|  |  |
| --- | --- |
| A logo of a triangle  Description automatically generated | **МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ** |
|  |
| **ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ «ЛИПЕЦКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»** |
|  |

|  |  |
| --- | --- |
| Институт | компьютерных наук |
| Кафедра | автоматизированных систем управления |

Лабораторная работа №1

По дисциплине: «Системны искусственного интеллекта»

Создание экспертной системы с использованием прямой цепочки рассуждений

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Студент | ПИ-22-1 |  |  |  | Баженов Г.С. |
|  | группа |  | подпись, дата |  | фамилия, инициалы |
|  |  |  |  |  |  |
| Руководитель |  |  |  |  |  |
| преподаватель | |  |  |  | Первушин О.С. |
| ученая степень, ученое звание | |  | подпись, дата |  | фамилия, инициалы |

Липецк 2025

**Задание кафедры:**

1. Разработайте оболочку экспертной системы, реализующую прямую цепочку рассуждений, на любом языке программирования.

2. Правила должны формироваться в следующем виде:

ЕСЛИ Объект1=Значение1 ТО Объект2=Значение2

3. Допускается использование логической связки "И".

Например,

ЕСЛИ объект1=значение1 И объект2=значение2 ТО объект3=значение3

4. База правил должна быть отделена от программного модуля и представлять собой отдельный текстовый файл.

5. Должна быть реализована возможность редактирования базы правил (изменения, дополнения, удаления правил).

6. В программе должны быть предусмотрены средства для отображения состояния рабочей базы данных.

7. Задайте стартовую ситуацию – начальное состояние рабочей базы данных.

8. Если ни одно из правил не может сработать, программа должна запрашивать у пользователя новые сведения о некотором объекте. Если таких сведений нет, программа завершает работу.

9. Выберите предметную область, которая знакома вам (разработка программного обеспечения, образовательный процесс, игра и т.д.). Составьте базу правил для этой области. Протестируйте работу вашей программы на этой базе правил.

Экспертная система на основе данной стартовой ситуации и своей базы правил должна сделать определенный вывод и отобразить его на экране.

**Ход работы.**

Задача разрабатываемой экспертной системы – диагностика неисправностей компьютера.

В ходе выполнения работы были разработаны следующие правила:  
ЕСЛИ компьютер\_включается=нет И индикатор\_питания=нет ТО проблема=блок\_питания

ЕСЛИ компьютер\_включается=нет И индикатор\_питания=мигает ТО проблема=проблемы\_с\_электросетью

ЕСЛИ компьютер\_включается=нет И индикатор\_питания=да И звуковые\_сигналы\_bios=нет\_сигналов ТО проблема=материнская\_плата

ЕСЛИ компьютер\_включается=нет И индикатор\_питания=да И звуковые\_сигналы\_bios=непрерывный ТО проблема=блок\_питания

ЕСЛИ компьютер\_включается=нет И индикатор\_питания=да И звуковые\_сигналы\_bios=один\_длинный ТО проблема=оперативная\_память

ЕСЛИ компьютер\_включается=нет И индикатор\_питания=да И звуковые\_сигналы\_bios=один\_длинный\_один\_короткий ТО проблема=материнская\_плата

ЕСЛИ компьютер\_включается=нет И индикатор\_питания=да И звуковые\_сигналы\_bios=один\_длинный\_два\_коротких ТО проблема=видеокарта

ЕСЛИ компьютер\_включается=нет И индикатор\_питания=да И звуковые\_сигналы\_bios=один\_длинный\_три\_коротких ТО проблема=видеопамять

ЕСЛИ компьютер\_включается=нет И индикатор\_питания=да И звуковые\_сигналы\_bios=один\_длинный\_девять\_коротких ТО проблема=bios\_память

ЕСЛИ компьютер\_включается=нет И индикатор\_питания=да И звуковые\_сигналы\_bios=пять\_коротких ТО проблема=процессор

ЕСЛИ компьютер\_включается=нет И индикатор\_питания=да И звуковые\_сигналы\_bios=чередующиеся\_высокие\_низкие ТО проблема=перегрев\_процессора

ЕСЛИ компьютер\_включается=да И изображение\_на\_мониторе=нет И монитор\_включен=да ТО проблема=видеокарта

ЕСЛИ компьютер\_включается=да И изображение\_на\_мониторе=нет И монитор\_включен=нет ТО проблема=монитор

