Project Report: Expert System for Cardiovascular Disease Diagnosis

1. Introduction

Cardiovascular diseases are the primary cause of mortality worldwide, highlighting the need for timely diagnosis and treatment. This project focuses on developing an AI-based expert system designed to diagnose cardiovascular conditions based on patient symptoms and suggest corresponding treatments. The system utilizes both backward and forward chaining techniques to emulate a healthcare professional's diagnostic process.

2. Problem and Domain Overview

Problem Statement

The objective of this project is to build an expert system capable of:

- Diagnosing cardiovascular diseases by analyzing patient-reported symptoms.
- Recommending suitable treatments based on the diagnosis.

3. Decision Tree and Rule-based System

Decision Tree Structure

The system's diagnostic process is structured as a decision tree where each node represents a symptom, and the branches correspond to whether the symptom is present or not. The leaf nodes represent a diagnosis. This structure guides the system's questioning and helps narrow down possible diagnoses.

Rules

The expert system's knowledge base consists of 36 predefined rules that map specific symptom combinations to corresponding diagnoses. For example:

Rule 10:

- Conditions: CHEST_PAIN, RADIATING_PAIN, EXERTION_STRESS, RELIEF_WITH_REST
- Conclusion: Coronary Artery Disease

These rules are stored in a structured format, allowing the system to match patient symptoms to potential diagnoses.

4. Methodologies

Backward Chaining

Backward chaining is used in the diagnostic phase, where the system starts with potential diagnoses and works backward to see if the patient's symptoms match the conditions required for those diagnoses.

Steps:

1. Set a Goal: Aim to diagnose the patient.
2. Identify Relevant Rules: Look for rules that conclude with the goal.
3. Evaluate Conditions: For each rule, ask the patient about the symptoms.
4. Draw a Conclusion: If the rule's conditions are met, the system confirms the diagnosis.

Forward Chaining

Forward chaining is employed for treatment recommendations, starting with the diagnosis and moving forward to infer applicable treatments.

Steps:

1. Start with Known Information: The established diagnosis.
2. Apply Rules: Match the diagnosis to treatment options.
3. Derive Treatments: Recommend treatments based on the diagnosis.

5. System Implementation

Program Overview

The expert system is implemented in C++, with several classes representing key components.

Classes and Functions

- Rule Class:
    - Represents individual rules.
    - Attributes:
        - id: Unique identifier for the rule.
        - conditions: List of required symptoms.
        - conclusion: The diagnosis or treatment derived from the rule.
- KnowledgeBaseBackward Class:
    - Stores all rules used for backward chaining (diagnosis).

    - Components include:
        - rules: Vector of Rule objects.
        - conclusionList: Maps rule IDs to diagnoses.
        - questionVarList: Maps symptoms to user-friendly questions.
- VariableList Class:
    - Manages symptom status based on user input.
    - Attributes include:
        - variableList: Maps symptom variables to boolean values (present or absent).
        - derivedGlobalVariableList: Stores the final diagnosis.
- InferenceEngineBackward Class:
    - Implements backward chaining logic.
    - The Process() function iterates through rules and determines diagnoses based on patient symptoms.
- KnowledgeBaseForward Class:
    - Stores treatment recommendations for each diagnosis.
    - treatmentRecommendations: Maps diagnoses to treatment lists.
- InferenceEngineForward Class:
    - Implements forward chaining for treatment recommendations.
    - Functions include:
        - recommendTreatments(): Determines treatments based on the diagnosis.
        - printConclusions(): Displays the recommended treatments.

