МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ

по лабораторной работе №3
по дисциплине «Логическое программирование»
Тема: Возможности внутренних баз данных.
Построение экспертных систем.
Вариант 1

Студентка гр. 1304 _____ Чернякова В.А.
Студентка гр. 1304 _____ Ярусова Т.В.
Студент гр. 1304 _____ Байков Е.С.
Студент гр. 1304 _____ Мамин Р.А.
Преподаватель _____ Родионов С.В.

Санкт-Петербург
2025

Цели работы.

- Изучение возможностей по модификации внутренних баз данных языка Пролог, освоение принципов использования «статических переменных» в языке Пролог.
- Изучение возможности создания экспертных систем на языке Пролог, освоение принципов формирования полноценных приложений, которые могут взаимодействовать с пользователем для сбора дополнительной информации.

Задачи.

Для достижения поставленной цели требуется решить следующие задачи:

1) Изучить теоретический материал.

2) Создать правила в соответствии с вариантом задания и общей формулировкой задачи.

3) Проверить выполнение программы.

4) Составить отчет о выполнении работы.

5) Представить на проверку файл отчета и файл текста программы на языке GNU Prolog, решающей поставленные задачи.

Общие теоретические сведения.

1. **Управление динамической базой знаний**

1.1 Директива *:- dynamic*

Назначение: Позволяет объявлять предикаты как динамические, что даёт возможность добавлять, изменять и удалять их определения во время выполнения программы.

Применение: Используется для создания предикатов, которые могут модифицироваться на лету (например, при работе с базами знаний, где новые факты добавляются по ходу работы).

1.2 Предикаты добавления фактов: *asserta/1* и *assertz/1*

asserta/1: Добавляет новый факт или правило в начало списка определений предиката. Это может быть полезно, когда порядок обработки имеет значение.

assertz/1: Добавляет новый факт или правило в конец списка определений предиката. Использование данного предиката обеспечивает, что новые определения будут обрабатываться после уже существующих.

Назначение обоих: Обогащение базы знаний новыми данными без перезапуска программы.

1.3 Предикаты удаления фактов: *retract/1* и *retractall/1*

retract/1: Удаляет первое найденное определение, соответствующее заданному шаблону. Используется, когда необходимо убрать конкретный факт или правило.

retractall/1: Удаляет все определения, соответствующие заданному шаблону. Это полезно для полной очистки определенного предиката перед его повторным использованием или обновлением.

2. Операции ввода-вывода и работа с файлами

2.1 Работа с файлами: *open/3*, *read/2* и *close/1*

open/3: Открывает файл для работы в указанном режиме (например, чтение, запись или добавление). В результате создаётся поток, с которым можно работать дальше.

read/2: Считывает из открытого потока следующий терм (логический элемент) программы. Позволяет последовательно получать данные из файла.

close/1: Закрывает ранее открытый поток, освобождая ресурсы и завершая работу с файлом.

2.2 Форматированный вывод: *write/1*, *format/2* и *nl/0*

write/1: Выводит представление переданного терма в стандартное устройство вывода. Используется для демонстрации значений переменных или результатов вычислений.

format/2: Позволяет выводить текст с форматированием, используя специальные спецификаторы (аналогично функции printf в других языках). Это

даёт возможность комбинировать статические и динамические данные в одном сообщении.

nl/0: Выводит символ перевода строки, что упрощает организацию читаемого вывода и структурирование сообщений.

3. Операции над списками и агрегирование результатов

3.1 Проверка принадлежности: *member/2*

Назначение: Предикат *member/2* проверяет, принадлежит ли элемент заданному списку. Это часто используется при переборе элементов списка и проверке условий.

3.2 Сбор всех решений: *findall/3*

Назначение: *findall/3* выполняет запрос и собирает все найденные решения в список. Такой механизм позволяет агрегировать результаты логического вывода, что особенно полезно при поиске всех возможных вариантов удовлетворения условий.

