Early=YES
Treatment=Surgical Removal or Targeted Therapy then Removal

Is it Excisable? (YES/NO) NO

Diagnosis: Lung
Treatment: Surgical Removal or Targeted Therapy then Removal

Program ended with exit code: 0
```

c. Sample 3

```
This program will help diagnose cancer and provide a treatment plan.
Here are the possible conclusions:
CONCLUSION 4 Basal Skin Cancer
CONCLUSION 5 Squamous Skin Cancer
CONCLUSION 7 Breast Cancer
CONCLUSION 8 Lung Cancer
CONCLUSION 9 Pancreatic Cancer
CONCLUSION 10 Colon or Rectal Cancer
CONCLUSION 11 Thyroid Cancer
CONCLUSION 12 Leukemia
Please select a cancer type you want to check for by inputting the corresponding
number: 11
You selected: Thyroid Cancer

Do you have a constant cough? (Y/N): Y

Has your voice changed? (Y/N): Y

Do you have breathing or swallowing issues? (Y/N): Y

Do you have neck swelling or a lump? (Y/N): Y

Where are you experiencing pain? (C = Chest, S = Stomach, N = Neck, B = Bones): N

Your final diagnosis is Thyroid Cancer.

Do the tumors cover the entire thyroid? (YES/NO) NO
Is it Centralized? (YES/NO) YES

Cancer=Thyroid
Covered=NO
Centralized=YES
Treatment=Lobectomy


Diagnosis: Thyroid
Treatment: Lobectomy

Program ended with exit code: 0
```

VI. Program Analysis

a. Speed, user interface, etc.

Because this program is not as complex as a fully released application, our time and space complexities were limited to the hardcoded rule sets. Therefore there are no open loops, or areas that can be overflown with the wrong data. We also put safeguards to only respond to correct user inputs to ensure there is no overload of our program. The interface is entirely common line driven without the application of a GUI, though common english is used basic answers of Y or N or YES or NO are all that is required aside from reading the question.

b. Changes to source code

As for changes to the source code, Dr.Allis C based code took a bit for us to decipher. Once we understood what all the variables were and what their jobs were, we renamed and restructured them. Switching from a single file to .h based implementation we added a layer of security to our source code. Additionally a lot of the C based classes and GOTO calls were changed into C++ data structures and function calls.

VII. Conclusion

In conclusion, this project has taught us how to translate research into an expert system. With the cancer diagnosis and treatments we found online, we were able to successfully build a program that can accurately (to our knowledge) diagnose and treat 8 different kinds of cancer in just a few seconds. This shows the time and inconsistencies we can remove from the field. With more information from people more experienced in the subject we could continue our knowledge base to make the program even more accurate and complex without changing time complexities at all really.

Contributions:

This program was developed by Brittany Hale, John Courtright, and Robert Jones. I developed the forward chaining algorithm. Brittany developed portions of the diagnosis decision

tree, the conversion of rules for backward chaining, and portions of the backward chaining method, notably including Stack manipulation and the object-oriented approach. John developed portions of the diagnosis decision tree, the treatment decision tree, the conversion of rules for forward chaining, and portions of the backward chaining method, including the interface and object-oriented structure.

References

Cancer Statistics. (2024, May 9). Cancer.gov. https://www.cancer.gov/about-cancer/understanding/statistics

Cancer Types | Find Your Cancer Type. (2025). Cancer.org. https://www.cancer.org/cancer/types.html