

**Логическое программирование.  
Встроенные предикаты.  
Работа с динамической БД фактов.**

Лектор:  
доцент каф. АОИ  
Салмина Нина  
Юрьевна

# Предикаты вывода

**write (X1, X2, ..., XN)**

константы, переменные

write (“\nAge of Tom is: “, X).

**writeln (format , X1, X2, ..., XN)**

“...%m.p[f/e]...”

m – минимальное поле вывода (число)

p – количество цифр после запятой

X=2.345, A=asdf, writeln("X=%4.2f A=%3\n", X, A).

X=2.35 A=asdf

# Предикаты ввода

Чтение строки (с пробелами до “Enter”)

**readln (X)**

Чтение чисел

**readreal (X)**

**readint (X)**

# Преобразования типов

`str_int (S, X)`

`str_real (S, X)`

S – строка

X – число (целое / вещественное)

# Работа со строками

## Предикаты:

Str\_len (S, N)

Concat (X, Y, S)

Frontstr (N, S, X, Y)

Substring (S, NF, NL, X)

X, Y, S – строки

N, NFirst, NLength – целое

## Примеры:

Str\_len (asdf, Y)  $\Rightarrow$  Y=4

Concat (ab, cd, S)  $\Rightarrow$  S=abcd

Frontstr (2, abcde, X, Y)  $\Rightarrow$   
X=ab, Y=cde

Substring (abcde, 2, 3, X)  $\Rightarrow$   
X=bcd

# «Семейные отношения», вычисление среднего возраста всех мужчин/женщин

```
ave_age (X, D, Y) :- sum_age (X, D, [], 0, S, 0, K), Y=S\K.
```

```
sum_age (X, D, L, SN, S, KN, K) :- person (N, _, X, _),  
    not (member (N, L)), !,  
    age (D, N, Y),  
    KS = KN+1, SS = SN+Y,  
    sum_age (X, D, [N|L], SS, S, KS, K).
```

```
sum_age (_, _, _, X, X, Y, Y).
```

**goal**

```
ave_age (man, day(1,10,2020), Y).
```

# Выбор информации из набора фактов

## FINDALL (X, Y, L)

X – аргумент, значение которого заносится в L

Y – предикат, из которого выбирается информация

L – список значений X, выбранных из всех найденных предикатов Y

например:

```
findall (X, person(_, _, man, X), L)
```

L – список дат рождений всех мужчин

# «Семейные отношения», вычисление среднего возраста всех мужчин/женщин

```
ave_age (X, D, Z) :- findall (Y, person(_, _, X, Y), L),  
                    sum_age (D, L, S, K), Z=S/K.
```

```
sum_age (D, [X | T], S, K) :- sum_age (D, T, S1, K1),  
                             age (D, X, Y),  
                             K = K1+1, S = S1+Y.
```

```
sum_age (_, [], 0, 0).
```

**goal**

```
ave_age (man, day(1,10,2017), Y).
```

# Работа с динамической БД фактов. Хранение фактов в файле

Facts [– general / <имя секции фактов>]

% описание предикатов - фактов

<предикат (тип\_арг1, ..., тип\_аргN)>

...

Goal

consult (“<имя файла>” [, < имя секции фактов >]), ...

Загружает факты из текстового файла в заданную область ОП

Несколько БД –  
имя секции  
обязательно!

# Работа с динамической БД фактов. Удаление фактов.

**retract (<факт> [, < имя секции фактов >] )**  
**retractall (<факт> [, < имя секции фактов >] )**

## Примеры:

retract (person (\_, \_, man, \_)) – удалит 1-й  
найденный факт

retractall (person (\_, \_, man, \_)) – удалит ВСЕХ  
мужчин

# Работа с динамической БД фактов. Добавление фактов.

**assert** (<факт> [, < имя секции фактов >] ) – в конец БД

**assertz** (<факт> [, < имя секции фактов >] ) – в конец БД

**asserta** (<факт> [, < имя секции фактов >] ) – в начало БД

Изменение факта (путем замены):

```
retract ( person ( ivan, X, data(Y, Z, G) )),G1=G-1,  
assert ( person ( ivan, X, data(Y, Z, G1))).
```