4. Управление потоком выполнения и логикой вывода

4.1 Контроль над обратным поиском: оператор *! (cut)*

Назначение: Оператор «cut» используется для ограничения поиска, предотвращая обратный ход (backtracking) после его срабатывания. Это помогает оптимизировать выполнение и обеспечить корректное поведение при выборе вариантов.

4.2 Условные конструкции: -> и ;

-> (if-then): Позволяет задать условную ветвь, где если условие истинно, выполняется соответствующий блок команд.

; (else): Используется в связке с -> для обозначения альтернативного выполнения, если условие ложно.

Применение: Такие конструкции позволяют реализовывать логику ветвления, что критически важно при построении управляемых систем вывода и обработки данных.

5. Обработка ошибок и исключений

5.1 Предикат *catch/3*

Назначение: Обеспечивает механизм обработки исключений. Позволяет "поймать" ошибку, возникшую в ходе выполнения определённого предиката, и перейти к заданному обработчику.

Синтаксис: *catch(Цель, Исключение, Обработчик)*, где при возникновении ошибки управление передаётся в обработчик, что позволяет обеспечить устойчивость программы и корректное завершение операций.

6. Дополнительные инструменты для анализа и проверки терминов

6.1 Предикаты проверки типа и структуры

var/1 и *nonvar/1*: Используются для проверки, является ли переменная неинициализированной или, наоборот, содержит значение. Это важно для контроля над выполнением логических условий.

atomic/1: Проверяет, является ли терм атомом (константой) – простейшей единицей данных в *Prolog*.

Задание.

Реализуйте экспертную систему, на языке Пролог, согласно одному из предложенных вариантов в соответствии со своим номером варианта. Вопросы, задаваемые пользователю, не должны повторяться (дублироваться). Базу знаний (начальные факты) требуется придумать самостоятельно (не менее 20 фактов). Желательно, чтобы база знаний сохранялась в файле (и читалась при запуске программы). В ответах/фактах из нескольких слов вместо пробелов следует писать символ "_". Желательно, чтобы факты, вопросы и интерфейс были написаны на русском языке или на транслите.

[Задание 1, Электронный терапевт]

Имеется список болезней и характерных для них симптомов (в виде фактов базы знаний). (например, "грипп": кашель, высокая температура, головная боль). Требуется по введенным пользователем симптомам определить, чем он заболел (м.б. варианты).

Порядок выполнения работы.

Задание 1. Электронный терапевт

1. **Подготовительный этап**
    a. **Определение цели и задачи**

Основная цель лабораторной работы – создать экспертную систему, которая по ответам пользователя определяет возможное заболевание. При решении задачи используется база знаний с 26 фактами, где каждое заболевание описывается списком характерных симптомов.

    b. **Разбиение функционала на модули**

Программа разделена на два основных файла:

- *diseases.pl* – внешний файл, содержащий базу знаний (факты вида *disease(Заболевание, [Симптомы])*.)
- *expert_system.pl* – основной файл программы, содержащий логику диагностики, опроса пользователя, динамическое сохранение ответов и вывод результата.

    c. **Подготовка динамических предикатов**

В начале файла объявляются динамические предикаты *known/2* и *disease/2*, которые будут использоваться для хранения ответов пользователя и фактов о болезнях соответственно. Это позволяет во время работы программы изменять содержимое базы знаний и сохранять ответы, а также очищать их перед запуском диагностики.

2. **Загрузка базы знаний**
    a. **Открытие файла и чтение потока**

Предикат *load_database(File)* открывает указанный файл (*diseases.pl*) для чтения, затем вызывает предикат *process_stream/1* для последовательного считывания всех термов из файла и после завершения закрывает поток.

    b. **Рекурсивное считывание термов.**

Предикат *process_stream(Stream)* использует рекурсию: он считывает следующий терм с помощью *read/2* и проверяет, достигнут ли конец файла (*end_of_file*). Если файл не закончился, вызывается *handle_term/1*, который обрабатывает текущий терм, и затем функция рекурсивно продолжает чтение. Такой подход гарантирует, что все факты из файла будут прочитаны, даже если их количество неизвестно заранее.

c. Обработка считанного терма

Предикат *handle_term(Term)* проверяет, соответствует ли прочитанный терм формату факта *disease/2*. Если да, то он сохраняется в динамической базе знаний с помощью *assertz/1*. Все другие термы игнорируются.