Source Code:

```
#include <iostream>
#include <vector>
#include <map>
#include <string>
#include <algorithm>

using namespace std;
```

```
// Structure to represent a rule
struct Rule
{
  int id;
  vector<string> conditions;
  string conclusion;
};

// Class representing the backward knowledge base
class KnowledgeBaseBackward {
public:
  vector<Rule> rules;
  map<int, string> conclusionList;
  map<int, string> clauseVarList;
  map<string, string> questionVarList;

  // Constructor initializes the knowledge base with predefined rules and maps
  KnowledgeBaseBackward() {

    rules =
    {
      {10, {"CHEST_PAIN", "RADIATING_PAIN", "EXERTION_STRESS", "RELIEF_WITH_REST"}, "Coronary Artery
Disease"},
      {20, {"CHEST_PAIN", "RADIATING_PAIN", "EXERTION_STRESS", "NAUSEA_VOMITING"}, "Heart Attack
(Myocardial Infarction)"},
      {30, {"CHEST_PAIN", "RADIATING_PAIN", "EXERTION_STRESS", "DISCOMFORT_OVER_MINUTES"}, "Heart
Attack (Myocardial Infarction)"},
      {40, {"CHEST_PAIN", "RADIATING_PAIN", "EXERTION_STRESS", "CHEST_PAIN_RANDOM"}, "Unstable
Angina"},
      {50, {"CHEST_PAIN", "RADIATING_PAIN", "EXERTION_STRESS"}, "Stable Angina"},
      {60, {"CHEST_PAIN", "CHEST_PAIN_WORSE_LYING_DOWN"}, "Pericarditis"},
      {70, {"CHEST_PAIN", "FLUTTERING_HEARTBEAT", "DIZZINESS_FAINTING"}, "Arrhythmia"},
      {80, {"CHEST_PAIN", "FLUTTERING_HEARTBEAT", "IS_SHORTNESS_BREATH_WITH_EXERTION"},
"Arrhythmia"},
      {90, {"CHEST_PAIN", "FLUTTERING_HEARTBEAT"}, "Benign Palpitations"},
      {100, {"CHEST_PAIN", "SHORTNESS_BREATH_CHEST_TIGHTNESS"}, "Pulmonary Hypertension"},
      {110, {"CHEST_PAIN", "RECENT_FEVER_ILLNESS"}, "Endocarditis"},
      {120, {"CHEST_PAIN", "FEVER_WEIGHT_LOSS", "NIGHT_SWEATS_CHILLS", "NEW_HEART_MURMUR"},
"Endocarditis"},
      {130, {"CHEST_PAIN", "FEVER_WEIGHT_LOSS", "NIGHT_SWEATS_CHILLS", "PAINFUL_SWOLLEN_JOINTS"},
"Rheumatic Heart Disease"},
      {140, {"SWELLING_LEGS_ANKLES", "SHORTNESS_BREATH_LYING_DOWN",
"PRODUCTIVE_COUGH_WHEEZING"}, "Congestive Heart Failure"},
      {150, {"SWELLING_LEGS_ANKLES", "SHORTNESS_BREATH_LYING_DOWN",
"SWOLLEN_LIVER_ABDOMEN"}, "Right-Sided Heart Failure"},
      {160, {"SWELLING_LEGS_ANKLES", "SHORTNESS_BREATH_LYING_DOWN", "HIGH_BLOOD_PRESSURE"},
"Hypertensive Heart Disease"},
      {170, {"SWELLING_LEGS_ANKLES", "SHORTNESS_BREATH_LYING_DOWN",
"SHORTNESS_BREATH_CHEST_PAIN_EXERTION"}, "Congestive Heart Failure"},
      {180, {"SWELLING_LEGS_ANKLES", "SHORTNESS_BREATH_LYING_DOWN", "PERSISTENT_DRY_COUGH"},
"Pulmonary Hypertension"},
      {190, {"SWELLING_LEGS_ANKLES", "SHORTNESS_BREATH_LYING_DOWN"}, "Valvular Heart Disease"},
      {200, {"SWELLING_LEGS_ANKLES", "HISTORY_ALCOHOL_DRUGS"}, "Cardiomyopathy"},
      {210, {"SWELLING_LEGS_ANKLES", "FREQUENT_INFECTIONS_FEVER"}, "Endocarditis"},
      {220, {"SWELLING_LEGS_ANKLES", "RECENT_INFECTION_TRAUMA"}, "Pericarditis"},
      {230, {"SWELLING_LEGS_ANKLES"}, "Congestive Heart Failure"},
      {240, {"LEG_PAIN_WHEN_WALKING", "PAIN_SUBSIDES_AFTER_REST"}, "Peripheral Artery Disease"},
      {250, {"LEG_PAIN_WHEN_WALKING"}, "Aortic Aneurysm"},
      {260, {"NUMBNESS_COLDNESS_LIMBS"}, "Peripheral Artery Disease"},
      {270, {"PAIN_SUDDEN_SEVERE"}, "Aortic Aneurysm"},
      {280, {"NUMBNESS_COLDNESS_LIMBS"}, "Congenital Heart Disease"},
      {290, {"CHEST_PAIN", "FEVER_WEIGHT_LOSS", "SWELLING_LEGS_DIFFICULTY_BREATHING"}, "Congestive
Heart Failure"},
      {300, {"CHEST_PAIN", "SHARP_CHEST_PAIN_DEEP_BREATHS", "RECENT_INFECTION_TRAUMA"},
"Pericarditis"},
      {310, {"CHEST_PAIN", "SHARP_CHEST_PAIN_DEEP_BREATHS", "DIFFICULTY_BREATHING_LYING_DOWN"},
"Pericarditis"},
      {320, {"CHEST_PAIN", "SHARP_CHEST_PAIN_DEEP_BREATHS"}, "Stable Angina"},
      {330, {"CHEST_PAIN", "HIGH_BP_HEADACHES_BLURRED_VISION"}, "Hypertensive Heart Disease"},
