Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования «Пермский национальный исследовательский политехнический университет» (ПНИПУ)

Кафедра вычислительной математики и механики

**Лабораторная работа № 2**

**по дисциплине: «Интеллектуальные ИСИТ»**

Выполнил

студент группы ИСТ-19-2б

Кошкаров П.С.

Проверил

ассистент кафедры ВММБ

Нетбай Г.В.

Пермь, 2022

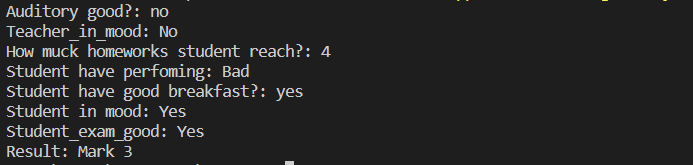
**Машина логического вывода:** Программный компонент, который обеспечивает формирование логического вывода (принимая решение о том, каким правилам удовлетворяют факты или объекты), располагает выполняемые правила по приоритетам и выполняет правило с наивысшим приоритетом.

**Цель:** построить учебный прототип экспертной системы, основанной на знаниях, включающих основные компоненты ИИС.

**Описание:** необходимо реализовать оболочку ЭС, которая была бы способна работать с продукционной БЗ. Также был создан минимальный прототип оболочки ЭС, позволяющий осуществить логический вывод. Было выбрано хранение в формате JSON, язык программирования Python

**Хранение данных в JSON файле:** данные представлены в виде словаря, который хранит словари-правила. Каждый словарь-правило хранит условия для выполнения и функции, которые в нём выполняются.

**Пример работы программы:**



# **Листинг кода Python**

import json

from random import choice

import pathlib

class controller:

    def \_\_init\_\_(self, jsonfile):

        """

        Класс для осуществления чтения из файла, осуществления работы с БЗ, сопоставления правил

        ### Properties

        raw\_data - сырое содержимое json файла

        rules - массив правил в виде класса

        answered - массив ответов на правила

        """

        self.raw\_data = {}

        self.rules = {}

        self.answered = {}

        with open(jsonfile, "r", encoding='utf-8') as read\_rules:

            self.raw\_data = json.load(read\_rules)

        for id, info in self.raw\_data.items():

            self.rules[id] = rule(id, info["conditions"], info["actions"])

        for r in self.rules.values():

            for condition in r.conditions:

                if isinstance(condition, list):

                    self.answered[condition[0]] = None

    def perform\_actions(self, rule\_id):

        """

        Функция, отвечающая за выполнение всех действий в правиле

        """

        for action in self.rules[rule\_id].actions:

            if action["func"] == "ask\_int":

                self.answered[action["arg\_1"]] = self.ask\_int(action["arg\_2"])

            elif action["func"] == "yes\_or\_no":

                self.answered[action["arg\_1"]] = self.yes\_or\_no(action["arg\_2"])

            elif action["func"] == "assert":

                self.answered[action["arg\_1"]] = action["arg\_2"]

            elif action["func"] == "print":

                self.print(action["arg\_1"])

    def ask\_int(question):

        """

        Функция для ответа на вопрос, предполагающий целочисленный ввод

        """

        while True:

            answer = input(question)

            try:

                answer = int(answer)

                return answer

            except:

                continue

    def yes\_or\_no(question):

        """

        Функция для ответа на вопрос да или нет

        """

        while True:

            answer = input(question)

            if answer.lower() in ["yes", "y"]:

                return "yes"

            elif answer.lower() in ["no", "n"]:

                return "no"

    def print(text):

        """

        Функция для вывода текста

        """

        print(text)

    def interprete(self, rule\_id):

        """

        Функция, интерпретирующая условия в правилах и проверяющая их правильность в виде true или false

        """

        text = ""

        for item in self.rules[rule\_id].conditions:

            temp = ""

            if isinstance(item, list):

                temp += f"self.answered[\"{item[0]}\"]"

                if item[1] == "eq":

                    temp += " == "

                elif item[1] == ">":

                    temp += " > "

                elif item[1] == "<":

                    temp += " < "

                elif item[1] == ">=":

                    temp += " >= "

                if item[2] == "null":

                    temp += "None"

                elif isinstance(item[2], int):

                    temp += f"{item[2]}"

                else:

                    temp += f"\"{item[2]}\""

            elif isinstance(item, str):

                temp = f" {item} "

            text += temp

        try:

            ready = eval(text)

        except:

            ready = False

        return text, ready

    def run(self):

        """

        Функция, являющаяся главным циклом, которая выполняет все допустимые правила в цикле

        """

        while True:

            options = []

            for rule in self.rules.values():

                cond, ready = self.interprete(rule.id)

                if ready == True and rule.activated == False:

                    options += [rule.id]

            try:

                chosen\_rule = choice(options)

                self.perform\_actions(chosen\_rule)

                self.rules[chosen\_rule].activated = True

            except:

                break

            self.implications()

    def implications(self):

        """

        Функция прохода по всем правилам и ответ на те, которые возможны

        """

        while True:

            done = True

            for rule in self.rules.values():

                cond, ready = self.interprete(rule.id)

                if ready == True and rule.activated == False and rule.requires\_input == False:

                    self.perform\_actions(rule.id)

                    rule.activated = True

                    done = False

                    break

            if done == True:

                break

class rule:

    def \_\_init\_\_(self, id:str, conditions: list, actions: list):

        """

        Класс для описания объекта-правила

        ### Properties

        id - идентификатор правила\n

        conditions - условия для выполнения\n

        actions - действия в правиле\n

        activated - выполнилось ли правило, да или нет

        requires\_input - требуется ли ввод, т.е. правило подразумевает ввод информации, да или нет

        """

        self.id = id

        self.conditions = conditions

        self.actions = actions

        self.activated = False

        if any([action["func"]=="yes\_or\_no" or action["func"]=="ask\_int" for action in self.actions]):

            self.requires\_input = True

        else:

            self.requires\_input = False

def main():

    json\_path = str(pathlib.Path(\_\_file\_\_).parent.resolve()) + "/rules.json"

    m = controller(json\_path)

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

    main()