3. Инициализация и запуск диагностики

a. Очистка базы динамических фактов

Предикат *diagnose/0* начинает с очистки всех ранее сохранённых ответов (с помощью *retractall(known(_,_))*) и фактов болезни (с помощью *retractall(disease(_,_))*). Это гарантирует, что при каждом запуске диагностики система работает с чистым состоянием.

b. Вывод приветственного сообщения и инструкции

После очистки динамической базы система выводит сообщение с приветствием и краткую инструкцию о том, что все вопросы требуют ответа «da» или «net». Это помогает пользователю сразу понять формат ввода.

c. Сбор списка заболеваний

Используется предикат *findall/3* для извлечения списка всех заболеваний, представленных в базе знаний. Полученный список затем используется для последовательной проверки каждого диагноза.

4. Диагностика заболеваний

a. Проверка каждого заболевания

Предикат *diagnose_diseases/1* перебирает список болезней. Если список пуст, выводится сообщение о невозможности установить диагноз. Если список не пуст, для каждой болезни вызывается *check_disease/1*.

b. Извлечение симптомов

Для каждого заболевания вызывается предикат *check_disease/1*, который получает список характерных симптомов (например, для гриппа – *[кашель, высокая_температура, головная_боль, ломота_в_мышцах, усталость]*).

c. Рекурсивная проверка симптомов

check_symptoms/1 проходит по списку симптомов. Для каждого симптома вызывается предикат *ask/1*, который обеспечивает запрос у пользователя. Если для всех симптомов дан положительный ответ (значение *da*), болезнь считается подтверждённой.

5. Опрос пользователя по симптомам

a. Запрос наличия симптома.

Предикат *ask/1* проверяет, задан ли уже ответ для данного симптома (используя динамический предикат *known/2*). Если ответ уже существует, запрос не повторяется.

Если ответа нет, система выводит вопрос с помощью *format/2*, запрашивая наличие симптома (например, «*U vas est simptom "kashel"? (da/net):* ») и считывает ответ через *read/1*, после чего сохраняет его с помощью *asserta/1*. Это гарантирует, что пользователь не будет опрашиваться повторно по одному и тому же симптому, что делает опрос более удобным. Если на вопрос дан не валидный ответ, он задаётся повторно.

6. Формирование и вывод результатов диагностики

a. Формирование итогового списка заболеваний

После проверки всех заболеваний система с помощью *findall/3* собирает список тех заболеваний, для которых все характерные симптомы

были подтверждены (то есть для каждого симптома был получен ответ yes).

b. Обработка отсутствия совпадений

Если итоговый список оказывается пустым, система выводит сообщение о том, что по введённым симптомам диагноз установить невозможно.

7. Вывод результата диагностики

a. Сбор подтверждённых болезней

После проверки всех болезней предикат *diagnose_diseases/1* формирует список болезней, для которых все симптомы были подтверждены пользователем.

b. Вывод результата

Если список подтверждённых болезней пуст, выводится сообщение о том, что диагноз установить невозможно. Если список содержит одну или несколько болезней, они выводятся на экран с помощью предиката *print_list/1*, который рекурсивно печатает каждый элемент списка. Таким образом, пользователь получает итоговую информацию о возможном диагнозе (или диагнозах) на основе своих ответов.

Инструкция по запуску.

Диаграмма выполнения программы приведена в файле *lb3.svg* и по ссылке https://clck.ru/3GWSmc

Инструкция по запуску.