      {340, {"CHEST_PAIN", "BLUISH_SKIN_LIPS", "PRESENT_SINCE_BIRTH"}, "Congenital Heart Disease"},
      {350, {"CHEST_PAIN", "BLUISH_SKIN_LIPS", "EXTREME_SHORTNESS_BREATH_FATIGUE_EXERTION"},
"Pulmonary Hypertension"},
      {360, {"CHEST_PAIN"}, "Benign Condition"}
    };

    // Populate conclusionList with rule IDs and their conclusions
    for (const auto& rule : rules) {
      conclusionList[rule.id] = rule.conclusion;
    }

    // Initialize clauseVarList with the provided map
    clauseVarList =
    {
      {1, "CHEST_PAIN"},
      {2, "RADIATING_PAIN"},
      {3, "EXERTION_STRESS"},
      {4, "RELIEF_WITH_REST"},
      {5, "CHEST_PAIN"},
      {6, "RADIATING_PAIN"},
      {7, "EXERTION_STRESS"},
      {8, "NAUSEA_VOMITING"},
      {9, "CHEST_PAIN"},
      {10, "RADIATING_PAIN"},
      {11, "EXERTION_STRESS"},
      {12, "DISCOMFORT_OVER_MINUTES"},
      {13, "CHEST_PAIN"},
      {14, "RADIATING_PAIN"},
      {15, "EXERTION_STRESS"},
      {16, "CHEST_PAIN_RANDOM"},
      {17, "CHEST_PAIN"},
      {18, "RADIATING_PAIN"},
      {19, "EXERTION_STRESS"},
      {21, "CHEST_PAIN"},
      {22, "CHEST_PAIN_WORSE_LYING_DOWN"},
      {25, "CHEST_PAIN"},
      {26, "FLUTTERING_HEARTBEAT"},
      {27, "DIZZINESS_FAINTING"},
      {29, "CHEST_PAIN"},
      {30, "FLUTTERING_HEARTBEAT"},
      {31, "SHORTNESS_BREATH_WITH_EXERTION"},
      {33, "CHEST_PAIN"},
      {34, "FLUTTERING_HEARTBEAT"},
      {37, "CHEST_PAIN"},
      {38, "SHORTNESS_BREATH_CHEST_TIGHTNESS"},
      {41, "CHEST_PAIN"},
      {42, "RECENT_FEVER_ILLNESS"},
      {45, "CHEST_PAIN"},
      {46, "FEVER_WEIGHT_LOSS"},
      {47, "NIGHT_SWEATS_CHILLS"},
      {48, "NEW_HEART_MURMUR"},
      {49, "CHEST_PAIN"},
      {50, "FEVER_WEIGHT_LOSS"},
      {51, "NIGHT_SWEATS_CHILLS"},
      {52, "PAINFUL_SWOLLEN_JOINTS"},
      {53, "SWELLING_LEGS_ANKLES"},
      {54, "SHORTNESS_BREATH_LYING_DOWN"},
      {55, "PRODUCTIVE_COUGH_WHEEZING"},
      {57, "SWELLING_LEGS_ANKLES"},
      {58, "SHORTNESS_BREATH_LYING_DOWN"},
      {59, "SWOLLEN_LIVER_ABDOMEN"},
      {61, "SWELLING_LEGS_ANKLES"},
      {62, "SHORTNESS_BREATH_LYING_DOWN"},
      {63, "HIGH_BLOOD_PRESSURE"},
      {65, "SWELLING_LEGS_ANKLES"},
      {66, "SHORTNESS_BREATH_LYING_DOWN"},
      {67, "SHORTNESS_BREATH_CHEST_PAIN_EXERTION"},
      {69, "SWELLING_LEGS_ANKLES"},
      {70, "SHORTNESS_BREATH_LYING_DOWN"},
      {71, "PERSISTENT_DRY_COUGH"},
      {73, "SWELLING_LEGS_ANKLES"},
      {74, "SHORTNESS_BREATH_LYING_DOWN"},
      {77, "SWELLING_LEGS_ANKLES"},
      {78, "HISTORY_ALCOHOL_DRUGS"},
      {81, "SWELLING_LEGS_ANKLES"},
      {82, "FREQUENT_INFECTIONS_FEVER"},
      {85, "SWELLING_LEGS_ANKLES"},
      {86, "RECENT_INFECTION_TRAUMA"},
      {89, "SWELLING_LEGS_ANKLES"},
      {93, "LEG_PAIN_WHEN_WALKING"},
      {94, "PAIN_SUBSIDES_AFTER_REST"},
      {97, "LEG_PAIN_WHEN_WALKING"},
      {101, "NUMBNESS_COLDNESS_LIMBS"},
      {105, "PAIN_SUDDEN_SEVERE"},
      {109, "NUMBNESS_COLDNESS_LIMBS"},
      {113, "CHEST_PAIN"},
      {114, "FEVER_WEIGHT_LOSS"},
      {115, "SWELLING_LEGS_DIFFICULTY_BREATHING"},
      {117, "CHEST_PAIN"},
      {118, "SHARP_CHEST_PAIN_DEEP_BREATHS"},
        {119, "RECENT_INFECTION_TRAUMA"},
        {121, "CHEST_PAIN"},
        {122, "SHARP_CHEST_PAIN_DEEP_BREATHS"},
        {123, "DIFFICULTY_BREATHING_LYING_DOWN"},
        {125, "CHEST_PAIN"},
        {126, "SHARP_CHEST_PAIN_DEEP_BREATHS"},
        {129, "CHEST_PAIN"},
        {130, "HIGH_BP_HEADACHES_BLURRED_VISION"},
        {133, "CHEST_PAIN"},
        {134, "BLUISH_SKIN_LIPS"},
        {135, "PRESENT_SINCE_BIRTH"},
        {137, "CHEST_PAIN"},
        {138, "BLUISH_SKIN_LIPS"},
        {139, "EXTREME_SHORTNESS_BREATH_FATIGUE_EXERTION"},
        {141, "CHEST_PAIN"}
    };

    // Initialize questionVarList with condition variables and corresponding questions