ЕСЛИ компьютер\_включается=да И изображение\_на\_мониторе=искаженное ТО проблема=видеокарта\_или\_драйверы

ЕСЛИ компьютер\_включается=да И изображение\_на\_мониторе=мерцающее ТО проблема=монитор\_или\_кабель

ЕСЛИ компьютер\_включается=да И система=медленная И диск\_загружен=100 ТО проблема=жесткий\_диск

ЕСЛИ компьютер\_включается=да И система=медленная И память\_загружена=100 ТО проблема=недостаточно\_оперативной\_памяти

ЕСЛИ компьютер\_включается=да И система=медленная И процессор\_загружен=100 ТО проблема=фоновые\_процессы\_или\_вирусы

ЕСЛИ компьютер\_включается=да И система=перегревается И вентиляторы=шумные ТО проблема=система\_охлаждения

ЕСЛИ компьютер\_включается=да И система=перегревается И вентиляторы=тихие ТО проблема=пыль\_или\_термопаста

ЕСЛИ компьютер\_включается=да И система=самовыключается ТО проблема=перегрев

ЕСЛИ компьютер\_включается=да И клавиатура=не\_работает И мышь=не\_работает ТО проблема=usb\_контроллер

ЕСЛИ компьютер\_включается=да И клавиатура=не\_работает И мышь=работает ТО проблема=клавиатура

ЕСЛИ компьютер\_включается=да И звук=отсутствует И наушники=работают ТО проблема=динамики

ЕСЛИ компьютер\_включается=да И звук=отсутствует И наушники=не\_работают ТО проблема=звуковая\_карта

ЕСЛИ компьютер\_включается=да И интернет=не\_работает И другие\_устройства=работают ТО проблема=сетевая\_карта

ЕСЛИ компьютер\_включается=да И интернет=не\_работает И другие\_устройства=не\_работают ТО проблема=роутер\_или\_провайдер

ЕСЛИ компьютер\_включается=да И интернет=медленный И локальная\_сеть=быстрая ТО проблема=провайдер

ЕСЛИ компьютер\_включается=да И интернет=медленный И локальная\_сеть=медленная ТО проблема=сетевое\_оборудование

Код разработанной программы:

import re

class ExpertSystem:

    def \_\_init\_\_(self, rule\_file):

        self.rule\_file = rule\_file

        self.rules = []

        self.working\_memory = {}

        self.variable\_options = {}

        self.skipped\_vars = set()

        self.load\_rules()

        self.extract\_variable\_options()

    def load\_rules(self):

        self.rules = []

        try:

            with open(self.rule\_file, 'r', encoding='utf-8') as f:

                for line in f:

                    line = line.strip()

                    if line and not line.startswith('#'):

                        self.parse\_rule(line)

        except FileNotFoundError:

            print(f"Файл правил {self.rule\_file} не найден. Создан новый файл.")

            open(self.rule\_file, 'w').close()

    def extract\_variable\_options(self):

        self.variable\_options = {}

        for rule in self.rules:

            for condition\_var, condition\_val in rule['conditions']:

                if condition\_var not in self.variable\_options:

                    self.variable\_options[condition\_var] = set()

                self.variable\_options[condition\_var].add(condition\_val)

        if 'изображение\_на\_мониторе' in self.variable\_options:

            self.variable\_options['изображение\_на\_мониторе'].add('да')

        for var in self.variable\_options:

            self.variable\_options[var] = sorted(list(self.variable\_options[var]))

    def parse\_rule(self, line):

        pattern = r'ЕСЛИ\s+(.+)\s+ТО\s+(.+)'

        match = re.match(pattern, line)

        if match:

            conditions\_str = match.group(1)

            conclusion\_str = match.group(2)

            conditions = []

            for condition in conditions\_str.split(' И '):

                parts = condition.split('=')

                if len(parts) == 2:

                    conditions.append((parts[0].strip(), parts[1].strip()))

            parts = conclusion\_str.split('=')

            if len(parts) == 2:

                conclusion = (parts[0].strip(), parts[1].strip())

                self.rules.append({'conditions': conditions, 'conclusion': conclusion})

            else:

                print(f"Ошибка в формате заключения: {line}")

        else:

            print(f"Ошибка в формате правила: {line}")

    def save\_rules(self):

        with open(self.rule\_file, 'w', encoding='utf-8') as f:

            for rule in self.rules:

                conditions = ' И '.join([f"{var}={val}" for var, val in rule['conditions']])