1. Сохраните файл с базой знаний как *diseases.pl* и основной файл как *expert_system.pl* в одной директории.

2. Запустите *GNU Prolog* и загрузите основной файл командой:*?- consult('C:/.../expert_system.pl')*.

3. Для начала диагностики введите:*?- diagnose*.

4. Отвечайте на вопросы, вводя *da*. или *net.* (с точкой в конце).

5. По завершении опроса система выведет список возможных заболеваний.

Пример выполнения программы.

На рисунках 1-4 примеры приведены работы программы:

```
| ?- consult('C:/Users/Roman/Documents/prolog/expert_system.pl').
compiling C:/Users/Roman/Documents/prolog/expert_system.pl for byte code...
C:/Users/Roman/Documents/prolog/expert_system.pl compiled, 90 lines read -

(16 ms) yes
| ?- diagnose.

Dobro pozhalovat v ekspertnoj sisteme "Elektronnyj terapevt"!
Otvechajte, pozhalujsta, na voprosy, vvodja "da" ili "net".
```

Рисунок 1. Запуск и приветствие программы.

```
U vas est simptom "kashel"? (da/net):
da
.
U vas est simptom "vysokaya_temperatura"? (da/net):
da.
U vas est simptom "golovnaya_bol"? (da/net):
da.
U vas est simptom "lomota_v_myshcakh"? (da/net):
da.
U vas est simptom "ustalost"? (da/net):
da.
U vas est simptom "nasmork"? (da/net):
net.
```

Рисунок 2. Опрос программы.

```
----.
U vas est simptom "vnezapnaya_slabost"? (da/net):
net.

Vozmozhnye zabolevaniya:
- gripp

true ?
```

Рисунок 3. Вывод диагноза.

```
U vas est simptom "ustalost"? (da/net):
da.
U vas est simptom "nasmork"? (da/net):
sds.
Nepravilnyy vvod. Poprobuyte eshche raz.
U vas est simptom "nasmork"? (da/net):
sd
.
Nepravilnyy vvod. Poprobuyte eshche raz.
U vas est simptom "nasmork"? (da/net):
net
.
U vas est simptom "bol_v_gorle"? (da/net):
```

Рисунок 4. Обработка некорректного ввода.

Выводы.

В ходе выполнения лабораторной работы была создана экспертная система на языке *GNU Prolog*, реализующая диагностику заболеваний на основе введённых пользователем симптомов.

При запуске программы сначала происходит очистка динамических предикатов (для ответов и базы знаний), затем вызывается предикат *load_database/1*, который открывает файл с базой знаний, рекурсивно считывает все факты, обрабатывает их (сохраняя только факты вида *disease/2*) и закрывает файл. Далее программа выводит приветственное сообщение и задаёт пользователю вопросы по каждому симптому, при этом ответ на один и тот же симптом запрашивается только один раз. На основе полученных ответов формируется список болезней, для которых подтверждены все симптомы. Если ни для одной болезни все симптомы не подтверждены, выводится сообщение об отсутствии диагноза, иначе – список возможных заболеваний.

Данный вариант программы удовлетворяет поставленным требованиям: база знаний хранится во внешнем файле, вопросы к пользователю не дублируются, а в ответах и фактах вместо пробелов используется символ подчёркивания. Интерфейс и сообщения выведены на транслите.

Зоны ответственности членов бригады:

- Чернякова В.А. – тестирование программы

- Ярусова Т.В. – составление отчета
- Байков Е.С. – написание программы
- Мамин Р.А. – написание программы

Каждый участник бригады проконтролировал действия других участников и разобрался в проделанной ими работе.