    questionVarList =
    {
        {"CHEST_PAIN", "Chest pain or discomfort?"},
        {"RADIATING_PAIN", "Pain radiating to arms, neck, jaw, or back?"},
        {"SWELLING_LEGS_ANKLES", "Swelling in the legs or ankles?"},
        {"EXERTION_STRESS", "Occurs after exertion or stress?"},
        {"CHEST_PAIN_WORSE_LYING_DOWN", "Does chest pain worsen when lying down?"},
        {"SHORTNESS_BREATH_LYING_DOWN", "Do you have shortness of breath while lying down?"},
        {"LEG_PAIN_WHEN_WALKING", "Leg pain or cramping while walking?"},
        {"RELIEF_WITH_REST", "Relief with rest or nitroglycerin?"},
        {"NAUSEA_VOMITING", "Nausea, vomiting, sweating, shortness of breath?"},
        {"DISCOMFORT_OVER_MINUTES", "Discomfort lasts over ___ minutes?"},
        {"CHEST_PAIN_RANDOM", "Chest pain occurs randomly?"},
        {"DIZZINESS_FAINTING", "Do you experience dizziness or fainting?"},
        {"FLUTTERING_HEARTBEAT", "Do you feel a fluttering or irregular heartbeat?"},
        {"SHORTNESS_BREATH_CHEST_TIGHTNESS", "Shortness of breath with chest tightness?"},
        {"IS_SHORTNESS_BREATH_WITH_EXERTION", "Is there shortness of breath with exertion?"},
        {"RECENT_FEVER_ILLNESS", "Recent fever or illness?"},
        {"FEVER_WEIGHT_LOSS", "Fever with unexplained weight loss?"},
        {"NIGHT_SWEATS_CHILLS", "Do you experience night sweats or chills?"},
        {"NEW_HEART_MURMUR", "Do you have a new or worsening heart murmur?"},
        {"PAINFUL_SWOLLEN_JOINTS", "Do you have painful, swollen joints?"},
        {"SWELLING_LEGS_DIFFICULTY_BREATHING", "Do you have swelling in your legs and difficulty breathing?"},
        {"SHARP_CHEST_PAIN_DEEP_BREATHS", "Do you experience sharp, stabbing chest pain that worsens with
deep breaths?"},
        {"RECENT_INFECTION_TRAUMA", "Was there a recent infection or trauma?"},
        {"HIGH_BP_HEADACHES_BLURRED_VISION", "High blood pressure with severe headaches or blurred
vision?"},
        {"BLUISH_SKIN_LIPS", "Bluish skin or lips (cyanosis)?"},
        {"PRESENT_SINCE_BIRTH", "Has this been present since birth?"},
        {"EXTREME_SHORTNESS_BREATH_FATIGUE_EXERTION", "Do you experience extreme shortness of breath or
fatigue with exertion?"},
        {"HIGH_BLOOD_PRESSURE", "Do you have high blood pressure?"},
        {"SHORTNESS_BREATH_CHEST_PAIN_EXERTION", "Do you have shortness of breath and chest pain with
exertion?"},
        {"PERSISTENT_DRY_COUGH", "Do you have a persistent dry cough?"},
        {"PRODUCTIVE_COUGH_WHEEZING", "Is there a productive cough or wheezing?"},
        {"SWOLLEN_LIVER_ABDOMEN", "Do you have swollen liver or abdomen?"},
        {"HISTORY_ALCOHOL_DRUGS", "Do you have a history of alcohol abuse or drug use?"},
        {"FREQUENT_INFECTIONS_FEVER", "Do you have frequent infections or fever?"},
        {"PAIN_SUBSIDES_AFTER_REST", "Does the pain subside after rest?"},
        {"NUMBNESS_COLDNESS_LIMBS", "Do you experience numbness or coldness in your limbs?"},
        {"PAIN_SUDDEN_SEVERE", "Does the pain or numbness occur suddenly with severe intensity?"},
        {"DIFFICULTY_BREATHING_LYING_DOWN", "Difficulty breathing when lying down?"}
    };
}
};

// Class representing the list of variables
class VariableList {
public:
map<string, bool> variableList;
map<string, bool> derivedGlobalVariableList;

// Retrieves the status of a given variable
bool getStatus(const string& variable) const {
    auto it = variableList.find(variable);
    if (it != variableList.end()) {
        return it->second;
    }
    return false;
}

// Sets the status of a given variable
void setStatus(const string& variable, bool status) {
    variableList[variable] = status;
}

// Adds a variable to the list of derived global variables