# Работа с динамической БД фактов. Сохранение фактов.

Если после работы надо СОХРАНИТЬ все изменения:

**save** (“<имя файла>” [, < имя секции фактов >] )

**!! Без данного предиката после окончания работы изменения в БД не сохраняются !!**

# Выезд на бал

Андрей Иванович, Федор Петрович, Валерий Сергеевич и Григорий Алексеевич сопровождают своих дочерей на бал.

Заключительный танец каждая девушка танцевала не со своим отцом. Образовались следующие пары:

- Лена с Андреем Ивановичем,
- Анна с отцом Кати,
- Таня с отцом Анны,
- Федор Петрович с дочерью Валерия Сергеевича,
- Валерий Сергеевич с дочерью Андрея Ивановича.

Кто кому приходится дочерью?

# Вопрос – допустимая перестановка

```
question(L,S) :- another (L,S),  
    may_be ( [ai, fp, vs, ga], S).
```

Андрей Иванович, Федор Петрович, Валерий Сергеевич  
и Григорий Алексеевич

goal

```
question( [lena, any, katy, tany], Y).
```

# Вспомогательные предикаты

% выбор из списка N-го элемента

```
n_elem (1, [X | _], X) :- !.
```

```
n_elem (N, [_ | T], X) :- N1=N-1,  
                           n_elem(N1,T,X).
```

% определить номер элемента в списке

```
douther (X, [X | _], 1) :- !.
```

```
douther (X, [_ | T], N1) :- douther (X, T, N),  
                             N1=N+1.
```

# Работа с динамической БД фактов.

## Facts

% факт, определяющий родственную пару  
relatives (string douthier, string father)

% факт, определяющий танцующую пару  
para (string she, string he)

# Определение пары танцующих

% пара танцующих уже определена

```
f_para(X,Y):-para(X,Y),!.
```

% девочка/отец уже танцует с другим

```
f_para(X,_):-para(X,_),!,fail.
```

```
f_para(_,Y):-para(_,Y),!,fail.
```

% пара танцующих не родственники – добавить в БД

```
f_para (X,Y) :- not (relatives(X,Y)),  
                assert (para(X,Y)).
```

# Проверка допустимости выбранной перестановки

```
may_be(F,[D1,D2,D3,D4]) :- assert (relatives(D3,vs)),
                             assert (relatives(D2,fp)),
                             Андрей Иванович, Федор Петрович, Валерий Сергеевич  
и Григорий Алексеевич assert (relatives(D1,ai)),
                             assert (relatives(D4,ga)),
                             f_para(D3,fp), f_para(D1,vs), f_para(lena,ai),
                             douter(any,[D1,D2,D3,D4],X2),
                             douter(katy,[D1,D2,D3,D4],X3),
                             n_elem(X3,F,Yk), f_para(any,Yk),
                             n_elem(X2,F,Ya), f_para(tany,Ya),!.
```

```
may_be (_, _) :- retractall (relatives(_, _)),
                    retractall (para(_, _)), fail.
```

Лена с Андреем Ивановичем,  
Анна с отцом Кати,  
Таня с отцом Анны,  
Федор Петрович с дочерью  
Валерия Сергеевича,  
Валерий Сергеевич с дочерью  
Андрея Ивановича.

# Результат

Y = ["tany", "katy", "any", "lena"]

1 Solution

соответствующий список отцов

[ ai, fp, vs, ga],

# Задача классификации объектов

В базе данных (файле) содержатся результаты теннисных партий, сыгранных членами некоторого клуба:

Победил (Победитель, Проигравший).

Необходимо определить отношение

Класс (Игрок, Категория)

где **победитель** – игрок, победивший во всех сыгранных им играх;

**боец** – игрок, в некоторых играх победивший, в некоторых – проигравший;

**спортсмен** – игрок, проигравший во всех сыгранных им играх.

Вывести результаты на экран по классам.