В ходе выполнения лабораторной работы возникли следующие трудности:

- **Работа с динамическими предикатами и очистка базы данных**
  - **Описание проблемы:** В процессе разработки возникли сложности с пониманием механизма динамических предикатов в *GNU Prolog*. Первоначально не до конца было понятно, как правильно объявлять и очищать предикаты, такие как *known/2* и *disease/2*.
  - **Конкретные трудности:**
    - Использование :- *dynamic(known/2)*. и :- *dynamic(disease/2)*. – сначала возникали синтаксические ошибки из-за неверного оформления (например, пропущенная закрывающая скобка в динамической декларации)
    - Правильное применение *retractall/1* для очистки базы данных перед запуском диагностики оказалось критичным, так как без этого система могла использовать устаревшие данные, что приводило к неверным результатам.
  - **Решение:** После изучения документации было утверждено, что динамические предикаты необходимо объявлять строго в синтаксически правильном виде, а очистку базы следует выполнять с помощью *retractall/1* в начале диагностики.
- **Загрузка базы знаний из файла**
  - **Описание проблемы:** Для загрузки фактов о болезнях было принято решение использовать чтение термов из файла с помощью потоков. Однако изначальная реализация с использованием конструкции *repeat* и *fail* оказалась неочевидной для понимания, и в ней возникали проблемы с корректным завершением чтения.

- **Конкретные трудности:**
  - Определение корректного условия завершения чтения – нужно было правильно обрабатывать термин *end_of_file*, чтобы избежать бесконечного цикла.
  - При использовании рекурсии возникали сомнения в том, что все термы будут прочитаны корректно, особенно при наличии ошибок в файле базы знаний.
  - Возможны проблемы с форматированием и кодировкой файла (например, наличие BOM или невидимых символов), что могло привести к синтаксическим ошибкам при чтении.
- **Решение:** Были переписаны предикаты загрузки базы знаний, создан предикат *process_stream/1*, который рекурсивно читает термы и проверяет условие окончания файла с помощью конструкции (*Term == end_of_file -> true ; ...*). Это позволило добиться корректного и полного считывания данных из файла.

- **Обеспечение недублирования вопросов пользователю**
  - **Описание проблемы:** Одним из требований задачи было, чтобы вопросы, задаваемые пользователю, не повторялись. При первоначальной реализации было трудно обеспечить, чтобы при повторном обращении к одному и тому же симптому не запрашивался ответ.
  - **Конкретные трудности:**
    - Необходимо было грамотно использовать динамический предикат *known/2* для сохранения полученных ответов.
    - Логика предиката *ask/1* должна была учитывать наличие уже сохранённого ответа и избегать повторного запроса, что требовало добавления оператора отсечения (*!*).
  - **Решение:** Реализован предикат *ask/1* таким образом, что сначала проверяется, есть ли уже ответ на данный симптом, и если да, то запрос не производится. Это было решено с помощью конструкции

ask(Symptom) :- known(Symptom, _), !. и последующего запроса, если ответа нет.

- **Отладка логики диагностики**
  - **Описание проблемы:** На этапе тестирования выявлялись случаи, когда система некорректно определяла диагноз или вовсе не могла установить его, даже при наличии совпадающих симптомов.
  - **Конкретные трудности:**
    - Неправильная логика в предикатах *diagnose_diseases/1* и *check_symptoms/1* могла приводить к тому, что болезнь считалась подтверждённой, если подтверждены не все её симптомы.
    - При использовании *findall/3* иногда получался пустой список, что требовало дополнительной проверки на отсутствие совпадений.
  - **Решение:** Логику проверки была разделена на несколько четких предикатов: сначала для каждой болезни извлекается список симптомов, затем рекурсивно проверяется наличие каждого симптома. Если для всех симптомов получен ответ da, болезнь добавляется в список возможных диагнозов. Это позволило добиться корректного формирования итогового списка болезней.