void addDerived(const string& variable) {
    derivedGlobalVariableList[variable] = true;
}

// Checks if a variable is a derived global variable
bool isDerived(const string& variable) const {
    return derivedGlobalVariableList.find(variable) != derivedGlobalVariableList.end();
}
};

// Class representing the backward inference engine
class InferenceEngineBackward {
private:
KnowledgeBaseBackward* kb;
VariableList* vl;

public:
// Constructor initializes the inference engine with references to the knowledge base and variable list
InferenceEngineBackward(KnowledgeBaseBackward* kb_ptr, VariableList* vl_ptr) {
```

```
    kb = kb_ptr;
    vl = vl_ptr;
}

// Function 1: search_con(string variable)
// Finds the rule ID (Ri) that concludes the given variable
int search_con(const string& variable) const {
    for (const auto& [rule_id, conclusion] : kb->conclusionList) {
        if (conclusion == variable) {
            return rule_id;
        }
    }
    return -1; // Indicates not found
}

// Function 2: rule_to_clause(int Ri)
// Converts Rule Number Ri to Clause Number Ci based on the provided formula
int rule_to_clause(int Ri) const {
    // Check if rule IDs are sequenced like 1,2,3,... or 10,20,30,...
    if (Ri % 10 == 0) { // Assuming rules like 10, 20, 30, ...
        return 4 * (Ri / 10 - 1) + 1;
    } else { // Assuming rules like 1, 2, 3, ...
        return 4 * (Ri - 1) + 1;
    }
}

// Function 3: update_VL(int Ci)
// Instantiates variables starting from Clause Number Ci, handling up to 4 variables
void update_VL(int Ci) {
    const int max_slots = 4;
    for (int i = Ci; i < Ci + max_slots; ++i) {
        if (kb->clauseVarList.find(i) != kb->clauseVarList.end()) {
            const string& variable = kb->clauseVarList[i];
            // If variable is not already in variableList, process it
            if (vl->variableList.find(variable) == vl->variableList.end()) {
                // Recursive call to Process for the variable
                string dummy_diagnosis;
                int dummy_ruleNumber;
                Process(variable, dummy_diagnosis, dummy_ruleNumber);
            }
        }
    }
}

// Function 4: validate_Ri(int Ri, string &conclusion)
// Validates if all conditions of Rule Ri are satisfied
bool validate_Ri(int Ri, string& conclusion) {
    // Find the rule with ID Ri
    auto it = find_if(kb->rules.begin(), kb->rules.end(), [Ri](const Rule& rule) {
        return rule.id == Ri;
    });

    if (it == kb->rules.end()) {
        return false; // Rule not found
    }

    const Rule& rule = *it;
    bool all_conditions_met = true;

    for (const auto& condition : rule.conditions) {
        if (!vl->getStatus(condition)) {
            all_conditions_met = false;
            break;
        }
    }

    if (all_conditions_met) {
        conclusion = rule.conclusion;
        vl->addDerived(conclusion);
        return true;
    }

    return false;
}

// Processes the goal by evaluating rules and determining the diagnosis
bool Process(string goal, string &diagnosis, int &usedRuleNumber) {
    // If the goal is a variable, search for the corresponding rule
    if (kb->conclusionList.find(search_con(goal)) != kb->conclusionList.end()) {
        int Ri = search_con(goal);
        if (Ri == -1) {
            return false; // Conclusion not found
        }

        // Convert Rule Number Ri to Clause Number Ci
        int Ci = rule_to_clause(Ri);

        // Update Variable List starting from Ci
        update_VL(Ci);

        // Validate Rule Ri
        if (validate_Ri(Ri, diagnosis)) {
            usedRuleNumber = Ri;
            return true;
        }
    }

    // If the goal is not a specific variable, perform backward chaining
    // Sort rules based on their IDs
    vector<Rule> sortedRules = kb->rules;