                conclusion = f"{rule['conclusion'][0]}={rule['conclusion'][1]}"

                f.write(f"ЕСЛИ {conditions} ТО {conclusion}\n")

        self.extract\_variable\_options()

    def add\_rule(self):

        print("\nДобавление нового правила")

        print("Формат: ЕСЛИ условие1=значение1 И условие2=значение2 ТО вывод=значение")

        conditions = []

        while True:

            condition\_var = input("Введите переменную условия (или Enter для завершения): ").strip()

            if not condition\_var:

                break

            if condition\_var in self.variable\_options:

                print(f"Возможные значения для {condition\_var}: {', '.join(self.variable\_options[condition\_var])}")

            condition\_val = input("Введите значение условия: ").strip()

            conditions.append(f"{condition\_var}={condition\_val}")

        if not conditions:

            print("Необходимо указать хотя бы одно условие!")

            return

        conclusion\_var = input("Введите переменную вывода: ").strip()

        conclusion\_val = input("Введите значение вывода: ").strip()

        conditions\_str = " И ".join(conditions)

        rule\_text = f"ЕСЛИ {conditions\_str} ТО {conclusion\_var}={conclusion\_val}"

        self.parse\_rule(rule\_text)

        self.save\_rules()

        print("Правило успешно добавлено!")

    def delete\_rule(self):

        self.show\_rules()

        try:

            rule\_num = int(input("Введите номер правила для удаления: "))

            if 1 <= rule\_num <= len(self.rules):

                del self.rules[rule\_num-1]

                self.save\_rules()

                print("Правило успешно удалено!")

            else:

                print("Неверный номер правила!")

        except ValueError:

            print("Ошибка: введите число!")

    def show\_rules(self):

        print("\nБаза правил:")

        if not self.rules:

            print("Правил нет.")

            return

        for i, rule in enumerate(self.rules, 1):

            conditions = ' И '.join([f"{var}={val}" for var, val in rule['conditions']])

            conclusion = f"{rule['conclusion'][0]}={rule['conclusion'][1]}"

            print(f"{i}. ЕСЛИ {conditions} ТО {conclusion}")

    def show\_facts(self):

        print("\nТекущие факты:")

        if not self.working\_memory:

            print("Фактов нет.")

            return

        for var, val in self.working\_memory.items():

            print(f"{var} = {val}")

    def add\_fact(self):

        var = input("Введите имя переменной: ").strip()

        if var in self.variable\_options:

            print(f"Возможные значения для {var}:")

            for i, val in enumerate(self.variable\_options[var], 1):

                print(f"{i}. {val}")

            try:

                choice = int(input("Выберите значение (введите номер): "))

                if 1 <= choice <= len(self.variable\_options[var]):

                    val = self.variable\_options[var][choice-1]

                    self.working\_memory[var] = val

                    print(f"Добавлен факт: {var} = {val}")

                else:

                    print("Неверный выбор!")

            except ValueError:

                print("Ошибка: введите число!")

        else:

            val = input("Введите значение переменной: ").strip()

            self.working\_memory[var] = val

            print(f"Добавлен факт: {var} = {val}")

    def logic(self):

        added = True

        iteration = 0

        max\_iterations = 20

        while added and iteration < max\_iterations:

            iteration += 1

            added = False

            for rule in self.rules:

                rule\_applicable, missing\_var = self.check\_rule\_with\_missing(rule)

                if rule\_applicable:

                    conclusion\_var, conclusion\_val = rule['conclusion']

                    if conclusion\_var in self.working\_memory:

                        if self.working\_memory[conclusion\_var] != conclusion\_val:

                            print(f"Конфликт: {conclusion\_var} уже имеет значение {self.working\_memory[conclusion\_var]}, но правило пытается установить {conclusion\_val}")

                            continue

                    if conclusion\_var not in self.working\_memory:

                        self.working\_memory[conclusion\_var] = conclusion\_val

                        print(f"Выведен новый факт: {conclusion\_var} = {conclusion\_val}")

                        added = True

                        if conclusion\_var == 'проблема':

                            return True

                        break

                elif missing\_var and missing\_var not in self.skipped\_vars:

                    if self.query\_missing\_fact(missing\_var):

                        added = True

                        if missing\_var == 'изображение\_на\_мониторе' and self.working\_memory.get(missing\_var) == 'да':

                            self.skipped\_vars.add(missing\_var)

                        if 'проблема' in self.working\_memory:

                            return True

                        break

            if not added:

                if not self.query\_any\_missing\_fact():

                    break

        return False

    def check\_rule(self, rule):

        for condition\_var, condition\_val in rule['conditions']:

            if condition\_var not in self.working\_memory:

                return False

            if self.working\_memory[condition\_var] != condition\_val:

                return False

        return True

    def check\_rule\_with\_missing(self, rule):

        for condition\_var, condition\_val in rule['conditions']:

            if condition\_var not in self.working\_memory:

                return False, condition\_var

            if self.working\_memory[condition\_var] != condition\_val:

                return False, None

        return True, None

    def query\_missing\_fact(self, var):

        if var in ['проблема', 'решение']:

            self.skipped\_vars.add(var)

            return False

        if var not in self.working\_memory and var not in self.skipped\_vars:

            if var in self.variable\_options:

                print(f"\nДля продолжения диагностики нужно знать значение {var}:")

                options = self.variable\_options[var]

                for i, val in enumerate(options, 1):

                    print(f"{i}. {val}")

                print(f"{len(options) + 1}. Пропустить")

                try:

                    choice = int(input("Ваш выбор (введите номер): "))

                    if 1 <= choice <= len(options):

                        val = options[choice-1]

                        self.working\_memory[var] = val

                        print(f"Добавлен факт: {var} = {val}")

                        return True

                    elif choice == len(options) + 1:

                        print(f"Пропускаем переменную {var}")

                        self.skipped\_vars.add(var)

                        return False

                    else:

                        print("Неверный выбор!")

                        return False

                except ValueError:

                    print("Ошибка: введите число!")

                    return False

            else:

                val = input(f"Введите значение для {var} (или Enter для пропуска): ")

                if val:

                    self.working\_memory[var] = val

                    return True

                else:

                    print(f"Пропускаем переменную {var}")

                    self.skipped\_vars.add(var)

                    return False

        return False

    def query\_any\_missing\_fact(self):

        for rule in self.rules:

            rule\_applicable, missing\_var = self.check\_rule\_with\_missing(rule)

            if rule\_applicable:

                conclusion\_var, conclusion\_val = rule['conclusion']

                if conclusion\_var not in self.working\_memory:

                    self.working\_memory[conclusion\_var] = conclusion\_val

                    print(f"Выведен новый факт: {conclusion\_var} = {conclusion\_val}")

                    if conclusion\_var == 'проблема':

                        return True

                    return True

        missing\_vars = set()

        for rule in self.rules:

            for condition\_var, \_ in rule['conditions']:

                if (condition\_var not in self.working\_memory and

                    condition\_var not in self.skipped\_vars):

                    missing\_vars.add(condition\_var)

        if not missing\_vars:

            return False

        print("\nДля продолжения диагностики нужна дополнительная информация:")

        for var in missing\_vars:

            if var not in self.working\_memory and var not in self.skipped\_vars:

                if var in self.variable\_options:

                    print(f"\nВыберите значение для {var}:")

                    options = self.variable\_options[var]

                    for i, val in enumerate(options, 1):

                        print(f"{i}. {val}")

                    print(f"{len(options) + 1}. Пропустить")

                    try:

                        choice = int(input("Ваш выбор (введите номер): "))

                        if 1 <= choice <= len(options):

                            val = options[choice-1]

                            self.working\_memory[var] = val

                            print(f"Добавлен факт: {var} = {val}")

                            if 'проблема' in self.working\_memory:

                                return True

                            return True

                        elif choice == len(options) + 1:

                            print(f"Пропускаем переменную {var}")

                            self.skipped\_vars.add(var)

                        else:

                            print("Неверный выбор!")

                    except ValueError:

                        print("Ошибка: введите число!")

                else:

                    val = input(f"Введите значение для {var} (или Enter для пропуска): ")

                    if val:

                        self.working\_memory[var] = val

                        if 'проблема' in self.working\_memory:

                            return True

                        return True

                    else:

                        print(f"Пропускаем переменную {var}")

                        self.skipped\_vars.add(var)

        return False

    def clear\_facts(self):

        self.working\_memory = {}

        self.skipped\_vars = set()

        print("Рабочая память очищена.")