Приложение А

Исходный код программы

Название файла: *expert_system.pl*

```
% Файл: expert_system.pl
% Экспертная система "Электронный терапевт"
% Лабораторная работа №3 по дисциплине "Логическое программирование"
% Выполнили: Чернякова В.А., Ярусова Т.В., Байков Е.С., Мамин Р.А.

:- dynamic(known/2).
:- dynamic(disease/2).
% known(Simptom, Answer). Answer может быть: da или net.

% Открывает файл с базой знаний, читает все термы и закрывает поток.
load_database(File) :-
  % Открываем файл File для чтения
    open(File, read, InStream),
  % Рекурсивно читаем все термы из потока
    process_stream(InStream),
  % Закрываем поток после чтения
    close(InStream).

% Рекурсивно считывает термы из открытого потока Stream.
% Если достигнут конец файла (end_of_file), завершает чтение,
% иначе обрабатывает текущий терм и продолжает.
process_stream(Stream) :-
% Читаем следующий терм из потока
    read(Stream, Term),
    ( Term == end_of_file ->
   % Если достигнут конец файла, завершаем чтение
        true
    ;   % Обрабатываем прочитанный терм
        handle_term(Term),
       % Рекурсивно продолжаем чтение
        process_stream(Stream)
    ).
```

```
% Если Term соответствует факту disease/2,
% сохраняет его в динамической базе знаний.
% Остальные термы игнорируются.
handle_term(disease(Disease, Symptoms)) :-
    assertz(disease(Disease, Symptoms)).
handle_term(_).  % Остальные термы отбрасываем

%% Основной предикат диагностики

diagnose :-
  % Удаляем все ранее сохранённые ответы
    retractall(known(_,_)),
  % Очищаем динамическую базу данных болезней
    retractall(disease(_,_)),
  % Загружаем базу знаний из файла "diseases.pl"
    load_database('diseases.pl'),
    nl, write('Dobro pozhalovat v ekspertnoj sisteme "Elektronnyj
terapevt"!'), nl,
    write('Otvechajte, pozhalujsta, na voprosy, vvodja "da" ili "net".'), nl,
nl,
    % Получаем список всех болезней из базы знаний
    findall(D, disease(D, _), Diseases),
  % Запускаем диагностику по списку болезней
    diagnose_diseases(Diseases).

% Диагностика по списку заболеваний
% Если список болезней пуст, выводим сообщение об отсутствии диагноза.
% Иначе, для каждой болезни проверяем, подтверждены ли все её симптомы.
diagnose_diseases([]) :-
    nl, write('Po vashim simptomam ne najdeno sootvetstvennoe zabolevanie.'),
nl.
diagnose_diseases(Diseases) :-
    findall(D, (member(D, Diseases), check_disease(D)), Matched),
    ( Matched = [] ->
        nl, write('Po vashim simptomam ne najdeno sootvetstvennoe
zabolevanie.'), nl
      ;
        nl, write('Vozmozhnye zabolevaniya:'), nl,
        print_list(Matched)
```

```
    ).

% Для данной болезни Disease извлекает список симптомов и проверяет,
% что пользователь подтвердил наличие каждого симптома.
check_disease(Disease) :-
    disease(Disease, Symptoms),
    check_symptoms(Symptoms).

% Рекурсивно проверяет, что для каждого симптома из списка Symptoms
% пользователь дал положительный ответ (da).
check_symptoms([]).
check_symptoms([Symptom|Rest]) :-
    ask(Symptom),
    known(Symptom, da),
    check_symptoms(Rest).

% Запрос симптома у пользователя
% Если по данному симптому уже получен ответ, используется он; иначе задаётся
вопрос.
% При некорректном вводе (не "da" и не "net") происходит повторный запрос.
% Запрос симптома у пользователя с обработкой некорректного ввода
% Запрос симптома у пользователя с обработкой ошибок ввода через catch/3
ask(Symptom) :-
    known(Symptom, _), !.
ask(Symptom) :-
    format('U vas est simptom "~w"? (da/net): ', [Symptom]), nl,
    catch(read(Response), _Error,
          ( write('Nepravilnyy vvod. Poprobuyte eshche raz.'), nl,
ask(Symptom) )),
    ( (Response == da ; Response == net) ->
          asserta(known(Symptom, Response))
    ;
          write('Nepravilnyy vvod. Poprobuyte eshche raz.'), nl,
          ask(Symptom)
    ).

% Рекурсивный вывод списка заболеваний
print_list([]).
print_list([H|T]) :-
```

```
    write('- '), write(H), nl,
    print_list(T).

% Для запуска диагностики вызовите:
% ?- diagnose.
```