    sort(sortedRules.begin(), sortedRules.end(), [&](const Rule &a, const Rule &b) -> bool {
        return a.id < b.id;
    });

    // Iterate through each rule to check if conditions are satisfied
    for (const auto& rule : sortedRules) {
            bool ruleSatisfied = true;

            for (const auto& condition : rule.conditions) {
                // If condition status is already known, verify it
                if (vl->variableList.find(condition) != vl->variableList.end()) {
                    if (!vl->variableList[condition]) {
                        ruleSatisfied = false;
                        break;
                    }
                    continue;
                }

                // If condition requires user input, ask the corresponding question
                if (kb->questionVarList.find(condition) != kb->questionVarList.end()) {
                    string question = kb->questionVarList[condition];
                    cout << question << " (1 for Yes / 0 for No): ";
                    int response;
                    while (true) {
                        cin >> response;
                        if (response == 1 || response == 0) {
                            break;
                        } else {
                            cout << "Please enter 1 for Yes or 0 for No: ";
                        }
                    }
                    vl->setStatus(condition, response == 1);
                    if (!vl->variableList[condition]) {
                        ruleSatisfied = false;
                        break;
                    }
                } else {
                    // If condition is not recognized, set it to false
                    vl->setStatus(condition, false);
                    ruleSatisfied = false;
                    break;
                }
            }

            // If all conditions are satisfied, set the diagnosis
            if (ruleSatisfied) {
                diagnosis = rule.conclusion;
                usedRuleNumber = rule.id;
                vl->addDerived(rule.conclusion);
                return true;
            }
        }

        return false;
    }
};

// Class representing the forward knowledge base for treatments
class KnowledgeBaseForward {
```

```
public:
  map<string, vector<string>> treatmentRecommendations;

  // Constructor initializes treatment recommendations for each diagnosis
  KnowledgeBaseForward() {
    treatmentRecommendations = {
      {"Coronary Artery Disease", {"Lifestyle changes", "Medications (e.g., aspirin, beta-blockers)", "Angioplasty or
surgery"}},
      {"Heart Attack (Myocardial Infarction)", {"Immediate medical attention", "Medications (e.g., thrombolytics)",
"Surgical procedures (e.g., bypass surgery)"}},
      {"Unstable Angina", {"Medications (e.g., nitrates, anticoagulants)", "Lifestyle modifications", "Surgical
interventions if necessary"}},
      {"Stable Angina", {"Lifestyle changes", "Medications (e.g., nitrates, beta-blockers)", "Possible angioplasty"}},
      {"Pericarditis", {"Anti-inflammatory medications", "Rest", "Treatment of underlying cause"}},
      {"Arrhythmia", {"Medications to control heart rate", "Pacemaker or defibrillator", "Lifestyle modifications"}},
      {"Benign Palpitations", {"Reassurance", "Lifestyle changes if necessary"}},
      {"Pulmonary Hypertension", {"Medications to dilate blood vessels", "Oxygen therapy", "Lifestyle changes"}},
      {"Endocarditis", {"Antibiotics", "Surgery in severe cases"}},
      {"Rheumatic Heart Disease", {"Antibiotics to treat strep infection", "Medications to manage symptoms"}},
      {"Congestive Heart Failure", {"Medications (e.g., diuretics, ACE inhibitors)", "Lifestyle changes", "Surgical
options"}},
      {"Right-Sided Heart Failure", {"Treat underlying cause", "Medications to reduce fluid buildup"}},