    def show\_variable\_options(self):

        print("\nВозможные значения переменных (только для условий):")

        for var, options in self.variable\_options.items():

            print(f"{var}: {', '.join(options)}")

    def get\_input\_with\_options(self, var):

        if var in self.variable\_options:

            print(f"\nВыберите значение для {var}:")

            for i, val in enumerate(self.variable\_options[var], 1):

                print(f"{i}. {val}")

            while True:

                try:

                    choice = int(input("Ваш выбор (введите номер): "))

                    if 1 <= choice <= len(self.variable\_options[var]):

                        return self.variable\_options[var][choice-1]

                    else:

                        print("Неверный выбор!")

                except ValueError:

                    print("Ошибка: введите число!")

        else:

            return input(f"Введите значение для {var}: ").strip()

def main():

    rule\_file = "rules.txt"

    es = ExpertSystem(rule\_file)

    while True:

        print("\n" + "="\*60)

        print("Экспертная система диагностики неисправностей компьютера")

        print("1. Выполнить диагностику")

        print("2. Показать правила")

        print("3. Добавить правило")

        print("4. Удалить правило")

        print("5. Выход")

        choice = input("Выберите действие: ")

        if choice == '1':

            es.working\_memory = {}

            es.skipped\_vars = set()

            print("\nДля начала диагностики ответьте на несколько вопросов:")

            if 'компьютер\_включается' in es.variable\_options:

                print("Выберите значение для компьютер\_включается:")

                options = es.variable\_options['компьютер\_включается']

                for i, val in enumerate(options, 1):

                    print(f"{i}. {val}")

                try:

                    choice = int(input("Ваш выбор (введите номер): "))

                    if 1 <= choice <= len(options):

                        es.working\_memory['компьютер\_включается'] = options[choice-1]

                    else:

                        print("Неверный выбор! Установлено значение по умолчанию: нет")

                        es.working\_memory['компьютер\_включается'] = 'нет'

                except ValueError:

                    print("Ошибка: введите число! Установлено значение по умолчанию: нет")

                    es.working\_memory['компьютер\_включается'] = 'нет'

            print("\nНачинаем диагностику...")

            es.show\_facts()

            problem\_found = es.logic()

            print("\nДиагностика завершена!")

            if 'проблема' in es.working\_memory:

                print(f"\nВыявленная проблема: {es.working\_memory['проблема']}")

                if 'решение' in es.working\_memory:

                    print(f"Рекомендуемое решение: {es.working\_memory['решение']}")

            else:

                print("\nНе удалось выявить конкретную проблему.")

                print("Возможные причины:")

                print("- Компьютер работает нормально")

                print("- Проблема не описана в базе знаний")

                print("Рекомендуется обратиться в сервисный центр для профессиональной диагностики.")

            es.show\_facts()

        elif choice == '2':

            es.show\_rules()

        elif choice == '3':

            es.add\_rule()

        elif choice == '4':

            es.delete\_rule()

        elif choice == '5':

            break

        else:

            print("Неверный выбор!")

if \_\_name\_\_ == "\_\_main\_\_":

    main()

Пример выполнения программы представлен на рисунке 1.

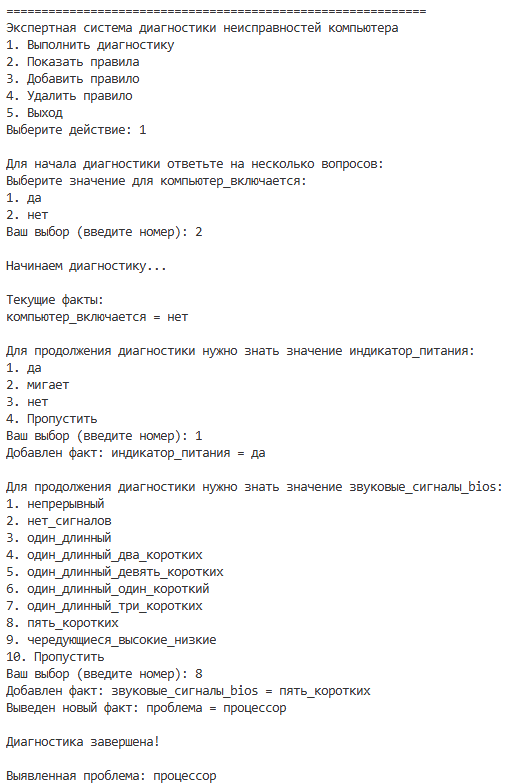


Рисунок 1 – Пример выполнения программы

**Вывод:**

В ходе выполнения лабораторной работы была разработана экспертная система с прямой цепочкой рассуждений для диагностики неисправностей компьютера.