Название файла: *diseases.pl*

```
% Файл: diseases.pl
% База знаний для экспертной системы "Электронный терапевт"
% Каждая запись имеет вид: disease(Заболевание, [Симптом1, Симптом2, ...]).
% Для многословных названий симптомов используется символ подчёркивания.

disease('gripp', ['kashel', 'vysokaya_temperatura', 'golovnaya_bol',
'lomota_v_myshcakh', 'ustalost']).
disease('prostuda', ['nasmork', 'chikhaniye', 'legkaya_temperatura']).
disease('angina', ['bol_v_gorle', 'zatrudnennoye_glotanie', 'vysokaya_temperatura']).
disease('allergiya', ['chikhaniye', 'zud', 'slezotecheniye', 'nasmork']).
disease('pnevmoniya', ['silnyy_kashel', 'vysokaya_temperatura', 'odyshka',
'bol_v_grudi']).
disease('bronhit', ['kashel', 'vydeleniya_iz_legkikh', 'ustalost',
'nebolshaya_temperatura']).
disease('pishchevoye_otravleniye', ['toshnota', 'rvota', 'bol_v_zhivote', 'diareya']).
disease('migreny', ['silnaya_golovnaya_bol', 'toshnota', 'svetoboyazn']).
disease('diabet', ['zhazhda', 'chastoye_mocheispuskaniye', 'ustalost',
'povyshennyy_sakhar']).
disease('gipertonya', ['golovnaya_bol', 'golovokruzheniye', 'shum_v_usakh']).
disease('ostraya_angina', ['bol_v_gorle', 'likhoradka', 'vysokaya_temperatura']).
disease('sinusit', ['zalozhennost_nosa', 'golovnaya_bol', 'bol_v_litse']).
disease('tonzillit', ['bol_v_gorle', 'uvelichenie_limfaticheskikh_uzlov',
'vysokaya_temperatura']).
disease('meningit', ['silnaya_golovnaya_bol', 'rigidnost_shei', 'vysokaya_temperatura',
'toshnota']).
disease('uretrit', ['boleznennoye_mocheispuskaniye', 'vydeleniya', 'zhzheniye']).
disease('tsistit', ['boleznennoye_mocheispuskaniye',
'chastye_pozyvy_k_mocheispuskaniyu', 'bol_v_zhivote']).
disease('artrit', ['bol_v_sustavakh', 'otek', 'skovannost']).
disease('revmatizm', ['bol_v_sustavakh', 'ustalost', 'likhoradka', 'slabost']).
disease('gepatit', ['zheltyuka', 'bol_v_zhivote', 'ustalost', 'poterya_appetita']).
disease('infektsiya_pishchevaritelnogo_trakta', ['bol_v_zhivote', 'diareya', 'rvota',
'toshnota']).
disease('koronavirus', ['kashel', 'vysokaya_temperatura', 'odyshka',
'poterya_obonyaniya', 'utomlyaemost']).
disease('otit', ['bol_v_uhe', 'snizheniye_slukha', 'likhoradka']).
```

```
disease('zheludochnaya_yazva', ['bol_v_zhivote', 'izzhoga', 'toshnota', 'rvota']).
disease('pankreatit', ['bol_v_zhivote', 'toshnota', 'rvota', 'likhoradka']).
disease('epilepsiya', ['pristupy', 'poterya_soznaniya', 'sudorogi']).
disease('insult', ['vnezapnaya_slabost', 'zatrudnennaya_rech', 'poterya_koordinatsii',
'golovnaya_bol']).
```