      {"Hypertensive Heart Disease", {"Antihypertensive medications", "Lifestyle modifications"}},
      {"Valvular Heart Disease", {"Medications to manage symptoms", "Surgical repair or replacement of valves"}},
      {"Cardiomyopathy", {"Medications to manage symptoms", "Lifestyle changes", "Surgical interventions if
necessary"}},
      {"Peripheral Artery Disease", {"Lifestyle changes", "Medications to manage symptoms", "Surgical procedures"}},
      {"Aortic Aneurysm", {"Regular monitoring", "Surgical repair if necessary"}},
      {"Congenital Heart Disease", {"Surgical interventions", "Medications", "Lifestyle adjustments"}},
      {"Benign Condition", {"Reassurance and regular monitoring"}}
    };
  }
};

// Class representing the forward inference engine for treatments
class InferenceEngineForward {
private:
  KnowledgeBaseForward* kb_forward;
  VariableList* vl_forward;
  vector<string> treatments;

public:
  // Constructor initializes the forward inference engine with references to the knowledge base and variable list
  InferenceEngineForward(KnowledgeBaseForward* kb_ptr, VariableList* vl_ptr) {
    kb_forward = kb_ptr;
    vl_forward = vl_ptr;
  }

  // Recommends treatments based on the provided diagnosis
  void recommendTreatments(const string& diagnosis) {
    if (kb_forward->treatmentRecommendations.find(diagnosis) != kb_forward->treatmentRecommendations.end())
{
      treatments = kb_forward->treatmentRecommendations[diagnosis];
```

```
        } else {
            treatments.push_back("No treatment recommendations available.");
        }
    }

    // Prints the recommended treatments to the console
    void printConclusions() const {
        if (!treatments.empty()) {
            cout << "Recommended Treatments:\n";
            for (const auto& treatment : treatments) {
                cout << "- " << treatment << "\n";
            }
        } else {
            cout << "No treatment recommendations available.\n";
        }
    }
};

int main(){

    // Initialize backward and forward knowledge bases
    KnowledgeBaseBackward kb_backward;
    KnowledgeBaseForward kb_forward;

    // Initialize variable lists for backward and forward engines
    VariableList varList_backward;
    VariableList varList_forward;

    // Initialize inference engines with references to their respective knowledge bases and variable lists
    InferenceEngineBackward ie_backward(&kb_backward, &varList_backward);
    InferenceEngineForward ie_forward(&kb_forward, &varList_forward);

    // Display program header
    cout << "------------------------------------------------------\n";
    cout << "=== Cardiovascular Diseases Diagnosis Expert System ===\n";
    cout << "------------------------------------------------------\n\n";

    cout << "=== Diseases Diagnosis (Backwards) ===\n";
    cout << "Determining diagnosis - Please answer the following symptoms:\n\n";

    // Variables to store diagnosis and the rule number used
    string diagnosis;
    int usedRuleNumber = -1;

    // Start the inference process with the goal "DIAGNOSIS"
    bool found = ie_backward.Process("DIAGNOSIS", diagnosis, usedRuleNumber);

    if(found && !diagnosis.empty()){
        cout << "\nDiagnosis: " << diagnosis << " (Rule " << usedRuleNumber << ")\n";
    }
    else{
        cout << "\nGoal cannot be determined.\n";
        return 0;
```

```
  }

  cout << "\n=== Treatment Recommendations (Forwards) ===\n";

  // Get treatment recommendations based on the diagnosis
  ie_forward.recommendTreatments(diagnosis);
  ie_forward.printConclusions();

  // Prompt for exiting the program
  cout << "\nPress Enter to exit the program...";
  cin.ignore(); // To ignore the newline character left in the buffer
  cin.get();
  return 0;

}
```