# Информация в файле

```
won("ivan","petr")
```

```
won("ivan","tom")
```

```
won("ivan","jim")
```

```
won("jim","tom")
```

```
won("jim","petr")
```

```
won("petr","tom")
```

# Описание термов и предикатов

domains

sp1 = string\*

iss = winner; athlete; fighter

facts - winner

won (symbol, symbol)

predicates

klass (string, iss)

% определение класса

del\_double (sp1,sp1)

% удаление повторных имен

member (string,sp1)

write\_klass (sp1,iss)

% печать списка спортсменов

% заданного класса

all

% основная функция

# Классификация объектов. Программа.

clauses

```
class (X, fighter) :- won ( X, _), won ( _, X), !.
```

```
class (X, winner) :- won (X, _), !.
```

```
class (X, athlete) :- won ( _, X).
```

```
del_double ([X | XT], [X | L]) :- not (member (X, XT)), !,  
                                del_double (XT, L).
```

```
del_double ([_ | XT], L) :- del_double (XT, L), !.
```

```
del_double ([ ], [ ]).
```

```
write_class ([L | LT], X) :- class (L, Y), Y=X, !,  
                             write (L, " \n"), write_class (LT, X).
```

```
write_class ([_ | LT], X) :- write_class (LT, X).
```

```
write_class ( _ , _ ) :- nl.
```

# Классификация объектов. Программа.

```
all :- findall (X, won (X, _), L1), del_double (L1, L),  
write ("winners:\n"), write_klass (L, winner),  
write ("fighters:\n"), write_klass (L, fighter),  
findall (Y, won (_, Y), L2), del_double (L2, L3),  
write ("athletes:\n"), write_klass (L3, athlete).
```

goal

```
consult ("competition", winner), all.
```

# Результат работы программы

winners:

ivan

fighters:

jim

petr

athletes:

tom

yes

# Пример работы со строками

Построить программу "сжать", назначение которой - преобразование английских слов в их "звуковой" код. Этот процесс предусматривает "сжатие" примерно одинаково звучащих слов в одинаковый их код - своего рода, аббревиатуру этих слов. Слова "сжимаются" в соответствии со следующими правилами:

- первая буква слова сохраняется;
- все последующие за ней гласные, а также буквы "h", "w" и "y" удаляются;
- удвоенные буквы заменяются одиночными;
- закодированное слово состоит не более чем из четырех букв, остальные буквы удаляются.

Примеры: сжать(barrington, brng); сжать(llewellyn, ln).

# Описание предикатов

domains

sp1 = string\*

predicates

str\_list(string,sp1) % разбиваем строку на список букв

less1(sp1,sp1) % убираем лишние буквы

list\_str(sp1,string) % сливаем список букв в строку

less(sp1,string) % «сжимаем» слово за исключением первой  
буквы

go(string,string) % основная процедура

# Программа

```
str_list (X,[X]) :- str_len(X,1),!.
str_list (X,[A|Y]) :- frontstr(1,X,A,X1), str_list(X1,Y).
less1([ ],[ ]).
less1([X|Y],Y1) :- member(X,[a,e,y,u,i,o,a,h,w]), !, less1(Y,Y1).
less1([X1,X2|Y],Y1) :-X1=X2, !, less1([X2|Y],Y1).
less1([X|Y],[X|Y1]) :- less1(Y,Y1).
less(X,Y) :- less1(X,Z), list_str(Z,Y).
list_str([ ],"").
list_str([X|Y],Z) :- list_str(Y,Z1), concat(X,Z1,Z).
go(X,Y) :- str_list (X, [X1|XL]), less(XL,Z), str_len(Z,N), N<3,!,
    concat(X1,Z,Y).
go(X,Y):- str_list(X, [X1|XL]), less(XL,Z), frontstr(3,Z,X2,_),
    concat(X1,X2,Y).
```