Program Workflow

1. Initialization: Load rules and initialize knowledge bases.
2. Diagnosis Phase (Backward Chaining):
   - Start with the goal of diagnosing.
   - Ask the patient about their symptoms based on the rules.
   - Conclude a diagnosis if rule conditions are met.
3. Treatment Recommendation Phase (Forward Chaining):
   - Based on the diagnosis, recommend appropriate treatments.
   - Display the treatment recommendations to the user.

6. Program Enhancements

Rule Sorting

- Change: Introduced sorting of rules based on their IDs.
- Benefit: Ensures a more logical and consistent evaluation order, which can improve diagnosis efficiency.

Enhanced Input Validation

- Change: Implemented input validation to manage invalid user entries.
- Benefit: Prevents crashes due to incorrect input and enhances overall user experience.

Optimized Variable Management

- Change: Improved variable management to avoid asking the same symptom-related questions multiple times.
- Benefit: Reduces redundancy, resulting in a faster diagnosis process.

7. Example Program Runs

Patient 1:

- Symptoms: Chest pain, radiating pain, occurs after exertion, relief with rest.
- Diagnosis: Coronary Artery Disease.
- Treatment: Lifestyle changes, medications (aspirin, beta-blockers), surgery or angioplasty.

Patient 2:

- Symptoms: Chest pain, radiating pain, exertion-induced pain, nausea.
- Diagnosis: Heart Attack (Myocardial Infarction).
- Treatment: Immediate medical intervention, thrombolytics, bypass surgery.

Patient 3:

- Symptoms: Swelling in legs, shortness of breath lying down, productive cough.
- Diagnosis: Congestive Heart Failure.
- Treatment: Medications, lifestyle changes, surgery.

Patient 4:

- Symptoms: Chest pain, worsened by deep breaths, recent infection or trauma.
- Diagnosis: Pericarditis.
- Treatment: Anti-inflammatory drugs, rest, underlying cause treatment.

Patient 5:

- Symptoms: Leg pain during walking, pain relief with rest.
- Diagnosis: Peripheral Artery Disease.
- Treatment: Lifestyle changes, medications, surgical intervention.

8. Analysis of Results and System Performance

Diagnostic Accuracy

The expert system provided accurate diagnoses for all five test cases, demonstrating the robustness of the backward chaining approach.

Memory Usage

- Memory Assessment: Utilizes efficient data structures (vectors, maps) for rule and symptom management.
- Optimization: Minimal memory footprint, as the dataset is not large.

Speed and Efficiency

- Execution Speed: The program responds quickly to user input.
- Optimizations:
    - Sorting rules accelerates processing.
    - Efficient variable management reduces redundant questions and speeds up diagnosis.

Impact of Enhancements

- Rule Sorting: Improved logical flow of diagnosis and potentially reduced rule evaluations.
- Input Validation: Improved user interaction without performance trade-offs.
- Variable Management Optimization: Faster diagnostic process due to fewer repetitive queries.

9. Contributions

Samuel Poulis:
Researched treatments
Created Forwards Chaining Tree
Variables
Backward Chaining Rules
Forwards Chaining Rules
Clause Variable List
Search_con
Process
Map questionVarList
InferenceEngineBackward
InferenceEngineForwards

Umer Seliya:

Researched Symptoms
Created Backwards Chaining Tree
Backward Chaining Rules
Conclusion list
Forwards Chaining Rules
Clause Variable List
update_VL
Process
Rules vector
InferenceEngineBackward I
nferenceEngineForwards

Alex Hakimzadeh:
Researched conditions
Created Backwards Chaining Tree
Backward Chaining Rules
Variable List
Forwards Chaining Rules
Fixed Clause Variable List
Rule_to_clause
update_VL
Process Map
conclusionList
InferenceEngineBackward